Application of Machine Learning in Software Quality: a Mini-review

Ozias Bombiri
Laboratoire LAMDI
Université Nazi BONI
Bobo Dioulasso, Burkina Faso
ozibombe@hotmail.com

Pasteur PODA
Laboratoire LAMDI
Université Nazi BONI
Bobo Dioulasso, Burkina Faso
pasteur.poda@u-naziboni.bf

T. Frédéric OUEDRAOGO
UFR Sciences et Technologies
Université Norbert ZONGO
Koudougou, Burkina Faso
frederic.ouedraogo@unz.bf

2023 IEEE Multi-conference on Natural and Engineering Sciences for Sahel's Sustainable Development (MNE3SD) | 979-8-3503-1180-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/MNE3SD57078.2023.10079800

***Abstract*—The digital transformation underway in Sahelian countries is accompanied by a growing interest in software engineering products in both the formal and informal sectors. The developed software mostly meet the needs of the customers, but it is not excluded that they suffer from compliance to the software quality standard. Thus, having intelligent systems that can assess whether important quality attributes are verified may be helpful. In this paper, we propose a mini-survey of different research works focusing on the improvement of software quality based on Machine Learning (ML) methods. The lessons learnt show that software quality management through ML methods is an active research field. Software defect prediction and maintainability prediction are problems that had been treated by ML methods. Datasets, software metrics and ability of ML algorithms to build such predictive models have been the main target of the studies.**

Keywords–Machine Learning, software quality, defect prediction, maintainability prediction

I. Introduction

There is no doubt that Information and Communication Technologies are spreading at an unprecedented rate, but their patterns of diffusion are quite difficult to discern and are constantly changing. This evolution may well increase the digital divide between developed and developing countries. Those of the actors of the formal or informal sectors who have understood this issue have embarked on a digital transformation, which is very much characterized by a strong interest in software products. Clients in the informal sector in particular mostly purchase software products from inexperienced developers, so there is no warranty that quality is always up to par. This low level of quality is explained by the lack of respect for the principles and rules defined in software engineering (SE). SE quality standards and frameworks, require the measurement of software quality. Good software is whose that works, but it must also meet client's expectations, be easy to use, perform well, and be able to evolve as needed. It must also be maintainable. The prerequisite for software quality assurance is to respect the rules of the art of development: uniform code respecting coding standards, decentralized management of source code, and automation of the entire software construction chain. However, it is questionable if all these best practices can be easily applied, and if the way to evaluate the quality of a software product is straightforward. A major milestone will be to upskill developers by providing them with automatic software quality management tools. In this perspective, solutions based on Machine Learning (ML) could be a last resort. Indeed, it has been proven in several domains that ML methods can overcome some of the limitations of traditional methods. It is therefore relevant to turn to these methods in the search for solutions to improve software quality.

In this paper we propose a literature review of the different studies related to software quality in SE. Several literature review studies on software quality have been done [1–8] with few addressing the use of ML [2], [7], [8]. However these surveys do not cover the whole software quality aspects. Our objective through this review is, for the community of researchers and engineers in the Sahelian space,

to contribute to the knowledge of ML-based tools for software quality assessment.

The remainder of this paper is organized as follows: Section II presents the methodology that is applied for data collection. Section III presents answers to the research questions.

II. Materials and methods

The purpose of this survey is to present a repository for the use of ML for software quality improvement. We adopted the systematic literature review method described in [9]. However, as we are doing a non-exhaustive literature review, we simplified the original method. The selection of resources was based on scientific publications in relevance to the predefined research questions. These questions are given in Table I.

TABLE I
Research questions

Questions	**Motivations**
What are the software quality features addressed in the research?	Identify the software quality problems for which there are treatment proposals based on ML
What data are used for model building?	Identify the different data sources that are used for model building and describe these data
What are the algorithms used in the model proposals?	Identify the learning algorithms that have been proposed
What are the performances achieved?	Review the applicability of ML in SE, particularly with respect to software quality

The selection process adopted consists of four steps, namely the choice of scientific publication sources, the identification of additional research sources, the definition of a publication period and the definition of research terms. For the selection of sources, five most popular and important online digital libraries in the field of computer science that publish peer-reviewed articles were selected. These are:

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

In addition, manual searches were applied to include relevant journal and conference proceedings in the field of software quality. We selected these journals and conferences especially since they involve empirical studies or literature reviews and are well-established and highly relevant software engineering publications.

Regarding the publication period, the search was limited to articles published between 2010 and 2020.

III. Results and discussion

The bibliographic exploration that we have done has allowed us to understand that researchers have turned to ML to find solutions to software engineering problems. To respond our first research question, software reliability through defect prediction, effort estimation (development time and cost), and maintainability are aspects that have been addressed. In this study, we are interested in software quality, and therefore in the themes directly related to it: defect and maintainability prediction. Answers to the remaining research questions are presented by theme.

A. Software defect prediction

Reliability is one of the most important attributes of software quality. It is the ability of the software to provide correct results under any operating conditions, assuming no defects. A software defect is a component or system fault that can prevent the component or system from performing its functions. To detect defects and improve quality, engineers use code inspection or unit testing. These operations usually cost about 80 percent of the project budget [10]. Moreover, these tests do not guarantee the total reliability of the system. For large systems, it is almost impossible to achieve complete test coverage. Then, it would be very useful to have an approach that allow prioritizing the tests in order to target the most critical modules, both for reasons of efficiency [11] (concerning the deployment of test resources) and reliability. That is the objective of defect prediction. Based on the ML, there has been an increasing amount of research into finding models that can predict potential defects.

1) Software metrics and data sets for defect prediction: Software defect prediction has become one of the main research areas for software quality improvement since the 1990s [12]. First, there were studies that designed software metrics and the publication of datasets in the field. Substantial research has been done to define criteria to characterize source code. These metrics have been used largely to propose defect prediction models according to our references. Studies have also concluded about the effectiveness of some of these metrics [13]. Some researchers argue that software metrics cannot be used to characterize faulty source code because they do not capture the syntactic and semantic structure of the code. They propose the use of techniques such as CNN (convolutional neural networks) [14] or LSTM (long short-term memory) [11] for the automatic extraction of features from the abstract syntax tree or from the tree or the source code.

In addition to metrics, works have also made available to researchers datasets on software defects. The NASA repository: PROMISE (PRedictOr Models In Software Engineering) was initiated by Sayyad Shirabad and Tim Menzies, in December 2004 [15]. This repository was created to encourage the production of reproducible, verifiable, refutable and/or improvable predictive models in SE. It makes available to the public several datasets for building predictive models on software. The work of Jureczko and Madeyski [16] and Jureczko and Spinellis [17] contributed to the enrichment of this repository with data from open source software belonging to The Apache Software Foundation. The data from this repository were used to build several fault prediction models [13, 18–21]. Other case studies have also used proprietary data or data not found in the PROMISE repository, such as [21].

2) Algorithms used for the construction of predictors of defects: A wide variety of ML classification algorithms have been used to propose software defect prediction models (predicting whether the module is faulty or not). However, Lei Qiao *et al.* [10] found that it would be more interesting for an engineer to know the number of defects in a module, because it may help prioritize modules in code inspection or testing and thus allows for a limited resource allocation reducing manpower and time. They proposed a neural network-based model that can predict the number of defects in software modules. This survey allowed us to identify a number of ML algorithms on the basis of which approaches for predicting software defects have been proposed. Among these algorithms, some have proven to be powerful while others have shown their limitations. Studies [18, 22] have proven that SVM (Support Vector Machine), LVQ (Learning Vector Quantization) and MultiLVQ are less suitable for software fault prediction as they give poor results. Table II gives an overview of the different algorithms used.

TABLE II
ALGORITHMS USED IN SOFTWARE DEFECT PREDICTION

Algorithms	References
Logistic regression	[14]
Naïve bayes	[12, 13, 23]
Support vector machine	[13, 18]
Decision tree	[12, 13, 23]
Random forest	[12, 13]
Multilayer percetron	[12, 13, 22–24]
Convolutional neural networks	[14, 19, 22, 24]
Long short-term memory	[11, 20, 22]
LVQ et MultiLVQ	[22]
Deep Belief Network	[14, 18]
Particle swarm optimization	[23]
Self organisation mapping	[22]
Artificial immune recognition system	[12]

3) Defect predictors performance achievements: It is less obvious to make an objective statement about the performance of software defect predictors. Indeed, there is a great diversity of studies using different performance evaluation metrics and data sets for models fitting. Some studies simply conclude whether a particular technique can predict defects. All these factors make it difficult to verify and compare results. Nevertheless, all studies are unanimous on the issue of software defect predictability using ML techniques. The best results show an AUC value around 0.8 with RF [13] and CNN [22] techniques.

In [25], M. Shepperd *et al.* conducted a study to find out what factors influence performance in

the design of fault prediction models. By reverse engineering a common response variable, they constructed a random-effects ANOVA (Analysis of variance) model to evaluate the relative contribution of four factors in model construction, namely the algorithm used, the dataset used to train the model, the software metrics, and the group of researchers who conducted the study. Their finding is that the choice of the algorithm has little impact on the model performance (1.3%) and that instead the main explanatory factor is the group of researchers (31%). To address this high level of researcher bias in the models, they proposed three measures for researchers to consider: conducting a blind analysis, improving reporting protocols, and conducting more cross-group studies to mitigate expertise issues.

B. Software maintainability prediction

The software quality assurance process goes beyond the testing phase. Indeed, maintenance is the longest phase of the software process. After delivery, many tasks are performed to ensure the improvement, non-regression and evolution of the product in order to keep it running in a sustainable manner. The main challenge of maintenance is to implement changes to the software while avoiding degradation of its quality. Maintainability is one of the fundamental attributes of software quality and is the ease with which a software system or component can be modified to correct faults, improve performance or other attributes or adapt to a changed environment. Its accurate prediction is a challenge for an effective management of the maintenance process and is recognized as a research area of primary importance in SE [26].

Maintainability can be measured indirectly by certain metrics or directly by observation during maintenance by factors such as time, effort, number of modules examined. Direct measurements cannot be used, hence the challenge of indirect measurement methods through the design of predictors. The obvious difficulty of these methods is the identification of appropriate metrics, from the software, for the maintainability calculation functions.

Software maintainability prediction models have been studied to help organizations reduce costs, allocate resources, and achieve an accurate management plan and efficient maintenance process [27]. For these models, the goal is to predict the CHANGE[1] maintenance effort or the maintainability index (MI)[2].

1) Datasets and software metrics used: Many of the studies on exploring the capabilities of ML techniques in predicting maintainability use the two popular object-oriented datasets published by Li and Henry [28]: the UIMS and QUES datasets. The UIMS dataset contains class-level metrics data collected from a user interface management system, while the QUES dataset contains the same metrics collected from a quality assessment system. These data were collected by observing the changes made to these software packages over a three-year period.

2) Algorithms used and performance achieved: The maintenance effort CHANGE is a numerical value that indicates the maintainability of the software. The higher it is, the more difficult the system is to maintain. In the literature, ML regression methods have been used to build maintainability predictors.

Ruchika Malhotra and Anuradha Chug have done a quantitative study on the topic of maintainability prediction by proposing several ML techniques in [29]. The proposed models are the Group Method of Data Handling (GMDH), Genetic Algorithms (GA) and Probabilistic Neural Network (PNN) with a Gaussian activation function. The performances of these techniques were evaluated using QUES and UIMS data. Based on experiments conducted, it was found that GMDH can be applied as a sound alternative to existing techniques for predicting maintainability, as it predicts maintainability more accurately than the dominant models.

Shikha Gupta and Anuradha Chug presented in [30] an improved RF approach for software maintainability prediction. In this study, the approach

[1]The CHANGE maintenance effort, which is a dependent metric, is calculated based on the number of lines of code changed in a class. In this metric an insertion or deletion is counted as 1 while a change in a line is counted as 2

[2]MI is a composite metric that incorporates a number of traditional source code metrics into a single number that indicates relative maintainability

combines the RF algorithm with three common feature selection techniques, namely a chi-square filter, an RF filter, and a linear correlation filter, along with a sampling technique to improve the prediction accuracy of the basic RF algorithm. The improved RF is applied on the QUES and UIMS datasets. The results indicate that the enhanced RF approach performs significantly better than RF for both datasets with improved $R2$[3] values equal to 69.50%, 65.57% and 69.40% for QUES and 31.90%, 44.94% and 51.81% for UIMS using chi-square, RF filtering and linear correlation techniques respectively.

Several other studies have been conducted to explore the capabilities of ML methods in predicting software maintainability [31–33].

C. Discussion

Many studies have addressed the improvement of software products quality through ML tools. As the main concern in ML is data, studies have focused on the data to use, metrics for describing these data. Repositories provided can no longer permit to design real usable models regarding the diversity of source code languages and specificity of each software. Their utility lies in the fact that they are used in research to assess ability of algorithms and for research findings comparison. Therefore apply model in a real context requires collecting data and careful study.

The quality attributes most concerned are reliability and maintainability. These are the ones which strongly tied to the source code. The others like functional capacity, efficiency and portability are rather related to the requirements, design and implementation options. Their measurement can be assessed before or during the deployment. But for reliability, code source inspecting and exhaustive test are required. The maintainability measure requires doing maintenance phases.

In Sahelian context, the availability of ML tools for software quality improvement must be received as good news. Firstly, the number of products with insufficient quality will be reduce by the assessment before deployment. Secondly, the idea behind predictive models is to help management process by allowing more efficient resources assignment (time and effort). The knowledge of the quality level makes it possible to plan this. Finally, financial gains can be made by using ML tools, as they improve management of product life cycle.

Further studies can be performed for assessment of models efficiency on software products in use and to construct datasets for more ML models in Sahelian software products context. Integration of the research findings to integrated development environments may also help developers in managing their products quality.

[3]the proportion of the variance in the dependent variable that is explained by the model

IV. Conclusion

As the need for software systems grows, the complexity and size evolve so that it is difficult to ensure their quality through testing and code inspection processes. Also the implementation of changes during maintenance must be done with great care to avoid altering the system. In the context of a great interest in software products in Sahelian countries, quality improvement solutions should turn to Machine Learning (ML). So the survey made herein aimed at measuring the contribution of ML methods to the improvement of software quality. The conclusions tell us that software defect prediction and maintainability are topics that have been widely addressed in software quality improvement research. Regarding software defect prediction, several algorithms have been proposed for building models to classify software modules into defective and non-defective modules. As for the prediction of maintainability, studies have been based on UIMS and QUES data to propose predictors of CHANGE maintenance effort with regression algorithms. More researches must be made for the applying of these findings in software practices and for quality assessment of software products.

References

[1] Jose P. Miguel, David Mauricio, and Glen Rodriguez. “A Review of Software Quality Models for the Evaluation of Software Products”. In: *International Journal of Software Engineering & Applications* 5.6 (Nov. 2014). arXiv:1412.2977 [cs], pp. 31–53. ISSN: 09762221, 09759018.

[2] Hamdi A. Al-Jamimi and Moataz Ahmed. "Machine Learning-Based Software Quality Prediction Models: State of the Art". In: *2013 International Conference on Information Science and Applications (ICISA)*. 2013, pp. 1–4.

[3] Lars M. Karg, Michael Grottke, and Arne Beckhaus. "A systematic literature review of software quality cost research". In: *Journal of Systems and Software* 84.3 (2011), pp. 415–427. ISSN: 0164-1212.

[4] Brijendra Singh and Suresh Prasad Kannojia. "A Review on Software Quality Models". In: *2013 International Conference on Communication Systems and Network Technologies*. 2013, pp. 801–806.

[5] Ayse Tosun, Ayse Basar Bener, and Shirin Akbarinasaji. "A systematic literature review on the applications of Bayesian networks to predict software quality". en. In: *Software Quality Journal* 25.1 (Mar. 2017), pp. 273–305. ISSN: 1573-1367.

[6] Mawal Ali and Mahmoud O. Elish. "A Comparative Literature Survey of Design Patterns Impact on Software Quality". In: *2013 International Conference on Information Science and Applications (ICISA)*. 2013, pp. 1–7.

[7] Romi Satria Wahono. "A systematic literature review of software defect prediction". In: *Journal of Software Engineering* 1.1 (2015), pp. 1–16.

[8] Faseeha Matloob et al. "Software defect prediction using ensemble learning: A systematic literature review". In: *IEEE Access* (2021).

[9] Barbara A. Kitchenham. "Systematic Review in Software Engineering: Where We Are and Where We Should Be Going". In: *Proceedings of the 2nd International Workshop on Evidential Assessment of Software Technologies*. EAST '12. Lund, Sweden: Association for Computing Machinery, 2012, pp. 1–2. ISBN: 9781450315098.

[10] Lei Qiao et al. "Deep learning based software defect prediction". en. In: *Neurocomputing* 385 (Apr. 2020), pp. 100–110. ISSN: 0925-2312.

[11] Hoa Khanh Dam et al. "A deep tree-based model for software defect prediction". In: *arXiv:1802.00921 [cs]* (Feb. 2018). arXiv: 1802.00921.

[12] Ahmed Haouari, Labiba Souici-Meslati, and Fadila Atil. *Prédiction de défauts logiciels par des classifieurs immunologiques*. June 2017.

[13] Diana-Lucia Miholca, Gabriela Czibula, and Istvan Gergely Czibula. "A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks". en. In: *Information Sciences* 441 (May 2018), pp. 152–170. ISSN: 0020-0255.

[14] J. Li et al. "Software Defect Prediction via Convolutional Neural Network". In: *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. July 2017, pp. 318–328.

[15] J.S. Shirabad and T. Menzies. "Predictor models in software engineering (PROMISE)". In: *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* 2005, pp. 692–692.

[16] Marian Jureczko and Lech Madeyski. "Towards identifying software project clusters with regard to defect prediction". In: *Proceedings of the 6th international conference on predictive models in software engineering*. 2010, pp. 1–10.

[17] Marian Jureczko and Diomidis Spinellis. "Using object-oriented design metrics to predict software defects". In: *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wroc\lawskiej* (2010), pp. 69–81.

[18] X. Yang et al. "Deep Learning for Just-in-Time Defect Prediction". In: *2015 IEEE International Conference on Software Quality, Reliability and Security*. Aug. 2015, pp. 17–26.

[19] Thong Hoang et al. "DeepJIT: An End-to-End Deep Learning Framework for Just-in-Time Defect Prediction". In: *2019 IEEE/ACM 16th International Conference on*

Mining Software Repositories (MSR). 2019, pp. 34–45.

[20] Amirabbas Majd et al. “SLDeep: Statement-level software defect prediction using deep-learning model on static code features”. In: *Expert Systems with Applications* 147 (2020), p. 113156. ISSN: 0957-4174.

[21] Ruchika Malhotra. “An empirical framework for defect prediction using machine learning techniques with Android software”. In: *Applied Soft Computing* 49 (2016), pp. 1034–1050. ISSN: 1568-4946.

[22] G. P. Bhandari and R. Gupta. “Measuring the Fault Predictability of Software using Deep Learning Techniques with Software Metrics”. In: *2018 5th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*. Nov. 2018, pp. 1–6.

[23] P. Deep Singh and A. Chug. “Software defect prediction analysis using machine learning algorithms”. In: *2017 7th International Conference on Cloud Computing, Data Science Engineering - Confluence*. Jan. 2017, pp. 775–781.

[24] O. A. Qasem, M. Akour, and M. Alenezi. “The Influence of Deep Learning Algorithms Factors in Software Fault Prediction”. In: *IEEE Access* 8 (2020). Conference Name: IEEE Access. ISSN: 2169-3536.

[25] Martin Shepperd, David Bowes, and Tracy Hall. “Researcher Bias: The Use of Machine Learning in Software Defect Prediction”. In: *IEEE Transactions on Software Engineering* 40.6 (2014), pp. 603–616.

[26] Hadeel Alsolai and Marc Roper. “A systematic literature review of machine learning techniques for software maintainability prediction”. In: *Information and Software Technology* 119 (2020), p. 106214. ISSN: 0950-5849.

[27] Mahmoud O. Elish and Karim O. Elish. “Application of TreeNet in Predicting Object-Oriented Software Maintainability: A Comparative Study”. In: *2009 13th European Conference on Software Maintenance and Reengineering*. 2009, pp. 69–78.

[28] Wei Li and Sallie Henry. “Object-oriented metrics that predict maintainability”. en. In: *Journal of Systems and Software*. Object-Oriented Software 23.2 (Nov. 1993), pp. 111–122. ISSN: 0164-1212.

[29] Ruchika Malhotra[1] and Anuradha Chug. “Software maintainability prediction using machine learning algorithms”. In: *Software engineering: an international Journal (SeiJ)* 2.2 (2012).

[30] Shikha Gupta and Anuradha Chug. “Software maintainability prediction using an enhanced random forest algorithm”. In: *Journal of Discrete Mathematical Sciences and Cryptography* 23.2 (2020), pp. 441–449. eprint: https://doi.org/10.1080/09720529.2020.1728898.

[31] Sandeep Sharawat. “Software maintainability prediction using neural networks”. In: *environment* 3.5 (2012), pp. 750–755.

[32] Sanjay Kumar Dubey, Ajay Rana, and Yajnaseni Dash. “Maintainability Prediction of Object-Oriented Software System by Multilayer Perceptron Model”. In: *SIGSOFT Softw. Eng. Notes* 37.5 (Sept. 2012), pp. 1–4. ISSN: 0163-5948.

[33] Sudan Jha et al. “Deep Learning Approach for Software Maintainability Metrics Prediction”. In: *IEEE Access* 7 (2019), pp. 61840–61855.