2016 IEEE International Conference on Software Quality, Reliability and Security

Empirical Evaluation of Cross-Release Effort-Aware Defect Prediction Models

Kwabena Ebo Bennin
Department of Computer Science
City University of
Hong Kong, China
kebennin2-c@my.cityu.edu.hk

Koji Toda
Department of Computer Science
Fukuoka Institute of Technology
Japan
toda@fit.ac.jp

Yasutaka Kamei
Graduate School and Faculty of Information
Science and Electrical Engineering
Kyushu University, Japan
kamei@ait.kyushu-u.ac.jp

Jacky Keung
Department of Computer Science
City University of
Hong Kong, China
Jacky.Keung@cityu.edu.hk

Akito Monden
Department of Computer Science
Okayama University
Okayama, Japan
monden@okayama-u.ac.jp

Naoyasu Ubayashi
Graduate School and Faculty of Information
Science and Electrical Engineering
Kyushu University, Japan
ubayashi@ait.kyushu-u.ac.jp

***Abstract*—To prioritize quality assurance efforts, various fault prediction models have been proposed. However, the best performing fault prediction model is unknown due to three major drawbacks: (1) comparison of few fault prediction models considering small number of data sets, (2) use of evaluation measures that ignore testing efforts and (3) use of n-fold cross-validation instead of the more practical cross-release validation. To address these concerns, we conducted cross-release evaluation of 11 fault density prediction models using data sets collected from 2 releases of 25 open source software projects with an effort-aware performance measure known as Norm(P_{opt}). Our result shows that, whilst M5 and K* had the best performances; they were greatly influenced by the percentage of faulty modules present and size of data set. Using Norm(P_{opt}) produced an overall average performance of more than 50% across all the selected models clearly indicating the importance of considering testing efforts in building fault-prone prediction models.**

Key words : fault-density estimation, empirical study, open source project, cross version prediction, Demšar's significance diagram

I. Introduction

In software testing and maintenance process, prediction of fault-prone modules (having high probability of containing a fault) is helpful in determining the allocation and priorities of review/test effort [1]. Fault prediction models predict the presence or absence of a fault in a module (single source code file) using metrics such as number of source code lines, cyclomatic complexity number and others, as explanatory variables [2], [3].

In addition, prediction of the number of faults [4], [5] and the fault density [6] have also been studied. However, in conventional studies, there are problems with the experimental settings for prediction model evaluation. Therefore, the stable or conclusive ranking of fault prediction models has not been firmly established yet. Specifically, these are the following three major problems:

- Small number of predictive models and small sample size of data sets are used for performance comparison and evaluation. As a result, the best prediction model varies among different research papers as observed by Myrtveit et al. [7].
- Cost-effectiveness (hereby refered to as testing effort in this paper) of applying fault-prone prediction models during evaluation and comparison of their prediction performance is ignored in most research studies as pointed out by Arisholm et al. [8]. Several previous studies predict the presence or absence of faults in a module, and evaluate the prediction performance by precision/recall/F-measure/ROC-Curve and others [9], [10], [8]. However, most of these performance measures ignores effort needed to fix faults, that is, they do not distinguish between a predicted fault in a small module and a predicted fault in a large module. From the viewpoint of finding more bugs with less testing effort (cost of additional quality assurance such as verification cost which includes code inspections, coverage testing and others) on selected faulty modules, it would be better to predict high fault-density modules, which contain many faults in a few lines of code and to evaluate the prediction performance using effort-aware measures [6].
- N-fold cross-validation has been commonly used in previous studies for evaluating prediction models. It derives both fit and test data set from a single data set. However, this is a quite unrealistic situation in actual software development environment. In the software development environment, it is assumed that a model is constructed using the data of the developed software in the past (old version), to predict the fault of a developing software (the current version) and therefore evaluation of predictive model that span such version is required [11].

Based on these current issues, we address the following two Research Questions (RQ) using a data set from 25 open-source projects.

RQ1. Which is the best fault prediction model when testing effort is considered?

For RQ1, we experimentally demonstrate the superiority

978-1-5090-4127-5/16 $31.00 © 2016 IEEE
DOI 10.1109/QRS.2016.33

or inferiority of prediction performance of 11 kinds of fault density prediction models. In this experiment, we use a rating scale that takes into account the testing effort required after prediction. The experimental results shows that K* outperformed the other 10 prediction models although it was not statistically significant in comparison to the other prediction models.

RQ2. Which is the best fault prediction model when assuming cross-release prediction?

For RQ2, considering the use of a real past project, a model is constructed using data of the developed software (old version) to predict the fault density of each module in the next version. The proposed approach is a more reliable approach in the software development environment as noted by Kamei et al. [12]. We assess how the choice of using a previous release of a software for training a predictive model affects the accuracy of the predictive models based on the evaluation criteria which considers the testing effort. The M5 tree model performed better in comparison to the other prediction models.

The remainder of this paper is organized as follows. Background of research questions and the eleven predictive models are discussed in section II. Section III presents the design of the experiment and Section IV presents and discusses the results. Section V presents the threats to validity and Section VI gives an overview of related works. Finally, we conclude in Section VII.

II. Research Questions

We address the research questions for the purpose of 11 predictive models in a real project environment.

A. Which is the best fault prediction model when testing effort is considered?

Kamei et al. [12] proposed a fault prediction model which is evaluated considering the testing effort. Kamei et al's experiment considers the number of source code lines as testing efforts and used fault density (number of fault/source code number of lines) as the target variable. With the use of prediction performance, developers can observe that for high fault density, it is possible to prioritize and allocate testing effort for the modules with large fault number per unit lines of code thus, creating an efficient test plan.

However, comparison of prediction performances of fault density prediction models using large number of data sets have not been well implemented in several previous studies. Therefore, to compare the types of fault density prediction models considering the number of source code lines in RQ1, 11 techniques were chosen to cover six categories of prediction models. We conduct an empirical experiment to reveal the superiority of the various prediction models by analysing their performances when testing effort is considered. Overview and name of each defect prediction model used and their hyper parameter values are shown in Table I.

B. Which is the best fault prediction model when assuming cross-release prediction?

For a developer to use the fault density prediction model in real projects, the model should be built using data of the past developed software (the last version) as pointed out by Radjenovic´ et al. [11]. However, in conventional studies, cross-validation methods have been widely evaluated on the same version. Therefore, in RQ2 we build a fault density prediction model by using the previous version of the project or data, to evaluate the prediction performance when applied to the next version. Furthermore, upon examining and verifying the differences and correlation between the results of RQ1 and RQ2, developers could select the best fault density prediction model to use across versions.

III. Experiment Description

In this section, the experiment setting is described. Target projects and measurement metrics used are discussed.

Data is collected from 25 Open Source Software (OSS) projects and fault density (number of faults divided by number of lines of code) is predicted using 11 different approaches and finally compared considering the effort-aware measure Norm(P_{opt}). We describe the preprocessing activities performed on the data sets, the hyper parameters configuration, modelling tool used for the execution of this experiment and evaluation criteria used to verify the RQs.

A. Target Project

The records of each metric value was collected from 25 OSS project data sets used for the experiment. The number of project modules and faults are shown in Table II. Buggy modules were labelled in the commit logs as "error" and thus were used to compute the error density. Bugs which were fixed

Table I: Prediction methods and hyper parameters in our experiments

Category	Model	Overview	Hyper parameters
Statistical approach	LM	Linear regression	No
	LARS	Least Angle Regression [14]	No
	RVM	Relevance Vector Machine [15]	No
Neighborhood Law	K-NN	k neighborhood method [10]	k={1,. . . . . , 15}
	K*	k neighborhood method using distance based on entropy [16]	No
Neural Net	NNET	3-layer neural net [4]	size = {4,..,28}, decay = {0.1,0.2}
Support Vector Machine	SVM	support vector machine [17]	C = {2^-6,. . . .,2^10}
The tree model	M5	classification tree [18]	No
	RPART	CART [19]	cp= {0.05, 0.1, ..., 0.7}
Collective Learning	GBM	classification using boosting trees [20]	n.tree= { 10,50, 100, 200 }
	RF	Random Forest [21]	mtry= { 10, 50, 100, 200, 250, 500, 1000}

Table II: The number of modules and their respective faults of two versions' projects used in experiments

Project Name	Short Name	Version A		Version B	
		Module	Fault	Module	Fault
Clam Antivirus	CLAM	1597	93	8723	56
eCos	ECOS	630	110	3480	67
Exim Internet Mailer	EXIM	184	28	61	31
freedesktop	FREE	1982	477	2855	603
Ganymede	GANY	184	29	95	30
Helma	HELM	312	38	100	17
Hibernate Core	HIBE	406	87	5032	1191
HylaFAX	HYLA	137	70	109	35
iptables	IPTA	151	96	276	91
Koffice	KOFF	930	363	990	232
NetBSD	NBSD	6781	548	10960	299
OpenBSD	OBSD	1706	275	5964	158
OpenCms	OCMS	1727	193	2821	93
openNMS	ONMS	1203	123	1416	112
OTRS	OTRS	420	111	763	169
PostgreSQL	POST	863	435	1211	325
ProFTPD	PROF	195	64	220	61
Samba	SAMB	1623	543	2559	439
Scilab	SCIL	2636	239	1248	244
Spring Security	SPRI	926	126	639	132
squid	SQUI	250	59	185	92
Wine	WINE	1627	548	1382	288
Xdoclet	XDOC	105	30	135	5
XFree86	XFRE	713	350	761	259
XORP	XORP	1696	158	1755	217

Table III: Measurement Metrics

Name	Overview
Codechurn	Number of changed rows (number of additional lines + number of deleted rows)
LOCAdded	Additional number of rows (Lines of codes)
LOCDeleted	Deleted rows (Lines of codes)
Revisions	Revision number
Age	Last number of days since it was fixed
BugFixes	Number of corrected faults
Refactorings	Number of refactoring

in the next release were also labelled as "Bugfixes". For all the data set, version B is a newer version of A.

B. Measurement Metrics

One module is regarded as one source code file. For this experiment, we collected seven typical process metrics as proposed by Moser et al [22] for a module unit. We used the commit logs from collection of version control tools (such as CVS, Git, etc.) as the target repository. Names of each metric and overview is shown in Table III. In addition, we considered the specific words in the commit log measured in module units ("Fix" and "Refactoring") as BugFixes and Refactorings.

C. Experimental data set

For RQ1, the fit data is randomly selected using 2/3 of the modules from the data set Version A and the remaining 1/3 as the test data as shown in Table II

For RQ2, the model was built using the data set version A as the fit data and predicted using data set version B as test data as shown in Table II.

D. Pre-processing of Data

Because refactoring in several projects were not performed, the value of refactorings for these projects were zero (0). Hence, metric values of refactorings which were zero were excluded in advance from the model building to prevent any adverse effect on the model accuracy. In addition, multi-collinearity tends to decrease the reliability of the model and as such, metrics causing multicollinearity were also excluded from the model building. The Codechurn and LOCAdded metrics were excluded since they had high correlation values above 0.9.

E. Estimation of hyper parameters

During prediction model building, it is necessary to set the parameters. Among the parameters that affect the entire model are the hyper parameters which has a significant impact on classification accuracy. Therefore, the optimal hyper parameter values were estimated using a 10-fold cross-validation method in this experiment. Candidate values of hyper parameters and the estimation of each method is shown in Table I. However, the statistical approach is labeled "No" because they have no hyper parameters.

F. Model Evaluation criteria

For the experiment evaluation, we evaluated the performance of the prediction models based on testing efforts as proposed and used by Kamei et al [12]. They used the modification of P_{opt} evaluation metrics [23] called normalized P_{opt} due to its ability to evaluate the performance considering the testing effort. To compute P_{opt} , we first find the difference between the LOC-based cumulative lift charts of the optimal model and the prediction model represented as $\triangle_{opt}$ (Figure 1). By intuition, a larger module will require more time to review and fix compared to a small module. As such, smaller modules would always be prioritized if the predicted faults are the same for both modules. P_{opt} is thus computed as P_{opt} = 1-$\triangle_{opt}$. A larger P_{opt} value implies a small difference between the optimal and predicted model and hence the best performing model. P_{opt} are dependent on the number of faults for each data set thus, normalising the P_{opt} values for the prediction model is necessary since this will make comparison easier. The normalized P_{opt} is defined as

$$Norm(P_{opt}) = \frac{P_{opt} - min(P_{opt})}{max(P_{opt}) - min(P_{opt})} \quad (1)$$

where max(P_{opt}) and min(P_{opt}) are calculated on the LOC-based cumulative lift chart in which all modules are ordered by fault density. Clearly, max(P_{opt}) will always be 1 because $\triangle_{opt}$ = 0 for the best curve. For computation of min(P_{opt}), modules are ordered in an increasing order on the x-axis by their actual fault densities which is the same as finding the worst case of prediction. To find the best performing prediction model, we employed Nemenyi statistical test and Deḿsar's significance diagram to find the significant differences between the different models. The statistical test considers the rank orders of the Norm(P_{opt}) values. The null hypothesis that the

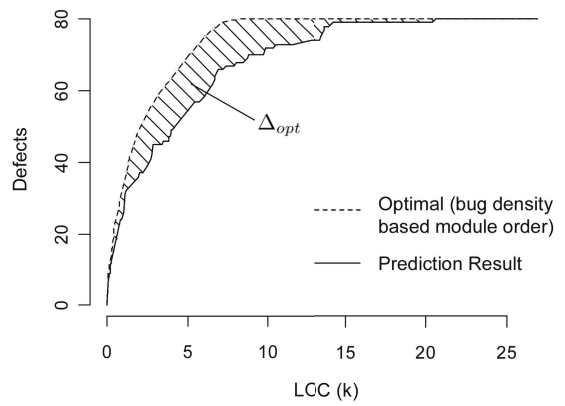

Figure 1: Example of LOC-based cummulative lift chart [12]

average rank (AR) values for each predictive model are equal and cannot be rejected and a significant level of 5% was used for the test.

G. Experiment execution environment

Construction and evaluation of the fault density prediction model was done using R [24] which is an open source statistical processing environment. Also, determination of hyper parameters and prediction model involving different data set split of 10-fold cross-validation was implemented using the caret packages [25] that is available in R.

IV. Results and Discussion

A. Evaluation of fault density prediction models that take into account testing effort

Table IV shows the experimental results for the cross-validation analysis using the same data set or release. From the table, each row represent the Norm(P_{opt}) values for each data set for a specific prediction model. The AR at the bottom represents rank averages of Norm(P_{opt}) values on all data sets for a specific technique. The equation (2) shows the AR_i formula for the entire data set k. Here, r_j^i represents the rank order of Norm(P_{opt}) values (descending) when you apply the ith classifier to the jth data set.

$$AR_i = \frac{1}{k}\sum_{j=1}^{k} r_j^i \tag{2}$$

From Table IV, the bolded and underlined value in each row represents the highest Norm(P_{opt}) value, which indicates higher accuracy hence, the best performing predictive model for that specific data set. AVG is the average value of Norm(P_{opt}) values for each prediction model on all data sets. In the AR region, the most accurate value is bolded and underlined, which indicates a low (high accuracy) value. For almost all the data sets, Norm(P_{opt}) values for the prediction models are more than 50% and even some models had values greater than 70%. It was observed that different predictive models performed quite well for different data sets, that is no single predictive model had the best performance for all the data sets. However, two models K* and RF performed admirably well (9 and 7 out of 25 with respect to their ranks) whilst RVM and RPART couldn't perform for any of the data set. K* also had the best AVG and AR values of 72.79 and 3.16 respectively.

Furthermore, for the comparison of the prediction models, we followed the experimental framework proposed by Lessmann et al. [10]. Nemenyi test results' was expressed using Deḿsar's significance diagram (Figure 2) to statistically compare the relative merits between different models. In the diagram, the classification techniques are listed in ascending order of ranked performance on the y-axis and the average rank (AR) of prediction performance across all 25 data sets are displayed on the x-axis. From Figure 2, the deeply shaded bullet beginning each solid horizontal line represents the AR value for each effort-aware model and the vertical dashed line identifies the end of the best model's tail that is the lowest AR value (lower average of ranked performance) which is K*. It also represents the maximum number of AR values that cannot be rejected by the null hypothesis. The diagram shows that we cannot reject the null hypothesis since none of the AR values (deeply shaded bullet) started after the vertical dashed lines thus there is no significant difference among the models. However, RPART performs worse than the other effort-aware models with an AR value of 8.84. The results implies that prediction performance of all models improved since we couldn't reject all the models. Observing the average (AVG) Norm(P_{opt}) for each predictive model on each data set, the lowest value was more than 50% which agrees with Mende et al's [23] conclusion that inclusion of effort-awareness in prediction models improves their performance from a practical point of view.

B. Evaluation of fault density prediction model across versions

For fault density prediction model constructed using data set version A to predict fault-prone accuracy of data set version B, the Norm(P_{opt}) values for the cross-release are represented in Table V. The values in each row represents the Norm(P_{opt}) for each predictive model on each data set. For almost all the data sets, Norm(P_{opt}) values for the prediction models are more than 50% and values greater than 60% were observed on some few models with the best performing model for each data set bolded and underlined. Similar to the experimental results of the cross-validation study on a single software version, no single predictive model outperformed the others. However, M5 was the best performing model across all data sets with average rank (AR) value of 3.44 whilst RVM, SVM and RPART could not perform for any of the data set hence having the lowest average(AVG) Norm(P_{opt}) values or highest AR values.

Comparing the significant differences between the models using Deḿsar's significance diagram from Figure 3, there were no significant differences statistically since the vertical dashed line fails to reject all the models.

Table IV: Prediction performance of each model when testing effort is considered

	Predictive Models										
Data sets	LM	LARS	RVM	k-NN	K*	NNET	SVM	M5	RPART	GBM	RF
CLAM	**94.2**	93.7	50.1	62.7	93.4	38.9	88.2	92.3	73.8	93.1	93.4
ECOS	58.3	62.6	51.6	49.6	**73.3**	54.4	46.5	64.4	60.4	65.5	72.3
EXIM	44.3	51.8	24.4	58.7	**72.8**	23.8	68.1	54.0	24.4	55.6	70.1
FREE	71.4	75.7	61.5	75.0	**80.2**	68.0	63.4	76.6	59.0	72.1	**80.2**
GANY	43.2	46.4	49.4	51.4	53.9	**68.2**	49.2	46.4	45.6	54.9	54.8
HELM	69.9	70.4	32.6	67.6	64.6	30.9	64.5	64.2	36.2	61.8	**73.3**
HIBE	67.0	**76.7**	59.6	50.6	74.0	76.1	49.4	56.8	52.6	72.0	71.2
HYLA	46.9	88.3	78.6	74.3	83.1	55.0	70.7	82.7	87.6	76.9	**92.5**
IPTA	57.8	57.9	65.0	66.3	65.8	**72.7**	68.8	69.8	58.0	61.9	68.0
KOFF	52.8	60.6	59.3	62.9	61.6	64.6	51.1	67.6	58.2	**75.8**	62.2
NBSD	67.6	73.7	44.5	64.1	**78.4**	43.1	64.9	74.6	35.3	71.2	**78.4**
OBSD	75.3	81.9	29.4	61.2	81.5	51.4	65.4	**82.1**	25.8	69.3	78.9
OCMS	**55.8**	53.7	42.2	51.1	51.8	47.8	48.5	54.6	41.9	51.2	45.3
ONMS	56.8	61.0	60.5	46.7	**74.0**	65.8	48.2	73.5	51.7	72.5	69.0
OTRS	52.0	60.6	**70.6**	59.2	65.6	65.8	66.7	64.2	70.0	58.0	61.1
POST	47.0	75.4	60.8	70.9	73.3	74.6	64.1	**76.1**	42.1	67.1	70.9
PROF	71.0	67.4	55.5	60.4	**71.1**	43.3	67.3	68.8	56.9	68.4	63.0
SAMB	46.6	53.7	42.3	65.9	**75.2**	68.0	56.7	75.0	41.6	42.2	58.9
SCIL	73.1	70.4	58.0	59.5	76.8	67.9	58.1	76.1	66.5	74.3	**78.7**
SPRI	73.4	73.0	61.9	66.5	72.7	70.2	59.0	76.1	63.3	74.7	**76.7**
SQUI	66.5	62.1	37.2	49.6	60.5	38.5	63.9	**68.0**	37.5	63.5	64.1
WINE	62.9	**76.6**	50.3	68.2	66.2	53.0	56.1	76.2	48.9	62.8	66.7
XDOC	72.6	79.7	55.7	58.2	**84.8**	61.4	64.5	82.4	46.8	82.5	82.9
XFRE	51.0	77.6	61.6	75.8	77.1	75.0	67.8	68.5	71.9	75.1	**81.2**
XORP	84.3	80.8	30.9	72.8	**88.1**	47.9	67.7	82.4	79.7	83.0	86.0
AVG	62.47	69.27	51.74	61.97	**72.79**	57.05	61.55	70.94	53.43	68.22	71.99
AR	6.52	4.64	8.80	6.88	**3.16**	7.00	7.56	3.84	8.84	5.20	3.32

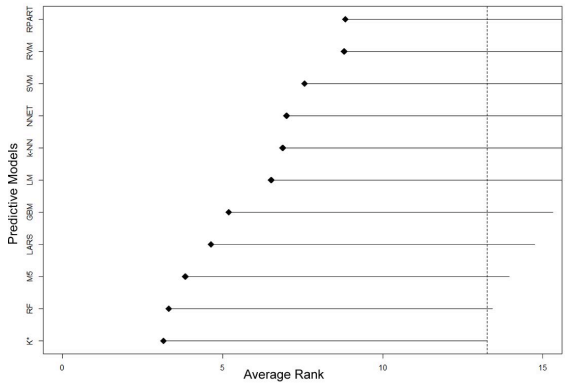

Figure 2: Demšar's significance diagram in RQ1

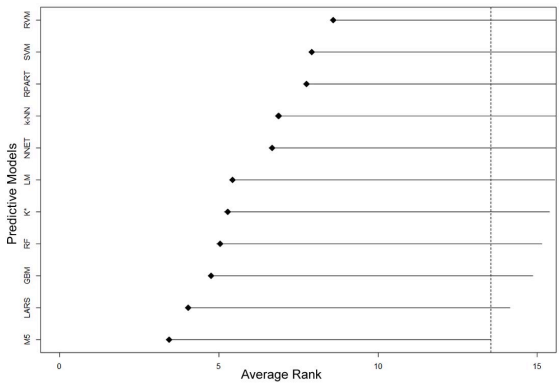

Figure 3: Demšar's significance diagram in RQ2

Comparing the results to that of RQ1 in the case of fault-prone module prediction on single software versions, the overall Norm(P_{opt}) values were low. Furthermore, examining the relationship between the prediction models of RQ1 and RQ2, the correlation coefficient of the average ranks of RQ1 and RQ2 showed a strong positive correlation of 0.87. This implies that for fault-prone module prediction using fault density, utilization of cross-validation methods for the model building in previous versions may help in the selection of the best fault-prone models for cross versions prediction.

C. Discussions

In light of the analysis conducted for the two RQs, it was observed that not one effort-aware model proved very dominant and was statistically significant from the others. Also the best prediction model varied according to the data sets in both RQs. We sought to find the external factors causing the oscillation in the results and thus considered the P_{fp} (% of fault-prone modules) in the fit data set for each OSS project. It was evident that the data set size was one key reason for the variation in the performances of the models as was observed from the results presented in Tables IV and V. We next considered the P_{fp} because of the unpredictability of real world data sets. Also, the advent of several sampling techniques provides the means to modify or control the P_{fp} in defect data sets.

For RQ1, a stratification method was employed in the splitting of each data set to get the fit and test data sets ensuring that the P_{fp} was same and maintained in both fit and test data sets. We present a bubble chart to show the

Table V: Prediction performance considering testing effort for cross-release analysis

	Predictive Models										
data sets	LM	LARS	RVM	K-NN	K*	NNET	SVM	M5	RPART	GBM	RF
CLAM	79.2	94.7	83.3	69.3	88.8	67.1	85.1	90.1	87.9	**96.5**	94.6
ECOS	58.0	**59.8**	51.7	55.8	55.6	45.7	52.9	58.6	49.9	53.4	53.1
EXIM	55.1	52.8	56.0	42.1	24.3	**83.1**	55.2	34.7	56.0	32.5	36.4
FREE	73.2	79.2	66.4	81.1	81.2	**81.6**	64.2	81.4	69.2	78.2	80.0
GANY	**56.7**	53.6	50.8	53.5	49.4	54.9	55.5	54.6	51.0	47.5	48.4
HELM	41.7	40.6	48.1	43.7	53.8	**58.2**	42.4	40.5	48.3	50.2	44.9
HIBE	42.3	49.4	45.4	55.2	54.6	58.5	45.1	**58.9**	41.7	56.4	53.4
HYLA	73.8	77.6	73.6	68.2	65.8	57.9	70.3	73.5	**78.0**	76.5	76.7
IPTA	64.3	58.4	52	56.5	**72.4**	59.6	58.4	60.6	49.6	65.3	64.2
KOFF	50.8	62.5	51.4	59.8	55.1	56.4	53.2	**63.9**	54.5	56.4	53.6
NBSD	61.4	**66.7**	46.8	56.8	61.8	42.9	55.1	65.0	37.4	65.8	61.2
OBSD	70.6	**74.9**	39.6	64.6	69.9	52.8	46.3	74.7	27.1	58.7	61.2
OCMS	**76.4**	75.7	64.8	63.3	65.7	50.7	68.8	66.9	70.6	68.2	66.5
ONMS	54.7	57.3	48.9	60.1	53.3	55.2	41.9	60.2	47.4	**61.3**	60.0
OTRS	80.9	**88.8**	88.4	65.3	86.6	79.7	61.9	85.2	88.5	88.1	78.0
POST	59.6	56.5	59.4	60.3	64.1	62.0	55.6	**65.7**	48.0	60.6	62.9
PROF	**68.5**	64.1	58.1	51.8	53.6	58.7	54.5	65.6	60.1	57.5	67.9
SAMB	62.9	68.7	37.2	62.2	68.3	49.4	56.4	69.5	37.3	56.8	**69.8**
SCIL	58.2	62.7	40.5	55.3	62.5	61.8	52.4	**63.3**	51.8	61.5	62.4
SPRI	66.3	**69.3**	49	59.3	60.4	64.1	57.7	**69.3**	53.9	65.9	68.0
SQUI	64.2	**75.4**	37.0	54.4	59.5	44.9	56.0	70.7	36.1	70.3	48.6
WINE	64.7	68.7	50.8	65.6	61.2	58.7	60.3	**71.8**	57.0	68.1	66.7
XDOC	69.4	60.1	34.4	55.8	**74.6**	37.3	66.1	55.6	43.0	66.0	69.2
XFRE	65.0	63.6	64.1	64.4	73.7	66.3	53.5	**74.3**	62.7	68.5	66.6
XORP	67.2	80.2	22.8	60.2	81.3	51.2	67.1	**85.8**	82.1	81.9	82.5
AVG	63.40	**66.45**	52.82	59.38	63.90	58.35	57.44	66.42	55.56	64.48	63.87
AR	5.44	4.04	8.60	6.88	5.28	6.68	7.92	**3.44**	7.76	4.76	5.04

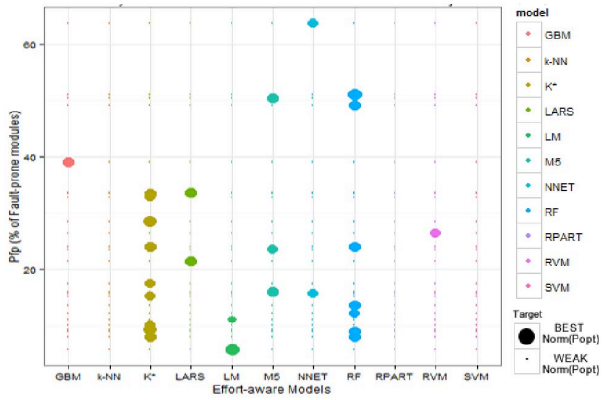

Figure 4: Prediction performance of each model on the 25 OSS projects sorted according to P_{fp}

relationship between the P_{fp} and performance of each effort-aware model. Figure 4 displays a bubble chart where the deeply shaded bubbles represent the highest Norm(P_{opt}) value of a specific prediction model for each data set horizontally which have been sorted according to their P_{fp} on the y-axis.

It could be observed that K* and RF won in most cases when P_{fp} was below 40%. When P_{fp} was above 40%, RF and M5 performed best and above 60%, NNET performed best. Contrasting results were achieved in RQ2 as presented in Figure 5, it is evident that variation in the P_{fp} of the test data greatly affects performance of effort-aware models. The LARS and M5 models were dominant when P_{fp} was below 40%.

The prediction models having best performance are different in the case of cross validation and that of cross release analysis due to the variation of data set size and P_{fp} modules in each data set. Comparing our results to the results achieved by D'Ambros et al. [26], we do attain similar results when the effort-aware performance measure was used. Factors such as the data set size and class distribution had significant effect on the results, hence not one prediction model could outperform the others. We thus concur with their conclusion that external circumstances do affect prediction performance and thus, limits the generalization of our results to other contexts. We however, recommend that in using effort-aware models, K* model should be considered if the P_{fp} is the same in both train and test data set which is less likely in a real world data set hence M5 could be considered since it performed well for test data sets of varying P_{fp} values in both RQ's.

V. Threats to Validity

In the conduct of our experiments, several potential threats which could undermine our results are discussed in this section. The data sets used are a possible source of bias in relation to the representativeness and measurement accuracy to aid in the generalization of our conclusion results. Since we considered purely OSS data sets, it may be difficult to generalize the results to closed source projects or the industrial settings. Also, relying solely on faults recorded in the commit logs from version control tools such as CVS and GIT implies that faults not recorded in such tools would be ommited from our study. Considering the keywords used to find the faults in our study, likely faults which were not labeled as "bugfix" or "fix" would also be excluded. Further analysis is required to accurately ascertain and collect all faults.

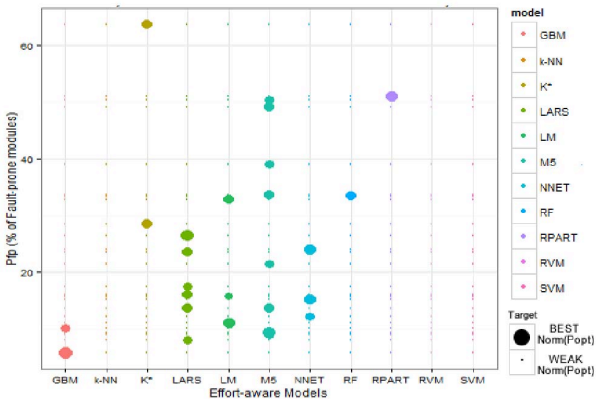

Figure 5: Cross-release prediction performance of models on the 25 data sets sorted according to P_{fp}

The sampling and splitting techniques could also affect our results. We randomly split the data sets into fit (2/3) and test (1/3) for RQ1. This procedure has been well applied in several empirical experiments and thus an established approached to predictive data modelling [27]. Also, the large size of each data set complements the splitting procedure. The selection of the hyper parameters could also affect our results since the values were estimated using a 10-fold cross-validation method.

Finally, the predictive models used for this experiment may not be exhaustive of all models available. With the availability of several models, other models which could have performed better were not considered. However, the selection was made with the aim of finding a meaningful balance between the several categories of techniques (machine leaning, statistical, data mining and others). We thus believe that, the models applied are representative of the most important techniques found in literature.

VI. Related Work

A. Evaluation of models that take into account the testing effort

Arisholm et al. [8] stipulated the need of evaluating predictive models by assessing their potential cost-effectiveness and not only by considering their accuracy if the economic viability and benefits of these fault-proneness prediction models are to be realised. A study by Mende et al. [23] showed an example of module size used to evaluate a fault-prone module prediction model considering the cost-effectiveness of applying the model. They proposed bug prediction models that are effort-aware (effort for review or code testing) and considered number of lines of code of the module to be proportional to the testing effort which were evaluated with the prediction accuracy of each model. Further, Kamei et al. [12] performed a fault-prone module prediction by extending Mende et al.'s method and observed very interesting findings. For example, the process metrics when the testing effort was considered showed that it is possible to construct a good predictive model than product metrics. D'Ambros et al. [26] extensively experimented and evaluated several prediction models using traditional and effort-aware performance measures considering different software metrics. They conclude that some approaches performed better than others statistically but external validity limits the generalization of results to different contexts.

Therefore, among the various evaluation measures (such as precision/recall/F-measure/ROC-Curve), the normalised P_{opt} proposed by Kamei et al. [12] is a more realistic performance evaluation of predictive models due to its ability to consider testing effort. Monden et al. [28] also applied normalised P_{opt} with the assumption that effort is not distributed uniformly across all modules since some modules will be more expensive to test or review than others. The normalised P_{opt} was further discussed in Section III.

In this paper, considering the testing effort (module scale) to compare the performance of the prediction model, 25 distinct projects were used in the experiment as a way of increased number of projects.

B. Evaluation of models that span the version

Traditionally, cross-validation as an evaluation method of fault-prone module prediction model (N-fold cross validation) has been widely used [10], [23], [22]. Menzies et al. [29] argued that for a model to have useful future validity, the learned theory should be tested on data which was never used in building the model since failure to do that could result in over-estimating the learned model. More recently, evaluation of prediction model's performance has been carried out using multiple versions [12], [30], [31]. For example, Kamei et al. [12] constructed a model from modules of some specific old versions of three open source software types.

In this paper, for a more realistic evaluation performance, we built a prediction model using the historical version of the data and evaluated its predictive performance when applied to the next version.

VII. Conclusions

The purpose of this paper is to empirically evaluate the best performing fault-prone prediction model when testing effort is being considered using two proposed research questions (RQs). Based on 25 open source projects, we extensively experimented and found the best models by evaluating them using the normalized P_{opt} because it considers testing efforts which helps in finding the cost of additional quality assurance activities. For RQ1, we compared 11 different kinds of fault prediction models on the data sets of same version. Among all the models used, K* was the best performing model. As a result of comparison with the other techniques to K*, lower differences was observed statistically to the remaining techniques but not significant. In RQ2, 11 types of fault density prediction models were compared using previous software versions as fit data set and current (new) version as test data set. As expected, the Norm(P_{opt}) values dropped in comparison to the cross validation method for the same software version (RQ1) and the M5 tree model had the highest value between versions but no significant differences were

also observed statistically in relation to the other models. Two different models performed well for both research questions and this could be explained by the different data sets used for testing in both instances. Also, the data set size and percentage of faulty modules present in the data set greatly affected the performance of the prediction models. This is in concordance to what was observed by [10] that the distribution of target variable and size of the data set (fit and test) has an effect on the performance measure. Further analysis of the average Norm(P_{opt}) values for both RQs shows that prediction performance of all models considered had an overall average performance value of more than 50%.

Also, it was observed that there was a strong correlation between average rank of the prediction accuracies for both the intra-release and cross-release versions when cross-validation method was employed. Therefore, it is possible to make use of cross-validation method in previous versions which may be useful for model building techniques for fault density prediction across versions.

From these results, an appropriate scenario of applying fault prediction models in a real development environment has been shown in this paper. For future studies, we intend to apply sampling techniques to fix the imbalanced data set condition between two versions before applying the models.

ACKNOWLEDGEMENT

Special thanks to Hirotake Kobayashi and Tsunenori Mine for help in running some of the experiments and review of content. This research is supported by the City University of Hong Kong research funds (Project No. 7200354, 7004222, 7004474).

REFERENCES

[1] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *Software Engineering, IEEE Transactions on*, vol. 26, no. 8, pp. 797–814, 2000.

[2] A. Mockus and D. M. Weiss, "Predicting risk of software changes," *Bell Labs Technical Journal*, vol. 5, no. 2, pp. 169–180, 2000.

[3] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 452–461.

[4] T. M. Khoshgoftaar, A. S. Pandya, and D. L. Lanning, "Application of neural networks for predicting program faults," *Annals of Software Engineering*, vol. 1, no. 1, pp. 141–154, 1995.

[5] P. L. Li, J. Herbsleb, M. Shaw, and B. Robinson, "Experiences and results from initiating field defect prediction and product test prioritization efforts at abb inc." in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 413–422.

[6] P. Knab, M. Pinzger, and A. Bernstein, "Predicting defect densities in source code files with decision tree learners," in *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 2006, pp. 119–125.

[7] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *Software Engineering, IEEE Transactions on*, vol. 31, no. 5, pp. 380–391, 2005.

[8] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.

[9] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Software Engineering*, vol. 13, no. 5, pp. 561–595, 2008.

[10] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *Software Engineering, IEEE Transactions on*, vol. 34, no. 4, pp. 485–496, 2008.

[11] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and Software Technology*, vol. 55, no. 8, pp. 1397–1418, 2013.

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

[13] N. F. Schneidewind and H.-M. Hoffmann, "An experiment in software error data collection and analysis," *Software Engineering, IEEE Transactions on*, no. 3, pp. 276–286, 1979.

[14] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani *et al.*, "Least angle regression," *The Annals of statistics*, vol. 32, no. 2, pp. 407–499, 2004.

[15] C. M. Bishop and M. E. Tipping, "Variational relevance vector machines," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 2000, pp. 46–53.

[16] J. G. Cleary, L. E. Trigg *et al.*, "K*: An instance-based learner using an entropic distance measure," in *Proceedings of the 12th International Conference on Machine learning*, vol. 5, 1995, pp. 108–114.

[17] F. Xing, P. Guo, and M. R. Lyu, "A novel method for early software quality prediction based on support vector machine," in *Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on*. IEEE, 2005, pp. 10–pp.

[18] E. Frank, Y. Wang, S. Inglis, G. Holmes, and I. H. Witten, "Using model trees for classification," *Machine Learning*, vol. 32, no. 1, pp. 63–76, 1998.

[19] T. M. Khoshgoftaar and N. Seliya, "Comparative assessment of software quality classification techniques: An empirical case study," *Empirical Software Engineering*, vol. 9, no. 3, pp. 229–257, 2004.

[20] P. Bühlmann and T. Hothorn, "Boosting algorithms: Regularization, prediction and model fitting," *Statistical Science*, pp. 477–505, 2007.

[21] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of fault-proneness by random forests," in *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on*. IEEE, 2004, pp. 417–428.

[22] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*. IEEE, 2008, pp. 181–190.

[23] T. Mende and R. Koschke, "Effort-aware defect prediction models," in *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*. IEEE, 2010, pp. 107–116.

[24] R. C. Team, "R: A language and environment for statistical computing," *Vienna, Austria: R Foundation for Statistical Computing*, 2012.

[25] M. Kuhn, J. Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, and Z. Mayer, "caret: classification and regression training. r package version 6.0-24," 2014.

[26] M. DŠAmbros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.

[27] X. Tan, X. Peng, S. Pan, and W. Zhao, "Assessing software quality by program clustering and defect prediction," in *Reverse Engineering (WCRE), 2011 18th Working Conference on*. IEEE, 2011, pp. 244–248.

[28] A. Monden, T. Hayashi, S. Shinoda, K. Shirai, J. Yoshida, M. Barker, and K. Matsumoto, "Assessing the cost effectiveness of fault prediction in acceptance testing," *Software Engineering, IEEE Transactions on*, vol. 39, no. 10, pp. 1345–1357, 2013.

[29] T. Menzies, J. DiStefano, A. Orrego, and R. Chapman, "Assessing predictors of software defects," in *Proc. Workshop Predictive Software Models*, 2004.

[30] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *Software Engineering, IEEE Transactions on*, vol. 31, no. 10, pp. 897–910, 2005.

[31] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*. IEEE, 2005, pp. 284–292.