Machine Learning and value generation in Software Development: a survey

9 authors, including:

Barakat Akinsanya
Innopolis University
3 PUBLICATIONS   **0** CITATIONS

Mariia Charikova
Innopolis University
2 PUBLICATIONS   **0** CITATIONS

Luiz Jonatã Pires de Araújo
Innopolis University
29 PUBLICATIONS   **54** CITATIONS

Susanna Gimaeva
Innopolis University
4 PUBLICATIONS   **1** CITATION

Some of the authors of this publication are also working on these related projects:

Project   Dynamic Reconfiguration of Business Workflows View project

Project   Approaches to Tackle the Nesting Problems View project

Machine Learning and value generation in Software Development: a survey

Barakat. J. Akinsanya, Luiz J.P. Araújo[1], Mariia Charikova, Susanna Gimaeva, Alexandr Grichshenko, Adil Khan, Manuel Mazzara, Ozioma Okonicha N, and Daniil Shilintsev

Innopolis University, Innopolis 420500, Russia
[1] Corresponding author: `l.araujo@innopolis.university`

Abstract. Machine Learning (ML) has become a ubiquitous tool for predicting and classifying data and has found application in several problem domains, including Software Development (SD). This paper reviews the literature between 2000 and 2019 on the use the learning models that have been employed for programming effort estimation, predicting risks and identifying and detecting defects. This work is meant to serve as a starting point for practitioners willing to add ML to their software development toolbox. It categorises recent literature and identifies trends and limitations. The survey shows as some authors have agreed that industrial applications of ML for SD have not been as popular as the reported results would suggest. The conducted investigation shows that, despite having promising findings for a variety of SD tasks, most of the studies yield vague results, in part due to the lack of comprehensive datasets in this problem domain. The paper ends with concluding remarks and suggestions for future research.

Keywords: Machine learning · Software engineering · Literature review.

1 Introduction

Software has become an essential part of modern everyday life and has a ubiquitous presence in diverse sectors including manufacturing, agriculture and health industries, to mention a few [9]. Efficient software development is, therefore, essential for organisations and requires proper planning and execution in order to generate high quality software at appropriate time and cost. There are several activities involved in this developmental process of a software such as coding, testing and management of the software development cycle. Not surprisingly, issues may arise during the software life-cycle including underestimation of necessary programming effort, poor code and external aspects that implicate in risks to the project [40]. These challenges hinder the growth of businesses since it is considered the top priority for most organisations.

The prediction, mitigation and identification of response actions to issues during software development are complex tasks often performed by human agents who use information and employ subjective expertise [20]. The support and automatising of such tasks has gained increasing attention in the literature.

Researchers over the years have produced different ideas to enhance software development by introducing statistical and regressional models. Some of the prevalent statistical models used for this purpose include Bayesian networks [31], fuzzy logic [16] and system dynamics and discrete event simulation-based models [43].

The use of machine learning (ML) techniques has become increasingly popular in the context of software development [37]. ML is a subfield of artificial intelligence (AI) in which mathematical models identify patterns in the input data and reach a conclusion judging by the data. Thus, such algorithms can learn some information from the input (training data) and afterwards predict the answer for new data (test data). ML techniques include supervised learning, an approach characterised by the existence of prior knowledge of the input-output mapping for a training set; unsupervised learning, which algorithms proceed with no labelled data, and reinforcement learning (reward-based approach) [24]. There are two tasks supervised learning handles: regression (predicting a continuous numerical value) and classification (assigning a label to an item). As it will be indicated in the survey, supervised learning algorithms can be employed to generate software value both for customers and developers.

Software development is a very complicated process which includes many non-obvious things to consider when developing products. Reducing the number of software failures is one of the most challenging problems of software production. This survey aims to investigate different approaches and applications for the use of ML to software development process.

The remaining of this paper is summarised as follows. Section 2 presents the main ML techniques employed for predicting and estimating programming effort. Section 3 shows how these techniques can be used to mitigate risks to the software project. Identifying software defects is performed in the Defects Section (Section 4). A discussion of the main findings from studies on ML embedded into software development processes is presented in Section 5. Suggestions for future work is shown in Section 6.

2   Predicting programming effort

Software effort estimation has received attention since the late 1970s and has been noticed to affect the workflow of the project and its overall success significantly. Moreover, programming effort underestimation often leads to missed deadlines and deterioration of the software quality; effort overestimation, on the other hand, is one of the reasons for project deceleration caused by budget constraints [29]. Many software effort estimation methods have been proposed to accurately estimate effort as a function of a large number of factors. The most widely employed methods [37] include expert models and logical statistical models (parametric models SLIM, COCOMO; regression analysis), traditional machine learning algorithms (Fuzzy Logic, Genetic Algorithms and Regression Trees) and Artificial Neural Networks. According to [37], the coding effort is most often estimated in lines of code (LOC), function points (FP) [14]; use case points (UCP) [1] or in labour hours [46]. This section depicts the most common

approaches for software development effort estimation (SDEE) in the literature, as well as their characteristics.

The importance of accurate effort predictions and the demand for automation of the estimation process have motivated the researchers to propose first parametric models in the early 80s. These models were then tested on the software datasets comprised from the real industrial data of completed projects [22]. According to Srinivasan and Fisher, the three most prominent models are COCOMO, SLIM and Function Points [40]. COCOMO and SLIM models rely almost exclusively on source lines of code (SLOC) as a major input, while the function point approach utilises the number of transactions and other few additional processing characteristics (online updating and transaction rates). Despite being evaluated on the available historical data (COCOMO dataset), the above models have been proven to suffer from inconsistent performances due to the noisy nature of software datasets [2]. Bayesian Networks (BN) is a statistical model used for estimating Agile development effort [15]. Dragicevic, Celar and Turic outlined the benefits of BNs which include the capability of handling vast uncertainties caused by the shortage of relevant information, subjective nature of a number of metrics and difficulties in gathering them. [15].

Another common technique for predicting effort is expert estimation, which is suitable when the domain knowledge is not leveraged by the models [18]. Despite its popularity, expert systems exhibit considerable human bias. One example of such system is Planning Poker, a gamified baseline strategy for SDEE in Agile environments in which developers make estimates by playing numbered cards. In a study by Moharreri et al. Plannig Poker was proven to overestimate in 40% of instances and was shown to have a very high MMRE score of 106.8% [28]. Parametric models and expert systems are still widely used in industry and studies, however the need for better generalisation and overall performance has driven the researchers to apply machine learning methods [40].

Case-based reasoning (CBR) and decision trees (DT) have been among the most effective and researched ML models for SDEE [45]. Results of these models are highly interpretable and are recognised as superior or at least compatible with those of parametric and effort estimation models [5]. It was also asserted by Wen et al. that CBR is more suitable than DTs for this task since it is favourable towards smaller datasets, which is one of the biggest limitations in SDEE research [45]. It is worth mentioning that ensemble models that different methods are often used to gain an even better precision. Moharreri et al. presented experimental evidence that DT coupled with Planning Poker produce better estimations than these models do on their own [28]. Genetic algorithms and fuzzy logic have been used in ensemble models, primarily handling feature selection and imprecise information provided in the datasets [45].

The idea of Artificial Neural Networks (ANNs, or simply NNs), a model that has proven its potential and outperformed traditional ML methods in a number of areas, was first proposed in the 1940s and inspired by biological neurons. ANNs are an attractive approach due to their remarkable computational power: an ability to learn nonlinear relations, high parallelism, noise tolerance, learning

and generalisation capabilities [4]. The drawbacks of applying Neural Networks are as follows: a necessity of large datasets, computational expensiveness and the fact that the results are significantly less interpretable compared to traditional machine learning methods [23]. However, there are some methods to overcome this limitation of interpretability [41].

Comparative study of techniques such as regression tree, k-nearest neighbour, regression analysis and neural networks when applied for software development effort estimation has shown neural networks best estimation ability [23]. Further consideration was given to neural networks by various researchers to emphasize their superior capabilities in effort prediction [14]. Thus, neural networks based models most often provide the best effort estimation compared to traditional ML and their accuracy increases with the amount of data supplied [3].

3   Predicting risks to the project

Several aspects can affect and abuse the software development cycle. Predicting risks is important because it helps to mitigate delays and unforeseen expenses and dangers to the project. As it was mentioned in [13], software development projects always have more risks than other management projects because it has more technical uncertainty and complexity. Most developers look for methodology to minimize the important risks to improve their management, because the risk factor affects the success or failure of any project.

Hu et al. identified the four main types of risks [17]: schedule: the wrong schedule may break the development even at its very first stage; budget: the correct financing is a process that requires the utmost attention to avoid the risks in software development; technical: the developers trying to make changes or fixes in the unknown code will make the relatively big amount of mistakes until they get deep into the details of their task. Even if the damage of one mistake is minor, a big number of such mistakes can be a critical fact for the project; and management risks: risks which may include the bad working environment, insufficient hardware reliability, low effectiveness of the programming etc.

Wauters and Vanhouke proposed a method for continuously assessing schedule risks which uses support vector regression which reads periodic earned value management data from the project control environment, resulting in a more reliable time and cost forecasts [44]. The parameters of the Support Vector Machine have been tuned using a cross-validation and grid search procedure, after which a large computational experiment is conducted. The results showed that the Support Vector Machine Regression outperforms the currently available forecasting methods. Additionally, a robustness experiment has been set up to investigate the performance of the proposed method when the discrepancy between training and test set becomes larger.

The wrong finance distribution will later lead to the unreasonable use of the finances and overall project fault. For solving this problem and predicting risks related with budget and finances distribution Ceylan, Kutlubay and Bener employed regression techniques to detect and identify software defects budget-related [12]. These techniques are used to identify potentially defective software

and allow corrective action to be taken before software is released to the production environment. The results of the initial system structure show that the methods have many faulty defect predictions when the entire dataset is used. When the results are considered in terms of algorithm performances, it is seen that all of the learning algorithms used in the research have similar prediction performances having similar mean square error values.

Even a small number of technical mistakes could be a critical factor for the project. In [39], machine learning classifiers have emerged as a way to predict the existence of a bugs in a change made to a source code file. The classifier is first trained on software history data, and then used to predict bugs. Large numbers of features adversely impact scalability and accuracy of the approach. This technique is applied to predict bugs in software changes, and performance of Na ve Bayes and Support Vector Machine classifiers is characterized.

Management risks in software development are one of the most global type of risks, because if they exist, most of the time they present the most prominent damage. [13] aimed to predict the risks in software development projects by applying multiple logistic regression. The logistic regression was used as a tool to control the software development process. The logistic regression analyses can grade and help to point out the risk factors, which were important problems in development processes. These analytic results can lead to creation and development of strategies and highlighted problems, which are important issues to manage, control and reduce the risks of error.

4   Predicting defects

Software fault prediction is a process which involves the use of software metrics and algorithms to detect software components prone to error. One of the most crucial stages of software development life cycle is the testing stage which involves the most time and effort. It is necessary to detect faults in a software early in the software development life cycle in order to reduce software testing costs. In recent years, researchers have come up with different approaches from machine learning in order to improve the effectiveness of software testing. [30] introduces a model of software testing which uses fault prediction to estimate cost effectiveness.

In machine learning, the task of predicting which part of software prone to fault is known to be a classification task. Classification is the process in which the computer program learns from the data input given to it, alongside algorithms known as machine learning algorithms and then uses this learning to classify new observations. The idea behind these machine learning algorithms is for machines to learn and be able to predict faults in the future. In order for this learning to happen, they have to first identify the defects then classify them. In research, software metrics are put in place to help identify the faults and test the machine learning models. A lot of metrics are used, either method level metric or class level metric. Among them are: lines Of code (LOC), weighted methods for class (WMC), coupling between objects (CBO), response for class (RFC), branch count, unique operand and total operand.

In the work of [11], Artificial Immune Recognition System (AIRS), an immune-insprired supervised learning algorithm, was used to create a defect model based

on method-level metrics and Chidamber-Kemerer metrics suite. [10] on other research work examined nine classifiers for each of the five public NASA datasets. According to the research, naive Bayes algorithm provides the best prediction performance for small datasets, while random forest is the best prediction algorithm for large datasets. [36] compared four classifiers (Naive Bayes, K-star, Random Forest and SVM). Random Forest classifier showed better results for method level metric and SVM for class level metric. [25] used Random Forest, Adaboost, Bagging, Multilayer Perceptron, VM, Genetic Programming. Prediction models to estimate fault proneness using dataset of Open Source "Apache POI" (pure Java library for manipulating Microsoft documents). The best result is shown by random forest and bagging algorithm.

A notable issue in designing machine learning models for software fault detection is the imbalance of data sets. Most researchers focus on developing models which solve this imbalance either by directly influencing the data or not. [34] used the Asymmetric Kernel Principal Component Analysis Classification (AKPCAC) method based off of the kernel principal component regression algorithm proposed by[35] and Asymmetric Kernel Partial Least Squares Classier (AKPLSC) method. [26]. use fuzzy decision tree, a hybrid of fuzzy logic and decision tree which proves better than the decision tree approach.

In fault prediction studies, class level metric show better prediction performance compared to method level metric [21]. The major machine learning algorithms used are Fuzzy Decision Trees, Random Forest, Bagging, AKPCAC, SVM, Naive Bayes, Regression Trees and K-Star. SVM and Random Forests provide best fault prediction models as SVM produces the best accuracy in detecting faults and Random Forest is known to be good for huge dataset. On the whole, a lot of research uses various software metrics and improved machine learning algorithms to detect and predict faults.

Within development philosophies, DevOps is becoming an increasingly adopted approach and attention is rising in both industry and academia giving rise to new project, conferences and training programs [8] [27] [6] [7]. Considering that the DevOps toolchain generates a large quantity of data allowing the extraction of information regarding the status and the evolution of a project, this domain is emerging as particularly suitable for ML applications for SD. Our team is currently working on the implementation of an a ML-based Anomaly Detection System (ADS) and we expect the research community to increasingly focus on this aspect.

5   Discussion

Machine learning techniques have been consistently used in the last decades to provide some assistance for generating high-quality software and a smoother development process (citation needed). An overview of the literature shows that most of research has been focusing on the task of predicting both software quality or error appearance (citation needed). As a result, the software life-cycle is often shortened and the maintaining costs reduced. Moreover, by predicting the occurrence of risks, project managers can mitigate delays and reduce (again) the

chances of project failures (citation needed). The implications and limitations of the use of this computational techniques are discussed as follows.

The survey of the scientific papers on predicting programming effort has shown that machine learning models are continuously gaining popularity in the academic community. The complexity of applied algorithms is rising as more researchers focus on Deep Learning and continue refining less sophisticated ML models with optimisation algorithms [2]. The obtained results challenge the claims of [19] that expert estimation is the most reliable method of effort estimation. Instead the study confirms the potential of ML models to provide reliable solutions to SDEE problem, which was first suggested by [40] as early as 1995. Empirical evidence of ML models' performance allows the developers to have a greater freedom in selecting various models and tailoring them to a specific project. Subsequently, recent progress in the field encourages more and more publications on the topic. However, when it comes to the direct applications in the industry, these models are not used as frequently as their reported performance would suggest. For instance, among Agile practitioners 63% use Planning Poker as the primary estimation tool and 38% prefer expert estimation [42], despite the results of [28]. The reasons behind this phenomenon are a few limitations of the reviewed scientific papers that hinder the reliability of the results. Due to the lack of large software datasets to use as training data, the studies cannot confirm that their particular results will generalise to every real industrial project. Future studies should make an attempt to gather information about recent software projects, as the majority of currently considered datasets are outdated.

In third section we have wanted to consider the most popular types of risks related to software development, which we have chosen from [17], and decide which of them are more important for development process. This information should be taken into account when considering how to manage software project with minimal losses in the development process. We cannot decide which of these risks are most significant, so, as it was said in [13] developers and managers should take into account them all to design really good software project. Because of big difference between considered risks we should use different methods of Machine Learning. Further research is needed to observe a real IT project to find out which of the risks (schedule, budget, technical and management) may affect the development of the project the most negatively. We are also going to find out which risks can be predicted to the maximum extent using Machine Learning.

A substantial amount of research has been conducted with respect to predicting faults and defects using machine learning. The results of the survey conducted show that in predicting faults, machine learning algorithms such as Naive Bayes, K-star, Random Forest and Support Vector Machine have proven to be very beneficial [21] and more favoured. Moreover, some researchers such as [33] and [32] suggest that Case-based reasoning approach using similarity functions such as Euclidean distance and Manhattan distance to determine the most similar cases, yields encouraging results. While previous research failed to take

into consideration the problem of data-set imbalance [38], the outcome of the survey demonstrates that the imbalance has been accommodated. However, it is beyond the scope of this study to specify the metrics which are relevant in predicting faults. Further research has to make plans for generating new datasets as the available ones, mostly NASA and PROMISE have been used severally.

The overview of the literature shows that some ML techniques, namely case-based reasoning and neural networks, are particularly popular in this field, as shown in Table 1. Case-based reasoning is favoured due to its ability to produce high accuracy given limited data, while neural networks are popular due to their ability to learn complex functions and handle outliers [45]. The reported results build on existing evidence of the usefulness of ML embedded into the software development process. The reliability of such data, however, is affected by the limited available data and the lack for a united and shared dataset.These aspects indicate the need for the development of larger datasets that are representative of current tendencies in software engineering in order to provide researchers with quality training data and allow them to draw reliable conclusions. Future studies should take into account recent developments in the field of ML, such as reinforcement learning, convolutional and recurrent neural networks, providing their applications to software development, which have been scarce to the best of the authors' knowledge.

6   Conclusion and future research

The survey of Machine Learning applications to software development process showcases a considerable progress in the field over last decades. Across three outlined subfields (effort estimation, risks and defects prediction) ML models have been deployed and achieved satisfactory results that are in the majority of cases commensurate to traditional approaches or even surpass them. Literature analysis have also established that increasing research interest in this area provides practitioners a variety of models to apply to their particular project. Given this abundance of models, comparative studies rarely reach consensus about whether traditional regression, classification or deep learning approach is generally preferable in software development.

In the subfield of predicting risks to the software project regression models are considered dominant over other ML models as well as state-of-the-art non-ML methods. Specifically, the performance of Support Vector Machine is frequently noted in regards to predicting schedule and budget risks. On the other hand, defect prediction favours classification algorithms with Random Forest being one of the most reliable models. Research in programming effort estimation initially favoured regression models, however recent breakthroughs confirmed superior accuracy by Cascade Correlation Neural Networks.

Notable gaps in the current state of the research on the topic include investigating larger scope of applications for Artificial Neural Networks and reinforcement learning. Despite that ANNs have shown very promising results in software effort estimation, the research about their applications in two other subfields have been rather scarce. Similar patter is observed regarding reinforce-

ment learning, which was not yet applied to any of the software development tasks mentioned in this paper.

For future work, it is recommended that researchers attempt to use larger datasets and those that are more representative of the current state of software engineering in order for the models' assessment to be complete and reliable. Moreover, it is advised that closer interaction between academic and industrial communities needs to be established to facilitate deployment of ML models on real-world software projects.

References

1. Ajitha, S., Kumar, T.S., Geetha, D.E., Kanth, K.R.: Neural network model for software size estimation using use case point approach. In: 2010 5th International Conference on Industrial and Information Systems. pp. 372–376. IEEE (2010)
2. Azzeh, M.: Software effort estimation based on optimized model tree. In: Proceedings of the 7th International Conference on Predictive Models in Software Engineering. p. 6. ACM (2011)
3. Bardsiri, A.K., Hashemi, S.M.: Machine learning methods with feature selection approach to estimate software services development effort. International Journal of Services Sciences **6**(1), 26–37 (2017)
4. Basheer, I.A., Hajmeer, M.: Artificial neural networks: fundamentals, computing, design, and application. Journal of microbiological methods **43**(1), 3–31 (2000)
5. Baskeles, B., Turhan, B., Bener, A.: Software effort estimation using machine learning methods. In: 2007 22nd international symposium on computer and information sciences. pp. 1–6. IEEE (2007)
6. Bobrov, E., Bucchiarone, A., Capozucca, A., Guelfi, N., Mazzara, M., Masyagin, S.: Teaching devops in academia and industry: reflections and vision. CoRR **abs/1903.07468** (2019), http://arxiv.org/abs/1903.07468
7. Bobrov, E., Bucchiarone, A., Capozucca, A., Guelfi, N., Mazzara, M., Naumchev, A., Safina, L.: Devops and its philosophy : Education matters! CoRR **abs/1904.02469** (2019), http://arxiv.org/abs/1904.02469
8. Bruel, J., Mazzara, M., Meyer, B. (eds.): Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment - First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018, Revised Selected Papers, Lecture Notes in Computer Science, vol. 11350. Springer (2019)
9. Casale, G., Chesta, C., Deussen, P., Di Nitto, E., Gouvas, P., Koussouris, S., Stankovski, V., Symeonidis, A., Vlassiou, V., Zafeiropoulos, A., et al.: Current and future challenges of software engineering for services and applications. Procedia Computer Science **97**, 34–42 (2016)
10. Catal, C., Diri, B.: Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Information Sciences **179**(8), 1040–1058 (2009)
11. Catal, C., Diri, B., Ozumut, B.: An artificial immune system approach for fault prediction in object-oriented software. In: 2nd International Conference on Dependability of Computer Systems (DepCoS-RELCOMEX'07). pp. 238–245. IEEE (2007)
12. Ceylan, E., Kutlubay, F.O., Bener, A.B.: Software defect identification using machine learning techniques. In: 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06). pp. 240–247. IEEE (2006)

13. Christiansen, T., Wuttidittachotti, P., Prakancharoen, S., Vallipakorn, S.A.o.: Prediction of risk factors of software development project by using multiple logistic regression. ARPN Journal of Engineering and Applied Sciences **10**(3), 1324–1331 (2015)
14. Dave, V.S., Dutta, K.: Neural network based models for software effort estimation: a review. Artificial Intelligence Review **42**(2), 295–307 (2014)
15. Dragicevic, S., Celar, S., Turic, M.: Bayesian network model for task effort estimation in agile software development. Journal of systems and software **127**, 109–119 (2017)
16. Engel, A., Last, M.: Modeling software testing costs and risks using fuzzy logic paradigm. Journal of Systems and Software **80**(6), 817–835 (2007)
17. Hu, Y., Huang, J., Chen, J., Liu, M., Xie, K.: Software project risk management modeling with neural network and support vector machine approaches. In: Third International Conference on Natural Computation (ICNC 2007). vol. 3, pp. 358–362. IEEE (2007)
18. Jørgensen, M.: A review of studies on expert estimation of software development effort. Journal of Systems and Software **70**(1-2), 37–60 (2004)
19. Jørgensen, M.: What we do and don't know about software development effort estimation. IEEE software **31**(2), 37–40 (2014)
20. Jorgensen, M., Shepperd, M.: A systematic review of software development cost estimation studies. IEEE Transactions on software engineering **33**(1), 33–53 (2006)
21. Karim, S., Warnars, H.L.H.S., Gaol, F.L., Abdurachman, E., Soewito, B., et al.: Software metrics for fault prediction using machine learning approaches: A literature review with promise repository dataset. In: 2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom). pp. 19–23. IEEE (2017)
22. Kemerer, C.F.: An empirical validation of software cost estimation models. Communications of the ACM **30**(5), 416–429 (1987)
23. Kim, Y., Lee, K.: A comparison of techniques for software development effort estimating. SYSTEM p. 407 (2005)
24. Lison, P.: An introduction to machine learning. Language Technology Group (LTG), 1 **35** (2015)
25. Malhotra, R., Jain, A.: Fault prediction using statistical and machine learning methods for improving software quality. Journal of Information Processing Systems **8**(2), 241–262 (2012)
26. Marian, Z., Mircea, I.G., Czibula, I.G., Czibula, G.: A novel approach for software defect prediction using fuzzy decision trees. In: 2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). pp. 240–247. IEEE (2016)
27. Mazzara, M., Naumchev, A., Safina, L., Sillitti, A., Urysov, K.: Teaching devops in corporate environments - an experience report. In: Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment - First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018, Revised Selected Papers. pp. 100–111 (2018)
28. Moharreri, K., Sapre, A.V., Ramanathan, J., Ramnath, R.: Cost-effective supervised learning models for software effort estimation in agile environments. In: 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC). vol. 2, pp. 135–140. IEEE (2016)
29. Molokken-Ostvold, K., Jorgensen, M.: A comparison of software project overruns-flexible versus sequential development models. IEEE Transactions on Software Engineering **31**(9), 754–766 (2005)

30. Monden, A., Hayashi, T., Shinoda, S., Shirai, K., Yoshida, J., Barker, M., Matsumoto, K.: Assessing the cost effectiveness of fault prediction in acceptance testing. IEEE Transactions on Software Engineering **39**(10), 1345–1357 (2013)
31. Perkusich, M., Soares, G., Almeida, H., Perkusich, A.: A procedure to detect problems of processes in software development projects using bayesian networks. Expert Systems with Applications **42**(1), 437–450 (2015)
32. Rashid, E., Patnayak, S., Bhattacherjee, V.: A survey in the area of machine learning and its application for software quality prediction. ACM SIGSOFT Software Engineering Notes **37**(5), 1–7 (2012)
33. Rashid, E.A., Patnaik, S.B., Bhattacherjee, V.C.: Machine learning and software quality prediction: as an expert system. International Journal of Information Engineering and Electronic Business **6**(2), 9 (2014)
34. Ren, J., Qin, K., Ma, Y., Luo, G.: On software defect prediction using machine learning. Journal of Applied Mathematics **2014** (2014)
35. Rosipal, R., Girolami, M., Trejo, L.J., Cichocki, A.: Kernel pca for feature extraction and de-noising in nonlinear regression. Neural Computing & Applications **10**(3), 231–243 (2001)
36. Shanthini, A., Chandrasekaran, R.: Applying machine learning for fault prediction using software metrics. International Journal of Advanced Research in Computer Science and Software Engineering **2**(6), 274–284 (2012)
37. Sharma, P., Singh, J.: Systematic literature review on software effort estimation using machine learning approaches. In: 2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS). pp. 43–47. IEEE (2017)
38. Shatnawi, R.: Improving software fault-prediction for imbalanced data. In: 2012 international conference on innovations in information technology (IIT). pp. 54–59. IEEE (2012)
39. Shivaji, S., Whitehead Jr, E.J., Akella, R., Kim, S.: Reducing features to improve bug prediction. In: 2009 IEEE/ACM International Conference on Automated Software Engineering. pp. 600–604. IEEE (2009)
40. Srinivasan, K., Fisher, D.: Machine learning approaches to estimating software development effort. IEEE Transactions on Software Engineering **21**(2), 126–137 (1995)
41. Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 3319–3328. JMLR. org (2017)
42. Usman, M., Mendes, E., Börstler, J.: Effort estimation in agile software development: a survey on the state of the practice. In: Proceedings of the 19th international conference on Evaluation and Assessment in Software Engineering. p. 12. ACM (2015)
43. Uzzafer, M.: A simulation model for strategic management process of software projects. Journal of Systems and Software **86**(1), 21–37 (2013)
44. Wauters, M., Vanhoucke, M.: Support vector machine regression for project control forecasting. Automation in Construction **47**, 92–106 (2014)
45. Wen, J., Li, S., Lin, Z., Hu, Y., Huang, C.: Systematic literature review of machine learning based software development effort estimation models. Information and Software Technology **54**(1), 41–59 (2012)
46. Wright, I., Ziegler, A.: The standard coder: a machine learning approach to measuring the effort required to produce source code change. arXiv preprint arXiv:1903.02436 (2019)

Table 1. Machine Learning for Software Development in academic literature.

Reference	Task	ML model	Data
Azzeh (2011)	SEE[1]	Decision Tree	PROMISE and ISBSG datasets
Bardsiri and Hashemi (2017)	SEE[1]	Regression Trees, ANN	ISBSG and NASA datasets
Baskeles, Turhan, and Bener (2007)	SEE[1]	Multilayer Perception, Regression Trees, Support Vector Regression	NASA and USC datasets
Catal, Diri, and Ozumut (2007)	SFP[2]	Artificial Immune Systems paradigm	PROMISE dataset
Ceylan, Kutlubay, and Bener (2006)	SFP[2]	Decision Trees, Multilayer Perception, Radial Basis Functions	NASA dataset
Clemente, Jaafar, and Malik (2018)	SFP[2]	ANN, Random Forest, Decision Trees, Naive Bayes, SVM	SeaMonkey, Mozilla Firefox
Dragicevic, Celar, and Turic (2017)	SEE[1]	Bayesian Network	Historical data
Hu et al. (2007)	SRP[3]	ANN, Support Vector Machine	Questionnaire based data
Joseph (2015)	SRP[3]	ANN	Oracle dataset
Karim et al. (2017)	SFP[2]	SVM, ANN, Naive Bayes, Random Forest	PROMISE dataset
Kim and Lee (2005)	SEE[1]	ANN, Regression Tree	ISBSG dataset
Marian et al. (2016)	SFP[2]	Fuzzy decision tree	JEdit(version4.2), Ant(version 1.7)
Moharreri et al. (2016)	SEE[1]	Decision Trees, Random Forest, Logistic Model Tree, Naive Bayes	IBM Rational Team Concert data
Nassif et al. (2016)	SEE[1]	ANN	ISBSG dataset
Panda,Satapathy, and Rath (2015)	SEE[1]	ANN	Zia dataset
Perkusich et al. (2015)	SFP[2]	Bayesian Networks	Case studies in software companies
Ren et al. (2014)	SFP[2]	Partial Least Squares and Kernel principal component analysis	NASA and SOFTLAB datasets
Sharma and Singh (2017)	SEE[1]	ANN, Fuzzy logic, Genetic Algorithms, Regression Trees	NASA, ISBSG, Desharnais and COCOMO datasets.
Shepperd and Schofield (1997)	SEE[1]	Case-Based Reasoning	Albrecht, Atkinson, Desharnais, Finnish and MM2 datasets
Wright and Ziegler (2019)	SEE[1]	Neural Hidden Markov Model, Deep Mixture Density Networks	LGTM dataset

[1]SEE: Software Effort Estimation
[2]SFP: Software Fault Prediction
[3]SRP: Software Risks Prediction