An Extensive study of Bug Prediction Approaches

***Abstract*—Software metrics has been used to describe the complexity of the program and, to estimate software development time. "How to predict the quality of software through software metrics, before it is being deployed" is a burning question, triggering the substantial research efforts to uncover an answer to this question. Knowing the locations of future software defects allows project managers to optimize the resources available for the maintenance of a software project by focusing on the most problematic components which can be done by Defect analysis using Predictor models. Defect data analysis is of two types; Classification and prediction that can be used to extract models describing significant defect data classes or to predict future defect trends. In this paper, various statistical and machine learning methods are applied on the metrics and defects are predicted.**

***Keywords*—Defect data analysis, Software metrics, Classification.**

I. Introduction

Defect prediction is comparatively a novel research area of software quality engineering. Defects can be defined in a disparate ways but are generally defined as aberration from specifications or ardent expectations which might lead to failures in procedure.Defect data analysis can help us for providing better understanding of the software defect data at large.

A software defect is an error, flaw, bug, mistake, failure, or fault in a computer program or system that may generate an inaccurate or unexpected outcome, or precludes the software from behaving as intended. A project team always aspires to procreate a quality software product with zero or little defects. High risk components within the software project should be caught as soon as possible, in order to enhance software quality. Software defects always incur cost in terms of quality and time. Moreover, identifying and rectifying defects is one of the most time consuming and expensive software processes. It is not practically possible to eliminate each and every defect but reducing the magnitude of defects and their adverse effect on the projects is achievable. Allocating quality assurance resources wisely is a risky task. If some nondefective module is tested for months and months, this is a waste of resources. If a defective module is not tested enough, a defect might escape into the field, causing a failure and potential subsequent damage. Therefore, identifying defect-prone modules is a crucial task for management.

A metric is defined as quantitative measure of the degree to which a system, component or process possesses a given attribute.Applied to software, a metric becomes a software metric. Software metrics play an essential part in understanding and controlling the overall software engineering process.

To construct a prediction model, we must have defect and measurement data collected from actual software development efforts to use as the learning set. In this paper, the dataset is a collection of models and metrics of five software systems and their histories.The five Software systems are namely Eclipse, Lucene. Pde, Equinox, Mylene. Each software system has different metrics namely bugs, wmc, fanout etc. We have presented machine learning techniques like Linear Regression, Negative Binomial Generalized Linear model and Tree regression on these datasets for the software defect prediction problem.

The overall design of this approach is described in Fig.[1]. The top metrics which are correlated with the bugs are found. The metrics which are highly correlated with the bugs are used as independent variables and "bugs" is used as dependent variables Then they are tested on three regression models based on the two parameters, Mean Absolute Error and p-value in Spearman correlation. The combination of those popularity metrics are also analysed if they would improve the prediction performance or not..Atlast, the results are analysed and discuss about the performance of the models.

We generate "popularity" metrics which are highly correlated with "bugs". Our hypothesis is that such metrics are an indicator of possible flaws in software components, thus being correlated with the number of defects. We aim at answering the following research questions:

1.What are the metrics with highest correlation to the number of post-release defects ("bugs") and How these metrics correlate with each other?
2. Is a regression model based on the popularity metrics a good predictor for software defects?
3. Does the addition of popularity metrics improve the prediction performance of existing defect prediction techniques?

The organization of the paper is as follows: Section 2 explains about the methodology of the design. Section 2A discusses about the metrics which are correlated with bugs or defects in each dataset.Section 2B discusses about the prediction performances of each of the popularity metrics with the bug using various models. Section 2C discusses about the prediction performances of the metrics when combining with each other and discuss about the performances of each of that models. Section 3 discusses about the results that are generated in the design. Section 4 concludes with the final discussion about the results.

II. Methodology

A. Generation of "Popularity" metrics

As mentioned earlier, the "Popularity" metrics are the metrics which are highly correlated with "bugs". To answer the research question Q1 .What are the metrics with highest

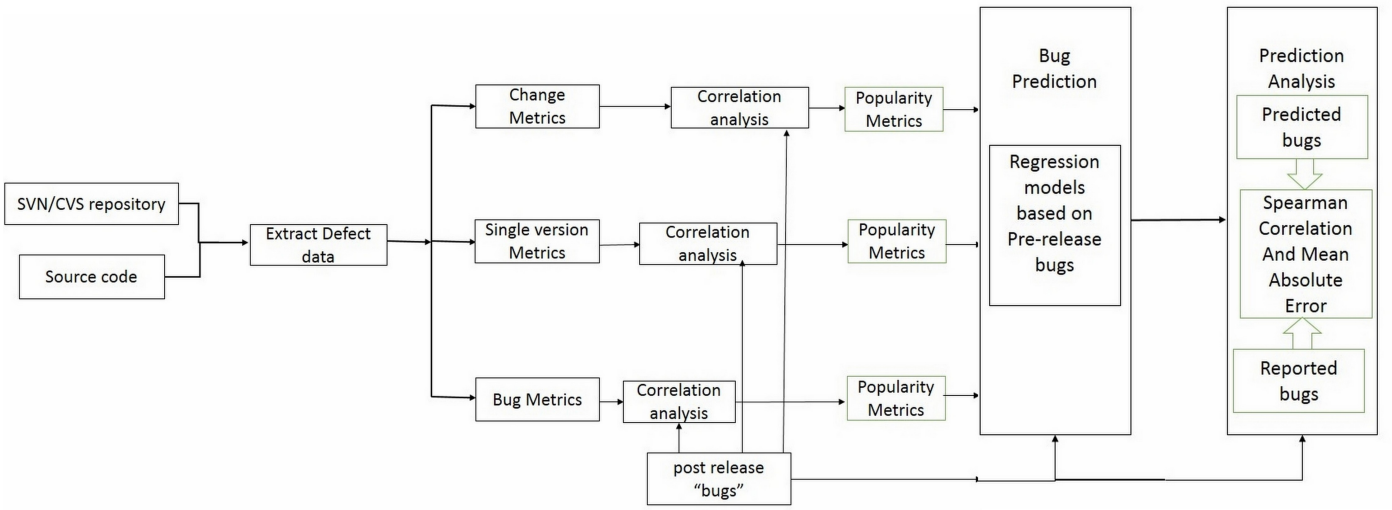

Fig. 1: Design in this paper

correlation to the number of post-release defects (”bugs”) and How these metrics correlate with each other?, we compute the top 5 metrics in ”single-version-ck-oo” and ”change-metrics”, top 2 metrics in ”bug-metrics” and find the correlation with the number of defects per class (or ”bugs”). These metrics are ”Popularity” metrics. We compute the correlation in terms of the Spearmans correlation coefficient. The Spearmans rank correlation test is a non-parametric test that uses ranks of sample data consisting of matched pairs. The correlation coefficient varies from 1, i.e., ranks are identical, to -1, i.e., ranks are the opposite, where 0 indicates no correlation. So for this research question, first the metrics as mentioned above are observed. The reason for this is since these metrics are highly correlated with ”bugs”, in the later sections we can use these metrics and the combinations as the independent variables. The correlation between the metrics is also found. In TABLE I, the correlation between the top metrics in Eclipse dataset are shown.

TABLE I: Correlation between top metrics in Eclipse

	noOfBugsFoundUntil.	no.OfNonTrivialBugsUntil.			
no.OfBugsFoundUntil.	1	***0.988247699***			
noOfNonTrivialBugsUntil.	0.988247699	1			
	linesAddedUntil.	numberOfVersionsUntil.	linesRemovedUntil.	maxLinesAddedUntil.	maxCodeChurnUntil.
linesAddedUntil.	1	0.925752963	***0.945540268***	***0.960879733***	0.797327713
numberOfVersionsUntil.	0.925752963	1	0.935454374	0.846205377	0.659263535
linesRemovedUntil.	0.945540268	0.935454374	1	0.89946031	0.62320185
maxLinesAddedUntil.	0.960879733	0.846205377	0.89946031	1	0.78674442
maxCodeChurnUntil.	0.797327713	0.659263535	0.62320185	0.78674442	1
	wmc	numberOfLinesOfCode	fanOut	rfc	cbo
wmc	1	***0.984106655***	0.809187176	***0.945245055***	0.776388774
numberOfLinesOfCode	0.984106655	1	0.830415002	***0.958082128***	0.786894148
fanOut	0.809187176	0.830415002	1	0.870887235	0.881132526
rfc	0.945245055	0.958082128	0.870887235	1	0.81413988
cbo	0.776388774	0.786894148	0.881132526	0.81413988	1

In the table, the correlation values between the metrics that are in 95% are shown in bold. The following inferences can be drawn from TABLE I.

(1) We can observe that the metrics having less correlation with ”bugs” (say ”cbo”) shows less correlation with all other metrics. This is evident because this metrics show less correlation with bugs and others show high correlation. (2) The metrics that are corresponding to the bold letters are later can be used as the independent vaiables in regression and are tested if they show high performance or not.

B. Analysis with single metrics as predictors

To answer the research question Q2 Is a regression model based on the popularity metrics a good predictor for software defects? , we create and evaluate regression models in which the independent variables are the class level popularity metrics, while the dependent variable is the number of post-release defects per class. We use single metrics as the predictors in this model and then compare the prediction performances of such models to answer research question Q3 Does the addition of popularity metrics improve the prediction performance of existing defect prediction techniques? The prediction performance of each of the top metrics are used on each of the following regression models: 1) Linear regression model 2) Tree regression model 3) Negative Binomial Generalized Linear model

We compute Spearmans correlation between the predicted number of postrelease defects and each of the top metric. Such evaluation approach has been broadly used to assess the predictive power of a number of predictors. In the cross-validation, for each random split, we use the training set (80% of the dataset) to build the regression model, and then we apply the obtained model on the validation set (20% of the dataset), producing for each class the predicted number of post-release defects. Then, to evaluate the performance of the performed prediction, we compute Spearmans correlation and the mean absolute error between the actual value and the predicted

TABLE II: Top metrics that are correlated with bugs in each software system

Eclipse

numberOfBugsFoundUntil.	0.449297	bug-metrics.csv
noOfNonTrivialBugsUntil.	0.442842	bug-metrics.csv
linesAddedUntil.	0.396619	change-metrics.csv
numberOfVersionsUntil.	0.39478	change-metrics.csv
linesRemovedUntil.	0.389293	change-metrics.csv
maxLinesAddedUntil.	0.357867	change-metrics.csv
maxCodeChurnUntil.	0.350057	change-metrics.csv
wmc	0.415327	single-version-ck-oo.csv
numberOfLinesOfCode	0.40898	single-version-ck-oo.csv
fanOut	0.393638	single-version-ck-oo.csv
rfc	0.389678	single-version-ck-oo.csv
cbo	0.374994	single-version-ck-oo.csv

Equinox

numberOfBugsFoundUntil.	0.545822	bug-metrics.csv
noOfNonTrivialBugsUntil.	0.542863	bug-metrics.csv
numberOfVersionsUntil.	0.57845	change-metrics.csv
linesAddedUntil.	0.570063	change-metrics.csv
linesRemovedUntil.	0.553085	change-metrics.csv
maxLinesRemovedUntil.	0.522504	change-metrics.csv
maxLinesAddedUntil.	0.520491	change-metrics.csv
cbo	0.540313	single-version-ck-oo.csv
lcom	0.540029	single-version-ck-oo.csv
wmc	0.537694	single-version-ck-oo.csv
numberOfMethods	0.536436	single-version-ck-oo.csv
numberOfLinesOfCode	0.521144	single-version-ck-oo.csv

Lucene

numberOfBugsFoundUntil.	0.335855	bug-metrics.csv
noOfNonTrivialBugsUntil.	0.335855	bug-metrics.csv
maxCodeChurnUntil.	0.27034	change-metrics.csv
linesAddedUntil.	0.263961	change-metrics.csv
numberOfVersionsUntil.	0.262111	change-metrics.csv
linesRemovedUntil.	0.261066	change-metrics.csv
maxLinesAddedUntil.	0.254417	change-metrics.csv
numberOfAttributes	0.182739	single-version-ck-oo.csv
cbo	0.180724	single-version-ck-oo.csv
lcom	0.175389	single-version-ck-oo.csv
numberOfMethods	0.175385	single-version-ck-oo.csv
fanIn	0.174705	single-version-ck-oo.csv

Mylene

noOfMajorBugsFoundUntil.	0.196314	bug-metrics.csv
noOfNonTrivialBugsUntil.	0.137329	bug-metrics.csv
linesAddedUntil.	0.150401	change-metrics.csv
maxLinesAddedUntil.	0.148624	change-metrics.csv
avgLinesAddedUntil.	0.148003	change-metrics.csv
codeChurnUntil.	0.14766	change-metrics.csv
maxCodeChurnUntil.	0.14401	change-metrics.csv
numberOfPrivateMethods	0.297007	single-version-ck-oo.csv
fanOut	0.266321	single-version-ck-oo.csv
cbo	0.24039	single-version-ck-oo.csv
numberOfLinesOfCode	0.230193	single-version-ck-oo.csv
rfc	0.229526	single-version-ck-oo.csv

Pde

noOfNonTrivialBugsUntil.	0.308351	bug-metrics.csv
numberOfBugsFoundUntil.	0.289567	bug-metrics.csv
linesAddedUntil.	0.272807	change-metrics.csv
numberOfVersionsUntil.	0.262673	change-metrics.csv
linesRemovedUntil.	0.257998	change-metrics.csv
maxLinesAddedUntil.	0.251103	change-metrics.csv
maxLinesRemovedUntil.	0.242397	change-metrics.csv
rfc	0.267993	single-version-ck-oo.csv
numberOfLinesOfCode	0.266392	single-version-ck-oo.csv
fanOut	0.24693	single-version-ck-oo.csv
wmc	0.236028	single-version-ck-oo.csv
numberOfPrivateMethods	0.228614	single-version-ck-oo.csv

value, on the validation set, between the lists of classes ranked according to the predicted and actual number of post-release defects. Since we perform 30 folds cross-validation, the final values of the Spearmans correlation averages over 30 folds. (4307 is set as the random seed in this paper).

C. Analysis with Combined metrics as predictors

To answer the research question Q3 " Does the addition of popularity metrics improve the prediction performance of existing defect prediction techniques?". For this question, the metrics are combined in different ways and some metrics are produced and tested against single metrics as in previous section. The metrics are calculated as below:

Suppose A1,A2 are top 2 metrics of single version metrics, B1,B2 are top 2 metrics of bug metrics and C1,C2 are top 2 metrics of change metrics. The 5 combined metrics are: A1+A2, B1+B2, C1+C2, A1+B1+C1, A1+A2+B1+B2+C1+C2.
Apart from these metrics the combination of the metrics that are shown in bold in Fig.[1] are also considered.

In the next section the various results are analysed and discussed.

III. Results and Discussion

Software metrics that do correlate with "bugs": The top metrics that are highly correlated with "bugs" are found out in all the 5 software systems. Two software systems ("Eclipse" and "Equinox") out of five show a strong rank correlation, i.e., coefficients ranging from .35 to.57, between defects of software components and metrics. In other three , i.e in Lucene they varies from .17 to .33, in mylene they vary from .14 to .29 and in pde from .22 to .3. We can observe that numberOfBugsFoundUntil and numberOfNonTrivialBugsFoundUntil, linesAddedUntil are always the top metrics in all the datasets.

TABLE III: Prediction performance of the metrics in mylene

Metrics	mae(for lm)	mae(for rpart)	mae(for glm)	rcor(for lm)	rcor(for rpart)	rcor(for glm)
nBFU	0.39653125	0.445452213	1.818311178	0.440750326	0.499786077	0.440750326
nnTBFU	0.396596533	0.446330761	1.821226363	0.43472379	0.489386067	0.43472379
lAU	0.475890109	0.461420019	1.682174727	0.384342939	0.464225146	0.384342939
no.OfVersUntil.	0.44056911	0.458554893	1.797575032	0.385451673	0.473801395	0.385451673
linesRemovedUntil.	0.487865511	0.470746165	1.654373436	0.381234797	0.437096913	0.381234797
maxLinesAddedUntil.	0.495937445	0.52116124	1.685535836	0.34998221	0.367123084	0.34998221
maxCodeChurnUntil.	0.516488978	0.527016174	1.634237511	0.33081871	0.337297893	0.33081871
wmc	0.454030465	0.453691931	1.811471157	0.401555004	0.478621199	0.401555004
noOfLinesOfCode	0.458230679	0.457013289	1.77579593	0.396121413	0.465333441	0.396121413
fanOut	0.451893741	0.487894435	1.821750086	0.385755417	0.440737368	0.385755417
rfc	0.453437945	0.460482978	1.758734503	0.378953146	0.486821295	0.378953146
cbo	0.467398879	0.494864367	1.716658647	0.361927195	0.425090105	0.361927195
nBFU + nnTBFU	0.398082814	0.444974364	1.822416663	0.433248491	0.433248491	0.424226366
linesAddedUntil.+no.OfVersUntil	0.432724209	0.466410284	1.804296461	0.38730932	0.38730932	0.378910235
wmc+numberOfLinesOfCode	0.456400519	0.455408523	1.823046285	0.39946535	0.39946535	0.400568029
nBFU+linesAddedUntil.+wmc	0.396760421	0.495820053	1.844000863	0.450631603	0.495820053	0.446361171
nBFU+nnTBFU+lAU+nOVU+wmc+nOLC	0.401089522	0.448590426	1.860298673	0.436581658	0.48570374	0.416938652
linesAddedUntil.+linesRemovedUntil.	0.470967647	0.463674762	1.678106151	0.326589912	0.453141961	0.366869357
linesAddedUntil.+maxLinesAddedUntil.	0.479918172	0.463302488	1.689803519	0.379262591	0.466534144	0.373741851
numberOfLinesOfCode+rfc	0.453130699	0.463570106	1.772093158	0.391280614	0.463902406	0.388803482
wmc+rfc	0.449016771	0.452180124	1.810341996	0.396646337	0.481796172	0.40060486

Importance of correlation among the top metrics : As mentioned in Section 2A and from Fig.[1] the correlation among the top metrics is useful in combining multiple metrics. This is tested by combining various metrics as shown in TABLE III. From this figure, we can observe that when we combine metrics with higher correlation with each other the Mean Absolute Error (mae) is decreased than the individual metrics as independent variables. So combining the metrics with higher correlation will result in higher prediction performance.

Prediction performance of single metric as independent variables in regression model: From TABLE III, as the correlation of the "bugs" with the metric decreases, the mean absolute error increases and spearmann correlation between the actual value and the predicted value also decreases. So it can concluded that we can predict software defects, but without major improvements over previously established techniques when the single metrics are used as the independent variables in regression model.

TABLE IV: Prediction performance of the metrics in lucene

Metrics	mae(for lm)	mae(for rpart)	mae(for glm)	rcor(for lm)	rcor(for rpart)	rcor(for glm)
numberOfBugsFoundUntil.	0.200876304	0.181296099	2.624164992	0.333814123	0.311231009	0.333814123
numberOfNonTrivialBugsFoundUntil.	0.200876304	0.181296099	2.624164992	0.333814123	0.311231009	0.333814123
maxCodeChurnUntil.	0.195448237	0.201421783	2.545685506	0.261112975	0.382191552	0.261112975
linesAddedUntil.	0.182506023	0.19705783	2.574761251	0.260095769	0.326768466	0.260095769
numberOfVersionsUntil.	0.209634761	0.198760748	2.563447247	0.263904561	0.377128688	0.263904561
linesRemovedUntil.	0.199630546	0.204384574	2.53511395	0.253675974	0.324361075	0.253675974
maxLinesAddedUntil.	0.21125718	0.213050718	2.488706217	0.24133283	0.293994321	0.24133283
numberOfAttributes	0.23805784	0.241592579	2.333595626	0.176945816	0.127052928	0.176945816
cbo	0.236190814	0.233054759	2.327173274	0.176992999	0.2237902	0.176992999
lcom	0.225402244	0.238896893	2.270461001	0.184936343	0.125683385	0.184936343
numberOfMethods	0.242222921	0.238896893	2.322215461	0.184932922	0.125683385	0.184932922
fanIn	0.243408772	0.243363011	2.264563242	0.175930927	0.09035038	0.175930927
nBFU+nNTBFU+mCCU+lAU+noOfAttr+cbo	0.196127152	0.179067906	2.166821063	0.306363021	0.374084761	0.277388158
nBFU+nNTBFU	0.200876304	0.181296099	2.624164992	0.333814123	0.311231009	0.333814123
maxCodeChurnUntil.+linesAddedUntil.	0.179798437	0.18725031	2.606372895	0.263282151	0.366275478	0.260201964
numberOfAttributes+cbo	0.237579303	0.22679628	2.398984076	0.223250825	0.214300782	0.221925787
nOBFU.+mCCU.+numberOfAttributes	0.20475097	0.176835359	2.66074266	0.324876636	0.378659039	0.295113028

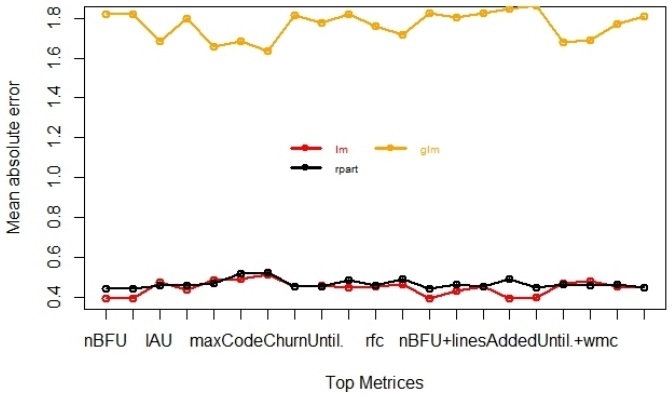

Fig. 2: Mean absolute error of metrics in Eclipse

Prediction performance of combining of metrics as independent variables in regression model: From TABLE III and TABLE IV, it is evident that on the basis of mean absolute error, the prediction performances of the Linear and Tree regression models are same on the average. But, Negative Binomial Generalized linear model has higher mean absolute error than the other 2 models but have higher rho value than other two. These observations are evident from Fig.[6] and Fig.[7]. We can also observe that combining the metrics always give better performance than the single metrics. So in the bug prediction model, Combining the metrics is always the best choice than using single metrics as predictors. From TABLE III and TABLE IV we may conclude that:
Linear Regression model is better for combining all the 6 metrics.
Tree Regression model is better for combining 3 metrics.

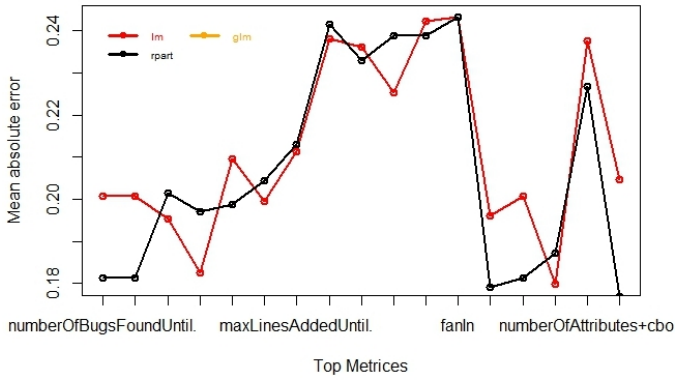

Fig. 3: Mean absolute error of metrics in mylene

IV. Conclusion and Future Work

This paper discusses about the Prediction Performance of the top metrics on the three regression models Linear regression model, Tree regression model, Negative Binomial generalized linear model. Developing a good classifier for predicting the number of bugs totally depends up on the independent variables chosen and the statistical tool we have employed to work with these variables. Future work may include developing a good classifier using Random Forests and analysing using less variables.

V. References

1. Bacchelli, Alberto, Marco DAmbros, and Michele Lanza. ”Are popular classes more defect prone?.” Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg, 2010. 59-73.
2 .D’Ambros, Marco, Michele Lanza, and Romain Robbes. ”An extensive comparison of bug prediction approaches.” Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on. IEEE, 2010.
3. T. Zimmermann, R. Premraj, and A. Zeller, Predicting defects for eclipse, in Proceedings of PROMISE 2007. IEEE CS, 2007, p. 76.