This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

A Literature Review of Machine Learning and Software Development Life cycle Stages

SAAD SHAFIQ, ATIF MASHKOOR, CHRISTOPH MAYR-DORN AND ALEXANDER EGYED
Johannes Kepler University, Linz, Austria

Corresponding author: Saad Shafiq (e-mail: saad.shafiq@jku.at)

The research reported in this article has been partly funded by the LIT Artificial Intelligence Lab and the LIT Secure & Correct Systems Lab supported by the state of Upper Austria.

ABSTRACT The software engineering community is rapidly adopting machine learning for transitioning modern-day software towards highly intelligent and self-learning systems. However, the software engineering community is still discovering new ways how machine learning can offer help for various software development life cycle stages. In this article, we present a study on the use of machine learning across various software development life cycle stages. The overall aim of this article is to investigate the relationship between software development life cycle stages, and machine learning tools, techniques, and types. We attempt a holistic investigation in part to answer the question of whether machine learning favors certain stages and/or certain techniques.

INDEX TERMS Software engineering, Machine learning, Literature review

I. INTRODUCTION

The software engineering (SE) community is continuously looking for better and more efficient ways of building high-quality software systems. However, in practice, the strong emphasis on time to market tends to ignore many, well-known SE recommendations. That is, practitioners focus more on programming as compared to requirements gathering, planning, specification, architecture, design, and documentation – all of which are ultimately known to greatly benefit the cost-effectiveness and quality of software systems. Lack of human resources is often cited as the main reason for doing so. Herein lies the great potential for machine learning (ML) since its algorithms are proven to be most befitting to problem domains that aim to replicate human behavior. Hence, it stands to reason that human-centric SE activities should also benefit from ML [1].

The growing demand for agility and the ability to solve complex problems in SE has already led researchers to explore the potential of ML in this field. To date, ML has many demonstrated benefits. Applications of ML for SE range from resolving ambiguous requirements to predicting software defects [2]. For example, Sultanov et al. [3] used reinforcement learning (a type of ML) on understanding the relationships among software requirements at different levels of abstraction. Their approach shows how ML can automatically generate traceable links between high-level and low-level requirements. However, ML is not a single technique but rather an assortment of techniques. The challenge of using ML for SE is thus not only about finding the right way of solving the problem but also about comparing various ML techniques and their potential. For example, several researchers have explored predictions in order to better estimate the time to market for software products. For this purpose, various ML techniques were used and compared, e.g., artificial neural networks, rule induction, case-based reasoning, support vector machines, regression-based decision trees, and random forest [4, 5, 6, 7].

In many areas of science and engineering, such as image recognition or autonomous driving, ML has already revolutionized development. The applications of ML to SE are also increasing significantly, which is evident through the exponential growth in the number of articles on ML for SE being published every year. Consequently, it is of interest to understand, which software development life cycle (SDLC) stages benefit the most from this trend; or even to understand which ML techniques are most suitable for which SDLC stage(s). This leads to the motivation of conducting this study.

In this article, we provide a *bird's-eye view* on the current state-of-the-art regarding the causal relationship between ML and SDLC stages and suggest the open areas of research where more primary studies are needed. The fairly broad scope of this study is *by design*. While this article sets out to explore the causal relationship between machine learning and SDLC stages in the form of a *literature review*, it should be noted that some specialized studies already exist, e.g., ML for automated software testing [8]. Similar to exploratory studies conducted in the past, such as Bindewald et al. [9], our review is based on the quantitative analysis of the articles present in the literature addressing the application of ML to various SDLC stage(s).

The rest of the article is organized as follows. The related work is discussed in Section II. Section III presents a brief introduction to ML. Section IV explains the research methodology and protocol followed in the study. Results of the study are discussed in Section V. Further analysis on the presented state-of-the-art is discussed in Section VI. Section VII presents challenges, limitations, and future research directions of this field. Section VIII discusses different threats to the validity of the presented results. The article is concluded in Section IX.

II. RELATED WORK

Some studies, e.g., [8, 10, 11, 12], have already analyzed the application of ML in SE in the past. Durelli et al. [8] conducts a systematic mapping study on the application of ML for software testing. The study highlights the use of ML techniques in various software testing activities such as test-case generation and oracle construction. Results of the study show that a vast majority of articles employ supervised learning, such as ANN and DT, to solve testing-related problems. Moreover, the key advantages and disadvantages of using ML for software testing are discussed. Mainly, the advantage of ML techniques is their scalability and efficient application to large-scale and complex software systems. The disadvantage, on the other hand, is the unavailability of data that fits well with the learning process.

Fajardo et al. [10] provides an extensive overview of applying data mining techniques to SE tasks including open issues and recommendations for improvements. The study provides a comprehensive list of data mining techniques applicable in SE, e.g., aspects related to clustering, regression, and performance metrics. Moreover, the study highlights some widely used datasets of SE employed in the data mining articles and states key advantages of mining SE data.

Wan et al. [11] performed a survey by interviewing 14 people from 3 companies and 342 respondents from 26 countries. The aim of the study was to understand and highlight the key differences in the software development practices followed in building ML and non-ML software systems. Results suggested that ML engineers should focus on handling the uncertain randomness of ML systems and work on employing version control systems specifically for ML applications in order to improve reproducibility.

Zhang et al. [12] conducted the research focusing on the application of ML in SE. The study provides a comprehensive list of SE tasks whose problems can be addressed using ML techniques. The study also emphasizes the fact of SE to be a highly fertile area to explore with regards to applying ML techniques by formulating SE tasks as learning problems and addressed using ML techniques.

In contrast to the aforementioned focused studies, our study provides a broader context and a comprehensive list of articles that help identify the relationship between various ML techniques and SDLC stages. We also provide the relationships of ML types, tools, and techniques with respect to SE stages, which help better understanding the current progress of the adoption of ML for SE.

III. INTRODUCTION TO ML

ML has evolved drastically over the recent years and is now being employed on a global scale. As a subset of artificial intelligence, ML has been considered vital when developing software systems for domains such as speech/image recognition [13] or automotive [14]. ML techniques have also been used to address various SE issues and activities. Most commonly, ML has been employed in defect prediction, effort estimation, and identifying patterns and similarities in the source code.

The ML techniques are essentially targeted to solve problems, which can often become hard to analyze by people causing misinformation [15]. These problems have various types, which can be categorized as ML types. ML types generally include supervised, unsupervised, and reinforcement learning. Most of the applications of ML consist of problems that can be deemed of type supervised learning. It refers to learning from features along with their known class labels. Then, predicting the class labels from new unseen features. These problems are also often categorized as classification problems.

Arguably, ML techniques can also be classified into two divisions. 1) Traditional ML techniques that include decision trees (DT), random forest (RF), linear regression, logistic regression (LR), support vector machines (SVM), k-nearest neighbors (KNN), and naive bayes (NB). 2) Neural network-based ML techniques that include artificial neural network (ANN), recurrent neural network (RNN), and convolutional neural network (CNN). Deep learning (DL) – also known as deep neural networks (DNN) – is a subset of ANN. DL was introduced mainly to address the data scalability problems such as handling high-dimensional and large-scale datasets. Structurally, instead of comprising a single hidden layer within the input and output layer as in ANN, RNN and CNN techniques are composed of multiple hidden layers of interconnected neurons. The processing inside the hidden layers is based on the principle of weighted connections. In general, each hidden neuron is comprised of predetermined weight and bias values, which are later adjusted during the training process until the desired output is reached. Lastly, the output layer holding the acquired weight and bias values represents the solution to the given problem.

IV. RESEARCH METHODOLOGY

In order to direct the study, we followed the Goal, Question, and Metric (GQM) paradigm suggested by Basili et al. [16]. The aim of the GQM paradigm is to guide the study by specifying its goals in order to have an objective-oriented data extraction process. It also helps in tracing goals to formulated questions leading to better interpretation of the data in line with goals stated before.

A. GOALS

The overall aim of this study is to evaluate the relation between ML and SDLC stages. The considerably broader aim of this study differentiates it from other similar studies, such as the one by Durelli et al. [8], which have quite a narrow focus. Following are the goals formulated for this study.

G1. To identify the susceptibility of various ML types and techniques to SDLC stages

G2. To understand the maturity of research in this area

G3. To identify the demographics in this area

G4. To understand the implications, challenges, limitations, and future research directions in this area

The first three goals lead to the research questions discussed in the following subsection. Due to the descriptive and elaborate nature of the fourth goal, we decided to thoroughly discuss it in Sections VI and VII.

B. QUESTIONS

In order to meet the outlined goals, we have formulated concrete questions and identified suitable metrics (quantifiable attributes). The questions and metrics (emphasized) are explained as follows:

G1. The susceptibility of various ML types and techniques to SDLC stages

Q1.1. What *SDLC stages* are being focused on by academic and industrial researchers in this area?
Rationale: Our interest is to understand what SDLC stage the researchers tend to focus on, whether, a particular SDLC stage or the amalgamation of two or more. The SDLC stages are based on, but not limited to, the knowledge areas mentioned in SWEBOK [17] characterizing the practice of SE, e.g., Software Requirements, Software Design, or Software Maintenance.

Q1.2. What are the *applications of ML* for SE?
Rationale: We are interested to know about the specific applications of ML that exist in SE, e.g., whether an ML technique was used to automate test case generation or to predict potential bugs in the system.

Q1.3. What *type of ML and its techniques* are being employed for SE?
Rationale: We are interested to know whether a particular type/technique was consistently employed for a specific life cycle stage. The type of ML refers to how the models have been trained, e.g., supervised, semi-supervised, or unsupervised. Whereas the ML techniques are the algorithms used for classification or clustering problems, e.g., SVM, RF, or ANN.

G2. The maturity of research in the area

Q2.1. What is the *contribution facet* of the articles?
Rationale: The contribution facet refers to the novel contributions made by the researchers in the articles. It partially corroborates the attributes provided by Banerjee et al. [18] and Petersen et al. [19], and are supplemented by our own views obtained by analyzing the extracted articles. The attributes are defined as follows:

- **Tool:** Article proposing a new tool or improving an existing one and describing its evaluation.
- **Approach/method:** Article proposing a new approach/method or improving the existing one.
- **Model/framework:** Article introducing a new model/framework or improving the existing one.
- **Algorithm/process:** Article proposing a new algorithm/SE process or improving the existing one.
- **Comparative analysis:** Article evaluating different approaches and reporting results of the comparative study.

Q2.2. What is the *research facet* of the articles?
Rationale: The research facet of an article refers to the maturity of the research in terms of empirical evidence provided in the article or whether an article was proposing a solution or evaluating an existing approach. The research facet is defined as follows:

- **Evaluation:** Article evaluating or validating the proposed approach using empirical methods.
- **Knowledge:** Article describing the experiences and opinions of authors on the existing approaches.
- **Solution:** Article proposing a new solution and describing its applicability with the help of examples and arguments.

We further explored the evaluation facet in order to understand the empirical methods employed in the articles.

Q2.3. What *datasets* are commonly employed in the articles?
Rationale: We are interested to know about the datasets that are most commonly used to evaluate the research results in the domain of ML for SE.

G3. The demographics of research in the area

Q3.1. What are the *trends in terms of years* of publications in the area?
Rationale: The trends in terms of years refer to the number of publications varying from a year to another. Here, we want to assess how active this research area is.

Q3.2. What are the *highest publishing venues* of the area?
Rationale: We are interested to know about the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

venues, which have the highest publications with respect to the area of ML for SE.

C. ARTICLES EXTRACTION

Query formulation

In order to search for relevant literature, following the guidelines proposed by Petersen et al. [19], we devised a query that uses a two-element PICO search. **Problem 'P':** (requirement, specification, design, model, analysis, architecture, implementation, code, test, verification, validation, maintenance), and **Intervention 'I':** (machine learning, deep learning). We have not considered Comparison 'C' and Outcome 'O' as this is out of the scope of this study. Following is the resultant query that was eventually used in all digital libraries:

("machine learning" OR "deep learning") AND software AND requirement OR specification* OR design* OR model* OR analysis OR architecture OR implementation OR code OR test* OR verification OR validation OR maintenance*[1]

Digital libraries

The query was applied to titles and abstracts of articles in five well-known digital libraries: ACM Digital Library[2], IEEEXplore[3], ScienceDirect[4], Springer[5], and Web of Science[6]. According to [20], these digital libraries are among the most popular sources in computer science and engineering that ensure high coverage of potentially relevant studies. We did not include Google Scholar[7] in our study as the search results of Google Scholar tend to be repetitive with respect to results from the included digital libraries, and its unique contribution to the search process is unclear [20].

Time period

We scope the time period of related studies published from 1991 to 2020. The earliest paper we could find in our study was published in 1991, hence the starting time. We conducted the search in Q1 2021 and made sure that the results are reproducible until 2020, hence the ending time.

Articles selection

All repositories, except Springer, returned the number of articles as shown in Fig. 1. Springer initially yielded 4,502 articles as a result of the query; however, most of these articles were quite irrelevant to the scope of our study even after applying filters, such as "Computer Science" as discipline, and "SE" and "AI" as sub-disciplines, to reduce the search space. We then went through the titles and abstracts of the articles (if the goal of the article is unclear from the title) and stopped the search process when the first page with all irrelevant articles was reached. This resulted in 46 articles.

[1] Asterisk (*) is a wildcard that refers to zero or more characters in a word
[2] https://dl.acm.org/
[3] https://ieeexplore.ieee.org/
[4] https://www.sciencedirect.com/
[5] https://www.springer.com/
[6] https://apps.webofknowledge.com/
[7] https://scholar.google.com

In total, we extracted 565 articles, as shown in Fig. 1. However, many of them were duplicated as one article may appear in many digital libraries. We then removed duplicates, which resulted in 501 unique articles. The articles then underwent a screening process and were scrutinized based on the following inclusion criteria.

1) Articles that were relevant to the scope, i.e., relevance and appropriateness of the article correspond to the research goals of the study, were included.
2) Articles that were available in the full-text format were included.
3) Articles demonstrating well-established empirical soundness were included.
4) Articles of more than a single page were included.
5) Articles that were peer-reviewed were included.
6) Articles that were entirely written in English were included.

Consequently, a total of 263 relevant articles were selected and included in the final pool. Fig. 1 shows the overall article extraction process including the number of articles extracted from each repository and the final pool.

Tool

Conducting a literature review is a tedious and time-consuming task. It usually involves the search, collection, filtration, and classification of a huge amount of papers. Without a helping tool, this is a very difficult endeavor. In this work, we used Mendeley[8] and Google Sheets.[9] These tools helped us in importing, organizing, and analyzing search results.

D. CLASSIFICATION SCHEME

We later defined a classification scheme to ensure accurate assessment of attributes [19]. The generalized attributes obtained were then sorted by the authors of this study based on the knowledge areas provided in SWEBOK [17]. In fact, the knowledge areas mentioned in SWEBOK were not strictly used in the categorization but merely employed as a defining factor to providing a high-level abstraction of attributes that represented the set of articles. However, we referred to the following knowledge areas while devising the categories in this study: 1) software requirements, 2) software design, 3) software construction, 4) software quality, and 5) software maintenance. During the article sorting process, certain articles were found to be equivocal. In such cases, we associated those attributes to the articles that received majority votes from the authors of this study. To get a better understanding, a graphical representation of the workflow starting from the attribute extraction process leading to the resulting classification scheme is shown in Fig. 2.

V. STUDY FINDINGS

Here we answer the RQs, which we discussed in Section IV.

[8] https://www.mendeley.com
[9] https://www.google.com/sheets/about

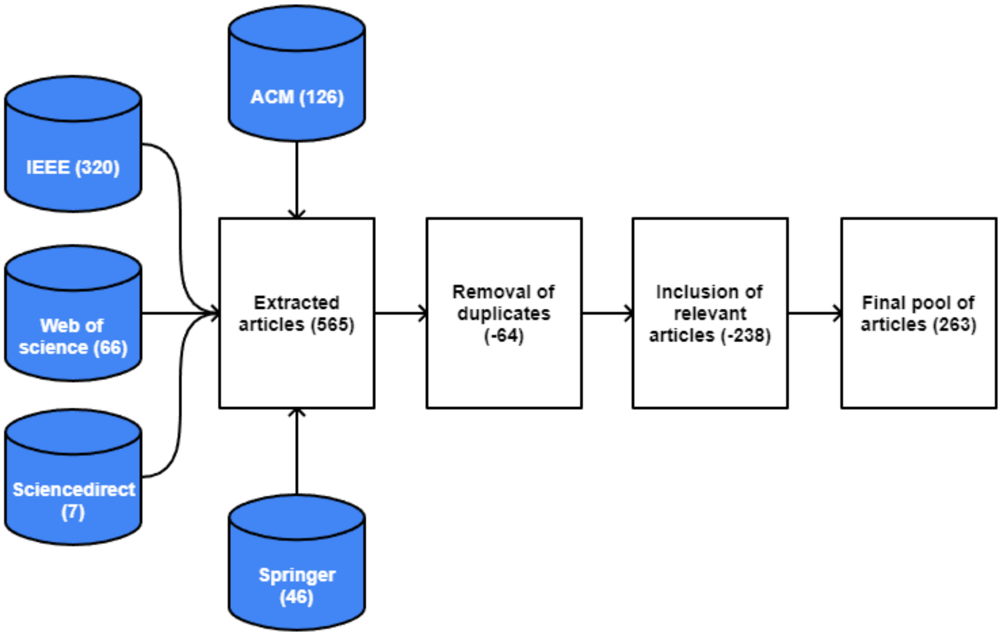


FIGURE 1: Article extraction process

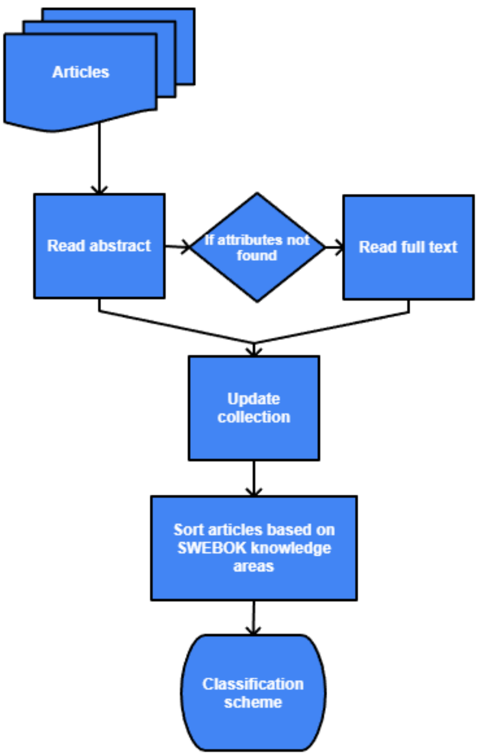


FIGURE 2: Attribute extraction leading to the classification scheme

A. SDLC STAGES (Q1.1)

As already discussed, based on the analysis of articles, we have grouped them into 5 major categories (inspired by the aforementioned knowledge areas in SWEBOK [17]). These categories are briefly described as follows:

Software requirements

We group all those articles in this category, which are concerned with the elicitation, modeling, analysis, prioritization, and validation of software requirements.

Software architecture and design

We group all those articles in this category, which deal with the process of specifying the architectural components and interfaces of software, and the description of how components of a software system are organized.

Software implementation

We group all those articles in this category, which are concerned with the development or construction of software achieved through a combination of design artifacts and coding.

Software quality assurance and analytic

We group all those articles in this category, which deal with fundamental elements such as quality characteristics, quality process improvement or assessment, or verification and validation. We have also included articles referring to software testing in this category.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

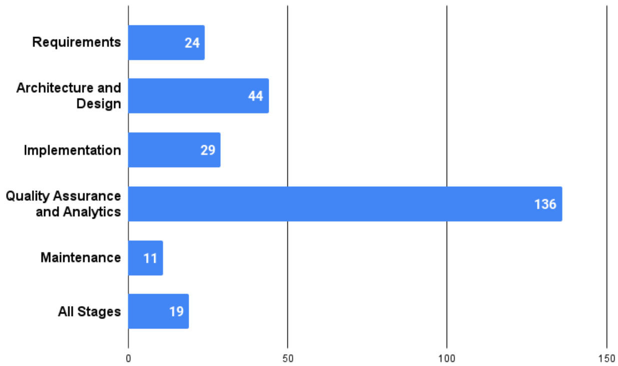


FIGURE 3: Articles by SDLC stages

Software maintenance

We group all those articles in this category, which deal with software adherence activities in order to meet new or changed operating environments such as refactoring, maintenance cost estimation, defect correctness, and factors related to software aging (e.g., resource depletion).

The SE stages and the number of articles that are associated with those stages are shown in Fig. 3. 136 out of 263 (52%) articles belong to quality assurance and analytic. 44 out of 263 (17%) articles have focused on architecture and design. 29 out of 263 (11%) articles have addressed the implementation followed by requirements engineering stage with 24 out of 263 (9%) articles. 11 (4%) articles were focusing on the maintenance phase. The rest of the articles were not particularly focusing on any stage but were generally applicable to SE.

B. APPLICATIONS OF ML FOR SE (Q1.2)

To address this question, we have developed a classification scheme based on the identified applications of ML for SE in order to characterize the obtained articles into appropriate categories. We have organized the applications of ML for SE as branches, which belong to five life cycle stages of SE (knowledge areas). The applications of ML for SE that come under corresponding SDLC stages along with the references of articles are shown in Table 1. Note that the applications highlighted in this study may not cover the entire knowledge base but rather should be deemed as stemming research indicating key applications of ML for SE in literature. The applications falling under the SE stages are described below.

Applications of ML aiming at software requirements

ML has been widely used to facilitate the software requirements stage. For instance, in requirements modeling and analysis, articles focused on distinguishing ambiguous requirements [21], resolving incompleteness, the correctness of requirements [22], etc. Requirements selection/prioritization/ classification deals with articles proposing ML techniques that emphasize on automating prioritization of requirements or their classification. Perini et al. [23] employed an ML technique to generate approximate rank in order to prioritize requirements. Navarro-Almanza et al. [24] used a convolutional neural network (CNN) to classify functional requirements by analyzing textual features. We further found articles focusing on requirements traceability. Requirements traceability refers to the ML approaches that assist in linking requirements to code or other artifacts as shown by Guo et al. [25], who used deep learning (DL) techniques in order to generate a trace link of requirements with other artifacts.

Applications of ML aiming at software architecture and design

Many types of research in the past have applied ML to software architecture and design. The applications include design models, which are comprised of recommendation models for software processes/services. Apart from this, model smells and refactoring techniques of object-oriented structures using ML have also been proposed in the articles. White et al. [26] introduced DL to software language modeling based on information retrieval models. Design pattern prediction primarily focuses on recognizing design patterns in software through source code or user interface layout using ML techniques. For example, Nguyen et al. [27] proposed an approach known as DeepUI in order to semi-automate the design tasks by learning from previous UI design patterns. Development effort estimation refers to estimating effort for the development of software projects using ML techniques. Ionescu [28] used ANN to automate effort estimation by learning from textual features of project tasks.

Applications of ML aiming at software implementation

We found several studies on ML assisting the software implementation stage. Among many applications, code clone/ localization/refactoring/ labeling aims at finding code duplication, specific location of code in software, refactoring of code, or labeling of code with the help of ML, e.g., Alahmadi et al. [29] employed CNN in order to predict the code blocks in video tutorials. Code/bad smell detection focuses on applying ML in order to detect code and bad smells in software source code and design models, respectively. Code smells are indications of poor software code quality leading to the rise of technical debt. It generally includes god classes, spaghetti code, etc., whereas bad smells in design models have similar characteristics such as lazy classes and middle man. Pecorelli et al. [30] investigated data balancing techniques and addressed unbalanced dataset issues when employing ML for code smell detection. Maneerat et al. [31] proposed an approach to predict bad smells from design models such as class diagrams. Code inspection/analysis represents the class in which an ML technique is employed for the purpose of code reviews. For instance, Lal et al. [32] proposed an ML approach to automate code reviews for the pushed code. The code/program similarity category refers to the identification of specific piece(s) of code, which are similar between two or more software projects. Additionally, Kim et al. [33] proposed an ML technique in order to reduce the number

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

of program similarity comparisons aimed at distinguishing between original and pirated/cracked software.

Applications of ML aiming at software quality and analytic

Most of the articles we found were focusing on applying ML to various software quality assurance and analytic tasks. The applications include: fault/bug/defect prediction category, which revolves around the prediction of faults, bugs, or defects using ML techniques [34, 35, 36, 37, 38, 39]. Test case/ data/oracle generation surrounds ML techniques that help in generating test data, test oracles, or entire test suites. Braga et al. [40] proposed an ML technique to automate the process of test oracle generation. Test case selection/prioritization/ classification deals with the class that particularly focuses on test case prioritization or classification techniques using ML. Rosenfeld et al. [41] employed an ML technique in order to select generic test cases for android applications. The technique is aimed at reducing the manual testing efforts by classifying the activities and automatically selecting the activity-specific test cases. Vulnerability/anomaly/malware discovery/analysis mostly concerns the security aspect of the software, e.g., Huang et al. [42] employed the term frequency-inverse document frequency (TF-IDF) technique and deep neural network to automatically classify software vulnerabilities. Software analysis, technique assessment, and software process assessment come under assessment and analysis of software. In this regard, Fu et al. [43] proposed a regression-based ML technique in order to estimate software energy consumption by analyzing software performance features. The verification and validation category specifically addresses prediction and verification of software reliability through ML, e.g., Tamura et al. [44] proposed a DL-based technique to select the most suitable software reliability model for the development project. Testing effort estimation refers to the amount of testing effort required in order to test a software system using ML techniques, e.g., Silva et al. [6] evaluated various ML tools in order to estimate the execution times for running functional test cases.

Applications of ML aiming at software maintenance

The software maintenance stage has been found as the least focused stage for researchers in this domain. In this category, the research is more inclined towards cost/effort estimation than the rest of the maintenance tasks. We found articles focusing on software maintainability prediction, which refers to the proposed ML techniques in order to assist the prediction of maintainability metrics appropriate for specific software projects [45]. Software aging detection refers to the use of ML in order to detect software maturity and its aging in terms of resource depletion such as memory leaks, high CPU usage, and overtime. In this regard, Andrzejak et al. [46] investigated the feasibility of ML techniques for classification in detecting early performance degradation due to software image aging. The maintenance effort estimation class aims at estimating the amount of effort required for the maintenance of a software system using ML, e.g., Chandra et al. [47] used an SVM-based regression model in order to forecast maintenance effort with univariate and multivariate approaches.

Effect and significance of applying ML at each SDLC stage

ML aims to automate and support the SE activities, which are considered to be performed intensively by humans. ML allows systems to perform human-centric activities at a much larger scale [48]. In fact, an empirical study [49] has been conducted to understand whether software engineers can utilize ML techniques for the improvement of their SE process and whether solutions proposed by engineers still outperform ML techniques. However, the need for ML techniques is still pertinent due to their ability to outperform in most SE activities. We highlight some of these activities with respect to SDLC stages which are as follows:

In the requirements stage, writing requirements specifications is highly deemed to be a human-centric task. Prior work by Pandita et al. [50] and Jahan et al. [51] have inferred the most probable specifications and identified its unexpected behaviors from various artifacts by employing ML techniques, respectively. Ferrari et al. [52] identified ambiguous requirements among different domains using ML. In the architecture and design stage, predicting design patterns is an important reverse engineering activity to improve software integrity. However, it often suffers from false positives and negatives [53]. As the number of patterns is increasing rapidly due to their variations, the process of recognizing these patterns can be effectively learned using ML [53]. In the implementation stage, detecting code smells in a large codebase can be extremely difficult for a human as opposed to a machine, thus ML techniques can greatly reduce this effort of detecting code smells or technical debt [30, 31]. In quality assurance, there is a need to ensure that the system remains error-free or to be able to timely identify the cause of failure. ML techniques employed in literature for this purpose proved to be promising in detecting software faults [34, 35, 36]. Test generation is also considered to be a task that requires human intelligence. Zhang et al. [54] have employed ML to automatically generate test data in order to improve return on investment. In software maintenance, Malgonde et al. [55] have shown ML techniques perform significantly better at predicting the effort as compared to the team estimates (human-centric).

Despite the intriguing tendency of full automation, complete automation could often result in a potentially fallible system, therefore, practitioners are encouraged to employ ML techniques with humans in the loop wherever there is a presence of criticality [1, 49]. In addition, there is a significant lack of studies showing the cost-benefit analysis of their proposed ML techniques, which would be vital for ML-based approaches to be feasible for adaptation in the industry.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

TABLE 1: Classification by articles

SDLC Stages	Applications of ML for SE	Articles
All Stages	N/A	[1, 11, 14, 48, 49, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69]
Requirements	Requirements Modeling and Analysis	[21, 22, 51, 70, 71, 72, 73, 74, 75]
	Requirements Selection/Prioritization/Classification	[23, 24, 76, 77, 78, 79, 80, 81]
	Requirements Traceability	[3, 25, 82, 83, 84, 85, 86]
Architecture and Design	Design Modeling	[9, 26, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102]
	Design Pattern Prediction	[27, 53, 103, 104, 105, 106]
	Development Effort Estimation	[4, 5, 28, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123]
Implementation	Code Clone/Localization/Refactoring/Labeling	[29, 124, 125, 126, 127, 128, 129, 130, 131, 132]
	Code/Bad smell detection	[30, 31, 133, 134]
	Code Inspection/Analysis	[32, 135, 136, 137, 138, 139, 140, 141, 142]
	Code/Program Similarity	[33, 143, 144, 145, 146, 147]
Quality Assurance and Analytic	Fault/Bug/Defect Prediction	[7, 34, 35, 36, 37, 38, 39, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203]
	Test Case/Data/Oracle Generation	[40, 54, 204, 205, 206, 207, 208, 209]
	Test Case Selection/Prioritization/Classification	[41, 210, 211, 212, 213]
	Vulnerability/Anomaly/Malware Discovery/Analysis	[42, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232]
	Software Analysis	[43, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243]
	Technique Assessment	[244, 245, 246, 247, 248]
	Software Process Assessment	[249, 250, 251]
	Verification and Validation	[44, 246, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265]
	Testing Effort Estimation	[6, 266, 267, 268]
Maintenance	Software Maintainability Prediction	[45, 269, 270, 271]
	Software Aging Detection	[46, 272, 273, 274, 275]
	Maintenance Effort Estimation	[47, 276]

C. ML TYPE AND TECHNIQUES (Q1.3)

ML types

By the type of ML, we mean how the models have been trained, i.e., supervised, semi-supervised, unsupervised, reinforcement, or analytical learning. Supervised learning is based on a training set and a test set taken from the dataset. The model training is done by taking multiple labeled samples from the train set. After the model is trained, its performance is evaluated using the test set. In semi-supervised learning, both labeled and unlabelled data are employed in order to train the model. The dataset is divided into unsupervised clusters as such. Then, the class information is obtained by learning the clustering outcomes [216]. Unsupervised learning requires no training dataset. For instance, in unsupervised learning for fault detection, software instances are usually grouped into clusters and each cluster is labeled as "Buggy" or "Correct". However, each cluster needs to be labeled manually by the individuals with expertise [198]. Reinforcement learning refers to unsupervised goal-oriented learning performed by an agent directly interacting with the environment. Analytical learning is aimed at generating solutions based on background knowledge and improving inference iteratively [253].

As shown in Fig. 4, 193 out of 263 (73%) articles employed supervised learning, 15 out of 263 (6%) articles employed unsupervised learning, 6 out of 263 (2%) articles employed semi-supervised learning, 4 out of 263 (2%) articles addressed reinforcement learning, and 1 out of 263 (0.4%) focused on analytical (inference-based) learning. The rest of the articles 44 out of 263 (17%) did not explicitly report the employed ML type.

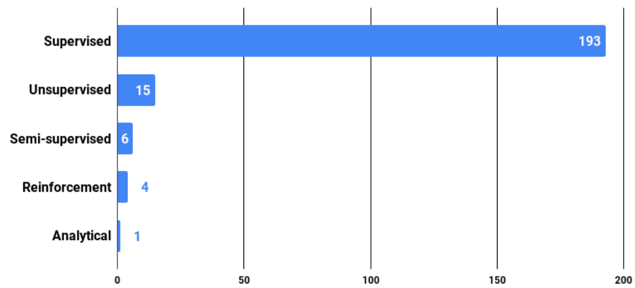


FIGURE 4: Articles by ML type

ML techniques

ML techniques are the algorithms used for classification, regression, or clustering problems, e.g., SVM, RF, or ANN. The employed techniques in the selected pool of articles are shown in Fig. 5. The topmost commonly used techniques are ANN, RF, DT, and NB, respectively. While 51 out of 263 (19%) articles employed ANN, 45 out of 263 (17%) articles have used RF and SVM, and 40 out of 263 (15%) articles used DT and NB for model training.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

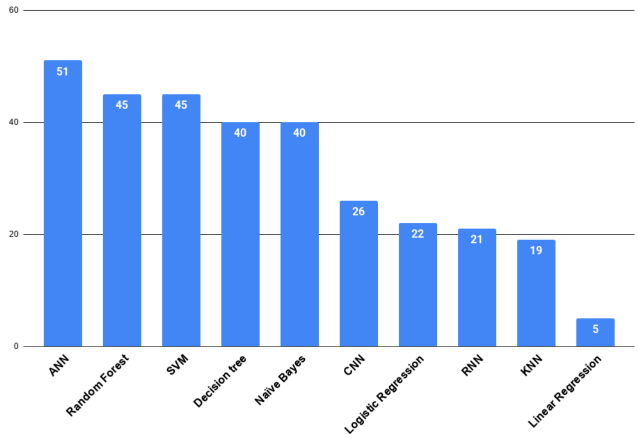


FIGURE 5: Articles by techniques

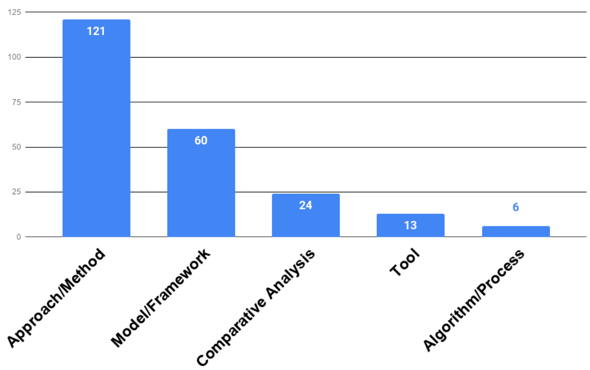


FIGURE 6: Articles by contribution facet

D. CONTRIBUTION FACET OF THE ARTICLES (Q2.1)

The contribution facet addresses the novel propositions of the articles. It is derived by analyzing the contribution of the articles, which represents the current state-of-the-art and enables researchers and industrial practitioners to get an overview of the existing tools and techniques in the literature. As shown in Fig. 6, 121 out of 263 (46%) articles focused on approaches/methods, followed by 60 (23%) articles proposing models/frameworks, 24 (9%) articles focusing on comparative analysis of existing techniques, 13 (5%) articles focusing on tools, and 6 (2%) articles focusing on algorithms/processes. The rest of the articles – 39 out of 263 (15%) – reported no new propositions. These articles were either investigating existing approaches, discussing opinions, or reporting their experiences.

Table 2 shows the names of the propositions along with the contribution facet and references of the articles. Interestingly, only 25 out of 263 (9%) articles have explicitly named their propositions.

TABLE 2: Named propositions in the articles

Sr. no.	Name	Contribution Facet	Article
1	Trace-by-Classification	Approach	[86]
2	DeepSim	Approach	[147]
3	CDGDroid	Approach	[200]
4	SLDeep	Approach	[175]
5	REMI	Approach	[168]
6	Feature Maps	Algorithm	[105]
7	ProbPoly	Framework	[75]
8	ExploitMeter	Framework	[231]
9	DLFuzz	Framework	[256]
10	DARVIZ	Framework	[98]
11	Seml	Framework	[172]
12	CroLSim	Model	[85]
13	DeepGauge	Process	[206]
14	WIRECAML	Tool	[224]
15	SOA-based integrated software	Tool	[151]
16	Modelware	Tool	[102]
17	Featuretools	Tool	[67]
18	Code-Buff	Tool	[128]
19	AppFlow	Tool	[211]
20	CloneCognition	Tool	[126]
21	ArchLearner	Tool	[95]
22	SZZ Unleashed	Tool	[149]
23	Auto-sklearn	Tool	[194]
24	RIVER	Tool	[209]
25	InSet	Tool	[91]

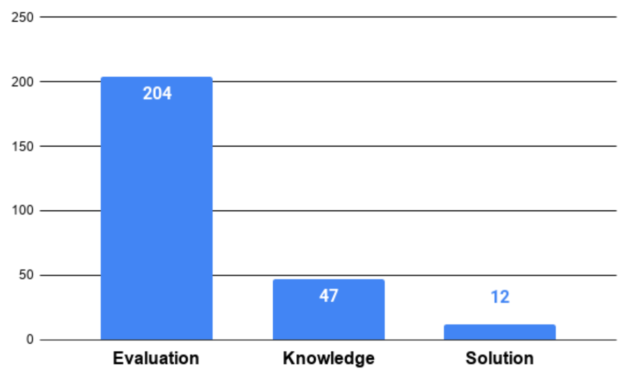


FIGURE 7: Articles by research facet

E. RESEARCH FACET OF THE ARTICLES (Q2.2)

The research facet describes the nature of articles in terms of their purpose of conducting the research, such as evaluations (articles employing empirical methods such as controlled experiments or case studies), solutions (articles proposing solutions to underlying problems without empirical evidence), and knowledge (articles expressing experiences and opinions). Fig. 7 shows the articles by their research facet. 204 out of 263 (78%) articles have contributions with empirically evaluated propositions, whereas 47 out of 263 (18%) articles are knowledge-based, and 12 out of 263 (5%) articles have proposed solutions without any empirical evaluation.

The evaluation facet, in turn, represents the type of evaluation that has been performed in the articles in order to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

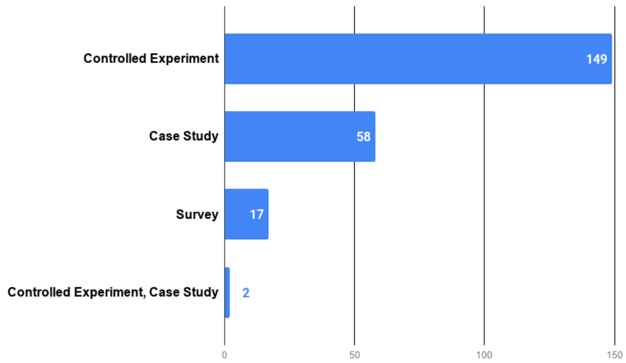


FIGURE 8: Articles by evaluation facet

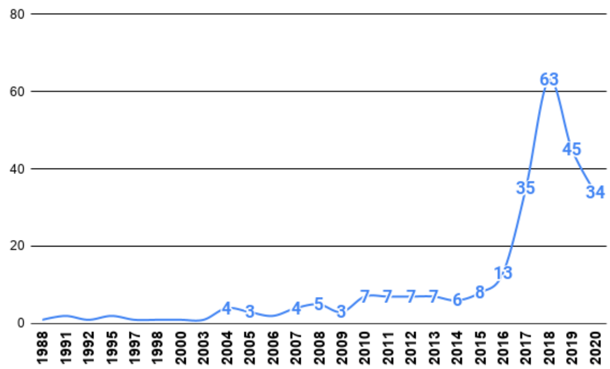


FIGURE 9: Articles by year

FIGURE 10: Articles by venues (Top 5)

evaluate their propositions. The articles by the evaluation facet are shown in Fig. 8. Controlled experiments have been performed in 148 out of 204 (73%) articles followed by case studies in 58 out of 204 (28%) articles and surveys in 16 out of 204 (8%) articles. 2 out of 204 (1%) articles have employed both a controlled experiment and a case study for an empirical evaluation; whereas, rest of the articles did not use any empirical method for evaluation purposes. Moreover, we found no article employing ethnography or action research as empirical methods for evaluation. Among the articles those performed control experiments, 78 articles proposed approaches/techniques/methods, and 41 articles proposed models/frameworks. While 15 articles focused on comparative analysis, 8 articles proposed tools, and 4 articles introduced new algorithms/processes.

F. DATASETS (Q2.3)

We further explored the datasets that have been used in most of the articles in order to evaluate their proposed approaches or comparative studies. Evidently, the majority of articles employed datasets obtained from PROMISE[10] repository followed by repositories made publicly available by NASA[11], StackOverflow[12], Github[13], and JAVA projects.

G. TRENDS IN TERMS OF YEAR (Q3.1)

This refers to the trends in terms of publication years of articles. It shows the evolution of the adoption of ML for SE. As shown in Fig. 9, the use of ML for SE is consistently growing over the passage of time. One can also observe an exponential growth in this trend from 2016 - 2018, where 2018 proved to be the highest publication year with 63 (24%) publications. In 2019 and 2020, we recorded relatively fewer publications: 45 out of 263 (17%) and 34 out of 263 (13%), respectively. There could be two plausible reasons for that. Either some databases are not updated completely (as this study was conducted in Q4 of 2020) or like any hype cycle, the peak of inflated expectations regarding ML for SE was reached in 2018 and now the trend is slowly going towards the trough of disillusionment. We believe the latter is the case here.

H. VENUES WITH HIGHEST PUBLICATIONS (Q3.2)

Fig. 10 shows the top 5 venues where most researchers of the domain tend to publish. International Conference on Software Engineering (ICSE) is leading by 11 out of 263 (4%) and the second most publishing venue is Transactions on Software Engineering (TSE) journal with 10 out of 263 (4%). They are followed by International Workshop on Machine Learning and Software Engineering, which featured 5 out of 263 (2%) articles, European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), which also featured 5 out of 263 (2%) articles, and International Conference on Cloud Computing, Data Science & Engineering (Confluence), which featured 3 out of 263 (1%) articles. Moreover, Fig. 11 shows the overall distribution of articles with respect to publishing venues. Here one can observe that 155 out of 263 (59%) articles have been published in conferences, 51 out of 263 (19) articles have been published in journals, 26 out 263 (10%) articles have been published in workshops, and 18 out of 263 (7%) articles have been published in symposia.

VI. ANALYSIS AND DISCUSSION

This section relates to the fourth goal of this study (G4) and deals with implications and analysis of the aforementioned articles.

[10]http://promise.site.uottawa.ca/SERepository/datasets-page.html
[11]https://data.nasa.gov/
[12]https://archive.org/details/stackexchange
[13]https://ghtorrent.org/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

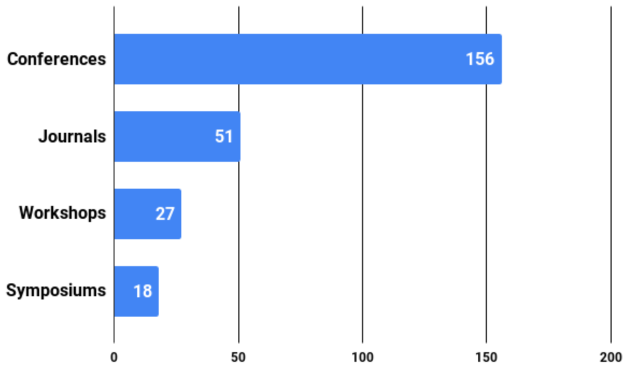


FIGURE 11: Articles by publishing venues

A. RELATION OF SDLC STAGES WITH RESEARCH AND CONTRIBUTION FACETS

Fig. 12 shows the relationship of the contribution and research facets explored in this study with the SDLC stages. Moreover, the figure provides a bird's-eye view of the current studies falling into the respective SDLC stages along with their contribution type and research purpose. For instance, 55 articles belonging to the quality assurance stage have proposed a new approach or method as their primary contribution, and the contributions of 107 articles at this stage were evaluated empirically. In addition, we can observe that no tool has been proposed for the requirements and maintenance stage indicating less interest of researchers in prototyping their proposition.

B. RELATION OF SDLC STAGES WITH ML

As shown in Fig. 3, 52% of the articles were dedicated to the quality assurance and analytic stage, which shows that software quality[14] is the prime focus for the researchers of this domain. Indeed, quality assurance, along with requirements and design, are human-centric stages of the SDLC and the high number of articles in these areas highlight the fact that ML is able to offer help here. As shown in Tab. 1, we further observed that fault/bug/defect prediction has been the major focus of researchers within quality assurance. Certainly, the nature of ML types and techniques is more supportive for this kind of activities, but we hope that in the future other SE activities may also similarly benefit from ML. This is particularly valid for the maintenance stage, which has been the least interesting area for the application of ML. We encourage researchers to investigate how ML can be used to automate certain tasks in this area. We further encourage researchers to adopt combinations of ML techniques and use diverse datasets from different sources in order to train the ML models so that the applicability of the techniques can be generalized as also observed in [99, 115, 188, 237].

[14]Our criteria for software quality assurance is shown in Tab. 1

C. RELATION OF SDLC STAGES WITH ML TYPES

As shown in Fig. 4, a vast majority of articles falling into requirements, architecture and design, and implementation categories are addressing the problems using supervised learning. For instance, [25] used supervised DL technique to identify trace links and predict associations within artifacts. A similar supervised learning technique has been proposed in [86] order to generate trace links from commonly occurring artifacts in the project. The reason supervised learning is mostly employed in the articles could be that supervised learning models are comparatively simple and produce results with high confidence and accuracy. We also noticed that only 4 out of 263 (2%) articles [3, 61, 225, 238] used reinforcement learning. This implies a little interest of researchers in the applications of reinforcement learning to SE. Reinforcement learning has proven to be beneficial in solving complex problems especially in healthcare, business, and robotics [277]. Thus, we believe it would be an interesting area to explore in terms of facilitating SE. For instance, software simulations can be deemed as an environment in which the RL agent can interact and reach various goal-oriented outcomes [278].

D. RELATION OF SDLC STAGES WITH ML TOOLS

As shown in Fig. 6, only 13 articles proposed a new tool to facilitate SDLC stages. As further can be observed in Fig. 12, 6 out of those 13 tools have been proposed for quality assurance purposes, e.g., the tool named "Appflow", which is proposed by Hu et al. [211] and predicts reusable UI test cases using neural networks. Tools are indeed a valuable contribution when it comes to the practicality and applicability of the proposed approach. In the future, more tools are desirable that are targeting other SDLC stages.

E. RELATION OF SDLC STAGES WITH ML TECHNIQUES

Although all ML techniques have certain pros and cons, the selection of the most suitable technique depends on the type of dataset being constructed or employed and what problem is being addressed. The SDLC stage-wise breakdown of ML techniques is shown in Fig. 13. As anticipated, mostly ML techniques were employed to solve problems related to the quality assurance and analytic stage. ANN was the most commonly used technique here (30 articles), followed by SVM (28 articles) and RF (24 articles), respectively. NB was next in line with 21 articles. ANN, which was used in 30 articles in the quality assurance stage was also a subject of interest for the researchers working in the architecture and design stage (15 articles).

As shown in Fig. 13, ANN is the most widely employed ML technique for SDLC stages in general due to its simplicity and strong classification and regression capabilities. CNN is mostly used in supervised learning problems, whereas RNN has been used to address both supervised and unsupervised learning problems. In traditional ML techniques, KNN, k-means clustering, NB, and SVM are mostly employed to address semi-supervised and unsupervised learning

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

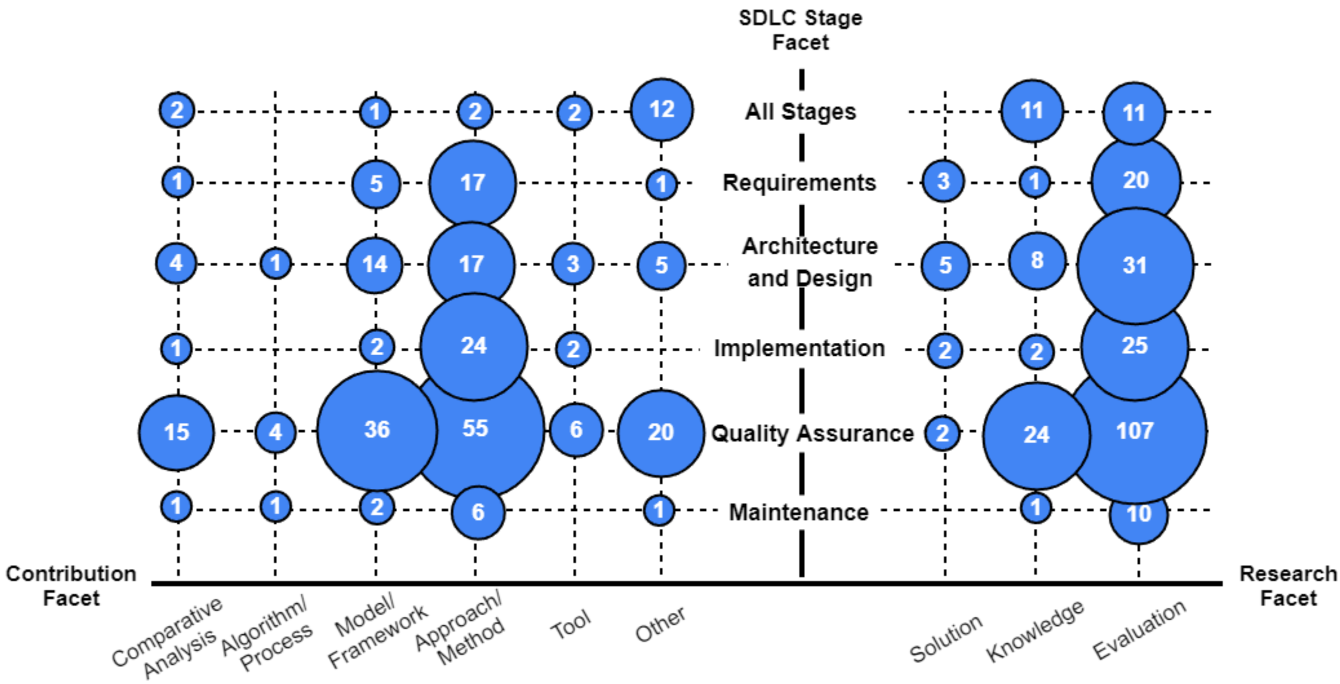


FIGURE 12: Relationship of contribution/research facets with the SDLC stage facet

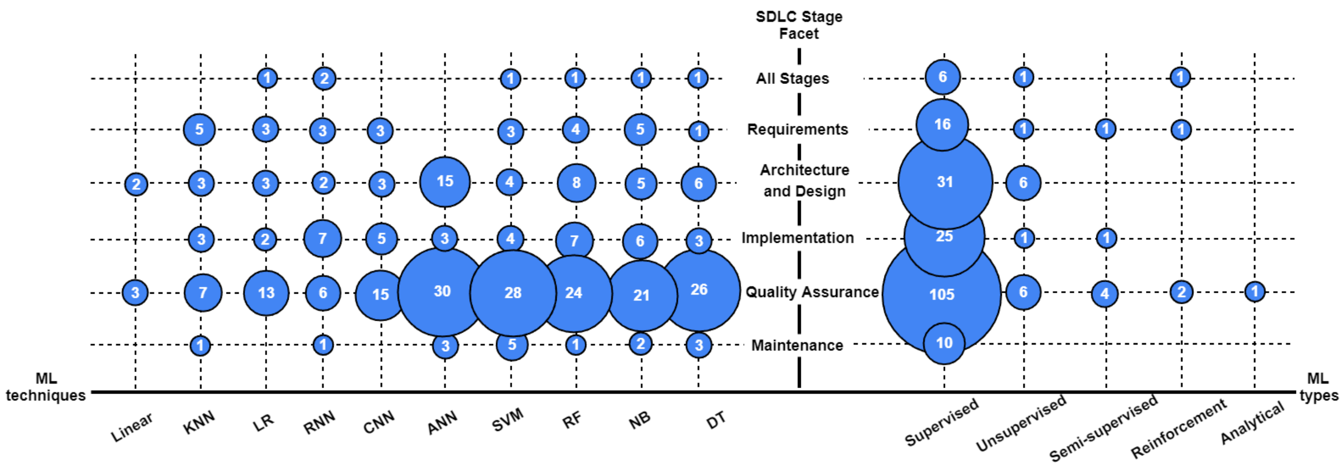


FIGURE 13: ML techniques usage in SDLC

problems. In the case of reinforcement learning, Q-learning technique and its variants have been mostly employed in the literature.

When it comes to neural networks-based techniques, our findings show that simple neural networks, e.g., ANN (51 out of 263 (19%)) and shallow neural networks, e.g., CNN and RNN (containing one or more hidden layers) (combined 47 out of 263 (18%)) are the most widely used ML techniques in SE. Neural networks are mostly employed for software architecture and design, and software implementation. Apart from neural networks, traditional ML techniques such as Boosting, NB, and case-based ranking, were popular in requirements engineering, particularly. The SVM technique has been mostly employed for the software maintenance stage. Apart from the ML techniques, most of the articles addressed problems related to supervised learning indicating classification as a major area of interest. While unsupervised and semi-supervised learning has been less employed in the area. The wide adoption of neural networks-based techniques in articles indicate their suitability and potential for achieving good results in this area. Mainly due to the reason that a neural network-based model is capable of learning from high dimensional large scale input data and an appropriate selection of cost function leads to the development of a more robust model. Moreover, neural network-based techniques are highly customizable and can be applied to various learning problems, such as supervised, unsupervised, or reinforcement, which make them highly flexible in terms of applicability.

Table 3 contains the complete list of articles (263) used in this paper showing ML techniques employed in those articles with respect to SDLC stages along with their contribution facet.

VII. CHALLENGES, LIMITATIONS AND FUTURE RESEARCH DIRECTIONS

This section also relates to the fourth goal of this study (G4) and deals with challenges, limitations, and future research directions in this field.

A. CHALLENGES

One of the major challenges in this domain, as also reported by other experts, e.g., [150, 157], is the uncertain and stochastic nature of the employed ML techniques, and the difference in the captured data and results, e.g., the difference in the DL model output values when executing it multiple times over the same input data. The approaches need to be reproducible and rigorous in order to build high confidence for their application.

The availability of sufficiently labeled and structured datasets is also a challenge as also reported by other researchers, e.g., [32, 170, 184]. However, this can be overcome rather easily as more and more researchers have started sharing their datasets publicly. An associated issue is the imbalanced sizes of software projects and datasets. Using new techniques for dataset balancing, such as SMOTE and ClassBalancer (both evaluated by Percorelli et al. [30]), or devising new ones is highly recommended in this context.

The ever-increasing software complexity is also one of the greater challenges for this domain. Meinke et al. [63] also attest to our observation and further suggest that the scalability problem should be given proper attention by researchers of this domain. We also invite researchers to conduct more studies investigating the impact of ML techniques on different software sizes.

B. LIMITATIONS

As observed in some studies, e.g., [140, 176], the lack of generalizability regarded as over-fitting problems is one of few major limiting factors, which decreases the accuracy of results. This leads to lesser results when ML models are applied to diverse cross-project datasets. Using standard data preprocessing techniques such as SMOTE, ClassBalancer, and Resample [30], and performing K-fold cross-validation or hold-out validation could reduce the problem of over-fitted and under-fitted models.

As observed in some studies, e.g., Ghaffarian et al. [219], the current state of evaluation of ML techniques, especially for software vulnerability testing is not well grounded. The dataset often lacks sufficient vulnerability types, which results in less generalizable outcomes. In order to improve results' precision, lesser false positives, and false negatives while maintaining recall can help produce meaningful results.

In a distributed software development environment, manual inspection/allocation of work items, excessive time consumption, potentially fallible outcomes, and lack of production-ready approaches are some of the limitations identified by Barcus et al. [279] and Achimugu et al. [280].

C. FUTURE RESEARCH DIRECTIONS

In order to facilitate requirements traceability, researchers have suggested that devising a feedback mechanism, such as adding user feedback during the model training process in order to improve feature selection and performance, can really help the cause of generalizability. One of such works is presented by Sultanov et al. [3], which provides a very good basis for further developments.

In order to improve prediction accuracy and better reliability of results, more experiments using larger numbers of datasets and software applications have also been suggested [99, 115, 188, 237].

Researchers in the articles have also suggested investigating further regarding the suitable metrics and loss functions employed in the evaluation of ML for SE-focused techniques, especially for multi-class classification problems [125].

Future research directions also include automata learning for emergent middle-wares and using ML to address complex system integration problems, especially in system of systems such as the internet of things. Moreover, researchers are encouraged to devise adaptable, easily integrable, and scalable solutions in the area.

VIII. THREATS TO VALIDITY

Similar to other secondary studies, this study is also prone to some validity threats. The threats and their mitigation strategies are described in this section.

A. EXTERNAL VALIDITY

The extraction of articles and choice of repositories constitute a threat to internal validity. In order to minimize the former, we adopted the PICO (Population, Intervention, Comparison, Outcomes) criteria suggested by Petersen et al. [19] to formulate the search terms. The selected terms unequivocally represent the goals of our work. An associated issue corresponds to the frequently used specific ML terms. Although the query used did not explicitly include ML terms, such as classification, regression, SVM, ANN, inductive logic, Bayesian network, or deep belief network, this would not affect the analysis much because such information is usually available in abstracts, hence accessible. In order to minimize the latter, we used five digital libraries as the primary source for this research. All selected digital libraries are well known in the computer science discipline for including the most relevant results [281]. Additionally, according to Wohlin et al. [282], having a larger set of papers is not necessarily better for such reviews. The important thing is that the found studies are a good representation of the population, which we ensured in this study by adopting a rigorous paper selection process.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

TABLE 3: Articles by ML techniques

	Refs.	SDLC Stages	Contribution Facet	ML techniques
1	[58]	All Stages	Approach/Technique/Method	RNN
2	[63]	All Stages	Other	Other
3	[64]	All Stages	Other	Other
4	[56]	All Stages	Comparative Analysis	Other
5	[57]	All Stages	Other	Other
6	[69]	All Stages	Other	RNN, RBM
7	[14]	All Stages	Model/Framework	Other
8	[67]	All Stages	Tool	RF
9	[49]	All Stages	Other	Other
10	[62]	All Stages	Tool	NLP
11	[61]	All Stages	Other	DT
12	[48]	All Stages	Other	Other
13	[65]	All Stages	Other	Other
14	[1]	All Stages	Other	Other
15	[60]	All Stages	Other	Other
16	[68]	All Stages	Comparative Analysis	LR, SVM, NB
17	[66]	All Stages	Approach/Technique/Method	LSTM
18	[11]	All Stages	Other	Other
19	[59]	All Stages	Other	Other
20	[21]	Requirements	Approach/Technique/Method	NB, KNN, RF
21	[70]	Requirements	Approach/Technique/Method	SVM, SMO, NB
22	[82]	Requirements	Approach/Technique/Method	PN
23	[75]	Requirements	Model/Framework	ProbPoly
24	[71]	Requirements	Approach/Technique/Method	Text2Model
25	[84]	Requirements	Approach/Technique/Method	RF
26	[72]	Requirements	Approach/Technique/Method	Other
27	[22]	Requirements	Approach/Technique/Method	NB, RF, LR, SGD, DT
28	[23]	Requirements	Approach/Technique/Method	Boosting
29	[81]	Requirements	Approach/Technique/Method	NSGA-II algorithm
30	[24]	Requirements	Model/Framework	CNN
31	[73]	Requirements	Approach/Technique/Method	FL
32	[76]	Requirements	Approach/Technique/Method	LP, SMO, NB, KNN
33	[86]	Requirements	Approach/Technique/Method	J48, FSS, CFS
34	[25]	Requirements	Model/Framework	RNN
35	[85]	Requirements	Model/Framework	KNN
36	[79]	Requirements	Other	Other
37	[3]	Requirements	Approach/Technique/Method	RL
38	[80]	Requirements	Model/Framework	LSTM, GRU, CNN
39	[51]	Requirements	Approach/Technique/Method	LSTM
40	[74]	Requirements	Approach/Technique/Method	Spacy NLP model
41	[77]	Requirements	Comparative Analysis	LR, SVM, MNB, kNN
42	[83]	Requirements	Approach/Technique/Method	RNN
43	[78]	Requirements	Approach/Technique/Method	RNN, CNN, SVM, KNN, LR, NB, RF
44	[117]	Architecture and Design	Approach/Technique/Method	KNN, CTM, MARS, CART
45	[102]	Architecture and Design	Tool	Modelware
46	[98]	Architecture and Design	Model/Framework	DARVIZ
47	[96]	Architecture and Design	Approach/Technique/Method	RF
48	[97]	Architecture and Design	Model/Framework	Other
49	[87]	Architecture and Design	Model/Framework	CNN
50	[27]	Architecture and Design	Approach/Technique/Method	RNN, GAN
51	[89]	Architecture and Design	Approach/Technique/Method	SVM
52	[122]	Architecture and Design	Other	CBR, ANN, DT, BN, SVR, GA, AR
53	[112]	Architecture and Design	Comparative Analysis	CBR, ANN, CART
54	[101]	Architecture and Design	Approach/Technique/Method	NB, SMO, RF
55	[92]	Architecture and Design	Model/Framework	Restricted Boltzmann Machine
56	[99]	Architecture and Design	Model/Framework	GRBF
57	[53]	Architecture and Design	Approach/Technique/Method	RNN, DT
58	[111]	Architecture and Design	Model/Framework	ANN

TABLE 3: Articles by ML techniques

	Refs.	SDLC Stages	Contribution Facet	ML techniques
59	[93]	Architecture and Design	Model/Framework	Other
60	[113]	Architecture and Design	Other	NN
61	[110]	Architecture and Design	Comparative Analysis	SVR
62	[115]	Architecture and Design	Approach/Technique/Method	NB
63	[90]	Architecture and Design	Model/Framework	NN
64	[116]	Architecture and Design	Model/Framework	GP, LMS, LR, MP, RBFN, SMO, AR, BAG, CR, DT, MSR, ZR, DS, RT
65	[94]	Architecture and Design	Model/Framework	CNN
66	[120]	Architecture and Design	Approach/Technique/Method	DT, NN
67	[119]	Architecture and Design	Comparative Analysis	GP, NN
68	[108]	Architecture and Design	Model/Framework	RBF, SVR, PCA
69	[4]	Architecture and Design	Approach/Technique/Method	RT, MLP, SVR
70	[5]	Architecture and Design	Comparative Analysis	ANN, RI, FL, CART, CBR
71	[26]	Architecture and Design	Other	NN
72	[114]	Architecture and Design	Approach/Technique/Method	ANN, SVM
73	[118]	Architecture and Design	Other	ANN, GA
74	[28]	Architecture and Design	Approach/Technique/Method	ANN
75	[121]	Architecture and Design	Other	ANN, GA
76	[100]	Architecture and Design	Approach/Technique/Method	Other
77	[109]	Architecture and Design	Approach/Technique/Method	NB, LR, RF
78	[105]	Architecture and Design	Algorithm/Process	CNN, RF
79	[123]	Architecture and Design	Model/Framework	DNN
80	[95]	Architecture and Design	Tool	LSTM
81	[106]	Architecture and Design	Approach/Technique/Method	SBL
82	[107]	Architecture and Design	Model/Framework	RF
83	[88]	Architecture and Design	Approach/Technique/Method	LR, NB, DT, RF, KNN
84	[9]	Architecture and Design	Model/Framework	k-means clustering
85	[104]	Architecture and Design	Approach/Technique/Method	Other
86	[103]	Architecture and Design	Approach/Technique/Method	ANN, SVM, RF
87	[91]	Architecture and Design	Tool	NB, NN, KNN, RF, SVM, DT
88	[131]	Implementation	Approach/Technique/Method	RNN
89	[29]	Implementation	Approach/Technique/Method	CNN
90	[128]	Implementation	Tool	KNN
91	[132]	Implementation	Approach/Technique/Method	Fica
92	[33]	Implementation	Approach/Technique/Method	NN, RF
93	[125]	Implementation	Model/Framework	CNN
94	[145]	Implementation	Approach/Technique/Method	RNN
95	[127]	Implementation	Approach/Technique/Method	CNN
96	[147]	Implementation	Approach/Technique/Method	DNN
97	[133]	Implementation	Approach/Technique/Method	DT
98	[138]	Implementation	Approach/Technique/Method	OGUST
99	[32]	Implementation	Approach/Technique/Method	NB, DT, SVM
100	[140]	Implementation	Approach/Technique/Method	RF, NB, KNN
101	[31]	Implementation	Approach/Technique/Method	RF, NB, LR
102	[137]	Implementation	Comparative Analysis	NB
103	[146]	Implementation	Model/Framework	RNN
104	[143]	Implementation	Approach/Technique/Method	CNN, RNN, LSTM
105	[135]	Implementation	Approach/Technique/Method	SVM
106	[126]	Implementation	Tool	ANN
107	[30]	Implementation	Approach/Technique/Method	Other
108	[124]	Implementation	Approach/Technique/Method	LSTM
109	[141]	Implementation	Approach/Technique/Method	RF, J48, SMO, MLP, NB, LogitBoost, AdaBoost
110	[134]	Implementation	Approach/Technique/Method	DT, GBT, SVM, RF, ANN
111	[142]	Implementation	Approach/Technique/Method	RNN
112	[129]	Implementation	Approach/Technique/Method	RNN
113	[130]	Implementation	Approach/Technique/Method	KNN, RF
114	[136]	Implementation	Approach/Technique/Method	DNN
115	[139]	Implementation	Approach/Technique/Method	NB, LR, SVM, RF, XGB, CNN

TABLE 3: Articles by ML techniques

	Refs.	SDLC Stages	Contribution Facet	ML techniques
116	[144]	Implementation	Approach/Technique/Method	RNN
117	[150]	Quality Assurance and Analytic	Approach/Technique/Method	SVM, DT
118	[267]	Quality Assurance and Analytic	Approach/Technique/Method	COBWEB/3
119	[224]	Quality Assurance and Analytic	Tool	DT, RF, LR, NB, TAN
120	[256]	Quality Assurance and Analytic	Model/Framework	CNN
121	[161]	Quality Assurance and Analytic	Approach/Technique/Method	OC-SVM
122	[219]	Quality Assurance and Analytic	Other	Other
123	[41]	Quality Assurance and Analytic	Approach/Technique/Method	KStar
124	[163]	Quality Assurance and Analytic	Approach/Technique/Method	NN
125	[266]	Quality Assurance and Analytic	Other	DT
126	[211]	Quality Assurance and Analytic	Tool	NN
127	[228]	Quality Assurance and Analytic	Model/Framework	RF, NB
128	[40]	Quality Assurance and Analytic	Approach/Technique/Method	AdaBoostM1, JRIP 3
129	[207]	Quality Assurance and Analytic	Approach/Technique/Method	ANN, DT, KNN, NB, RF, SVM
130	[254]	Quality Assurance and Analytic	Model/Framework	RNN
131	[206]	Quality Assurance and Analytic	Algorithm/Process	DNN
132	[255]	Quality Assurance and Analytic	Approach/Technique/Method	LSTM
133	[157]	Quality Assurance and Analytic	Approach/Technique/Method	SVM, CNN
134	[265]	Quality Assurance and Analytic	Model/Framework	RNN
135	[210]	Quality Assurance and Analytic	Algorithm/Process	SVM
136	[54]	Quality Assurance and Analytic	Model/Framework	GA
137	[253]	Quality Assurance and Analytic	Model/Framework	EDAs
138	[262]	Quality Assurance and Analytic	Model/Framework	MBR, BBN
139	[205]	Quality Assurance and Analytic	Other	Other
140	[212]	Quality Assurance and Analytic	Approach/Technique/Method	K-means clustering, Expectation–Maximization, Incremental Conceptual Clustering
141	[252]	Quality Assurance and Analytic	Other	Other
142	[246]	Quality Assurance and Analytic	Comparative Analysis	DT, BNN, RBNN, SVM
143	[44]	Quality Assurance and Analytic	Approach/Technique/Method	NN
144	[223]	Quality Assurance and Analytic	Model/Framework	Other
145	[257]	Quality Assurance and Analytic	Approach/Technique/Method	SVM
146	[233]	Quality Assurance and Analytic	Other	Other
147	[170]	Quality Assurance and Analytic	Approach/Technique/Method	NB, DT, SVM
148	[182]	Quality Assurance and Analytic	Comparative Analysis	ANN, Particle Swarm Optimization, DT, NB
149	[151]	Quality Assurance and Analytic	Tool	SVM, DT
150	[268]	Quality Assurance and Analytic	Model/Framework	ANN
151	[264]	Quality Assurance and Analytic	Other	STP, LTP
152	[220]	Quality Assurance and Analytic	Approach/Technique/Method	DT, RF, KNN , SVM
153	[213]	Quality Assurance and Analytic	Approach/Technique/Method	NB
154	[204]	Quality Assurance and Analytic	Algorithm/Process	GA
155	[200]	Quality Assurance and Analytic	Approach/Technique/Method	CNN
156	[208]	Quality Assurance and Analytic	Approach/Technique/Method	Evolutionary Algorithm
157	[158]	Quality Assurance and Analytic	Approach/Technique/Method	DT
158	[188]	Quality Assurance and Analytic	Comparative Analysis	LR, ANN
159	[173]	Quality Assurance and Analytic	Comparative Analysis	NB
160	[176]	Quality Assurance and Analytic	Approach/Technique/Method	LR
161	[187]	Quality Assurance and Analytic	Approach/Technique/Method	ANN
162	[258]	Quality Assurance and Analytic	Other	NN
163	[229]	Quality Assurance and Analytic	Model/Framework	NN
164	[218]	Quality Assurance and Analytic	Model/Framework	RF, PART
165	[245]	Quality Assurance and Analytic	Comparative Analysis	NN, NB
166	[261]	Quality Assurance and Analytic	Other	Other
167	[263]	Quality Assurance and Analytic	Model/Framework	ANN
168	[167]	Quality Assurance and Analytic	Other	SVM, RF
169	[244]	Quality Assurance and Analytic	Approach/Technique/Method	SBL
170	[184]	Quality Assurance and Analytic	Model/Framework	DT, SVM, ANN

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

TABLE 3: Articles by ML techniques

	Refs.	SDLC Stages	Contribution Facet	ML techniques
171	[153]	Quality Assurance and Analytic	Comparative Analysis	LM, MAE, LR, PR, SVR, NNC, SVLR, NND, LoR, NB, IBL, JDT, 1R
172	[152]	Quality Assurance and Analytic	Model/Framework	DT, MLP, RBF
173	[6]	Quality Assurance and Analytic	Comparative Analysis	SVR, ANN
174	[246]	Quality Assurance and Analytic	Comparative Analysis	DT, SVM
175	[249]	Quality Assurance and Analytic	Approach/Technique/Method	C4.5, NB, SVM
176	[214]	Quality Assurance and Analytic	Comparative Analysis	DT, NB, SVM-C, KNN, RF
177	[247]	Quality Assurance and Analytic	Approach/Technique/Method	SVM
178	[160]	Quality Assurance and Analytic	Other	Other
179	[251]	Quality Assurance and Analytic	Approach/Technique/Method	NN
180	[250]	Quality Assurance and Analytic	Model/Framework	NN
181	[216]	Quality Assurance and Analytic	Model/Framework	SVM
182	[186]	Quality Assurance and Analytic	Other	DT, CBR, ANN, SVM
183	[217]	Quality Assurance and Analytic	Approach/Technique/Method	Recurrent Neural Network, LSTM
184	[248]	Quality Assurance and Analytic	Other	NN
185	[178]	Quality Assurance and Analytic	Model/Framework	CNN
186	[177]	Quality Assurance and Analytic	Approach/Technique/Method	CNN
187	[165]	Quality Assurance and Analytic	Approach/Technique/Method	SVM, RNN
188	[171]	Quality Assurance and Analytic	Model/Framework	CNN
189	[231]	Quality Assurance and Analytic	Model/Framework	FL
190	[237]	Quality Assurance and Analytic	Model/Framework	SVM
191	[230]	Quality Assurance and Analytic	Approach/Technique/Method	CNN
192	[189]	Quality Assurance and Analytic	Comparative Analysis	MLP, RBF, CART, KNN
193	[221]	Quality Assurance and Analytic	Approach/Technique/Method	CNN
194	[174]	Quality Assurance and Analytic	Other	Single Layer Perceptron, Multi Layer Perceptron, LVQ, SOM, AIRS, CLONAL, Immune
195	[193]	Quality Assurance and Analytic	Other	RF, DT, SVM, NB, LR
196	[222]	Quality Assurance and Analytic	Approach/Technique/Method	LSTM, NB, RF
197	[197]	Quality Assurance and Analytic	Approach/Technique/Method	DBN
198	[162]	Quality Assurance and Analytic	Model/Framework	CNN
199	[38]	Quality Assurance and Analytic	Comparative Analysis	ANN, CNN, SOM, LVQ, LVQ
200	[43]	Quality Assurance and Analytic	Model/Framework	Linear Regression, Ridge, Lasso, Random Forest Regression
201	[259]	Quality Assurance and Analytic	Other	Other
202	[34]	Quality Assurance and Analytic	Approach/Technique/Method	DT, LR
203	[236]	Quality Assurance and Analytic	Approach/Technique/Method	SVM
204	[198]	Quality Assurance and Analytic	Approach/Technique/Method	RNN
205	[155]	Quality Assurance and Analytic	Comparative Analysis	DNN
206	[202]	Quality Assurance and Analytic	Model/Framework	RNN
207	[35]	Quality Assurance and Analytic	Comparative Analysis	LR, NB, DT, J48
208	[37]	Quality Assurance and Analytic	Approach/Technique/Method	DT, RF, NB, SVM, ANN
209	[234]	Quality Assurance and Analytic	Other	NN, RF, DT
210	[203]	Quality Assurance and Analytic	Model/Framework	SDNN
211	[164]	Quality Assurance and Analytic	Comparative Analysis	GMMs, ANN
212	[180]	Quality Assurance and Analytic	Other	CNN
213	[242]	Quality Assurance and Analytic	Approach/Technique/Method	DT, KNN, SVM, NB
214	[149]	Quality Assurance and Analytic	Tool	RF
215	[239]	Quality Assurance and Analytic	Approach/Technique/Method	SGD
216	[227]	Quality Assurance and Analytic	Approach/Technique/Method	LSTM
217	[175]	Quality Assurance and Analytic	Approach/Technique/Method	LSTM
218	[39]	Quality Assurance and Analytic	Other	Other
219	[192]	Quality Assurance and Analytic	Approach/Technique/Method	ANN
220	[226]	Quality Assurance and Analytic	Approach/Technique/Method	RF, NB, J48
221	[201]	Quality Assurance and Analytic	Model/Framework	LSTM
222	[241]	Quality Assurance and Analytic	Model/Framework	LSTM
223	[36]	Quality Assurance and Analytic	Approach/Technique/Method	SVM
224	[191]	Quality Assurance and Analytic	Model/Framework	NaN
225	[185]	Quality Assurance and Analytic	Approach/Technique/Method	RF

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

TABLE 3: Articles by ML techniques

	Refs.	SDLC Stages	Contribution Facet	ML techniques
226	[156]	Quality Assurance and Analytic	Approach/Technique/Method	LSTM
227	[169]	Quality Assurance and Analytic	Approach/Technique/Method	GA
228	[42]	Quality Assurance and Analytic	Model/Framework	TF-IDF, IG, DNN
229	[215]	Quality Assurance and Analytic	Model/Framework	LSTM
230	[240]	Quality Assurance and Analytic	Model/Framework	RFCM, LR, CART, KNN
231	[225]	Quality Assurance and Analytic	Algorithm/Process	RL
232	[194]	Quality Assurance and Analytic	Tool	RF, DT
233	[179]	Quality Assurance and Analytic	Comparative Analysis	LR, KNN, DT, RF, SVM, NN
234	[7]	Quality Assurance and Analytic	Approach/Technique/Method	RF
235	[172]	Quality Assurance and Analytic	Model/Framework	LSTM
236	[209]	Quality Assurance and Analytic	Tool	RNN
237	[238]	Quality Assurance and Analytic	Model/Framework	RL
238	[159]	Quality Assurance and Analytic	Model/Framework	LR, DT, RF
239	[235]	Quality Assurance and Analytic	Other	CART, kNN, KRR, MR, RF, SVR
240	[243]	Quality Assurance and Analytic	Other	Other
241	[196]	Quality Assurance and Analytic	Approach/Technique/Method	NHANES dataset
242	[190]	Quality Assurance and Analytic	Approach/Technique/Method	SVM, RF, ANN, DT, NBG, LR, CNN
243	[195]	Quality Assurance and Analytic	Approach/Technique/Method	NN
244	[232]	Quality Assurance and Analytic	Approach/Technique/Method	ANN
245	[154]	Quality Assurance and Analytic	Approach/Technique/Method	CNN
246	[260]	Quality Assurance and Analytic	Model/Framework	SVM, ANN, NB
247	[183]	Quality Assurance and Analytic	Approach/Technique/Method	RF, NB, SVM, ANN
248	[166]	Quality Assurance and Analytic	Model/Framework	RF, NB, DT, LR, ANN
249	[181]	Quality Assurance and Analytic	Approach/Technique/Method	MLP, CNN
250	[148]	Quality Assurance and Analytic	Model/Framework	NB, LR, C4.5, SVM, RF, MLP
251	[199]	Quality Assurance and Analytic	Approach/Technique/Method	CNN
252	[168]	Quality Assurance and Analytic	Approach/Technique/Method	RF
253	[274]	Maintenance	Model/Framework	SVM
254	[46]	Maintenance	Approach/Technique/Method	NB, SMO
255	[269]	Maintenance	Algorithm/Process	FL
256	[47]	Maintenance	Approach/Technique/Method	SVM
257	[272]	Maintenance	Approach/Technique/Method	M5P
258	[275]	Maintenance	Approach/Technique/Method	DT, ANN, SVM
259	[273]	Maintenance	Comparative Analysis	DT, SVM, DBN
260	[45]	Maintenance	Model/Framework	LSTM
261	[271]	Maintenance	Approach/Technique/Method	RF, NB, KNN, SVM, ANN
262	[270]	Maintenance	Other	ANN, SVM/R, DT
263	[276]	Maintenance	Approach/Technique/Method	RNN

B. INTERNAL VALIDITY

Another threat is regarding the quality assessment of this study. As discussed by Petersen et al. [283] and Kitchenham et al. [284], quality assessment is not common in such kind of studies as their overall aim is to give a broad overview of the topic area. However, despite these observations, we have adopted a rigorous process for the inclusion and classification of papers, which ensures that only high-quality related papers are selected as primary studies.

C. CONCLUSION VALIDITY

Each article in this study was reviewed by the first author, which may lead to a threat to the reliability of the results. This threat was reduced by double-checking the article by the second, the third, and the fourth author. A random set of articles was distributed among the second and the third author. Their review results were then compared with the results of the first author. In case of a disagreement, the opinion of the fourth author was sought. Although this did not happen much.

IX. CONCLUSION

The conclusion of the study is manifold. We have provided an overview of the state-of-the-art in the area of machine learning for software engineering by evaluating carefully selected studies. We also proposed a classification scheme that highlights the overall applications of machine learning for software engineering in terms of SDLC stages. The classification shows the primary focus of researchers towards specific stages. This observation is one of the major contributions of this study. This study also reveals that the quality of primary studies in the domain of ML and SE is evidence-based with respect to the techniques being empirically evaluated by the researchers. We have also shown the relationship of SDLC stages with ML types, tools, and techniques. Although this research area is showing moderate growth in terms of the number of publications, further primary studies need to be conducted to emphasize other lesser explored SDLC stages such as maintenance. The challenges, limitations and future directions reported in this article should motivate and further guide researchers in the future. We believe this study provides the necessary impetus and further motivation to explore those SDLC stages, which have been given lesser attention to date with respect to the application of ML.

In the future, we intend to perform a more comprehensive study investigating the relationship between ML and SDLC stages. To this end, we intend to narrow down our search query by including ML terms such as classification, regression, SVM, ANN, inductive logic, Bayesian network, or deep belief network. We believe in this way, we can grasp a more focused view of the state-of-the-art.

REFERENCES

[1] Mark Harman. The Role of Artificial Intelligence in Software Engineering. In 2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE), pages 1–6. IEEE, 2012.

[2] Du. Zhang. Machine learning and software development. In 14th IEEE International Conference on Tools with Artificial Intelligence, number May, pages 87–119. IEEE, 2010.

[3] Hakim Sultanov and Jane Huffman Hayes. Application of reinforcement learning to requirements engineering: Requirements tracing. In 2013 21st IEEE International Requirements Engineering Conference, RE 2013 - Proceedings, pages 52–61. IEEE, 2013.

[4] Petrônio L Braga, Adriano L I Oliveira, and Silvio R L Meira. Software Effort Estimation using Machine Learning Techniques with Robust Confidence Intervals. In Seventh International Conference on Hybrid Intelligent Systems, pages 352–357, 2007.

[5] Juan J Cuadrado-gallego, Pablo Rodríguez-soria, and Borja Martín-herrera. Analogies and differences between Machine Learning and Expert based Software Project Effort Estimation. In 11th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, pages 269–276, 2010.

[6] Daniel G Silva, Mario Jino, and Bruno T De Abreu. Machine learning methods and asymmetric cost function to estimate execution effort of software testing. In Third International Conference on Software Testing, Verification and Validation, pages 275–284. IEEE, 2010.

[7] Hung Duy Tran, Le Thi My Hanh, and Nguyen Thanh Binh. Combining feature selection, feature learning and ensemble learning for software fault prediction. In Proceedings of 2019 11th International Conference on Knowledge and Systems Engineering, KSE 2019, pages 1–8. IEEE, 2019.

[8] Vinicius H. S. Durelli, Rafael S. Durelli, Simone S. Borges, Andre T. Endo, Marcelo M. Eler, Diego R. C. Dias, Marcelo P Guimar, and Marcelo P. Guimaraes. Machine Learning Applied to Software Testing : A Systematic Mapping Study. IEEE Transactions on Reliability, pages 1–24, 2019.

[9] Carlos Vinicius Bindewald, Willian M. Freire, Aline M.M.Miotto Amaral, and Thelma Elita Colanzi. Supporting user preferences in search-based product line architecture design using machine Learning. ACM International Conference Proceeding Series, pages 11–20, 2020.

[10] Santiago Fajardo, García-Galvan, Federico R., Violeta Barranco, Juan C. Galvan, and Sebastian Feliu Batlle. Data Mining and Machine Learning for Software Engineering. Intech, i(tourism):13, 2016.

[11] Zhiyuan Wan, Xin Xia, David Lo, and Gail C. Murphy. How does Machine Learning Change Software Development Practices? IEEE Transactions on Software Engineering, PP(c):1–1, 2019.

[12] Du Zhang. Machine Learning and Software Engineering. Software Quality Journal, 11(3):87–119, 2003.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, pages 1–14, 2015.

[14] Fabio Falcini, Giuseppe Lami, Information Science, Alessandra Mitidieri Costanza, and Fiat Chrysler Automobiles. Deep Learning in Automotive Software. IEEE Software, 34(3):56–63, 2017.

[15] S. B. Kotsiantis. Supervised Machine Learning: A Review of Classification Techniques. Informatica, 31:249–268, 2007.

[16] Victor R Basili, Gianluigi Caldiera, and H Dieter Rombach. The goal question metric approach. Encyclopedia of software engineering, 2:528–532., 1994.

[17] IEEE Computer Society, Pierre Bourque, and Richard E Fairley. Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0. IEEE Computer Society Press, Los Alamitos, CA, USA, 3rd edition, 2014.

[18] Ishan Banerjee, Bao Nguyen, Vahid Garousi, and Atif Memon. Graphical user interface (GUI) testing: Systematic mapping and repository. Information and Software Technology, 55(10):1679–1694, 2013.

[19] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic Mapping Studies in Software Engineering. 12Th International Conference on Evaluation and Assessment in Software Engineering, 17(June):10, 2008.

[20] Lianipng Chen, Muhammad Ali Babar, and He Zhang. Towards an Evidence-Based Understanding of Electronic Data Sources. In 14th International Conference on Evaluation and Assessment in Software Engineering (EASE, number April, pages 1–4, 2010.

[21] Richa Sharma, Jaspreet Bhatia, and K. K. Biswas. Machine learning for constituency test of coordinating conjunctions in requirements specifications. In International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, pages 25–31. ACM, 2014.

[22] Maninder Singh, Vaibhav Anu, Gursimran S. Walia, and Anurag Goswami. Validating Requirements Reviews by Introducing Fault-Type Level Granularity. In

SIGSOFT Innovations in Software Engineering Conference, number M, pages 1–11, 2018.
[23] Anna Perini, Angelo Susi, and Paolo Avesani. A Machine Learning Approach to Software Requirements Prioritization. IEEE Transactions on Software Engineering, 39(4):445–461, apr 2013.
[24] Guillermo Licea. Towards supporting Software Engineering using Deep Learning : A case of Software Requirements Classification. In 5th International Conference in Software Engineering Research and Innovation, pages 116–120, 2017.
[25] Jin Guo, Jinghui Cheng, and Jane Cleland-huang. Semantically Enhanced Software Traceability Using Deep Learning Techniques. In IEEE/ACM 39th International Conference on Software Engineering Semantically, pages 3–14. IEEE, 2017.
[26] Martin White, Christopher Vendome, Mario Linares-v, and Denys Poshyvanyk. Toward Deep Learning Software Repositories. In 12th Working Conference on Mining Software Repositories Toward, pages 334–345, 2015.
[27] Tam The Nguyen, Phong Minh Vu, Hung Viet Pham, and Tung Thanh Nguyen. Deep learning UI design patterns of mobile apps. In 40th International Conference on Software Engineering: New Ideas and Emerging Results Deep, pages 65–68, 2018.
[28] Vlad-sebastian Ionescu. An approach to software development effort estimation using machine learning. In 13th IEEE International Conference on Intelligent Computer Communication and Processing, pages 197–203, 2017.
[29] Mohammad Alahmadi, Jonathan Hassel, Biswas Parajuli, Sonia Haiduc, and Piyush Kumar. Accurately Predicting the Location of Code Fragments in Programming Video Tutorials Using Deep Learning. In Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering, pages 2–11, 2018.
[30] Fabiano Pecorelli, Dario Di Nucci, Coen De Roover, and Andrea De Lucia. On the role of data balancing for machine learning-based code smell detection. In 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, pages 19–24, 2019.
[31] Nakarin Maneerat. Bad-smell Prediction from Software Design Model Using Machine Learning Techniques. In Eighth International Joint Conference on Computer Science and Software Engineering, pages 331–336. IEEE, 2011.
[32] Harsh Lal. Code Review Analysis of Software System using Machine Learning Techniques. In 2017 11th International Conference on Intelligent Systems and Control (ISCO), pages 8–13. IEEE, 2017.
[33] Yesol Kim, Jonghyuk Park, and Minkyu Park. Machine Learning-based Software Classification Scheme for Efficient Program Similarity Analysis. In Conference on research in adaptive and convergent systems, pages 114–118, 2015.
[34] Jaswitha Abbineni and Ooha Thalluri. Software Defect Detection Using Machine Learning Techniques. In 2nd International Conference on Trends in Electronics and Informatics, number Icoei, pages 471–475. IEEE, 2018.
[35] Yasser Ali Alshehri, Katerina Goseva-popstojanova, Dale G Dzielski, Thomas Devine, and West Virginia. Applying machine learning to predict software fault proneness using change metrics , static code metrics , and a combination of them. In SoutheastCon, pages 1–7. IEEE, 2018.
[36] Sam Halali, Miroslaw Staron, Miroslaw Ochodek, and Wilhelm Meding. Improving Defect Localization by Classifying the Affected Asset Using Machine Learning. In International Conference on Software Quality, volume 338, pages 125–148. Springer International Publishing, 2019.
[37] Guru Prasad Bhandari and Ratneshwer Gupta. Machine learning based software fault prediction utilizing source code metrics. In IEEE 3rd International Conference on Computing, Communication and Security, pages 40–45. IEEE, 2018.
[38] Guru Prasad Bhandari and Ratneshwer Gupta. Measuring the Fault Predictability of Software using Deep Learning Techniques with Software Metrics. In 5th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics, pages 1–6. IEEE, 2018.
[39] R. Bharathi and R. Selvarani. A Machine Learning Approach for Quantifying the Design Error Propagation in Safety Critical Software System. IETE Journal of Research, 0(0):1–15, 2019.
[40] Ronyérison Braga, Pedro Santos Neto, Ricardo Rabêlo, José Santiago, and Matheus Souza. A machine learning approach to generate test oracles. In XXXII BRAZILIAN SYMPOSIUM ONSOFTWARE ENGINEERING, pages 142–151, 2018.
[41] Ariel Rosenfeld, Odaya Kardashov, and Orel Zang. Automation of Android Applications Testing Using Machine Learning Activities Classification. In 5th International Conference on Mobile Software Engineering and Systems Automation, pages 122–132, 2017.
[42] Guoyan Huang, Yazhou Li, Qian Wang, Jiadong Ren, Yongqiang Cheng, and Xiaolin Zhao. Automatic classification method for software vulnerability based on deep neural network. IEEE Access, 7:28291–28298, 2019.
[43] Cuijiao Fu. Estimating Software Energy Consumption with Machine Learning Approach by Software. In IEEE Confs on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, Congress on Cybermatics, pages 490–496. IEEE, 2018.

[44] Yoshinobu Tamura. Software Reliability Model Selection Based on Deep Learning. In 2016 International Conference on Industrial Engineering, Management Science and Application (ICIMSA), pages 1–5. IEEE, 2016.
[45] Raghvendra Kumar, L E Hoang Son, Mohamed Abdel-basset, Ishaani Priyadarshini, Rohit Sharma, and Hoang Viet Long. Deep Learning Approach for Software Maintainability Metrics Prediction. IEEE Access, 7:2169–3536, 2019.
[46] Artur Andrzejak, Luis Silva, and Dep Engenharia Informática. Using Machine Learning for Non-Intrusive Modeling and Prediction of Software Aging. In NOMS 2008 - 2008 IEEE Network Operations and Management Symposium, pages 25–32. IEEE, 2008.
[47] Dimple Chandra and Mehak Choudhary. Prophecy of Software Maintenance Effort with Univariate and Multivariate approach. In International Conference on Computing, Communication and Automation, number 1, pages 876–880, 2017.
[48] Tao Xie. The Synergy of Human and Artificial Intelligence in Software Engineering. In 2013 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), pages 4–6. IEEE, 2013.
[49] Nathalia Nascimento. Toward Human-in-the-Loop Collaboration Between Software Engineers and Machine Learning Algorithms. In IEEE International Conference on Big Data, pages 3534–3540. IEEE, 2018.
[50] Rahul Pandita, Xusheng Xiao, Hao Zhong, Tao Xie, Stephen Oney, and Amit Paradkar. Inferring method specifications from natural language API descriptions. In Proceedings - International Conference on Software Engineering, pages 815–825. IEEE, 2012.
[51] Munima Jahan, Zahra Shakeri Hossein Abad, and Behrouz Far. Detecting emergent behaviors and implied scenarios in scenario-based specifications: A machine learning approach. Proceedings - 2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering, MiSE 2019, pages 8–14, 2019.
[52] Alessio Ferrari and Andrea Esuli. An NLP approach for cross-domain ambiguity detection in requirements engineering. Automated Software Engineering, 2019.
[53] Ashish Kumar Dwivedi, Anand Tirkey, Ransingh Biswajit Ray, and Santanu Kumar Rath. Software Design Pattern Recognition using Machine Learning Techniques. In 2016 IEEE Region 10 Conference (TENCON), pages 222–227. IEEE, 2016.
[54] Du Zhang. Machine Learning in Value-Based Software Test Data Generation. In 18th IEEE International Conference on Tools with Artificial Intelligence, pages 732–736. IEEE, 2006.
[55] Onkar Malgonde and Kaushal Chari. An ensemble-based model for predicting agile software development effort, volume 24. 2019.
[56] Lucy Ellen Lwakatare B, Aiswarya Raj, Jan Bosch, and Helena Holmstr. A Taxonomy of Software Engineering Challenges for Machine Learning Systems: An Empirical Investigation. International Conference on Agile Software Development, 149:227–243, 2013.
[57] Tao Xie. Dependable Software Engineering. Theories, Tools, and Applications. In International Symposium on Dependable Software Engineering: Theories, Tools, and Applications, volume 10998, pages 3–7. Springer International Publishing, 2018.
[58] Bernd Bruegge, Joern David, Jonas Helming, and Maximilian Koegel. Classification of tasks using machine learning. In 5th International Conference on Predictor Models in Software Engineering, pages 1–11, 2009.
[59] W. T. Tsai, K. G. Heisler, D. Volovik, and I. A. Zualkernan. A critical look at the relationship between AI and software engineering. In 1988 IEEE Workshop on Languages for Automation@m_Symbiotic and Intelligent Robotics, pages 2–18, 1988.
[60] Hoa Khanh Dam. Empowering Software Engineering with Artificial Intelligence. In Australian Symposium on Service Research and Innovation, pages 3–12. Springer International Publishing, 2019.
[61] Dimitris Kalles. Artificial Intelligence meets Software Engineering in Computing Education. In 9th Hellenic Conference on Artificial Intelligence, pages 1–5, 2016.
[62] Bin Lin, Gabriele Bavota, Massimiliano Di Penta, and Michele Lanza. Sentiment Analysis for Software Engineering : How Far Can We Go ? In ACM/IEEE 40th International Conference on Software Engineering, pages 94–104. ACM, 2018.
[63] Karl Meinke and Amel Bennaceur. Machine Learning for Software Engineering: Models, Methods, and Applications. In Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, number 1, pages 548–549. ACM, 2018.
[64] D Michie. Methodologies from Machine Learning in Data Analysis and Software. THE COMPUTER JOURNAL, 34(6), 1991.
[65] Florham Park. Artificial Intelligence and Software Engineering : Breaking the Toy Mold. Automated Software Engineering, 270:255–270, 1997.
[66] Romain Robbes and Andrea Janes. Leveraging small software engineering data sets with pre-trained neural networks. In Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE-NIER 2019, pages 29–32. IEEE, 2019.
[67] Benjamin Schreck, Shankar Mallapur, Sarvesh Damle, and Nitin John James. Augmenting Software Project Managers with Predictions from Machine Learning. In IEEE International Conference on Big Data, pages

2004–2011. IEEE, 2018.
[68] Jingyi Shen, Olga Baysal, and M. Omair Shafiq. Evaluating the performance of machine learning sentiment analysis algorithms in software engineering. In Proceedings - IEEE 17th International Conference on Dependable, Autonomic and Secure Computing, IEEE 17th International Conference on Pervasive Intelligence and Computing, IEEE 5th International Conference on Cloud and Big Data Computing, 4th Cyber Scienc, pages 1023–1030. IEEE, 2019.
[69] Martin White. Deep Representations for Software Engineering. In 37th IEEE International Conference on Software Engineering, pages 781–783. IEEE, 2015.
[70] Syed Nadeem Ahsan and Franz Wotawa. Impact analysis of SCRs using single and multi-label machine learning classification. In International Symposium on Empirical Software Engineering and Measurement, pages 1–4, 2010.
[71] Erol Valeriu Chioaşcă. Using machine learning to enhance automated requirements model transformation. Proceedings - International Conference on Software Engineering, pages 1487–1490, 2012.
[72] Hojat Khosrowjerdi and Karl Meinke. Learning-based testing for autonomous systems using spatial and temporal requirements. In 1st International Workshop on Machine Learning and Software Engineering in Symbiosis, pages 6–15, 2018.
[73] Andres J. Ramirez and Betty H.C. Cheng. Automatic derivation of utility functions for monitoring software requirements. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6981 LNCS:501–516, 2011.
[74] Francois Georis Matthias Galster, Fabian Gilson. What Quality Attributes Can We Find in Product Backlogs? A Machine Learning Perspective. In European Conference on Software Architecture, pages 88–96, 2019.
[75] Cualin-Rarecs Turliuc. ProbPoly: A Probabilistic Inductive Logic Programming Framework with Application in Model Checking. Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering, pages 43–50, 2011.
[76] Syed Nadeem Ahsan, Javed Ferzund, and Franz Wotawa. Automatic Classification of Software Change Request Using Multi-Label Machine Learning Methods. In 2009 33rd Annual IEEE Software Engineering Workshop, pages 79–86. IEEE, 2009.
[77] Edna Dias Canedo and Bruno Cordeiro Mendes. Software requirements classification using machine learning algorithms. Entropy, 22(9), 2020.
[78] Armin Kobilica, Mohammed Ayub, and Jameleddine Hassine. Automated Identification of Security Requirements: A Machine Learning Approach. ACM International Conference Proceeding Series, (1):475–480, 2020.
[79] Giovanni Moranna. Natural Engineering Applying a Genetic Computing Model to Engineering Self-Aware Software Abhi. In 1st International Workshop on Software Engineering for Cognitive Services, pages 25–28, 2018.
[80] Md. Abdur Rahman, Md. Ariful Haque, Md. Nurul Ahad Tawhid, and Md. Saeed Siddik. Classifying non-functional requirements using RNN variants for quality software development. In 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, pages 25–30, 2019.
[81] Lorijn Van Rooijen, Frederik Simon B, Marie Christin Platenius, Michaela Geierhos, Heiko Hamann, and Gregor Engels. From User Demand to Software Service : Using Machine Learning to Automate the Requirements Specification Process. In 25th International Requirements Engineering Conference Workshops, pages 379–385, 2017.
[82] Jane Cleland-Huang, Adam Czauderna, Marek Gibiec, and John Emenecker. A machine learning approach for tracing regulatory codes to product specific requirements. In 32nd ACM/IEEE International Conference on Software Engineering, page 155, 2010.
[83] Ana C. Marcén, Raúl Lapeña, Óscar Pastor, and Carlos Cetina. Traceability Link Recovery between Requirements and Models using an Evolutionary Algorithm Guided by a Learning to Rank Algorithm: Train control and management case. Journal of Systems and Software, 163, 2020.
[84] Chris Mills and Sonia Haiduc. A machine learning approach for determining the validity of traceability links. Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017, pages 121–123, 2017.
[85] Kawser Wazed Nafi, Banani Roy, Chanchal K Roy, and Kevin A Schneider. CroLSim : Cross Language Software Similarity Detector using API documentation. In 18th International Working Conference on Source Code Analysis and Manipulation CroLSim:, pages 139–148. IEEE, 2018.
[86] Mateusz Wieloch, Sorawit Amornborvornwong, and Jane Cleland-huang. Trace-by-Classification : A Machine Learning Approach to Generate Trace Links for Frequently Occurring Software Artifacts. In 7th International Workshop on Traceability in Emerging Forms of Software Engineering, pages 110–114. IEEE, 2013.
[87] Shinyoung Ahn and Joongheon Kim. Poster : A Novel Shared Memory Framework for Distributed Deep Learning in High-Performance Computing Architecture. In 2018 ACM/IEEE 40th International Conference on Software Engineering: Companion Proceedings Poster:, pages 191–192, 2018.
[88] Younes Boubekeur, Gunter Mussbacher, and Shane McIntosh. Automatic assessment of students' software models using a simple heuristic and machine

learning. Proceedings - 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS-C 2020 - Companion Proceedings, pages 84–93, 2020.

[89] Oscar Castro-Lopez and Ines F. Vega-Lopez. Fast deployment and scoring of support vector machine models in CPU and GPU. In 1st International Workshop on Machine Learning and Software Engineering in Symbiosis, pages 45–52, 2018.

[90] Stéphanie Chollet, Philippe Lalanda, and Jonathan Bardin. Software Service Recommendation Base on Collaborative Filtering Neural Network Model. In International Conference on Service-Oriented Computing, pages 1–20. Springer International Publishing, 2013.

[91] Warteruzannan Soyer Cunha, São Carlos, and São Carlos. InSet: A Tool to Identify Architecture Smells Using Machine Learning. Number July 2017, pages 760–765, 2020.

[92] Ahmed Dawoud, Seyed Shahristani, and Chun Raun. Internet of Things Deep learning and software-defined networks : Towards secure IoT architecture. Internet of Things, 3-4:82–89, 2018.

[93] Quality Factors, Life-cycle Revised, José Hernández-orallo, and Mª José Ramírez-quintana. Software as Learning: Quality Factors and Life-Cycle Revised. In International Conference on Fundamental Approaches to Software Engineering, pages 147–162, 2000.

[94] Panagiotis G. Mousouliotis; and Loukas P. Petrou. Software-Defined FPGA Accelerator Design for Mobile Deep Learning Applications, volume 11444. Springer International Publishing, 2019.

[95] Henry Muccini and Karthik Vaidhyanathan. ArchLearner: Leveraging Machine-learning Techniques for Proactive Architectural Adaptation. Proceedings of the 13th European Conference on Software Architecture - Volume 2, pages 38–41, 2019.

[96] Kenneth O Neal, Philip Brisk, Ahmed Abousamra, Zack Waters, Emily Shriver, and Intel Corporation. GPU Performance Estimation using Software Rasterization. Transactions on Embedded Computing Systems, 16(5), 2017.

[97] Universitat Oberta, De Catalunya Uoc, and Robert Clarisó. Applying Graph Kernels to Model-Driven Engineering Problems. In 1st International Workshop on Machine Learning and Software Engineering in Symbiosis, pages 1–5, 2018.

[98] Anush Sankaran, Rahul Aralikatte, Senthil Mani, Shreya Khare, Naveen Panwar, and Neelamadhav Gantayat. DARVIZ : Deep Abstract Representation , Visualization , and Verification of Deep Learning Models. In International Conference on Software Engineering: New Ideas and Emerging Technologies Result, pages 47–50, 2017.

[99] Miyoung Shin and Amrit L Goel. Modeling Software Component Criticality Using a Machine Learning Approach. In International Conference on AI, Simulation, and Planning in High Autonomy Systems, pages 440–448, 2005.

[100] Brahmaleen K Sidhu, Kawaljeet Singh, and Neeraj Sharma. A Catalogue of Model Smells and Refactoring Operations for Object - Oriented Software. In 2nd International Conference on Inventive Communication and Computational Technologies, number Icicct, pages 313–319. IEEE, 2018.

[101] Qinbao Song, Xiaoyan Zhu, Guangtao Wang, Heli Sun, He Jiang, and Chenhao Xue. A machine learning based software process model recommendation method. Journal of Systems and Software, 118:85–100, 2016.

[102] Xizhu WU and Zhihua ZHOU. Model reuse with domain knowledge. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, volume 47, pages 1483–1492, 2017.

[103] C. Lakshmi Prabha and N. Shivakumar. Improving Design Quality of Software Using Machine Learning Techniques. 2020 6th International Conference on Advanced Computing and Communication Systems, ICACCS 2020, pages 583–588, 2020.

[104] R. Rajaraman, P. K. Kapur, and Deepak Kumar. Determining Software Inter-Dependency Patterns for Integration Testing by applying Machine learning on Logs and Telemetry data. ICRITO 2020 - IEEE 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions), pages 1080–1084, 2020.

[105] Hannes Thaller, Lukas Linsbauer, and Alexander Egyed. Feature Maps : A Comprehensible Software Representation for Design Pattern Detection. In 26th International Conference on Software Analysis, Evolution and Reengineering, pages 207–217. IEEE, 2019.

[106] Nuno Silva Vasil Borozanov, Simon Hacks. Using Machine Learning Techniques for Evaluating the Similarity of Enterprise Architecture Models. In International Conference on Advanced Information Systems Engineering, pages 563–578. Springer International Publishing, 2019.

[107] Mahmood Mohd Al Asheeri and Mustafa Hammad. Machine learning models for software cost estimation. In 2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies, 3ICT 2019, pages 1–6. IEEE, 2019.

[108] Bilge Ba and Burak Turhan. Software Effort Estimation Using Machine Learning Methods. In 22nd international symposium on computer and information sciences, pages 1–6. IEEE, 2007.

[109] Ahmed Banimustafa. Predicting Software Effort Estimation Using Machine Learning Techniques. In 8th International Conference on Computer Science and Information Technology, number 1, pages 249–256.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

IEEE, 2018.

[110] Anna Corazza, Sergio Di Martino, Filomena Ferrucci, Carmine Gravino, and Emilia Mendes. Using Support Vector Regression for web development effort estimation. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5891 LNCS:255–271, 2009.

[111] Egemen Ertugrul, Zakir Baytar, Çagatay Çatal, and Ömer Can Mauratli. Performance tuning for machine learning-based software development effort prediction models. Turkish Journal of Electrical Engineering & Computer Sciences, pages 1308–1324, mar 2019.

[112] Jianglin Huang, Yan-Fu Li, and Min Xie. An empirical analysis of data preprocessing for machine learning-based software cost estimation. Information and Software Technology, 67:108–127, nov 2015.

[113] Ali Idri, Azeddine Zahi, Emilia Mendes, and Abdelali Zakrani. Software cost estimation models using radial basis function neural networks. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4895 LNCS:21–31, 2008.

[114] Kazunori Iwata, Toyoshiro Nakashima, Yoshiyuki Anan, and Naohiro Ishii. Effort Estimation for Embedded Software Development Projects by Combining Machine Learning with Classification. In 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science & Engineering, pages 265–270. IEEE, 2016.

[115] Osamu Mizuno, Takanari Hamasaki, Yasunari Takagi, and Tohru Kikuno. An Empirical Evaluation of Predicting Runaway Software Projects Using Bayesian Classification. In International Conference on Product Focused Software Process Improvement, pages 263–273, 2010.

[116] Juan Murillo-Morera, Christian Quesada-López, Carlos Castro-Herrera, and Marcelo Jenkins. A genetic algorithm based framework for software effort prediction. Journal of Software Engineering Research and Development, 5(1), 2017.

[117] Shashank Mouli Satapathy and Santanu Kumar Rath. Empirical Assessment of Machine Learning Models for Effort Estimation of Web-based Applications. In 10th Innovations in Software Engineering Conference, pages 74–84, 2017.

[118] Pinkashia Sharma. Systematic Literature Review on Software Effort Estimation Using Machine Learning Approaches. In International Conference on Next Generation Computing and Information Systems, pages 43–47. IEEE, 2017.

[119] Evandro N. Regolin, Gustavo A De Souza, Aurora R T Pozo, and Silvia R Vergilio. Exploring Machine Learning Techniques for Software Size Estimation. In XXIII International Conference of the Chilean Computer Science Society, pages 130–136, 2003.

[120] Krishnamoorthy Srinivasan and Douglas Fisher. Machine Learning Approaches to Estimating Software Development Effort. TRANSACTIONS ON SOFTWARE ENGINEERING, 21(2), 1995.

[121] Om Prakash Sangwan. Software Effort Estimation using Machine Learning Techniques. In 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence, volume 5, pages 92–98, 2017.

[122] Jianfeng Wen, Shixian Li, Zhiyong Lin, Yong Hu, and Changqin Huang. Systematic literature review of machine learning based software development effort estimation models. Information and Software Technology, 54(1):41–59, jan 2012.

[123] Ian Wright and Albert Ziegler. The standard coder: A machine learning approach to measuring the effort required to produce source code change. Proceedings - 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE 2019, pages 1–7, 2019.

[124] José Cambronero, Seohyun Kim, and Satish Chandra. When Deep Learning Met Code Search. In ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 964–974, 2019.

[125] Ben Gelman, Bryan Hoyle, Jessica Moore, Joshua Saxe, and David Slater. A language-agnostic model for semantic source code labeling. In 1st International Workshop on Machine Learning and Software Engineering in Symbiosis, pages 36–44, 2018.

[126] Golam Mostaeen, Jeffrey Svajlenko, Banani Roy, Chanchal K. Roy, and Kevin A. Schneider. CloneCognition: Machine learning based code clone validation tool. ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 1105–1109, 2019.

[127] Jordan Ott, Abigail Atchison, Paul Harnack, Adrienne Bergh, and Erik Linstead. A Deep Learning Approach to Identifying Source Code in Images and Video. In 15th International Conference on Mining Software Repositories, pages 376–386, 2018.

[128] Terence Parr. Towards a Universal Code Formatter through Machine Learning. In SIGPLAN International Conference on Software Language Engineering, pages 137–151, 2016.

[129] Max Eric Henry Schumacher, Kim Tuyen Le, and Artur Andrzejak. Improving Code Recommendations by Combining Neural and Classical Machine Learning Approaches. Proceedings - 2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW 2020, pages 476–482, 2020.

[130] Abdullah M. Sheneamer. An Automatic Advisor

for Refactoring Software Clones Based on Machine Learning. IEEE Access, 8:124978–124988, 2020.

[131] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. Deep Learning Code Fragments for Code Clone Detection. In 31st IEEE/ACM International Conference on Automated Software Engineering, pages 87–98, 2016.

[132] Jiachen Yang, Keisuke Hotta, Yoshiki Higo, Hiroshi Igaki, and Shinji Kusumoto. Filtering Clones for Individual User Based on Machine Learning Analysis. In 6th International Workshop on Software Clones, pages 76–77, 2012.

[133] Marco Azadi, Umbarto, Fontana, Arcelli, Francesca, Zanoni. Poster: Machine Learning Based Code Smell Detection Through WekaNose. In ACM/IEEE 40th International Conference on Software Engineering: Companion Proceedings, pages 29–32, 2018.

[134] Mohammad Y Mhawish and Manjari Gupta. Predicting Code Smells and Analysis of Predictions: Using Machine Learning Techniques and Software Metrics. Science of Computer Programming Journal - Elsevier, 35(6):1428–1445, 2020.

[135] Noor Ayesha and N G Yethiraj. Review on Code Examination Proficient System in Software Engineering by Using Machine Learning Approach. In Proceedings of the International Conference on Inventive Research in Computing Applications, number Icirca, pages 324–327. IEEE, 2018.

[136] Krishna Teja Ayinala, Kwok Sun Cheng, Kwangsung Oh, Teukseob Song, and Myoungkyu Song. Code Inspection Support for Recurring Changes with Deep Learning in Evolving Software. Proceedings - 2020 IEEE 44th Annual Computers, Software, and Applications Conference, COMPSAC 2020, pages 931–942, 2020.

[137] Kanika Chandra, Gagan Kapoor, Rashi Kohli, and Archana Gupta. IMPROVING SOFTWARE QUALITY USING MACHINE LEARNING. In 1st International Conference on Innovation and Challenges in Cyber Security, number Iciccs, pages 115–118. IEEE, 2016.

[138] Gilles Fouqu and Christel Vrain. Building a Tool for Software Code Analysis A Machine Learning Approach structures. In International Conference on Advanced Information Systems Engineering, pages 278–289, 1992.

[139] Verena Geist, Michael Moser, Josef Pichler, Stefanie Beyer, and Martin Pinzger. Leveraging Machine Learning for Software Redocumentation. SANER 2020 - Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution, and Reengineering, pages 622–626, 2020.

[140] Michał Madera and Rafał Tomoń. A case study on machine learning model for code review expert system in software engineering. In Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, volume 11, pages 1357–1363, 2017.

[141] Mehwish Naseer, Wu Zhang, and Wenhao Zhu. Prediction of coding intricacy in a software engineering team through machine learning to ensure cooperative learning and sustainable education. Sustainability (Switzerland), 12(21):1–15, 2020.

[142] Anubhav Trivedi. Code Nano-Pattern Detection using Deep Learning. In Proceedings of the 13th Innovations in Software Engineering Conference on Formerly known as India Software Engineering Conference, 2020.

[143] Alexander Leclair, Zachary Eberhart, and Collin Mcmillan. Adapting Neural Text Classification for Improved Software Categorization. In IEEE International Conference on Software Maintenance and Evolution, pages 461–472. IEEE, 2018.

[144] Zhensu Sun, Yan Liu, Ziming Cheng, Chen Yang, and Pengyu Che. Req2Lib: A semantic neural model for software library recommendation. arXiv, pages 542–546, 2020.

[145] Michele Tufano, College William, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. Deep Learning Similarities from Different Representations of Source Code. In 15th International Conference on Mining Software Repositories, pages 542–553, 2018.

[146] Xian Zhang and Kerong Ben. A Neural Language Model with a Modified Attention. In 9th International Conference on Software Engineering and Service Science, pages 232–236. IEEE, 2018.

[147] Gang Zhao. DeepSim : Deep Learning Code Functional Similarity. In 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 141–151, 2018.

[148] Samar M Abozeed. Software Bug Prediction Employing Feature Selection and Deep Learning. 2020.

[149] Markus Borg, Oscar Svensson, Kristian Berg, and Daniel Hansson. SZZ unleashed: an open implementation of the SZZ algorithm - featuring example usage in a study of just-in-time bug prediction for the Jenkins project. In 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, pages 7–12, 2019.

[150] Y. Brun and M.D. Ernst. Finding latent code errors via machine learning over program executions. In 26th International Conference on Software Engineering, pages 480–490, 2004.

[151] Mariela Cerrada, Diego Cabrera, Jean Macancela, Pablo Lucero, Fannia Pacheco, Rene Vinicio Sanchez, Diego Cabrera, Jean Macancela, and Pablo Lucero. SOA Based Integrated Software to Develop Fault Diagnosis Models Using Machine Learning in Rotating Machinery. In Proceedings - 11th IEEE International Symposium on Service-Oriented System Engineering,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

SOSE 2017, pages 28–37, 2017.
[152] Evren Ceylan, F Onur Kutlubay, and B Bener. Software Defect Identification Using Machine Learning Techniques. In 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, pages 240–247, 2006.
[153] Venkata U B Challagulla, Farokh B Bastani, I-ling Yen, and Raymond A Paul. Empirical Assessment of Machine Learning based Software Defect Prediction Techniques. In 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, pages 389–400, 2005.
[154] Jinyin Chen, Keke Hu, Yue Yu, Zhuangzhi Chen, Qi Xuan, Yi Liu, and Vladimir Filkov. Software visualization and deep transfer learning for effective software defect prediction. Proceedings - International Conference on Software Engineering, pages 578–589, 2020.
[155] Caesar Jude Clemente, Fehmi Jaafar, and Yasir Malik. Is Predicting Software Security Bugs using Deep Learning Better than the Traditional Machine Learning Algorithms ? In IEEE International Conference on Software Quality, Reliability and Security Is, pages 95–102. IEEE, 2018.
[156] Hoa Khanh Dam, Trang Pham, Shien Wee Ng, Truyen Tran, John Grundy, Aditya Ghose, Taeksu Kim, and Chul Joo Kim. Lessons learned from using a deep tree-based model for software defect prediction in practice. In IEEE International Working Conference on Mining Software Repositories, volume 2019-May, pages 46–57. IEEE, 2019.
[157] Anurag Dwarakanath, Manish Ahuja, Samarth Sikand, Raghotham M. Rao, R. P. Jagadeesh Chandra Bose, Neville Dubash, and Sanjay Podder. Identifying Implementation Bugs in Machine Learning based Image Classifiers using Metamorphic Testing. In 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, pages 118–128, 2018.
[158] Javed Ferzund, Syed Nadeem Ahsan, and Franz Wotawa. Analysing bug prediction capabilities of static code metrics in open source software. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5338 LNCS:331–343, 2008.
[159] Aashish Gupta, Shilpa Sharma, Shubham Goyal, and Mamoon Rashid. Novel XGBoost Tuned Machine Learning Model for Software Bug Prediction. Proceedings of International Conference on Intelligent Engineering and Management, ICIEM 2020, pages 376–380, 2020.
[160] Tracy Hall and David Bowes. The State of Machine Learning Methodology in Software Fault Prediction. In 11th International Conference on Machine Learning and Applications, pages 308–313. IEEE, 2012.
[161] Kihong Heo, Hakjoo Oh, and Kwangkeun Yi. Machine-Learning-Guided Selectively Unsound Static Analysis. Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE 2017, pages 519–529, 2017.
[162] Xuan Huo, Yang Yang, Ming Li, and De-chuan Zhan. Learning Semantic Features for Software Defect Prediction by Code Comments Embedding. In IEEE International Conference on Data Mining, pages 1049–1054. IEEE, 2018.
[163] Vasu Jindal. Towards an Intelligent Fault Prediction Code Editor to Improve Software Quality using Deep Learning. In 2nd International Conference on the Art, Science, and Engineering of Programming, pages 222–223, 2018.
[164] Julen Kahles and Alexander Jung. Automating Root Cause Analysis via Machine Learning in Agile Software Testing Environments. In 12th IEEE Conference on Software Testing, Validation and Verification, pages 379–390. IEEE, 2019.
[165] Neha M Kalibhat and Shreya Varshini. Software Troubleshooting using Machine Learning. In 24th International Conference on High Performance Computing Workshops, pages 3–10, 2017.
[166] Prashanth Kambli. Predicting Bug in a Software using ANN Based Machine Learning Techniques. pages 1–5, 2020.
[167] Syaeful Karim, Harco Leslie, Hendric Spits, Edi Abdurachman, and Benfano Soewito. Software Metrics for Fault Prediction Using Machine Learning Approaches. In IEEE International Conference on Cybernetics and Computational Intelligence, pages 19–23, 2017.
[168] Mijung Kim, Jaechang Nam, Jaehyuk Yeon, Soonhwang Choi, and Sunghun Kim. REMI: Defect prediction for efficient API testing. In 2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2015 - Proceedings, pages 990–993, 2015.
[169] Amod Kumar and Ashwni Bansal. Software Fault Proneness Prediction Using Genetic Based Machine Learning Techniques. In Proceedings - 2019 4th International Conference on Internet of Things: Smart Innovation and Usages, IoT-SIU 2019, pages 1–5. IEEE, 2019.
[170] Harsh Lal. Root Cause Analysis of Software Bugs using Machine Learning Techniques. In 2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence, pages 105–111. IEEE, 2017.
[171] Jian Li, Pinjia He, Jieming Zhu, and Michael R Lyu. Software Defect Prediction via Convolutional Neural Network. In IEEE International Conference on Software Quality, Reliability and Security, pages 318–328, 2017.
[172] Hongliang Liang, Yue Yu, Lin Jiang, and Zhuosi Xie. Seml: A Semantic LSTM Model for Software Defect

Prediction. IEEE Access, 7:83812–83824, 2019.
[173] Yunfeng Luo, Kerong Ben, and Lei Mi. Software metrics reduction for fault-proneness prediction of software modules. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6289 LNCS:432–441, 2010.
[174] Sushant; Priya Pragati Malhotra, Ruchika; Bahl, Laavanye; Sehgal. Empirical Comparison of Machine Learning Algorithms for Bug Prediction in Open Source Software. In International Conference on Big Data Analytics and Computational Intelligence, pages 40–45, 2017.
[175] Amirabbas Majd, Mojtaba Vahidi-Asl, Alireza Khalilian, Pooria Poorsarvi-Tehrani, and Hassan Haghighi. SLDeep: Statement-level software defect prediction using deep-learning model on static code features. Expert Systems with Applications, 147:113156, 2020.
[176] Chao Ni, Wang Shu Liu, Xiang Chen, Qing Gu, Dao Xu Chen, and Qi Guo Huang. A Cluster Based Feature Selection Method for Cross-Project Software Defect Prediction. Journal of Computer Science and Technology, 32(6):1090–1107, 2017.
[177] Anh Viet Phan and Minh Le Nguyen. Convolutional Neural Networks on Assembly Code for Predicting Software Defects. In 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems, pages 37–42, 2017.
[178] Anh Viet Phan and Minh Le Nguyen. Convolutional Neural Networks over Control Flow Graphs for Software Defect Prediction. In International Conference on Tools with Artificial Intelligence Convolutional, pages 45–52, 2017.
[179] Thi Minh Phuong Ha, Duy Hung Tran, Le Thi My Hanh, and Nguyen Thanh Binh. Experimental study on software fault prediction using machine learning model. In Proceedings of 2019 11th International Conference on Knowledge and Systems Engineering, KSE 2019, pages 1–5. IEEE, 2019.
[180] Sravya Polisetty, Andriy Miranskyy, and Ayşe Başar. On Usefulness of the Deep-Learning-Based Bug Localization Models to Practitioners. In Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering, number Ml, pages 16–25, 2019.
[181] Osama Al Qasem, Mohammed Akour, and Mamdouh Alenezi. The Influence of Deep Learning Algorithms Factors in Software Fault Prediction. IEEE Access, 8:63945–63960, 2020.
[182] Anuradha Chug Praman Deep Singh. Software Defect Prediction Analysis Using Machine Learning Algorithms. In 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence, pages 775–781, 2017.
[183] C. Lakshmi Prabha and N. Shivakumar. Software Defect Prediction Using Machine Learning Techniques. Proceedings of the 4th International Conference on Trends in Electronics and Informatics, ICOEI 2020, (Icoei):728–733, 2020.
[184] Rakesh Rana, Miroslaw Staron, Jörgen Hansson, Martin Nilsson, and Wilhelm Meding. A Framework for Adoption of Machine Learning in Industry for Software Defect Prediction. In 9th International Conference on Software Engineering and Applications, pages 383–392, 2003.
[185] M. Floramary S. Delphine Immaculate, M. Farida Begam. Software Bug Prediction Using Supervised Machine Learning Algorithms. In 2019 International Conference on Data Science and Communication (IconDSC), pages 1–7. IEEE, 2019.
[186] Martin Shepperd, Tracy Hall, David Bowes, and Tracy Hall. Researcher Bias : The Use of Machine Learning in Software Defect Prediction. TRANSACTIONS ON SOFTWARE ENGINEERING, 40(6):603–616, 2014.
[187] Suyash Shukla, Ranjan Kumar Behera, Sanjay Misra, and Santanu Kumar Rath. Software Reliability Assessment Using Deep Learning Technique. In International Conference on Computational Science and Its Applications, pages 57–68. Springer Singapore, 2018.
[188] Yogesh Singh, Arvinder Kaur, and Ruchika Malhotra. Predicting software fault proneness model using neural network. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5089 LNCS:204–214, 2008.
[189] Prabhpahul Singh and Ruchika Malhotra. Assessment of Machine Learning Algorithms for Determining Defective Classes in an Object-Oriented Software. In 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions), pages 204–209, 2017.
[190] Rituraj Singh, Jasmeet Singh, Mehrab Singh Gill, Ruchika Malhotra, and Garima. Transfer Learning Code Vectorizer based Machine Learning Models for Software Defect Prediction. 2020 International Conference on Computational Performance Evaluation, ComPE 2020, (Lm):497–502, 2020.
[191] Uma Subbiah, Muthu Ramachandran, and Zaigham Mahmood. Software Engineering Framework for Software Defect Management Using Machine Learning Techniques with Azure. In Software Engineering in the Era of Cloud Computing. Springer International Publishing, 2020.
[192] D. Sudharson and D. Prabha. A novel machine learning approach for software reliability growth modelling with pareto distribution function. Soft Computing, 23(18):8379–8387, 2019.
[193] Yuanyuan Sun and Yong Ming Wang. Utilizing Deep Architecture Networks of VAE in Software Fault Prediction. In Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing,

Social Computing & Networking, Sustainable Computing & Communications, pages 870–877, 2018.

[194] Kazuya Tanaka, Akito Monden, and Zeynep Yucel. Prediction of Software Defects Using Automated Machine Learning. In Proceedings - 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2019, pages 490–494, 2019.

[195] Junfeng Tian and Yongqing Tian. A Model Based on Program Slice and Deep Learning for Software Defect Prediction. Proceedings - International Conference on Computer Communications and Networks, ICCCN, 2020-Augus, 2020.

[196] Hannes Thaller, Lukas Linsbauer, Alexander Egyed, and Stefan Fischer. Towards Fault Localization via Probabilistic Software Modeling. arXiv, pages 24–27, 2020.

[197] Song Wang, Taiyue Liu, Jaechang Nam, and Lin Tan. Deep Semantic Feature Learning for Software Defect Prediction. Transactions on Software Engineering, 5589(c):1–26, 2018.

[198] Ming Wen, Rongxin Wu, and Shing-chi C. Cheung. How Well Do Change Sequences Predict Defects ? Sequence Learning from Software Changes. Transactions on Software Engineering, 5589(c):1–20, 2018.

[199] Kittisak Wongpheng and Porawat Visutsak. Software Defect Prediction using Convolutional Neural Network. ITC-CSCC 2020 - 35th International Technical Conference on Circuits/Systems, Computers and Communications, pages 240–243, 2020.

[200] Zhiwu Xu, Kerong Ren, Shengchao Qin, and Florin Craciun. CDGDroid: Android Malware Detection Based on Deep Learning Using CFG and DFG. In International Conference on Formal Engineering Methods, volume 3308, pages 177–193. Springer International Publishing, 2004.

[201] Siqi Yang, Shuaipeng Yang, Zigang Fang, Xiuzhi Yu, Lanlan Rui, and Yucheng Ma. Fault Prediction for Software System in Industrial Internet: A Deep Learning Algorithm via Effective Dimension Reduction. In International Conference on Cyber-Living, Cyber-Syndrome and Cyber-Health, volume 1137 CCIS, pages 572–580. Springer Singapore, 2019.

[202] Xian Zhang, Kerong Ben, and Jie Zeng. Cross-Entropy : A New Metric for Software Defect Prediction. In IEEE International Conference on Software Quality, Reliability and Security, pages 111–122. IEEE, 2018.

[203] Linchang Zhao, Student Member, and Zhaowei Shang. Siamese Dense Neural Network for Software Defect Prediction With Small Data. IEEE Access, 7:7663–7677, 2019.

[204] Faezeh Sadat Babamir, Alireza Hatamizadeh, and Seyed Mehrdad Babamir. Application of Genetic Algorithm in Automatic Software Testing. In International Conference on Networked Digital Technologies, pages 545–552, 2010.

[205] Lionel C Briand. Novel Applications of Machine Learning in Software Testing. In Eighth International Conference on Quality Software Novel, pages 3–10. IEEE, 2008.

[206] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems. In 33rd ACM/IEEE International Conference on Automated Software Engineering, pages 120–131, 2018.

[207] Valdivino Alexandre de Santiago, Leoni Augusto Romain da Silva, and Pedro Ribeiro de Andrade Neto. Testing Environmental Models supported by Machine Learning. In III Brazilian Symposium on Systematic and Automated Software Testing, pages 3–12, 2018.

[208] Norbert Oster. Automated generation and evaluation of dataflow-based test data for object-oriented software. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 3712 LNCS:212–226, 2005.

[209] Ciprian Paduraru, Marius-constantin Melemciuc, and Miruna Paduraru. Automatic Test Data Generation for a Given Set of Applications Using Recurrent Neural Networks. In International Conference on Software Technologies, volume 1, pages 307–326. Springer International Publishing, 2019.

[210] Robert Gove and Jorge Faytong. Machine Learning and Event-Based Software Testing : Classifiers for Identifying Infeasible GUI Event Sequences. Advances in Computers, 86:109–135, 2012.

[211] Gang Hu, Linjie Zhu, and Junfeng Yang. AppFlow: using machine learning to synthesize robust, reusable UI tests. In 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 269–282, 2018.

[212] Alexandre Rafael Lenz, Aurora Pozo, and Silvia Regina. Linking software testing results with a machine learning approach. Engineering Applications of Artificial Intelligence, 26(5-6):1631–1640, 2013.

[213] Wei Zheng, Yutong Bai, and Haoxuan Che. A computer-assisted instructional method based on machine learning in software testing class. Computer Applications in Engineering Education, 26(5):1150–1158, sep 2018.

[214] Javier Alonso. Predicting Software Anomalies using Machine Learning Techniques. In IEEE International Symposium on Network Computing and Applications Predicting, pages 163–170, 2011.

[215] Aziz Alotaibi. Identifying Malicious Software Using Deep Residual Long-Short Term Memory. IEEE Access, 7:163128–163137, 2019.

[216] Federica Bisio, Paolo Gastaldo, Rodolfo Zunino, and

Sergio Decherchi. Semi-supervised machine learning approach for unknown malicious software detection. In International Symposium on Innovations in Intelligent Systems and Applications, pages 52–59. IEEE, 2014.

[217] Hoa Khanh Dam, Truyen Tran, Trang Pham, Shien Wee Ng, John Grundy, and Aditya Ghose. Automatic feature learning for predicting vulnerable software components. IEEE Transactions on Software Engineering, 14(8):1–19, 2015.

[218] Xpdu H Dqg, Vrsqlo Vdpudw, Jpdlo Frp, Pdkexefvh Dkrr, F R P Dqg, and Pnudlkdq Jpdlo. Detection of Flow Based Anomaly in OpenFlow Controller : Machine Learning Approach in Software Defined Networking. In 4th International Conference on Electrical Engineering and Information & Communication Technology, pages 416–421, 2018.

[219] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques : A Survey. ACM Computing Surveys, 50(4), 2017.

[220] Sumanth Gowda, Divyesh Prajapati, Ranjit Singh, and Swanand S. Gadre. False Positive Analysis of software vulnerabilities using Machine learning. In 2018 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), pages 3–6. IEEE, nov 2018.

[221] Zhuobing Han, Xiaohong Li, Zhenchang Xing, Hongtao Liu, and Zhiyong Feng. Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description. In IEEE International Conference on Software Maintenance and Evolution Learning, pages 125–136, 2017.

[222] Fabian Huch, Mojdeh Golagha, Ana Petrovska, and Alexander Krauss. Machine Learning-Based Run-Time Anomaly Detection in Software Systems : An Industrial Evaluation. In IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation, pages 13–18. IEEE, 2018.

[223] Gong Jie, Kuang Xiao-Hui, and Liu Qiang. Survey on Software Vulnerability Analysis method based on Machine Learning. In 2016 IEEE First International Conference on Data Science in Cyberspace (DSC), pages 642–647. IEEE, jun 2016.

[224] Jorrit Kronjee and Harald Vranken. Discovering software vulnerabilities using data-flow analysis and machine learning. In International Conference on Availability, Reliability and Security, pages 1–10, 2018.

[225] Alexandr Kuznetsov, Yehor Yeromin, Oleksiy Shapoval, Kyrylo Chernov, Mariia Popova, and Kostyantyn Serdukov. Automated software vulnerability testing using deep learning methods. In 2019 IEEE 2nd Ukraine Conference on Electrical and Computer Engineering, UKRCON 2019 - Proceedings, pages 837–841. IEEE, 2019.

[226] Muhammad Noman Khalid, Humera Farooq, Muhammad Iqbal, Muhammad Talha Alam, and Kamran Rasheed. Predicting Web Vulnerabilities in Web Applications Based on Machine Learning Muhammad. In International Conference on Intelligent Technologies and Applications, volume 932, pages 496–510. Springer Singapore, 2019.

[227] Weina Niu, Xiaosong Zhang, Xiaojiang Du, Lingyuan Zhao, Rong Cao, and Mohsen Guizani. A Deep Learning Based Static Taint Analysis Approach for IoT Software Vulnerability Location. Measurement, 152:107139, 2019.

[228] Saahil Ognawala, Ricardo Nales Amato, Alexander Pretschner, and Pooja Kulkarni. Automatically Assessing Vulnerabilities Discovered by Compositional Analysis. In 1st International Workshop on Machine Learning and Software Engineering in Symbiosis, pages 16–25, 2018.

[229] Invited Paper. Deep Learning Approach for Network Intrusion Detection in Software Defined Networking. In International Conference on Wireless Networks and Mobile Communications, pages 1–6. IEEE, 2016.

[230] Edmar Rezende, Guilherme Ruppert, Tiago Carvalho, Fabio Ramos, and Paulo De Geus. Malicious Software Classification using Transfer Learning of ResNet-50 Deep Neural Network. In 16th IEEE International Conference on Machine Learning and Applications Malicious, pages 1011–1014, 2017.

[231] Guanhua Yan and Junchen Lu. ExploitMeter : Combining Fuzzing with Machine Learning for Automated Evaluation of Software Exploitability. In 2017 IEEE Symposium on Privacy-Aware Computing, pages 164–175, 2017.

[232] Mohammed Zagane, Mustapha Kamel Abdi, and Mamdouh Alenezi. Deep Learning for Software Vulnerabilities Detection Using Code Metrics. IEEE Access, 8:74562–74570, 2020.

[233] Falk Howar B, Karl Meinke, and Andreas Rausch. Learning Systems : Machine-Learning in Software Products and Learning-Based Analysis of Software Systems. In International Symposium on Leveraging Applications of Formal Methods, volume 1, pages 651–654, 2016.

[234] Hoa Khanh Dam. Explainable Software Analytics. In 40th International Conference on Software Engineering: New Ideas and Emerging Results, pages 53–56. ACM, 2018.

[235] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, and Sven Apel. The Interplay of Sampling and Machine Learning for Software Performance Prediction. IEEE Software, 37(4):58–66, 2020.

[236] Hyun-il Lim and A Code Vector. Applying Code Vectors for Presenting Software Features in Machine Learning. In 42nd IEEE International Conference on Computer Software & Applications Applying, pages 803–804. IEEE, 2018.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119746, IEEE Access

[237] Qin Liu, Xiaolong Li, Hongming Zhu, and Hongfei Fan. Acquisition of Open Source Software Project Maturity Based on Time Series Machine Learning. In 10th International Symposium on Computational Intelligence and Design, pages 10–13, 2017.
[238] Mahshid Helali Moghadam. Machine Learning-Assisted Performance Testing. In ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 12–14, 2019.
[239] Federico Quin, Danny Weyns, Thomas Bamelis, Singh Buttar Sarpreet, and Sam Michiels. Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning. In ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, volume 2019-May, pages 1–12, 2019.
[240] Gopi Krishnan Rajbahadur, Shaowei Wang, Yasutaka Kamei, and Ahmed E. Hassan. Impact of Discretization Noise of the Dependent variable on Machine Learning Classifiers in Software Engineering. IEEE Transactions on Software Engineering, (c), 2019.
[241] Sefa Eren Sahin, Kubilay Karpat, and Ayse Tosun. Predicting Popularity of Open Source Projects Using Recurrent Neural Networks. In IFIP International Conference on Open Source Systems, volume 556, pages 80–90. Springer International Publishing, 2019.
[242] Ramtine Tofighi-Shirazi, Irina-Mariuca Asavoae, Philippe Elbaz-Vincent, and Thanh-Ha Le. Defeating Opaque Predicates Statically through Machine Learning and Binary Analysis. In 3rd Software Protection Workshop, pages 3–14, 2019.
[243] Hironori Washizaki, Hironori Takeuchi, Foutse Khomh, Naotake Natori, Takuo Doi, and Satoshi Okuda. Practitioners' insights on machine-learning software engineering design patterns: A preliminary study. Proceedings - 2020 IEEE International Conference on Software Maintenance and Evolution, ICSME 2020, (Ml):797–799, 2020.
[244] Mehdi T Harandi and Hing-yan Lee. Acquiring Software Design Schemas : A Machine Learning Perspective. In 6th International Conference on Knowledge-Based Software Engineering, pages 188–197, 1991.
[245] Hakim Lounis and Lynda Ait-mehedine. Machine-Learning Techniques for Software Product Quality Assessment. In Fourth International Conference on Quality Software, pages 102–109. IEEE, 2004.
[246] Hakim Lounis, Tamer Fares Gayed, and Mounir Boukadoum. Machine-Learning Models for Software Quality : a Compromise Between Performance and Intelligibility Machine-Learning Models for Software Quality : a Compromise Between Performance and Intelligibility. In 23rd International Conference on Tools with Artificial Intelligence, number November 2011, pages 64–67, 2014.
[247] Hitesh Sajnani. Automatic Software Architecture Recovery : A Machine Learning Approach. In 20th IEEE International Conference on Program Comprehension, pages 265–268. IEEE, 2012.
[248] Matthew C Simpson. Automatic Algorithm Selection in Computational Software Using Machine Learning. In 15th IEEE International Conference on Machine Learning and Applications Automatic, pages 355–360. IEEE, 2016.
[249] Ning Chen, Steven C H Hoi, and Xiaokui Xiao. Software Process Evaluation : A Machine Learning Approach. In 26th IEEE/ACM International Conference on Automated Software Engineering, number ii, pages 333–342. IEEE, 2011.
[250] Cuauhtemoc Lopez-martin, Arturo Chavoya, and Maria Elena Meda-campaña. A Machine Learning Technique for Predicting the Productivity of Practitioners from Individually Developed Software Projects. In IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, pages 1–6, 2014.
[251] Diego Rughetti, Pierangelo Di Sanzo, Bruno Ciciani, and Francesco Quaglia. Machine Learning-based Self-adjusting Concurrency in Software Transactional Memory Systems. In 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pages 278–285. IEEE, 2012.
[252] 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), pages 1–4. IEEE, 2013.
[253] Nicolas Baskiotis, Marie-claude Gaudel, and Sandrine Gouraud. A Machine Learning Approach for Statistical Software Testing. In IJCAI International Joint Conference on Artificial Intelligence, pages 2274–2279, 2006.
[254] Chris Cummins, Pavlos Petoumenos, Alastair Murray, and Hugh Leather. Compiler fuzzing through deep learning. Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis - ISSTA 2018, pages 95–105, 2018.
[255] Hasan Ferit Enişer and Alper Sen. Testing service oriented architectures using stateful service visualization via machine learning. In ACM/IEEE 13th International Workshop on Automation of Software Test Testing, pages 9–15, 2018.
[256] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. DLFuzz: Differential Fuzzing Testing of Deep Learning Systems. In 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 739–743, 2018.
[257] Upulee Kanewala, James M. Bieman, and Asa Ben-

Hur. Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels. Software Testing, Verification and Reliability, 26(3):245–269, may 2016.
[258] Yu. Karpov, Yu. Smetanin, and L. Karpov. Adaptation of general software testing concepts to neural networks. Programming and Computer Software, 44(5):43–56, 2019.
[259] Satoshi Masuda, Kohichi Ono, Toshiaki Yasue, and Nobuhiro Hosokawa. A Survey of Software Quality for Machine Learning Applications. In International Conference on Software Testing, Verification and Validation Workshops, pages 279–284. IEEE, 2018.
[260] Maskura Nafreen, Saikath Bhattacharya, and Lance Fiondella. Architecture-based software reliability incorporating fault tolerant machine learning. Proceedings - Annual Reliability and Maintainability Symposium, 2020-Janua, 2020.
[261] Shin Nakajima. Quality Assurance of Machine Learning Software. In 7th Global Conference on Consumer Electronics, pages 601–604. IEEE, 2018.
[262] Raymond A Paul. A Machine Learning-Based Reliability Assessment Model for Critical Software Systems. In 31st Annual International Computer Software and Applications Conference, number Compsac, pages 79–86, 2007.
[263] Rakesh Rana and Miroslaw Staron. Machine Learning Approach for Quality Assessment and Prediction in Large Software Organizations. In 6th IEEE International Conference on Software Engineering and Service Science, pages 1098–1101. IEEE, 2015.
[264] D Shanthi, R K Mohanty, and G Narsimha. Application of Machine Learning Reliability Data Sets. In 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS), number Iciccs, pages 1472–1474. IEEE, 2018.
[265] Jinyong Wang and Ce Zhang. Software reliability prediction using a deep learning model based on the RNN encoder – decoder. Reliability Engineering & System Safety, 170(October 2017):73–82, feb 2018.
[266] Mauricio A De Almeida, De Mont, P O Box, Mcgill College Ave, and K I N On. An Investigation on the Use of Machine Learned Models for Estimating Correction Costs. In Proceedings of the 20th international conference on Software engineering, pages 473–476, 1998.
[267] Thomas J Cheatham and Nancy J Wahl. Software Testing : A Machine Learning Experiment. In 23rd annual conference on Computer science, pages 135–141, 1995.
[268] Subburaj Ramasamy and Indhurani Lakshmanan. Machine Learning Approach for Software Reliability Growth Modeling with Infinite Testing Effort Function. Mathematical Problems in Engineering, 2017, 2017.
[269] Hamdi A. Al-Jamimi and Moataz A. Ahmed. Machine learning approaches for predicting software maintainability: a fuzzy-based transparent model. IET Software, 7(6):317–326, 2013.
[270] Sara Elmidaoui, Laila Cheikhi, Ali Idri, and Alain Abran. Machine Learning Techniques for Software Maintainability Prediction: Accuracy Analysis. Journal of Computer Science and Technology, 35(5):1147–1174, 2020.
[271] Sandeep Reddivari and Jayalakshmi Raman. Software quality prediction: An investigation based on machine learning. In Proceedings - 2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science, IRI 2019, pages 115–122. IEEE, 2019.
[272] Javier Alonso and Jordi Torres. Adaptive on-line software aging prediction based on Machine Learning. In IEEEIIFIP International Conference on Dependable Systems & Networks, number July 2004, pages 507–516, 2010.
[273] Shouyu Huo, Dongdong Zhao, Xing Liu, Jianwen Xiang, Yingshou Zhong, and Haiguo Yu. Using Machine Learning for Software Aging Detection in Android System. In Tenth International Conference on Advanced Computational Intelligence, pages 741–746. IEEE, 2018.
[274] Sangho Lee, Changhee Jung, and Santosh Pande. Detecting memory leaks through introspective dynamic behavior modelling using machine learning. In 36th International Conference on Software Engineering, pages 814–824, 2014.
[275] Yongquan Yan and Ping Guo. A Practice Guide of Software Aging Prediction in a Web Server Based on Machine Learning. SECURITY SCHEMES AND SOLUTIONS, (June):225–235, 2016.
[276] Kui Zhang, Xu Wang, Jian Ren, and Chao Liu. Efficiency Improvement Of Function Point-Based Software Size Estimation With Deep Learning Model. IEEE Access, 4:1–1, 2020.
[277] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning, volume 11. 2018.
[278] Saad Shafiq, Christoph Mayr-Dorn, Atif Mashkoor, and Alexander Egyed. Towards Optimal Assembly Line Order Sequencing with Reinforcement Learning: A Case Study. IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, 2020-Septe:982–989, 2020.
[279] Ana Barcus and Gilberto Montibeller. Supporting the allocation of software development work in distributed teams with multi-criteria decision analysis. Omega, 36(3):464–475, 2008.
[280] Philip Achimugu, Ali Selamat, Roliana Ibrahim, and Mohd Naz Ri Mahrin. A systematic literature review of software requirements prioritization research. Information and Software Technology, 56(6):568–585,

2014.

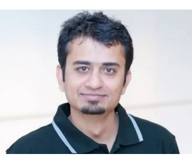
[281] Tore Dyba, Torgeir Dingsøyr, and Geir K. Hanssen. Impact analysis of missing values on the prediction accuracy of analogy-based software effort estimation method AQUA. In Proceedings - 1st International Symposium on Empirical Software Engineering and Measurement, ESEM 2007, number 7465, pages 126–135, 2007.

[282] Claes Wohlin, Per Runeson, Paulo Anselmo Da Mota Silveira Neto, Emelie Engström, Ivan Do Carmo Machado, and Eduardo Santana De Almeida. On the reliability of mapping studies in software engineering. Journal of Systems and Software, 86(10):2594–2610, 2013.

[283] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology, 64:1–18, 2015.

[284] Barbara A. Kitchenham, David Budgen, and O. Pearl Brereton. The value of mapping studies – A participant-observer case study. In 14th international conference on evaluation and assessment in software engineering (ease), pages 1–9, 2010.

SAAD SHAFIQ is a PhD student at the Johannes Kepler University. His research interests include applications of machine learning, requirements engineering, and empirical software engineering. He received his degree in Software Engineering (MSc) from the National University of Computer and Emerging Sciences. He worked as a Software Engineer with ample experience in software development and data science.



ATIF MASHKOOR is a Senior Research Scientist at Johannes Kepler University Linz (Austria). Previously, he was the Scientific Head at Software Competence Center Hagenberg GmbH – the Austrian center of excellence in data and software science. He has taught at the University of Lorraine (France), COMSATS Institute of Information Technology (Pakistan), and National University of Modern Languages (Pakistan). He worked as a Research Associate at the University of Minho (Portugal). He holds a doctoral degree from the University of Lorraine (France) and a master degree from Umeå University (Sweden), both in Computer Science. Additionally, he studied Computational Linguistics at Rovira i Virgili University (Spain).

CHRISTOPH MAYR-DORN is a Post-Doctoral Researcher at the Institute for Software Systems Engineering at the Johannes Kepler University, Linz. He currently leads the FWF-funded Project: "C4S – Coordination-centric Change and Consistency Support". Previously, Christoph held a University Assistant (post-doctoral) position at the Distributed Systems Group (TU Vienna). He has worked since 2006 as a research assistant at the Technical University of Vienna. He received his degree in Computer Science and Economics (Wirtschaftsinformatik) (MSocEcSc/Mag. rer. soc. oec.) in 2004 and his Dr. Techn./PhD in Computer Science in 2009. His research interest is focused on Collaborative Engineering Environments, Adaptive Collaboration Patterns, Software Architecture, Change Propagation, and Adaptive Processes. Awarded an FWF Schrödinger Mobility Fellowship (Marie Curie Co-funded), Christoph was a visiting researcher with Prof. Richard Taylor at U.C. Irvine from March 2011 to August 2012.

ALEXANDER EGYED received the doctorate degree from the University of Southern California (USC). He is a Full Professor at the Johannes Kepler University (JKU), Austria. He was with Teknowledge Corporation (2000-2007) and University College London, United Kingdom (2007-2008). He is most recognized for his work on software and systems modeling—particularly on the consistency and traceability of models. His work has been published in more than 100 refereed scientific books, journals, conferences, and workshops, with more than 3,000 citations to date. He was recognized as the 10th best scholar in software engineering in Communications of the ACM, was named an IBM Research Faculty Fellow in recognition to his contributions to consistency checking, received a Recognition of Service Award from the ACM, a Best Paper Award from COMPSAC, and an Outstanding Achievement Award from USC. He has given many invited talks, including four keynotes, served on scientific panels and countless program committees, and has served as program (co)chair, steering committee member, and editorial board member. He is a member of the IEEE, IEEE Computer Society, ACM, and ACM SigSoft.