IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 19, 2019, accepted June 8, 2019, date of publication June 20, 2019, date of current version July 15, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2924040

Experimental Validation of Inheritance Metrics' Impact on Software Fault Prediction

SYED RASHID AZIZ[1], TAMIM AHMED KHAN[1], AND AAMER NADEEM[2]
[1]Department of Software Engineering, Bahria University, Islamabad 44000, Pakistan
[2]Department of Software Engineering, Capital University of Science and Technology, Islamabad 45750, Pakistan

Corresponding author: Tamim Ahmed Khan (tamim@bahria.edu.pk)

ABSTRACT Software faults can cause trivial annoyance to catastrophic failures. Recent work in software fault prediction (SFP) advocates the need for predicting faults before deployment to aid testing process. Object-oriented programming is complex while comparing it with procedural languages having multiple dimensions wherein inheritance is an important aspect. In this paper, we aim to investigate how much inheritance metrics assist in predicting software fault proneness. We first select the Chidamber and Kemerer (CK) metrics, most accepted metric suite for predicting software faults and inheritance metrics. We use 65 publicly available base datasets having CK metrics and some other inheritance metrics to evaluate the impact of inheritance on SFP. We split each dataset into further two datasets: inheritance with CK and CK without inheritance for comparison of results. An artificial neural network (ANN) is used for model building, and accuracy, recall, precision, F1 measures, and true negative rate (TNR) are used for measuring performance. Comparison is made and the results show an acceptable contribution of inheritance metrics in SFP. The testing community can safely use inheritance metrics in predicting software faults. Moreover, high inheritance is not desirable, as this can potentially lead to software faults.

INDEX TERMS Inheritance, object oriented, software metrics, software fault prediction, machine learning.

I. INTRODUCTION

A bug in a software program is a failure or fault that prevents the program from running as intended, for example, generating an incorrect result. A software fault is a defect that causes a software failure at runtime. A strategy is needed to predict faults earlier, as it helps to reduce faults and improve the quality of software. The essential part of software development is to ensure that developed software has enhanced quality. It is a well-proven fact that the sooner a fault is detected, the sooner it is resolved, the lesser it costs [1].

The cost of business software worldwide was $ 3.8 billion in 2014 [2] which includes 23% of quality assurance and testing [3]. These results demonstrate the importance of software testing in the Software Development Life Cycle (SDLC). Modern software is usually large and complex which makes it extremely fault-prone. Ultimately the aim of software engineers is to launch high-quality software without faults. Unfortunately, faults are unavoidable therefore certain mechanisms are required to locate and correct them from software.

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

Early detection of faults can save time, cost and reduce software complexity as it is proportional to testing. In order to locate all residual faults, thorough testing is required. Exhaustive testing is impossible [4], [5]. So this is the reason testing sometimes exceeds 50% of total development cost [6]. This number could go up to 75% according to the IBM report [7].Testing of object oriented programs is vital and unavoidable to deliver fault free software. Thoroughly testing all the classes of programs with limited resources is a challenging job, relatively it would be better to locate and test the fault prone classes more thoroughly to aid testing process.

Experimental studies show that most techniques cover less than 40% for statement coverage [8]. The testing criteria are quite expensive for coverage such as Decision Coverage, Branch Coverage, Condition Coverage etc. Since these techniques require knowledge about internal program structure along with the generation of test case to follow a particular path. Machine learning instead requires neither of such requirements and aid the testing process by locating fault prone classes to reduce testing efforts. The faults are not dispersed evenly. Some classes are faultier than others and are grouped into clusters [9]. Research reveals that faults are

confined to 42.04% of total software in a project [10], while Ostrand et al. concludes that there are faults in only 20% of the total components of the software project [11]. This makes the effective prediction of faulty classes an essential measure of modern software. Since, after deployment, troubleshooting is many times more expensive than the repair during the process of software development [12]. In addition, the late prediction of the faults propagates into the subsequent stages of development and complicates the entire prediction process. Likewise, the prediction in the early stages is a challenge due to the nonexistence of faulty data in these stages [13].

The presence of faults considerably affects software reliability, quality, and maintenance cost. The ratio of faults is: 15-50 per thousand lines of code (KLoC) for an industrial project, 10-20 for KLoC for the Microsoft application [14] and 63000 errors in its 35MLoC for Windows 2000 [15] registered. Residual faults can lead to failure since several flaws were recently observed [16]–[18].

The internal metrics are measured through the source code whereas external metrics are measured by the behavior or functionalities of the software [3]. Generally both metrics are utilized for software quality measures to mark the level of software reliability. Changes in the main attributes of different entities may show design problems or a decrease in quality and reliability. This is helpful to recognize classes during the development process which are most expected to be faulty. The present literature on fault prediction indicates inadequate ability to attain a less expensive measurement process [19].

In software engineering, there are many prediction methods, including security prediction, test effort prediction, reusability prediction, cost prediction, fault prediction, effort prediction and quality prediction [20]. SFP is an emerging field of research that is used to locate faulty classes in the early stages of software development [21]–[23] using machine learning [20], [24]. Numerous approaches use typical machine learning (ML) techniques, for instance Naive Bayes (NB) [25], Support Vector Machines (SVM) [26], Decision Trees [27] and Neural Networks [28]. These techniques have been used in SFP, utilizing measurement of metrics and faulty data from similar projects [24], [29], [30] or former versions to build fault prediction models. Assume that the metric can be used to construct a fault prediction model [31]–[35] to measure software inheritance, coupling, cohesion, complexity, and size.

Inheritance is a key feature of the object-oriented paradigm, and its metrics help to identify the complexity of classes based on SFP [36]. Abreu and Carapuça [37] stated that the greater the inheritance relation is, the greater number of methods a class is likely to inherit, making it more complex and therefore requiring more testing. Inheritance helps reuse and other factors like complexity, testability etc. [38]. It must be within limits to aid complications. Many inheritance metrics have been proposed to detect the fault propensity and quality of software systems. The inheritance metric describes the inheritance tree of the software system, the hierarchy of the class and the relationship between the superclass and its subclasses. In addition to generalization and specialization, it provides code reusability. It should be used with proper range so that the software system does not become complex [38].

This paper exclusively validates the impact of inheritance on software fault prediction. The paper contributes in cataloging the numerous inheritance metrics defined so far in literature. In addition, explore publicly available datasets having CK with Inheritance metrics. Finally, experimental validation of inheritance metrics' impact on SFP on as many as 65 different publicly available datasets

The rest of the paper is organized as follows: Section II provides the theoretical background of Software inheritance, SFP and datasets. In section III, literature review is presented. Section IV elaborates methodology used in our research followed by Section V which gives an experimental evaluation of inheritance metrics including comparisons. Section VI, threats to validity and finally conclusion and future directions of the research are given in Section VII.

II. THEORETICAL BACKGROUND

A. SOFTWARE INHERITANCE

The software metrics are the basis to measure the complexity, quality of software and estimate cost along with efforts of projects. Traditional metrics such as function point and cyclomatic complexity have been employed in the procedure paradigm. But, these are not simply used in object-oriented paradigm [39], [40]. Object Oriented programming is complex while comparing it with procedural languages [41]. Most studies stated difficulty in switching to the object-oriented paradigm from the procedural approach [42]. In Object-oriented Programming, it is difficult to understand how features, for example abstraction, inheritance and encapsulation relate to each other [43].

It is essential to differentiate amongst the design principles of object-oriented approach and design principles of functionally oriented approach. To elucidate several characteristics of object orientation and to allow for better quality management and administration [32], [40], [44]–[46]. Pressman [47] points out five situations where object-oriented metrics can configure.

1) **Location:** It is related to the information trend when it is centralized.
2) **Encapsulation:** means that the objects contain their data and attributes.
3) **Information hiding:** means to conceal the features of the object containing data and attributes.
4) **Inheritance:** permits the option to deriving a new class and allowing it the attributes of one class or more partially or fully.
5) **Object abstraction :** technique permits the designer to concentrate only on the basic and necessary details of certain parts of the program.

In object-oriented paradigm, inheritance facilitates new objects to make use of the properties of existing objects. A superclass or base class is used as the basis for inheritance and a subclass or derived class that inherits from a superclass. The main class and the secondary class term, can also be used as superclass and subclass. The secondary class can have its own properties and methods, in addition to inheriting visible properties and methods of its main class. The inheritance provides:

1) **Reusability:-** Reusability is a form based on inheritance to use public methods of the main class without having to rewrite the same code in the class.
2) **Extensibility:-** Expand the parent class logic according to the business logic of the underlying class.
3) **Data hiding: -** The data storage facility is provided by inheritance. If the parent class declares the data as private, the subclass cannot use or change it.
4) **Overriding:-** Child class can override the parent class methods so that a meaningful implementation of the parent class method can be designed in the child class.
5) **Maintainability:-** It is easy to go through the code when the program is divided into parts.

In object-oriented paradigm, inheritance base is an IS-A relationship that expresses ''X is a Y-type''. Orange is a fruit; car is a vehicle, etc. The legacy is one way, ''house is a building'', but ''building is not a house''. The legacy has two other complementary functions:

1) **Specialization:** expanding the functionality of an existing class is called specialization.
2) **Generalization:** the sharing of similarities between two or more classes is called generalization [48].

In the literature, inheritance is of different types which are briefly explained in the following lines:

1) **Single inheritance:** In a single inheritance, subclasses inherit only from a parent class.
2) **Multiple inheritance:** In multiple inheritance, a subclass extends or inherits from several primary classes. The problem with multiple inheritance is that subclasses must manage dependencies in two or more primary classes.
3) **Multi-level inheritance:** In object-oriented technology, multilevel inheritance refers to a mechanism in which a class inherits from a derived class, which makes the derived class the main class of the new class.
4) **Hierarchical Inheritance:** In the hierarchical inheritance, the main class is inherited by many secondary classes.
5) **Hybrid inheritance:** the combination of multiple inheritance and multilevel inheritance is called hybrid inheritance. Subclasses are derived from two classes, as in multiple inheritances. However, the parent class is not a base class, but these derived classes.

B. INHERITANCE METRICS AND THEIR USAGE

Inheritance is a characteristic that supports class-level design. It captures IS-A relationships between classes because class design is a fundamental part of system development [49]. Employment of inheritance decreases costs of software maintenance and testing [32]. Likewise, reuse of software through inheritance will produce software that is easier to maintain, understandable and reliable [50]. Harrison et al. performed an experimental study which shows that lack of inheritance is easier to understand or maintain as compare to systems with inheritance [50]. However, Daley's experiment indicated that systems employing tertiary inheritance are easier to modify than systems without inheritance [51].

Inheritance metrics measure many aspects of inheritance, including depth and breadth in a hierarchy and predominant complexity [52]. Rajnish and Bhattacherjee also conducted research on class inheritance metrics as in [53]–[58]. However, it is an established fact that deeper the hierarchy of inheritance, reuse of the classes is better, but it is difficult to maintain the system. Designers try to maintain shallow inheritance hierarchies, discarding reuse through inheritance for ease of understanding [32]. That is why it is important to measure complexity of inheritance hierarchy in order to resolve the differences between depth and shallowness.

Several inheritance metrics have been proposed in the literature which are included in Table 1. In this paper, we exclusively take these metrics in context of fault prediction.

C. SOFTWARE FAULT PREDICTION

Object-oriented metrics have been empirically validated to predict design flaws. Large and complex software systems are usually faulty [11]. It is difficult to keep them away from faults or to decrease risk of faults in upcoming version. According to [75], focus of verification and verification activities is to classify and remove high-risk problems in software. In order to avoid or squeeze faults, quality control models for example fault-prone models can be used for prediction of classes likely to be faulty. In order to attain these objectives, several researchers studied faults in software and constructed fault-proneness models base, on the event of failure [10], [34], [47], [76], [77]. Software prediction models are constructed using a variety of machine learning methods, such as Genetic Programming [78], Decision Trees [79], Neural Networks [80], Naive Bayes (NB) [21], Case-Based Reasoning [81], and Blurring Logic [82].

Software metrics make it easy for developers to audit and monitor the quality of software design as the project progresses. In addition, predicting the probability of defective classes is needed to help the software engineers during development to enhance the quality of software and reduce testing and maintenance costs. For example, in order to increase the productivity of software testing, software evaluators can plan tests based on the parts most prone to faults.

TABLE 1. List of inheritance metrics.

Author Name	Metrics Name
Chidamber and Kemerer [32]	Depth of Inheritance Tree (DIT)
	Number of Children (NOC)
Abreu Mood metrics suit [59]	Attribute Inheritance Factor (AIF)
	Method Inheritance Factor (MIF)
Bansiya J. et al QMOOD [60]	Number of Hierarchies (NOH)
	Average number of Ancestors (ANA)
	Measure of Functional Abstraction (MFA)
Henry's & Kafura [61]	Fan in
	Fan out
Tang, Kao and Chen, [33]	inheritance coupling(IC)
Lorenz and Kidd [62]	Number of Method Inherited (NMI)
	Number of Methods Overridden (NMO)
	Number of New Methods(NNA)
	Number of Variable Inherited (NVI)
Henderson-Sellers [63]	AID (average inheritance depth)
Li [64]	NAC (number of ancestor classes)
	NDC (number of descendent classes)
Tegarden et al. [65]	CLD (class-to-leaf depth)
	NOA (number of ancestor)
Lake and Cook [66]	NOP (number of parents)
	NOD (number of descendants)
Rajnish et al. [67] [57]	DITC (Depth of Inheritance Tree of a Class)
	CIT (Class Inheritance Tree)
Sandip et al. [20] [68]	ICC (Inheritance Complexity of Class)
	ICT (Inheritance Complexity of Tree)
Gulia, Preeti, and Rajender S. Chillar. [69]	CCDIT (Class Complexity Due To Depth of Inheritance Tree)
	CCNOC (Class Complexity Due To Number of Children)
F. T. Sheldon et al [70]	Average Degree of Understandability (AU)
	Average Degree of Modifiability (AM)
Rajnish and Choudhary [49]	Derive Base Ratio Metric (DBRM)
	Average Number of Direct Child (ANDC)
	Average Number of Indirect Child (ANIC)
Mishra, Deepti, and Alok Mishra [71]	CCI (Class Complexity due to Inheritance)
	ACI (Average Complexity of a program due to Inheritance)
	MC (Method Complexity)
Abreu and Carapuc [71] [37]	Total Children Count (TCC)
	Total Progeny Count (TPC)
	Total Parent Count (TPAC)
	Total Ascendancy Count(TAC)
	Total Length of Inheritance chain(TLI)
	Method Inheritance Factor(MIF)
K. Rajnish and A. K. Choudhary [49]	Extended Derived Base Ratio Metrics (EDBRM)
	Extended Average Number of Direct Child (EANDC)
	Extended Average Number of Indirect Child (EANIC)
Rajnish and Bhattacherjee [72]	Inheritance Metric Tree (IMT)
Chen, J. Y., and J. F. Lu [73]	Class Hierarchy of Method (CHM)
Lee et al [74]	Information-flow-based inheritance coupling (IH-ICP)

Software metrics and fault data for an earlier software version are utilized to make an SFP model for the next software version [83]. When a software has faults, it leades to a disaster because it has a significant human dependency on it. Therefore, there is an emerging requirement for software without faults. Software Companies are expanding to discover faults in software. Producing software without faults is a challenging job. To lower costs and improve software usefulness, it is vital to classify classes which are faulty. Many software metrics are proposed by the researchers to gauge the software quality. In object-oriented paradigm, metrics are beneficial for software engineers to make additional information associated with software quality available. Quality of software can also be monitored through metrics, while software evaluators use metrics to increase the effectiveness of testing [84].

D. FAULT PREDICTION TECHNIQUES

The prediction of software faults is the subject of several studies. Many techniques have been proposed for predicting a software fault, including Statistical and Machine Learning methods. These are described as:

1) STATISTICAL METHODS

Statistical methods are used to find a clear mathematical formula that absolutely identifies how classification should be performed. Kapila and Singh [85] used two statistical approaches to carry out his study: Logistic Regression and Univariate Binary Logistic Regression (UBR) which are useful for analysis of data with binary variables. In Bayesian inference [85], the design of the model relates the metrics with the content of software faults and the tendency of faults. The regression analysis is extensively utilized for prediction

of bad smell in the code and linear regression in a case where only two classes of the dependent variable exist.

2) MACHINE LEARNING

Machine Learning deals with the design, development of techniques and algorithms which pull out patterns and rules from vast datasets. Neural Networks have already been employed in software to construct reliability growth models to predict overall change or reuse metrics. A Neural Network is trained to repeat a specified set of precise classification examples, instead of producing formulas or rules. According to Mahajan *et al.* [86] stated methods of machine learning are better to discover faults in the software since all the effort is done by a machine.

The Multilayer Perceptron (MLP) is utilized to control faulty classes and the Radial Base functions are used to classify the faults according to the different categories of faults [87]. Xing et al. defines the significance of the model of Support Vector Machine (SVM). The SVM model can be used for small amount of data. SVM delivers greater Accuracy as compare to other techniques for predicting quality of software, but the performance of SVM is low in public datasets [86].

E. DATASETS IN SFP

Many datasets are being used in SFP. They can be divided into categories of public, private, partial and unknown datasets [19]. Since 2005, the use of public datasets has increased from 31% to 52% [24]. In fact, fault information for private projects is generally not available, and publicly available datasets have faulty information and are available for download. There are many fault libraries in tera-PROMISE repository [88], and D'Ambros repository which is commonly used for faults [89].

The tera-PROMISE repository provides many datasets for many projects. It has an earlier version [90], the NASA dataset is a valuable asset in the PROMISE repository because it is a widely used fault prediction library, and 60% of the articles published from 1991 to 2013 use this library [91]. The PROMISE repository provides product and process metrics for nominal and digital class labels, which is useful for creating regression and classification models in the community.

The D'Ambros database contains datasets for five software systems, including Eclipse JDT Core, Eclipse PDE UI, Equinox, Framework, Lucene, and Mylyn.

III. LITERATURE REVIEW

In this section, the focus is on the inheritance metrics, which can be useful in predicting faults. There is no systematic review of the literature. Instead of delving into how different inheritance metrics are useful in SFP.

A. INHERITANCE IN SFP

Object-oriented metrics are used to predict the quality of object-oriented software. The attributes that determine software quality include maintainability, fault tolerance, understandability, fault density, standardized rework rate, reusability, etc. Several studies were carried out, including the empirical verification of the object-oriented metrics in open source software for the prediction of faults using CBO, LOC, LCOM, NOC and DIT [10], reuse analysis of object-oriented systems using metrics of inheritance, coupling and cohesion [92], heuristic review of CK metrics [93], reusable metrics for object-oriented design [94] and empirical analysis of CK metrics for object-oriented design complexity [95].

The CK metric suite is the most used set of metrics for object-oriented software. Chidamber et al. developed and implemented a new set of software statistics for object-oriented systems [32]. Briand *et al.* [76] investigated the collection of object-oriented design statistics introduced by Basili *et al.* [34]. R. Subramanyam validated the WMC, CBO and DIT statistics as predictors of class error counts [95].

Many empirical evaluations of classification algorithms for predicting errors are performed by investigations [96]. Findings by Basili et al. revealed that different CK metrics are related to fault proneness [34]. Tang et al. analyzed CK metric suite and found none of the examined parameters significant except for RFC and WMC [97]. Briand et al have extracted 49 metrics to identify a suitable model for SFP. The results showed that all metrics except NOC were significant predictors of fault-proneness [76]. Briand and Wust found that the DIT metric related to fault proneness in an inverse manner and the NOC metric is a non-significant predictor of fault-proneness [98]. Yu et al chose eight metrics and they investigated the relationship between these metrics and error sensitivity. First, they examined the correlation between the metrics and found four strongly correlated subsets. Then they used univariate analysis to find out which statistics could detect errors [35]. Malhotra and Jain studied the relationship between object-oriented measurement data and fault proneness using the logistic regression method. Receiver Operating Characteristic (ROC) analysis was used and the performance of predicted models is thus evaluated on the basis of ROC [35]. Yeresime et al studied the application of linear regression, logistic regression and artificial neural network methods for prediction of software errors on CK metrics. Their results indicate the importance of weighted method per class (WMC) for fracture classification [99].

The literature review reveals that the DIT and NOC metrics that address the inheritance aspect are used for SFP within the CK metrics. Therefore, it is considered essential to validate the usefulness of the inheritance metrics in the SFP context.

IV. METHODOLOGY

The aim of this experiment is to validate the impact of inheritance on SFP. From literature review it appeared that Chidamber and Kemerer metric suite (CK) are widely used in SFP, so we wanted to compare the results of CK metrics without Inheritance ($CK^{-inheritance}$) and Inheritance with CK ($Inheritance^{+CK}$).

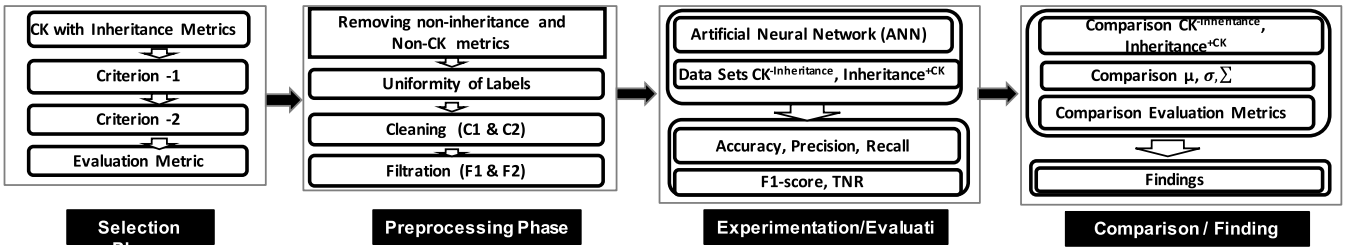


FIGURE 1. Research methodology.

In this experiment, dataset Inheritance^{+CK} consists of 6-10 metrics, having inheritance metrics {ic, mfa, noai, nomi, doc, fanin, fanout, ifanin} and CK metrics {wmc, cbo, rfc, lcom, dit, noc} depending on the availability in base datasets of Table 2 whereas CK$^{-inheritance}$ consist of 4 metrics {wmc, cbo, rfc, lcom}. Table 2 is a source table that contains 65 public base datasets having CK metrics and one or more Inheritance metrics. Then each base dataset is further split into two datasets for example Inheritance $^{+CK}$ (6-10 metrics) and CK$^{-inheritance}$ (4 metrics) to make comparison. Table 4 showed the preprocessing results where 35 base datasets are dropped. Finally table 5 showed the results of Inheritance^{+CK} and CK$^{-inheritance}$ exclusively for remaining 30 base datasets.

The research method consists of four interrelated stages, as shown in Figure 1, which includes the phases of selection, preprocessing, experimentation/evaluation and Comparison/Findings. The first stage includes the choice of CK metrics, inheritance metrics and evaluation metrics. The choice of CK and the inheritance metrics are based on two criteria, indicated as Criterion-1 and Criterion-2 in Figure 1.

In the second preprocessing phase, other metrics in addition to the CK and inheritance are discarded. The CK$^{-inheritance}$ and Inheritance^{+CK} dataset are created, and then these datasets are kept consistent. Then, the datasets will be cleaned and filtered to eliminate the associated anomalies. The final form of the dataset is used for the experimental phase in which the Artificial Neural Network (ANN) is constructed and cross-validation is performed on the datasets of CK$^{-inheritance}$ and Inheritance^{+CK}. This phase also includes the calculation of evaluation metrics; Accuracy, Precision, Recall, F1-score, and TNR score for CK$^{-inheritance}$ and Inheritance^{+CK} respectively. According to the evaluation metrics, a score is calculated for both dataset to determine the superior. In the last phase, comparisons are made and the results are taken to determine the impact of the inheritance metrics in the SFP context.

A. SELECTION PHASE

The first stage includes the choice of CK metrics, inheritance metrics and evaluation metrics. The choice of CK and inheritance metrics is based on two criteria, called Criterion-1 and Criterion-2 as below.

1) SELECTION OF CK METRICS AND INHERITANCE METRICS

The basic selection is to have a set of data that contains CK metrics in addition to inheritance metrics. With respect to the inheritance metrics discussed in section II, we only select those metrics that meet the following criteria.

2) CRITERION -1: DATASET MUST BE PUBLICLY AVAILABLE

This criterion is met because the fault information of the software projects is seriously less accessible. The reason is that fault information for large / business projects is accumulated in the software files that are propriety and in small projects, there is not enough fault information. As a result, tagged data is rarely available. The availability of the public dataset will allow the evaluation of the inheritance metrics in SFP. Finally, 65 base datasets with CK metrics and inheritance metrics were found [89], [100]–[109]. The CK metrics consist of the class-weighted method (WMC), the Depth of the Inheritance Tree (dit), the Number of Children (noc), the Coupling Between Objects (cbo), Response For a Class (rfc), the Lack of Cohesion in the Methods (lcom).). In addition to the CK metrics, in this dataset there are 8 metrics of inheritance in general; which are the Inheritance Coupling (ic), Functional Abstraction Measure (mfa), Inherited Attribute Number (noai), Inherited Method Number (nomi), Dependent of the Son (doc), number of classes that are called class methods (fanIn), number of methods called per class (fanOut) and number of immediate base classes(ifanin)

Out of these 65 datasets, 60 are located on tera-PROMISE servers [88] and 5 datasets on D'Ambros servers [110]. Table 2 shows the description of these datasets where the 1st column shows the name of the base dataset with the version (if applicable). The details on the total instances, the percentage of fault and number of metrics are shown in columns 2, 3 and 4 respectively. In general, CK and eight different inheritance metrics are found in these datasets, where ✓is marked if the metric data is present in the corresponding base dataset and × is marked in case the metric data is not available.

Unfortunately, the eight metrics along with the CK metrics could not be found together in a single dataset. However, together we can use CK metrics and {ifanin,dit,

TABLE 2. Source datasets.

Dataset Name	# Ins	% Falty	#F	CK	ic	mfa	noai	nomi	Doc	fanIn	fanOut	ifanin
churn-new	997	21	10	✓	×	×	✓	✓	×	✓	✓	×
Eclipse JDT Core	997	21	10	✓	×	×	✓	✓	×	✓	✓	×
Eclipse PDE UI	1497	14	10	✓	×	×	✓	✓	×	✓	✓	×
single-ver-ck-oo	997	20	10	✓	×	×	✓	✓	×	✓	✓	×
eclipse34_swt	1485	44	9	✓	×	×	✓	✓	×	×	×	✓
Kc1-class-binary	145	41	9	✓	✓	×	×	×	✓	✓	×	×
Kc1-class-numeric	145	41	9	✓	✓	×	×	×	✓	✓	×	×
kc1-numbericfault	145	41	9	✓	✓	×	×	×	✓	✓	×	×
kc1-top5	145	5	9	✓	✓	×	×	×	✓	✓	×	×
Lucene	691	9	9	✓	×	×	✓	✓	×	✓	✓	×
Arc	234	11	8	✓	✓	✓	×	×	×	×	×	×
berek	43	37	8	✓	✓	✓	×	×	×	×	×	×
camel-1.0	339	4	8	✓	✓	✓	×	×	×	×	×	×
camel-1.6	965	19	8	✓	✓	✓	×	×	×	×	×	×
ckjm	10	50	8	✓	✓	✓	×	×	×	×	×	×
e-learning	64	9	8	✓	✓	✓	×	×	×	×	×	×
forrest-0.6	7	14	8	✓	✓	✓	×	×	×	×	×	×
forrest-0.7	29	17	8	✓	✓	✓	×	×	×	×	×	×
forrest-0.8	32	6	8	✓	✓	✓	×	×	×	×	×	×
iny-1.1	111	57	8	✓	✓	✓	×	×	×	×	×	×
iny-1.4	241	7	8	✓	✓	✓	×	×	×	×	×	×
iny-2.0	352	11	8	✓	✓	✓	×	×	×	×	×	×
jedit-4.0	306	25	8	✓	✓	✓	×	×	×	×	×	×
jedit-4.1	312	25	8	✓	✓	✓	×	×	×	×	×	×
jedit-4.2	367	13	8	✓	✓	✓	×	×	×	×	×	×
jedit-4.3	492	2	8	✓	✓	✓	×	×	×	×	×	×
Kalkulator	27	22	8	✓	✓	✓	×	×	×	×	×	×
log4j-1.0	135	25	8	✓	✓	✓	×	×	×	×	×	×
log4j-1.1	109	34	8	✓	✓	✓	×	×	×	×	×	×
log4j-1.2	205	91	8	✓	✓	✓	×	×	×	×	×	×
lucene-2.2	247	58	8	✓	✓	✓	×	×	×	×	×	×
lucene-2.4	340	60	8	✓	✓	✓	×	×	×	×	×	×
nieruchomosci	27	37	8	✓	✓	✓	×	×	×	×	×	×
pdftranslator	33	45	8	✓	✓	✓	×	×	×	×	×	×
poi-1.5	237	59	8	✓	✓	✓	×	×	×	×	×	×
poi-2.0	314	12	8	✓	✓	✓	×	×	×	×	×	×
poi-2.5	385	64	8	✓	✓	✓	×	×	×	×	×	×
prop-2	2314	10	8	✓	✓	✓	×	×	×	×	×	×
prop-3	10274	11	8	✓	✓	✓	×	×	×	×	×	×
prop-4	8718	10	8	✓	✓	✓	×	×	×	×	×	×
prop-5	8516	16	8	✓	✓	✓	×	×	×	×	×	×
prop-6	660	10	8	✓	✓	✓	×	×	×	×	×	×
redaktor	176	15	8	✓	✓	✓	×	×	×	×	×	×
serapion	45	20	8	✓	✓	✓	×	×	×	×	×	×
skarbonka	45	20	8	✓	✓	✓	×	×	×	×	×	×
sklebagd	20	60	8	✓	✓	✓	×	×	×	×	×	×
synapse-1.0	157	10	8	✓	✓	✓	×	×	×	×	×	×
synapse-1.1	222	1	8	✓	✓	✓	×	×	×	×	×	×
systemdata	65	13	8	✓	✓	✓	×	×	×	×	×	×
szybkafucha	25	56	8	✓	✓	✓	×	×	×	×	×	×
tempoproject	42	30	8	✓	✓	✓	×	×	×	×	×	×
tomcat	858	8	8	✓	✓	✓	×	×	×	×	×	×
velocity-1.4	196	75	8	✓	✓	✓	×	×	×	×	×	×
velocity-1.5	214	69	8	✓	✓	✓	×	×	×	×	×	×
velocity-1.6	229	34	8	✓	✓	✓	×	×	×	×	×	×
workflow	39	51	8	✓	✓	✓	×	×	×	×	×	×
wspomaganiepi	18	67	8	✓	✓	✓	×	×	×	×	×	×
xalan-2.4	723	15	8	✓	✓	✓	×	×	×	×	×	×
xalan-2.7	909	98	8	✓	✓	✓	×	×	×	×	×	×
xerces-1.2	440	16	8	✓	✓	✓	×	×	×	×	×	×
xerces1.3	453	15	8	✓	✓	✓	×	×	×	×	×	×
xerces-init	162	47	8	✓	✓	✓	×	×	×	×	×	×
zuzel	29	44	8	✓	✓	✓	×	×	×	×	×	×
jEdit_4.0_4.2	274	49	6	✓	×	×	×	×	×	×	×	×
jEdit_4.2_4.3	369	55	6	✓	×	×	×	×	×	×	×	×

`noai,noc,nomi}` in one dataset, `{noai, nomi}` in one dataset, `{ic}` in two datasets, `{fanin, fanout, noai, nomi}` in 5 datasets, `{ic, mfa}` in 52 datasets and `{doc, ic, fanin}` in 4 datasets.

3) CRITERION -2: CORRELATION MUST NOT BE ≥ 0.7 OR ≤ -0.7

Metrics tend to correlate when they address a similar aspect of programming, such as inheritance in our case. A high correlation (such as ≥ 0.7 OR ≤ -0.7) is a form of redundancy, which requires the redundant metric to be removed.

TABLE 3. Pearson and Spearman's correlation coefficient between pair features.

	doc	fanIn	fanOut	ic	IFANIN	mfa	noai	noc	nomi
dit	0;0.05	-0.02;0.14	0.14;0.28	0.63;0.69	-0.03;0.04	0.62;0.68	0.25;0.42	0;-0.03	0.47;0.59
doc		0.06;0.09	Not found	Not found	Not found	Not found	Not found	0.13;0.3	Not found
fanIn			0.23;0.3	Not found	Not found	Not found	0.02;0.03	0.26;0.3	0.01;0.17
fanOut				Not found	Not found	Not found	0.08;0.15	0.01;0.06	0.14;0.26
ic					Not found	0.65;0.66	Not found	-0.01;-0.04	Not found
IFANIN						Not found	0.02;0.15	0;-0.1	0.07;0.21
mfa							Not found	0.01;-0.02	Not found
noai								0;0.01	0.35;0.4
noc									0;0.08
nomi									

The reason is that maintaining the redundant metric can be detrimental, cause confusion in the mining algorithm and uncover a low-quality pattern [111]. In addition, the benefits of discarding correlated metrics are much better than cost [112]. In the case of a lower correlation, even close to ≥ 0.7 OR ≤ -0.7, the abandonment of any metric would deprive the dataset of important exclusive information.

To examine, if the metrics show the second criterion, we perform the Pearson correlation coefficient (r) and the Spearman correlation coefficient (p) for the pairs found in the publicly available datasets. We take two unfiltered dataset characteristics that are shown in Table 3, where 27 pairs are found in the available datasets. When there are several datasets available against any pair, we combine them and calculate the value of r and p.

Table 3 shows the Pearson correlation coefficient and the Spearman correlation coefficient of each pair and the value of r and p in the respective columns separated by a semi-column. However, all relationships are positively correlated, but none is equal to ≥ 0.7 OR ≤ -0.7. Table 3 further shows that none of the pairs has non-linear correlation either. Finally, the ten metrics that include two of the CK metrics also meet the second criterion.

4) SELECTION OF EVALUATION METRICS

The machine learning models based on the classification are evaluated through their performance when classifying the unknown instances. A confusion matrix is a way of reflecting its performance, Catal enumerated numerous metrics, derived from the confusion matrix [35]. Malhotra and others also offer a general description of several evaluation measures used in SFP. According to their findings, TPR is the most commonly used evaluation measure in SFP, followed by Precision [24]. Consequently, we chose Accuracy, Precision, Recall, F1-score and TNR for the evaluation of the models used in this document.

In the SFP domain, the positive class is the faulty class. Then, if the classifier declares that any faulty instance is faulty, the classification is truly positive. If the classifier declares that an instance is fault-free when in fact it is faulty, the classification is false negative. If the classifier declares any instance as faulty, when in fact it has no faults, then the classification is false positive. Finally, if the classifier declares that any instance is free of faults when in fact it has no faults, then the classification is true negative.

Accuracy indicates the proportion of the total number of correct predictions between the total number of correct and

TABLE 4. Filtered datasets.

Dataset Name	Inheritance^{+CK}							CK$^{-Inheritance}$						
	#F	Cleaning		Filtration		Final Set		#F	Cleaning		Filtration		Final Set	
		Rednt	Incon	<100	Sqd	#ins	%Fault		Rednt	Incon	<100	Sqd	#ins	%Fault
churn-new	10	254	16	✓	✓	730	27	4	419	20	✓	✓	561	31
Eclipse JDT Cor	10	29	968	×	×	0	0	4	88	14	✓	✓	0	0
Eclipse PDE UI	10	69	4	✓	✓	1424	14	4	129	14	✓	✓	1354	15
single-v-ck-oo	10	29	968	×	×	0	0	4	88	14	✓	✓	0	0
eclipse34_swt	9	449	44	✓	✓	992	58	4	669	48	✓	✓	768	66
Kc1-class-binar	9	11	4	✓	✓	130	43	4	11	4	✓	✓	130	43
Kc1-class-num	9	11	4	✓	✓	130	43	4	11	4	✓	✓	130	43
kc1-numbfault	9	11	4	✓	✓	130	43	4	11	4	✓	✓	130	43
kc1-top5	9	13	132	×	×	0	0	4	13	132	×	×	0	0
Lucene	9	26	2	✓	×	0	0	4	58	6	✓	×	0	0
Arc	8	28	2	✓	✓	204	12	4	31	2	✓	✓	201	12
berek	8	0	43	×	×	0	0	4	0	43	×	×	0	0
camel-1.0	8	17	322	×	×	0	0	4	23	316	×	×	0	0
camel-1.6	8	123	16	✓	✓	826	21	4	159	42	✓	✓	764	21
ckjm	8	0	10	×	×	0	0	4	0	10	×	×	0	0
e-learning	8	12	52	×	×	0	0	4	15	49	×	×	0	0
forrest-0.6	8	0	6	×	×	0	0	4	0	6	×	×	0	0
forrest-0.7	8	1	28	×	×	0	0	4	1	28	×	×	0	0
forrest-0.8	8	2	30	×	×	0	0	4	2	30	×	×	0	0
iny-1.1	8	2	2	✓	✓	107	58	4	2	4	✓	✓	105	58
iny-1.4	8	9	232	×	×	0	0	4	11	230	×	×	0	0
iny-2.0	8	15	337	×	×	0	0	4	19	333	×	×	0	0
jedit-4.0	8	13	2	✓	✓	291	25	4	18	2	✓	✓	286	26
jedit-4.1	8	11	2	✓	✓	299	26	4	14	4	✓	✓	294	26
jedit-4.2	8	12	355	×	×	0	0	4	16	351	×	×	0	0
jedit-4.3	8	29	2	✓	×	0	0	4	36	2	✓	✓	0	0
Kalkulator	8	0	2	×	×	0	0	4	0	2	×	×	0	0
log4j-1.0	8	2	2	✓	✓	131	25	4	5	2	✓	✓	128	26
log4j-1.1	8	1	108	×	×	0	0	4	1	108	×	×	0	0
log4j-1.2	8	5	2	✓	×	0	0	4	7	2	✓	×	0	0
lucene-2.2	8	5	6	✓	✓	236	58	4	10	14	✓	✓	223	59
lucene-2.4	8	5	335	×	×	0	0	4	9	4	✓	✓	0	0
nieruchomosci	8	1	26	×	×	0	0	4	1	26	×	×	0	0
pdftranslator	8	0	33	×	×	0	0	4	0	33	×	×	0	0
poi-1.5	8	37	8	✓	✓	192	61	4	37	8	✓	✓	192	61
poi-2.0	8	56	4	✓	✓	254	13	4	57	4	✓	✓	253	13
poi-2.5	8	57	6	✓	✓	322	61	4	57	8	✓	✓	320	61
prop-2	8	13269	1406	✓	✓	8339	12	4	17004	1712	✓	✓	4298	13
prop-3	8	6622	1264	✓	×	0	0	4	7302	1282	✓	×	0	0
prop-4	8	5367	625	✓	✓	2726	7	4	6004	731	✓	✓	1983	7
prop-5	8	5042	1082	✓	✓	2392	16	4	5790	1106	✓	✓	1620	16
prop-6	8	269	60	✓	×	0	0	4	272	60	✓	×	0	0
redaktor	8	17	4	✓	✓	155	14	4	19	4	✓	✓	153	14
serapion	8	2	43	×	×	0	0	4	2	43	×	×	0	0
skarbonka	8	1	44	×	×	0	0	4	1	44	×	×	0	0
sklebagd	8	0	20	×	×	0	0	4	0	20	×	×	0	0
synapse-1.0	8	6	151	×	×	0	0	4	7	150	×	×	0	0
synapse-1.1	8	10	2	✓	✓	210	28	4	12	2	✓	✓	208	28
systemdata	8	2	63	×	×	0	0	4	2	63	×	×	0	0
szybkafucha	8	0	25	×	×	0	0	4	0	25	×	×	0	0
tempoproject	8	4	38	×	×	0	0	4	5	37	×	×	0	0
tomcat	8	111	2	✓	✓	745	10	4	141	6	✓	✓	711	10
velocity-1.4	8	19	4	✓	✓	173	74	4	24	4	✓	✓	168	74
velocity-1.5	8	25	4	✓	✓	185	66	4	25	4	✓	✓	185	66
velocity-1.6	8	26	6	✓	✓	197	36	4	27	6	✓	✓	196	36
workflow	8	2	37	×	×	0	0	4	2	37	×	×	0	0
wspomaganiepi	8	0	18	×	×	0	0	4	0	18	×	×	0	0
xalan-2.4	8	70	6	✓	✓	647	16	4	96	14	✓	✓	613	17
xalan-2.7	8	214	2	✓	×	0	0	4	248	2	✓	×	0	0
xerces-1.2	8	129	28	✓	✓	283	15	4	139	26	✓	✓	275	14
xerces1.3	8	139	6	✓	✓	308	21	4	147	6	✓	✓	300	21
xerces-init	8	21	10	✓	✓	131	44	4	25	10	✓	✓	127	45
zuzel	8	0	2	×	×	0	0	4	0	2	×	×	0	0
jEdit_4.0_4.2	6	9	8	✓	✓	257	50	4	12	10	✓	✓	341	57
jEdit_4.2_4.3	6	12	6	✓	✓	0	0	4	18	10	✓	×	0	0
Total	**32733**		**9085**	**31**		**23146**		**39352**		**7357**	**34**		**17017**	

incorrect predictions. Precision represents the proportion of correctly ranked error-prone classes across the total number of classified error-prone classes. Recall is the ratio of correctly predicted error-prone classes between all actual classes that are prone to errors. F1-score (also F-score or F-measure) is a measure of the Accuracy of a test. It concerns both the Precision and the Recall of the test. The F1-score is the harmonic mean of Precision and Recall, with an F1-score reaching the best value at 1 (perfect Precision and Recall) and the worst at 0.

B. PREPROCESSING PHASE

1) REMOVE NON-CK AND NON-INHERITANCE FEATURES

The selected dataset contains many non-CK and non-inheritance metrics, including loc, ca and so on. As we aim to evaluate CK$^{-inheritance}$ and Inheritance^{+CK} metrics in the SFP, these non-CK and non-inheritance metrics are removed. This may affect the performance of SFP, but it may be possible to summarize the impact of inheritance metrics on SFPs.

2) UNIFORMITY OF LABELS

All metrics have continuous values in the corresponding dataset, and inconsistencies can be found in the class tag (BUG), which is solved by the following rules.

$$BUG = \left\{ \begin{array}{ll} \text{FALSE} & \text{Defects} = 0, \text{N}, \text{No}, \text{FALSE} \\ \text{TRUE} & \text{Otherwise} \end{array} \right\} \quad (1)$$

Among them, FALSE is used for fault-free, TRUE is used for faulty instance.

3) SPLITTING/MERGING

Since our objective is to quantify the impact of the chosen inheritance metrics, we divided each dataset into further two datasets as CK$^{-inheritance}$ and Inheritance^{+CK} of these 65 base datasets (after discarding the non-CK and non-inheritance metrics). The CK$^{-inheritance}$ consists of 4 metrics by excluding the DIT and NOC metrics from the set of CK metrics, since these are inheritance metrics. The Inheritance^{+CK} consists of 6 metrics of CK with other available inheritance metrics. As a result, this dataset consists of 6-10 metrics established according to the availability of the inheritance metrics in the dataset. Finally, table 4 is divided into two equal parts to show the statistics of two datasets that are formed by dividing a base dataset that mentions the name of the dataset in the first column. Followed by the number of features in which the CK$^{-inheritance}$ has 4 features and Inheritance^{+CK} has between 6 and 10 features. The cumulative numbers of instances in these 65 base datasets are 70,101.

4) CLEANING

In this phase, the redundant instances were removed from the CK$^{-inheritance}$ and Inheritance^{+CK} datasets because they are useless and, at times, confusing for the model. Consequently, 39,352 of 70,101 instances in the CK$^{-inheritance}$ partition and 32,733 instances in the Inheritance^{+CK} partition are redundant against these 65 base datasets, which are deleted accordingly. After that, inconsistent instances are also removed. The inconsistency in the instances is an anomaly of the dataset [63], where the instances have the same values for all the metrics but have different class labels. In our case, there are 7,357 inconsistent instances in the CK$^{-inheritance}$ and 9,085 in the Inheritance^{+CK} dataset (after deleting the redundant instances). These non-inheritance and non-CK metrics segregated the instances with each other, and when discarded, these segregated instances become inconsistent. After removing redundant and inconsistent instances, 17,017 instances remain in 65 datasets of CK$^{-inheritance}$ dataset and 23,146 instances remain in Inheritance^{+CK} dataset.

5) FILTRATION

In this phase, two filters are used; Number of instances $\geq$100 and skewness $\leq$ 9: 1. The first filter is used so that a ten-fold cross-validation would be possible without replacement, which is usually the case with model validation. This filter deletes 25 datasets from $CK^{-inheritance}$ and 28 datasets in $Inheritance^{+CK}$.

6) SKEWNESS $\leq$ 9:1

Skewness indicates that a fault or no-fault instance should constitute $\geq$10% and $\leq$90% of the dataset. This filter is used because if there are only 100 instances in the worst case, there should be at least one instance from both classes if there is no hierarchical 10-fold cross validation that is replaced. Further, this filter deletes 6 each dataset from $CK^{-inheritance}$ and $Inheritance^{+CK}$ respectably. Finally, after cleaning and filtering, datasets were trimmed to only 34 in $CK^{-inheritance}$ and 31 in $Inheritance^{+CK}$ (shown in the last row of third and eight columns of Table 4).

In addition, one dataset jedit-4.2-4.3 was dropped while making one to one mapping between $CK^{-inheritance}$ and $Inheritance^{+CK}$ datasets. So finally 30 datasets each remained for the experiment.

V. EXPERIMENT AND RESULTS

A. EXPERIMENT SETUP

1) DATASET

30 filtered datasets each for $CK^{-inheritance}$ and $Inheritance^{+CK}$, as shown in Table 4.

2) TOOLS

R 3.4.3 Language [53] in R Studio 1.1.383 [54].

3) SPLIT-TECHNOLOGY VERIFICATION

Ten-fold stratified cross-validation without replacement.

4) CLASSIFIERS

ANN is used for classification. Unlike Naive Bayes and tree-based algorithms, neither of these algorithms requires discretization of the dataset. In addition, they can even handle a single feature, which exists in two datasets. ANN is most effective technique, used for classification task which is performed on object-oriented metrics [113]. The latest trend in software defect prediction is the use of ANN [4], [114]. Secondly the selection of technique for modeling is made while keeping in view the statistical description of the training dataset. It's the dataset that leads us to ANN and in turn we conclude results.

30 filtered-clean datasets shown in table 4 are used for building and validating ANN. Model building of ANN is done using nnet 7.3-12 package of R 3.4.3 language [115]. Stratified splitting without replacement is done for ten-fold cross-validation.

The input to ANN, datasets are normalized using the min-max algorithm. Min-max scaling is used to make the data aligned to the bounds of activation function which is sigmoid in our case. Detail design of ANN implementation is shown in Algorithm 1. ANN is set to have only one hidden layer. A loop changes the number of units for that single layer, from zero to 100 in each fold. For every iteration, a model is built and least error carrier model out of the 100 models has been selected for that fold. Accuracy Precision, Recall, F1-score and TNR are saved for $CK^{-inheritance}$ and $Inheritance^{+CK}$ datasets. Accumulative results of 30 datasets are shown in Table 5.

TABLE 5. Experimental results.

Dataset Name	Inheritance + CK						$CK^{-Inheritance}$					
	#F	Accuracy	Precision	Recall	F1-score	TNR	#F	Accuracy	Precision	Recall	F1-score	TNR
churn-new	10	0.999	0.995	1.000	0.997	0.998	4	0.995	1.000	0.983	0.991	1.000
Eclipse PDE UI	10	0.828	0.350	0.235	0.282	0.927	4	0.832	0.312	0.121	0.174	0.954
eclipse34_swt	9	0.826	0.866	0.828	0.846	0.822	4	0.822	0.884	0.841	0.862	0.785
Arc	8	0.828	0.292	0.280	0.286	0.905	4	0.801	0.174	0.160	0.167	0.892
camel-1.6	8	0.724	0.315	0.257	0.283	0.849	4	0.736	0.331	0.241	0.279	0.869
iny-1.1	8	0.589	0.667	0.581	0.621	0.600	4	0.590	0.655	0.623	0.639	0.545
jedit-4.0	8	0.715	0.440	0.446	0.443	0.806	4	0.692	0.391	0.338	0.362	0.816
jedit-4.1	8	0.766	0.551	0.494	0.521	0.860	4	0.707	0.429	0.395	0.411	0.817
Kc1-class-binary	8	0.669	0.614	0.625	0.619	0.703	4	0.608	0.547	0.518	0.532	0.676
Kc1-class-numeric	8	0.662	0.611	0.589	0.600	0.716	4	0.608	0.547	0.518	0.532	0.676
kc1-numbericfault	8	0.662	0.611	0.589	0.600	0.716	4	0.608	0.547	0.518	0.532	0.676
log4j-1.0	8	0.748	0.500	0.455	0.476	0.847	4	0.695	0.406	0.394	0.400	0.800
lucene-2.2	8	0.581	0.648	0.606	0.626	0.545	4	0.520	0.597	0.583	0.590	0.429
poi-1.5	8	0.703	0.770	0.737	0.753	0.649	4	0.667	0.725	0.737	0.731	0.554
poi-2.0	8	0.827	0.333	0.294	0.313	0.909	4	0.791	0.194	0.176	0.185	0.886
poi-2.5	8	0.717	0.768	0.772	0.770	0.632	4	0.713	0.771	0.755	0.763	0.645
prop-2	8	0.882	0.500	0.115	0.187	0.985	4	0.876	0.549	0.192	0.285	0.977
prop-4	8	0.862	0.213	0.323	0.256	0.905	4	0.844	0.220	0.386	0.281	0.883
prop-5	8	0.829	0.430	0.168	0.241	0.957	4	0.830	0.412	0.163	0.234	0.956
redaktor	8	0.871	0.542	0.591	0.565	0.917	4	0.817	0.375	0.409	0.391	0.885
synapse-1.1	8	0.757	0.569	0.559	0.564	0.834	4	0.692	0.453	0.407	0.429	0.805
tomcat	8	0.893	0.451	0.307	0.365	0.958	4	0.887	0.387	0.164	0.231	0.970
velocity-1.4	8	0.780	0.836	0.875	0.855	0.511	4	0.685	0.781	0.800	0.791	0.349
velocity-1.5	8	0.719	0.787	0.787	0.787	0.587	4	0.708	0.758	0.820	0.787	0.492
velocity-1.6	8	0.660	0.528	0.400	0.455	0.803	4	0.520	0.333	0.343	0.338	0.619
xalan-2.4	8	0.822	0.448	0.368	0.404	0.911	4	0.796	0.328	0.216	0.260	0.912
xerces-1.2	8	0.845	0.471	0.381	0.421	0.925	4	0.804	0.273	0.231	0.250	0.898
xerces1.3	8	0.766	0.446	0.446	0.446	0.852	4	0.750	0.403	0.397	0.400	0.844
xerces-init	8	0.695	0.641	0.707	0.672	0.685	4	0.677	0.660	0.579	0.617	0.757
jEdit_4.0_4.2	6	0.693	0.692	0.698	0.695	0.688	4	0.611	0.610	0.648	0.629	0.573
Sum		**22.915**	**16.884**	**15.511**	**15.95**	**24.004**		**21.881**	**15.052**	**13.655**	**14.071**	**22.939**
Median		**0.762**	**0.546**	**0.526**	**0.542**	**0.841**		**0.71**	**0.441**	**0.402**	**0.405**	**0.811**
Average		**0.764**	**0.563**	**0.517**	**0.532**	**0.8**		**0.729**	**0.502**	**0.455**	**0.469**	**0.765**
Standardevation		**0.094**	**0.181**	**0.219**	**0.205**	**0.137**		**0.11**	**0.206**	**0.236**	**0.224**	**0.172**

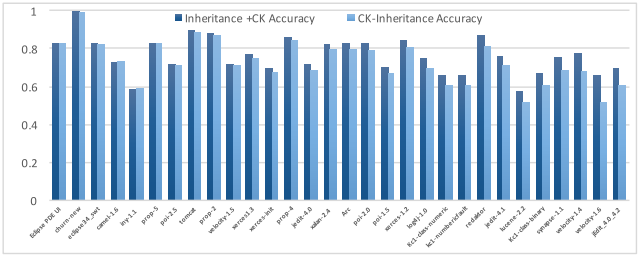


FIGURE 2. Accuracy comparison of $Inheritance^{+CK}$ with $CK^{-Inheritance}$.

B. COMPARISION AND FINDINGS

The main objective of this work is to experimentally validate inheritance metrics' impact in SFP, while the secondary objective is to obtain the best results of the machine learning algorithms. This is exactly how datasets are filtered and experiments are designed. Table 5 shows the results of the experiments where Accuracy, Recall, F1-score, and TNR are calculated for 30 datasets each for $CK^{-inheritance}$ and $Inheritance^{+CK}$. Figure 2-6 graphically compares the performance differences of ANN in $CK^{-inheritance}$ and $Inheritance^{+CK}$ through evaluation metrics that show that $Inheritance^{+CK}$ has superior results.

Figure 7 reflects the absence of outliers in the results across the datasets. It can therefore be safely stated that

Algorithm 1 Single-Layer ANN Model Building and Result Collection

```
function buildANN (30 datasets);
  Input  : 30 datasets
  Output: Accuracy, Precision, Recall, F1-score and TNR
          of the datasets with number of units of the best
          model in each dataset
for currentDataset ← 1 to 30 do
    currentDataset← minMaxScale(currentDataset)
    stratifiedSplit(currentDataset, k←10)
    for k ← 1 to 10 do
        trainset ← currentDataset[k-1]
        testset ← currentDataset[k]
        centModels ← ∅
        for units ← 1 to 100 do
            centModels ← {centModels} ∪
            trainANN(trainset, units)
        end
        currentModel ← bestModel(centModels)
        thisFoldAcc, thisFoldRecall ←
        currentModel(testset)
        thisDatasetAcc ← {thisDatasetAcc} ∪
        {thisFoldAcc}
        thisDatasetPre ← {thisDatasetPre} ∪
        {thisFoldPPre}
        thisDatasetRecall ← {thisDatasetRecall} ∪
        {thisFoldRecall}
        thisDatasetf1scr ← {thisDatasetf1scr} ∪
        {thisFoldf1scr}
        thisDatasettnr ← {thisDatasettnr} ∪
        {thisFoldtnr}
    end
    allDatasetAcc ← {allDatasetAcc} ∪
    mean(thisDatasetAcc)
    allDatasetPre ← {allDatasetPre} ∪
    mean(thisDatasetPre)
    allDatasetRecall ← {allDatasetRecall} ∪
    mean(thisDatasetRecall)
    allDatasetf1scr ← {allDatasetf1scr} ∪
    mean(thisDatasetf1scr)
    allDatasettnr ← {allDatasettnr} ∪
    mean(thisDatasettnr)
end
return allDatasetAcc, allDatasetPre, allDatasetRecall,
allDatasetf1scr, allDatasettnr
```

the averages of performance measures are not biased and hence the out performance of Inherticance + CK metrics prevails.

In order to further validate the impact of inheritance in SFP, the following comparisons are taken between these two datasets.

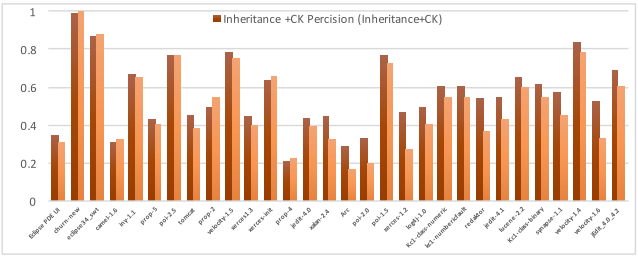


FIGURE 3. Precision comparison of Inheritance^{+CK} with CK$^{-Inheritance}$.

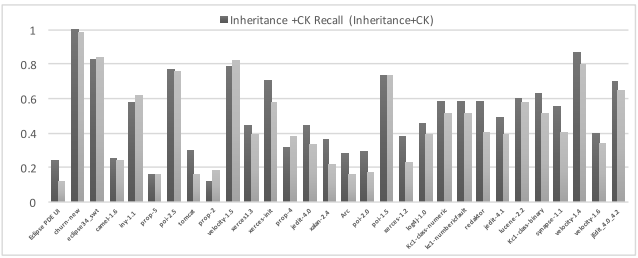


FIGURE 4. Recall comparison of Inheritance^{+CK} with CK$^{-Inheritance}$.

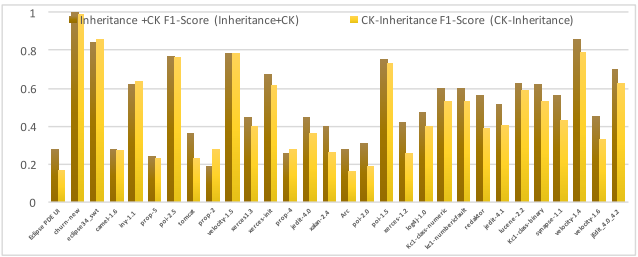


FIGURE 5. F1-score comparison of Inheritance^{+CK} with CK$^{-Inheritance}$.

1) OVERALL COMPARISON

In order to validate the overall comparison between CK$^{-inheritance}$ and Inheritance^{+CK} datasets for example which have yielded better results. In this regards, last four rows of table 5 show the sum, median, average and standard deviation of Accuracy, Precision, Recall, F1-score and TNR for these two datasets. The findings are explained as under:

(a) The overall sum of Inheritance^{+CK} dataset is 22.915, 16.884, 15.511, 15.950, 24.004 and CK$^{-Inheritance}$ dataset is 21.881, 15.057, 13.655, 14.071, 22.919 for Accuracy, Precision, Recall, F1-score, and TNR.

(b) The overall median of Inheritance^{+CK} dataset is 0.762, 0.546, 0.526, 0.542, 0.841 and CK$^{-Inheritance}$ dataset is 0.71, 0.441, 0.402, 0.405, 0.811 for Accuracy, Precision, Recall, F1-score and TNR.

(c) The overall average of Inheritance^{+CK} dataset is 0.764, 0.563, 0.517, 0.532, 0.8 and CK$^{-Inheritance}$ dataset is 0.729, 0.502, 0.455, 0.469, 0.765 for Accuracy, Precision, Recall, F1-score and TNR respectively.

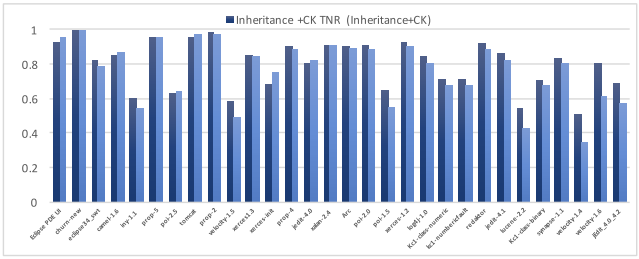


FIGURE 6. TNR comparison of Inheritance^{+CK} with CK$^{-Inheritance}$.

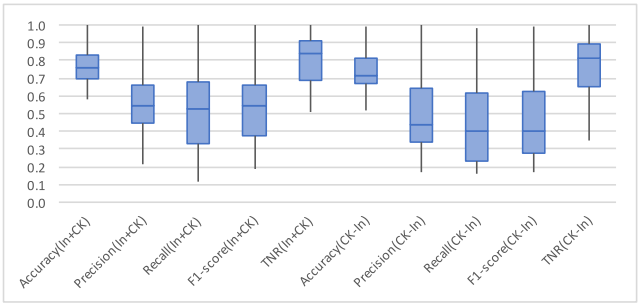


FIGURE 7. Box plot.

It is validated that Inheritance^{+CK} has better results of the sum, median and average for all evaluation metrics whereas standard deviation is better for CK$^{-Inheritance}$.

2) ONE-ONE COMPARISON

Experimental results of evaluation metrics including Accuracy, Precision, Recall, F1-score, and TNR are shown in table 5 for Inheritance^{+CK} and CK$^{-Inheritance}$ datasets which are created from the base dataset shown in the first column. In order to do one-on-one comparison, the results of evaluation metrics of Inheritance^{+CK} and CK$^{-Inheritance}$ datasets are compare to see the greater values for each base dataset. The results are summarized in table 6 which shows that:

(a) The Accuracy and TNR show that inheritance^{+CK} has superior results in 26 datasets whereas CK$^{-Inheritance}$ is superior in only 4 datasets. So inheritance^{+CK} attained 87% of total datasets as compare to 13% for CK$^{-Inheritance}$.

(b) The Precision shows that inheritance^{+CK} has superior results in 23 datasets whereas CK$^{-Inheritance}$ is superior in only 7 datasets. So Inheritance^{+CK} attained 77% of total datasets as compare to 23% for CK$^{-Inheritance}$.

(c) The Recall and F1-score show that inheritance^{+CK} has superior results in 24 datasets whereas CK$^{-Inheritance}$ is superior in only 5 datasets. So Inheritance^{+CK} attained 83% of total datasets as compare to 17% for CK$^{-Inheritance}$.

(d) The F1-score shows that inheritance^{+CK} has superior results in 25 datasets whereas CK$^{-Inheritance}$ is superior in only 5 datasets. So Inheritance^{+CK} attained 83% of total datasets as compare to 17% for CK$^{-Inheritance}$.

TABLE 6. One-one comparison of CK$^{-Inheritance}$ and Inheritance^{+CK}.

Group	Accuracy	Precision	Recall	F1-Score	TNR
Inheritance^{+CK}(#)	26	23	24	25	26
CK$^{-inheritance}$(#)	4	7	5	5	4
Inheritance^{+CK} (%)	87	77	83	83	87
CK$^{-inheritance}$(%)	13	23	17	17	13

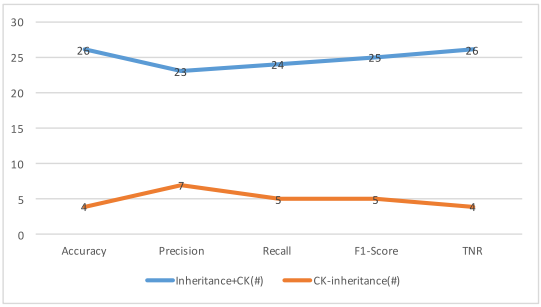


FIGURE 8. One to one comparison.

Out of total 30 datasets, Inheritance^{+CK} datasets has superior results as one-one comparison with CK$^{-inheritance}$ for all evaluation metrics. The graphical representation is shown in Figure 8.

3) CK METRICS COMPARISON

Most of the times CK metrics set is considered a better performer in SFP. In order to validate that adding inheritance metrics in CK metrics set will enhance the results of fault prediction. So average is calculated on the results of experiment shown in table 5 by grouping the datasets having `{CK}` metrics. Then adding two inheritance metrics `{ic, mfa}` into CK (8 metrics set). Followed by three inheritance metrics `{nomi, noai, ifanin}` (9 metrics set)and finally four inheritance metrics `{fanin, fanout, nomi, noai}`. THE Results are summarized in Table 7 and graphically represented in Figure 9. The findings are:

(a) The accuracy shows that CK metrics average results is 69.3 and after adding two inheritance metrics the result enhanced to 75.3. Subsequently after adding three inheritance metrics the result enhanced to 82.6 and finally 91.3 after adding four metrics. It clearly validated that adding inheritance metrics into CK metrics will increase the results gradually.

(b) Similarly TNR shows that CK average results is 68.8 and after adding two inheritance metrics the result enhanced to 79.1. Subsequently after adding three inheritance metrics the result enhanced to 82.2 and finally 96.3 after adding four metrics. It clearly validated that adding inheritance metrics into CK metrics will increase the results gradually

VI. THREATS TO VALIDITY

This study depends on datasets extracted from NASA and tera-PROMISE repositories and there is not enough informa-

TABLE 7. Comparison of CK with inheritance metric.

Metrics	Accuracy	Precision	Recall	F1-Score	TNR
CK	69.3	69.2	69.8	69.5	68.8
CK+[ic, mfa]	75.3	53.8	49	50.5	79.1
CK+ [nomi, noai, ifanin]	82.6	86.6	82.8	84.6	82.2
Ck+[fanin, fanout, nomi, noai]	91.3	67.3	61.8	63.9	96.3

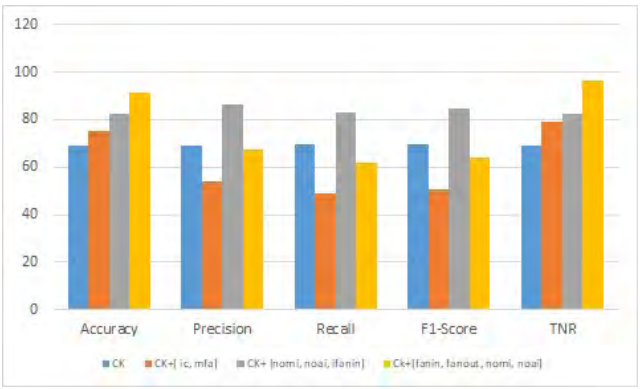


FIGURE 9. Comparison with CK metrics.

tion available about the faults whether these belongs to any particular family of software faults.

In this study, the fault does not specify any particular family of software fault. Hence the predictive ability may not be generalized to all families of software faults. Similarly, the selected datasets cover few software product with varying in design, team, scope etc. Occurrence of fault may not be the consequence of inheritance alone. Finally Selected inheritance metrics do not address all the dimensions associated with the inheritance in software product. Hence the generalization of selected inheritance metrics may not be result of all the dimensions of inheritance.

VII. CONCLUSION AND FUTURE WORK

In this paper, the impact of inheritance metrics on SFP is validated. The validation is performed through experiments on 65 publicly available datasets. The predictive capability of CK metrics is accepted in the testing community, which prevails in our experiments also. However, exclusion of inheritance metrics from CK suite can significantly degrade the predictive capability of the suite. Moreover, adding more inheritance metrics to the CK suite significantly upgrades the predictive capability. This advocates the viability of the inheritance metrics in combination with CK measures in software fault prediction.

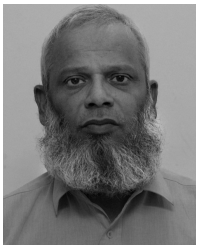
This work implies that testing community can safely use inheritance metrics to predict software faults. Moreover, the higher the figures of inheritance metrics indicate the induction of faults. This directs the software developers/designers to keep the inheritance metrics minimum. As regards future work, we expect some researchers would redo our experiment and also try to evaluate inheritance metrics other than the ones we used. Apart from that, regression-based machine learning techniques to predict faults using inheritance metrics would be interesting work to do.

REFERENCES

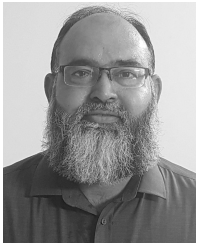
[1] *Software Bug*. Accessed: Nov. 30, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Softwarebug
[2] R. Van Der Meulen and J. Rivera, "Gartner says worldwide it spending on pace to reach $3.8 trillion in 2014," Gartner, Stamford, CT, USA, Jan. 2014.
[3] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," *Appl. Soft Comput.*, vol. 33, pp. 263–277, Aug. 2015.
[4] R. Jayanthi and L. Florence, "Software defect prediction techniques using metrics based on neural network classifier," *Cluster Comput.*, pp. 1–12, 2018.
[5] C. Kaner, J. Bach, and B. Pettichord, *Lessons Learned in Software Testing*. Hoboken, NJ, USA: Wiley, 2008.
[6] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
[7] B. Hailpern and P. Santhanam, "Software debugging, testing, and verification," *IBM Syst. J.*, vol. 41, no. 1, pp. 4–12, 2002.
[8] B. S. Ainapure, *Software Testing and Quality Assurance*. Pune, India: Technical Publications, 2009.
[9] S. A. Sherer, "Software fault prediction," *J. Syst. Softw.*, vol. 29, no. 2, pp. 97–105, 1995.
[10] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Trans. Softw. Eng.*, vol. 31, no. 10, pp. 897–910, Oct. 2005.
[11] 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.
[12] L. Pelayo and S. Dick, "Applying novel resampling strategies to software defect prediction," in *Proc. NAFIPS Annu. Meeting North Amer. Fuzzy Inf. Process. Soc.*, Jun. 2007, pp. 69–72.
[13] H. B. Yadav and D. K. Yadav, "A fuzzy logic based approach for phase-wise software defects prediction using software metrics," *Inf. Softw. Technol.*, vol. 63, pp. 44–57, Jul. 2015.
[14] S. McConnell, *Code Complete*. London, U.K.: Pearson, 2004.
[15] M. Shaw, "Sufficient correctness and homeostasis in open resource coalitions," in *Proc. ISAW-4-Int. Softw. Archit. Workshop*, 2000, pp. 46–50.
[16] G. Noto La Diega and I. Walden, "Contracting for the 'Internet of Things': Looking into the nest," School Law, Queen Mary, Univ. London, London, U.K., Res. Paper 219, 2016.
[17] K. Osborn, "Software glitch causes F-35 to incorrectly detect targets in formation," Military, San Francisco, CA, USA, Tech. Rep., Mar. 2015. [Online]. Available: https://www.military.com/defensetech/2015/03/24/software-glitch-causes-f-35-to-incorrectly-detect-targets-in-formation
[18] W. Grice, "Divorce error on form caused by uk government software glitch could affect 20,000 people," Independ. Digit. News Media Ltd., London, U.K., Tech. Rep., Dec. 2015.
[19] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Inf. Sci.*, vol. 179, no. 8, pp. 1040–1058, 2009.
[20] C. Catal, "Software fault prediction: A literature review and current trends," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 4626–4636, 2011.
[21] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.
[22] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 414–423.
[23] N. Seliya, T. M. Khoshgoftaar, and J. Van Hulse, "Predicting faults in high assurance software," in *Proc. IEEE 12th Int. Symp. High Assurance Syst. Eng.*, Nov. 2010, pp. 26–34.
[24] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Appl. Soft Comput.*, vol. 27, pp. 504–518, Feb. 2015.
[25] T. Bayes, "An essay towards solving a problem in the doctrine of chances.[facsimil]," *Revista de la Real Academia de Ciencias Exactas, Físicas y Naturales*, vol. 95, no. 1, pp. 11–60, 2001.
[26] N. Cristianini, J. Shawe-Taylor, and R. Holloway, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge, U.K.: Cambridge Univ. Press, 2000.
[27] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

[28] B. Kröse, B. Krose, P. van der Smagt, and P. Smagt, *An Introduction to Neural Networks*. Pennsylvania, PA, USA: College of Information Sciences and Technology, 1993.
[29] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Inf. Softw. Technol.*, vol. 58, pp. 388–402, Feb. 2015.
[30] A. A. Asad and I. Alsmadi, "Evaluating the impact of software metrics on defects prediction. part 2," *Comput. Sci. J. Moldova*, vol. 22, no. 1, pp. 127–144, 2014.
[31] J.-C. Chen and S.-J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability," *J. Syst. Softw.*, vol. 82, no. 6, pp. 981–992, 2009.
[32] 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.
[33] W. Li and S. Henry, "Maintenance metrics for the object oriented paradigm," in *Proc. 1st Int. Softw. Metrics Symp.*, 1993, pp. 52–60.
[34] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Softw. Eng.*, vol. 22, no. 10, pp. 751–761, Oct. 1996.
[35] R. Malhotra and A. Jain, "Fault prediction using statistical and machine learning methods for improving software quality," *J. Inf. Process. Syst.*, vol. 8, no. 2, pp. 241–262, 2012.
[36] L. H. Son, N. Pritam, M. Khari, R. Kumar, P. T. M. Phuong, and P. H. Thong, "Empirical study of software defect prediction: A systematic mapping," *Symmetry*, vol. 11, no. 2, p. 212, 2019.
[37] F. B. E. Abreu and R. Carapuça, "Candidate metrics for object-oriented software within a taxonomy framework," *J. Syst. Softw.*, vol. 26, no. 1, pp. 87–96, 1994.
[38] S. Chawla and R. Nath, "Evaluating inheritance and coupling metrics," *Int. J. Eng. Trends Technol.*, vol. 4, no. 7, pp. 2903–2908, 2013.
[39] N. Fenton, "Software measurement: A necessary scientific basis," *IEEE Trans. Softw. Eng.*, vol. 20, no. 3, pp. 199–206, Mar. 1994.
[40] R. G. Fichman and C. F. Kemerer, "Object-oriented and conventional analysis and design methodologies," *Computer*, vol. 25, no. 10, pp. 22–39, Oct. 1992.
[41] G. Pascoe, "Elements of object-oriented programming," *Byte*, vol. 11, no. 8, pp. 139–144, 1986.
[42] M. Kölling, "The problem of teaching object-oriented programming, Part 1: Languages," *J. Object-Oriented Program.*, vol. 11, no. 8, pp. 8–15, 1999.
[43] K. Eliason. *Difference Between Object-Oriented Programming and Procedural Programming Languages*. Accessed: Mar. 29, 2019. [Online]. Available: https://neonbrand.com/website-design/procedural-vs-object-oriented-programming-a-review
[44] S. Mäkelä and V. Leppänen, "Observation on lack of cohesion metrics," in *Proc. Int. Conf. Comput. Syst. Technologies-CompSysTech*, vol. 6, 2006, pp. II-10-1–II-10-6.
[45] M. Thapaliyal and G. Verma, "Software defects and object oriented metrics-an empirical analysis," *Int. J. Comput. Appl.*, vol. 9, no. 5, pp. 41–44, 2010.
[46] S. D. Conte, H. E. Dunsmore, and Y. Shen, *Software Engineering Metrics and Models*. Redwood City, CA, USA: Benjamin Cummings, 1986.
[47] L. C. Briand, J. Wüst, and H. Lounis, "Replicated case studies for investigating quality factors in object-oriented designs," *Empirical Softw. Eng.*, vol. 6, no. 1, pp. 11–58, 2001.
[48] D. Parson, *Object-Oriented Programming with C++*. DP Publications Ltd., 1994.
[49] K. Rajnish, A. K. Choudhary, and A. M. Agrawal, "Inheritance metrics for object-oriented design," *Int. J. Comput. Sci. Inf. Technol.*, vol. 2, no. 6, pp. 13–26, 2010.
[50] 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.
[51] J. Daly, A. Brooks, J. Miller, M. Roper, and M. Wood, "Evaluating inheritance depth on the maintainability of object-oriented software," *Empirical Softw. Eng.*, vol. 1, no. 2, pp. 109–132, 1996.
[52] G. Krishna and R. K. Joshi, "Inheritance metrics: What do they measure?" in *Proc. 4th Workshop Mech. Specialization, Generalization Heritance*, 2010, p. 1.
[53] K. Rajnish and V. Bhattacherjee, "Maintenance of metrics through class inheritance hierarchy," in *Proc. Int. Conf. Challenges Opportunities IT Ind.*, 2005, p. 83.
[54] K. Rajnish and V. Bhattacherjee, "A new metric for class inheritance hierarchy: An illustration," in *Proc. Nat. Conf. Emerg. Princ. Practices Comput. Sci. Inf. Technology*, 2006, pp. 321–325.
[55] K. Rajnish and V. Bhattacherjee, "Class inheritance metrics and development time: A study," *Int. J. Titled PCTE J. Comput. Sci.*, vol. 2, no. 2, pp. 22–28, 2006.
[56] K. Rajnish and V. Bhattacherjee, "Applicability of weyuker property 9 to object-oriented inheritance tree metric—A discussion," in *Proc. 10th Int. Conf. Inf. Technol. (ICIT)*, 2007, pp. 234–236.
[57] K. Rajnish and V. Bhattacherjee, "Class inheritance metrics-an analytical and empirical approach," *J. Comput. Sci.*, vol. 7, no. 3, pp. 25–34, 2008.
[58] K. Rajnish, V. Bhattacherjee, and S. Singh, "An empirical approach to inheritance tree metric," in *Proc. Nat. Level Tech. Conf. (Techno Vis.)*, 2007, pp. 145–150.
[59] F. B. Abreu and R. Carapuça, "Object-oriented software engineering: Measuring and controlling the development process," in *Proc. 4th Int. Conf. Softw. Qual.*, vol. 186, pp. 1–8, 1994.
[60] 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.
[61] S. Henry and D. Kafura, "Software structure metrics based on information flow," *IEEE Trans. Softw. Eng.*, no. 5, pp. 510–518, Sep. 1981.
[62] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics: A Practical Guide*. Upper Saddle River, NJ, USA: Prentice-Hall, 1994.
[63] B. Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.
[64] W. Li, "Another metric suite for object-oriented programming," *J. Syst. Softw.*, vol. 44, no. 2, pp. 155–162, 1998.
[65] D. P. Tegarden, S. D. Sheetz, and D. E. Monarchi, "A software complexity model of object-oriented systems," *Decis. Support Syst.*, vol. 13, nos. 3–4, pp. 241–262, 1995.
[66] A. Lake and C. Cook, "Use of factor analysis to develop oop software complexity metrics," in *Proc. 6th Annu. Oregon Workshop Softw. Metrics*, Silver Falls, OR, USA, 1994, pp. 251–266.
[67] K. Rajnish and Y. Singh, "An empirical and analytical view of new inheritance metric for object-oriented design," *Int. J. Comput. Appl.*, vol. 65, no. 12, pp. 44–50, 2013.
[68] S. Mal and K. Rajnish, "New quality inheritance metrics for object-oriented design," *Int. J. Softw. Eng. Appl.*, vol. 7, no. 6, pp. 185–200, 2013.
[69] P. Gulia and R. S. Chillar, "New proposed inheritance metrics to measure the software complexity," *Int. J. Comput. Appl.*, vol. 58, no. 21, pp. 1–4, 2012.
[70] F. T. Sheldon, K. Jerath, and H. Chung, "Metrics for maintainability of class inheritance hierarchies," *J. Softw. Maintenance Evolution, Res. Pract.*, vol. 14, no. 3, pp. 147–160, 2002.
[71] D. Mishra and A. Mishra, "Object-oriented inheritance metrics in the context of cognitive complexity," *Fundamenta Informaticae*, vol. 111, no. 1, pp. 91–117, 2011.
[72] K. Rajnish and V. Bhattacherjee, "Applicability of Weyuker property 9 to object-oriented inheritance tree metric—A discussion," in *Proc. 10th Int. Conf. Inf. Technol. (ICIT)*, 2007, pp. 234–236.
[73] J. Chen and J. Lu, "A new metric for object-oriented design," *Inf. Softw. Technol.*, vol. 35, no. 4, pp. 232–240, 1993.
[74] Y. Lee, "Measuring the coupling and cohesion of an object-oriented program based on information flow," in *Proc. Int. Conf. Softw. Qual.*, 1995, pp. 232–240.
[75] B. W. Boehm and P. N. Papaccio, "Understanding and controlling software costs," *IEEE Trans. Softw. Eng.*, vol. SE-14, no. 10, pp. 1462–1477, Oct. 1988.
[76] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," *J. Syst. Softw.*, vol. 51, no. 3, pp. 245–273, 2000.
[77] R. Shatnawi and W. Li, "The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process," *J. Syst. Softw.*, vol. 81, no. 11, pp. 1868–1882, 2008.
[78] M. Evett, T. Khoshgoftar, P.-D. Chien, and E. Allen, "GP-based software quality prediction," in *Proc. 3rd Annu. Conf. Genetic Program.*, 1998, pp. 60–65.
[79] T. M. Khoshgoftaar and N. Seliya, "Comparative assessment of software quality classification techniques: An empirical case study," *Empirical Softw. Eng.*, vol. 9, no. 3, pp. 229–257, 2004.
[80] T.-S. Quah and M. M. T. Thwin, "Application of neural networks for software quality prediction using object-oriented metrics," in *Proc. Int. Conf. Softw. Maintenance (ICSM)*, 2003, pp. 116–125.
[81] K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai, "Comparing case-based reasoning classifiers for predicting high risk software components," *J. Syst. Softw.*, vol. 55, no. 3, pp. 301–320, 2001.

[82] X. Yuan, T. M. Khoshgoftaar, E. B. Allen, and K. Ganesan, "An application of fuzzy clustering to software quality prediction," in *Proc. 3rd IEEE Symp. Appl.-Specific Syst. Softw. Eng. Technol.*, 2000, pp. 85–90.
[83] C. Catal, U. Sevim, and B. Diri, "Software fault prediction of unlabeled program modules," in *Proc. World Congr. Eng.*, vol. 1, 2009, pp. 1–6.
[84] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 25, no. 5, pp. 675–689, Sep./Oct. 1999.
[85] H. Kapila and S. Singh, "Bayesian inference to predict smelly classes probability in open source software," *Int. J. Current Eng. Technol.*, vol. 4, no. 3, pp. 1724–1728, 2014.
[86] R. Mahajan, S. K. Gupta, and R. K. Bedi, "Design of software fault prediction model using BR technique," *Procedia Comput. Sci.*, vol. 46, pp. 849–858, 2015.
[87] S. M. Jamali, "Object oriented metrics (a survey approach)," Citeseer, Tech. Rep., 2006.
[88] G. Boetticher. (2007). *The Promise Repository of Empirical Software Engineering Data*. [Online]. Available: http://promisedata.org/repository
[89] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Proc. 7th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, May 2010, pp. 31–41.
[90] J. S. Shirabad and T. J. Menzies, "The promise repository of software engineering databases," Ph.D. dissertation, School Inf. Technol. Eng., Univ. Ottawa, Ottawa, ON, Canada, 2005, vol. 24.
[91] D. N. Card and W. W. Agresti, "Measuring software design complexity," *J. Syst. Softw.*, vol. 8, no. 3, pp. 185–197, 1988.
[92] C. Catal, "Performance evaluation metrics for software fault prediction studies," *Acta Polytechnica Hungarica*, vol. 9, no. 4, pp. 193–206, 2012.
[93] R. Kumar and D. Gupta, "A heuristics based review on CK metrics," *Int. J. Appl. Eng. Res.*, vol. 7, no. 11, p. 2012, 2012.
[94] B. M. Goel and P. K. Bhatia, "Investigation of reusability metrics for object–oriented designing," in *Proc. NCETCIT*, May 2012, pp. 104–110.
[95] R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects," *IEEE Trans. Softw. Eng.*, vol. 29, no. 4, pp. 297–310, Apr. 2003.
[96] R. Bender, "Quantitative risk assessment in epidemiological studies investigating threshold effects," *Biometrical J., J. Math. Methods Biosci.*, vol. 41, no. 3, pp. 305–319, 1999.
[97] A. Kaur and I. Kaur, "An empirical evaluation of classification algorithms for fault prediction in open source projects," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 30, no. 1, pp. 2–17, 2018.
[98] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An empirical study on object-oriented metrics," in *Proc. 6th Int. Softw. Metrics Symp.*, 1999, pp. 242–249.
[99] P. Yu, T. Systa, and H. Muller, "Predicting fault-proneness using OO metrics. An industrial case study," in *Proc. 6th Eur. Conf. Softw. Maintenance Reeng.*, 2002, pp. 99–107.
[100] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proc. 6th Int. Conf. Predictive Models Softw. Eng.*, 2010, p. 9.
[101] T. Menzies and J. S. Di Stefano, "How good is your blind spot sampling policy," in *Proc. 8th IEEE Int. Symp. High Assurance Syst. Eng.*, 2004, pp. 129–138.
[102] T. Menzies, J. DiStefano, A. Orrego, and R. Chapman, "Assessing predictors of software defects," in *Proc. Workshop Predictive Softw. Models*, 2004, pp. 1–5.
[103] *Softlab*. Accessed: Feb. 10, 2009. [Online]. Available: http://softlab.boun.edu.tr
[104] N. Niu and A. Mahmoud, "Enhancing candidate link generation for requirements tracing: The cluster hypothesis revisited," in *Proc. 20th IEEE Int. Requirements Eng. Conf. (RE)*, Sep. 2012, pp. 81–90.
[105] S. Wagner, "A Bayesian network approach to assess and predict software quality using activity-based quality models," *Inf. Softw. Technol.*, vol. 52, no. 11, pp. 1230–1241, 2010.
[106] W. Abdelmoez, K. Goseva-Popstojanova, and H. Ammar, "Maintainability based risk assessment in adaptive maintenance context," in *Proc. 2nd Int. Predictor Models Softw. Eng. Workshop (PROMISE)*, Philadelphia, PA, USA, 2006.
[107] W. Abdelmoez, M. Shereshevsky, R. Gunnalan, H. H. Ammar, B. Yu, S. Bogazzi, M. Korkmaz, and A. Mili, "Quantifying software architectures: An analysis of change propagation probabilities," in *Proc. 3rd ACS/IEEE Int. Conf. Comput. Syst. Appl.*, Jan. 2005, p. 124.
[108] D. E. Monarchi and G. I. Puhr, "A research typology for object-oriented analysis and design," *Commun. ACM*, vol. 35, no. 9, pp. 35–48, 1992.
[109] 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.
[110] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Proc. 7th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, May 2010, pp. 31–41.
[111] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*. Amsterdam, The Netherlands: Elsevier, 2011.
[112] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, "The impact of correlated metrics on defect models," 2018, *arXiv:1801.10271*. [Online]. Available: https://arxiv.org/abs/1801.10271
[113] A. K. Luhach, D. Singh, P.-A. Hsiung, K. B. G. Hawari, P. Lingras, and P. K. Singh, in *Proc. 1nd Int. Conf. Adv. Informat. Comput. Res. (ICAICR)*, vol. 955. Shimla, India: Springer, Jul. 2018, 2018.
[114] D.-L. Miholca, G. Czibula, and I. G. Czibula, "A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks," *Inf. Sci.*, vol. 441, pp. 152–170, May 2018.
[115] R Core Team *et al.*, "R: A language and environment for statistical computing," Vienna, Austria, 2013.



SYED RASHID AZIZ received the M.Sc. degree in computer science from Al-Khair University, Islamabad, Pakistan, in 1998, and the M.S. degree in software engineering from COMSATS University, Islamabad, in 2008. He is currently pursuing the Ph.D. degree in software engineering with Bahria University, Islamabad. He has been involved in many national and enterprise level business application projects, since 1986 and provides consultancy to private, public military, and government organization for automation and teaching courses to students at various tiers. His research interests include big data, software fault tolerance, software reliability, software testing, the Internet of Things, service-oriented computing, and data warehousing.



TAMIM AHMED KHAN received the B.E. degree (Hons.) in software engineering from Sheffield University, U.K., in 1995, the M.B.A. degree in finance and accounting from Presston University, Islamabad, Pakistan, in 1997, the M.S. degree in computer engineering from CASE, Texila University, Pakistan, in 2006, and the Ph.D. degree in software engineering from Leicester University, U.K., in 2012. He is currently a Professor with the Department of Software Engineering, Bahria University, Islamabad, Pakistan. His research interests include service-oriented architectures, E-learning, and software quality assurance.

AAMER NADEEM received the M.Sc. degree in computer science from Quaid-i-Azam University (QAU), the M.S. degree in software engineering from the National University of Sciences and Technology (NUST), and the Ph.D. degree in computer science from Mohammad Ali Jinnah University (MAJU). During his Ph.D., he was a Visiting Scholar with The Chinese University of Hong Kong (CUHK) under a research collaboration. He is the Head of the Software Engineering Program at the Capital University of Science and Technology (CUST), where he is also the Head of the Center for Software Dependability (CSD), a research group, working in the areas of software reliability, software fault tolerance, formal methods, and safety-critical systems. He has over 30 years of teaching, research, and industry experience in computer science and software engineering. He has supervised 46 master's and two Ph.D. research theses in the areas of software testing, fault tolerance, and formal methods. He has authored or coauthored over 90 papers in reputable international journals and conferences. He is a Reviewer or Editorial Board Member of several international peer-reviewed journals and conferences. He is an Approved Ph.D. Supervisor for scholars funded by indigenous fellowship schemes of the Higher Education Commission (HEC) of Pakistan. He is a professional member of the Association for Computing Machinery (ACM).