IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 23 February 2023, accepted 23 March 2023, date of publication 27 March 2023, date of current version 30 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3262411

RESEARCH ARTICLE

Error-Type–A Novel Set of Software Metrics for Software Fault Prediction

KHOA PHUNG, EMMANUEL OGUNSHILE, AND MEHMET AYDIN, (Senior Member, IEEE)
School of Computing and Creative Technologies, University of the West of England, BS16 1QY Bristol, U.K.

Corresponding author: Khoa Phung (khoa.phung@uwe.ac.uk)

ABSTRACT In software development, identifying software faults is an important task. The presence of faults not only reduces the quality of the software, but also increases the cost of development life cycle. Fault identification can be performed by analysing the characteristics of the buggy source codes from the past and predict the present ones based on the same characteristics using statistical or machine learning models. Many studies have been conducted to predict the fault proneness of software systems. However, most of them provide either inadequate or insufficient information and thus make the fault prediction task difficult. In this paper, we present a novel set of software metrics called Error-type software metrics, which provides prediction models with information about patterns of different types of Java runtime error. Particular, in this study, the ESM values consist of information of three common Java runtime errors which are Index Out Of Bounds Exception, Null Pointer Exception, and Class Cast Exception. Also, we propose a methodology for modelling, extracting, and evaluating error patterns from software modules using Stream X-Machine (a formal modelling method) and machine learning techniques. The experimental results showed that the proposed Error-type software metrics could significantly improve the performances of machine learning models in fault-proneness prediction.

INDEX TERMS Error type prediction, machine learning, software fault prediction, software metrics, stream X-machine.

I. INTRODUCTION

Software testing plays an essential role in Software Development Life Cycle (SDLC) as it ensures the quality and correctness of the software system. However, complete testing of a software system is practically not possible as it consumes an enormous amount of time and resources [1], [2]. Furthermore, faults are normally not distributed uniformly across software modules. Therefore, it is mostly inefficient or impossible to spend the same amount of testing resources and efforts on every individual software module of the system under test. To overcome this problem, Software Fault Prediction (SFP) has been introduced to early identify faulty software modules prior to the testing phase so that the allocation of testing resources can be economically optimised.

Software Fault Prediction can be done by training statistical and/or machine learning models on data that includes both dependent (faultiness) and independent (software metrics) variables. In a real-life enterprise context, the software metrics and faults information would be obtained directly from the source code and bug tracker. Over the last three decades, a great number of SFP research has been conducted with the use of various statistical and machine learning models such as Logistic Regression, Naïve Bayes, Support Vector Machine, Decision Tree, Random Forest, Multilayer Perceptron, etc. In the literature, there are many different sets of software metrics such as Object-oriented (OO) metrics with CK metrics suite [3], MOODS metrics suite [4], Bansiya metrics suite [5], etc.; or Traditional metrics with Size metrics (e.g., Function Points – FP, Source lines of code – SLOC, Kilo-SLOC – KSLOC), Quality metrics (e.g., Defects per FP after delivery, Defects per SLOC or KSLOC after delivery), System complex metrics [6], Halstead metrics [7], etc.

According to Rathore and Kumar [8], the definition of software fault proneness is very ambiguous and can be measured in different ways since a fault/error can happen in any phase of the SDLC and some faults remain undetected during the testing phase and forwarded to regular use in the field.

The associate editor coordinating the review of this manuscript and approving it for publication was Baoping Cai.

Menzies et al. [9] also pointed out that the techniques/approaches used for SFP have hit the ''performance ceiling''. Thus, simply applying different or better techniques will not guarantee an improved performance. In order to achieve better prediction performance, Menzies et al. [10] suggested the use of additional information when building SFP models. In order to eliminate ambiguity and thus, reduce the chance of having errors during the SDLC, Hierons et al. [2] suggested using formal methods. Over the last 35 years, formal methods have reached a sufficient level of maturity so that they can be practically applied in software development, especially safety critical software. A variety of different formal specification techniques (e.g., model-based formal specification [11], [12], [13], [14], [15], finite-state-based formal specification [16] [17], [18], [19], etc.) have been proposed and most of them are backed up by a wide variety of support tools.

In [20], we proposed a novel SFP approach for predicting error-type proneness in software modules using a streamlined process linking Stream X-Machine (a formal method) [21] and machine learning techniques. In particular, Stream X-Machine is used to model and generate test cases for different types of Java runtime errors, which will be employed to extract error-type data from the source codes. This data is subsequently added to the collected software metrics to form new training datasets. The experimental results showed that each individual error-type data (herein later referred to as *Error Specification Machine* values or *ESM values*) extracted using Stream X-Machine provided machine learning models with meaningful information about patterns of a particular runtime error and thus, boosted their performances in predicting error-type proneness of software modules. Although the experimental results showed that the new datasets could significantly improve the performances of machine learning models in terms of predicting error-type proneness, there were some limitations and threats that could potentially hinder the findings of the study. Also, in [20], we only explored each ESM value individually for the purpose of predicting error-type proneness. In reality, this approach can be time consuming, especially during the training process because different prediction models will have to be trained for different types of runtime error.

This paper is an extension of the study we proposed in [20]. In this research, we aim to overcome the existing limitations in the previous work [20] by:

1) Employing an industrial project that has accessible source code and is actively maintained to collect ESM values and software metrics for the experiments.
2) Developing an algorithm, called PSI-E, to extract ESM values automatically.
3) Further investigating the usefulness and potential issues of ESM values in software fault prediction by employing Stream X-Machine and machine learning techniques.

More importantly, in this work, we primarily aim to present a novel set of software metrics, called ***Error-type (ET) software metrics***, by considering the ESM values as one combined objective to capture the patterns of different types of Java runtime error (JRE). The motivation for proposing a new set of software metrics is because Menzies et al. [9] suggested that leveraging training data with more information content can potentially help to break the ''performance ceiling'' of SFP models. To the best of our knowledge, this is the first attempt in the literature to propose a set of software metrics that provides prediction models with information about error types.

In order to achieve the aim of the study, firstly, we need to investigate the relationships of ESM values with each other and with other software metrics. Note that in this study, we focus primarily on ***class-level*** software metrics because ESM values are measured at the class level. Secondly, we need to evaluate the impacts of ESM values on the performances of four different machine learning models including Decision Tree, Logistic Regression, Multilayer Perceptron, and Naïve Bayes. Here, the whole study aims at finding empirical evidences to answer the following research questions:

RQ1: *Do the ESM values have relationships with each other and with other class-level software metrics?*

The goal of this research question is to investigate if the multi-collinearity issue exists amongst the ESM values and other software metrics. Multi-collinearity can lead to the difficulty in distinguishing between the contributions of independent variables (features) to that of the dependent variable (label/output) since they may compete to explain similar variance.

RQ2: *Can the ESM values be used as* ***a new set of software metrics*** *to incorporate with other class-level software metrics to improve performances of SFP models?*

The goal of this research question is to investigate the usefulness, relevance, and potential issues of ESM values in software fault prediction when being incorporated with the other software metrics.

The rest of the paper is organised as follows. Section II contains related work. Section III describes the proposed methodology. Section IV outlines the results of the experiments and discusses the threats to validity. Section V concludes the paper and provides the directions for the future research.

II. RELATED WORK

Generally, in SFP, a prediction model is used to predict the fault-proneness of software modules. The process of SFP typically includes training a prediction model using the underlying properties of the software project, and subsequently using the prediction model to predict faults for unknown software projects.

In the literature, SFP studies often fall into one of the three categories: binary-class classification of faults, number of faults/fault density prediction, and severity of fault prediction [8]. The use of SFP models for binary-class classification has been extensively investigated by various researchers and has been the most popular approach [1], [8]. The systematic reviews and analysis of these studies can be found in [8], [22] [23], and [24]. With binary-class classification of fault proneness, researchers use public datasets (e.g., NASA [25] and PROMISE [26]) to train different machine learning/deep learning models such as Decision Tree, Naïve Bayes in Bayesian Learners, Multilayer Perceptron in Neural Networks, and Random Forest in Ensemble Learners. However, according to Rathore and Kumar [8], binary-class classification of software fault proneness is very ambiguous. This approach provides a very general picture of fault prediction because some modules are indeed more fault-prone and thus, require more attention than the others.

There are not many approaches that focused on predicting the fault density or fault severity of software modules. With the motivation of addressing the lack of information issue in SFP (discussed in Section I), in what follows, we only focus on analysing and reviewing the SFP approaches that could provide more useful information about software modules for software developers such as number of errors, severity of errors, or error-type proneness.

In 2005, Ostrand et al. [27] proposed an approach for predicting the number of faults and fault density using negative binomial regression (NBR) technique. The study was performed over the code of the file in the current release, and fault and modification history of the file from previous releases. The prediction aimed to identify top 20% of files with the highest percentage of the predicted number of faults. The analysis indicated that NBR-based models could produce accurate results for the number of faults and fault density predictions. A similar type of work was also reported in [28]. A few years later, Yu [29] conducted a deeper study to investigate the effectiveness of NBR in the context of Apache Ant software system. The results showed that NBR could not outperform the so called Binary Logistic Regression in predicting fault prone modules. However, the study demonstrated that (1) the performance of forward assessment is better than or at least the same as the performance of self-assessment; and (2) NBR is effective in predicting multiple faults in one module.

In another study, Afzal et al. [30] applied genetic programming (GP) for predicting the number of faults in a given project. The independent variables used to train the model were the weekly fault count data collected from three industrial projects. The empirical results indicated a significant accuracy rate of GP-based model for fault count prediction. Also, in [31], Rathore and Kumar presented an approach for predicting the number of faults using GP over several open-source software projects. The results demonstrated the significant accuracy and completeness of GP-based model in predicting the number of faults in software modules. In [32], Gao et al. presented a comprehensive analysis of five count models including Poisson Regression model (PR) [33], Zero-Inflated Poisson model (ZIP) [34], NBR model, Zero-Inflated Negative Binomial model (ZINB) [35], and Hurdle Regression model (HR) [36]. The results showed that ZINB and HR models produced better prediction accuracy for fault counts. Recently, Rathore and Kumar [1] explored the capability of Decision Tree Regression (DTR) for the number of faults prediction in two different scenarios, intra-release and inter-release predictions for a given software project. Five open-source software projects with their nineteen releases collected from the PROMISE data repository were chosen to perform the experimental study. The results indicated that DTR-based model could produce significant accuracy in both the considered scenarios.

Yang et al. [37] believed that predicting the exact number of faults in a software module is difficult due to noisy data that exists in the fault dataset; therefore, the authors introduced a learning-to-rank (LTR) approach to construct the SFP models by directly optimising the ranking performance. The LTR approach has two major benefits which are (1) more robust against noisy data and (2) unlike the other approaches which have to predict the number of faults in the software modules before ranking them, it provides a way to rank the level of severity of the software modules directly. The empirical results showed the effectiveness of the LTR approach for the ranking task.

Although the previously mentioned approaches could provide some more useful information such as number of errors and severity of errors in software modules, they relied on public datasets (NASA and PROMISE repositories) as the input for their investigations. Shepperd et al. [25] and Petric et al. [38] criticised these public datasets for containing erroneous data with unnecessary or incorrect information that could lead to deteriorate the classifier performance. Also, with the NASA dataset, the source code and information about metric tools used are not available; therefore, the anomalies in the dataset cannot be verified or fixed. Particularly, in our research, we cannot extract ESM values without the availability of the source code. Furthermore, most of the studies in the literature reported results without any inspection of the data and assumed that the datasets were of reasonable quality for prediction [8], [25]. This means that data quality issues (e.g., outlier, missing value, repeated value, redundant or irrelevant value) existing in public datasets were not addressed adequately in several SFP approaches.

In 2021 [20], we proposed a novel SFP approach using a streamlined process linking Stream X-Machine and machine learning techniques to predict if software modules are prone to having a particular type of runtime error in Java programs. Particularly, Stream X-Machine is used to model and generate test cases for different types of JREs, which will be employed to extract error-type data (ESM values) from the source codes. Subsequently, each individual ESM value was incorporated with other software metrics to form new training datasets. We then evaluated the performances of three machine learning algorithms (Support Vector Machine, Decision Tree, and Multilayer Perceptron) on error-type proneness prediction. The experimental results showed that the new datasets could significantly improve the performances of machine learning models in terms of predicting error-type proneness. To the best of our knowledge, this is the first attempt in the literature that utilises a formal method (Stream X-Machine) to specify and introduce error patterns (ESM values) to SFP models to improve their performances in terms of predicting error-type proneness in software modules. However, there were some limitations associated with the study. First, the size of the dataset was not substantial compared to other public datasets like NASA or PROMISE. Second, the ESM values were extracted manually from source codes, which could potentially lead to some unwanted mistakes. Finally and most importantly, the ESM values were evaluated individually for the purpose of predicting error-type proneness, which can potentially be time consuming during the training process because different prediction models will have to be built for different types of runtime error. Also, when being investigated individually, the usefulness and relevance of ESM values in SFP were not thoroughly examined and thus, potential issues such as multi-collinearity can be left unhandled.

In [20], we introduced the ESM values and evaluated them individually as a proof of concept. In this work, we further investigate the usefulness and relevance of ESM values in SFP. This work differs from [20] in several ways.

1) Firstly, we present an algorithm, called PSI-E (see subsection III-B), which is used to automatically extract ESM values from software modules. The development of the PSI-E helps to overcome potential issues that might happen due to the manual ESM values extraction process we discussed in [20].
2) Secondly, instead of being investigated individually as in [20], in this work, the ESM values are evaluated as *one combined objective*. This means that we consider all the ESM values as a new set of software metrics that can be incorporated with other software metrics in the literature. Subsequently, we investigate the impacts of this new set of software metrics on the performances of SFP models.
3) Thirdly, we use software metrics collected from the BugHunter Dataset proposed by Ferenc et al. [39] for conducting the experiments.
4) Fourthly, we analyse and address several data quality issues (e.g., missing values, redundant/irrelevant values, and multi-collinearity) existing in the training datasets, which we did not consider in the previous work [20] and many other SFP studies have also not handled adequately [8], [25].
5) Finally and most importantly, we propose the novel set of ***Error-type software metrics***.

III. METHODOLOGY

This section describes the proposed methodology to derive the ESM values and evaluate their effectiveness in SFP. The general process consists of five sequential steps, each of which is described in detail in the following subsections.

A. GENERATE TEST CASES FOR JREs

A runtime error might occur when one object operates an action on another object. Therefore, the general equation is as follows:

$$\text{A operates on B} \tag{1}$$

where:

- A and B are the 2 operands. They are either literals (e.g., literal string, literal integer, etc.) or references (e.g., string reference, integer reference, object, etc.)
- ''operates on'' denotes any action that A acts on B (e.g., Number A divides Number B → *ArithmeticException error might occur*; Array A accesses Bth element of itself → *IndexOutOfBoundsException error might occur*, etc.)

With the information about A, B, and ''operates on'' from Equation 1, we can determine the characteristics of an error such as *what the error is*, *how it happens*, and *in which context it happens*. Equation 1 is specified as a Stream X-Machine that represents a type of runtime error. This runtime error SXM is also known as an Error Specification Machine (ESM). Each type of runtime error can be represented as an ESM, which is a tuple of 8 elements as follows:

$$ESM_i = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0) \tag{2}$$

where:

- Σ is a finite set of input symbols,
- Γ is a finite set of output symbols,
- Q is a finite set of states,
- M is a (possibly) infinite set called memory,
- Φ is a finite set of partial functions (processing functions), which map memory-input pair to output-memory pairs, $\phi : M \times \Sigma \rightarrow \Gamma \times M$,
- F is the next-state partial function, $F : Q \times \Phi \rightarrow Q$,
- $q_0 \in Q$ and $m_0 \in M$ are the initial state and memory, respectively,
- $i \in E$, where E is a finite set of different types of JRE. `E = {Arithmetic, NullPointer, ClassCast, IndexOutOfBounds, ...}`.

A Stream X-Machine can be thought as a finite automaton with the arcs labelled by functions from the type. The automaton $A_Z = (\Phi, Q, F, I, T)$ is called the associated finite automaton (FA) of a Stream X-Machine and is usually described by a state-transition diagram. In [20], we have presented a state-transition diagram of the ESM with 5 states (state 1, state 2, state 3, state 4, and state 5) and 4 processing functions (getA, getOperation, getB, and getErrorLabel). Each ESM_i has a corresponding FA as $A_{ESM_i} = (\Phi, Q, F, I, T)$.

One of the great benefits of using SXM to specify a system is its associated testing method which was initially developed for deterministic SXM [19], [40] and was further extended to non-deterministic SXM [41] and communicating SXM [42]. Under certain design for test conditions, this method can produce a test suite that can be used to verify

TABLE 1. Example test cases for arithmetic exception.

A	Operation	B	Error Label
Literal Number	Divide	Literal Number	Arithmetic Error
Literal Number	Divide	Reference Number	Arithmetic Error
Literal Number	Divide	Reference Object	Arithmetic Error
Reference Number	Divide	Literal Number	Arithmetic Error
Reference Number	Divide	Reference Number	Arithmetic Error
Reference Number	Divide	Reference Object	Arithmetic Error
Reference Object	Divide	Literal Number	Arithmetic Error
Reference Object	Divide	Reference Number	Arithmetic Error
Reference Object	Divide	Reference Object	Arithmetic Error

the correctness of the implementation under test, provided that the processing functions of the SXM specification have been correctly implemented [43]. In this research, since each ESM is a Stream X-Machine specification, test cases for each ESM can be generated by applying the Stream X-Machine testing method which is the *state-counting approach* [43], [44]. This means that each type of runtime error will have a corresponding set of test suite which is calculated using Equation 3 [44] as follows:

$$U_i = \bigcup_{q \in Q_r} \{p_q\} prefix(V(q)) W_s \tag{3}$$

In order to construct the test suite U_i, we first need to select two sets of sequences of processing functions, S_r and W_s, and of a relation d_s on the states of ESM_i as follows:

- S_r is a non-empty set of realisable sequences such that no state in ESM_i is reached by more than one sequence in S_r,
- p_a is a path in A_{ESM_i} where $p_a = \phi_1 \cdots \phi_k \in \Phi^*$,
- The definition of the set $V(q)$ can be found in [44],
- W_s is a finite set that separates between separable states of ESM_i. W_s is required to be non-empty,
- $d_s : Q \leftrightarrow Q$ is a relation on the states of ESM_i that satisfies the following conditions: for every two states $q_1, q_2 \in Q$, if $(q_1, q_2 \in d_s)$ then q_1 and q_2 are separated by W_s. The relation d_s identifies pairs of states that are known to be separated by W_s,
- The maximal set $Q_1 \cdots Q_j$ of states of ESM_i that are known to be pairwise separated by W_s,
- $i \in E$, where E is a finite set of different types of JRE. `E = {Arithmetic, NullPointer, ClassCast, IndexOutOfBounds, ...}`,
- U_i is a set of test cases associated with each type of JRE. For instance, *ArithmeticException* can be represented as *ESMArithmetic*, which can subsequently generate test cases (an example is shown in Table 1) for this error.

Based on Equation 3, in [45] and [46], we proposed ***T-SXM*** which is a modelling tool developed based on the concept of Stream X-Machine and state-counting approach for automatic test generation. The T-SXM tool has been used to model chronic diseases (e.g., Type II Diabetes) with promising results in [46] and [47]. In this study, we employed the T-SXM tool to model and generate test cases for three different types of JRE including *IndexOutOfBoundsException*, *ClassCastException*, and *NullPointerException*.

B. EXTRACT ESM VALUES

In a software module, an ESM value for a JRE (e.g., `Index Out Of Bounds Exception`) is the sum of all the code patterns that satisfy the two following conditions:

1) Match the pattern represented in Equation 1,
2) Match one or more test cases (generated from Step 1) of that particular error.

In other words, ESM values for different types of JRE for software modules in a software system can be represented as follow:

$$ESM\_values = \bigcup_{m \in S} \sum_{i=1}^{L} \sum_{j=1}^{K} matched\_pattern \tag{4}$$

where:

1) m is a software module in the software system S,
2) i is the i^{th} line in the total L lines of code in software module m,
3) j is the j^{th} code pattern in the total K code patterns in line i^{th},
4) $matched\_pattern$ is the pattern in line i^{th} that satisfies the 2 conditions mentioned above.

In this study, we introduced a new algorithm, called ***PSI-E***, that has been developed based on Equation 4 and integrated into the T-SXM tool so that it can interpret and extract ESM values for different types of JREs from software modules. The PSI-E algorithm has been built on top of the PSI (Program Structure Interface) - a layer in IntelliJ Platform [48] which is responsible for parsing files and creating syntactic and semantic code model.

C. EXTRACT SOFTWARE METRICS

In this study, software metrics have been imported from the BugHunter Dataset and re-validated using *MetricsReloaded* [49], which is a tool for obtaining/measuring software metrics from source codes. The software metrics we used in this study are at the ***class level*** and can be found in Table 2.

The reasons we employed the BugHunter Dataset are (1) the software metrics were collected from large industrial projects which are still actively maintained; (2) the source codes of the projects are publicly available via GitHub [50] for us to verify the correctness of the collected software metrics and extract the ESM values; and (3) the BugHunter Dataset collects software metrics based on before-fix and after-fix snapshots of the source code elements that were affected by bugs whilst leaving the source code elements that were not affected by bugs untouched. This approach is useful to capture the changes in software metrics when a bug is being fixed. The empirical evaluations showed that the dataset can be used for further investigations such as bug prediction.

D. CREATE NEW TRAINING DATASETS

To address the two research questions outlined in Section I, in this step, we have evaluated performances of machine learning models on two datasets.

TABLE 2. Class-level software metric used in this study.

Abbreviation	Full name
CLOC	Comment Lines of Code
LOC	Lines of Code
LLOC	Logical Lines of Code
NL	Nesting Level
NLE	Nesting Level Else-If
NII	Number of Incoming Invocations
NOI	Number of Outgoing Invocations
CD	Comment Density
DLOC	Documentation Lines of Code
TCD	Total Comment Density
TCLOC	Total Comment Lines of Code
NOS	Number of Statements
TLOC	Total Lines of Code
TLLOC	Total Logical Lines of Code
TNOS	Total Number of Statements
PDA	Public Documented API
PUA	Public Undocumented API
LCOM5	Lack of Cohesion in Methods 5
WMC	Weighted Methods per Class
CBO	Coupling Between Object classes
CBOI	Coupling Between Object classes Inverse
RFC	Response set For Class
AD	API Documentation
DIT	Depth of Inheritance Tree
NOA	Number of Ancestors
NOC	Number of Children
NOD	Number of Descendants
NOP	Number of Parents
NA	Number of Attributes
NG	Number of Getters
NLA	Number of Local Attributes
NLG	Number of Local Getters
NLM	Number of Local Methods
NLPA	Number of Local Public Attributes
NLPM	Number of Local Public Methods
NLS	Number of Local Setters
NM	Number of Methods
NPA	Number of Public Attributes
NPM	Number of Public Methods
NS	Number of Setters
TNA	Total Number of Attributes
TNG	Total Number of Getters
TNLA	Total Number of Local Attributes
TNLG	Total Number of Local Getters
TNLM	Total Number of Local Methods
TNLPA	Total Number of Local Public Attributes
TNLPM	Total Number of Local Public Methods
TNLS	Total Number of Local Setters
TNM	Total Number of Methods
TNPA	Total Number of Public Attributes
TNPM	Total Number of Public Methods
TNS	Total Number of Setters

The first dataset is the ***Software metrics dataset***. In this dataset, the independent variables are the software metrics extracted from the software modules and the dependent variable indicates the fault proneness of the software modules. Table 3 demonstrates an example of the Software metrics dataset.

The second dataset is the ***Full dataset*** (the new dataset). In this dataset, the independent variables are the *ESM values* and software metrics extracted from the software modules and the dependent variable indicates the fault proneness of the software modules. The *ESM values* present in this dataset include *ESM IndexOutOfBounds* (corresponding to the Index Out Of Bounds Exception), *ESM NullPointer* (corresponding to the Null Pointer Exception), and *ESM ClassCast* (corresponding to the Class Cast Exception). Table 4 shows an example of the Full dataset. The difference between the two datasets is that in addition to the software metrics, the Full dataset also consists of the ESM values.

The Software metrics dataset is used to reproduce the results that the authors of the BugHunter Dataset [39] had achieved so that we could conduct experiments on the Full dataset with no biases/doubts towards the ESM values. The Full dataset is used to investigate the impacts of ESM values on the performances of machine learning models.

Before starting the training process, the two datasets have been preprocessed carefully to resolve the issues of missing values and irrelevant features. The details will be further elaborated in subsection IV-B.

E. EVALUATIONS

Firstly, machine learning models are trained and evaluated on the Software metrics dataset. Secondly, machine learning models are trained and evaluated on the Full dataset. The differences in performances of machine learning models on the two datasets would indicate the impacts of ESM values. To evaluate the performances of the trained models, we use Accuracy, Precision, Recall, and F1-score metrics. We then compare our machine learning models with the ones in [39]. The reason we use the results recorded by Ferenc et al. [39] as the benchmark is that in our experiment, the datasets are created based on the BugHunter Dataset.

IV. EXPERIMENTS, RESULTS, AND DISCUSSIONS

This section describes the details of the experiments and their results. Subsequently, we present how the empirical results can address the research questions. Finally, we discuss some limitations and threats that could potentially hinder the findings of the research.

A. SOURCE OF DATASET

As mentioned earlier, software metrics used in this experiment have been imported from the BugHunter Dataset [39]. The primary difference between the BugHunter Dataset and other traditional datasets (e.g., NASA and PROMISE repositories) is that instead of gathering the characteristics of all source code elements (fault prone or not fault prone) at only one or more pre-selected release versions of the code, it captures the faulty and the fixed states of the same source code

TABLE 3. Example of the software metrics dataset.

Software modules	Independent variables (Input variables)				Dependent variable (Output variable)
	LOC	LCOM5	WMC	...	Fault proneness
org.junit.runners.BlockJUnit4ClassRunner	349	1	52	...	Yes
org.junit.runners.ParentRunner	377	2	55	...	Yes
org.junit.runner.notification.RunNotifier	53	1	4	...	No
org.junit.runners.Parameterized	41	2	8	...	No
org.junit.Assert	760	5	70	...	Yes

TABLE 4. Example of the full dataset.

Software modules	Independent variables (Input variables)							Dependent variable (Output variable)
	ESM IndexOutOfBounds	ESM NullPointer	ESM ClassCast	LOC	LCOM5	WMC	...	Fault proneness
org.junit.runners.BlockJUnit4ClassRunner	5	44	2	349	1	52	...	Yes
org.junit.runners.ParentRunner	7	47	0	377	2	55	...	Yes
org.junit.runner.notification.RunNotifier	2	8	6	53	1	4	...	No
org.junit.runners.Parameterized	10	7	2	41	2	8	...	No
org.junit.Assert	0	20	4	760	5	70	...	Yes

regardless of the release versions. The process of collecting software metrics for the BugHunter Dataset is described in detail in [39].

Menzies et al. [10] suggested that researchers should concentrate on finding solutions that work best for the groups of related projects rather than trying to seek general solutions that can be applied to many projects. Therefore, in this study, instead of training prediction models on all 15 projects (see detailed descriptions in [39]) of the BugHunter Dataset, we focused mainly on the JUnit project, which is a Java framework for writing unit tests. The JUnit project is an industrial project and has been actively maintained by the developers. At the time of conducting the experiment in this research, the project had 734 entries in total including 92 files, 216 classes, 426 methods, and 22,701 lines of code. Out of the 734 entries, 286 are not related to test code (43 files, 77 classes, and 166 methods). Therefore, in the experiment, 139 class entries have been used to extract ESM values and software metrics at class level. Note that the ESM values and class-level software metrics were extracted throughout multiple versions of the classes based on the commits on GitHub.

B. DATA PREPROCESSING

The authors of the BugHunter Dataset trained the machine learning models on this JUnit dataset and recorded the results which are shown in Table 5. As can be seen in Table 5, the performances of Decision Tree and Logistic Regression are around 0.5, which is not very high. According to Gray et al. [51], there are several quality issues (e.g., missing values, redundant and irrelevant values, outliers, multi-collinearity, etc.) associated with software fault datasets which researchers need to properly handle before using them to construct SFP models. Since the authors of the BugHunter Dataset did not mention how they preprocessed the data in their experiment, we had to investigate it to ensure that the data quality issues were adequately resolved before carrying on with the training process.

TABLE 5. Performances of decision tree and logistic regression on the JUnit dataset of the BugHunter dataset.

	F1-score	Precision	Recall
Decision Tree	0.5339	0.5345	0.5344
Logistic Regression	0.5872	0.5911	0.5893

Firstly, we discovered that the dataset consists of a number of constant features (software metrics) namely *Warning Blocker, Warning Critical, Warning Info, Warning Major, Android Rules, Basic Rules, Brace Rules, Code Size Rules, Comment Rules, Clone Implementation Rules, Controversial Rules, CouplingRules, Empty Code Rules, Finalizer Rules, Import Statement Rules, J2EE Rules*. These features contain only value 0 in all rows, which can potentially lead to heavy bias for machine learning algorithms. These features were not mentioned in the paper [39]; however, they exist in the actual BugHunter Dataset.

Furthermore, most of the features in the dataset are highly left skewed, which means they are not normally distributed. Also, many features (software metrics) are highly correlated with each other, which will potentially lead to the multi-collinearity issue. Multi-collinearity is a common phenomenon in statistics and happens when there are high correlations among independent variables (features). According to Zainodin et al. [52], multi-collinearity can lead to the difficulty in distinguishing between the contributions of these independent variables (features) to that of the dependent variable (label/output) since they may compete to explain similar variance. In general, correlated features do not improve performances of machine learning models and thus, removing multi-collinearity is recommended by Li et al. [53] to reduce dimensions, remove irrelevant data, increase learning accuracy, and improve result comprehensibility.

In order to further examine the multi-collinearity issue in the Full dataset, we employed the Variance Inflation Factor (VIF) technique, which is used to measure the degree of collinearity present for each factor. VIF indicates how much of the estimated coefficient increases is due to collinearity independent variables. In particular, VIF reports how much of a regressor's variability is explained by the rest of the regressors in the model due to correlation among those regressors [54]. VIF can be calculated by Equation 5 as follows:

$$VIF_i = \frac{1}{1 - R_i^2} \tag{5}$$

where, R_i^2 is the coefficient of determination obtained by fitting a regression model for the i^{th} independent variable on the other independent variables.

The values of VIF scores in the Full dataset range from 2.07 (ESM ClassCast) to infinity (LDC, LLDC, TNLS and NLS). According to Craney and Surles [54], although no formal cutoff value or method exists to determine when a VIF score is too large, the typical suggestions for a cutoff point are less than 5 or less than 10. Based on the suggestions of Craney and Surles [54], only 9 out of 61 features in the Full dataset satisfy the cutoff point namely *ESM ClassCast* (2.07), *ESM IndexOutOfBounds* (2.61), *ESM NullPointer* (6.55), PUA (6.70), NII (8.05), NL (8.10), LCOM5 (8.34), and CBO (9.12). NOI (16.49) is also considered to be relatively good since its VIF score is much closer to 10 compared to the VIF scores of the other features (software metrics). From the VIF analysis, it can be seen that apart from PUA, NII, NL, LCOM5, CBO, and NOI, the other software metrics have very high VIF scores ($\approx 40 \rightarrow \infty$), which means if all these features are kept in the datasets for the training process, they will compete with each other to explain the same variance and thus, affect negatively to the performances of machine learning models. Therefore, it can be concluded that the multi-collinearity exists in the Full dataset and the Software metrics dataset and thus, needs to be resolved.

With multiple issues existing in the Full dataset and the Software metrics dataset as mentioned above, before carrying on with the training process, we preprocessed the data with the following steps:

1) Remove constant features (software metrics) (*these are the features that contain only value 0 in all rows mentioned earlier*). The total number of features remaining in the Full dataset and Software metrics dataset are 61 and 58, respectively.
2) Split the Full dataset and Software metrics dataset into training (80%) and testing (20%) sets accordingly.
3) Scale the features to bring them into the same value range (0, 1).
4) Normalise the features to make them more normally distributed by applying Box-Cox transformation algorithm [55]. According to Osborne [56], Box-Cox applies a range of power transformations (rather than the traditional square root, log, and inverse) to easily improve the data normalisation process for both positively and negatively skewed features.
5) Apply feature selection using Linear Support Vector Classification (Linear SVC) with Lasso (Least Absolute Shrinkage and Selection Operator) regularisation to remove multi-collinearity. According to Fonti and Belitser [57], feature selection is a process that chooses a reduced number of explanatory variable to describe a response variable. Lasso is a powerful method for regularisation and feature selection. The method applies a shrinking (regularisation) process to penalise the coefficients of the regression variables and shrinks some of them to zero [57]. After the shrinking process, the variables that still have a non-zero coefficient are selected to be part of the model. Feature selection is applied to reduce irrelevant features (features that do not add any information to the dataset or features that are highly correlated with each other) and reduce overfitting. In [39], the authors did not explicitly mention if they had applied any feature selection method on the BugHunter Dataset or not. Therefore, we decided to conduct the experiment with both scenarios where feature selection would be applied and would not be applied. Note that, in the Full dataset, feature selection would only be applied on the *software metrics* variables.

C. EXPERIMENTAL RESULTS

Table 6 demonstrates the F1-score, Precision, Recall, and Accuracy rates of the four machine learning algorithms (DT, LR, MLP, and NB) on the two datasets (Full dataset and Software metrics dataset) in two scenarios:

1) Feature selection was not applied: In this experiment, the two datasets were used to train the four machine learning models without being pre-processed with feature selection. This experiment was conducted to reproduce the results recorded by Ferenc et al. [39] to investigate if the dataset had been adequately pre-processed to resolve data quality issues mentioned in subsection IV-B.
2) Feature selection was applied: In this experiment, the two datasets had been pre-processed with feature selection before being used to train the four machine learning models.

As can be seen from Table 6, when *feature selection was not applied*, the F1-scores of Decision Tree and Logistic Regressions are 0.3750 and 0.5514, respectively on the Software metrics dataset, which are quite close to the results recorded by Ferenc et al. [39]. This indicates that we were able to reproduce the results achieved in [39].

When *feature selection was applied*, the F1-scores were significantly boosted for DT (increased by 42% and 77% for the Software metrics dataset and Full dataset, respectively),

TABLE 6. Experimental results.

Machine learning algorithms	Feature selection	Datasets	F1-Score	Precision	Recall	Accuracy
Decision Tree (DT)	Not applied	Full dataset	0.3529	0.3750	0.3333	0.4500
		Software metrics dataset	0.3750	0.4286	0.3333	0.5000
	Applied	Full dataset	0.6250	0.7143	0.5550	0.7000
		Software metrics dataset	0.5333	0.6666	0.4444	0.6500
Logistic Regression (LR)	Not applied	Full dataset	0.5714	0.8230	0.4444	0.7156
		Software metrics dataset	0.5514	0.8000	0.4344	0.7000
	Applied	Full dataset	0.7143	0.9700	0.5555	0.8000
		Software metrics dataset	0.6154	0.9100	0.4344	0.7500
Multilayer Perceptron (MLP)	Not applied	Full dataset	0.4615	0.7500	0.3333	0.6500
		Software metrics dataset	0.4615	0.7500	0.3333	0.6500
	Applied	Full dataset	0.5714	0.8000	0.4444	0.7000
		Software metrics dataset	0.4615	0.7500	0.3333	0.6500
Naïve Bayes (NB)	Not applied	Full dataset	0.5882	0.6250	0.5556	0.6500
		Software metrics dataset	0.5882	0.6250	0.5556	0.6500
	Applied	Full dataset	0.6250	0.7143	0.5556	0.7000
		Software metrics dataset	0.5882	0.6250	0.5556	0.6500

LR (increased by 12% and 25% for the Software metrics dataset and Full dataset, respectively), MLP (increased by 24% for the Full dataset, and NB (boosted by 6% for the Full dataset). This means that in [39], the data quality issues were not adequately addressed.

With the results from the two experimented scenarios, the performances of the four machine learning models have been boosted significantly after feature selection was applied to the training datasets. As a result, it can be inferred that the multi-collinearity issue existing in the software metrics of both datasets negatively affects the performances of the machine learning models by introducing harmful biases and difficulties to distinguish the true effect of each independent variable. Due to this multi-collinearity issue, when feature selection was not applied, there were no clear differences between performances of machine learning models when being trained on the two datasets. For instance, with the DT model, F1-scores were 0.3750 and 0.3529 for Software metrics dataset and Full dataset, respectively. It can be seen that these two values are close to each other and are less than 0.5, which is very low. As a result, we could not confirm if the ESM values actually had effects/impacts on the learning process or not.

On the other hand, when feature selection was applied, the multi-collinearity issue has been resolved and thus, the effects of features (including ESM values) could be interpreted better by the machine learning algorithms. The results show that the F1-scores of the four machine learning models were significantly boosted as discussed above. More importantly, the F1-scores of the models trained on the Full dataset were always higher than the F1-scores of the same models trained on the Software metrics dataset. The highest F1-score was 0.7143 and the highest Accuracy was 0.8 from the Logistic Regression model trained on the Full dataset. For the other models trained on the Full dataset, the F1-score was also above 0.5 and the Accuracy was around 0.7. These results are appropriate for prediction models as according to Rathore and Kumar [8], the average accuracy of all SFP techniques was between 70% and 85%.

From the F1-score and Accuracy rates of the four machine learning models on the two datasets, it can be inferred that the ESM values provided the machine learning models with useful information about error patterns and thus, improved their performances.

D. ANSWER TO RESEARCH QUESTIONS

By investigating the correlations of all the features in the Full dataset as discussed in subsection IV-B, it can be inferred that *ESM IndexOutOfBounds*, *ESM NullPointer*, and *ESM ClassCast* do not have correlations with each other and with other software metrics. Also, the VIF scores of these ESM values are the lowest in the Full dataset (2.07 for *ESM ClassCast*, 2.61 for *ESM IndexOutOfBounds*, and 6.55 for *ESM NullPointer*) and are lower than the cutoff point suggested by Craney and Surles [54] (less than 5 or less than 10). From the empirical results, we could address research question RQ1 outlined in Section I by concluding that the ESM values do not have correlations with each other and with other software metrics.

From the empirical results, it can be seen that with the multi-collinearity issue being properly handled by applying feature selection, the ESM values in the Full dataset added more useful information to the machine learning models and thus, helped to improve their performances. Therefore, we could conclude that the ESM values can be used as *a set of software metrics* to incorporate with software metrics to improve fault-proneness prediction. As a result, the research question RQ2 outlined in Section I can be answered.

By fully addressing research questions with empirical evidence, we can infer that the ESM values do not compete with each other and other software metrics to explain similar variances. Also, ESM values can be used as *a set of software metrics* to provide machine learning models with more useful information about error patterns and thus, improve their performances. As a result, we conclude that ESM values can be used as a novel set of software metrics, called ***Error-type software metrics***, to incorporate with other software metrics in the context of software fault prediction.

E. COMPARE WITH OTHER SOFTWARE METRICS

According to Colakoglu et al. [58], class-level metrics that have been mostly used in SFP studies are CK metrics (WMC, RFC, LCOM, CBO, DIT, NOC), MC (Method/Message Complexity), CWC (Coupling Weight for a Class), AC (Attribute Complexity), CLC (Class Complexity), AMCC (Average Method Complexity per Class), ACC (Average Class Complexity), ACF (Average Coupling Factor) and AAC (Average Attributes Per Class). However, these software metrics can only be applied in particular application domains. For instance, although Rathore and Kumar [8] suggested that object-oriented metrics, particular CK metrics, are good predictors of fault severity in software project, in [59], Bansal argued that the DIT and LCOM metrics are unsuitable to measure quality and complexity for object-oriented design; whilst NOC cannot be used at all for fault-proneness prediction. As a result, when conducting SFP studies, there is always a need to select the best combination of metrics for each different application domain for the given application context.

On the other hand, the traditional sizing metrics such as LOC, SLOC, and KSLOC are considered as generic source code metrics and can be used in any application domains. These metrics have also been proven to be related to fault proneness in many SFP studies [8], [58]. The combination of LOC with WMC and RFC can be utilised to predict change-prone classes and reduce the cost of testing for various types of software projects [60].

Similar to the sizing metrics, the newly proposed Error-type metrics are derived based on the error type models as discussed in subsection III-A; therefore, they can be applied in any application domain since the runtime errors will always happen in the same manners. Also, the experimental results showed that after applying feature selection on the training data, the Error-type metrics, PUA, NII, NL, LCOM, CBO, and NI can make a good combination for fault-proneness prediction in object-oriented software (e.g., JUnit) with the accuracy rate of 80%.

One of the concerns with SFP is the continuous evolution of source codes. For instance, an SFP model is built using a set of metrics and subsequently, it is used to predict fault-proneness in a given software project. However, some of the faults are fixed afterwards. Thus, the software modules have evolved to accommodate the changes. However, in several scenarios, the values of the used set of metrics remained unchanged. As a result, if the built SFP model is reused, it will detect the same code areas as fault-prone. According to Rathore and Kumar [8], this is a well-known problem in SFP. To overcome this issue, it is necessary to select the software metrics based on the development process and self-adapting measurements that capture already fixed faults. Nachiappan et al. [61] and Matsumoto et al. [62] proposed different sets of metrics such as software change metrics, file status metrics, and developer metrics to build SFP models. However, these metrics rely on inputs that consist of human involvement. For example, software change metrics only work when changes are committed whenever the developer considers them as ready for release. On the other hand, our proposed Error-type metrics can automatically capture the differences in code/error patterns between two versions of the software project and do not require human involvement. As a result, the Error-type metrics can be used to construct SFP models that capture the buggy and the fixed states of the same source code elements from the narrowest timeframe regardless of release versions.

Additionally, according to Chhillar et al. [63], most software failures these days are caused by lack of testing of non-functional parameters such as security and performance. Also, it has become more challenging with the rapid development of trending technologies such as Internet of Things (IoT), Artificial Intelligence (AI), Machine Learning, and Robotics. However, we believe that the newly proposed Error-type metrics can be useful for predicting fault-proneness in these fields since the runtime errors will also happen in the same manners with traditional software.

In general, the contributions of the newly proposed Error-type software metrics based on empirical results are as follows.

- Error-type metrics are generic source code metrics and can be used to construct SFP models for a wide variety of application domains.
- Error-type metrics can capture the buggy and the fixed states of the same source code elements from the narrowest timeframe regardless of release versions.
- Error-type metrics are good predictors of fault-proneness.

F. LIMITATIONS AND THREATS TO VALIDITY

Although the results are promising to confirm the usefulness of ESM values in improving performances of machine learning models in predicting fault proneness, there are still some threats that could affect the findings of the proposed approach.

Firstly, the ESM values were collected automatically using the T-SXM tool. As a result, errors can potentially occur within the developed PSI-E algorithm when it encounters a code pattern that is abnormal. However, manually validating all code patterns in a large project would have been an enormous task.

Secondly, we conducted the experiment with three ESM values corresponding to the most three common JREs namely Index Out Of Bounds Exception, Null Pointer Exception, and Class Cast Exception. At the time of conducting the experiment, we could not find any correlation between ESM values with each other and with other class-level software metrics. However, in future research, when new ESM values are introduced, the multi-collinearity issue can potentially happen within the ESM values. As a result, the findings we have in this study could be hindered.

Thirdly, the experiment was conducted on one project, which is JUnit. Although JUnit is a fairly large Java project and we obtained the ESM values and software metrics from various versions of the project, it is still better to include the analysis results from other different projects as different projects possess different error patterns, which means one ESM value (e.g., *ESM Arithmetic*) might contribute to the error patterns in one project but it might be a noise or an outlier in another project. In this study, we have not investigated the issue of outlier of the ESM values.

Finally, although through the experiments, we could infer that the ESM values helped to improve performances (F1-score and Accuracy rates) of machine learning models, the low values of Recall was a concern for us. The Recall values from the four machine learning models in all experiments fluctuated between 0.3333 and 0.5556. This could be caused by the issue of skewed dataset we mentioned in subsection IV-B. Although we applied the Box-Cox transformation algorithm to make the dataset more normally distributed, there are still more rooms for further improvement.

V. CONCLUSION AND FUTURE RESEARCH

In this paper, we have investigated the usefulness and potential issues of the ESM values, which were initially introduced in [20], as a new set of software metrics, called Error-type software metrics, to improve performances of machine learning models in fault-proneness prediction. We have also proposed a methodology for modelling, extracting, and evaluating the ESM values using Stream X-Machine and machine learning techniques. We conducted the experiments with four machine learning models (Decision Tree, Logistic Regression, Multilayer Perceptron, and Naïve Bayes) on the JUnit project whose software metrics were imported from the BugHunter Dataset. Before carrying on with the training process, we had carefully investigated the dataset to address the data quality issues (e.g., redundant and irrelevant values, and multi-collinearity). The results showed that the newly proposed Error-type software metrics helped to improve performances of machine learning models by providing them with useful information about error patterns. Although there are still several limitations that could potentially hinder the findings of the study, the promising results from the experiments enabled us to conclude that the ESM values can be used a new set of ***Error-type software metrics*** that can be incorporated with software metrics to improve performances of SFP models in predicting fault-proneness.

In future research, we will introduce more ESM values (e.g., *ESM Arithmetic*, *ESM NumberFormat*, etc.) to the ***Error-type software metrics*** to capture more runtime error patterns in Java projects. We will also conduct experiments on more industrial projects in different application domains to consolidate our findings and address the threats outlined in subsection IV-F.

REFERENCES

[1] S. S. Rathore and S. Kumar, "A decision tree regression based approach for the number of software faults prediction," *ACM SIGSOFT Softw. Eng. Notes*, vol. 41, no. 1, pp. 1–6, Feb. 2016.

[2] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. Simons, S. Vilkomir, M. R. Woodward, and H. Zedan, "Using formal specifications to support testing," *ACM Comput. Surv.*, vol. 41, no. 2, pp. 1–76, 2009.

[3] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994.

[4] R. Harrison, S. J. Counsell, and R. V. Nithi, "An evaluation of the MOOD set of object-oriented software metrics," *IEEE Trans. Softw. Eng.*, vol. 24, no. 6, pp. 491–496, Jun. 1998.

[5] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Trans. Softw. Eng.*, vol. 28, no. 1, pp. 4–17, Jan. 2002.

[6] T. J. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.

[7] M. H. Halstead, *Elements of Software Science* (Operating and Programming Systems Series). Amsterdam, The Netherlands: Elsevier, 1977.

[8] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," *Artif. Intell. Rev.*, vol. 51, no. 2, pp. 255–327, Feb. 2019.

[9] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, "Implications of ceiling effects in defect predictors," in *Proc. 4th Int. Workshop Predictor Models Softw. Eng.*, May 2008, pp. 47–54.

[10] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, "Local vs. global models for effort estimation and defect prediction," in *Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2011, pp. 343–351.

[11] J. M. Spivey, *Understanding Z: A Specification Language and Its Formal Semantics*, vol. 3. Cambridge, U.K.: Cambridge Univ. Press, 1988.

[12] J. M. Spivey, *The Z Notation: A Reference Manual*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1992.

[13] L. Steels, "Components of expertise," *AI Mag.*, vol. 11, no. 2, p. 28, 1990.

[14] B. J. Wielinga, A. T. Schreiber, and J. A. Breuker, "KADS: A modelling approach to knowledge engineering," *Knowl. Acquisition*, vol. 4, no. 1, pp. 5–53, 1992.

[15] J.-R. Abrial and A. Hoare, *The B-Book: Assigning Programs to Meanings*, vol. 1. Cambridge, U.K.: Cambridge Univ. Press, 1996.

[16] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines—A survey," *Proc. IEEE*, vol. 84, no. 8, pp. 1090–1123, Aug. 1996.

[17] *Specification and Description Language (SDL)*, document ITU-T R Z.100, ITU Telecommunication Standardization Sector, Geneva, Switzerland, 2002.

[18] D. Harel and E. Gery, "Executable object modeling with statecharts," in *Proc. IEEE 18th Int. Conf. Softw. Eng.*, Mar. 1996, pp. 246–257.

[19] M. Holcombe and F. Ipate, *Correct Systems: Building a Business Process Solution*. Springer, 2012.

[20] K. Phung, E. Ogunshile, and M. Aydin, "A novel software fault prediction approach to predict error-type proneness in the Java programs using stream X-machine and machine learning," in *Proc. 9th Int. Conf. Softw. Eng. Res. Innov. (CONISOFT)*, Oct. 2021, pp. 168–179.

[21] D. Dranidis, K. Bratanis, and F. Ipate, "JSXM: A tool for automated test generation," in *Proc. 10th Int. Conf. Softw. Eng. Formal Methods (SEFM)*, Thessaloniki, Greece. Berlin, Germany: Springer, Oct. 2012, pp. 352–366.

[22] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Appl. Soft Comput.*, vol. 27, pp. 504–518, Feb. 2015.

[23] H. Alsolai and M. Roper, "A systematic literature review of machine learning techniques for software maintainability prediction," *Inf. Softw. Technol.*, vol. 119, Mar. 2020, Art. no. 106214.

[24] A. Kumar and A. Bansal, "Software fault proneness prediction using genetic based machine learning techniques," in *Proc. 4th Int. Conf. Internet Things, Smart Innov. Usages (IoT-SIU)*, Apr. 2019, pp. 1–5.

[25] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the NASA software defect datasets," *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, Sep. 2013.

[26] J. S. Shirabad and T. J. Menzies, "The PROMISE repository of software engineering databases," School Inf. Technol. Eng., Univ. Ottawa, Ottawa, ON, Canada, 2005. [Online]. Available: http://promise.site.uottawa.ca/SERepository

[27] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Trans. Softw. Eng.*, vol. 31, no. 4, pp. 340–355, Apr. 2005.

[28] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Where the bugs are," *ACM SIGSOFT Softw. Eng. Notes*, vol. 29, no. 24, pp. 86–96, 2004.

[29] L. Yu, "Using negative binomial regression analysis to predict software faults: A study of apache ant," 2012. [Online]. Available: https://scholarworks.iu.edu/dspace/handle/2022/20466

[30] W. Afzal, R. Torkar, and R. Feldt, "Prediction of fault count data using genetic programming," in *Proc. IEEE Int. Multitopic Conf.*, Dec. 2008, pp. 349–356.

[31] S. S. Rathore and S. Kumar, "Predicting number of faults in software system using genetic programming," *Proc. Comput. Sci.*, vol. 62, pp. 303–311, Jan. 2015.

[32] K. Gao and T. M. Khoshgoftaar, "A comprehensive empirical study of count models for software fault prediction," *IEEE Trans. Rel.*, vol. 56, no. 2, pp. 223–236, Feb. 2007.

[33] P. Consul and F. Famoye, "Generalized Poisson regression model," *Commun. Statist., Theory Methods*, vol. 21, no. 1, pp. 89–109, 1992.

[34] M. Xie, B. He, and T. N. Goh, "Zero-inflated Poisson model in statistical process control," *Comput. Statist. Data Anal.*, vol. 38, no. 2, pp. 191–201, Dec. 2001.

[35] M. Ridout, J. Hinde, and C. G. B. Demétrio, "A score test for testing a zero-inflated Poisson regression model against zero-inflated negative binomial alternatives," *Biometrics*, vol. 57, no. 1, pp. 219–223, Mar. 2001.

[36] S. Gurmu, "Semi-parametric estimation of hurdle regression models with an application to medicaid utilization," *J. Appl. Econometrics*, vol. 12, no. 3, pp. 225–242, May 1997.

[37] X. Yang, K. Tang, and X. Yao, "A learning-to-rank approach to software defect prediction," *IEEE Trans. Rel.*, vol. 64, no. 1, pp. 234–246, Mar. 2015.

[38] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, "The jinx on the NASA software defect data sets," in *Proc. 20th Int. Conf. Eval. Assessment Softw. Eng.*, Jun. 2016, pp. 1–5.

[39] R. Ferenc, P. Gyimesi, G. Gyimesi, Z. Tóth, and T. Gyimóthy, "An automatically created novel bug dataset and its validation in bug prediction," *J. Syst. Softw.*, vol. 169, Nov. 2020, Art. no. 110691.

[40] F. Ipate and M. Holcombe, "An integration testing method that is proved to find all faults," *Int. J. Comput. Math.*, vol. 63, nos. 3–4, pp. 159–178, Jan. 1997.

[41] F. Ipate and M. Holcombe, "Generating test sets from non-deterministic stream X-machines," *Formal Aspects Comput.*, vol. 12, no. 6, pp. 443–458, Dec. 2000.

[42] F. Ipate and M. Holcombe, "Testing conditions for communicating stream X-machine systems," *Formal Aspects Comput.*, vol. 13, no. 6, pp. 431–446, Aug. 2002.

[43] F. Ipate and D. Dranidis, "A unified integration and component testing approach from deterministic stream X-machine specifications," *Formal Aspects Comput.*, vol. 28, no. 1, pp. 1–20, Mar. 2016.

[44] F. Ipate, "Testing against a non-controllable stream X-machine using state counting," *Theor. Comput. Sci.*, vol. 353, nos. 1–3, pp. 291–316, Mar. 2006.

[45] K. Phung and E. Ogunshile, "An algorithm for implementing a minimal stream X-machine model to test the correctness of a system," in *Proc. 8th Int. Conf. Softw. Eng. Res. Innov. (CONISOFT)*, Nov. 2020, pp. 93–101.

[46] K. Phung, D. Jayatilake, E. Ogunshile, and M. Aydin, "A stream X-machine tool for modelling and generating test cases for chronic diseases based on state-counting approach," *Program. Comput. Softw.*, vol. 47, no. 8, pp. 765–777, Dec. 2021.

[47] S. Jayatilake, E. Ogunshile, M. Aydin, and K. Phung, "Modelling diseases with stream X-machine," in *Proc. 9th Int. Conf. Softw. Eng. Res. Innov. (CONISOFT)*, Oct. 2021, pp. 61–68.

[48] Z. Kurbatova, Y. Golubev, V. Kovalenko, and T. Bryksin, "The IntelliJ platform: A framework for building plugins and mining software data," in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng. Workshops (ASEW)*, Nov. 2021, pp. 14–17.

[49] L. Ardito, R. Coppola, L. Barbato, and D. Verga, "A tool-based perspective on software code maintainability metrics: A systematic literature review," *Sci. Program.*, vol. 2020, pp. 1–26, Aug. 2020.

[50] J. D. Blischak, E. R. Davenport, and G. Wilson, "A quick introduction to version control with git and GitHub," *PLOS Comput. Biol.*, vol. 12, no. 1, Jan. 2016, Art. no. e1004668.

[51] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "The misuse of the NASA metrics data program data sets for automated software defect prediction," in *Proc. 15th Annu. Conf. Eval. Assessment Softw. Eng. (EASE)*, 2011, pp. 96–103.

[52] H. J. Zainodin and S. J. Yap, "Overcoming multicollinearity in multiple regression using correlation coefficient," *AIP Conf. Proc.*, vol. 1557, no. 1, pp. 416–419, 2013.

[53] B. Li, Q. Wang, and J. Hu, "Feature subset selection: A correlation-based SVM filter approach," *IEEJ Trans. Elect. Electron. Eng.*, vol. 6, no. 2, pp. 173–179, 2011.

[54] T. A. Craney and J. G. Surles, "Model-dependent variance inflation factor cutoff values," *Qual. Eng.*, vol. 14, no. 3, pp. 391–403, Feb. 2002.

[55] G. E. Box and D. R. Cox, "An analysis of transformations," *J. Roy. Stat. Soc. B, Methodol.*, vol. 26, no. 2, pp. 211–243, 1964.

[56] J. Osborne, "Improving your data transformations: Applying the Box–Cox transformation," *Practical Assessment, Res., Eval.*, vol. 15, no. 1, p. 12, 2010.

[57] V. Fonti and E. Belitser, "Feature selection using lasso," *VU Amsterdam Res. Paper Bus. Anal.*, vol. 30, pp. 1–25, Mar. 2017.

[58] F. N. Colakoglu, A. Yazici, and A. Mishra, "Software product quality metrics: A systematic mapping study," *IEEE Access*, vol. 9, pp. 44647–44670, 2021.

[59] M. Bansal and C. P. Agrawal, "Critical analysis of object oriented metrics in software development," in *Proc. 4th Int. Conf. Adv. Comput. Commun. Technol.*, Feb. 2014, pp. 197–201.

[60] S. Eski and F. Buzluca, "An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes," in *Proc. IEEE 4th Int. Conf. Softw. Test., Verification Validation Workshops*, Mar. 2011, pp. 566–571.

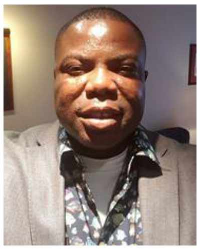
[61] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy, "Change bursts as defect predictors," in *Proc. IEEE 21st Int. Symp. Softw. Rel. Eng.*, Nov. 2010, pp. 309–318.

[62] S. Matsumoto, Y. Kamei, A. Monden, K.-I. Matsumoto, and M. Nakamura, "An analysis of developer metrics for fault prediction," in *Proc. 6th Int. Conf. Predictive Models Softw. Eng.*, Sep. 2010, pp. 1–9.

[63] D. Chhillar and K. Sharma, "Proposed T-model to cover 4S quality metrics based on empirical study of root cause of software failures," *Int. J. Electr. Comput. Eng.*, vol. 9, no. 2, p. 1122, Apr. 2019.

KHOA PHUNG received the bachelor's degree in computing and the master's degree in information technology from the University of the West of England (UWE), in 2018 and 2019, respectively. He is currently pursuing the Ph.D. degree in computer science.

He is also a Lecturer with UWE. He joined the Computer Science and Creative Technologies (CSCT) Department as a Lecturer, in April 2022. His research topic is "A novel approach for software fault prediction using stream X-machine and machine learning."



EMMANUEL OGUNSHILE received the B.Eng. degree (Hons.) in software engineering, the M.Sc. (Eng.) degree in advanced software engineering, and the Ph.D. degree in computer science from The University of Sheffield, U.K., in 2003, 2005, and 2011, respectively. His Ph.D. thesis was titled, "A Machine with Class: A Framework for Object Generation, Integration and Language Authentication: (FROGILA)."

From September 2011 to August 2013, he was with the prestigious Surrey Space Centre, University of Surrey, U.K., as a Research Fellow. From September 2013 to August 2014, he was with Loughborough University, U.K., as a Senior Research Fellow on the prestigious EPSRC and Jaguar Land Rover well over €12 million-funded collaborative research project involving six major U.K. universities. He is currently a Senior Lecturer in computer science and the Chair Athena SWAN process with the University of the West of England (UWE), Bristol, U.K.—conducting highly innovative teaching, research, scholarship, and administration in computer science and software engineering. He is a passionate British computer scientist, a software engineer, a system designer, a scientific inventor, a goal-getter, an achiever, and a gentleman. He has diversified experience having success as a software engineer, an intranet manager, a systems manager, a teaching assistant, and a senior research scientist in software and systems engineering on other occasions. He is a software professional with over 15 years of broad experience covering analysis and design, development, service, testing, technical writing, user training, team leading, and negotiation. Advocate for collaboration and adoption of new technologies to improve information sharing and collaboration. He has focused on a user-centric approach to software development to ensure usable software that solves real problems for real people. The University of Sheffield is a member of the prestigious Russell Group of research-intensive universities.

MEHMET AYDIN (Senior Member, IEEE) is currently a Senior Lecturer in computer science with the University of the West of England. He joined the Computer Science and Creative Technologies (CSCT) Department, in January 2015. Before this, he was in academic and research positions for various universities, including the University of Bedfordshire, London South Bank University, and the University of Aberdeen. He has a led an number of research projects as CoI and PI, published more than 100 articles in international peer reviewed journals and conferences. His research interest include machine learning, multi-agent systems, planning and scheduling.

Dr. Aydin is an Editorial Board member of a number of international peer-reviewed journals, and have been serving as a committee member of various international conferences. He is also a member of EPSRC Review College, senior member of ACM, and fellow of Higher Education Academy.