Software Maintainability: Systematic Literature Review and Current Trends

Article *in* International Journal of Software Engineering and Knowledge Engineering · October 2016

DOI: 10.1142/S0218194016500431

2 authors:

Ruchika Malhotra
University of Information Technology
66 PUBLICATIONS 1,444 CITATIONS
SEE PROFILE

Anuradha Chug
Guru Gobind Singh Indraprastha University
31 PUBLICATIONS 107 CITATIONS
SEE PROFILE

International Journal of Software Engineering
and Knowledge Engineering
Vol. 26, No. 8 (2016) 1221–1253
© World Scientific Publishing Company
DOI: 10.1142/S0218194016500431

Software Maintainability: Systematic Literature Review and Current Trends

Ruchika Malhotra[*,‡] and Anuradha Chug[†,§]

[*]*Department of Computer Engineering*
Delhi Technological University
Main Bawana Road, New Delhi 110042, India

[†]*University School of Information and Communication Technology*
Guru Gobind Singh Indraprastha University
Sector 16C Dwarka, New Delhi 110078, India
[‡]*ruchikamalhotra2004@yahoo.com*
[§]*a_chug@yahoo.co.in*

Received 20 November 2013
Revised 8 January 2014
Accepted 24 December 2015

Software maintenance is an expensive activity that consumes a major portion of the cost of the total project. Various activities carried out during maintenance include the addition of new features, deletion of obsolete code, correction of errors, etc. Software maintainability means the ease with which these operations can be carried out. If the maintainability can be measured in early phases of the software development, it helps in better planning and optimum resource utilization. Measurement of design properties such as coupling, cohesion, etc. in early phases of development often leads us to derive the corresponding maintainability with the help of prediction models. In this paper, we performed a systematic review of the existing studies related to software maintainability from January 1991 to October 2015. In total, 96 primary studies were identified out of which 47 studies were from journals, 36 from conference proceedings and 13 from others. All studies were compiled in structured form and analyzed through numerous perspectives such as the use of design metrics, prediction model, tools, data sources, prediction accuracy, etc. According to the review results, we found that the use of machine learning algorithms in predicting maintainability has increased since 2005. The use of evolutionary algorithms has also begun in related sub-fields since 2010. We have observed that design metrics is still the most favored option to capture the characteristics of any given software before deploying it further in prediction model for determining the corresponding software maintainability. A significant increase in the use of public dataset for making the prediction models has also been observed and in this regard two public datasets User Interface Management System (UIMS) and Quality Evaluation System (QUES) proposed by Li and Henry is quite popular among researchers. Although machine learning algorithms are still the most popular methods, however, we suggest that researchers working on software maintainability area should experiment on the use of open source datasets with hybrid algorithms. In this regard, more empirical studies are also required to be conducted on a large number of datasets so that a

[§]Corresponding author.

generalized theory could be made. The current paper will be beneficial for practitioners, researchers and developers as they can use these models and metrics for creating benchmark and standards. Findings of this extensive review would also be useful for novices in the field of software maintainability as it not only provides explicit definitions, but also lays a foundation for further research by providing a quick link to all important studies in the said field. Finally, this study also compiles current trends, emerging sub-fields and identifies various opportunities of future research in the field of software maintainability.

Keywords: Software maintainability; software design metrics; software measurement; maintainability prediction models; empirical studies of software maintenance.

1. Introduction

The formal definition of 'software maintenance' given by IEEE [1] is "Modification of a software product after delivery to correct faults, to improve the performance or other attributes, or to adapt to a modified environment." Software maintainability is defined as "the ease with which these modifications can be made" [2]. It has been observed that for the entire life cycle of the product, only 30–40% is consumed in development and balance 60–70% is consumed in the maintenance of the product in terms of resources, time, money and efforts [2]. Marco [3] suggested that we can control this cost only by measuring it. Even though it can be measured during the maintenance phase, but it would be too late by then. Researchers are constantly trying to devise a method to forecast the maintainability of any software in early phases of software development life cycle (SDLC) by measuring the characteristics of its design. In the current systematic review, all research papers, review articles, white papers, reports and proceedings of conferences known to authors since 1990 to date have been collected, scrutinized, compiled and analyzed. The objective of the current review is to organize empirical evidence in comprehensive form on the following aspect:

- Various factors that affect maintainability.
- Different means and methods to improve maintainability.
- The use of prediction models for maintainability in the early phases of development.
- Comparing the performance of various maintainability prediction models in terms of the accuracy.
- Identify the advantages and disadvantages of various prediction models over each other.
- Identify the software metrics which can be used in prediction model making process.
- Identifications of the existing gaps for future prospect of research in the field of software maintainability.

In order to achieve the aim, nine digital libraries were extensively searched and 96 primary studies were identified for inclusion in the study based on criteria discussed in later part of this paper. After preliminary investigations, research questions (RQs) were raised and detailed comprehensive reports were generated.

The rest of the paper is organized as follows: Section 2 describes the motivation of undertaking this review. Section 3 discusses the review methodology. Section 4 presents a detailed planning which includes identifications of keywords, raising the research questions and retrieval of studies. Section 5 presents various activities performed while conducting this review which includes the synthesis of information, a list of qualified studies, inclusion and exclusion criterion and assigning the identifier to each of the shortlisted studies. Reporting of reviews and answer to the research questions are presented in Sec. 6. Section 7 discusses current trends and loopholes as it sets the goals and directions for future, and finally Sec. 8 concludes the paper.

2. Motivation

While undertaking the current review, the two obvious questions arise as below:

- Why is it the right topic for a review?
- Is there any recent review carried in this area?

Recently, a survey conducted by Jones [4] reported that during the 1950s only 10% of the total professionals deployed in software industry were engaged in maintenance work, and by the year 2025 this figure would rise to 77%. Further, it claims that acute shortage of software personnel is also due to the burst of maintenance work. Hence, practitioners are trying hard to make maintainable software so that the overall project cost can be controlled and the product can be managed optimally. As part of the better planning, often developers predict maintainability of the software on the basis of its designed characteristics. This motivates us to compile all the studies in the said field in order to identify how much has been achieved as well as the potential areas of research based on the existing gaps. Many reviews have been conducted in the past to compile the studies related to software maintainability by Riaz *et al.* [5], Tieng *et al.* [6], Ghosh *et al.* [7], Saraiva [8] and Saraiva *et al.* [9], but this study is different from all of them in three aspects. Firstly, the current study is carried out as per the guidelines provided by Kitchenham and Charters [10] and Pickard *et al.* [11] for conducting a systematic review in the field of software engineering. Secondly, in this review, empirical studies on software maintainability prediction (SMP) are shortlisted and analyzed both qualitatively as well as quantitatively in tabulated form for easy understanding. A thorough analysis was conducted to cover various aspects of software maintainability predictions such as the prediction techniques, software design metrics, datasets, tools, prediction accuracy of the models, etc. Thirdly and most importantly, none of the reviews [5–9] is as comprehensive as the current one in which more than 96 studies published from the year 1991 till date are reviewed. The main guidelines provided by Kitchenham and Charters [10] and Pickard *et al.* [11] are covered in the next section.

3. Review Methodology

This section explains the procedure of conducting the systematic review adopted in this study. As depicted in Fig. 1, the review methodology adopted in this study is divided into three stages: planning, conducting and reporting. During the planning stage, search databases were identified and after the preliminary investigations, research questions were formulated. Policies regarding inclusion and exclusion of the studies were prepared and all the relevant papers were extracted. During the second stage, all duplicate and irrelevant studies were removed and shortlisted papers were organized. The synthesis was also carried during this stage on the basis of the information provided in each of the shortlisted study. In the last stage, answers to all research questions were reported and current trends in each of the sub-field were identified. An endeavor was also been made to highlight the constraints present in each of the paper which would lead us to the future directions of research.

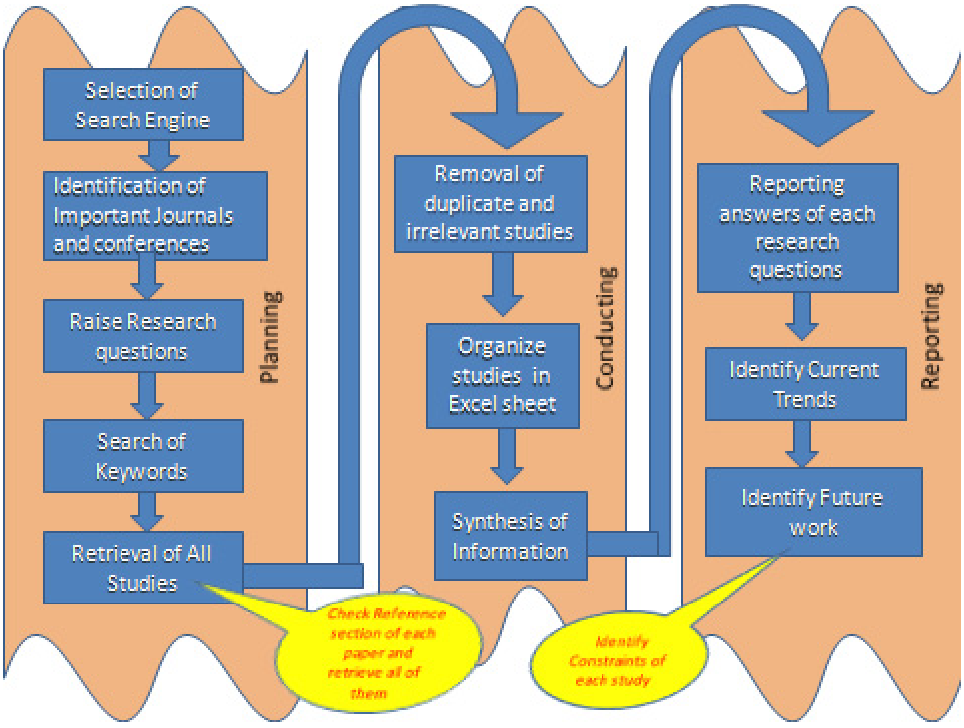
Selection of Search Engine
Identification of Important Journals and conferences
Raise Research questions
Search of Keywords
Retrieval of All Studies
Planning
Check Reference section of each paper and retrieve all of them
Removal of duplicate and irrelevant studies
Organize studies in Excel sheet
Synthesis of Information
Conducting
Identify Constraints of each study
Reporting answers of each research questions
Identify Current Trends
Identify Future work
Reporting

Fig. 1. Review methodology.

4. Detail Planning for Review

4.1. *Selection of search databases*

We began our search with the university website resource and EBSCO discovery services provided by the University Information Resource Centre (UIRC) of Guru Gobind Singh Indraprastha (GGSIP) University. The search process was accomplished in two stages. Firstly, search string such as "software maintainability" was used to get thousands of results. In this process, Google Scholar, Scopus, Science Direct, Springer, ACM Digital Library, IEEE Xplore, Wiley, Web of Science and Compendex were identified. In the second stage, publications in the important journals and conferences were identified. We restricted our search to the period from January 1991 to October 2015 only.

4.2. *Identification of important journals and conferences*

Before picking up good journals or conferences, their quality was determined through several perspectives to make the contents trustworthy. The basis for shortlisting of important journals and conferences in the said field in this review includes the number of citations, circulation of journals, the status of reviewers, impact factor and above all the quality of the editorial board. Journals shortlisted for review in the field of software maintenance along with their details are summarized in Table 1. Major

Table 1. List of important journals in the field of "software maintenance."

S. No.	Name of the journal	Published by	Impact factor
1.	*ACM Transactions on Software Engineering and Methodology*	ACM	3.958
2.	*IEEE Transactions on Software Engineering*	IEEE	3.569
3.	*Computer Science — Research and Development*	Springer	2.128
4.	*International Journal of Computer Science and Engineering Research and Development*	PRJ Publication	1.9022
5.	*Empirical Software Engineering*	Springer	1.854
6.	*Automated Software Engineering*	Springer	1.4
7.	*Asian Journal of Information Technology*	Medwell Publishing	1.2679
8.	*Journal of Systems and Software*	Elsevier	1.117
9.	*Software Quality Journal*	Springer	0.974
10.	*Journal of Software Maintenance and Evolution: Research and Practice*	John Wiley and Sons	0.844
11.	*International Journal of Software Engineering and Its Applications*	SERSC	0.7621
12.	*International Journal of Software Engineering and Knowledge Engineering*	World Scientific	0.447
13.	*International Journal of Software Engineering*	Software Engineering Competence Center	0.219
14.	*ACM SIGSOFT Software Engineering Notes*	ACM	NA

conferences of international repute that address issues in the field of software maintenance were also identified. Certain good conferences are *International Conference on Software Engineering* (*ICSE*), *International Conference on Software Maintenance* (*ICSM*), *Computer Software and Application Conference* (*COMPSAC*), *International Symposium on Empirical Software Engineering and Measurements* (*ESEM*), *International Conference on Program Comprehension* (*ICPC*), *Object-Oriented Programming, Systems, Languages and Applications* (*OOPSLA*), *European Conference on Object-Oriented Programming* (*ECOOP*), *World Conference on Reverse Engineering* (*WCRE*) and *European Conference on Software Maintenance and Re-engineering* (*ECSMR*). The research studies included in the proceedings of these conferences were also taken into consideration.

4.3. *Formulation of research questions*

Practitioners are consistently haunted by the fact that maintenance consumes a lion's share of the total project cost. After the research of almost four decades, we are in a position that we can predict the maintenance behavior of the software using mathematical prediction models in the early phases of SDLC on the basis of its design metrics. After the first round of investigations, next step was to raise certain research questions to identify the trends and scope of future research. Careful selection of research questions is very important as it helps us throughout our study from getting lost or being deviated off-track while navigating through vast information and identifying the related information from the shortlisted papers. Table 2 summarizes the list of important research questions set in the current review.

Table 2. List of research questions.

S. No.	Research questions
RQ1	Which techniques have been used for the software maintainability predictions? To identify the prevalent statistical techniques, machine learning (ML) techniques and evolutionary techniques (ETs).
RQ2	Which metrics are found useful and most cited? To identify the most influential metrics among various proposed metrics suites.
RQ3	What are the various tools prevalent in the industry for metrics collection? How to calculate values of design metrics to measure various design characteristics of any given software.
RQ4	What kinds of datasets are being used for the SMP? To be able to identify prevalent datasets and the need of new datasets.
RQ5	What kinds of prediction accuracy measures are used? To be able to identify the prevalent prediction accuracy measures used to judge the performances of prediction models.
RQ6	Whether the performance of ML techniques or ETs is better than the statistical techniques? To be able to compare the performances of the statistical, ML and ET.
RQ7	Is there any effect of refactoring on the software maintainability? To be able to identify various refactoring methods that could improve the code quality.
RQ8	How can we measure the 'maintainability'? Although subjective in nature, its answer would help us in identifying methods to measure maintainability.
RQ9	What are the advantages of SMP? Identify the advantages of SMP at early phases of SDLC.

4.4. *Search of keywords*

While conducting this systematic review, our intention was to compile all the papers on the subject under study and present a holistic view towards the end. The papers which include keywords such as 'software maintenance', 'software maintainability', 'software design metrics' and 'maintainability index' (ML) were identified during the initial search. After going through the contents, we went into further details and in the next phase other keywords such as 'changeability,' 'modifiability,' 'refactoring' and 'prediction modeling' were also used to refine our search process. Since our aim was to narrow down on the 'software maintainability prediction,' articles working on 'change-proneness,' 'fault-proneness,' 'error-proneness' and 'defect-prediction' were deliberately avoided. Final search string formulated as below:

> (Software Maintainability OR Software Maintenance) AND (Prediction OR probability) AND (Classification OR Regression OR Machine-Learning OR Artificial Neural Network OR Tree Net OR Multiple Regression OR Decision Tree OR Support Vector Machine OR Evolutionary Algorithm) OR (Software Maintenance AND Refactoring) OR (Accessing Software Maintenance) OR (Software Metrics AND Software Maintenance).

4.5. *Retrieval of studies*

Papers were retrieved by a team of seven members led by two Senior Assistant Professors, one from GGSIP University and another from Delhi Technological University (DTU) in India. All important studies were retrieved through respective digital libraries. During this process, reference section of each study was further explored to find relevant research publications/reports which were further retrieved and organized for review.

5. Conducting Review

During the conduct of the systematic review, all papers were studied thoroughly and adjudged against the research questions as compiled in Table 2. Constraints and limitations present in each of the shortlisted study were also identified and compiled which give directions for future research. Detailed synthesis of information took place during this stage only. We set certain criteria based on RQs and continue accessing each paper based on the above-mentioned criteria.

5.1. *Removal of duplicate and irrelevant studies (inclusion and exclusion of studies)*

All the shortlisted studies were arranged in chronological order so that the duplicate studies could be immediately identified and removed. Further, contents of the articles were thoroughly scanned and irrelevant studies were removed as discussed amongst the authors of this study. Inclusion and exclusion criteria were drawn directly from the research questions raised in Table 2. The review criterion was set on the basis to find some reasonable relevant contents in the context of software maintainability. Only those papers were shortlisted which predict software maintainability by proposing some models, measure maintainability using design metrics, discuss some empirical investigations or perform some activity which had a direct or indirect impact on maintainability such as refactoring, documentation, etc. along with their consequences on maintainability. Any paper diverting towards change-proneness or error-proneness was simply dropped. Criterion was kept neither too narrow which may result in an over-exclusion threat nor too broad as it may include poor or irrelevant studies.

Inclusion criteria:

- Empirical studies using the machine learning techniques.
- Empirical studies comparing the performance of machine learning techniques and statistical techniques.
- Empirical studies proposing some hybrid techniques by combining machine learning techniques with some non-machine learning techniques.

Exclusion criteria:

- Studies on software maintainability without empirical analysis of results.
- Empirical studies in which dependent variable other than 'change' was used.
- Studies using the machine learning techniques in any other context.
- Review studies [5–9].
- Replicated extended paper of the conference into the journal by the same author, however, utmost care was taken to identify whether the results are different, in which case both the studies were considered for the review.

Initially, 179 studies in total were retrieved using various search engines on the basis of the keywords identified for the search in Sec. 4.4. By applying the above-mentioned inclusion–exclusion criteria, 108 studies were shortlisted. In the next phase, we filtered out the irrelevant studies from the huge pool of available information by checking against the quality assessment criteria as presented in Table 3 to weigh the relevance of the study.

If the paper is qualified, it is given 1 mark, 0.5 marks for being partly qualified and 0 for not being qualified. The final score was calculated for each of the studies after adding the scores obtained for each of the quality assessment measures. Hence, a study could have a maximum score of 12 and a minimum score of 0. All those studies which scored less than 5 were again dropped from the review process. Many brainstorming sessions were conducted and 12 studies [134, 158–168] were further dropped and finally 96 primary studies were only shortlisted for review on software maintainability.

Table 3. Quality assessment measures.

S. No.	Research questions	Score Yes/Partly/No
Q1	Whether the aims of the research study were clearly stated?	
Q2	Whether the independent variables were clearly defined?	
Q3	Whether the data collection procedures were clearly defined?	
Q4	Whether any tool was used to collect the variables? If yes, is it explained?	
Q5	Whether the use of prediction techniques was clearly defined and justified?	
Q6	Whether threats to validity in the empirical study were clearly specified?	
Q7	Whether the adopted research methodology is repeatable?	
Q8	Whether the study is referring to a specific type of maintenance or it is picking maintainability as a whole problem?	
Q9	Whether comparison between performances of various techniques was conducted?	
Q10	Whether the proper and relevant literature survey was conducted?	
Q11	Does the study have consistent and adequate citations over the years?	
Q12	Whether prediction accuracy measures were clearly defined and used to measure the outcome in the study?	

5.2. *Data extraction*

"Place for everything and everything would be in place" as stated by the famous management expert Charles A. Goodrich was taken as a guiding principle to organize all shortlisted articles for easy and quick access and retrieval before setting up the stage for conducting such an extensive review. We prepared a database containing various attributes such as author's name, article name, date of publication, the source of publications (journal/conference/position paper/symposium/white paper), keywords, article's abstract and remarks. A separate column was also maintained using 'hyperlink' for creating a link to the corresponding files stored in a separate folder for easy access. No file was stored without creating a record in the database. We found this model to be very comfortable. A detailed synthesis of information present in each of the study was performed to find the answers of all RQs. Table 4 presents the list of shortlisted articles with the respective authors and reference numbers. It also assigns an identifier to each of the shortlisted studies which are further referred in the rest of the paper.

We examined all the shortlisted studies from numerous perspectives and tried to identify the relationship between these studies. If the collection of studies states similar or comparable viewpoints, it helped us in providing the evidence before reaching any generalized conclusions. Visualization techniques were also used as they present a considerably larger amount of data in a compressed form using a picture which is always worth more than a million words. In the present study, it helped us in quickly absorbing, interpreting and enhancing the clarity by proving aesthetic appeal to the compiled data. We have used various visualization techniques such as line graph, pie chart, bar chart, etc. to categorize various methods, models, performance measures, metrics and tools in the present study.

Table 4. List of shortlisted studies.

Stu. ID	Author	Ref.	Stu. ID	Author	Ref.	Stu. ID	Author	Ref.
S1	Aggarwal *et al.*	[63]	S33	Hirota *et al.*	[94]	S65	Prasanth *et al.*	[126]
S2	Aggarwal *et al.*	[64]	S34	Jeet *et al.*	[95]	S66	Prechelt *et al.*	[127]
S3	Aggarwal *et al.*	[65]	S35	Jin and Liu	[96]	S67	Rajaraman and Lyu	[128]
S4	Arisholm and Sjoberg	[66]	S36	Jorgensen	[97]	S68	Rana *et al.*	[129]
S5	Baker *et al.*	[67]	S37	Kabaili *et al.*	[98]	S69	Ramil and Lehman	[130]
S6	Balogh *et al.*	[68]	S38	Kataoka *et al.*	[99]	S70	Ramil and Smith	[131]
S7	Bandi *et al.*	[69]	S39	Kaur *et al.*	[100]	S71	Riaz *et al.*	[132]
S8	Banker *et al.*	[70]	S40	Kaur and Kaur	[101]	S72	Riaz *et al.*	[133]
S9	Baqais *et al.*	[71]	S41	Kaur *et al.*	[102]	S73	Schneberger	[134]
S10	Basgalupp	[72]	S42	Kumar and Dhanda	[103]	S74	Schneidewind	[135]
S11	Basili *et al.*	[73]	S43	Kumar	[104]	S75	Sheldon *et al.*	[136]
S12	Bhattacharya *et al.*	[74]	S44	Kemerer and Slaughter	[105]	S76	Shibata *et al.*	[137]
S13	Broy *et al.*	[75]	S45	Koten and Gray	[106]	S77	Stark *et al.*	[138]
S14	Chen and Huang	[76]	S46	Li and Henry	[107]	S78	Sneed	[139]
S15	Coleman *et al.*	[77]	S47	Lim *et al.*	[108]	S79	Sneed	[140]
S16	Dagpinar and Jahnke	[78]	S48	Lin and Wu	[109]	S80	Soni and Khaliq	[141]
S17	Dahiya *et al.*	[79]	S49	Lucia *et al.*	[110]	S81	Stavironoudis *et al.*	[142]
S18	Daly *et al.*	[80]	S50	Malhotra and Chug	[111]	S82	Sun and Wang	[143]
S19	Deißenböck *et al.*	[81]	S51	Malhotra and Chug	[112]	S83	Thongmak and Muenchaisri	[144]
S20	Deligiannis *et al.*	[82]	S52	Malhotra and Chug	[113]	S84	Thwin and Quah	[145]
S21	Elish and Elish	[83]	S53	Malhotra and Chug	[114]	S85	Upadhyay *et al.*	[146]
S22	Ferneley	[84]	S54	Malhotra *et al.*	[115]	S86	Velmourougan *et al.*	[147]
S23	Fioravanti and Nesi	[85]	S55	Misra	[116]	S87	Vivanco and Pizzy	[148]
S24	Floris *et al.*	[86]	S56	Misra and Sharma	[117]	S88	Welker *et al.*	[149]
S25	Genero *et al.*	[87]	S57	Mutanna *et al.*	[118]	S89	Wang *et al.*	[150]
S26	Grady	[29]	S58	Niessink and Vliet	[119]	S90	Xing and Stroulia	[151]
S27	Granja and Garcia	[88]	S59	Oman and Hagemeister	[120]	S91	Yamashita and Moonen	[152]
S28	Hanenberg *et al.*	[89]	S60	Oman and Hagemeister	[121]	S92	Ye *et al.*	[153]
S29	Harrison *et al.*	[90]	S61	Ping	[122]	S93	Ying *et al.*	[154]
S30	Hatton	[91]	S62	Pizka and Deissenboeck	[123]	S94	Zhang *et al.*	[155]
S31	Hays and Zhao	[92]	S63	Polo *et al.*	[124]	S95	Zhou and Leung	[156]
S32	Hegedus	[93]	S64	Prasanth *et al.*	[125]	S96	Zhou and Xu	[157]

5.3. *Distribution of papers*

All the shortlisted studies were divided into three parts as per the source in which they are published, i.e. published in journals, published in conferences and others which include book chapters, technical reports, white papers, study material, symposium, etc. Figure 2 depicts the distribution of the papers as per the three categories.

5.4. *Year of publication*

Maintainability was first introduced by Belady and Lahman [12] who are also known as the fathers of this field. During the period from 1969 to 1990, researchers concentrated more on good programming rather than designing to make the software maintainable [13–15]. Problems in maintenance process were identified by Lientz and Swanson [16], Martin and McClure [17] and Nosek and Palvia [18], to name a few, and many metrics were suggested [19, 20] to measure procedural languages. Rombach [21] argued that all those metrics which were useful for procedural language cannot be applied blindly on object-oriented (OO) languages. Practitioners started giving importance to design rather than code and proposed various metrics to measure different design aspects of OO paradigm such as coupling, cohesion, polymorphism, inheritance, etc. Further, with the help of empirical investigations, strong correlation between software design metrics and subsequent maintainability was

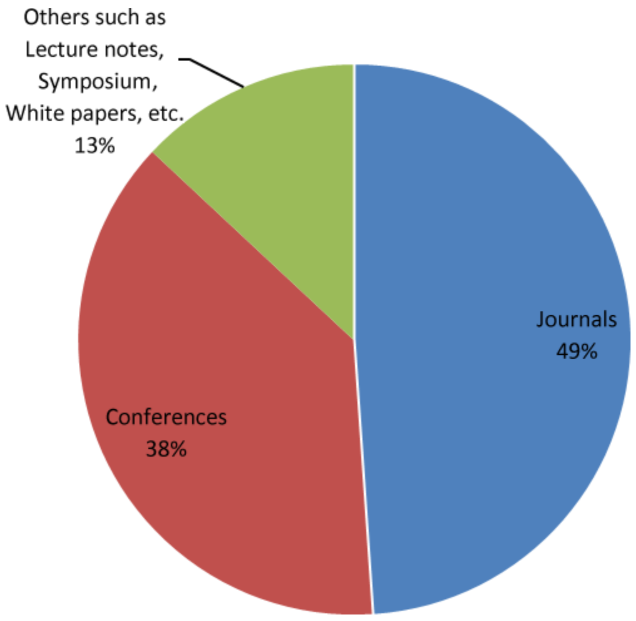

Fig. 2. Distribution as per the source of publication.

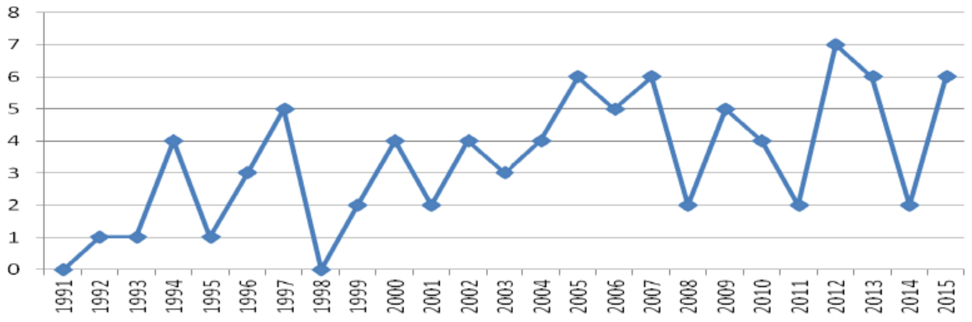

Fig. 3. Year-wise distribution of studies.

identified. Many metrics suites were proposed during this period like Chidamber and Kamerer [22], Li [23], Chen and Lum [24], e Abreu and Carapuça [25], Lorenz and Kidd [26] and Tang *et al.* [27]. Critical analysis of such metrics suite has also been done by Grady [28], Chucher and Martin [29], Mayer and Hall [30] and Hitz and Montazeri [31]. Many empirical investigations were also carried out to verify and validate the proposed metrics suites [23–31] in terms of their effect on maintainability. From the year 2000 onwards, researchers agreed that we can measure maintainability by measuring the number of changes during operation. Basili *et al.* [32] suggested that the design metrics can be used for quality prediction at quite early stages of SDLC. The distribution of years for all the shortlisted studies from the year 1991 to 2015 is presented in Fig. 3.

It is clearly evident that the research on software maintainability has been quite consistent over the years. When we analyzed the contents we found that from the year 1991 to 2004 few important studies built statistical models to predict maintainability using design metrics as they argued that since the value of change would be available only during operations and it is too late by then, we should be able to predict maintainability with the help of design metrics at the earlier stages of SDLC. From the year 2005 onwards, a shift has been observed towards the use of machine learning algorithms in prediction modeling. Hybrid models were also applied by many researchers during this period in prediction modeling process. Recently, interest in nature-inspired algorithms called as evolutionary methods has also been seen due to their obvious advantages as elaborated further in Sec. 6.6.

6. Reporting of Review

We have analyzed all shortlisted studies in the last two and half decades collected from journals, conferences, symposiums, etc. in the field of software maintainability with an aim to find answers of all the research questions raised in Sec. 4.3. The following sub-sections present summarized facts and answers to all research questions.

6.1. *RQ1: Techniques used for software maintainability prediction*

Researchers are experimenting new prediction models every day to predict the maintainability of software more precisely.

To establish the relationship between software design metrics as the independent variable and maintainability as the dependent variable, various techniques have been practiced in last two and half decades which can be broadly classified in five categories as summarized in Table 5. When we chronologically arranged the choice of

Table 5. Types of modeling/methods used to judge maintainability.

S. No.	Type of algorithms	Prominent prediction techniques	Respective study identifier
1.	Statistical algorithms	Binary logistic regression, multivariate binary logistic regression (MBLR), stepwise logistic regression, hidden Markov model (HMM), multiplicative adaptive spline regression, random forest, naive bayes classifier, multilayer perceptrons, bagging, boosting, projection pursuit regression and support vector machine	S7, S8, S12, S16, S22, S23, S27, S31, S35, S36, S38, S40, S41, S46, S57, S59, S63, S66, S84, S88, S89, S92, S95 and S96
2.	Machine learning algorithms	Neural network (NN)-based models, artificial neural network (ANN), fuzzy inference system (FIS), adaptive neuro FIS (ANFIS), fuzzy algorithms, fuzzy repertory table (FRT), Bayesian networks (BNs), TreeNets, decision trees (DTs), data clustering (DC), self-organizing map (SOM), generative topographic map (GTM), case-based reasoning (CBR), association rules (AR), etc.	S1, S2, S3, S13, S21, S34, S35, S39, S41, S43, S45, S49, S51, S53, S68, S72, S84 and S92
3.	Nature-inspired techniques	Evolutionary algorithms, genetic programming, ant colony optimization (ACO), swarm particle intelligence, simulated annealing and hill climbing	S5, S6, S10, S17, S50, S82 and S87
4.	Expert judgment	Feedbacks, questionnaires, surveys, opinions, etc.	S12, S18, S20, S22, S25, S29, S47, S55, S65, S66, S73 and S81
5.	Hybrid techniques	Genetic algorithm with neural network (GANN), neural network evolutionary programming (NNEP), evolutionary with fuzzy network, (GFS-GSP), genetic-based fuzzy rule base construction and membership functions tuning (GFS-RB) and evolutionary with neural network	S9 (evolutionary + neural), S10 (evolutionary + decision tree), S17 (fuzzy + genetic), S25 (UML + expert judgment), S47 (hybrid network with parallel computing), S82 (regression with neural network) and S92 (multiple classifier (MC))

models applied for SMP, it was quite evident that during the initial period statistical methods such as linear regression, MBLR, SLR, HMM, etc. were used. These methods were not only highly mathematical in nature but also unable to handle noise present in the data. In the later phases from the year 2000 onwards, more robust models based on machine learning techniques such as ANN [65, 145, 156] were applied. It works as an excellent alternative to logistic regression since it offers a number of advantages as follows: less formal statistical training is required, all possible interaction between independent and dependent variables can be identified and complex nonlinear relationship can be judged using ANN. In this method, available data from history is often divided into three sets: learning set, validating set and testing set. Learning set is the sequence in which the dependent variable is shown to the network during the learning phase and weights are assigned to the independent variables depending upon the values of the independent variables. The network continuously adapts itself to achieve the particular value of the dependent variable and during this process, values of the assigned weights are changed. The difference between the required output and actual output is measured using validating set to identify whether the learning can be finished. Lastly, testing set is used to test whether the network is capable of predicting for the unforeseen data or not. Various variants of ANN such as feed forward neural network (FFNN) [65, 145], back propagation network (BPN) [, Kohonen self-organizing network (KSON), radial basis function (RBF) network [100], probabilistic neural networks (PNN), general regression neural network (GRNN) [112, 145], ANFIS [100] were also explored for their predictive capabilities.

From the year, 2012 onwards, hybrid methods have proven to be even better in prediction, for example, multiple classifier (MC) is used by Ye *et al.* [153] and group method of data handling (GMDH) is used by Malhotra and Chug [114]. Nature-inspired algorithms are used in prediction modeling in various fields of software engineering other than maintenance such as ACO was applied by Azar and Vybihal [33] in software quality prediction, particle swarm optimization (PSO) was applied by Saed and Kadir [34] in software performance prediction and genetic model was applied by Burgess and Leey [35] for software effort estimation. However, the applications of these models for software maintainability prediction were reported to be very less wherein only seven studies could be found [S5, S6, S10, S17, S50, S82 and S87]. Well-accepted, established, recognized and generalized prediction method is still awaited by the software industry.

6.2. *RQ2*: *Metrics suite used in software maintainability prediction*

The relationship between software design metrics and corresponding maintainability has been proposed and validated by many researchers. With the help of many empirical studies, it has been established that the quality of the software design, as well as code, is very important to enhance software maintainability. We observed that during the period from 1969 to 1990, traditional metrics like function point, Halstead software science [19] and McCabe's Cyclomatic Complexity (CC) [20] were used to judge the quality of procedural languages. Quantitative measure to calculate maintainability index proposed by Welker *et al.* [149] is given in Eq. (1):

$$\mathrm{MI} = 171 - 5.2 * \ln(\mathrm{HV})\, 0.23 * \mathrm{CC} - 16.2 * \ln \mathrm{LOC} + 50 \sin \sqrt{2.4 * \mathrm{Comments}}, \quad (1)$$

where HV is Halstead volume metric [19], CC is Cyclomatic Complexity metric [20], LOC is counted as a line of code and COM is a percentage of comment lines. With the invention of OO paradigm, traditional metrics mentioned above were no longer effective since other characteristics such as inheritance, coupling, cohesion and polymorphism present in the code take the charge. Subsequently, the necessity was highlighted by Rombach [21] and new metric suite to measure the design characteristics of OO software was proposed by Chidamber and Kamerer [22] famously known as C&K metrics suite. Some revisions into the C&K metric suite were made by Li [23] and two more metrics were added. Although C&K metric suite was criticized by many researchers [29–31] for a number of reasons such as lack of cohesion (LCOM) is not a true representation of cohesiveness, however, the proposed metric suite became quite popular and was further referred by many researchers in their empirical studies from the year 1990 to date as compiled in Table 6. During the same period, many researchers proposed fresh metrics suites while others revised or modified the existing ones. We collected all proposed metrics suites (resulted in 41 studies) and filtered highly cited studies in the field of software maintenance, resulting in 16 studies as summarized in Table 6. It is quite evident from the table

Table 6. Metrics suites proposed and used in empirical investigations.

S. No.	Studies	Ref	Referred in following studies	Total
1.	Halstead	[19]	S23, S59, S60 and S81	5
2.	McCabe	[20]	S23, S59, S60 and S81	5
3.	Chidamber and Kamerer	[22]	S3, S9, S16, S20, S21, S23, S35, S39, S40, S45, S50, S51, S52, S53, S65, S68, S84 and S95	9
4.	Li	[23]	S39 and S55	10
5.	e Abreu and Carapuça	[25]	S55	1
6.	Lorentz and Kidd	[26]	S55	1
7.	Tang *et al.*	[27]	S84	2
8.	e Abreu and Carapuça	[25]	S55	
9.	Aggarwal *et al.*	[64]	S1, S17 and S48	3
10.	Chen and Huang	[76]	S14	1
11.	Fioravanti and Nesi	[85]	S23	1
12.	Li and Henry	[107]	S3, S9, S16, S20, S21, S23, S35, S39 and S40	9
13.	Lin and Wu	[109]	S48	2
14.	Prasanth *et al.*	[126]	S64 and S65	2
15.	Sheldon *et al.*	[136]	S75	2
16.	Stavironoudis *et al.*	[142]	S81	2

that the metrics suites proposed by Chidamber and Kamerer [22] and Li and Henry [107] are the most commonly used ones in empirical validations. We also observed that many researchers empirically evaluated the effect of only one particular design metric on maintainability, for example, inheritance was evaluated by Daly *et al.* [80], Harrison *et al.* [90], Prechelt *et al.* [127] and Sheldon *et al.* [136], coupling by Rajaraman and Lyu [128], UML diagram by Genero *et al.* [87], cohesion by Kabaili *et al.* [98] and code metrics by Polo *et al.* [124]. Inconsistencies in metric naming convention were also seen, for instance, somewhere two different names viz. Depth of Inheritance Tree (DIT) and Depth of Inheritance (DIH) represent the same concept as both represent inheritance to measure the longest distance from root node to leaf node, whereas at some other places the same name is used to represent two different concepts (DC is used to represent descendent class as well as for measuring direct cohesion).

6.3. *RQ*3: *What are the various tools to measure design metrics?*

In order to find the answer for RQ2, we observed that a large number of metrics has been suggested in the literature to measure the design characteristics of the given software. Numerous free as well as proprietary tools have also been developed to collect the values of these design metrics from the given source code so that specific characteristics of a software code could be measured. In Table 7, we have summarized few prevalent tools along with their details.

6.4. *RQ*4: *Kind of dataset used for empirical validations*

Many researchers have conducted empirical studies in part to prove that the values of design metrics significantly affect maintainability. All these studies are either based on small projects, proprietary software datasets', open source software, datasets published by NASA, PROMISE [44] repository or taken from students' projects as compiled in Table 8.

To empirically validate and evaluate the effect of each metrics on software maintainability, even though the availability of data is still a concern, some research studies have taken real-life data whereas some studies have used the dataset proposed by Li and Henry [108] from two commercial software packages namely user interface management system (UIMS) and quality evaluation system (QUES). Maintenance efforts are generally calculated by counting the number of lines added, deleted or modified during operations. The source codes of old and new versions were collected and analyzed against modifications made in every class. Values of OO software design metrics suite were calculated and combined with corresponding changes made into that class so as to generate datasets which were further divided into 3:1:1 for training, testing and validation, respectively, during model implementations. The accuracy was measured by comparing the actual value with predicted value generated through mathematical prediction model.

Table 7. List of important tools to measure design metrics.

Tool name	Free/proprietary	Developed by	Remarks
'C and C++ code counter' (CCCC) [36]	Open source	Tim Littlefair	It analyzes C++ and Java and generates reports for line of code, C&K metrics suite and Henry and Kafura metric suite.
Dependency finder [37]	Open source	Jean Tessier	This application comes as a command-line tool for analyzing compiled Java code and creating dependency graphs. It is also used for computing OO software metrics to give an empirical quality assessment of given code.
Eclipse Metrics plug-in 1.3.6 [38]	Open source	Frank Sauer	It is a plug-in for Eclipse platform which provides a calculation of metrics and dependency analyzer.
Eclipse Metrics plug-in 3.4 [38]	Open source	Lance Walton	
Vizz Analyzer [39]	Open source	Rüdiger Lincke	It is a framework designed to support maintenance and re-engineering.
Understand [40]	Proprietary	Proprietary	It is a reverse engineering, code exploration and metrics tool for Java source code. It is a static analysis tool for maintaining, measuring and analyzing critical or large code bases.
Analyst4j [41]	Proprietary	Proprietary	It comes as an Eclipse plug-in and offers an environment to visualize code quality with the help of metrics and charts. Apart from helping in estimating efforts, it also helps in identifying problem areas and respective refactoring methods.
OOMeter [42]	Software Metrics Research Group (SMRG)	Alghamdi *et al.*	It can be used to quantitatively measure a number of quality attribute including requirement specifications and design models. It supports OO metrics such as coupling, cohesion and code metrics such as LOC.
CKJM [43]	Open Source	Diomidis Spinellis	C&K Java Metrics is an open source command-line tool which calculates the metrics by processing the bytecode.

Table 8. List of datasets used in empirical validations and respective studies in references.

S. No.	Datasets	Referred in studies	Total
1.	Li and Henry	S3, S16, S21, S39, S40, S45, S51 and S84	8
2.	Open source	S9, S12, S55 and S66	4
3.	Proprietary software	S10, S14, S15, S20, S22, S36, S48, S52, S64 and S65	10
4.	Student's projects	S2, S25, S29 and S48	4

6.5. *RQ5: Accuracy measures to judge the performance of prediction models*

For the prediction of maintainability using whatever tools, methods or datasets suggested in the literature, it is observed that the predicted values of the dependent variables on test data are not very close to actual values. Hence, a number of statistical measures have also been proposed by Conte *et al.* [45], Kitchenham *et al.* [46, 47], Fenton and Neil [48] and Hatton [91] to measure the prediction accuracies. All these studies [45–48, 91] have presented some measures to ensure that the accurate prediction is not due to sheer coincidence rather it exists in reality by proposing some formulas with their corresponding interpretations.

Much attention has been paid for the development models capable of more accurate and precise predictions while adhering to few fundamental characteristics like the models must be independent of the language and technology, simple to calculate, straightforward for interpretations and easy to understand. Widespread parameters proposed to measure the significance of prediction model are residual error (RE), absolute residual error (ARE), mean of ARE (MARE), Standard deviation of ARE (StdevARE), magnitude of relative error (MRE), mean magnitude of relative error (MMRE), MaxMRE, Pred (q), R-square, P-values and root mean square error (RMSE) as summarized in Table 9. We found that MRE and MMRE are quite prevalent for measuring the prediction accuracies and used by many researchers [65, 71, 72, 74, 76–78, 82–84, 100, 101, 112–116, 126, 127, 145, 148, 157] to adjudge the performance of their prediction model.

6.6. *RQ6: Performance comparison of statistical techniques, ML techniques and ETs for SMP*

We analyzed all kinds of empirical studies carried for SMP and compared the prediction accuracies achieved by all the modeling techniques. Due to the diversity in the performance measures considered in different studies, we could not get a clear picture as different studies have considered different measures. MRE and MMRE were used by most of the studies whereas R-square and P-value were used by only study.

ML techniques-based prediction models were found to be better than statistical techniques-based prediction models. Six such studies were found in which MMRE is used as the prediction accuracy measure to judge the performances based on ANN as

Table 9. List of commonly used prediction accuracy measures.

Name	Definition	Referred in studies	Total
MRE	Measure of the discrepancy between actual values and predicted value	S35, S45, S51, S52, S53, S84 and S95	4
MARE	Normalized measure of the discrepancy between actual values and predicted value	S35, S39, S51, S52 and S53	2
MMRE	Average relative discrepancy	S3, S21, S45, S51, S52, S53, S84 and S95	5
RMSE	Root mean square error	S3	1
Pred	What proportion of the predicted values have MRE less than or equal to specified value	S21, S45, S51, S52, S53 and S95	2
R-square	Measure of how well the variation in the output is explained by the targets	S2, S21 and S35	2
P-values	Used for testing the hypothesis of no correlation	S3, S21 and S35	3
Pearson coefficient of correlation	Standard deviation of two series is compared	S35	1

compiled in Malhotra and Chug [112]. In this regard, its values achieved are as 0.59 using ANN [156], 0.403 using BPN [106], 0.265 using ANN [65], 0.23 using PNN [112], 0.765 using GRNN [145] and 0.242 using ANFIS [100]. Overall, in each of these six studies, irrespective of the kind of dataset used while making a prediction model, it is empirically proved that ML-based prediction techniques perform much better than statistical techniques.

Use of recently developed nature-inspired algorithms in maintainability prediction is also found to be very limited. Only one study by Malhotra and Chug [112] has applied genetic algorithms for software maintainability predictions although these techniques are successfully applied in related areas of maintainability like the prediction of development effort by Balogh *et al.* [68], prediction of maintenance effort by Basgalupp *et al.* [72], prediction of preventive maintenance by Sun and Wang [143] and identifying software metrics by Vivanco and Pizzi [148]. Evolutionary algorithms are found to be superior to machine learning algorithms due to various aspects which are listed below:

- Evolutionary algorithms are generally more robust in nature because there are no restrictions on the definition of the objective function.
- Use of evolutionary algorithms removes the possibility of biased results.
- Search for an optimized solution is performed in a parallel manner.
- Since the results are only influenced by objective function as well as fitness function, there is no such requirement of auxiliary knowledge.
- They can handle a large amount of noise present in the data as the transition rules are probabilistic in nature and not deterministic in nature.

- They are more capable of working in large and discontinuous search space and able to achieve global optima instead of local ones.
- Evolutionary algorithms can provide a number of potential solutions to a given problem and final choice always lies with the user.

6.7. *RQ7*: *Effects of refactoring on software maintainability*

Refactoring is the maintenance process in which the design of a OO software code is improved using various methods without affecting its behavior [49–56]. It is an effort to improve the quality of the software either by improving the design or by improving readability and understandability while preserving the correctness of the program. Many refactoring methods have been suggested in the literature and each has a particular purpose and corresponding effect. Refactoring is a process in which the internal structure of the OO software system is improved and complexity of the code is reduced however the external behavior of the system remains the same. The source code becomes simpler and easier to maintain as the changes made into the code are very systematic in nature. Few well-known methods of refactoring are dead code elimination, clone code removal, extract method, lazy classes, pull-up method, push-down method, hide methods, renaming, etc. Each method has its own effect on software quality attributes such as extensibility, modularity, reusability, complexity, maintainability and efficiency. It is important and essential to analyze the effects of refactoring on these quality attributes. Many studies have been undertaken where the effect of refactoring has been analyzed on software maintainability [49–56]. When we analyze the findings of these empirical investigations, it is found that even though refactoring is a very tedious process and might introduce errors if not implemented with utmost care, it is still advisable to refactor the code frequently in order to enhance the maintainability of software. Project managers must take utmost care in identifying the opportunities of refactoring in large code while maintaining a perfect balance between re-engineering and over-engineering.

6.8. *RQ8*: *How can we measure 'maintainability'?*

Many ways have been proposed to access the maintainability as compiled by Berns [57]. Floris *et al.* [86] suggested many ways to save the maintenance cost and, in turn, the overall project costs. There is a consensus among researchers in this field that there should be some quantified value to measure software maintainability either at the process level, architecture level or at the code level. A range of software maintenance parameters such as mean time to repair (MTTR), mean corrective maintenance time (MCMT), mean preventive maintenance time (MPMT) and maximum corrective maintenance time (MaCMT) is available. As defined in ISO 9126 quality model, maintainability consists of external quality attributes, i.e. analyzability, changeability, stability and testability. Ramil and Smith [131] suggested measuring it in terms of the time taken when the failure was reported to the time

Table 10. Advantages of software maintainability predictions.

S. No.	
1.	Managers would be able to compare the productivity and costs among different projects.
2.	Managers would be able to do more effective planning of the use of valuable resources.
3.	Managers can take an important decision regarding staff allocation.
4.	Identify the maintenance process efficiency as it helps in keeping the maintenance cost under control.
5.	The threshold values of various metrics which drastically affect maintainability of software can be checked and kept under control so as to achieve least maintenance cost.
6.	Developers can identify the determinants of software quality and hence they can improve the design.
7.	Practitioners would be able to improve the quality of systems and thus optimize maintenance costs.

taken in repairing it. Jorgensen [97] proposed measuring it in terms of changeability, i.e. time taken to implement the changes. Many researchers [65–67, 79, 83, 102, 106, 126] measured maintainability by measuring the 'change,' i.e. number of lines of source code added, modified or deleted during operations.

6.9. *RQ9*: *Advantage of software maintainability prediction*

The idea of predicting the maintainability has some inborn problems due to the fact that it is very subjective in nature. Maintainability predictions help us to reduce system's repair time thereby reducing the downtime and increasing system availability as everything can be planned in advance. In Table 10, we have enlisted certain advantages of maintainability prediction.

7. Current Trends and Future Opportunities

Whenever an error occurs in any software, a certain amount of time is needed to correctly identify, isolate and remove the fault. The longer it takes to recover from the occurrence of an error, the higher will be the costs associated with software maintenance. From as early as 1969 to date, the field of software maintenance has evolved over a period of time. Each year many research publications are added to the already available vast amount of knowledge. We have arranged all the qualified studies chronologically and observed the trend. After conducting the assessment of the results obtained in each of the shortlisted studies, we have evaluated them and identified constraints present in order to identify future directions in this field. Based on the results of primary studies, few emerging sub-fields in the field of software maintenance are highlighted as below:

7.1. *Addition of dynamic metrics along with static metrics*

In the previous section, while finding the answer for RQ2, we have discussed various kinds of metrics suites proposed, evaluated and validated empirically by many researchers. As evident from Table 6, C&K metric suite and Li and Henry metric suite are the most popular ones and used by many researchers in their respective studies. We have observed that unfortunately both the metric suites are actually static in nature. Since some lines of source code might not execute during execution of the software depending upon the inputs supplied and other conditions, hence relating these static design measures to maintainability may not be correct. As Bieman and Ott [58] has measured the function cohesion in their study, it would be of great interest to evaluate and analyze the dynamic dependencies between various software artifacts [59, 60]. One of the promising fields is to measure these design metrics dynamically instead of statically using Dynamic Lack of Cohesion (DLCOM), Dynamic Response For a Class (DRFC), etc.

7.2. *Equal importance to external quality attributes*

Even though C&K metric suite is quite popular among researchers, one limitation observed by us is that it takes into account only the internal design metrics while ignoring the importance of external quality attributes such as familiarity with the code, expertise level of programmer, development skills, etc. which are semantic in nature. Actually, internal design metrics suite is as per the specifications of ISO 9126 software quality model, therefore, studying the effect of external quality attributes on maintainability is another promising field which needs to be investigated.

7.3. *More metrics to measure current applications such as mobile applications, large databases and so on*

In earlier times, the data which is stored at the backend might have been accessed a couple of times a week, however with the increase in the use of mobile and mobile-based applications, now it is accessed multiple times per hour. As the software systems heavily use databases, hence we observed that C&K metric suite would not be adequate as it does not capture the database handling aspects of the applications. Another promising sub-field for research is to explore the significant set of metrics under the new circumstances wherein the applications are highly data intensive along with their respective empirical validations. One such empirical study has been carried out by the authors Malhotra and Chug [114] in the past. In this study, equal attention to the database accesses was given and the new metric suite was empirically proposed and verified to be superior. Analysis of a lot of prevalent metrics to measure the system's maintainability and exploiting the possible combination of these metrics into an index for the system's maintainability are still a challenge, yet to be solved.

7.4. *Use of hybrid techniques with more emphasis on nature-inspired techniques*

Synthesis of results suggests that initially statistical prediction modeling techniques were in use which were later on over-powered by the machine learning prediction modeling and it is claimed by the researchers that they are better than statistical techniques as they can capture the quality as well as quantity present in the data available for training. In order to achieve more and more accuracy in the prediction, various versions of NN was used. In this review, we have identified the maximum use of NN and seven such studies were found. Regression was used five times whereas fuzzy model was used in only two studies. We also found here a gap in existing research as many modeling techniques are yet not explored for their prediction capabilities. Ensemble learners (e.g. bagging and boosting) and instance-based machine learning (e.g. K-Star) are few methods which have not yet been applied. The reader is also advised to explore the use of evolutionary algorithms for maintainability prediction and combining them with other fields such as data mining, expert systems, genetic algorithms, artificial intelligence, nature-inspired algorithms, etc.

7.5. *Prediction models for systems other than OO systems*

Creating the prediction model(s) that can realistically predict maintainability of applications other than OO system is also yet to be explored. Only two studies were found on relational database-driven software application by Riaz *et al.* [132, 133] and one study was found on aspect-oriented systems by Thongmak and Muenchaisri [144].

7.6. *Use of agile methods and their effect on maintainability*

Agile software processes [61, 62] such as eXtreme Programming (XP) are another sub-field of software maintenance which consist of four things: communication, simplicity, feedback and courage. Recently, a trend is also observed where researchers are taking help of this methodology in keeping maintenance cost constant over time. Although the datasets taken in both the studies of Knippers [61] and Poole and Huisman [62] were comparatively small and medium-sized, this methodology was proved to be the best for software maintenance work because the human factor is considered as the main component. More empirical investigations on large or very large systems with the help of industry–institute partnership are required to further explore the use of XP.

7.7. *Effect of modern development techniques such as component-based development on maintainability*

Modern development techniques claim to make the code maintainable such as component-based software development, product line development, model-based development and design patterns; however, additional research exploration is required to verify such claims.

7.8. *Academia–industry partnership needs to be expanded*

While finding the answer of RQ4, we found that in the process of empirical investigations, only 16% worked on students' projects, 16% on open source, 40% on proprietary software and remaining 28% worked on the data made available in research studies. Out of the 40% studies which used proprietary software, 82% used small datasets mostly in academia. Therefore, in order to improve the industrial relevance as well as the validity of the research, it is highly desirable that large-sized industrial software must be used in the datasets, their access to the researchers should be provided, more and more industry–academia collaboration in research should be made and whenever possible the dataset should be made public for carrying out the future research.

7.9. *More studies with datasets from open source codes available in abundance*

In continuation with the above point, lots of open source codes are available due to the obvious advantage of the internet. Study of open source codes and their characteristics, working on the datasets obtained from open source codes and further relating them to the maintainability constitute another promising field while making the prediction models and conducting the empirical studies.

7.10. *Judge maintainability using other quality measures*

In addition to the prediction accuracy measures as described earlier in Table 9 to adjudge the quality of the prediction models, other performance measures should also be used to review the prediction quality such as generalization capability, interpretability, etc. One of the future directions of research could be carried out in this area wherein overall evaluation of the prediction models can be achieved.

7.11. *Investigate the effects of refactoring on software maintainability*

Investigating the effects of refactoring on software maintainability is also a very promising field. Many empirical studies have been conducted to prove that local code restructuring process makes the software code more cohesive, less coupled and it becomes easier to read and maintain. It is anticipated by the researcher fraternity working in the field of software maintainability that such refactoring operation or software transformations would certainly improve the maintainability of software, however, it is so far undecided that which quality factors are to be improved by applying the shortlisted refactoring methods in specific order. Empirical studies are already being undertaken by many researchers [99, 116] to solve this mystery, still more empirical studies are required to be conducted.

7.12. *Effect of other activities on maintainability such as risk analysis and effort requirement*

Most of the empirical researches have focused on specific aspects such as programmer productivity and error count which are measured mainly for the short term. It would be an interesting study if undertaken to measure the amount of hours required for maintaining a program developed using agile software development methods when compared to the program developed using a traditional plan-driven approach over a long term, by setting up experiments to specifically test the impact of software development methods on its maintainability. One such study was found to be conducted by Prasanth *et al.* [126] in which effect of risk analysis on maintainability is studied.

7.13. *Identification of the optimum point when dropping the old one and developing new one is more viable*

For every product whether it is a hardware or software, over a period of time it certainly reaches a stage where maintaining it becomes a more costly affair than developing the new one. The research fraternity still needs to find the equilibrium point between maintaining the existing software versus scrapping the existing one and developing a new one.

7.14. *Solve the mystery involved in maintainability*

Undoubtedly, accessing the maintainability is a bit subjective in nature. The understanding of the intricacies involved in maintainability would certainly assist researchers in the future work. In order to make software architecture and components more maintainable, mysteries of maintainability have to be solved. More studies are required on this field as only one study was found by Broy *et al.* [75].

8. Conclusions

In this work, we performed an exclusive survey and systematic review of the studies published in the field of software maintainability since 1991. Over a period, different researchers have adopted a diverse range of software engineering paradigms and studied their consequences on maintenance cost. New models and innovative techniques have been introduced so that software maintenance prediction could be estimated more accurately. In the current paper, an effort has been made to review all these models, variables, programming practices, etc. and identify various important aspects which could greatly affect the maintenance effort. An extensive search was performed using nine digital libraries after identifying the primary studies in the said field. Overall 108 papers were shortlisted in the initial search out of which only 96 studies were found to be suitable and the rest were discarded. These studies were further examined with respect to 12 quality assessment criteria questions and compiled to explore and achieve new insights. Meaningful presentations of the vast collection of the collected data were made using tables and graphs. An attempt has also been made to provide recommendations, constructive guidelines, an overall overview as well as the opportunities and challenges to carry out the future research in the field of software maintainability for researchers and practitioners. Our paper would also be helpful to the beginners as they can study the concepts of the related area and use the results of this study to identify the complete list of relevant papers in the field. In the end, this survey could be considered as significant for its contributions as well as timely support to the research community.

References

1. IEEE, IEEE Standard: 828-1998 — IEEE Standard for Software Configuration Management Plans, IEEE Computer Society (1998).
2. IEEE, IEEE Standard: 1219-1993 — IEEE Standard for Software Maintenance, INSPEC Accession No. 4493167 IEEE Computer Society (1993).
3. T. D. Marco, *Controlling Software Projects: Management Measurement and Estimation* (Prentice-Hall, Upper Saddle River, 1986).
4. C. Jones, The economics of software maintenance in twenty-first century, research paper, Software Productivity Research, Inc. (2006).
5. M. Riaz, E. Mendes and E. Tempero, A systematic review of software maintainability prediction and metrics, in *Third Int. Symp. Empirical Software Engineering and Measurements*, 2009, pp. 367–377.
6. W. K. Tieng, M. H. Selamat, A. Azim, A. Ghani and R. Abdullah, Review of complexity metrics for object-oriented software products, *Int. J. Comput. Sci. Netw. Secur.* **8**(11) (2008) 314–321.
7. S. Ghosh, S. K. Dubey and A. Rana, A comparative study of the factors that affect maintainability, *Int. J. Comput. Sci. Eng.* **3**(12) (2011) 3763–3770.
8. J. Saraiva, A roadmap for software maintainability measurement, in *Proc. Int. Conf. Software Engineering*, 2013, pp. 1453–1455.
9. J. Saraiva, S. Soares and F. Castor, Towards a catalog of object-oriented software maintainability metrics, in *WeTSom: Proc. 4th Int. Workshop Emerging Trends in Software Metrics*, 2013, pp. 84–87.
10. B. Kitchenham and S. Charters, Guidelines for performing systematic literature review in software engineering, Technical report no. EBSE 2007-001, Keele University and Durham University, 2007.
11. L. Pickard, B. Kitchenham and S. Linkman, An investigation of analysis techniques for software datasets, in *Proc. Sixth Int. Software Metrics Symposium*, 1999, pp. 130–138.
12. L. Belady and M. M. Lehman, A model of large program development, *IBM Syst. J.* **15** (3) (1969) 225–252.
13. E. B. Swanson, The dimensions of maintenance, in *Proc. 2nd Int. Conf. Software Engineering*, 1976, pp. 492–498.
14. S. S. Yau and J. S. Colloefello, Some stability measures for software maintenance, *IEEE Trans. Softw. Maint.* **6**(6) (1980) 545–552.
15. S. S. Yau, J. S. Colloefello and T. MacGregor, Ripple affect analysis of software maintenance, in *Proc. IEEE Conf. Computer Science and Application*, 1978, pp. 60–65.
16. B. P. Lientz and E. B. Swanson, Problems in application software maintenance, *Commun. ACM* **24**(11) (1981) 31–37.
17. J. Martin and C. McClure, *Software Maintenance: The Problem and Its Solutions* (Prentice-Hall, 1983).
18. J. T. Nosek and P. Palvia, Software maintenance management: Changes in the last decade, *J. Softw. Maint., Res. Pract.* **2**(3) (1990) 157–174.

19. H. Halstead, *Elements of Software Science* (North-Holland, Amsterdam, 1977).
20. T. J. McCabe, A complexity measure, *IEEE Trans. Softw. Eng.* **72**(2) (1976) 308–320.
21. H. D. Rombach, Design metrics: Some lessons learned, *IEEE Softw. J.* (1990) 17–25.
22. S. F. Chidamber and C. F. Kamerer, A metrics suite for object-oriented design, *IEEE Trans. Softw. Eng.* **20**(6) (1994) 476–493.
23. W. Li, Another metric suite for object-oriented programming, *J. Syst. Softw.* **44**(2) (1998) 155–162.
24. J. Y. Chen and J. F. Lum, A new metrics for object-oriented design, *J. Inf. Softw. Technol.* **35**(4) (1993) 232–240.
25. F. B. e Abreu and R. Carapuça, Candidate metrics for object-oriented software within a taxonomy framework, *J. Syst. Softw.* **26**(1) (1994) 87–96.
26. M. Lorenz and J. Kidd, *Object-Oriented Software Metrics* (Prentice-Hall, 1994).
27. M. H. Tang, M. H. Kao and M. H. Chen, An empirical study on object-oriented metrics, in *Proc. Sixth Int. Symp. Software Metrics*, 1999, pp. 242–249.
28. R. B. Grady, Successfully applying software metrics, *Computer* **27**(9) (1994) 18–25.
29. N. I. Chucher and J. S. Martin, Comments on a metrics suite for object-oriented design, *IEEE Trans. Softw. Eng.* **21**(3) (1995) 263–265.
30. T. Mayer and T. Hall, A critical analysis of current OO design metrics, *Softw. Qual. J.* **8**(2) (1999) 97–110.
31. M. Hitz and B. Montazeri, Chidamber and Kemerer's metrics suite: A measurement theory perspective, *IEEE Trans. Softw. Eng.* **22**(4) (1996) 267–271.
32. V. R. Basili, L. C. Briand and W. L. Melo, A validation of object-oriented design metrics as quality indicators, *IEEE Trans. Softw. Eng.* **22**(10) (1996) 751–761.
33. D. Azar and J. Vybihal, An ant colony optimization algorithm to improve software quality prediction models: Case of class stability, *J. Inf. Softw. Technol.* **53**(4) (2011) 388–393.
34. A. Saed and W. Kadir, Applying particle swarm optimization to software performance prediction: An introduction to the approach, in *5th Malaysian Conf. Software Engineering*, 2011, pp. 207–212.
35. C. J. Burgess and M. Leey, Can genetic programming improve software effort estimation? A comparative evaluation, *Inf. Softw. Technol.* **43**(14) (2001) 863–873.
36. SourceForge, CCCC-C and C++ code counter (2013), http://cccc.sourceforge.net/.
37. SourceForge, Dependency finder (2013), http://depfind.sourceforge.net/.
38. Eclipse Foundation, Eclipse software (2013), https://www.eclipse.org/.
39. ARiSA, Analyzer: VizzAnalyzer (2013), http://www.arisa.se/tools.php?lang=en.
40. Scientific Toolworks, Inc., Understand tool (2013), http://www.scitools.com/.
41. CodeSWAT, Analyst4j, http://www. codeswat.com.
42. SMRG, OOMeter (2013), http://www.ccse.kfupm.edu.sa/∼oometer/oometer.
43. D. Spinellis, CKJM 1.8.: The official documentation of CKJM (2007), http://www.spinellis.gr/sw/ckjm/.
44. PROMISE FORUM, Data repository (2013), http://www.Promisedata.org.
45. S. D. Conte, H. E. Dunsmore and Y. V. Shen, *Software Engineering Metrics and Models* (Benjamin Cummings, 1986).
46. B. Kitchenham, L. M. Pickard, S. G. MacDonell and M. J. Shepperd, What accuracy statistics really measure, *IEE Proc. Softw.* **148**(3) (2001) 81–85.
47. B. Kitchenham, P. Fleeger, B. McColl and S. Eagan, An empirical study of maintenance and development estimation accuracy, *J. Syst. Softw.* **64**(1) (2002) 57–77.
48. N. Fenton and M. Neil, A critique of software defect prediction model, *IEEE Trans. Softw. Eng.* **25**(3) (1999) 1–5.

49. W. Opdyke, Refactoring: A program restructuring aid in designing object-oriented application frameworks, Ph.D. thesis, University of Illinois at Urbana-Champaign, 1992.
50. T. Mens and T. Tourwe, A survey of software refactoring, *IEEE Trans. Softw. Eng.* **30**(2) (2004) 126–139.
51. M. Fowler, *Refactoring: Improving the Design of Existing Code* (Addison-Wesley, 1999).
52. D. Wilkins, U. F. Khan and S. Kowalewski, An empirical evaluation of refactoring, *e-Informatica Softw. Eng. J.* **1**(1) (2007) 27–42.
53. M. O. Keeffe and M. O. Cinneide, Search-based refactoring for software maintenance, *J. Syst. Softw.* **81** (2008) 502–516.
54. A. Ouni, M. Kessentini and H. Sahraoui, Search based refactoring using recorded code changes, in *7th European Conf. Software Maintenance and Reengineering*, 2013, pp. 221–230.
55. M. Zhang, T. Hall and N. Baddoo, Code bad smells: A review of current knowledge, *J. Softw. Maint. Evol. Res. Pract.* **23**(3) (2011) 179–202.
56. B. Geppert, A. Mockus and F. Robler, Refactoring for changeability: A way to go?, in *Proc. 11th IEEE Int. Software Metrics Symp.*, 2005, pp. 10–13.
57. G. M. Berns, Assessing software maintainability, *ACM Commun.* **27**(1) (1984) 14–23.
58. J. M. Bieman and L. M. Ott, Measuring functional cohesion, *IEEE Trans. Softw. Eng.* **20**(8) (1994) 644–657.
59. S. M. Yacoub, H. H. Ammar and T. Robinson, Dynamic metrics for object oriented design, in *Proc. IEEE 6th Int. Symp. Software Metrics*, 1999, pp. 50–61.
60. S. Babu and R. M. S. Parvathi, Design dynamic coupling measurement of distributed object-oriented software using trace events, *J. Comput. Sci.* **7**(5) (2011) 770–778.
61. D. Knippers, Agile software development and maintainability, in *Proc. 15th Twente Student Conf. IT*, 2011.
62. C. Poole and J. W. Huisman, Using extreme programming in a maintenance environment, *IEEE Soft.* **18**(6) (2001) 42–50.
63. K. K. Aggarwal, Y. Singh, P. Chandra and M. Puri, Measurement of software maintainability using a fuzzy model, *J. Comput. Sci.* **1**(4) (2005) 538–542.
64. K. K. Aggarwal, Y. Singh and J. Chhabra, An integrated measure of software maintainability, in *Proc. Ann. Reliability and Maintainability Symp.*, 2002, pp. 235–241.
65. K. K. Aggarwal, Y. Singh, A. Kaur and R. Malhotra, Application of artificial neural network for predicting maintainability using object-oriented metrics, in *Proc. World Academy of Science, Engineering and Technology*, 2006, pp. 285–289.
66. E. Arisholm and D. Sjoberg, Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software, *IEEE Trans. Softw. Eng.* **30**(8) (2004) 521–534.
67. A. D. Baker, A. B. Sultan, H. Zulzalil and J. Din, Applying evolution programming search based software engineering (SBSE) in selecting the best open source maintainability metrics, in *International Symp. Computer Applications and Industrial Electronics*, 2012.
68. G. Balogh, A. Zoltan and A. Baszedes, Prediction of software development effort enhanced by a genetic algorithm, in *Symp. Search Based Software Engineering*, 2012.
69. R. K. Bandi, V. K. Vaishnavi and D. E. Turk, Predicting maintenance performance using object-oriented design complexity metrics, *IEEE Trans. Softw. Eng.* **29**(1) (2003) 77–87.
70. R. D. Banker, M. D. Srikant, C. F. Kemerer and D. Zweig, Software complexity and maintenance cost, *Commun. ACM* **36**(11) (1993) 81–94.
71. A. B. Baqais, M. Alshayeb and Z. A. Baig, Hybrid intelligent model for software maintenance prediction, in *Proc. World Congress on Engineering*, Vol. I, 2013.

72. M. P. Basgalupp, R. C. Barros and D. D. Ruiz, Predicting software maintenance effort through evolutionary-based decision trees, in *Proc. 27th Ann. ACM Symp. Applied Computing*, 2012, pp. 1209–1214.
73. V. R. Basili, L. Briand, S. Condon, W. Melo and J. Valett, Understanding and predicting the process of software maintenance releases, in *18th Int. Conf. Software Engineering*, 1996.
74. P. Bhattacharya and I. Neamtiu, Assessing programming language impact on development and maintenance: A study on C and C++, in *Proc. 33rd ICSE Conf.*, 2011.
75. M. Broy, F. Deißenböck and M. Pizka, Demystifying maintainability, in *Proc. 4th Workshop Software Quality*, 2006.
76. J. C. Chen and S. J. Huang, An empirical analysis of the impact of software development problem factor on software maintainability, *J. Syst. Softw.* **82**(6) (2009) 981–992.
77. D. Coleman, D. Ash, B. Lowther and P. Oman, Using metrics to evaluate software system maintainability, *Computer* **27**(8) (1994) 44–49.
78. M. Dagpinar and J. H. Jahnke, Predicting maintainability with object-oriented metric — An empirical comparison, in *Proc. 10th Working Conf. Reverse Engineering*, 2003, pp. 155–164.
79. S. S. Dahiya, J. C. Chhabra and S. Kumar, Use of genetic algorithm for software metrics conditioning, in *15th Int. Conf. Advance Computing and Communication*, 2007, pp. 87–92.
80. J. Daly, A. Brooks, J. Miller, M. Roper and M. Wood, Evaluating inheritance depth on the maintainability of object-oriented systems, *Empir. Softw. Eng.* **1**(2) (1996) 109–132.
81. F. Deißenböck, S. Wagner, M. Pizka, S. Teuchert and J.-F. Girard. An activity-based quality model for maintainability, in *Proc. 23rd Int. Conf. Software Maintenance*, 2007.
82. I. Deligiannis, I. Stamelos, L. Angelis, M. Roumelitis and M. Shepperd, A controlled experiment investigation of an object-oriented design heuristic for maintainability, *J. Syst. Softw.* **72**(1) (2004) 129–143.
83. M. O. Elish and K. O. Elish, Application of tree net in predicting object-oriented software maintainability: A comparative study, in *European Conf. Software Maintenance and Reengineering*, 2009, pp. 69–78.
84. E. H. Ferneley, Design metrics as an aid to software maintenance: An empirical study, *J. Softw. Maint. Evol.* **11** (1999) 55–72.
85. F. Fioravanti and P. Nesi, Estimation and prediction metrics for adaptive maintenance effort of object-oriented system, *IEEE Trans. Softw. Eng.* **27**(12) (2001) 1062–1084.
86. I. P. Floris, B. Engel and V. H. Harald, How to save on software maintenance costs, Omnext White Paper, Omnext BV, The Netherlands (2010).
87. M. Genero, M. Piattini, E. Manso and G. Cantone, Building UML class diagram maintainability prediction models based on early metrics, in *Ninth Int. Software Metrics Symp.* 2003, pp. 263–275.
88. J. C. Granja and M. J. B. García, A method for estimating maintenance cost in a software project: A case study, *J. Softw. Maint. Evol.* **9**(3) (1997) 161–175.
89. S. Hanenberg, S. Kleinschmager, R. Robbes, E. Tanter and A. Stefik, An empirical study on the impact of static typing on software maintainability, *Empir. Softw. Eng.* **19**(5) (2014) 1335–1382.
90. R. Harrison, S. Counsell and R. Nithi, Experimental assessment of the effects of inheritance on the maintainability of object-oriented systems, *J. Syst. Softw.* **52** (2000) 173–179.
91. L. Hatton, How accurately do engineers predict software maintenance tasks? *Computer* **40**(2) (2007) 64–69.

92. J. H. Hays and L. Zhao, Maintainability prediction: A regression analysis of measure of evolving systems, in *Proc. 21st IEEE Int. Conf. Software Maintenance*, 2005, pp. 601–604.
93. P. Hegedus, Revealing the effect of coding practices on software maintainability, in *29th IEEE Int. Conf. Software Maintenance*, 2013, pp. 578–581.
94. T. Hirota, M. Tohki, C. M. Overstreet, M. Masaaki and R. Cherinka, An approach to predict software maintenance cost based on ripple complexity, in *Proc. Conf. APSEC*, 1994, pp. 439–444.
95. K. Jeet, R. Dhir and H. Verma, A comparative study of Bayesian and fuzzy approach to assess and predict maintainability of the software using activity-based quality model, *ACM SIGSOFT Softw. Eng. Notes* **37**(3) (2012) 1–9.
96. C. Jin and J. A. Liu, Applications of support vector machine and unsupervised learning for predicting maintainability using object-oriented metrics, in *Second Int. Conf. Multimedia and Information Technology*, 2010, pp. 24–27.
97. M. Jorgensen, Experience with the accuracy of software maintenance task effort prediction models, *IEEE Trans. Softw. Eng.* **21**(8) (1995) 674–681.
98. H. Kabaili, R. K. Keller and F. Lustman, Cohesion as changeability indicator in object-oriented systems, in *Fifth European Conf. Software Maintenance and Reengineering*, 2001, pp. 39–46.
99. Y. Kataoka, T. Imai, H. Andou and T. Fukaya, A quantitative evaluation of maintainability enhancement by refactoring, in *Proc. Int. Conf. Software Maintenance*, 2002, pp. 576–585.
100. A. Kaur, K. Kaur and R. Malhotra, Soft computing approaches for prediction of software maintenance effort, *Int. J. Comput. Appl.* **1**(16) (2010) 69–75.
101. A. Kaur and K. Kaur, Statistical comparison of modelling methods for software maintainability prediction, *Int. J. Softw. Eng. Knowl. Eng.* **23**(06) (2013) 743–774.
102. A. Kaur, K. Kaur and K. Pathak, Software maintainability prediction by data mining of software code metrics, in *Int. Conf. Data Mining and Intelligent Computing*, 2014, pp. 1–6.
103. R. Kumar and N. Dhanda, Maintainability measurement model for object oriented design, *Int. J. Adv. Res. Comput. Commun. Eng.* **4**(5) (2015) 68–71.
104. L. Kumar, Predicting object-oriented software maintainability using hybrid neural network with parallel computing concept, in *Proc. 8th India Software Engineering Conf.*, 2015, pp. 100–109.
105. C. F. Kemerer and S. A. Slaughter, Determinants of software maintenance profiles: An empirical investigation, *J. Softw. Maint.* **9**(1) (1997) 235–251.
106. C. V. Koten and A. R. Gray, An application of Bayesian network for predicting object-oriented software maintainability, *Inf. Softw. Technol.* **48**(1) (2006) 59–67.
107. W. Li and S. Henry, Object-oriented metrics that predict maintainability, *J. Syst. Softw.* **23**(2) (1993) 111–122.
108. J. S. Lim, S. R. Jeong and S. R. Schach, An empirical investigation of the impact of the object-oriented paradigm on the maintainability of real-world mission-critical software, *J. Syst. Softw.* **77**(1) (2005) 131–138.
109. J. C. Lin and K. C. Wu, A model for measuring software understandability, in *Proc. Sixth Int. Conf. Computer and Information Technology*, 2006, pp. 192–198.
110. A. D. Lucia, E. Pompella and S. Stefanucci, Assessing effort estimation models for corrective maintenance through empirical studies, *J. Inf. Softw. Technol.* **47**(1) (2005) 3–15.

111. R. Malhotra and A. Chug, Application of evolutionary algorithms for software maintainability prediction using object-oriented metric, in *8th Int. Conf. Bio-Inspired Information and Communications Technologies*, 2014.
112. R. Malhotra and A. Chug, Software maintainability prediction using machine learning algorithms, *Softw. Eng.* **2**(2) (2012) 19–36.
113. R. Malhotra and A. Chug, Metric suite for predicting software maintainability in data intensive applications, in *Transactions on Engineering Technologies* (Springer, 2014), pp. 161–175.
114. R. Malhotra and A. Chug, Application of group method of data handling model for software maintainability prediction using object oriented systems, *Int. J. Syst. Assur. Eng. Manage.* **5**(2) (2014) 165–173.
115. R. Malhotra, A. Chug and P. Khosla, Prioritization of classes for refactoring: A step towards improvement in software quality, in *Third Int. Symp. Women in Computing and Informatics*, 2015.
116. S. C. Misra, Modeling design/coding factors that drive maintainability of software systems, *Soft. Qual. J.* **13**(3) (2005) 297–320.
117. S. Mishra and A. Sharma, Maintainability prediction of object-oriented software by using adaptive network-based fuzzy system technique, *Int. J. Comput. Appl.* **119**(9) (2015) 24–27.
118. S. Mutanna, K. Kontogiannis, K. Ponnambalam and B. Stacey, A maintainability model for industrial software system using design level metrics, in *Seventh Working Conf. Reverse Engineering*, 2000, pp. 248–256.
119. F. Niessink and H. V. Vliet, Predicting maintenance effort with function points, in *Proc. Int. Conf. Software Maintenance*, 1997, pp. 32–39.
120. P. Oman and J. Hagemeister, Construction and testing of polynomials predicting software maintainability, *J. Syst. Softw.* **24** (1994) 251–266.
121. P. Oman and J. Hagemeister, Metrics for assessing a software system's maintainability, in *IEEE Conf. Software Maintenance*, 1992, pp. 337–344.
122. L. Ping, A quantitative approach to software maintainability prediction, *Int. Forum Information Technology and Applications* **1**(1) (2010) 105–108.
123. M. Pizka and F. Deissenboeck, How to effectively define and measure maintainability, in *Software Measurement European Forum*, 2007.
124. M. Polo, M. Paittini and F. Ruiz, Using code metrics to predict maintenance of legacy programs: A case study, in *Proc. Int. Conf. Software Maintenance*, 2001, pp. 202–208.
125. N. N. Prasanth, S. Ganesh and G. A. Dalton, Prediction of maintainability using software complexity analysis: An extended FRT, in *Proc. 2008 Int. Conf. Computing, Communication and Networking*, 2008, pp. 1–9.
126. N. N. Prasanth, S. P. Raja, X. Birla, K. Navaz, S. Arif and A. Rahuman, Improving software maintainability through risk analysis, *Int. J. Recent Trends Eng.* **2**(4) (2009) 198–200.
127. L. Prechelt, U. Barbara, M. Philippsen and W. Tichy, A controlled experiment on inheritance depth as a cost factor for code maintenance, *J. Syst. Softw.* **65**(2) (2003) 115–126.
128. C. Rajaraman and M. R. Lyu, Reliability and maintainability related software coupling metrics in C++ programs, in *Third Int. Symp. Software Reliability Engineering*, 1992, pp. 303–311.
129. A. Rana, S. K. Dubey and Y. Dash, Maintainability prediction of object-oriented software systems by multilayer perceptron model, *ACM SIGSOFT Softw. Eng. Notes* **37**(5) (2012) 1–4.

130. J. F. Ramil and M. M. Lehman, Metrics of software evolution as effort predictors — A case study, in *Proc. ICSM 2000*, 2000, pp. 163–172.
131. J. F. Ramil and N. Smith, Qualitative simulation of models of software evolution, *Softw. Process, Improv. Pract.* **7**(3–4) (2002) 95–112.
132. M. Riaz, E. Mendes, E. Tempero and M. Sulayman, Using CBR and CART to predict maintainability of relational database-driven software applications, in *Proc. 17th Int. Conf. Evaluation and Assessment in Software Engineering*, 2013, pp. 132–143.
133. M. Riaz, E. Tempero, M. Sulayman and E. Mendes, Maintainability predictors for relational database-driven software applications: Extended results from a survey, *Int. J. Softw. Eng. Knowl. Eng.* **23**(04) (2013) 507–522.
134. S. L. Schneberger, Distributed computing environments: Effects on software maintenance difficulty, *J. Syst. Softw.* **37** (1997) 101–116.
135. N. F. Schneidewind, Software quality control and prediction model for maintenance, *Ann. Softw. Eng.* **9** (2000) 79–101.
136. T. Sheldon, K. Jerath and H. Chung, Metric for maintainability of class inheritance hierarchies, *J. Softw. Maint. Evol. Res. Pract.* **14** (2002) 1–14.
137. K. Shibata, K. Rinsaka, T. Dohi and H. Okamura, Quantifying software maintainability based on a fault-detection/correction model, in *Proc. Symp. 13th Pacific Rim Int. Dependable Computing*, 2007, pp. 35–42.
138. G. E. Stark, L. C. Kern and C. W. Vowell, Software metric set for program maintenance, *J. Syst. Softw.* **24**(3) (1994) 239–249.
139. H. Sneed, A cost model for software maintenance and evolution, in *20th IEEE Int. Conf. Software Maintenance*, 2004.
140. H. Sneed, Estimating the costs of software maintenance tasks, in *Proc. Int. Conf. Software Maintenance*, 1995, p. 168.
141. N. Soni and M. Khaliq, Maintainability estimation of object-oriented software: Design phase perspective, *Int. J. Adv. Res. Comput. Commun. Eng.* **4**(3) (2015) 538–542.
142. D. Stavironoudis, M. Xenos and D. Christodolakis, Relation between software metrics and maintainability, in *Proc. Federation of European Software Measurement Association*, 1999, pp. 465–476.
143. P. Sun and A. Wang, Application of ant colony optimization in preventive software maintenance policy, in *IEEE Int. Conf. Information Science and Technology*, 2012.
144. M. Thongmak and P. Muenchaisri, Maintainability metrics for aspect-oriented software, *Int. J. Softw. Eng. Knowl. Eng.* **19**(03) (2009) 389–420.
145. M. M. T. Thwin and T. S. Quah, Application of neural networks for software quality prediction using object-oriented metrics, *J. Syst. Softw.* **76**(2) (2005) 147–156.
146. N. Uoadhyay, B. M. Deshpande and V. P. Agarwal, Developing maintainability index of a software component: A digraph and matrix approach, *ACM SIGSOFT Softw. Eng. Notes* **35**(5) (2010) 1–11.
147. S. Velmourougan, P. Dhavachelvan, R. Baskaran and B. Ravikumar, Software development life cycle model to improve maintainability of software applications, in *Fourth Int. Conf. Advanced Computing and Communications*, 2014, pp. 270–273.
148. R. Vivanco and N. Pizzi, Finding effective software metrics to classify maintainability using a parallel genetic algorithm, in *Genetic and Evolutionary Computing — GECCO 2004*, 2004, pp. 1388–1399.
149. K. D. Welker, P. Oman and G. G. Atkinson, Development and application of an automated source code maintainability index, *J. Softw. Maint. Res. Pract.* **9**(3) (1997) 127–159.

150. L. J. Wang *et al.*, Predicting object-oriented software maintainability using projection pursuit regression, in *1st Int. Conf. Information Science and Engineering*, 2009, pp. 3827–3835.
151. Z. Xing and E. Stroulia, Refactoring practice How it is and how it should be supported — An Eclipse case study, in *Proc. Int. Conf. Software Maintenance*, 2006, pp. 458–468.
152. A. Yamashita and L. Moonen, Do code smells reflect important maintainability aspects?, in *Proc. Int. Conf. Software Maintenance*, 2012.
153. F. Ye, X. Zhu and Y. Wang, A new software maintainability evaluation model based on multiple classifier combination, in *Int. Conf. Quality, Reliability, Maintenance and Safety Engineering*, 2013, pp. 1588–1591.
154. A. T. T. Ying, G. C. Murphy, R. Ng and M. C. Chu-Carroll, Predicting source code changes by mining change history, *IEEE Trans. Softw. Eng.* **30**(9) (2004) 574–586.
155. W. Zhang, L. G. Huang, V. Ng and J. Ge, SMPLearner: Learning to predict software maintainability, *Autom. Softw. Eng.* **22**(1) (2015) 111–141.
156. Y. Zhou and H. Leung, Predicting object-oriented software maintainability using multivariate adaptive regression splines, *J. Syst. Softw.* **80**(8) (2007) 1349–1367.
157. Y. Zhou and B. Xu, Predicting the maintainability of open source software using design metrics, *Wuhan Univ. J. Nat. Sci.* **13**(1) (2008) 14–21.
158. K. Kaur and H. Singh, Determination of maintainability index for object-oriented systems, *ACM SIGSOFT Softw. Eng. Notes* **36**(2) (2011) 1–6.
159. D. Kafura and G. R. Reddy, The use of software complexity metrics in software maintenance, *IEEE Trans. Softw. Eng.* **13**(3) (1987) 335–343.
160. Y. Singh and B. Goel, A step towards software preventive maintenance, *ACM SIGSOFT Softw. Eng. Notes* **32**(4) (2007) 1–5.
161. J. C. Miller, *Techniques of Program and System Maintenance*, ed. G. Parikh (Winthrop, 1981), pp. 181–182.
162. N. Chapin, Do we know what preventive maintenance is?, in *Proc. Int. Conf. Software Maintenance*, 2000, pp. 15–17.
163. K. H. Bennett and V. T. Rajlich, Software maintenance and evolution: A roadmap, in *Proc. Conf. Future of Software Engineering*, 2000, pp. 73–87.
164. M. Kajko-Mattsson, Can we learn anything from hardware preventive maintenance?, in *Proc. Seventh IEEE Int. Conf. Engineering of Complex Computer Systems*, 2001, pp. 106–111.
165. P. Bhatt, G. Shroff and A. K. Misra, Dynamics of software maintenance, *ACM SIGSOFT Softw. Eng. Notes* **29**(5) (2004) 1–5.
166. J. Bosch and P. O. Bengtsson, Assessing optimal software architecture maintainability, in *Fifth European Conf. Software Maintenance and Reengineering*, 2001.
167. W. Abdelmoez, K. Goseva-Popstojanova and H. H. Ammar, Maintainability based risk assessment in adaptive maintenance context, in *Int. Conf. Software Engineering*, 2006.
168. Saraiva, Juliana, E. Barreiros, A. Almeida, F. Lima, A. Alencar, G. Lima, S. Soares and F. Castor, Aspect-oriented software maintenance metrics: A systematic mapping study, in *16th Int. Conf. Evaluation & Assessment in Software Engineering*, 2012, pp. 253–262.