IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)
e-ISSN: 2278-1676,p-ISSN: 2320-3331, Volume 12, Issue 4 Ver. IV (July-Aug. 2017), PP 83-88
www.iosrjournals.org

Application of GA in designing Cost based Software Defect Prediction System

Mohini Prasad Mishra, Madhushree Kuanr

Abstract: *Research has yielded approaches to predict future defects in software artifacts based on historical information, thus assisting companies in effectively allocating limited development resources and developers in reviewing each other's' code reduces the cost of maintenance. Developers are unlikely to devote the same effort to inspect each software artifact predicted to contain defects, since the effort varies with the artifacts' size (the number of LOC and cost) of defects that it exhibits (effectiveness). We propose to use Genetic Algorithms (GAs) for training prediction models to maximize their cost-effectiveness. We evaluate the approach on two well-known models, Regression Tree and Generalized Linear Model, and predict defects between multiple releases of 5 open source projects. Our results show that regression models trained by GAs significantly outperform their conventional counterparts, improving the cost-effectiveness by up to 120 %.*
Key Word: *Machine Learning, Defect Prediction, Software Engineering, Statistical Methods, Expert Systems, Feature Selection, Regression Tree, Generalized Linear Model (GLM)*

I. Introduction

Statistical modeling has been often utilized in software engineering to assess quality of software projects. One of its frequent applications would be to produce prediction models to anticipate exactly where defects occur in a software system. Such models are actually valuable in several contexts: For instance, Kim et al. show the importance of theirs in effective API testing, where prediction models take the testing effectiveness in manufacturing environment [20]. Researchers and practitioners underline the importance of proper allocation of computing resources [11], for instance during code assessment [5],to save the appraisal cost of the code.

During initial research, researchers (e.g., [41]) had examined prediction models to make a binary distinction of each an application artifact: Likely or perhaps not very likely to incur in succeeding defects. Widely used evaluation metrics were precision as well as recall [41] or maybe the area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) curve. The AUC plots the classes properly classified as defective against individuals improperly classified as defective, as the prediction model 's discrimination threshold can vary.

Recently, scientists observed that the effort needed by designers towards inspecting artifacts recommended by binary distinction can vary based on the artifact [25]. Bigger and much more complicated software artifacts require more inspection effort, therefore hindering both the effectiveness and the usefulness of the prediction. As a solution, scientists have suggested to rethink prediction in conditions of cost effectiveness: Artifacts must be inspected in the order which maximizes the ratio between the amount of defects discovered as well as the effort spent (effort typically approximated by the dimensions of the artifacts) [11]. With this context, generally used evaluation metrics are : (i) the cost effective AUC (AUC CE) [11], that represented a weighted model of the more conventional AUC metric; and (ii) the P effort metric[11].In every effort aware prediction model presented very far| no matter the employed statistical mechanism the design is not immediately taught to get the best match to rank on the cost effectiveness, quite to foresee the raw selection of defects, i.e., an approximation of it. The concept we offer and assess in this paper is using Genetic algorithms (GAs) to immediately tweak the coefficients of a prediction version like that the cost effectiveness on the instruction set is actually maximized. Menzies et al. [28] had been the first person to suggest the thought of self-tuning the inner parameters of a principle learner to obtain the correct settings, therefore leading to a learner which considerably outperforms regular learning strategies [28]; here, we wish to immediately train statistical models. We utilize Gas to evolve the coefficients of regression algorithms to create an unit optimizing the cost effectiveness on the instruction set, under the presumption that it'll also predict cost effectiveness more efficiently on the examination set. We evaluate the idea of ours by applying it and doing an empirical analysis on a selection of unique application systems and releases. As the baseline of ours, we think about extensive statistical regression models (i.e., generalized linear regression version (GLM) as well as regression trees (RT)) as well as metrics (i.e., Kemerer and Chidamber (CK) metrics as well as Lines of Code (LOC)). Our results indicate that our strategy significantly outperforms traditional models.

In earlier endeavors, researchers (e.g., [41]) had studied prediction models to present a binary classification of each software program artifact. Commonly used evaluation metrics were precision in addition to recall [41] or the area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) curve. The

AUC plots the classes correctly classified as defective against those wrongly classified as defective, as the prediction model 's discrimination threshold may differ.Not too long ago, researchers pointed out that the effort expected by developers towards inspecting artifacts indicated by binary classification models differs based on the artifact [25]. Larger and a lot more complex software artifacts require extra inspection effort, thereby hindering both the usefulness and the effectiveness of the prediction. As a solution, researchers have recommended to rethink prediction in terms of cost-effectiveness: Artifacts have to be inspected in the order which often maximizes the ratio between the number of defects detected together with the effort spent (effort often approximated by the size of the artifacts) [11]. In this particular context, often used evaluation metrics are: (i) the cost-effective AUC (AUC-CE) [11], which in turn represents a weighted model of the more classic AUC metric; and (ii) the P effort metric [11].

In every effort aware prediction model presented very far| no matter the employed statistical mechanism the design is not immediately taught to get the best match to rank on the cost effectiveness, quite to foresee the raw selection of defects, i.e., an approximation of it. The concept we offer and assess in this paper is using Genetic Algorithms (GAs) to immediately tweak the coefficients of a prediction version like that the cost effectiveness on the instruction set is actually maximized. Menzies et al. [28] had been the first person to suggest the thought of self-tuning the inner parameters of a principle learner to obtain the correct settings, therefore leading to a learner which considerably outperforms regular learning strategies [28]; here, we wish to immediately train statistical models. We utilize GAs to evolve the coefficients of regression algorithms to create an unit optimizing the cost effectiveness on the instruction set, under the presumption that it'll also predict cost effectiveness more efficiently on the examination set. We evaluate the idea of ours by applying it and doing an empirical analysis on a selection of unique application systems and releases. As the baseline of ours, we think about extensive statistical regression (i.e., generalized linear regression version (GLM) as well as regression trees (RT)) as well as metrics (i.e., Kemerer and Chidamber (CK) metrics as well as Lines of Code (LOC)). Our results indicate that our strategy significantly outperforms conventional models.

II. Existing Work

Researchers devised a number of defect prediction approaches to guide software maintenance as well as evolution by identifying more defect software artifacts [11]. These approaches are based on statistical models, whose main difference is actually the various sets of predicting metrics and also the underlying algorithms that learn from these metrics and make predictions [15,37]. Examples of metrics are the Chidamber and Kemerer's object oriented (CK) metrics [10,6], structural metrics [3] or process metrics [29]. Examples of algorithms are logistic regression used by Zimmermann et al. [40]; Multi-Layer Perceptron (MLP),radial foundation functionality (RBF), k nearest neighbor (KNN), regression tree (RT), dynamic evolving neuro-fuzzy inference body (DENFIS), and Support Vector Regression (SVR)used by Elish [fourteen]; Bayesian networks utilized by Bechta [31];and Naive Bayes, J48, Alternative Decision Tree (ADTree),and One R believed by Nelson et al. [30]. Lately, various other researchers have suggested more advanced machine learning strategies, including ensemble learning [23], clustering algorithms [36], and combined techniques [thirty two]. Lessman et al.[22]evaluated twenty two classification models and showed that there's simply no statistical difference between the top-17 models when classifying software modules as defect susceptible. Meta-heuristics have been additionally investigated, such as using genetic algorithms (GAs) [13,19,23] or maybe genetic programming (GP) [1] to build prediction design aimed at optimizing standard performance metrics for classification problems, such as precision, recall, and f-measure.

Mende et al. [25], Kamei et al. [18], Menzies et al. [28], and D 'Ambros et al. [11] are actually among the first to warn of the significance of taking into consideration the effort necessary to review the documents recommended by prediction models. Traditional performance metrics employed for binary predictions (precision, recollection, f-measure, AUC [32], error variance, median error, error sum, and correlation [11]) aren't well suited to assessprediction since they provide exactly the same priority/importance to all defect prone software components. Rather, in useful scenario engineers would gain from determining those software components apt to contain far more defects earlier, or perhaps requiring lower inspection price at the exact same number of defects. Consequently, prediction techniques must be cost e ective, the place that the effectiveness is actually number of defects to foresee as well as the assessment cost is actually approximated by the collections of code (LOC) metric, depending on the intuition that bigger files call for effort and time more to discuss than smaller files [11,25,18].Previous labor suggested performance metrics (e.g., AUC CE and Peffort) created for evaluating the cost effectiveness of prediction models [11, 34,17,32, 33,28]. Nevertheless, the designs were still made working with conventional knowledge algorithms; for instance, D 'Ambros et al. [11] trained conventional linear regression models with the classical iteratively re weighted very least square algorithm; Rahman and Devanbu [33] used 4 different machine learning strategies (i.e., SVM, J48, Logistic Regression, and Naive Bayes) that had been used in utilizing the corresponding classical instruction algorithms.

III. Generalized Linear Model(GLM) Regression

A Generalized Linear Model (GLM)is a model ,is basically a regression model with relaxed rules, consists of three components: (I)a link function (II) a variance function(μ) and(III) Independent Variables . The link relates the means of the observations to predictors: linearization (μ) The variance function relates the means to the variances. In our case the (i) independent variables I = {I1,I2….Ik }are the software metrics used as explanatory variables of the scalar dependent variable Y , i.e., the number of defects. The (ii) linear function condenses the independent variable into a scalar value with a set of linear coefficients β ={α,β1,β2…βk} such that

$\eta=\alpha+þ1\times I1 \ +þ2 \ \times I2 + \cdots + þk\times Ik$ (1)

The link function ꜰ that provide the relationship between the expectation of the outcome and the linear function.

$ꜰ=\alpha+þ1\times I1 \ +þ2 \ \times I2 + \cdots + þk\times Ik$ (2)

If the above function represents the variables from normal distribution then the equation corresponds to traditional multinomial regression. The General Linear Model (GLM) ,in Eq (2) find the set of coefficients β ={α,þ1, þ2, … . . þk} such that the corresponding model ꜰ minimizes
the Mean Squared Error (MSE) between predicted value and actual outcome.

3.1 Regression Tree

A regression tree is similar to a decision tree where internal decision nodes contains the decision rules on software metrics while leaf nodes are defect prediction outcomes. The decision rules are based on the software metric Mi and coefficient xi verifies the test condition is reached or not , thus by partitioning the space in to two branches (true or false)

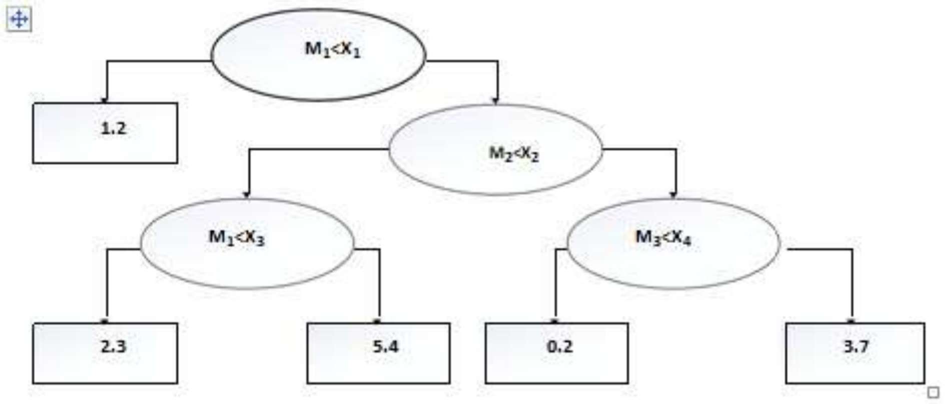


Figure 1 Regression Tree

IV. Proposed Approach

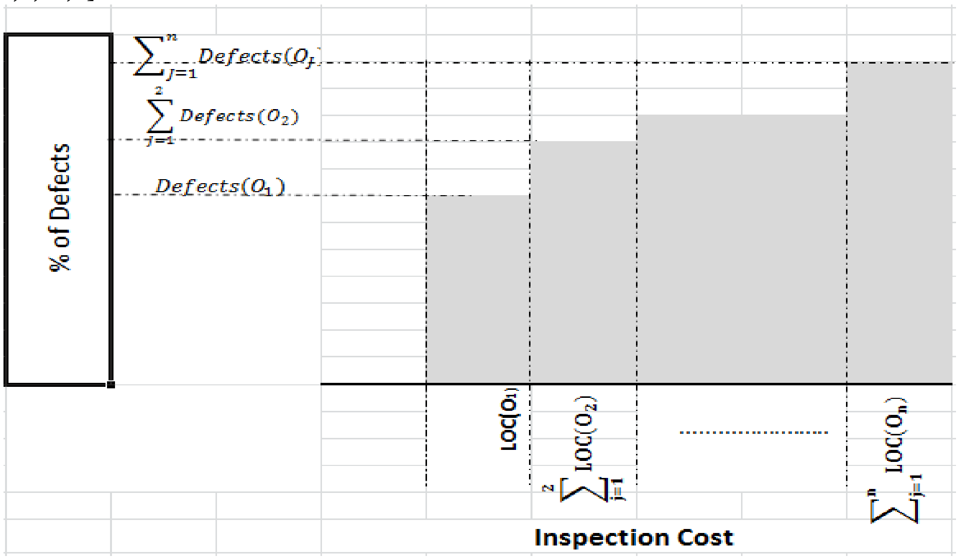
We propose that if the aim of the prediction model is cost effectiveness and evaluated differently with respect to traditional models then the models trained using deferent approach would show better results. Hence we propose to modify model training such that software engineering artifacts with higher defect density will be given higher priority.

In this regard, we wish to maximize the Mean Squared Error (MSE), which is ration between cumulative number of defects and amount of code under inspection. Let A={a1a2….ak} be the list of artifacts to be reviewed in the training set ordered by their predicted scores computed by the regression equation from Eq (2) where β= {þ1þ2….þk} is a set of regression coefficients. We hypothesize the defect prediction as:

To find the set of regression coefficients β = {þ1, þ2….þk} from regression equation (2)
such that coefficients maximizes the cost-effectiveness i.e maximizing the cumulative number
of defects while inspecting the Lines of Code for the predicted artifacts in the order O={o1o2….ok}
φmax(ꜰ) = ∑n { ∑i–1 {defects Oj × LOC(Oi)}} Eq(3)
i=2 j=1 Eq(3)

Where ∑i–1 actual Oj

represents cumulative defects for the first(i-1) artifacts.
FinallyLOC (Oi) measures the lines of code(LOC) of ith artifact as shown in following figure (Fig (2)).[25,4,11,8,32,9]



% of Defects
$\sum_{j=1}^{n} Defects(O_j)$
$\sum_{j=1}^{2} Defects(O_2)$
$Defects(O_1)$
$LOC(O_1)$
$\sum_{i=1}^{2} LOC(O_2)$
$\sum_{j=1}^{n} LOC(O_n)$
Inspection Cost

Figure 2 Graphical representation of proposed approach

The function in Equation (3) represents the AUC-CE using the rectangle rule The abovementioned figure (Fig (2)) represents the cost effectiveness of software artifacts from the regression model in the equation 2.In the above figure Y-axis plots the cumulative defects when analyzing the first i artifacts whose cumulative effort (cost) is on x-axis

Training the Model with GA

To solve the maximization problem we apply nature inspired GA on the Regression Trees and GLM. The search space represents possible sets of linear combination of coefficients β =
{α,þ1…..þn} for GLM and X= {x1, x2……xn} Decision coefficients of RT.

The algorithm starts with a randomly generated GLM or RT configurations. During the subsequent iterations, the population is evolved with genetic operators such as (i) selection (ii) Crossover and (iii) Mutation . During each iteration, the coefficients are evaluated according the function represented in equation (2). The best or fittest coefficients are selected using selection operator. During the same time new set of coefficients are generated by re- combination technique using Crossover and Mutation operators. After each generation finally selected coefficients are used as starting point for next selection or iteration cycle.

V. Experimental Result

In order to evaluate the cost – effectiveness in our approach compared to traditional approach we present the design of study as follows.

Data from open source, PROMISE repository, projects based on Java are selected for our study and details are mentioned on the following table. (ref Tab (1))

Table 1 Projects used for our study

Dataset	Release	# Classes	% Defective Classes	Average No.Of Defects
Velocity	1.4	197	74.62	1.4
	1.5	215	66.05	2.3
	1.6	237	33.91	2.4
Xlan	2.4	724	15.2	1.4
	2.4	804	48.1	1.3
	2.6	886	46.4	1.5
	2.7	910	98.7	1.4
Synapse	1.0	157	10.2	1.31
	1.1	222	27.0	1.65
	1.2	256	33.6	1.67
Lucene	2.0	196	46.4	2.9
	2,2	248	58.1	2.8
	2.4	341	59.53	3.1
Poi	1.0	237	59.5	2.4
	1.5	315	11.8	1.1
	2.0	387	64.4	2
	2.5	443	65.6	1.8

We have selected the datasets which are having multiple releases to prove our approach. For each project we have considered LOC and CK metrics for as they are reliable and evaluated for cost –effectiveness in our approach. We have selected CK metrics as they are widely used for quality assessment of Object Oriented projects.

The following tabulated shows the result of P effort scored by traditional Rregression Tree (RT) and Ga based RT. The Peffort achieved over 30 runs as well as corresponding standard deviation. From the results it is very clear that the GA based regression models significantly outperform the their counterparts for predicting P effort .

Table 2 The average Peffot by RT model and GA trained RT model

Dataset	**Training**	**Test**	**RT**		**RT-GA**		**Improvement**
			Mean	St Dev	Mean	St Dev	
Velocity	1.4	1.5	0.692		0.87	0.01	26%
	1.5	1.6	0.45		0.81	0.03	80%
Xlan	2.4	2.5	0.43		0.73	0.09	70%
	2.5	2.6	0.55		0.81	0.006	47%
	2.6	2.7	0.43		0.91	0.03	112%
Synapse	1.0	1.1	0.54		0.57	0.003	6%
	1.1	1.2	0.64		0.68	0.035	6%
Lucene	2.0	2.2	0.64		0.8	0.017	25%
	2,2	2.4	0.52		0.7	0.037	35%
Poi	1.0	1.5	0.39		0.53	0.024	36%
	2.0	2.5	0.46		0.63	0.025	37%

VI. Conclusion

In this paper we have hypothesized that the defect prediction may not reach its entitlement as they are trained on a task that is totally deferent from defect prediction cost-effectiveness. Current statistical based models are used to find the best fit to predict the artifacts having defects while our approach is to rank the artifacts based on defect density for most cost-effectiveness.The proposed GA based approach is designed overcome the limitations of traditional methods so that artifacts exhibit more defects are given higher priority. Results from the Table (2) clearly suggest that the GA based approach is useful.

Future research need to investigate if our approach can be extended to other models such as Naïve Bayesian and Artificial Neural Networks as well as to other software metrics other than CK metrics. Further research is also required to study the influence of project size on the same.

References

[1]. W. Afzal and R. Torkar. On the application of genetic programming for software engineering predictive modeling: A systematic review. Expert Systems with Applications, 38(9):11984 – 11997, 2011.

[2]. Arcuri and L. Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In Software Engineering (ICSE), 2011 33rd International Conference on, pages 1–10, May 2011.

[3]. E. Arisholm and L. C. Briand. Predicting fault-prone components in a java legacy system. In Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, ISESE ’06, pages 8–17. ACM, 2006.

[4]. E. Arisholm, L. C. Briand, and E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. J. Syst. Softw., 83(1):2–17, January 2010.

[5]. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In Proceedings of the 35th International Conference on Software Engineering, pages 712–721, 2013.

[6]. V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. IEEE Trans. Software Eng., 22(10):751–761, 1996.

[7]. N. Bettenburg, M. Nagappan, and A. E. Hassan. Towards improving statistical modeling of software engineering data: think locally, act globally! Empirical Software Engineering, 20(2):294–335, 2015.

[8]. G. Canfora, A. De Lucia, M. Di Penta, R. Oliveto,Panichella, and S. Panichella. Multi- objective cross-project defect prediction. In The 6th International Conference on Software Testing, Verification and Validation, pages 252–261, 2013

[9]. G. Canfora, A. D. Lucia, M. D. Penta, R. Oliveto,Panichella, and S. Panichella. Defect prediction as a multi-objective optimization problem. Software Testing, Verification and Reliability, 25(4):426–459, June 2015.

[10]. S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. IEEE Trans. Software Eng., 20(6):476–493, June 1994.

[11]. M. D'Ambros, M. Lanza, and R. Robbes. Evaluating defect prediction approaches: A benchmark and an extensive comparison. Empirical Software Engineering, 17(4-5):531– 577, 2012.

[12]. Cost Sensitive Software Defect Prediction Technique Using Genetic Algorithm

[13]. D. Deb and K. Deb. Investigation of mutation schemes in real-parameter genetic algorithms. In Swarm, Evolutionary, and Memetic Computing, volume 7677 of Lecture Notes in Computer Science, pages 1–8. Springer Berlin Heidelberg, 2012.

[14]. S. Di Martino, F. Ferrucci, C. Gravino, and F. Sarro. A genetic algorithm to configure support vector machines for predicting fault-prone components. In Product-Focused Software Process Improvement, volume 6759 of Lecture Notes in Computer Science, pages 247–261. Springer Berlin Heidelberg, 2011.

[15]. M. Elish. A comparative study of fault density prediction in aspect-oriented systems using MLP, RBF, KNN, RT, DENFIS and SVR models. Artificial Intelligence Review, 42(4):695–703, 2014.

[16]. T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. IEEE Trans. Software Eng., 38(6):1276–1304, Nov 2012.

[17]. F. Herrera, M. Lozano, and A. M. Sa´nchez. A taxonomy for the crossover operator for real- coded genetic algorithms: An experimental study. International Journal of Intelligent Systems, 18(3):309–338, 2003.

[18]. T. Jiang, L. Tan, and S. Kim. Personalized defect prediction. In IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), pages 279–289, Nov 2013.

[19]. Y. Kamei, S. Matsumoto, A. Monden, K.-i. Matsumoto, B. Adams, and A. Hassan. Revisiting common bug prediction findings using effort-aware models. In Software Maintenance (ICSM), 2010 IEEE International Conference on, pages 1–10, 2010.

[20]. T. Khoshgoftaar, N. Seliya, and Y. Liu. Genetic programming-based decision trees for software quality classification. In Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence, pages 374–383, Nov 2003.

[21]. M. Kim, J. Nam, J. Yeon, S. Choi, and S. Kim. REMI: Defect prediction for efficient api testing. In Proceedings of ESEC/FSE, pages 990–993, 2015 (8)

[22]. S. Kpodjedo, F. Ricca, P. Galinier, Y. Gu´eh´eneuc, andG. Antoniol. Design evolution metrics for defect prediction in object oriented systems. Empirical Software Engineering, 16(1):141–175, 2011.

[23]. S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Trans. Software Eng., 34(4):485–496, 2008.

[24]. Y. Liu, T. M. Khoshgoftaar, and N. Seliya. Evolutionary optimization of software quality modeling with multiple repositories. IEEE Trans. Software Eng., 36(6):852–864, Nov. 2010.

[25]. T. Mende and R. Koschke. Revisiting the evaluation of defect prediction models. In Proceedings of the 5th International Conference on Predictor Models in Software Engineering, page 7. ACM, 2009.

[26]. T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. IEEE Trans. Software Eng., 33(1):2–13, Jan 2007.

[27]. T. Menzies, R. Krishna, and D. Pryor. The promise repository of empirical software engineering data, 2015.

[28]. T. Menzies, Z. Milton, B. Turhan, B. Cukic, and Y. J. A. Bener. Defect prediction from static code features: current results, limitations, new approaches.Automated Software Engineering, 17:375–407, 2010.

[29]. R. Moser, W. Pedrycz, and G. Succi. 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, pages 181–190. ACM, 2008.

[30]. A. Nelson, T. Menzies, and G. Gay. Sharing experiments using open-source software. Software: Practice and Experience, 41(3):283–305, 2011.