Exclusive use and evaluation of inheritance metrics viability in software fault prediction - an experimental study

Syed Rashid Aziz Corresponding Author, Tamim Ahmed Khan 1, Aamer Nadeem 2

1 Department of Software Engineering, Bahria University, Islamabad, Pakistan
2 Department of Software Engineering, Capital university of Science and Technology, Islamabad, Pakistan

Corresponding Author: Syed Rashid Aziz
Email address: rashid_cbr@yahoo.com

Software Fault Prediction (SFP) assists in the identification of faulty classes, and software metrics provide us with a mechanism for this purpose. Besides others, metrics addressing inheritance in Object-Oriented (OO) are important as these measure depth, hierarchy, width, and overriding complexity of the software. In this paper, we evaluated the exclusive use, and viability of inheritance metrics in SFP through experiments. We perform a survey of inheritance metrics whose data sets are publicly available, and collected about 40 data sets having inheritance metrics. We cleaned, and filtered them, and captured nine inheritance metrics. After preprocessing, we divided selected data sets into all possible combinations of inheritance metrics, and then we merged similar metrics. Resultantly, we formed 67 data sets containing only inheritance metrics that have nominal binary class labels. We performed a model building, and validation for Support Vector Machine (SVM). Results of Cross-Entropy, Accuracy, F-Measure, and AUC advocate viability of inheritance metrics in software fault prediction. Furthermore, ic, noc, and dit metrics are helpful in reduction of error entropy rate over the rest of the 67 feature sets.
Exclusive Use And Evaluation of Inheritance Metrics Viability In Software Fault Prediction - An Experimental Study

Syed Rashid Aziz\textsuperscript{1}, Tamim Ahmed Khan\textsuperscript{2}, and Aamer Nadeem\textsuperscript{3}

\textsuperscript{1}Department of Software Engineering, Bahria University, Islamabad, Pakistan. \\
\textsuperscript{2}Department of Software Engineering, Bahria University, Islamabad, Pakistan. \\
\textsuperscript{3}Department of Software Engineering, Capital University of Science & Technology, Islamabad, Pakistan.

Corresponding author: Syed Rashid Aziz\textsuperscript{1} \\
Email address: e-mail: rashid_cbr@yahoo.com)

ABSTRACT

Software Fault Prediction (SFP) assists in the identification of faulty classes, and software metrics provide us with a mechanism for this purpose. Besides others, metrics addressing inheritance in Object-Oriented (OO) are important as these measure depth, hierarchy, width, and overriding complexity of the software. In this paper, we evaluated the exclusive use, and viability of inheritance metrics in SFP through experiments. We perform a survey of inheritance metrics whose data sets are publicly available, and collected about 40 data sets having inheritance metrics. We cleaned, and filtered them, and captured nine inheritance metrics. After preprocessing, we divided selected data sets into all possible combinations of inheritance metrics, and then we merged similar metrics. Resultantly, we formed 67 data sets containing only inheritance metrics that have nominal binary class labels. We performed a model building, and validation for Support Vector Machine(SVM). Results of Cross-Entropy, Accuracy, F-Measure, and AUC advocate viability of inheritance metrics in software fault prediction. Furthermore, ic, noc, and dit metrics are helpful in reduction of error entropy rate over the rest of the 67 feature sets.

1 INTRODUCTION

Object-Oriented software development is a widely used software development technique. It is not possible to produce a completely fault-free software system. The failure rates in a thousand lines of code for industrial projects are 15-50, 10-20 for Microsoft (McConnell, 2004), and specifically in Windows 2000, which comprises thirty-five million Lines of Code(LOC) recorded sixty-three thousand errors (Foley, 2007). Residual errors may cause failure since many types of errors have been newly detected (Bilton, 2016) (Osborn, 2016) (Grice, 2015).

Early detection of faults may save time, costs, and decrease the software complexity since it is proportional to the testing. Extensive testing is required to locate all remaining errors. The extensive tests are impossible (Jayanthi and Florence, 2018) (Kaner et al., 2008). This is why the cost of testing at times elevated to over 50\% of the total software development cost (Majumdar, 2010). This number may rise to seventy-five percent as reported by IBM (Hailpern and Santhanam, 2002). Software testing is essential, and inevitable to produce software without errors. Thorough testing of entire classes with limited manpower is a challenging task. It is more feasible to identify faulty classes, and test them to produce software with good quality. It is observed those faults are not uniformly dispersed throughout the software product. Certain classes are more faulty as compare to others, and are clustered in a limited number of classes (Sherer, 1995). Research shows that errors are limited to 42.04\% of the entire components in a software project (Gyimothy et al., 2005). Similarly, findings of Ostrand et al. discovered that only 20\% of the entire software components are faulty in a project (Ostrand et al., 2004).

In software engineering, presently numerous prediction approaches exist that include reuse prediction, test effort prediction, security prediction, cost prediction, fault prediction, quality prediction, and stress.
SFP is an up-and-coming research area, in which defective classes are discovered in the initial period of project development (Menzies et al., 2007) (Jing et al., 2014) (Seliya et al., 2010) with machine learning (Catal, 2011) (Malhotra, 2015). It is assumed that with the help of metrics, fault prediction model may be constructed to calculate coupling, inheritance, cohesion, size, and complexity of software (Chen and Huang, 2009) (Chidamber and Kemerer, 1994) (Li and Henry, 1993) (Basili et al., 1996) (Malhotra and Jain, 2012).

The fault prediction process typically includes the training, and prediction phases. In the first phase of training, a prediction model is constructed utilizing software metrics at class or method level together with fault information associated with each module of the software. Later, the said model is employed on a newly developed software version for faults prediction.

We utilize methods of classification for labeling classes into fault-free or faulty classes by using software metrics and fault information. Quality of software is improved by detecting classes with faults within the software using fault prediction models. The model performance has an effect on the modeling technique (Elish and Elish, 2008a)(Catal and Diri, 2009) and metrics (Briand et al., 2000)(Ostrand et al., 2005). Several researchers have developed and validated fault prediction models based on statistical techniques or machine learning. They have used software metrics, data sets, and feature reduction methods to achieve improvement in the performance of model.

The software development industry is utilizing Object-Oriented paradigm, however the usage of metrics at class-level in fault prediction is less common while comparing with other types of metrics (Catal and Diri, 2009). Our literature review findings show that method level software metrics employing machine learning techniques are in use by the researchers (Catal, 2011). It is beneficial to use publicly available data sets for SFP since they help to develop verifiable and repeatable models (Catal and Diri, 2009). It is pertinent to mention here that models of machine learning have superior precision and accessibility as comparing it with expert opinion-based approaches or statistical methods.

Besides others, inheritance is a prominent feature of Object-Oriented programming. Inheritance is the class’s ability to obtain the properties of some other class. It is divided into several types, namely single, multiple, multi-level, and so on. Object-Oriented supports reuse in three different ways (Karunamithi and Bieman, 1993), where the primary technique of reuse is through inheritance. It is the relationship between classes, in which a class object gets properties from other classes(Rosenberg, 1998). Forming classes into a classification hierarchy, it provides an additional dimension to the encapsulation of the abstract data types. It allows classes to inherit methods and attributes from other classes (Breesam, 2007). Use of inheritance reduces the essential software maintenance cost and testing burden (Chidamber and Kemerer, 1991) (Dale and Van Der Zee, 1992) (Fenton, 1994). Software reuse through inheritance yields more maintainable, comprehensible, and reliable software (Chidamber and Kemerer, 1991).

We validated inheritance metrics impact in SFP by experiments, in which we used Artificial Neural Network (ANN) for the construction of models. We used Accuracy, Recall, True negative rate (TNR), Precision, and F1-Measures as performance measures (Aziz et al., 2019). The outcomes indicated the satisfactory influence of metrics of inheritance on software fault prediction. The professionals associated with testing may securely utilize metrics of inheritance for fault prediction in software projects. A higher value of inheritance is also undesirable, as consequently, it might introduce faults in software.

Generally utilized metrics of inheritance include Number of Children (noc), Depth of Inheritance (dit), Total Progeny Count (tpc), Class-to-leaf depth (cld), Total Ascendancy Count (tac), Number of Descendant Classes (nac), Number of Ancestor Classes (nac), Number of Overridden Methods (norm), Number of Methods Inherited (nmi) and Number of Attributes Inherited (nai), Average Inheritance Depth (aid), Number of children in the hierarchy (nocc), Maximum DIT (MaxDit), Specialization Ration (SRatio), Total length of Inheritance chain(tii), Reuse Ration (RRatio), Method Inheritance Factor (mif), Attribute Inheritance Factor (aif), Number of Attributes Inherited in Hierarchy (naih), Specialization Index (si) and Number of Methods Inherited in Hierarchy (nmih) (Reddy and Ojha, 2018).

It is noticed during the survey on SFP that researchers have already performed so much work on the metrics of Object-Oriented software for example cohesion, coupling, etc. All these are utilized either individually or in a group with other metrics. The C&K metrics set is an example where these are used in a group and commonly recognized in the research arena. However, It is perceived that inheritance metrics are utilized collectively in C&K suite, but solo use and assessment of inheritance metrics is neglected. This prompts to perform experimentation to concentrate explicitly on inheritance aspect to demonstrate the viability of its metrics in the perspective of SFP.
This paper is a novel representation of the following:

1. Exploring inheritance metrics having publicly available data sets.
2. Exclusive experimental evaluation of the inheritance metrics on as large as 40 data sets.
3. An empirical validation of exclusive viability of inheritance aspect on software fault prediction.
4. Across the data sets/products software fault prediction.

The paper is categorized into sections where Section 2 explains theoretical context of SFP, Inheritance, Inheritance metrics, and their usage and data sets in SFP. The literature review explaining inheritance in SFP is presented in Section 3. Section 4 explains experimental approach utilized in this study and an experimental assessment of the inheritance measures in Section 5. Finally, Section 6 explains the threats to validity, concluding remarks, and future directions of the investigation.

2 THEORETICAL BACKGROUND

2.1 Software Fault Prediction

The process of SFP typically includes two phases: training and prediction. In the training phase, a method or class level software metrics with faulty data associated with the individual module of the software are used to construct a prediction model. Later, this model is used for the prediction of fault-prone classes in a newer version of the software. Fault prediction is beneficial to improve software quality and reduce testing costs. Moreover, it enables testing teams to limit testing on fault-prone classes only. In software, faults prediction theoretically established the target to grasp which parts may require concentration. Several SFP methods have been used (Rathore and Kumar, 2017a) (Catal, 2011), which share three main ingredients (Beecham et al., 2010): Feature set, Class label, and Model.

The feature set consists of the metrics from a software artifact. It is believed that these are decent class label predictors. The metrics are classified into product, project, and process metrics. Out of these product metrics are mostly utilized (Gómez et al., 2006). The product metrics are further grouped into class, method, and file levels. Generally, 60% metrics at method-level are applied after that 24% metrics at class-level (Catal and Diri, 2009). Product metrics also comprise design, code, volume, and complexity metrics. The fault prediction model performance is so much dependent on these metrics. Scholars have evaluated the usage frequency of metrics in (Malhotra, 2015) (Catal and Diri, 2009) (Beecham et al., 2010) (Radjenović et al., 2013) (Gondra, 2008) (Chappelly et al., 2017) (Nair et al., 2018), where the utmost commonly applied software product metrics in software fault prediction are Halstead (Halstead, 1977), McCabe (McCabe, 1976), LoC in structural programming, and C&K metrics suite (Chidamber and Kemerer, 1991) in Object-Oriented paradigm. These metrics have become the de-facto standard metrics in SFP. PROMISE (Boetticher, 2007) and D’Ambros (D’Ambros et al., 2010) are more often used data sets repositories containing these metrics. These repositories contain data sets of about 52% of the studies published after 2005 (Malhotra, 2015). Since these data sets are available publicly therefore they are used very frequently. The second reason is the absence of fault data of industrial projects of software.

The second very significant element is the class label in software fault prediction, which contains metrics actual value. Within the software fault prediction domain, fault free or faulty are shown as continuous or nominal-binary or to point total faults in an occurrence. Though, usage of continuous labels present in the literature (Rathore and Kumar, 2017b), but leading class labels are nominal class labels in SFP (Catal, 2011) (Rathore and Kumar, 2017a).

In software fault prediction, model building is a third key factor, which is a relationship between feature set and label of class. It can be applied by the use of Machine learning (ML) algorithms, statistical methods, or even expert opinion (Catal and Diri, 2009), where ML is an extensively utilized method for model building (Beecham et al., 2010). It expressively expands the accuracy of classification (Han et al., 2011). In the SFP domain, several ML algorithms are utilized. The performance of these ML algorithms has been compared by Malhotra, whose findings conclude that Bayesian networks and Random forest are outperformers while compared with other algorithms of ML (Malhotra, 2015).

The findings of literature review disclosed that 22% statistical methods are utilized (Grice, 2015), and 59% is machine learning. Several performance measures and machine learning methods are explored that make use of object-oriented metrics to predict faults. These studies are cataloged into multiple tables.
The studies from 1990-2003 are shown in Table 1, studies from 2004 to 2007 are listed in Table 2 and finally studies from 2008 to 2020 are listed in Table 3 (Aziz et al., 2020).

Table 1. SFP studies (1990-2003)

| Reference                | Algorithms                              | Performance Measure                  |
|--------------------------|-----------------------------------------|--------------------------------------|
| Porter and Selby (1990)  | Classification Tree                     | Accuracy                             |
| Briand, Basil and Hetmanski (1993) | Logistic Regression, Classification Tree, Optimized Set Reduction | Correctness, Completeness |
| Lanubile, Loniqro, and Visaggio (1995) | PCA, Discriminant Analysis, Logistic Regression, Logical Classification | Misclassification Rate |
| Cohen and Devanbu (1997) | Foi, Fliper on IPL                       | Error Rate                           |
| Khoshgoftar, Allen Hudepohl, Aud 1997 | ANN and Discriminant | Type-I, Type-II, Misclassification Rate |
| Evett, Khoshgoftar, Chien, Allen (1998) | Ordinal Evaluation Procedure            |                                      |
| Ohlson, Zhai, Helander 1998 | PCA, Discriminant Analysis for Classifica, Multivariate Analysis | Misclassification Rate |
| Blinkley, Schach 1998    | Spearman Rank Correlation Test           |                                      |
| De Almeida and Matwin 1999 | C4.5, CN2, FOIL, NewID                   | Correctness, Completeness, Accuracy  |
| Kaszycki 1999            |                                        | TN, TP                               |
| Yuan, Khoshgoftar, Allen, Ganesan 2000 | Fuzzy Subtractive Clustering            | Type-I, Type-II, Overall Misclassification Rate, Effectiveness, Efficiency |
| Denaro 2000              | Logistic Regression                      | R2                                   |
| Khoshgoftar, Allen Bushboom 2000 | PCA, ENR                                |                                      |
| Xu, Khoshgoftar, Allen 2000 | Average Absolute Error                 |                                      |
| Guo, Lyu 2000            | Finite Mixture Model Analysis, Expectation Maximization (EM) | Type-II Error |
| Khoshgoftar, Gao, Szabo 2001 | ZIP                                     | AAE, ARE                             |
| Schneidewind              | BDF, LRF                                | Type-I, Type-II, Misclassification Rate |
| Elnam, Melo, Machado 2001 | Logistic Regression                     | J-Coefficient                        |
| Khoshgoftar, Geley, Gao 2002 | PRM, ZIP, Module Order Modelling       | Average Relative Error               |
| Khoshgoftar 2002          | GBDF                                    |                                      |
| Mahaweerawar, Sohasathit, Lursinsap 2002 | Fuzzy Clustering, RBF                | Type-I, Type-II, Misclassification Rate |
| Khoshgoftar, Selviya 2002a | SPRINT(Classification Tree), CART(Decision Tree) | Type-I, Type-II, Misclassification Rate |
| Pizzi, Summers, Pedrycz 2002 | Median-Adjusted Class Labels(Pre-Processing), Multilayer Perception | Accuracy |
| Khoshgoftar, Selviya 2002b | CART-LS, S-PLUS, CART-LAD               | AAE, ARE                             |
| Reformats 2003            | Classification Models                   | Rate Change                          |
| Koru, Tian 2003          | Tree Base Models                        | U-Test                               |
| Denaro, Lavazza, Pezze 2003 | Logistics Regression                  |                                      |
| Thwin, Quah 2003         | GRNN                                    | R2, R, ASE, AAE, Min AE, Max AE     |
| Khoshgoftar, Selviya 2003 | CART-LS, S-PLUS, CART-LAD               | Probability of Detection, Accuracy   |
| Denaro, Pezze, Morasca 2003 | Logistic Regression                     | R2, Completeness                     |

2.2 Software Inheritance

Inheritance makes it possible to make use of the components of previous objects by calling these into recently created objects in an object-oriented paradigm. The superclass or base class is a source of inheritance and a subclass or derived class inherits from a superclass. The term main class and secondary class can also be used interchangeably for super and subclass. The sub-class can possess its components and methods in addition to inherit visible methods and properties from the main class. Inheritance offers (Aziz et al., 2019):

1. **Reusability**: reuse is a resource available by the inheritance where superclass’s public methods are utilized into subclass without code rewriting.
2. **Overriding**: define a new behavior for a method that already exists. This happens when the class in question extends any other class and creates a method with the same signature as the "parent" class in the subclass.

3. **Extensibility**: extend the logic of the supper class according to the business logic of the sub-class.

### Table 2. SFP studies (2004-2007)

| Reference | Algorithms | Performance Measure |
|-----------|------------|----------------------|
| Menzies, Disterfano, Orrego, Chapman 2004 | Naïve Bayes, J48 | PF |
| Khosghoftaar, Selvia 2004 | CART, S-PLUS, SPRING-Sliq, C4.5 | Misclassification Rate |
| Wang, Yu, Zhu 2004 | CGA, ANN | Accuracy |
| Mahaweeraat, Sophatsathit, Lursin sap, Musilek 2004 | RBF | Accuracy, Type-I, Type-II |
| Menzies, Distefano 2004 | LSR, Model Trees, ROCKY | Accuracy, Sensitivity, Precision |
| Kamiński, Boetticher 2004 | Genetic Algorithm | T-Test |
| Kamnani, Uthariaraj, Thambidurai 2004 | GRNN, PCA | r, R2, ASE, AAE, Max AE, Min AE |
| Zhong, Khosghoftaar, Selvia 2004 | K-means, Neural-Gas clustering | MSE, FPR, FNR, Misclassification Rate |
| Xing, Guo, Lyu 2005 | SVM, QDA, Classification Tree | Type-I, Type-II Error |
| Koru, Liu 2005 | J48, Kstar, Bayesian Networks, ANN, SVM | F-measure |
| Khosghoftaar, Selvia, Gao 2005 | C4.5, Decision Tree, Discriminant Analysis, Logistic Regression | Misclassification Rate |
| Koru, Lin 2005 | J48, K-Star, Random Forests | F-measure |
| Challagulla, Bastani, P, Paul 2005 | Linear Regression, SVM, Naïve Bayes, J48 | AAE |
| Gymothy, Ference, Sliket 2005 | Logistic Regression, Linear Regression, Decision trees, NN | Completeness, Correctness, Precision |
| Ostrand, Weyuker, Bell 2005 | Negative Binomial Regression Model | Accuracy |
| Tomaszewski, Lundberg, Grahn 2005 | Regression Technique, PCA | R2 |
| Hassan, Holt 2005 | Rule, Rate, APA | |
| Ma, Guo, Cukic 2006 | Random Forests, LR, DA, Naïve Bayes, J48, ROCKY | G-mean, G-mean II, F-measure, ROC, PD, Accuracy |
| Challagulla, Bastani 2006 | MBR | PD, Accuracy |
| Khosghoftaar, Selvia, Sundaresh 2006 | MLR, CBR | AER, AAE |
| Nikora, Munson 2006 | Rules for Fault Definition | |
| Zhou, Leung 2006 | Logistic Regression, Naïve Bayes, Random Forest | Correctness, Completeness, Precision |
| Mertik, Lenic, Stigl, Kokol 2006 | C4.5, SVM, RBF | PD, PF, Accuracy |
| Gao, Khosghoftaar 2007 | Poisson Regression, Negative Binomial Regression, Harline Regression | AAE, AER |
| Li, Reformat 2007 | SimBoost | Accuracy |
| Mathaweeraat, Sophatsathit, Lursin sap 2007 | RBP, Self-Organizing Map Clustering | MAR |
| Menzies et al. 2007 | Naïve Bayes, J48 | PD, PF, Balance |
| Ostrand, Weyuker, Bell 2007 | Negative Binomial Regression Model | accuracy |
| Pal, Dugan 2007 | Linear Regression, Poisson Regression, Logistic Regression | Sensitivity, Specificity, Precision, TP, FN |
| Wang, Zhu, Yu 2007 | S-PLUS, TreeDisc | Type-I, Type-II |
| Selvia, Khosghoftaar 2007 | EM Techniques | Type-I, Type-II |
| Tomaszewski, Hakansson, Grahn, Lundberg 2007 | Univariate Linear regression Analysis | Accuracy |
| Selvia, Khosghoftaar 2007 | Semi-Supervised Clustering, K-means Clustering | Type-I, Type-II |
| Oluage, Ghoshon, Quattlebaum 2007 | UBIR, Spearman Correlation | Accuracy |
| Jiang, Cukic, Menzies 2007 | Naïve Bayes, Logistic Regression, J48, IBK, Random Forests | PD, PF |
4. **Maintainability**: it is straightforward to walk-through the source code as soon as the software program is split into portions.

5. **Data hiding**: Inheritance presents a feature to hiding data by marking a method as a private in the main class so that it cannot be utilized or alter by the sub-class.

In the object-oriented paradigm, the basis of inheritance is an "IS-A" bond, which describes "R is a Z type of thing", blue is a color, the bus is a vehicle. The inheritance is uni-directional, "the house is a building", but "the building is not a house" etc. The inheritance has other additional important characteristics(Aziz et al., 2019):

1. **Generalization**: dissemination of commonalities amongst several classes is termed as generalization (Pason, 1994).

### Table 3. SFP studies (2008-2020)

| Reference | Algorithms | Performance Measure |
|-----------|------------|---------------------|
| Bibi, Tsoumakas, Stamelos, Vlahv as 2008 | RVG | AAE, Accuracy |
| Bingbing, Qian, Shengyong, Pang 2008 | K-means, Affinity Propagation | Type-I, Type-II |
| Marcus, Poshvanyk, Ference 2008 | Univariate Logistic Regression | Precision, Correctness |
| Shafi, Hassan, Arshag, Shammai 2008 | Classification Via Regression | Precision, Recall, Accuracy |
| Catal, Diri 2009 | Random Forests [Artificial Immune Systems], Naïve Bayes | AUC |
| Turihan, Kocak, Ener 2009 | CBGR, Nearest Neighbor Sampling | |
| Catal 2009 | X-means Clustering, | |
| Catal, Diri 2009 | Naïve Bayes, YATSI | |
| Alan, Catal 2009 | | |
| Yeresime Suresh, Lov Kumar (2014) | Linear Regression, Logistic Regression, ANN | Precision, Correctness, Completeness, MAE, MAE, MRE, MRE, SEM |
| Salqa Aleem, Luis Fernando (2015) | NaiveBayes, MLP, SVM, AdaBoost, Bagging, Decision Tree, Random Forest, J48, KNN, RBF and K-means | Accuracy, Mean absolute error and F-measure |
| Santosh, Sandeep (2015) | Genetic Programming (GP) | Error rate, Recall, Completeness |
| Yohannese, Tinnui Li (2016) | NB, NN, SVM, RF, KNN, DT, DTa, and RTr | ROC |
| Ankit Pahal, R. S. Chilhar (2017) | ANN, SSO | |
| Yogomaya Mohapatra (2018) | GSO-GA, SVM | Accuracy, Value, Accuracy |
| Balthomiej Wójcicki, Robert (2018) | Machine Learning | Recall, False Positive Rate |
| Patil, Nagaraja (2018) | Linear Regression, FCM | Accuracy, Precision, Standard Errors And T-Values |
| Bahman Arasteh (2018) | Naïve Bayes, ANN, SVM | Accuracy, Precision, Specifity, Recall, F-Measure |
| Hiba Alghaier, Akour (2019) | GA, SVM | Area Under Curve (AUC) |
| Balogun, Shuib Basri (2019) | RIPPER, Bayesian Network, Random Tree, and Logistic Model Tree | |
| Rajkumar N, Vijil C (2019) | SVM, ANN, KNN | Accuracy, Sensitivity, Specificity, Precision |
| Aliaeedl, Zubair (2019) | SVM, DS, RF | Accuracy, Precision, Recall, F-Score, ROC-AUC |
| Wasiur Rhaman, Babita Pandey (2020) | Random Forest, J48 | Precision, Recall |
| Deepak, Pravin (2020) | Factor Analysis (FA) | R2, Adjusted R2 |
| Ruzo, Asraf (2020) | SVM | Precision, Recall, Specificity, F1 Measure, Accuracy |
2. **Specialization**: increasing the functionality of a class is described as specialization (Breesam, 2007).

The research shows that inheritance has various forms. There are described in the subsequent lines (Shivam, 2013):

1. **Single Inheritance**: when a sub-class only inherits through a single main-class is denoted as single inheritance.

2. **Multiple Inheritances**: in case of multiple inheritances, a sub-class is inheriting or expanding through many main-classes. The problem in multiple inheritance is the sub-class would manage the dependencies of many main-classes.

3. **Multilevel Inheritance**: multilevel inheritance in object-oriented paradigm, refers to an approach when a sub-class spread out from a derived class, making the derived class a main class of the freshly formed class.

4. **Hierarchical Inheritance**: in the situation of hierarchical inheritance one superclass is expanded by means of many sub-classes.

5. **Hybrid Inheritance**: is a mixture of multi-layer and multiple inheritance. In multiple inheritance, subclasses are expended from two superclasses. Although these superclasses are derived classes rather than the base classes.

### 2.3 Inheritance Metrics

#### Depth of Inheritance Tree (DIT) (Chidamber and Kemerer, 1994):

The DIT metrics is a measurement of how significantly subclasses may efface the metrics of this class. In the case of multiple inheritances, DIT would be the maximum distance from the node to the root of a tree.

\[
DIT = \text{Max inheritance path from the class to the root}
\]  

- The lowest class in the hierarchy would inherit a larger amount of methods, consequently, it would be hard to predict their behavior.
- While in the design phase, deeper trees will create more complexity since several classes and methods are being used.
- The lower a particular class is in the hierarchy, the higher possibility of reuse of inherited methods.

#### Number of Children (NOC) (Chidamber and Kemerer, 1994)

- Increasing the NOC will rise in reuse since inheritance is a type of reuse.
- The larger the sub-class, the bigger the probability of inadequate abstraction of the main-class. In the case where a class has a large number of subclasses, it would be a situation of misappropriation of a child-class.
- The quantity of subclasses presents an impression of the possible impact on the design of a class. In case when a class contains larger numbers of subclasses, it would require further testing of methods present within the class.

\[
NOC = \text{number of immediate sub – classes of a class}
\]  

#### Attribute Inheritance Factor (AIF) (Abreu and Carapuça, 1994): AIF is the ratio of the sum of all classes inherited attributes in the system to the all classes total number of available attributes. It is a metric at system-level which gauges the range of inherited attribute within the system. The equation to calculated AIF is as under:

\[
AIF = \frac{\sum Ai(Ci)}{\sum Aa(Ci)}
\]
Method Inheritance Factor (MIF) (Abreu and Carapuça, 1994): MIF is the ratio of sum of all classes inheritance methods of the system with the total number of all classes presented methods. It is a metric at the system-level. It is proposed to maintain MIF in between 0.25 and 0.37. The equation to calculate MIF is under:

\[
MIF = \frac{\sum Mi(Ci)}{\sum Ma(Ci)}
\]  

Number Of Methods Inherited (NMI) (Lorenz and Kidd, 1994): NMI metric calculates the total methods inherited by a sub-class.

Number of Methods Overridden (NMO): a larger value of NMO reveals a design issue, showing that these methods were overridden as a last-minute design. It is recommended that a sub-class should be a specialization of its main classes, which results in a brand-new distinctive name for the methods.

Number of New Methods (NNA): the usual anticipation of a sub-class is how to additionally specialize or add up objects of the main class. If there is not any method with a similar name in any superclass, the said method is defined as an additional method in the subclass.

Inheritance Coupling (Li and Henry, 1993): is an association between classes that facilitates to use of earlier defined objects, consist of variables and methods. Inheritance reduces class complexity by decreasing the number of methods of a single class, but then this becomes design and maintenance difficult. Inheritance improves reusability and efficiency when using current objects. Simultaneously, inheritance has led to complexities in testing and understanding software. This implies that inheritance coupling affects several quality attributes such as complexity, reusability, efficiency, maintainability, understandability, and testability.

Number Of Inherited Methods (NIM): is a simple metric that describes the extent to which a particular class may reuse. It calculates the number of methods a class may gain access to in its main class. The greater the inheritance of methods, the greater the reuse of a class will be via subclasses. Comparing this metric amongst the number of superclasses referred and the manner it is referred to methods not specified in the class could be exciting since it shows how much internal reuse occurred within the calling class and its superclass. It may be an inward call to an inward method, even though it is hard to measure it. Also, inheriting from larger superclasses could be a problem for the reason that only a subset of the behavior may be used/needed in subclasses. This is a limit of the single inheritance based on the object-oriented paradigm.

Fan-In and Fan-Out Metric (Henry and Kafura, 1981): Henry and Kafura first defined the Fan-In and Fan-Out metrics (Henry and Kafura, 1981). These are "module-level" metrics and expanded for the object-oriented paradigm. Assuming a class X, we note its Fan-In such as the number of classes, that make use of characteristics of class X. Likewise, the Fan-Out for a class X is the number of classes utilized by X.

Sheetz, Tegarden, and Monarchi originate a set of basic counts. Fundamental complexity or inter-module complexity (Card and Glass, 1990) has been recognized as an important part of the complexity of a structured system. Several researchers have utilized module-defined Fan in and Fanout (Belady and Evangelist, 1981)(Card and Glass, 1990)(Monarchi and Puhur, 1992). Extending these ideas to variables in object-oriented systems seems appropriate and straightforward. The number of methods using variables (variable Fan-In) is very similar to the number of modules calling the module (Fan-In), and the number of objects accessed by the variable (variable Fan-Out) and the digital module called by the module (Fan-Out).

Fan-Down: Fan-Down is the number of objects below the object hierarchy (subclasses).

Fan-Up: Fan-Up is the number of objects above in hierarchy (superclasses).

Class-To-Root Depth: the maximum number of levels in the hierarchy that are above the class.
Class-To-Leaf Depth: the maximum number of levels in the hierarchy that are below the class.

Measure of Functional Abstraction (MFA): MFA is the share of the count of methods inherited by a class with the sum of methods of the class. Its range is from 0 to 1.

IFANIN: IFANIN metric counts the immediate base classes in the hierarchy.

2.4 Inheritance Metrics And Their Usage

Inheritance is a key characteristic of the object-oriented paradigm. It facilitates the class level design and forms the "IS-A" relationship among classes since the basic segment of the development of a system is the design of class (Rajnish et al., 2010). The utilization of inheritance shrinkages the costs of testing efforts and maintenance of the system (Chidamber and Kemerer, 1994). The reuse employing inheritance will thus deliver software, that is greatly understandable, maintainable, and reliable (Basili et al., 1996). In an experiment, Harrison et al. describe the absence of inheritance as easier to control or grasp as compared to the software that makes use of inheritance aspect (Harrison et al., 1998). However, Daley’s experiments reveal software with tertiary inheritance may possibly be simply revised as compared to the software with no inheritance aspect (Daly et al., 1996).

The Inheritance metrics calculate numerous aspects of inheritance, that include breadth and depth of the hierarchy, besides the overriding complexity (Krisha and Joshi, 2010). Similarly, Bhattacherjee and Rajnish executed a study about inheritance metrics related with classes (Rajnish et al., 2007) (Rajnish and Bhattacherjee, 2008) (Rajnish and Bhattacherjee, 2007) (Rajnish and Bhattacherjee, 2006a) (Rajnish and Bhattacherjee, 2006b) (Rajnish and Bhattacherjee, 2005). However, it is agreed the deeper the inheritance hierarchy, the class reusability will be enhanced but system maintainability will be complicated. In order to streamline the insight, software designers striving to keep inheritance hierarchy narrow and dispose of reusability by the usage of inheritance (Chidamber and Kemerer, 1994). Hence, it’s important to assess the difficulty of the inheritance hierarchy to resolve the disparity among depth and shallowness.

Several metrics focus on inheritance are well-defined by the researchers. These metrics with their references are listed in Table 4 (Aziz et al., 2019). Some of these Inheritance metrics listed in Table 4 are discussed briefly in Section 2.3 of the paper. We specifically take these inheritance metrics in the background of object oriented software fault prediction.

2.5 Data Sets In SFP

In software fault prediction, so many data sets are being used. These data sets are categorized into public, private, partial, and unknown data sets types Catal and Diri (2009). Out of these, public type data sets utilization has increased from 31% to 52% since 2005 onward (Malhotra, 2015). It is a fact that fault information normally not available for private projects however there are public data sets widely available with fault information, these can be downloaded for free. Also, there are many fault repositories, out of this Tera-PROMISE (Boetticher, 2007) warehouses, and D’Ambros warehouses are usually utilized for fault predicting (D’Ambros et al., 2010).

A publicly available repository identified as Tera-PROMISE presents substantial data sets of many projects. Its previous edition was named as NASA repository (Shirabad and Menzies, 2005). The data sets of NASA are a vital resource of the Tera-PROMISE repository because its data sets are a widely used library of SFP. Nearly 60% of articles published between 1991 and 2013 take advantage of this archive (Card and Agresti, 1988). The library of Tera-PROMISE presents metrics related to product and process along with digital and nominal class labels for buildups regression and classification models.

The D’Ambros library retains data sets of six software systems. These are Equinox, Eclipse JDT Core, Eclipse PDE UI, Framework, and Mylyn.

During the review of the literature, it is identified that scholars make use of private, and public data sets for the proof of their study. In this respect, Table 5 indicates the name of the author, publication year, and public data sets or private data sets applied in said experiments.

3 LITERATURE REVIEW

In this section, emphases are on inheritance metrics to find out how these might be effective in SFP. This paper is not performing a systematic literature review. It explores how various inheritance metrics are advantageous in fault prediction.
### 3.1 Inheritance In SFP

Object-Oriented metrics are employed for the prediction of faults to produce quality software. The attributes that ascertain software quality are fault tolerance, understandability, defect density, maintainability, normalized rework rate, reusability, and many more others.

There are several metrics levels comprising class level, method level, file level, process level, component level, and quantitative levels. The method-level metrics are comprehensively applied for the prediction of faults problem. Halstead (1977) and McCabe (1976) metrics suggested in the year 1970s though these are still the greatest predominant metrics at the method-level. The class-level metrics are merely applied in object-oriented programs. The C&K (Chidamber & Kemerer, 1994) set of metrics is yet the ultimate predominant metrics suite at class-level being employed for fault prediction. Table 6 recapped the commonly employed metrics set at class, method, and file-level for software fault prediction. Many studies have been carried out on SFP which also takes into account the object-oriented metrics. These studies includes, empirical study on open source software for fault prediction using \( \{ \text{loc}, \ \{ \text{dit} \}, \ \{ \text{noc} \}, \ \{ \text{lcom} \} \) and \( \{ \text{cbo} \} \) metrics (Gyimothy et al., 2005). The reusability study on object-oriented software using inheritance, cohesion, and coupling metrics (Catal, 2012). The experimental-based assessment of C&K metrics (Kumar and Gupta, 2012), reusability metrics for object-oriented design (Goel and Bhatia, 2012), and empirical analysis of C&K metrics for the object-oriented design complexity (Subramanyam and Krishnan, 2003).

### Table 4. Inheritance Metrics

| Author Name                      | Metrics Name                      |
|----------------------------------|-----------------------------------|
| Chidamber and Kemerer (1994)     | Depth of Inheritance Tree (DIT)   |
| Abreu Mood (1994)                | Number of Children (NOC)          |
| Bansiya and Davis (2002)         | Average number of Ancestors (ANA) |
| Henry’s & Kafura (1981)          | Fan in                            |
| Tang, Kao and Chen (1993)        | Fan out                           |
| Lorenzo and Kidd (1994)          | Number of Method Inherited (NMI)  |
| Henderson-Sellers (1995)         | AID (average inheritance depth)   |
| Li (1998)                        | NAC (number of ancestor classes)  |
| Tegarden et al. (1995)           | CLD (class-to-leaf depth)         |
| Lake and Cook (1994)             | NOA (number of ancestor)          |
| Rajnish et al. (2013)            | DITC (Depth of Inheritance Tree of a Class) |
| Sandip et al. (2011)             | CH (Class Inheritance)            |
| Gulia, Preeti, and Rajender S. Chillar (2012) | CCI (Class Complexity due to Depth of Inheritance Tree) |
| F. T. Sheldon et al. (2002)      | Average Degree of Understandability (AU) |
| Rajnish and Choudhary (2010)     | Derive Base Ratio Metric (DBRM)   |
| Mishra, Deepit, and Alok Mishra (2011) | CCI (Class Complexity due to Inheritance) |
| Abreu and Carapuça (2011)        | Total Children Count (TCC)        |
| Abreu and Carapuça (2011)        | Total Progeny Count (TPC)         |
| K. Rajnish and A. K. Choudhary (2010) | Total Ascendancy Count (TAC)    |
| Rajnish and Bhattacharjee (2007) | Inheritance Metric Tree (IMT)     |
| Lee et al. (1995)                | Information-flow-based inheritance coupling (IH-ICP) |

---

Manuscript to be reviewed

PeerJ Comput. Sci. reviewing PDF | (CS-2020:10:54207:2:1:NEW 23 Apr 2021)
Table 5. Data usage by studies

| Author          | Year | Dataset                                                                 |
|-----------------|------|-------------------------------------------------------------------------|
| Briand et al.   | 2000 | Hypothetical video rental business                                      |
| Cartwright et al.| 2000 | Large European telecommunication industry, which consists of 32 classes and 133KLOC. |
| Emam et al.     | 2001 | Used two versions of Java application: Ver 0.5 and Ver 0.6 consisting of 69 and 42 classes. |
| Gymothy et al.  | 2005 | Source code of Mozilla with the use of Columbus framework               |
| Nachippan et al.| 2005 | Open source eclipse plug-in                                            |
| Zhou et al.     | 2006 | NASA consisting of 145 classes, 2107 methods and 40 KLOC                |
| Olague H.M et al.| 2007 | Mozilla Rhino project                                                  |
| Kannan et al.   | 2007 | Library management system consists of 1185 classes                      |
| Pai et al.      | 2007 | Public domain dataset consists of 2107 methods, 145 classes, and 43 KLOC |
| Tomaszewski et al.| 2007 | Two telecommunication project developed by Ericsson                     |
| Shatnawi et al. | 2008 | Eclipse project: Bugzilla database and Change log                       |
| Aggarwal et al. | 2009 | Student projects at University School of Information Technology         |
| Singh et al.    | 2009 | NASA consists of 145 classes, 2107 methods and 40K LOC                  |
| Cruz et al.     | 2009 | 638 classes of Mylyn software                                           |
| Burrows et al.  | 2010 | IBATIS, Health watchter, Mobile media                                   |
| Singh et al.    | 2010 | NASA consists of 145 classes, 2107 methods, and 40K LOC                 |
| Zhou et al.     | 2010 | Three releases of Eclipse, consisting of 6751, 7905, 10635 java classes and 796, 988, 1306 KLOC |
| Fokaefs et al.  | 2011 | NASA datasets                                                           |
| Malhotra et al. | 2011 | Open source software                                                   |
| Mishra et al.   | 2012 | Eclipse and Equinox datasets                                            |
| Malhotra et al. | 2012 | Apache POI                                                              |
| Heena           | 2013 | Open Source Eclipse System                                             |
| Rinkai Goyal et al. | 2014 | Eclipse, Mylyn, Equinox and PDE                                         |
| Yeresime et al. | 2014 | Apache integration framework (AIF) Ver 1.6                             |
| Etgi Erulk et al.| 2015 | Promise software engineering repository                                 |
| Golnoush Alaaei et al.| 2015 | NASA datasets                                                           |
| Saja Alem et al.| 2015 | PROMISE data repository                                                 |
| Santosh et al.  | 2015 | 10 Datasets from PROMISE repository                                     |
| Yohannense et al.| 2016 | AEEEM Datasets & Four datasets PROMISE repository                        |
| Ankit Paal et al.| 2017 | Four projects from the NASA repository                                  |
| Bartlomeij et al.| 2018 | Github Projects: Flask, Odoo, GitPython, Ansible, Grab                  |
| Patil et al.    | 2018 | Real-time data set, Attitude Survey Data                                |
| Bahman et al.   | 2018 | Five NASA datasets                                                      |
| Hiba Alghaleri et al.| 2019 | 12-NASA MDP and 12-Java open-source projects                           |
| Balogun et al.  | 2019 | NASA and PROMISE repositories                                           |
| Alsaedi et al.  | 2019 | 10 NASA datasets                                                        |
| Wasiur Rhaman et al.| 2020 | GIT repository, Android-4 & 5 versions                                  |
| Deepak et al.   | 2020 | Open source bug metrics dataset                                         |
| Razu et al.     | 2020 | 3 open source datasets from PROMISE                                     |

The metrics collection of C&K crafted and applied by Chidamber et al. are the ultimate frequently utilize metrics set for software related to object oriented (Chidamber and Kemerer, 1994). Basili et al. (Briand et al., 2000) have analyzed the collection of object oriented design metrics suggested by Chidamber et al. (Basili et al., 1996). R. Subramanyam validated that \{dit\}, \{cbo\} and \{wmc\} metrics are fault predictor at class level (Subramanyam and Krishnan, 2003).

Experimental evaluations of the classification algorithm have been built for fault prediction through researches (Kaur and Kaur, 2018). Basili et al. revealed that many C&K metrics are observed to be associated with failure propensity (Basili et al., 1996). Tang et al. assessed C&K metrics suite and discovered that any of these metrics except \{rfc\} and \{wmc\} were deemed vital (Tang et al., 1999). Briand et al. carryout forty-nine metrics to ascertain, which model to apply for the prediction of faults. Conclusions shows apart from \{noc\} all metrics are useful to predict faults tendency (Briand et al., 2000). Wust and Briand determined that \{dit\} metrics are inversely correlated to fault proneness and
Table 6. Frequently used Metrics in Software Fault Prediction

| Method Level Metrics | Class Level Metrics | File Level Metrics |
|----------------------|---------------------|-------------------|
| 1. loc McCabe’s line count of code | 1. Coupling between Objects (CBO) | 1. # of times the source file was inspected prior to system test release. |
| 2. v(g) McCabe “cyclomatic complexity” | 2. Depth of Inheritance Tree (DIT) | 2. # of LOC for the source file prior to coding phase (auto-generated code) |
| 3. ev(g) McCabe “essential complexity” | 3. Lack of Cohesion of Methods (LCOM) | 3. # of LOC for the source file prior to system test release. |
| 4. lv(g) McCabe “design complexity” | 4. Number of Children (NOC) | 4. # of lines of commented code for the source file prior to code (auto-generated) |
| 5. n Halstead total operators + operands | 5. Response for Class (RFC) | 5. # of lines of commented code for the source file prior to system test release. |
| 6. v Halstead “volume” | 6. Weighted Method Per Class (WMC) | |
| 7. l Halstead “program length” | Percent_Pub_Data | |
| 8. d Halstead “difficulty” | Access_To_Pub_Data | |
| 9. i Halstead “intelligence” | Dep_On_Child | |
| 10. e Halstead “effort” | Fan_In is the # of calls by higher modules. | |
| 11. b Halstead “bug” | No_of_Method | |
| 12. t Halstead’s time estimator | No_Of_Attribute | |
| 13. IOCode Halstead’s line count | No_of_Attribute_Inher | |
| 14. IOComment Halstead’s count of lines of comments | No_of_Method_Inher | |
| 15. IOBlank Halstead’s count of blank lines | Fan_in | |
| 16. IOCodeAndComment Lines of comment and code | Fan_out | |
| 17. uniq_op Halstead Unique operators | No_of_Pvt_Method | |
| 18. uniq_opnd Halstead Unique operands | No_of_Pvt_Attribute | |
| 19. total_op Halstead Total operators | No_Pub_Method | |
| 20. total_opnd Halstead Total operands | No_Pub_Attribute | |
| 21. branchCount Branch count of the flow graph | NLOC | |

\(\text{(n}_{\text{oc}}\text{)}\) metrics is an insignificant predictor of fault tendency (Briand et al., 2001). Yu et al. selected eight metrics to explored the relationship amongst these metrics and the tendency to identify faults. So firstly they explored the correlation among metrics and found four closely associated sets. After this, they utilize univariate analysis to observe, which set can classify faults (Yu et al., 2002). Malhotra and Jain applied logistic regression methods to examine the correlation amongst metrics of object-oriented along with faults tendency. The receiver operating characteristics (ROC) evaluation was employed. The predictive model performance was assessed through ROC (Malhotra and Jain, 2012). Yeresime et al. have investigated using linear regression, logistic regression, and artificial neural network methods for the prediction of software faults making use of C\&K metrics. The findings show the significance of the weighted method per class (\(\text{wmc}\)) metric for classification of fault (Suresh et al., 2014). The impact of inheritance metrics in SFP is authenticated in experimentation, where the artificial neural network (ANN) is utilized for model building. Recall, Accuracy, F1 measures, True negative rate (TNR), and Precision
are employed for performance measures. The conclusions reveal the acceptable impact of inheritance metrics SFP. The practitioners of software testing can strongly use inheritance metrics to predict faults in software projects. A higher value of inheritance is also detrimental, because subsequently, it may generate faults in software (Aziz et al., 2019).

The literature review shows that the \{dit\} and \{noc\} metrics, related to inheritance feature are utilized in the prediction of faults jointly inside C&K metrics suite only. Consequently, it is considered important to validate the exclusive value of inheritance metrics in the context of software fault prediction.

4 METHODOLOGY

The research approach comprises on three interconnected phases, as shown in Figure 1, which includes selection, preprocessing, and experimentation/evaluation phases. The first phase comprises on the choice of data sets of inheritance metrics and performance measure. The selection of inheritance metrics data sets is based on dual criteria, the data set should be publicly available and metrics correlation should not be \( \geq 0.7 \) OR \( \leq -0.7 \).

Figure 1. Research Methodology

In the second pre-processing phase, other metrics in addition to the inheritance for example \(\text{loc} \), \(\text{cbo} \), \(\text{wmc} \) are removed to keep the data set consistent. These data sets are then split by all possible combinations of their features sets. After this, the data sets are cleaned, filtered, and remove the related anomalies. Finally, in the experiment/evaluation phase, the final form of the data set is used for the experiment, in which SVM (Support Vector Machine) is built and cross-validated. The calculation of cross entropy losses, Accuracy, F-Measure and AUC is performed. Accordingly, a score is calculated for each selected inheritance metric to determine the superior.

4.1 Selection Phase

1) Selection of inheritance metrics: from the inheritance metrics mentioned in Section 2 of this paper, we choose only those metrics that meet the criteria stated as follows.

Data set must be publicly available This condition is comprehended since the software projects fault information is extremely rarely approachable. The fundamental issue is that the information of fault for large enterprise projects is stored digitally and propriety. The bug information on small projects is too less but available for the public. Thus, labeled data is infrequently accessible. The accessibility of the data set, which is publicly available will permit the assessment of the metrics related to inheritance in fault prediction. Lastly, 40 data sets were discovered with inheritance metrics (Jureczko and Madeyski, 2010) (Menzies and Di Stefano, 2004) (Menzies et al., 2004) (a51, 2009) (Niu and Mahmoud, 2012) (D’Ambros et al., 2010) (Wagner, 2010) (Abdelnoe et al., 2006) (Abdelnoe et al., 2005) (Monarchi and Puhr, 1992) (Shepperd et al., 2013). A total of nine inheritance metrics are found in this data set; which are Inheritance Tree Depth (dit), Number of Children (noc), Functional Abstraction Measure (mfa), Inheritance Coupling (ic), Number of Method Inherited (nomi), Inherited Attribute Number (noai), Dependent on Child (doc), Number of methods called per class (fanOut) and Number of classes that call class methods (fanIn).

A total of 40 data sets, out of which about 35 data sets are found on the servers of tera-PROMISE repository (Boetticher, 2007) and five data sets are located in the D’Ambros repository (D’Ambros et al., 2010). In this regard, Table 7 depicts the detailed information of these data sets. The first column indicates the data set name along with the version number if exist. Second column shows the detail about the total number of records and third column shows the percentage of fault for each base data sets. Overall, nine distinct metrics of inheritance are discovered in 40 data sets, where \(\checkmark\) is used to label the presence of a metric in the associated data set and \(\times\) is labeled where the metrics are not present in the data set.
Unluckily, all nine inheritance metrics do not exist in a single data set. However, set of inheritance metrics comprising \{dit, noc, ic, mfa\} are found in 30 data sets, \{dit, noc, nomi, noai\} in 2 data sets, \{dit, fanin, fanout, noai, noc, nomi\} in 7 data sets and \{dit, noc, ic, doc, fanin, fanout\} in 1 data set.

Overall \{dit\} and \{noc\} features exist in all forty data sets. The feature \{ic\} is found in 31 data sets, feature \{mfa\} in 30 data sets and similarly other in multiple data sets through these 659 data sets created for the experiment that will be explained in subsequent sections. The conclusion drawn is based upon the predictive ability of \{ic\} on 31 data sets. Same goes for rest of the feature sets.

It is prominent to mentioned here that some of the data sets are already utilized in an experiment to comparing inheritance metrics with C&K metrics (Aziz et al., 2019).

**Correlation should not be $\geq 0.7$ OR $\leq -0.7$.** Software metrics have a tendency of correlation as these focus on the related characteristic of object-oriented programming for example in our case inheritance. A high value of correlation as $\geq 0.7$ or $\leq -0.7$ is a category of repetition that needs that repetition metric should be eliminated. The issue is the effect of managing the repetition metric might be negative, triggering uncertainty for the mining algorithm and determine a depleted pattern of quality (Han et al., 2011). Furthermore, the advantages of removing correlated metrics are significantly better as compare to the cost (Jiarppakdee et al., 2018). In the event of a lesser value of correlation, almost near to $\geq 0.7$ or $\leq -0.7$, the rejection of a metric may deprive the data set of significant important information.

In the study, in the case metrics illustrated in the second criterion, we execute the Pearson $r$ and the Spearman $p$ correlation coefficient for the pairs discovered in forty selected public data sets. Correlation between the features of each data set is computed in unfiltered data set. In order to explain the feature to feature correlation analysis further data set ant-1.7 has four features \{dit\}, \{noc\}, \{ic\}, and \{mfa\}.

### Table 7. Source Data-sets

| Dataset Name                  | # Ins | % Faulty | \{dit\} | \{noc\} | \{ic\} | \{mfa\} |
|-------------------------------|-------|----------|---------|---------|--------|---------|
| ant-1.7                       | 745   | 22       | ✓       | ✓       | ✓      | ✓       |
| Arc                           | 234   | 31       | ✓       | ✓       | ✓      | ✓       |
|.eclipse JDT Core              | 997   | 21       | ✓       | ✓       | ✓      | ✓       |
| Eclipse PDE UI                | 1497  | 14       | ✓       | ✓       | ✓      | ✓       |
| eclipse34_debug               | 1065  | 25       | ✓       | ✓       | ✓      | ✓       |
| eclipse34_xwi                 | 1485  | 44       | ✓       | ✓       | ✓      | ✓       |
| e-learning                    | 64    | 9        | ✓       | ✓       | ✓      | ✓       |
| Equinox Framework             | 324   | 40       | ✓       | ✓       | ✓      | ✓       |
| forest-0.6                    | 7     | 14       | ✓       | ✓       | ✓      | ✓       |
| my-1.1                        | 111   | 57       | ✓       | ✓       | ✓      | ✓       |
| gedit-3.2                     | 272   | 33       | ✓       | ✓       | ✓      | ✓       |
| Kalkulator                    | 27    | 22       | ✓       | ✓       | ✓      | ✓       |
| Kc1-class-binary             | 145   | 41       | ✓       | ✓       | ✓      | ✓       |
| singly-1.0                    | 135   | 25       | ✓       | ✓       | ✓      | ✓       |
| Lucene                        | 691   | 9        | ✓       | ✓       | ✓      | ✓       |
| mylyn                         | 1862  | 13       | ✓       | ✓       | ✓      | ✓       |
| nisusomosci                  | 27    | 37       | ✓       | ✓       | ✓      | ✓       |
| pdftranslator                 | 33    | 45       | ✓       | ✓       | ✓      | ✓       |
| poi-1.5                      | 1237  | 59       | ✓       | ✓       | ✓      | ✓       |
| prop-1                        | 18471 | 15       | ✓       | ✓       | ✓      | ✓       |
| redaktor                      | 176   | 15       | ✓       | ✓       | ✓      | ✓       |
| serapion                     | 45    | 20       | ✓       | ✓       | ✓      | ✓       |
| single-version-cx-oo          | 997   | 20       | ✓       | ✓       | ✓      | ✓       |
| sklebagd                      | 85     | 8        | ✓       | ✓       | ✓      | ✓       |
| synapse-1.0                   | 157   | 10       | ✓       | ✓       | ✓      | ✓       |
| systemddata                   | 65    | 13       | ✓       | ✓       | ✓      | ✓       |
| szybkafucha                   | 25    | 56       | ✓       | ✓       | ✓      | ✓       |
| tempoproject                  | 92    | 30       | ✓       | ✓       | ✓      | ✓       |
| tspci                        | 351   | 8        | ✓       | ✓       | ✓      | ✓       |
| velocity-1.4                  | 196   | 75       | ✓       | ✓       | ✓      | ✓       |
| workflow                      | 39    | 51       | ✓       | ✓       | ✓      | ✓       |
| wspomaganiepi                 | 18    | 67       | ✓       | ✓       | ✓      | ✓       |
| xalan-2.4                     | 723   | 15       | ✓       | ✓       | ✓      | ✓       |
| xerces-util                   | 162   | 48       | ✓       | ✓       | ✓      | ✓       |
| zuzel                         | 29    | 44       | ✓       | ✓       | ✓      | ✓       |
as shown in Table 7. First the correlation of \{dit\} with other available features \{noc\}, \{ic\}, and \{mfa\} is computed. After this, the correlation of second feature of ant-1.7 data set \{noc\} with other features \{ic\} and \{mfa\} is computed. Finally \{ic\} with \{mfa\} is computed. The same process is separately applied between the features of all selected 40 data set shown in Table 7.

The strongly correlated features are dropped in their corresponding data sets only. It is important to mention here that correlation is only a precautionary step otherwise it does not manipulate feature set.

The presence of strongly correlated features in the SVM models make it difficult to converge and justify the generality of the results. Therefore, strongly correlated features are identified and dropped.

Pearson correlation coefficient is used for the purpose.

Nevertheless, all combinations are positively correlated, where not a single pair is equal to $\geq 0.7$ or $\leq -0.7$. The nine inheritance metrics also meet the second criterion.

Tools/programming language and environments used to compute Pearson $r$ and the Spearman $p$ correlation coefficient is R platform with ggplot2 R package.

2) Selection of Performance Measure. The models of machine learning built using classification are measured with their accomplishment by categorizing the unidentified occurrences. A confusion matrix is a method of showing its capability. Catal et al. have computed many performance metrics, derived and originated through confusion matrix (Catal, 2012). Malhotra also has suggested the overall explanation of many assessment measures used in software fault prediction. The finding revealed that True Positive Rate(TPR) is the frequently utilized performance measure in software fault prediction, succeeding measures are Precision and AUC (Malhotra, 2015).

Cross entropy is a mean to compute the overall deviation of the models’ probability from the actual label. Independent of the threshold is a key property of cross entropy (Hinojosa et al., 2018). It is effective in both training and testing phases (Golik et al., 2013) (Kline and Berardi, 2005). So cross entropy is choose as a primary performance measure and Accuracy, F-Measure and AUC are selected as a supporting measure for this experiment.

4.2 Preprocessing Phase

1) Remove non-inheritance metrics. Collected data sets have numerous metrics other than inheritance metrics, including \{loc\}, \{wmc\}, \{ca\}, \{cbo\}, and several others. Because we are planning to assess the inheritance metrics in the background of SFP, so all non-inheritance metrics are removed. This might affect the performance but it will be better to evaluate the inheritance metrics viability on software fault prediction.

2) Uniformity of Labels. Every metrics encompass continuous numeric values with the associated data set however discrepancies are located in the class namely [bug]. These are settled through the guidelines mentioned as under:

$$\text{bug} = \begin{cases} \text{False} & \text{defects} = \text{No, False, 0, N, No} \\ \text{True} & \text{Otherwise} \end{cases}$$  \quad (5)$$

Within the guidelines, [False] is employed when there are no faults and [True] represents faulty instances.

3) Splitting. The main objective of this study is to measure the significance of nine selected metrics of inheritance so forty data sets are separated into numerous possible sets of features after removing non-inheritance metrics.

The objectives of splitting process are to visualize the impact of every possible feature set and determine the most significant feature set out of the available data sets.

Each data set out of these forty data sets are separated into numerous possible sets of features by splitting and combining features into all possible unique combinations. In order to explain further, data set ant-1.7 has four features \{dit\}, \{noc\}, \{ic\}, and \{mfa\} as shown in Table 7. Splitting this single data set into all possible unique combinations will create about 15 sub-data sets as explained by the formula $2^{(\text{number of features})} - 1$ where $2^{4} - 1 = 15$ unique sub-data sets.

Resultantly 659 sub-data sets are formed by applying the same process on all forty publicly acquired data sets. Overall 67 unique features has been generated from these sub-data sets. First column of Table 8 shows these 67 unique features under the heading "features", 2nd column depicts the number of metrics combined to generate a unique feature and third column shows total number of sub-data sets formed by applying splitting process on to forty data sets that generated overall 659 sub-data sets.
Finally, 659 sub-data sets were formed, which contains 67 different features sets (column 3 of Table 8). Afterwards, all 659 sub data sets have been passed through three phases; dropping same instances, dropping inconsistent instances, and filtration.

4) Cleaning. The next step is cleaning where identical instances in the data sets are eliminated since these instances are worthless and, sometimes, confusing for the model. Afterwards, inconsistent instances are also removed, since these are inconsistency in data sets (Henderson-Sellers, 1995). In inconsistency, the occurrences of all the metrics contain same values and contain dissimilar class tags.

Our objective is only to identify the anomaly on the data sets. This problem may possibly be addressed in four different ways. First option is to drop both the instances, consequently, information will be lost. Second option is to drop instances of minor class, due to this, data set will become more skewed. Third option is to drop instances of major class that resultantly produces less skewness, and in fourth option keep both instances resultantly reciprocities negates the effect of each other. In this study, third option is applied to keep the impact minimum on the data sets.

5) Filtration. Small data sets have been dropped where the number of instances are \( \leq 100 \) in the filtration phase. This filter is applied for the reason to employ ten-fold cross validation, not including replacement, that is typically the situation with validation of the model. Consequently, 293 data sets are eliminated while using this filter.

6) Skewness \( \leq 9:1 \). The objective is to identify the skewed data sets out of 659 sub-data sets and drop them. Skewness has not been addressed further in this study.

Skewness shows faulty or free of fault occurrences that must contain a percentage of data sets to \( \leq 90 \) and \( \geq 10 \). The skewness filter is applied in the case where only 100 instances in the data set, then a minimum one record from both groups exists in the case where no hierarchical 10-fold cross validation that is replaced. After applying this filter onto all 659 sub-data sets only one sub-data set found skewed, which is hence eliminated.

Though remaining data sets are imbalance yet it is addressed in two folds, model building algorithm selection and performance measure selection. In first case, we use SVM which is usually the choice of modeling for imbalance data set (Xing et al., 2005; Elish and Elish, 2008b; Singh et al., 2009; Di Martino et al., 2011; Yu, 2012; Malhotra et al., 2010). In the case of performance evaluation we use cross entropy which is, again independent of imbalance in the data set.

Lastly, after applying cleaning, filtering, and skewness, 659 data sets were decreased to 365, which are shown in the last columns of Table 8.

5 EXPERIMENT AND RESULTS

5.1 Experiment Setup

Dataset: 365 preprocessed data sets, as depicted in Table 8.

Tools: R Language version 3.4.3 (Rajnish and Bhattacherjee, 2005) in R Studio 1.1.383 (Rajnish and Bhattacherjee, 2006b).

Data Splitting technique: Ten-fold stratified cross-validation without replacement. The stratified splitting maintains the ratio of classes in all the folds. Moreover, we reported the average results attained in the 10 folds

Classifiers Algorithm: SVM is generally appreciated by the SFP community (Xing et al., 2005; Elish and Elish, 2008b; Singh et al., 2009; Di Martino et al., 2011; Yu, 2012), for its applicability on real-world applications, non-linear data coverage, and well generalization in high dimensional space. SVM is utilized for model building. Stratified splitting without replacement is done for ten-fold cross validation. Finally, we reported the average results computed in the 10-folds splitting in the form Cross entropy loss, Accuracy, F-Measure and AUC for all data sets.

SVM parameters: Gaussian kernel: The kernel has following equation:

\[
k(x, y) = \exp\left(-\frac{||x - y||}{2\sigma^2}\right)
\]

(6)

It is a general-purpose kernel. It does not require any specific pattern of data. Moreover, SVM has been built and validated Gaussian kernel functions of the kernel and the best model. Complete working of SVM model building and result collection is shown in Algorithm 1.

The focus of this article is the viability of inheritance metrics in software fault prediction. It is found that the conventional techniques nevertheless being quite old, are still being used by the SFP community.
### Table 8. Filtered Data

| Features       | # F | Total Datasets | Small Datasets | skewed | Remaining |
|----------------|-----|----------------|----------------|--------|-----------|
| mit            | 1   | 28             | 25             | 0      | 3         |
| noe            | 1   | 9              | 3              | 0      | 6         |
| nomi           | 1   | 9              | 3              | 0      | 3         |
| dit.fanIn      | 2   | 9              | 5              | 0      | 4         |
| dit.fanOut     | 2   | 7              | 3              | 0      | 4         |
| dit.mfa        | 2   | 28             | 25             | 0      | 3         |
| dit.noai       | 2   | 9              | 4              | 0      | 5         |
| dit.noc        | 2   | 40             | 38             | 0      | 2         |
| dit.nomi       | 2   | 9              | 3              | 0      | 6         |
| fanIn.fanOut   | 2   | 6              | 0              | 0      | 6         |
| fanIn.noai     | 2   | 7              | 1              | 0      | 6         |
| fanIn.noc      | 2   | 8              | 4              | 0      | 4         |
| fanIn.nomi     | 2   | 6              | 0              | 0      | 6         |
| fanOut.noai    | 2   | 7              | 2              | 0      | 5         |
| fanOut.noc     | 2   | 7              | 3              | 0      | 4         |
| fanOut.nomi    | 2   | 6              | 0              | 0      | 6         |
| ic.mfa         | 2   | 27             | 24             | 0      | 3         |
| noe            | 2   | 9              | 1              | 0      | 8         |
| noe.mfa        | 2   | 28             | 25             | 0      | 3         |
| noe.noai       | 2   | 9              | 4              | 0      | 5         |
| noe.noci       | 2   | 9              | 2              | 0      | 7         |
| dit.fanIn.fanOut| 3 | 6              | 0              | 0      | 6         |
| dit.fanIn.noai| 3   | 7              | 1              | 0      | 6         |
| dit.fanIn.noc  | 3   | 8              | 3              | 0      | 5         |
| dit.fanIn.nomi | 3   | 6              | 0              | 0      | 6         |
| dit.fanOut.noai| 3 | 7              | 1              | 0      | 6         |
| dit.fanOut.noc | 3   | 7              | 2              | 0      | 5         |
| dit.fanOut.nomi| 3   | 6              | 0              | 0      | 6         |
| dit.ic.mfa     | 3   | 28             | 24             | 0      | 4         |
| dit.noai.nomi  | 3   | 9              | 1              | 0      | 8         |
| dit.mfa.noc    | 3   | 27             | 24             | 0      | 3         |
| dit.noc.noai   | 3   | 9              | 2              | 0      | 7         |
| dit.noc.nomi   | 3   | 9              | 1              | 0      | 8         |
| fanIn.fanOut.noai| 3 | 6              | 0              | 0      | 6         |
| fanIn.fanOut.noc| 3 | 6              | 0              | 0      | 6         |
| fanIn.fanOut.nomi| 3 | 6              | 0              | 0      | 6         |
| fanIn.noai.nomi| 3   | 7              | 1              | 0      | 6         |
| fanIn.noc.nomi | 3   | 6              | 0              | 0      | 6         |
| fanOut.noai.nomi| 3 | 6              | 0              | 0      | 6         |
| fanOut.noc.nai | 3   | 7              | 1              | 0      | 6         |
| fanOut.noc.nomi| 3   | 6              | 0              | 0      | 6         |
| ic.mfa.noc     | 3   | 27             | 24             | 0      | 3         |
| noe            | 3   | 9              | 1              | 0      | 8         |
| noe.mfa        | 3   | 28             | 25             | 0      | 3         |
| noe.noai       | 3   | 9              | 4              | 0      | 5         |
| doe            | 3   | 9              | 2              | 0      | 7         |
| dit.fanIn.fanOut| 4 | 6              | 0              | 0      | 6         |
| dit.fanIn.noc  | 4   | 6              | 0              | 0      | 6         |
| dit.fanIn.nomi | 4   | 6              | 0              | 0      | 6         |
| dit.fanOut.noc | 4   | 6              | 0              | 0      | 6         |
| dit.fanOut.nomi| 4   | 6              | 0              | 0      | 6         |
| dit.ic.mfa     | 4   | 27             | 24             | 0      | 3         |
| dit.mfa.noc    | 4   | 6              | 0              | 0      | 6         |
| dit.noai.nomi  | 4   | 6              | 0              | 0      | 6         |
| dit.mfa.noc    | 4   | 6              | 0              | 0      | 6         |
| dit.noai.noai  | 4   | 6              | 0              | 0      | 6         |
| dit.fanIn.fanOut.noai| 4 | 6              | 0              | 0      | 6         |
| dit.fanIn.fanOut.noc| 4 | 6              | 0              | 0      | 6         |
| dit.fanIn.fanOut.nomi| 4 | 6              | 0              | 0      | 6         |
| dit.fanIn.noai.nomi| 4 | 6              | 0              | 0      | 6         |
| dit.fanIn.noc.nomi| 4   | 6              | 0              | 0      | 6         |
| dit.fanOut.noc.nomi| 5 | 6              | 0              | 0      | 6         |
| dit.fanOut.noc.noai| 5 | 6              | 0              | 0      | 6         |
| dit.fanOut.noc.nomi| 5   | 6              | 0              | 0      | 6         |
| dit.ic.mfa.noc | 5   | 6              | 0              | 0      | 6         |
| dit.mfa.noc    | 5   | 6              | 0              | 0      | 6         |
| dit.noai.nomi  | 5   | 6              | 0              | 0      | 6         |
| dit.mfa.noc    | 5   | 6              | 0              | 0      | 6         |
| dit.noai.noai  | 6   | 6              | 0              | 0      | 6         |
| dit.fanIn.fanOut.noai| 6 | 6              | 0              | 0      | 6         |
| dit.fanIn.fanOut.noc| 6   | 6              | 0              | 0      | 6         |
| dit.fanIn.fanOut.nomi| 6 | 6              | 0              | 0      | 6         |

Total 659 293 1 365

The obvious reason is the outperformance of the discussed algorithms on the data set used. We didn’t use the advanced machine learning techniques, explicitly. The reason is that they are accumulated using the
The fundamental objective of this work is to assess the exclusive viability of inheritance metrics in SFP, whereas the secondary aim is to achieve the greatest outcomes from the algorithms of machine learning. Keeping these in view, filtration is applied to data sets, and experiments are planned.

The experiment was conducted while keeping the industrial objective in mind where model is deployed with single threshold. Although model is supposed to be checked at every threshold but needs to be deployed with the threshold that gives the best results. So performance measure that visualize the performance of the model on that very threshold is of our interest, exactly where error entropy sits well, whereas AUC computes the models performance at every threshold, which does not suit with the objective.

### 5.2 Results and Discussion

There are facts that inheritance metrics are significantly different from other metrics. Firstly, semantic distinction of inheritance metrics from other metrics need no comments, and justification. Secondly, data set view point, Rashid et al. (Aziz et al., 2019) proved the empirical distinction between inheritance, and non-inheritance metrics in his study. Moreover, inheritance metrics itself quite distinct from each other. This has been approved by visualizing the tendency of correlation between inheritance metrics. To illustrate this we compute the Pearson $r$, and the Spearman $p$ correlation coefficient for the pairs discovered in forty selected public data sets. Correlation between the features of each data set is computed in unfiltered data set. Nevertheless, all combinations are positively correlated, where not a single pair is equal to $\geq 0.7$ or $\leq -0.7$.

The fundamental objective of this work is to assess the exclusive viability of inheritance metrics in SFP, whereas the secondary aim is to achieve the greatest outcomes from the algorithms of machine learning. Keeping these in view, filtration is applied to data sets, and experiments are planned.

#### 5.2.1 Overall Cross Entropy Loss

In this context, Table 9 displays the outcomes of the experiments where the name of the feature is shown in the first column, the number of features in the second column, the overall number of data sets in the third column. Averages of Cross Entropy, Accuracy, F-Measure, and AUC are in column four, five, six, and seven. The minimum values of Cross Entropy, Accuracy, F-measure, and AUC are in column eight, nine, ten, and eleven respectively. Cross-entropy loss, Accuracy, F-Measure, and AUC are computed for all 365 data sets comprising of 67 unique combinations. These unique combinations are plotted in Figure 2 where the number of features are increasing from bottom to top. The upper part depicts the least mean entropy loss, average in the middle, and maximum at the bottom. This graph proves our objective that adding inheritance metrics will reduce the entropy loss. The findings are also validated by Accuracy, F-Measure, and AUC. Figure 2 graphically compares the cross entropy loss calculated through SVM on 1, 2, 3, 4, 5, and 6 feature sets of inheritance metrics on 365 data sets comprising 67 unique features. The lower part of Figure 2 is heavy since the features are less. The results are gradually decreasing while moving upward since feature sets are increasing from 2 to 6. This graph proves our objective that adding inheritance metrics will reduce the entropy loss. The findings are also validated by Accuracy, F-Measure, and AUC.

---

**Algorithm 1: SVM Model Building and Cross Validation**

```plaintext
function SVMBuild (365 data sets):
    Input : 365 data sets
    Output: Cross entropy, Accuracy, F-Measure and AUC of the model in every data set

for cur – Dset ← 1 to 365 do
    curr-Dset ← zScoreScale(cur-Dset)
    stratifiedSplit(curr-Dset, j ← 10)
    for j ← 1 to 10 do
        train-set ← curr-Dset[j-1]
        test-set ← curr-Dset[j]
        four-Models ← \emptyset
        kernel ← GaussianRadial
        SVMModel ← trainSVM(train-set, kernel)
        PerformanceMeasures ← CrossEntropy, Acc, F-Measure and AUC (SVMModel)
    end
    all-Dataset-PerformanceMeasures ← \{all-Dataset-PerformanceMeasures\} ∪ \{PerformanceMeasures\}
end
return all-Dataset-PerformanceMeasures
```
Figure 2. Feature wise cross entry loss of all data sets
Table 9. Evaluation Parameters for all features sets

| Features | # F | # Sets | Cross Entropy | Accuracy | AUC | Minimum | Cross Entropy | Accuracy | AUC | Minimum |
|-----------|-----|--------|---------------|----------|-----|---------|---------------|----------|-----|---------|
| dit, ic    | 1   | 1      | 0.00094       | 0.3198   | 0.141 | 0.9999  | 0.2696        | 0.141   | 0.9999 | 0.2696 |
| fanOut, noc | 2   | 2      | 0.00134       | 0.3215   | 0.141 | 0.9999  | 0.2696        | 0.141   | 0.9999 | 0.2696 |
| dit, ic    | 3   | 3      | 0.00189       | 0.3215   | 0.141 | 0.9999  | 0.2696        | 0.141   | 0.9999 | 0.2696 |
| noc, mfa   | 4   | 4      | 0.00254       | 0.3215   | 0.141 | 0.9999  | 0.2696        | 0.141   | 0.9999 | 0.2696 |
| fanOut, noc | 5   | 5      | 0.00319       | 0.3215   | 0.141 | 0.9999  | 0.2696        | 0.141   | 0.9999 | 0.2696 |
| dit, ic    | 6   | 6      | 0.00384       | 0.3215   | 0.141 | 0.9999  | 0.2696        | 0.141   | 0.9999 | 0.2696 |
| noc, mfa   | 7   | 7      | 0.00449       | 0.3215   | 0.141 | 0.9999  | 0.2696        | 0.141   | 0.9999 | 0.2696 |
| fanOut, noc | 8   | 8      | 0.00514       | 0.3215   | 0.141 | 0.9999  | 0.2696        | 0.141   | 0.9999 | 0.2696 |

Table 9 shows that overall (dit, ic, noc, mfa) achieved least entropy rate of 0.000723, and (fanIn, fanOut, noc, noai, nomi) achieved average least entropy rate of 0.001707.

Figure 3 shows the absence of outliers in the results across all performance measures. It can therefore be safely stated that the averages of performance measures are not biased.

### 5.2.2 Feature wise Cross Entropy Loss

Regarding the exclusive assessment of inheritance metrics in the context of SFP, which is the main goal of this article, adding inheritance metrics will reduce the Cross Entropy Loss. In this regard feature wise Cross Entropy Loss, Accuracy, F-Measure, and AUC are extracted from Table 9. The first column of
Table 10 shows the feature set, number of features from 1 to 6 in the second column, and least Cross Entropy Loss in the third column. In order to support the findings Accuracy, F-measure, and AUC in column four, five, and six respectively. The overall findings are:

1. The results shown in Table 10 contain two distant feature sets, feature number 1 to 4, and 5 to 6. The first set comprises of \{mfa, ic, noc, dit\}, and second set comprises of \{dit, fanIn, fanOut, noc, noai, nomi\}.

2. Considering the single inheritance feature, \(\text{mfa}\) achieved the highest entropy rate of 0.0024225. After inserting another inheritance feature \(\text{ic}\) with \(\text{mfa}\) the error rate is further reduced to 0.0011558. Similarly adding \(\text{noc}\) with \(\text{ic, mfa}\) the entropy rate further reduced to 0.0008679. The rate further reduced to 0.0007233 by adding \(\text{dit}\) into the existing combination of \{mfa, ic, noc\}.

3. Considering feature set of five inheritance metrics \{fanIn, fanOut, noc, noai, nomi\} the Cross Entropy error rate is 0.0011961. When adding \(\text{dit}\) the rate is dropped to 0.0011813.

Table 10. Feature wise Minimum Rate

| Feature set          | # F | Cross Entropy | Accuracy | F-Measure | AUC  |
|----------------------|-----|---------------|----------|-----------|------|
| mfa                  | 1   | 0.0024225     | 0.1176500| 0.2105300| 0.4442000|
| ic, mfa              | 2   | 0.0011558     | 0.1162800| 0.2083300| 0.4311300|
| ic, mfa, noc         | 3   | 0.0008679     | 0.1250000| 0.2222000| 0.4165100|
| dit, ic, mfa, noc    | 4   | 0.0007233     | 0.1200000| 0.0434800| 0.5000000|
| fanIn, fanOut, noc, noai, nomi | 5   | 0.0011961     | 0.1100000| 0.1982000| 0.3809100|
| dit, fanIn, fanOut, noc, noai, nomi | 6   | 0.0011813     | 0.1089100| 0.1964300| 0.6171000|

The graphical representation of Table 10 is depicted in Figure 4, which shows the results of two distant feature sets. The first set comprises of \{mfa, ic, noc, dit\}, and second set comprises of \{dit, fanIn, fanOut, noc, noai, nomi\}. Adding inheritance metrics into \{mfa\} the Cross Entropy rate is reduced significantly from 1 to 4 features sets. Similarly adding \{dit\} into the 5th set will further reduce the Entropy rate. The findings are also supported by the results of other performance measure Accuracy, F-measure, and AUC in Table 10.
Regarding the overall exclusive assessment of inheritance metrics on to 365 data sets comprising 67 unique features, the average is calculated for all unique features to make an overall assessment. Table 11 shows the results where the first column contains a feature set, number of inheritance features in the 2nd column, and average score for Cross Entropy Loss in 3rd column. In order to support the findings Accuracy, F-measure, and AUC in column four, five, and six respectively. The overall findings are:

1. The results are shown in Table 11 also contain two distant feature sets, feature number 1 to 2, and 3 to 6. The first set comprises on \{mfa, ic\}, and second set comprises on \{dit, fanIn, fanOut, noc, noai, nomi\}.

2. Considering the single inheritance feature, \{mfa\} achieved the highest average of entropy rate of 0.0073134. After inserting another inheritance feature \{ic\} with \{mfa\} the average is reduced to 0.0031541.

3. Considering feature set of three inheritance metrics \{fanIn, fanOut, nomi\} the average entropy rate is 0.0020645. Adding \{noai\} the average entropy rate is reduced to 0.0017731. Similarly adding \{noc\} the average entropy rate is reduced to 0.0017595, and finally adding \{dit\} the rate further reduced to 0.0017428.

Graphical representation of Table 11 is depicted in Figure 5, which shows the results of two distant feature sets. The first set comprises of \{mfa, ic\}, and the second set comprises of \{dit, fanIn, fanOut, noc, noai, nomi\}. Adding inheritance metrics into \{mfa\}, the entropy rate is reduced significantly from 1 to 2 features sets. Similarly adding \{noai, noc\}, and \{dit\} into \{fanIn, fanOut, nomi\} will further reduce the error rate. The findings are also validated by the results of other performance measure Accuracy, F-measure, and AUC in Table 11.
5.2.4 Cardinality of Features

The cardinality of the feature set is 6 in this paper. Figure 6 shows the average Cross Entropy rate of a single feature set, double feature set, and up to 6 feature set. This clearly depicted, adding inheritance metrics will gradually reduce the entropy rate. The average entropy rate is 83.3 when single inheritance metrics is used. The average rate is reduced to 63.1 when two inheritance metrics are used. The average rate is further reduced to 39.8 when three inheritance metrics are used. The average rate is gradually reduced further to 26.3, 20.3, and 17.4 when four, five, and six inheritance metrics are used respectively.

Figure 5. Feature wise average

Figure 6. Entropy error computed across the varying feature sets’ cardinality

Averages of minimum, and average performance measures for single, double, triple, and up to six feature set are calculated from Table 9, and depicts in Table 12. The first column shows the Cardinality of Feature set, and column two, three, four, and five contains averages of Cross entropy, Accuracy, F-Measure, and AUC. The average of minimum values of Cross Entropy, Accuracy, F-Measure, AUC in column six, seven eight, and nine. The graphical representation of Cross Entropy Loss is shown in Figure 6 which proves our objects that adding inheritance matrices will reduce the cross entropy rate. Similarly, the results of Accuracy, F-Measure, and AUC depicts in Table 12 also endorse these findings.

Furthermore, we conducted an experimental study to see how significantly metrics of inheritance assist
in the prediction of software faults. Thus we made compassion of inheritance metrics, and Chidamber and Kemerer (C&K) metrics suite. The findings demonstrate an appropriate impact of metrics of inheritance in software fault prediction (Aziz et al., 2019).

6 THREATS, CONCLUSION AND FUTURE WORK

6.1 Threats To Validity

Our study relies on the data sets obtained from the repositories of tera-PROMISE, NASA, and D’Ambros. In these repositories, insufficient information is available regarding the faults nor it indicates any certain type of software fault.

Primarily, faults are not indicated for the specific category of software fault. Therefore the projection might not be generalized for all categories of faults associated with the software. Likewise selected data set to encompass limited software products by diversification in team, design, scope, etc. The circumstance of fault might not be due to the inheritance aspect only.

Since the information associated with the projects are not available in the selected data sets, therefore, the identification of most associated factor to faults cannot be determined. Though the experiments do advocate the predictive ability of inheritance metrics yet the causation relationship may not be guaranteed.

We do not claim the general representation of the results across the algorithm. Yet the applicability of the result may conclude the mapping of independent variable over the dependent variable.

We selected SVM as a modeling algorithm, for being its general acceptance by the SFP community. Since the training process is heavily dependent on the data set available thus results may vary by varying the modeling algorithm. Likewise, results may also differ by varying the kernel function of SVM.

Lastly, selected metrics of inheritance are not covering every aspect related to inheritance in software products. Therefore generalization of selected metrics of inheritance might not possibly be the effect of every aspect of inheritance.

6.2 Conclusion And Future Work

In this paper, we assessed software metrics of inheritance exclusively for their viability on software fault prediction. Experiments on forty distinct data sets argue inheritance metrics viability to faults prediction. The consensus of SVM revealed that inheritance metrics may accomplish the least entropy rate with a set of common characteristics. Overall {fanIn, fanOut, noc, noai, nomi}, and individually (dit, ic, noc, mfa) proved to be the best predictor with the least entropy rate, whereas {dit}, (noc), (ic) are helpful in reduction of entropy rate. We report that adding inheritance metrics is useful for predicting faults. These findings are also validated through performance measures of Accuracy, F-Measure, and AUC.

Regarding future work, we anticipate that some scholars may rebuild our experimentation, and attempt to assess other metrics of inheritance than those we have employed. Since only nine metrics of inheritance are reviewed, and assessed in this article, and several other metrics of inheritance are defined in the literature. Due to the lack of availability of public data sets for these remaining metrics, these may not be assessed. This stimulates the requirement to build a data set that transmits data for the rest of the metrics. Also in this paper classification has been used due to less availability of continuous labels. So, the data set for the continuous value might be used for regression, to further evaluate the performance of inheritance metrics.

REFERENCES

(2009). Softlab. In ar 1-6, February.
Abdelmoez, W., Goseva-Popstojanova, K., and Ammar, H. (2006). Maintainability based risk assessment in adaptive maintenance context. In 2nd International Predictor Models in Software Engineering Workshop (PROMISE 2006), Philadelphia, PA.

Abdelmoez, W., Shereshevsky, M., Gunnalan, R., Ammar, H. H., Yu, B., Bogazzi, S., Korkmaz, M., and Mili, A. (2005). Quantifying software architectures: an analysis of change propagation probabilities. In The 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005., page 124. IEEE.

Abreu, F. B. and Carapuça, R. (1994). Object-oriented software engineering: Measuring and controlling the development process. In Proceedings of the 4th international conference on software quality, volume 186, pages 1–8.

Aziz, S. R., Khan, T. A., and Nadeem, A. (2019). Experimental validation of inheritance metrics’ impact on software fault prediction. IEEE Access.

Aziz, S. R., Khan, T. A., and Nadeem, A. (2020). Efficacy of inheritance aspect in software fault prediction: a survey paper. IEEE Access, 8:170548–170567.

Bansiya, J. and Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. IEEE Transactions on software engineering, 28(1):4–17.

Basili, V. R., Briand, L. C., and Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. IEEE Transactions on software engineering, 22(10):751–761.

Beecham, S., Hall, T., Bowes, D., Gray, D., Counsell, S., and Black, S. (2010). A systematic review of fault prediction approaches used in software engineering. The Irish Software Engineering Research Centre: Limerick, Ireland.

Belady, L. A. and Evangelisti, C. J. (1981). System partitioning and its measure. Journal of Systems and Software, 2(1):23–29.

Bilton, N. (2016). Nest thermostat glitch leaves users in the cold. The New York Times, Jan.

Boetticher, G. (2007). The promise repository of empirical software engineering data. http://promisedata.org/repository.

Breesam, K. M. (2007). Metrics for object-oriented design focusing on class inheritance metrics. In 2nd International Conference on Dependability of Computer Systems (DepCoS-RELCOMEX’07), pages 231–237. IEEE.

Briand, L. C., Wüst, J., Daly, J. W., and Porter, D. V. (2000). Exploring the relationships between design measures and software quality in object-oriented systems. Journal of systems and software, 51(3):245–273.

Briand, L. C., Wüst, J., and Lounis, H. (2001). Replicated case studies for investigating quality factors in object-oriented designs. Empirical software engineering, 6(1):11–58.

Card, D. and Glass, R. (1990). Measuring software design quality. Prentice-Hall. Eaglewood Cliffs, New Jersey, USA.

Card, D. N. and Agresti, W. W. (1988). Measuring software design complexity. Journal of Systems and Software, 8(3):185–197.

Catal, C. (2011). Software fault prediction: A literature review and current trends. Expert systems with applications, 38(4):4626–4636.

Catal, C. (2012). Performance evaluation metrics for software fault prediction studies. Acta Polytechnica Hungarica, 9(4):193–206.

Catal, C. and Diri, B. (2009). Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Information Sciences, 179(8):1040–1058.

Chappelly, T., Cifuentes, C., Krishnan, P., and Gey, S. (2017). Machine learning for finding bugs: An initial report. In 2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE), pages 21–26. IEEE.

Chen, J. and Lu, J. (1993). A new metric for object-oriented design. Information and software technology, 35(4):232–240.

Chen, J.-C. and Huang, S.-J. (2009). An empirical analysis of the impact of software development problem factors on software maintainability. Journal of Systems and Software, 82(6):981–992.

Chidamber, S. R. and Kemerer, C. F. (1991). Towards a metrics suite for object oriented design.

Chidamber, S. R. and Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on software engineering, 20(6):476–493.

Dale, C. and Van Der Zee, H. (1992). Software productivity metrics: who needs them? Information and...
716 Software Technology, 34(11):731–738.
717 Daly, J., Brooks, A., Miller, J., Roper, M., and Wood, M. (1996). Evaluating inheritance depth on the
718 maintainability of object-oriented software. Empirical Software Engineering, 1(2):109–132.
719 D’Ambros, M., Lanza, M., and Robbes, R. (2010). An extensive comparison of bug prediction approaches.
720 In 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), pages 31–41.
721 IEEE.
722 Di Martino, S., Ferrucci, F., Gravino, C., and Sarro, F. (2011). A genetic algorithm to configure support
723 vector machines for predicting fault-prone components. In International Conference on Product
724 Focused Software Process Improvement, pages 247–261. Springer.
725 E Abreu, F. B. and Carapuça, R. (1994). Candidate metrics for object-oriented software within a taxonomy
726 framework. Journal of Systems and Software, 26(1):87–96.
727 Elish, K. O. and Elish, M. O. (2008a). Predicting defect-prone software modules using support vector
728 machines. Journal of Systems and Software, 81(5):649–660.
729 Elish, K. O. and Elish, M. O. (2008b). Predicting defect-prone software modules using support vector
730 machines. Journal of Systems and Software, 81(5):649–660.
731 Fenton, N. (1994). Software measurement: A necessary scientific basis. IEEE Transactions on software
732 engineering, 20(3):199–206.
733 Foley, M. J. (2007). Bugfest! win2000 has 63,000 defects’.
734 G, B. (1991). Object-oriented design and application.
735 Goel, B. M. and Bhatia, P. K. (2012). Investigation of reusability metrics for object–oriented designing.
736 Proceeding of NCETCIT-May, pages 104–110.
737 Golik, P., Doetsch, P., and Ney, H. (2013). Cross-entropy vs. squared error training: a theoretical and
738 experimental comparison. In Interspeech, volume 13, pages 1756–1760.
739 Gómez, O., Oktaba, H., Piattini, M., and García, F. (2006). A systematic review measurement in
740 software engineering: State-of-the-art in measures. In International Conference on Software and Data
741 Technologies, pages 165–176. Springer.
742 Gondra, I. (2008). Applying machine learning to software fault-proneness prediction. Journal of Systems
743 and Software, 81(2):186–195.
744 Grice, W. (2015). Divorce error on form caused by uk government software glitch could affect 20,000
745 people.
746 Gulia, P. and Chillar, R. S. (2012). New proposed inheritance metrics to measure the software complexity.
747 International Journal of Computer Applications, 58(21).
748 Gyimothy, T., Ferenc, R., and Siket, I. (2005). Empirical validation of object-oriented metrics on open
749 source software for fault prediction. IEEE Transactions on Software engineering, 31(10):897–910.
750 Hailperm, B. and Santhanam, P. (2002). Software debugging, testing, and verification. IBM Systems
751 Journal, 41(1):4–12.
752 Halstead, M. H. (1977). Elements of software science.
753 Han, J., Pei, J., and Kamber, M. (2011). Data mining: concepts and techniques. Elsevier.
754 Harrison, R., Counsell, S. J., and Nithi, R. V. (1998). An evaluation of the mood set of object-oriented
755 software metrics. IEEE Transactions on Software Engineering, 24(6):491–496.
756 Henderson-Sellers, B. (1995). Object-oriented metrics: measures of complexity. Prentice-Hall, Inc.
757 Henry, S. and Kafura, D. (1981). Software structure metrics based on information flow. IEEE transactions
758 on Software Engineering, (5):510–518.
759 Hinojosa, S., Oliva, D., Cuevas, E., Pérez-Cisneros, M., and Pájares, G. (2018). Real-time video
760 thresholding using evolutionary techniques and cross entropy. In 2018 IEEE Conference on Evolving
761 and Adaptive Intelligent Systems (EAIS), pages 1–8. IEEE.
762 Jayanthi, R. and Florence, L. (2018). Software defect prediction techniques using metrics based on neural
763 network classifier. Cluster Computing, pages 1–12.
764 Jiarpakdee, J., Tantithamthavorn, C., and Hassan, A. E. (2018). The impact of correlated metrics on defect
765 models. arXiv preprint arXiv:1801.10271.
766 Jing, X.-Y., Ying, S., Zhang, Z.-W., Wu, S.-S., and Liu, J. (2014). Dictionary learning based software
767 defect prediction. In Proceedings of the 36th International Conference on Software Engineering, pages
768 414–423. ACM.
769 Jureczko, M. and Madeyski, L. (2010). Towards identifying software project clusters with regard to
770 defect prediction. In Proceedings of the 6th International Conference on Predictive Models in Software
Osborn, K. (2016). Software glitch causes f-35 to incorrectly detect targets in formation. Technical report, Fox News, May.

Ostrand, T. J., Weyuker, E. J., and Bell, R. M. (2004). Where the bugs are. In ACM SIGSOFT Software Engineering, page 9. ACM.

Kaner, C., Bach, J., and Pettichord, B. (2008). Lessons learned in software testing. John Wiley & Sons.

Karunanithi, S. and Bieman, J. M. (1993). Candidate reuse metrics for object oriented and ada software. In [1993] Proceedings First International Software Metrics Symposium, pages 120–128. IEEE.

Kaur, A. and Kaur, I. (2018). An empirical evaluation of classification algorithms for fault prediction in open source projects. Journal of King Saud University-Computer and Information Sciences, 30(1):2–17.

Kline, D. M. and Berardi, V. L. (2005). Revisiting squared-error and cross-entropy functions for training neural network classifiers. Neural Computing & Applications, 14(4):310–318.

Krishna, G. and Joshi, R. K. (2010). Inheritance metrics: what do they measure? In Proceedings of the 4th Workshop on Mechanisms for Specialization, Generalization and Inheritance, page 1. ACM.

Kumar, R. and Gupta, D. (2012). A heuristics based review on ck metrics. International Journal of Applied Engineering Research, 7(11):2012.

Lake, A. and Cook, C. (1994). Use of factor analysis to develop oop software complexity metrics. In Proc. 6th Annual Oregon Workshop on Software Metrics, Silver Falls, Oregon. Citeseer.

Lee, Y. (1995). Measuring the coupling and cohesion of an object-oriented program based on information flow. In Proc. Int’l Conf. Software Quality, 1995.

Li, W. (1998). Another metric suite for object-oriented programming. Journal of Systems and Software, 44(2):155–162.

Li, W. and Henry, S. (1993). Maintenance metrics for the object oriented paradigm. In [1993] Proceedings First International Software Metrics Symposium, pages 52–60. IEEE.

Lorenz, M. and Kidd, J. (1994). Object-oriented software metrics: a practical guide. Prentice-Hall, Inc.

Majumdar, R. (2010). Paul ammann and jeff offutt introduction to software testing. cambridge university press (2008). isbn: 978-0-521-88038-1.£ 32.99. 322 pp. hardcover. The Computer Journal, 53(5):615–615.

Mal, S. and Rajnish, K. (2013). New quality inheritance metrics for object-oriented design. International Journal of Software Engineering and Its Applications, 7(6):185–200.

Malhotra, R. (2015). A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing, 27:504–518.

Malhotra, R. and Jain, A. (2012). Fault prediction using statistical and machine learning methods for improving software quality. Journal of Information Processing Systems, 8(2):241–262.

Malhotra, R., Kaur, A., and Singh, Y. (2010). Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines. International Journal of System Assurance Engineering and Management, 1(3):269–281.

McCabe, T. J. (1976). A complexity measure. IEEE Transactions on software Engineering, (4):308–320.

McConnell, S. (2004). Code complete. Pearson Education.

Menzies, T. and Di Stefano, J. S. (2004). How good is your blind spot sampling policy. In Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings., pages 129–138. IEEE.

Menzies, T., DiStefano, J., Orrego, A., and Chapman, R. (2004). Assessing predictors of software defects. In Proc. Workshop Predictive Software Models.

Menzies, T., Greenwald, J., and Frank, A. (2007). Data mining static code attributes to learn defect predictors. IEEE transactions on software engineering, 33(1):2–13.

Mishra, D. and Mishra, A. (2010). Object-oriented inheritance metrics: what do they measure? In Proceedings of the 4th Workshop on Mechanisms for Specialization, Generalization and Inheritance, page 1. ACM.

Monarchi, D. E. and Puhr, G. I. (1992). A research typology for object-oriented analysis and design. Communications of the ACM, 35(9):35–48.

Nair, V., Agrawal, A., Chen, J., Fu, W., Mathew, G., Menzies, T., Minku, L., Wagner, M., and Yu, Z. (2018). Data-driven search-based software engineering. In 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR), pages 341–352. IEEE.

Niu, N. and Mahmoud, A. (2012). Enhancing candidate link generation for requirements tracing: the cluster hypothesis revisited. In 2012 20th IEEE International Requirements Engineering Conference (RE), pages 81–90. IEEE.

Osborn, K. (2016). Software glitch causes f-35 to incorrectly detect targets in formation. Technical report, Fox News, May.

Ostrand, T. J., Weyuker, E. J., and Bell, R. M. (2004). Where the bugs are. In ACM SIGSOFT Software Engineering, page 9. ACM.
Engineering Notes, volume 29, pages 86–96. ACM.

Ostrand, T. J., Weyuker, E. J., and Bell, R. M. (2005). Predicting the location and number of faults in large software systems. IEEE Transactions on Software Engineering, 31(4):340–355.

Pason, D. (1994). Object-oriented Programming. DP Publication.

Radjenović, D., Heričko, M., Torkar, R., and Živković, A. (2013). Software fault prediction metrics: A systematic literature review. Information and software technology, 55(8):1397–1418.

Rajnish, K. and Bhattacharjee, V. (2005). Maintenance of metrics through class inheritance hierarchy. In proceedings of International conference on Challenges and Opportunities in IT Industry, page 83.

Rajnish, K. and Bhattacharjee, V. (2006a). Class inheritance metrics and development time: A study. International Journal Titled as PCTE Journal of Computer Science, 2(2):22–28.

Rajnish, K. and Bhattacharjee, V. (2006b). A new metric for class inheritance hierarchy: An illustration. In proceedings of National Conference on Emerging Principles and Practices of Computer Science & Information Technology, pages 321–325.

Rajnish, K. and Bhattacharjee, V. (2007). Applicability of weyuker property 9 to object-oriented inheritance tree metric—a discussion. In 10th International Conference on Information Technology (ICIT 2007), pages 234–236. IEEE.

Rajnish, K. and Bhattacharjee, V. (2008). Class inheritance metrics—an analytical and empirical approach. INFOCOMP, 7(3):25–34.

Rajnish, K., Bhattacharjee, V., and Singh, S. (2007). An empirical approach to inheritance tree metric. In proceedings of National Level Technical Conf.(Techno Vision-2007), pages 145–150.

Rajnish, K., Choudhary, A. K., and Agrawal, A. M. (2010). Inheritance metrics for object-oriented design. International Journal of Computer Science & Information Technology, 2(6).

Rajnish, K. and Singh, Y. (2013). An empirical and analytical view of new inheritance metric for object-oriented design. International Journal of Computer Applications, 65(12):44–50.

Rathore, S. S. and Kumar, S. (2017a). A decision tree logic based recommendation system to select software fault prediction techniques. Computing, 99(3):255–285.

Rathore, S. S. and Kumar, S. (2017b). An empirical study of some software fault prediction techniques for the number of faults prediction. Soft Computing, 21(24):7417–7434.

Reddy, B. R. and Ojha, A. (2018). Discrimination of inheritance patterns: An improved metric. In 2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence), pages 653–657. IEEE.

Rosenberg, L. (1998). Applying and interpreting object oriented metrics. In Software Technology Conference, Utah, April 1998.

Seliya, N., Khoshgoftaar, T. M., and Van Hulse, J. (2010). Predicting faults in high assurance software. In 2010 IEEE 12th International Symposium on High Assurance Systems Engineering, pages 26–34. IEEE.

Sheldon, F. T., Jerath, K., and Chung, H. (2002). Metrics for maintainability of class inheritance hierarchies. Journal of Software Maintenance and Evolution: Research and Practice, 14(3):147–160.

Shepperd, M., Song, Q., Sun, Z., and Mair, C. (2013). Data quality: Some comments on the nasa software defect datasets. IEEE Transactions on Software Engineering, 39(9):1208–1215.

Sherer, S. A. (1995). Software fault prediction. Journal of Systems and Software, 29(2):97–105.

Shirabad, J. S. and Menzies, T. J. (2005). The promise repository of software engineering databases. School of Information Technology and Engineering, University of Ottawa, Canada, 24.

Shivam (2013). A study on inheritance using object oriented programming with c++. In International Journal of Advance Research in Computer Science and Management Studies, volume 1, pages 375–380. IJARCSMS.

Singh, Y., Kaur, A., and Malhotra, R. (2009). Software fault proneness prediction using support vector machines. In Proceedings of the world congress on engineering, volume 1, pages 1–3.

Subramanyam, R. and Krishnan, M. S. (2003). Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. IEEE Transactions on software engineering, 29(4):297–310.

Suresh, Y., Kumar, L., and Rath, S. K. (2014). Statistical and machine learning methods for software fault prediction using ck metric suite: a comparative analysis. ISRN Software Engineering, 2014.

Tang, M.-H., Kao, M.-H., and Chen, M.-H. (1999). An empirical study on object-oriented metrics. In Proceedings sixth international software metrics symposium (Cat. No. PR00403), pages 242–249.
IEEE.

Tegarden, D. P., Sheetz, S. D., and Monarchi, D. E. (1995). A software complexity model of object-oriented systems. *Decision Support Systems*, 13(3-4):241–262.

Wagner, S. (2010). A bayesian network approach to assess and predict software quality using activity-based quality models. *Information and Software Technology*, 52(11):1230–1241.

Xing, F., Guo, P., and Lyu, M. R. (2005). A novel method for early software quality prediction based on support vector machine. In *16th IEEE International Symposium on Software Reliability Engineering (ISSRE’05)*, pages 10 pp.–222.

Yu, L. (2012). An evolutionary programming based asymmetric weighted least squares support vector machine ensemble learning methodology for software repository mining. *Inf. Sci.*, 191:31–46.

Yu, P., Systa, T., and Muller, H. (2002). Predicting fault-proneness using oo metrics. an industrial case study. In *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, pages 99–107. IEEE.