Software Metrics Selection for Fault Prediction: A Review

Anil Kumar Pandey[*1] and Manjari Gupta[1]

1.DST-CIMS, Banaras Hindu University, Lanka ,Varanasi-221005,Uttar Pradesh,India

[1]akpandey@bhu.ac.in,[2]majarigupta.cs@gmail.com

Abstract

Software testing is a critical phase that is extreme importance in the development life cycle of a software. It helps to identify software faults or loacate defects. Removal of bugs and correction of faults in a software is a time and resource thirsty activity that requires cpmrehensive planning and proper optimization of resources. And, if software metrics are involved in large numbers, it further complicates the process of software defect prediction. Thus, proper identification and selection of metrics which can enhance the performance of software defect prediction methods become highly pertinent as well as extremely challenging. This paper reviews often used software metrics for predicting software fault, specifically those, that do so by using machine learning algorithm.

Keywords: Software metrics, Fault prediction, Defect prediction, Cyclomatic Complexity

1. Introduction

Relevant literatures tell us that the expenditure in commercial software usage throughout the world was $3.8 trillion in 2014 [1]. A hefty portion of 23% of the amount was spent on ensuring high quality amd testing efficacy [2]. The expenditure percentage allocated for software testing in software development lifecycle (SDLC) signifies the importance and criticality of this process. However, largeness and complexities that are generally inherent in software easily give rise to attributes defects(faults). Providing quality assurance for such complex and sometimes highly evolving software systems becomes very challenging. Though it is unfortunate, we can see that defects in large-scale software systems are inevitable and proper mechanisms for locating and correcting the faults need to be in place. For the sake of ensuring the system's quality this mechanism has to execute a large number of exhaustive tests, making the job extremely tedious and highly expensive.

Hence, effective and accurate prediction mechanism about components that are defect-prone in softwares has become an integral part and is highly essential part of software engineering today. Further, validating the age old adgae that 'a stitch in time saves nine,' post deployment location and correction of defect in a software is often far more expensive than its indentification and repair during the process of development itself [4]. Moreover, delay in fault prediction gives it an easy chance to the fault being propagated to subsequent stages of development and thereby making the process of fault prediction far more complicaed. Furthermore, the unavailabilty of the relevant data for fault prediction which have to be based on its own failures at those stages gives rise to a catch 22 situation and make the whole endeavour extremely challenging [5]. In general the level of softwae reliability is

designated by using both internal and external metrics as software quality measures. It is only the software's source code that is considered in measuring the internal metrics whereas it is the behaviour and the functionalities of the software that is used while measuring external metrics [2]. The identification and location of the key internal metrics that are the cause of defects at an early of development is imperative for software engineers to be able to predict the key metrics of entities of develpoment life-cycles at a later stage for being able to montor and control the performance of the products. Changes across various entities such as process, product and resources also need to be tracked with released of different and newer versions. With every subsequent release we can find changes happening in the design, code, regression tests, test requirements, processes and resources. If changes are found in the key attributes of different entities they can well be harbingers of problems which might become embedded in design, or a possible compromise in the quality and reliability of the product. Therefore, identifying the modules at the development phase which might become error prone and might be difficult to manage and test later is always a better thing to do. For this purpose we have to measure a large number of internal metrics of different entities. The monitoring, measurement, analysis and tracking of diffeernt entitties of a software that is highly evolving becomes resource consuming and expensive for the software developers which cannot be overlooked if the quality and reliability of the final product is to be ensured. Contemporary and current defect predection literatures show limited capability and are distinctly under-achievers as far as less expensive and better optimized measurement processes are concerned [6] [7].

Generally, every software has a relation to certain functional properties or the other of the software project e.g., coupling, cohesion, inheritance, code change, etc., and these properties are indicative of external quality attributes like reliability, testability, or fault-proneness [8].

Table 1.Taxonomy of Software Metrics

Software Metrics			
Process Metrics	**Product Metrics**		
Code Delta: Delta of LOC, Delta of Changes **Code Churn:** Total LOC Churned, LOC Deleted LOC , File Count, weeks churn, churn cost and files churn etc.	**Dynamics Metrics**	**OO Metrics**	**Traditional Metrics**
	Yacoub Metrics Suit: Export Object Coupling (EOC), Import Object Coupling (IOC) **Airdelm Metrics Suit**: IC_OD,IC_OM, IC_OC,IC_CD, IC_CM, IC_CC,	**CK Metrics Suit:** CBO, LCOM, DIT, NOC,RFC and WMC **Wd Li Metrics:** CTI,CTM,CTA, NOM, SIZE1,SIZE2 **Lorene and Kidd's**	**Size Based :** Functional Point(FP), Source Lines of Code (SLOC), Kilo-SLOC **Quality based :** Defect per FP after delivery, Defect per

Change Metrics, Revisions, Refactoring , Bug fixes , ashes, etc. **Developer Based** Personal commit Sequence, No. of corrections etc **Requirement of Metrics:** Action, Conditional, Continuance , Imperative etc.	EC_OD,EC_OM, EC_OC, EC_CD, EC_CM, EC_CC, **Michel Metrics Suits:** Dynamic CBO for a class, Degree of Dynamic Coupling between two classes at runtime, Degree of Dynamic Coupling within a given set of classes, R1,R2,RD1,RD2	**Metrics Suit:** PIM,NIM,NIV,NCM,NC,NMO,NMI,NMA,SIX,APM **Mood metrics suit:** MHF,AHF, MIF,AF,PF,CF **Brizzed Metrics Suits:** IFCAIC,ACAIC,OCAIC,FCAEC,DCAEC etc. **Bensiva metrics Suit:** DAM,DCC,CIS,MOA,MFA,DSC,NOH,ANA,CAM,NOP,NPM	SLOC after delivery **System Complexity**: Cyclomatic Complexity, McCabe Complexity etc. **Halsted metrics:** Number of Distinct operators and Number.of distinct operands etc.

A broad classification of different metrics are illustrated in the above table1.

There are two broad classification of software metrics-

i. Product metrics : Calcultion of product metrics are done using different features of the finally developed software. These metrics confirm whether certain norms of standardisation like that of ISO-9126 are met or not. Broadly, we have three classification of product metrics i.e., traditional metrics, object-oriented metrics, and dynamic metrics [9].

a) Traditional metrics : It comprises the software metrics that were designed during the initial periiods when the field of sofware engineering was nascent and slowly emerging, these can been classified as traditional metrics. It mainly includes the following metrics: –

- Metrics of size like, Source lines of code (SLOC), Kilo-SLOC (KSLOC) and Function Points (FP).
- Metrics of quality like, Defects per FP after delivery, Defects per SLOC (KSLOC) after delivery –
- Metrics of system complexity like, McCabe Complexity, Cyclomatic Complexity, and Structural complexity [10].
- Halstead metrics [11].n1, n2, N1, N2, n, v, N, D, E, B, T

b) Chidamber and Kemerer proposed a software metrics suite for OO software known as CK metrics suite. Given below are some of the OO metrics suites : –

- Chidamber-Kemerer (CK) metrics suite like, Depth of Inheritance Tree (DIT) metric, Coupling between Object class (CBO) metric, a Class (RFC) metric, Lack of Cohesion in Methods (LCOM) metric, Response for Weighted Method Count (WMC) metric and Number of Children (NOC) metric etc. [12].

- MOODS metrics suite like, Attribute Hiding Factor (AHF) metric, Method Hiding Factor (MHF) metric, Method Inheritance Factor (MIF) metric, Attribute Inheritance Factor (AIF) metric, Polymorphism Factor (PF) metric and Coupling Factor (CF) metric etc. [13]
- Wei Li and Henry metrics suite like, Coupling Through Inheritance, Coupling Through Message passing (CTM), Coupling Through ADT (Abstract Data Type), Number of local Methods (NOM), SIZE1 and SIZE2 [14].
- Lorenz and Kidd's metrics suite like, PIM, NIM, NIV, NCM, NCV, NMO, NMI, NMA, SIX and APPM [15].
- Bansiya metrics suite like, DAM, DCC, CIS, MOA, MFA, DSC, NOH, ANA, CAM, NOP and NOM [16].
- Briand metrics suite like, IFCAIC, ACAIC, OCAIC, FCAEC, DCAEC, OCAEC, IFCMIC, ACMIC, OCMIC, FCMEC, DCMEC, OCMEC, IFMMIC, AMMIC, OMMIC, FMMEC, DMMEC, OMMEC [17].

c) Dynamic metrics: Dynamic metrics are those metrics that depend on the features that are collected from a program in use. Their importance lies in the fact that they are real time and based on how the collected software components behave at the actual time of software's execution, they measure specified runtime properties of programs, components, and systems [18]. Contrary to the static metrics that use non-executing static models for gathering features and calculations, similarly the most run-time coupled and highly complex objects and features are identified by dynamic metrics. These metrics give us a very distinct indication on the design quality [19]. Some examples of the dynamic metrics suites are given below:

- Yacoub metrics suite like, Export Object Coupling (EOC) and Import Object Coupling (IOC) [19].
- Arisholm metrics suite like, IC_OD, IC_OM, IC_OC, IC_CD, IC_CM, IC_CC, EC_OD, EC_OM, EC_OC, EC_CD, EC_CM, EC_CC [20]
- Mitchell metrics suite like, Dynamic CBO for a class, Degree of dynamic coupling between two classes at runtime, Degree of dynamic coupling within a given set of classes, RI , RE , R DI , R DE [21].

ii. Process metrics : As the name suggests process metrics are based upon the features collected across the complete gamut of the software development life cycle. They help formulate better strategies for future processes of software development. They are extremely useful in helping in standardization of group of process measures which consequently lead to a software process improvement in the long term [9]. For the measurement of effectiveness of a process which is designed in this way one deerives a set of metrics that based on results from the process, such as

Number of modules that have been changed for a specific bug-fix, finally delivered Work products, Calendar time expended, Schedule Conformance, and effort that was put in as well as time taken to complete each and every distinct activity [9]. Some Process metrics are as follows :

a) Code delta metrics like, Delta of changes and Delta of LOC [22].
b) Code churn metrics like, Total LOC, Churned LOC, Deleted LOC, File count, Weeks of churn, Churn count and Files churned [23].
c) Change metrics like, Revisions, Refactorings, Bugfixes, Authors, LOC added, Max. LOC Added, Ave. LOC Added, LOC Deleted, Max. LOC Deleted, Ave. LOC Deleted, Codechurn, Max. Codechurn, Ave. Codechurn, Max. Changeset, Ave. Changeset and Age [22].
d) Developer based metrics like, Personal Commit Sequence, Number of Commitments, Number of Unique Modules Revised, Number of Lines Revised, Number of Unique Package Revised, Average number of Faults Injected by Commit, Number of Developers Revising Module and number of Lines of Code that were Revised by Developer [24].
e) Requirement metrics like, Action, Conditional, Continuance, Imperative, Incomplete, Option, Risk level, Source and Weak phrase [25].
f) Network metrics like, Betweenness centrality, Closeness centrality, Eigenvector Centrality, Bonacich Power, Structural Holes, Degree centrality and Ego network measure [26].

2. Related Literature Review

Achievement of a predecided goal with an ensured high quality of software is a formidable challenge. To this end quality assurance cannot depend upon traditional testing approaches which fall short because they tend to have limited opportunities of being tested by a user during the process of development of software. Automated techniques are now commonly used to predict software defects at different stages all across the development life cycle off software with an aim of monitoring software quality. This entails tracking and measurement of a vast number of metrics making it time consuming, resource thirsty and expensive. Making the determination of significant metrics for which software faults can be identified as highly crucial.

There have been multiple endeavours for analyzing the possible inherent capabilities of the software metrics intended for fault prediction purposes in a software. In this field a paradigm shift came about when the data repositories of NASA and PROMISE became publicly available, and the using open source software projects (OSS) became very commonplace among the researchers for their studies. The use of OSS allows easy replication and verification of the findings of the investigations.

An extensive review of the reports of various studies that was done by us is summarized in the Table 2 [63][64].

Table 2. Studies by Authors regarding Software Matric

Study/Aim of Metrics Evaluation	Method Used	Outcome
Li and Henry [14]. Fault Proneness	All CK Metrics,Two commercial software Logistic Regression	CK metrics suite were evaluated for the first time for fault prediction. It was found that with and exception of LCOM all the metrics accurately predicted fault proneness. However, the Correlation of considered metrics with fault proneness was not investigated
Ohlsson et al. [7]. Fault Proneness	Various Design Metrics. Ericsson Telecom AB System. Principal component and discriminant analysis	The capability of fault prediction of various design metrics was evaluated. Found a significant correlation of all the used metrics to fault proneness. However this evaluation study was performed over a single software project. Evaluation of fault prediction models not thorough
Tang et al. [27]. Fault Proneness	All CK Metrics. Three industrial real time systems. Logistics regression analysis	The correlation of OO metrics with fault proneness was investigated. A significant correlation between RFC and WMC metrics to fault proneness was found. There was an absence of formation of a fault prediction model to evaluate the capability of considered metrics for fault prediction.
Chidamber et al. [12]. Productivity and design efforts	All CK Metrics. Three Commercial Trading Systems Stepwise Regression	Evaluation of OO Metrics for practicing project management information. A significant correlation between CBO and LCOM to fault proneness was found. The capabilty of the considered metrics has been assessed by the use of correlation analysis only. There was an absence of use of any fault prediction model.
Emam and Melo [28]. Fault Proneness	All Briand Metrics.One Version of a commercial Java application. Logistic Regression	Comparative capability of the software metrics were evaluated over the subsequent releases of software projec. Only OCAEC, ACMIC and OCMEC metrics showed a correlation with fault-proneness. However only one software system was studied for evaluation. An exhaustive evaluation of fault prediction models was not done.
Briand et al. [29]. Fault Proneness	All metrics of CK metrics suites and Briand metrics suite. An open multi-agent system development environment. Logistics regression and principal component	The relationship between fault proneness and OO desig metrics was investigated. Importance of coupling metric as predictor of faults was established. Import coupling has a stronger impact on fault pronenes than the impact of export coupling. The evaluation c considered metrics has been based only on correlatio measure. The experiments have been performed usin only one software.

	analysis.	
Shanthi and Duraiswamy [30]. Error Proneness	All MOOD metrics Mozilla e-mail suite Logistic regression, decision tree and neural network	First study to evaluate MOOD metrics suite for fault prediction. Found a significant correlation between all the metrics with error proneness. However, it lacked a thorough evaluation of fault prediction models and correlation of the metrics used with fault prediction was not calculated as well.
Shatnawi et al. [31]. Error proneness and design efforts	Various OO design metrics Eclipse 2.0 Uni-variate binary regression and stepwise regression	Prediction of many quality factors of OO metrics were evaluated. A association between CTA and CTM with error proneness was established. Experiments have been performed using only a single project. Evaluation of the results have only been based on statistical measures.
Shatnawi and Li [32]. Error Proneness	Various OO design metrics Three release of Eclipse Project Multivariate logistic regression	Prediction for severity of faults of software metrics were evaluated. Established that CTA, CTM and NOA metrics predicted class-error probability in all error severity categories quite well. The collection of dataset of faults has been done with the help of some commercial tools leasding to questionable accuracy.
Kpodjedo et al. [33]. Number of defects	All CK Metrics, class rar (CR), evolution cost (EC) Ten versions of Rhino Logistic Regressio classification regressic trees	A couple of new search-based metrics were proposed and studied. It was found that WMC, LCOM, RFC and EC metrics were quite promising for defect prediction. The proposal lacks theoretical validation of the concerned metrics. Metrics was extracted using home made tools without any proper validation.
Selvarani et. al. [34]. Defeat proneness	RFC, WMC, and DIT Two commercial projects Property based analysis domain knowledge of experts	Evaluation on the basis of the threshold values of OO metrics was done. The study showed that the influence of DIT on defect proneness was between 10-33% for a given value of 1-3, when the value of RFC crosses 100 it causes more defects. No faults are caused when the value of WMC lies between 25 and 60. This study is based only on three metrics and we see that there is a lack of details of fault datasets as well.
Elish et. al. [35]. Fault Proneness in package level	Martin, MOOD, and CK metrics suites Eclipse project Spearman's correlation and multivariate regression	On comparative evaluation of three package level metrics (i.e., Martin, MOOD and CK metric suites), it was found that Martin metrics suite is more accurate than the MOOD and CK suites for fault prediction. A few more case studies are needed to properly validate this finding.
Singh and Verma [36]. Fault prediction	All CK metrics Two version of iText, a JAVA-PDF library software J48 and Naïve Bayes	Some OSS (open source projects) was used for evaluation of CK metric suite for fault prediction. It was found that fault proneness of the software was well indicated by CK metrics. The use of some commercial tools in the collection fo

		the fault dataset leads to an uncertainty in accuracy limiting the evaluation results.
Chowdhury and Zulkernine [37]. Prediction of vulnerability	Complexity, coupling a cohesion metrics (CC metrics) Mozilla Firefox Decision tree, Naïve Baye random forest, and logis regression	Vulnerability prediction of OO metrics was evaluated. Results showed that vulnerability prediction can be made with the use of CCC, and this was irrespective of the prediction technique used. The vulnerability prediction of only a single metrics i.e., CCC metrics have been evaluated. It needs further evaluation of other metrics for vulnerability.
Dallal and Briand [38]. Early stage fault prediction	Connectivity based object oriented class cohesion metrics Four open- sources software projects Correlation and principal component analyses, logistic regression	It showed that with a lower value of cohesion we tend to get more faults. Path-connectivity cohesion metrics showed a greater promise than most of the cohesion metrics. If metrics have two or more features that are similar, these features tend to get merged by the metric, rendering it difficult to discriminate which attribute is expected to be accessed by a method, since the methods of same types are being called.
Rathore and Gupta [39]. Fault prediction	18 OO Metrics 6 Releases of PROP dataset from PROMISE repository Spearman correlation, logistic and linear regression	Individual evaluation of various OO metrics along with an evaluation in combination with other metrics was made. Results showed that classes with high coupling tend to show a higher degree of fault proneness. There is no relevance of Cohesion and Inheritance metrics in predicting fault proneness. The utility of the proposed methodology needs to be established properly by more datasets.
Rathore and Gupta [40]. Fault Prediction	18 OO metrics 5 softwares with their 1 releases from PROMIS repository Spearman Correlatio logistics and linea regression	Fault prediction capabilities of OO metrics over multiple releases of software was validated by this study. It showed that with the exception of cohesion metric, all the considered metrics significantly predicted faults. More case studies are needed to establish the usefulness of the proposed methodology.
Peng et al. [41]. Software Fault prediction.	CK, Martin's, QMOOD, extended CK metrics suites, complexity metrics, and LOC 10 PROMISE project datasets with their 34 releases J48, logistics regression, Naïve Bayes, decision table, SVM, and Bayesian network	This study helped determine a subset of metrics that show a promising usefulness for fault prediction. An evaluation of top five frequently used software metrics was done and it was found that they produced comparable results of fault prediction to the results that were produced by using full set of metrics. The models for fault prediction comprised of all, filter and top 5 metrics only. This does not take into account the fact the possibilty of other combination of software metrics. There has been no analysis of the data distribution while exercising the statistical test.
Process Metrics Graves et al.	Change history A legacy system written in C	Age and change history metrics were evaluated for fault prediction. It was found that the better predictor was numbers of changes to the module whereas when

[42]. Predicting number of faults	Generalized linear models	predicting the number of fault is concerned number of developers was of almost no help at all. The evaluation experiment used just one software project, and as far as evaluation of the software metric is concerned no separate and distinct fault prediction model was built.
Nikora and Munson [43]. Fault Prediction	Source code metrics, change metrics Darwin System Principle component Analysis	A new method for selecting significant metrics for fault prediction was defined. Results showed an enhanced and high quality fault prediction models were possible with these new defined metrics. Here also we see a lack of comparison analysis and a lack of distinctive and separate fault prediction model.
Hassan [44]. Prediction of Faults	Change complexity metrics 6 open source projects Linear regression and statistical test	Code change process was used as a base for complexity metrics. It was found that in comparison to traditional software metrics change complexity metrics were a better predictors of fault proneness. However, the proposed metrics have not been substantiated by any theoretical validation. This prediction methodology stands on the validations given by very small number of fault datasets.
Bird et al.[45]. Prediction of software failures	Socio- Technical networks metrics Windows vista and the ECLIPSE IDE Principal component analysis and logistic regression	Combined socio-technical metrics were investigated for their influence on fault prediction. The socio-technical metrics seemed to produce better recall values in comparison to dependency and contribution metrics. It has a limited proof as the number of case studies were low. A higher number of case studies are required to ascertain its usefulness..
Nachiappan et al. [22]. Prediction of defect-prone components	Change bursts suite Windows Vista Stepwise regression	Capabilities of change metrics were studied for fault prediction. It showed that change burst metrics provide excellent defect predictions. There is a lack of thoroughness as far as evaluation and validation of the results are concerned.
Mastsumoto et al. [24]. Fault Prediction	Developer metric suite Eclipse Project dataset Correlation and linear regression analysis	Influence of developer features on fault prediction was studied. The conclusion was that developers metrics make good predictors of defects. There is a lack of validation of proposed metrics, with only a single software project being used for evaluation experiments.
Krishnan et al. [46]. Prediction of fault prone files.	Change metrics suite Three releases of Eclipse J48 algorithm	Change metrics over multiple releases of the software project were evaluated for fault prediction. Showing that all change metrics worked as nice predictors of defects. However again just a single fault prediction technique has been used for the validation of the results. Only one dataset over three releases were used for the studying software fault prediction.

Devine et al. [47]. Faults Prediction	Various source code and change metrics Poly Flow a suite of software testing tools Spearman correlation	A componenet level investigation of the association of software faults with other metrics was done. Results indicated that with the exceptions of average file churn metrics and average complexity metrics, all other metrics showed a positive correlation with defects. For the proposed methodology to be established, a higher number of case studies need to be performed.
Ihara et al. [48]. Bug-fix prediction	21 variables of base, status, period and developer metrics Eclipse project Regression analysis	A model using many software metrics for bug-fix prediction was developed. It established the all importance of the base metrics while the building and formulation of the model. The validation was based on the evaluation and study of a single fault prediction technique, this was done using only one software with a small timeframe of 3 months of releases to its credit.
Rahman and Devanbu [49]. Defect prediction	14 process metrics, Various code metrics. 12 projects developed by Apache Logistic regression, J48, SVM and Naïve Bayes	For this purpose combined capability of process and code metrics for fault prediction was studied, with a result that process metrics almost always outperformed code metrics. The accuracy is not well ascertaind as commercial tools have been used to calculate code. The results are not based on exhaustive evaluation.
Ma et al. [50]. Fault proneness	Requirement metrics and design metrics CMI, PCI Naïve Bayes, AdaBoost, bagging, random forest, logistic regression	Requirement metrics were evaluated for fault prediction for the first time. Results indicated that if the requirement metrics were combined with design metrics the fault prediction capabilities showed a marked improvement. A limited number ie., a couple of datasets were used for evaluation studies of considered software metrics. No analysis of distribution of data for statistical tests was done.
Wu et al. [51]. Fault prediction	8 developer metrics, 22 process and product metrics 8 open source Java projects Principle component analysis	The influence of quality of developer on software fault prediction was evaluated. The result indicated improvementFound that the combination of developer, process, and product metrics produced improved fault prediction results. The methodology needs to be evaluated with more datasets of different domains for the purpose of validation..
Xia et al. [52]. Faults Prediction	Code metrics and process metrics Tracking telemetry and control (TT and C) software Improved PSO and optimized SVM	A Selection approach was proposed for useful software metrics for fault prediction. Results indicated that out of number of code and process metrics used, CM, MMSLC, and HM metrics showed a significant influence on fault prediction. With only a couple of fault datasets being used for the purpose of validation of the results, the selection approach which has been proposed is based on

		inadequate validation to establish its applicability.
Other metrics Zhang [53]. Defect Prediction	LOC Three versions of the Eclipse system (2.0, 2.1 and 3.0), 9 NASA projectsSpearman correlation, multilayer perceptron, logistics regression, Naïve Bayes, decision tree	A relation between LOC and software was analyzed. defects. Results showed that a weak but positive relationship exists between LOC and defects. 20% of the largest files are responsible for 62.29% prelease defects and 60.62% post-release defects. However this study too is based on just one metric (LOC). The domain of data has not been analyzed for the statistical tests.
Rana et al. [54]. Number of defects prediction	Sotware Science metrics (SSM) KCI dataset Bayesian, decision tree, linear regression, support vector regression	Effectiveness of SSM metrics in classification of software modules as defective or defect free was investigated. Its performance for identification of number of defects prediction was not up to mark. Validation of results has been based on a very small dataset and concerned metrics' correlation with fault proneness has not been evaluated.
Mizuno and Hata [55]. Fault prone module detection	Complexity and text feature metrics Three versions of the Eclipse System (2.0, 2.1 and 3.0) Logistic regression	A number of complexity and text metrics were evaluated for fault prediction. Results indicate the better performance of complexity metrics over text metrics for the purpose of fault prediction. The study hasn't been substantiated by providing the details of collection of data and its preprocessing. The metrics has also not been validated theoretically.

3. Discussion

Achievement of a predecided goal with an ensured high quality of software is a formidable challenge. To this end quality assurance cannot depend upon traditional testing approaches which fall short due to the limited opportunities of user testing in this approach during the development of software. The differing results seem to arise due to contest between and nature of data compiled. Hall et al [63] submitted that models between the context may affect prediction result.

The correlation between fault proneness and size metric (LOC) and fault proneness has been investigated many times [53][56] Ostrand et a.[57] came up with the model for prediction of fault density with the use of LOC metrics and came to the result that a significant correlation between LOC metric and prediction of fault density exists. Zhang [53], in a different study came to the conclusion that sufficient statistical evidence existed to show that a weak relationship though positive, between LOC and defects does exist. But Rosenberg [59] gave a contrary opinion pointing out that though, there exists a relationship between defect density and LOC, it is negative. Further, they came to a conclusion that LOC in combination with other software metrics becomes the most pertinent feature in software fault prediction. In another study, Emam and Melo [28] demonstrated that a simple relationship did exist between class size and faults, they further went on to demonstrate that class size

had no threshold effect on the occurrences of faults. Various researchers [14][17][59][60] have examined the use of complexity metrics for building fault prediction models. The predictive capabilitis of complexity metrics were confirmed by some of the studies [59][60], while others reported that they did not perform so well [61]. In a study, Olague et al. [59] came up with the conclusion that the complexity metric shape gave a better result in prediction of faults. Further, it became evident that less commonly used metrics such as AMC and SDMC outperformed metrics like LOC and WMC as good predictors of fault proneness. Zhouetal.[60] reported that when used in a stand alone way complexity metrics exhibited a predictive ability that was average. But their explanatory ability increased formidably when they were used in combination with LOC metric. The evaluation of how appropriate are process metrics for fault proneness has been done through multiple studies [47][62]. After investigating many process metrics Devine et al. [47] found a positive correlation between most of the process metrics and defects. In a comparative study between many process metrics in one hand and code metrics on another, Moser et al.[62] came to the conclusion that process metrics have the capability to differentiate faulty software modules from non faulty ones and perform better in fault prediction when compared to source code metrics. However, Hall et al. [63] came to the conclusion that process metrics do not perform better in comparison to OO metrics. It has been observed that the results do differ when studies are performed on a set of metrics. this probably happens because the context in which data is collected are varied and it also depends on the varied usage of dependent variable (such as fault density, fault proneness, pre-release faults, and post-release faults) during investigation and the implication of linear relationship.

4. Conclusion

We see a strong coupling of established metrics with size measures, like lines of code Complexity metrics and in turn size measures seem to have a certain amount of predictive capabilities. OO (Object-oriented) metrics perform better than complexity and size metrics when compared for prredicting faults and defects. Though showing a certain corelation to size they encompass certain some additional properties as well. Static code metrics, just as complexity, size and OO metrics, are suited for observing a particular version of software, but with a descending accuracy with each software iteration. Hence, they are not suitable for highly iterative, post-release software, where the main cause of faults is because of the development process and not so much because of the properties of size and design. In such cases and environment, we find a better performance of process metrics in comparison to static code metrics as far as the accuracy of prediction of fault is concerned. Most of the studies performed for fault prediction capabilities used size, complexity and OO metrics (70%), whereas process metrics, with apparently high possibility, were used less often (23%). A tendency for researchers and scientist from the industry to use process metrics more was seen, whereas academia based researchers seemed to prefer OO metrics. This might be so, probably because the process

metrics are better performers when used for industrial purposes in comparison to static code metrics: and process metrics fulfill the industrial need better. Hence we suggest and appeal to researchers to take up process metrics in larger numbers for future studies. Further, seemingly one of the three following programming languages was frequently used, i.e. C++ (35%), Java (34%) and C (15%). Other OO languages (e.g. C#) should also be considered in investigating the performance of metrics. Since there seems to be a variation affecting the performance of metrics when different programming languages are used, impetus should be laid on their use in further investigations.

References

[1] Gartner Says Worldwide It Spending on Pace to Reach $3.8 Trillion in 2014. Accessed: September 21, 2018. [Online]. Available: http://www.gartner.com/newsroom/id/2643919

[2] Arar, Ö.F. and Ayan, K., 2015. Software defect prediction using cost-sensitive neural network. *Applied Soft Computing*, *33*, pp.263-277.

[3] Huda, S., Alyahya, S., Ali, M.M., Ahmad, S., Abawajy, J., Al-Dossari, H. and Yearwood, J., 2018. A Framework for Software Defect Prediction and Metric Selection. *IEEE access*, *6*, pp.2844-2858.

[4] Pelayo, L. and Dick, S., 2007, June. Applying novel resampling strategies to software defect prediction. In *Fuzzy Information Processing Society, 2007. NAFIPS'07. Annual Meeting of the North American* (pp. 69-72). IEEE.

[5] Yadav, H.B. and Yadav, D.K., 2015. A fuzzy logic based approach for phase-wise software defects prediction using software metrics. *Information and Software Technology*, *63*, pp.44-57.[6] C. Catal and B. Diri, ‘‘Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem,’’ Inf. Sci., vol. 179, no. 8, pp. 1040–1058, 2009.

[7] Zhao, M., Wohlin, C., Ohlsson, N. and Xie, M., 1998. A comparison between software design and code metrics for the prediction of software fault content. *Information and Software Technology*, *40*(14), pp.801-809.

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

[9] Bundschuh, M. and Dekkers, C., 2008. *The IT measurement compendium: estimating and benchmarking success with functional size measurement*. Springer Science & Business Media.

[10] McCabe, T.J., 1976. A complexity measure. *IEEE Transactions on software Engineering*, (4), pp.308-320.

[11] Halstead, M.H., 1977. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., New York, NY.

[12] Chidamber, S.R. and Kemerer, C.F., 1994. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, *20*(6), pp.476-493.

[13] Harrison, R., Counsell, S.J. and Nithi, R.V., 1998. An investigation into the applicability and validity of object-oriented design metrics. *Empirical Software Engineering*, *3*(3), pp.255-273.

[14] Li, W. and Henry, S., 1993. Object-oriented metrics that predict maintainability. *Journal of systems and software*, *23*(2), pp.111-122.

[15] Lorenz, M. and Kidd, J., 1994. *Object-oriented software metrics* (Vol. 131). Englewood Cliffs: Prentice Hall.

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

[17] Briand, L., Devanbu, P. and Melo, W., 1997, May. An investigation into coupling measures for C++. In *Proceedings of the 19th international conference on Software engineering*(pp. 412-421). ACM.

[18] Tahir, A. and MacDonell, S.G., 2012, September. A systematic mapping study on dynamic metrics and software quality. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on* (pp. 326-335). IEEE.

[19] Yacoub, S.M., Ammar, H.H. and Robinson, T., 1999. Dynamic metrics for object oriented designs. In *Software Metrics Symposium, 1999. Proceedings. Sixth International* (pp. 50-61). IEEE.

[20] Arisholm, E., Briand, L.C. and Foyen, A., 2004. Dynamic coupling measurement for object-oriented software. *IEEE Transactions on software engineering*, *30*(8), pp.491-506.

[21] Mitchell, A. and Power, J.F., 2003. Toward a definition of run-time object-oriented metrics.

[22] Vishwanath, K.V. and Nagappan, N., 2010, June. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*(pp. 193-204). ACM.

[23] Nagappan, N. and Ball, T., 2005, May. Use of relative code churn measures to predict system defect density. In *Proceedings of the 27th international conference on Software engineering* (pp. 284-292). ACM.

[24] Matsumoto, S., Kamei, Y., Monden, A., Matsumoto, K.I. and Nakamura, M., 2010, September. An analysis of developer metrics for fault prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering* (p. 18). ACM.

[25] Jiang, Y., Cukic, B. and Ma, Y., 2008. Techniques for evaluating fault prediction models. *Empirical Software Engineering*, *13*(5), pp.561-595.

[26] Premraj, R. and Herzig, K., 2011, September. Network versus code metrics to predict defects: A replication study. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on* (pp. 215-224). IEEE.

[27] Tang, M. H., Kao, M. H., & Chen, M. H. (1999)."An empirical study on object-oriented metrics. In Proceedings of the 1999 international workshop on Software metric symposium (pp. 242-249). IEEE.

[28] El Emam, K., Melo, W. and Machado, J.C., 2001. The prediction of faulty classes using object-oriented design metrics. *Journal of Systems and Software*, *56*(1), pp.63-75.

[29] Briand, L.C., Bunse, C. and Daly, J.W., 2001. A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. *IEEE Transactions on Software Engineering*, *27*(6), pp.513-530.

[30] Shanthi, P.M. and Duraiswamy, K., 2011. An empirical validation of software quality metric suits on open source software for fault-proneness prediction in object oriented system. *European journal of Scientific Research*, *5*(2), pp.168-181.

[31] Li, W. and Shatnawi, R., 2007. An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution. *Journal of systems and software*, *80*(7), pp.1120-1128.

[32] Shatnawi, R. and Li, W., 2008. The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. *Journal of systems and software*, *81*(11), pp.1868-1882.

[33] Kpodjedo, S., Ricca, F., Galinier, P. and Antoniol, G., 2009, March. Recovering the evolution stable part using an ecgm algorithm: Is there a tunnel in mozilla?. In *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on* (pp. 179-188). IEEE.

[34] Selvarani, R., Nair, T.G. and Prasad, V.K., 2009, May. Estimation of defect proneness using design complexity measurements in object-oriented software. In *2009 International Conference on Signal Processing Systems* (pp. 766-770). IEEE.

[35] Elish, K.O. and Alshayeb, M., 2011. A classification of refactoring methods based on software quality attributes. *Arabian Journal for Science and Engineering*, *36*(7), pp.1253-1267.

[36] Singh, P. and Verma, S., 2012, May. Empirical investigation of fault prediction capability of object oriented metrics of open source software. In *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*(pp. 323-327). IEEE.

[37] Chowdhury, I. and Zulkernine, M., 2011. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture*, *57*(3), pp.294-313.

[38] Al Dallal, J. and Briand, L.C., 2010. An object-oriented high-level design-based class cohesion metric. *Information and software technology*, *52*(12), pp.1346-1361.

[39] Rathore, S.S. and Gupta, A., 2012, September. Investigating object-oriented design metrics to predict fault-proneness of software modules. In *Software Engineering (CONSEG), 2012 CSI Sixth International Conference on* (pp. 1-10). IEEE.

[40] Rathore, S.S. and Gupta, A., 2012, December. Validating the effectiveness of object-oriented metrics over multiple releases for predicting fault proneness. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific* (Vol. 1, pp. 350-355). IEEE.

[41] He, P., Li, B., Liu, X., Chen, J. and Ma, Y., 2015. An empirical study on software defect prediction with a simplified metric set. *Information and Software Technology*, *59*, pp.170-190.

[42] Graves, T.L., Karr, A.F., Marron, J.S. and Siy, H., 2000. Predicting fault incidence using software change history. *IEEE Transactions on software engineering*, *26*(7), pp.653-661.

[43] Munson, J.C., Nikora, A.P. and Sherif, J.S., 2006. Software faults: A quantifiable definition. *Advances in Engineering Software*, *37*(5), pp.327-333.

[44] Hassan, A.E., 2009, May. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering* (pp. 78-88). IEEE Computer Society.

[45] Bird, C., Bachmann, A., Aune, E., Duffy, J., Bernstein, A., Filkov, V., & Devanbu, P. (2009). *Fair and balanced?: bias in bug-fix datasets.* Paper presented at the Proceedings of the the 7th

joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering.

[46] Krishnan, S., Strasburg, C., Lutz, R.R. and Goševa-Popstojanova, K., 2011, September. Are change metrics good predictors for an evolving software product line?. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering* (p. 7). ACM.

[47] Devine, T.R., Goseva-Popstajanova, K., Krishnan, S., Lutz, R.R. and Li, J.J., 2012, April. An empirical study of pre-release software faults in an industrial product line. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on* (pp. 181-190). IEEE.

[48] Ihara, A., Kamei, Y., Monden, A., Ohira, M., Keung, J.W., Ubayashi, N. and Matsumoto, K.I., 2012, December. An Investigation on Software Bug-Fix Prediction for Open Source Software Projects--A Case Study on the Eclipse Project. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific* (Vol. 2, pp. 112-119). IEEE.

[49] Rahman, F. and Devanbu, P., 2013, May. How, and why, process metrics are better. In *Software Engineering (ICSE), 2013 35th International Conference on* (pp. 432-441). IEEE.

[50] Ma, Y., Luo, G., Zeng, X., & Chen, A. (2012). Transfer learning for cross-company software defect prediction. *Information and Software Technology, 54*(3), 248-256.Mahalanobis, P. C. (1936). On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta), 2*, 49-55.

[51] Wu, Y., Yang, Y., Zhao, Y., Lu, H., Zhou, Y. and Xu, B., 2014, June. The influence of developer quality on software fault-proneness prediction. In *Software Security and Reliability (SERE), 2014 Eighth International Conference on* (pp. 11-19). IEEE.

[52] Xia, H., Tan, W., Miladinovic, N. and Yang, S., LSI Corp, 2014. *Systems and methods for data detection using distance based tuning*. U.S. Patent 8,699,167.

[53] Zhang, H., 2009, September. An investigation of the relationships between lines of code and defects. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on* (pp. 274-283). IEEE.

[54] Rana, Z.A., Shamail, S. and Awais, M.M., 2009, May. Ineffectiveness of use of software science metrics as predictors of defects in object oriented software. In *Software Engineering, 2009. WCSE'09. WRI World Congress on* (Vol. 4, pp. 3-7). IEEE.

[55] Hata, H., Mizuno, O. and Kikuno, T., 2010. Fault-prone module detection using large-scale text features based on spam filtering. *Empirical Software Engineering*, *15*(2), pp.147-165.

[56] Zhang, W., Chen, J., Yang, Y., Tang, Y., Shang, J. and Shen, B., 2011. A practical comparison of de novo genome assembly software tools for next-generation sequencing technologies. *PloS one*, *6*(3), p.e17915.

[57] 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), pp.340-355.

[58] Rosenberg, J., 1997. Some misconceptions about LOC. In *Intl. Symp. on Software Metrics* (pp. 137-143).

[59] Olague, H.M., Etzkorn, L.H., Gholston, S., Quattlebaum, S.(2007): Empirical Validation of Three Software Metrics Suites to Predict FP of Object-Oriented Classes Developed Using Highly

Iterative or Agile Software Development Processes. IEEE Trans. Software Eng. No.33, pp.402—419.

[60] Zhou, Y., Xu, B. and Leung, H.(2010): On the ability of complexity metrics to predict fault-prone classes in object-oriented systems. The Journal of Systems and Software No. 83, pp. 660–674.

[61] Binkley, A.B. and Schach, S.R., 1998, April. Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures. In *Proceedings of the 20th international conference on Software engineering* (pp. 452-455). IEEE Computer Society.

[62] Moser, R., Pedrycz, W. and Succi, G., 2008, May. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th international conference on Software engineering* (pp. 181-190). ACM.

[63] Hall, T., Beecham, S., Bowes, D., Gray, D. and Counsell, S., 2012. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, *38*(6), pp.1276-1304.

[64] Rathore, S.S. and Kumar, S., 2017. A study on software fault prediction techniques. *Artificial Intelligence Review*, pp.1-73.