Revisiting Dependence Cluster Metrics based Defect Prediction

Qiguo Huang*,Xiang Chen*†,Zhengliang Li*,Chao Ni*,Qing Gu*‡
*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
†School of Computer Science and Technology, Nantong University, Nantong 226019, China

***Abstract*—A dependence cluster is a set of program elements that all depend upon each other. Prior empirical studies have found that the dependence cluster based metrics are useful in effort-aware defect prediction. However, it is still unknown whether they are useful in non-effort-aware defect prediction. In this paper, we perform empirical studies to investigate this issue. We use the product, process, and network metrics to build the "B" model (baseline model), and then use the product, process, network and dependence cluster metrics to build the "B+C" model. Our experimental results, based on five well-known open-source systems, show that the dependence clusters are useful for non-effort-aware defect prediction. These findings help us better understand how dependence clusters influence non-effort-aware defect prediction.**

***Index Terms*—dependence cluster, metrics, defect prediction, revisiting, non-effort-aware**

I. Introduction

A dependence cluster is a set of program elements that all depend upon each other [1], [2]. Prior studies showed that large dependency clusters are widely existed in all kinds of source code. The existence of the large dependent clusters will result in ripple effects — a code change in one element of the dependent cluster will produce potential impact on other elements of the cluster [1], [2]. A high-quality software system should reduce or even eliminate large dependency clusters since they not are easier to develop, maintain, and reuse. Therefore, The detection and analysis of dependency clusters is the key point. Binkley et al. [1] used the technology of program slicing to solve this problem by defining the program system dependence graph. Based on their observation, dependence clusters can influence software quality. Yang et al. [3] first applied dependence cluster metrics to software defect prediction, and their empirical studies have found that the dependence cluster based metrics are useful in effort-aware defect prediction. However, it is still unknown whether they are useful in non-effort-aware defect prediction. In this paper, our study attempts to fill this gap.

[‡]Corresponding Author, Email: guq@nju.edu.cn
DOI reference number: 10.18293/SEKE2020-060

Based on this motivation, we investigate the effectiveness of dependence cluster based metrics in terms of non-effort-aware performance indicators. Our main contributions are the following:1) We investigate whether dependence clusters are useful for non-effort-aware defect prediction. Our results show that the dependence clusters are still useful for non-effort-aware defect prediction. 2) We examine whether our conclusions change if the potentially confounding effect of module size is excluded. The results show that the "B+C" model still performs better than the "B" model. 3) We examine whether our conclusions change if the class imbalanced method is used. The results show that the "B+C" model still performs better than the "B" model.

The rest of this paper is organized as follows. In Section II, we summarize related work. In Section III, we present the research questions and research method. We describe experiment setup, including studied projects, data collection, and performance indicators in Section IV. In Section V, we report out experimental results. Section VI discusses our findings. Finally, we conclude the paper and direct future work.

II. Related Work

In this section, we summarize related work on non-effort-aware defect prediction and dependence clusters.

A. Non-effort-aware Defect Prediction

The non-effort-aware defect prediction is also called traditional defect prediction [4]. It includes within-project defect prediction and cross-project defect prediction. For within-project defect prediction, Hall et al. [5] investigated the effect of model independent variables and model techniques on the performance of defect prediction model. Their results showed that simple modeling techniques, such as Logistic Regression, tended to perform well. For cross-project defect prediction, To the best of our knowledge, the earliest study on CPDP(Cross-Project Defect Prediction) was performed by Briand et al. [6], they used logistic regression to build defect prediction models based on the Xpose project. The results showed that the CPDP model is better than the random model, but lower than the within-project defect prediction performance. Ulike the above stduies, we investigate whether dependence clusters have practical value in non-effort-ware defect prediction.

B. Dependence Clusters

The concept of dependency clusters based on program slicing at the statement level was first proposed by Binkley et al. [1]. The dependence cluster is a set of program elements that all depend upon each other. Later, Harman et al. [2] extended Binkley's study to modules with coarser granularity. Their results showed that the accuracy of the "same slice size" method proposed by Binkley is very high. In addition, they found that large dependency clusters are widely existed in analyzed software projects. Yang et al. [3] first applied dependence cluster based metrics to effort-aware software defect prediction. Their empirical result showed the combination of the product, process, network and dependence cluster metrics produce more effective models for the prediction of post-release defect than the combination of the product, process and network metrics alone. Our study is different from their studies, we study the defect prediction model in non-effort-aware evaluations with respect to within-project and cross-project.

III. RESEARCH METHODOLOGY

In this section, we first introduce the research questions, then give the research method for the research questions.

A. Research Questions

In order to be easily understand research questions, we use the system dependence clusters shown in Figure 1 [3] to illustrate the questions. In Figure 1, the nodes indicate functions and the directed edges indicate dependencies between functions, which includes the data dependencies and the function call dependencies. Such as, from $f1$ to $f16$ are functions, labeled "d" and "c" indicate data dependencies and function call dependencies, respectively. In this dependency graph, there are 16 functions and 3 dependency clusters which are $dc1$, $dc2$ and $dc3$. In Figure 1, the functions are divided into two groups: functions inside dependence clusters and functions outside dependence clusters. Such as, from $f1$ to $f4$ are functions inside $dc1$, and from $f12$ to $f16$ are functions outside dependence clusters. Functions inside dependence clusters and functions outside dependence clusters form the subgraphs SubG$_{in}$ and SubG$_{out}$, respectively.

Based on the above the preliminary knowledge, we aim to investigate whether dependence clusters are useful for non-effort-aware defect prediction Therefore, our research questions are set up as follows:

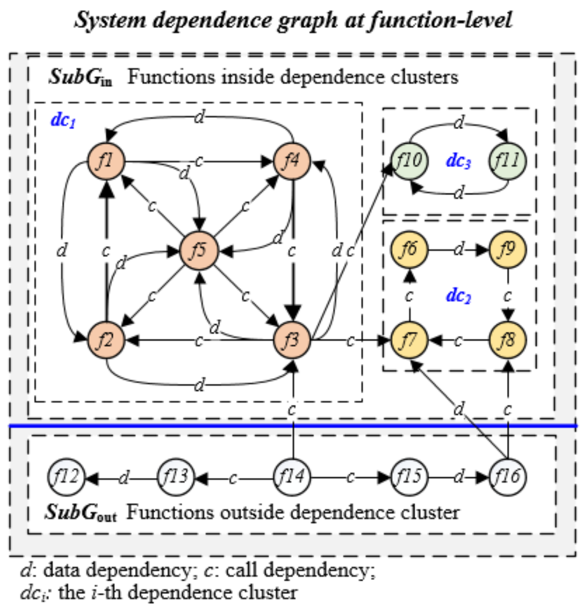
System dependence graph at function-level
SubG_in Functions inside dependence clusters
dc1
dc2
dc3
f1
f2
f3
f4
f5
f6
f7
f8
f9
f10
f11
f12
f13
f14
f15
f16
d
c
SubG_out Functions outside dependence cluster
d: data dependency; c: call dependency;
dc_i: the i-th dependence cluster

Fig. 1: An SDG with dependence clusters

RQ1. In the scenario of within-project defect prediction, are dependence cluster based metrics useful for non-effort-aware prediction?

RQ2. In the scenario of cross-project defect prediction, are dependence cluster based metrics useful for non-effort-aware prediction?

These research questions are important to both software researchers and practitioners, as they help us better understand the effects of dependence clusters on software quality.

B. Research Method

In order to answer RQ1 and RQ2, we use AIC(Akaike Information Criterion) as the criteria to perform a forward stepwise variable selection procedure to build the following two types of multivariate logistic regression models: (1) the "B" model (using product, process and network metrics); (2) the "B+C" model (using product, process, network and dependence clusters metrics). The logistic regression has been widely used for building defect prediction models [7], [8]. We choose the forward stepwise variable selection rather than the backward stepwise variable selection because the forward stepwise variable selection is less time consuming on stepwise variable selection especially for plenty of independent metrics. The AIC criteria is a widely used variable selection [9].

IV. EXPERIMENTAL SETUP

In this section, we first introduce the projects used in our study and the method of collection the data. Then, we give a description of the performance indicators in this study.

A. Experimental Subjects

We use the five well-known open source projects to investigate the predictive capability dependence clusters based metrics for non-effort-aware defect prediction: Gstreamer (GSTR), Glibc (GLIB), Gimp (GIMP) and Bash (BASH). They are all GNU projects. In Table I, from the second to seventh columns are respectively the version number, the release date, the total source lines of code in the each studied project, the number of functions, the number of faulty functions, and the percentage of faulty functions. From eighth to the ninth columns are the previous version number and the release date of the previous version, which used to compute the process metrics. The last two columns are the fixing version number and the release date of the fixing release, which are used to determine the faulty or not faulty label for each function.

B. Data Collection

We used the Understand[1] tool and R package igrah[2] to collect the data from the above-mentioned five projects. Metrics for each project consist of: 1) 16 product metrics (i.e. SLoC metrics); 2) 3 process metrics (i.e. code churn metrics); 3) 21 network metrics (i.e. Ties metrics); 4) Collected the dependence clusters for each system ; 5) Collect the importance metrics for dependence clusters [3]; and 6) the faulty or not-faulty labels of the functions after version release.

Table II describes the dependence clusters in the experimental systems. The second to the fifth columns respectively show the number of functions, the number of clusters, the percentage of functions inside dependence clusters, and the size of the largest cluster in each experimental system. We can see that there exist many clusters in these systems from Table II. Table III describes the importance metrics for dependence clusters in the experimental systems. These metrics are widely used network metrics [10].

C. Performance Indicators

At present, most of the existing research work regards the problem of the defect prediction as a binary classification problem. We set defective functions as positive and non-defective functions as negative. We combine the real results of function with the predicted results of model and divide into true positive (TP), false positive (FP), true negative (TN) and false negative (FN). Let TP, FP, TN and FN denote the corresponding numbers of functions, respectively. These values are stored in the confusion matrix, and the confusion matrix is used to compute the Precision, Recall, and F-measure performance indicators. These indicators are defined as follows:

- Precision: The ratio of correctly predicted defective functions over all the functions predicted as being defective. It is calculated as:

$$Precision = \frac{TP}{TP+FP} \tag{1}$$

- Recall: The ratio of correctly predicted defective functions over all of the true defective functions. It is calculated as:

$$Recall = \frac{TP}{TP+FN} \tag{2}$$

- F-measure: The indicator is harmonic mean of the precision and recall. It is calculated as:

$$F-measure = \frac{2 \times precision \times recall}{precision + recall} \tag{3}$$

These indicators are widely used for non-effort-aware defect prediction [11].

V. EXPERIMENTAL RESULTS

In this section, we first describe the models ("B" model and "B+C" model), then we present the experimental results for RQ1 and RQ2.

A. The Models

Figure 2 [3] provides an overview of analysis method for RQ1 and RQ2. In order to answer RQ1 and RQ2, we first use the procedure described in section 3.2 to build "B" model and "B+C" model on each data set, respectively. The introduction of "B" and "B+C" models as follows:

(1) **The "B" model.** It is the baseline model, which is built with product, process, and network metrics. These metrics are described in Table IV. We choose the metrics as the baseline metrics since they are widely used in defect prediction [12].

(2) **The "B+C" model.** The functions are divided into two groups: functions inside dependence clusters and functions outside dependence clusters, the "B+C" model is segmented model which consists of two independent models. They are the "B+C$_{in}$" model and the "B+C$_{out}$" model. "B+C$_{in}$" and "B+C$_{out}$" models are respectively used for predicting the probability that a function inside and outside dependence clusters are faulty. They are both built with product, process, network and the importance metrics described in Table III. After building the models, we can verify RQ1 and RQ2.

[1]https://scitools.com/

[2]https://igraph.org/r/

TABLE I: Studied projects and version information

System name	Subject release						Previous release		Fixing release	
	Version number	Release Date	Total SLoC	# functions	# faulty functions	% faulty functions	Version	Release Date	Version	Release Date
Gstreamer	1.0.0	2012-09-24	75985	3946	146	3.70%	0.11.90	2011-08-02	1.0.10	2013-08-30
Glibc	2.1.1	1999-05-24	172599	5923	417	7.04%	2.0.1	1997-02-04	2.1.3	2000-02-25
Gimp	2.8.0	2012-05-12	557436	19978	818	4.10%	2.7.0	2009-08-15	2.8.16	2015-11-21
Gcc-core	4.0.0	2005-04-21	422182	13612	430	3.16%	3.4.0	2004-04-20	4.0.4	2007-01-31
Bash	3.2	2006-10-11	49608	1947	68	3.49%	3.1	2005-12-08	3.2.57	2014-11-07

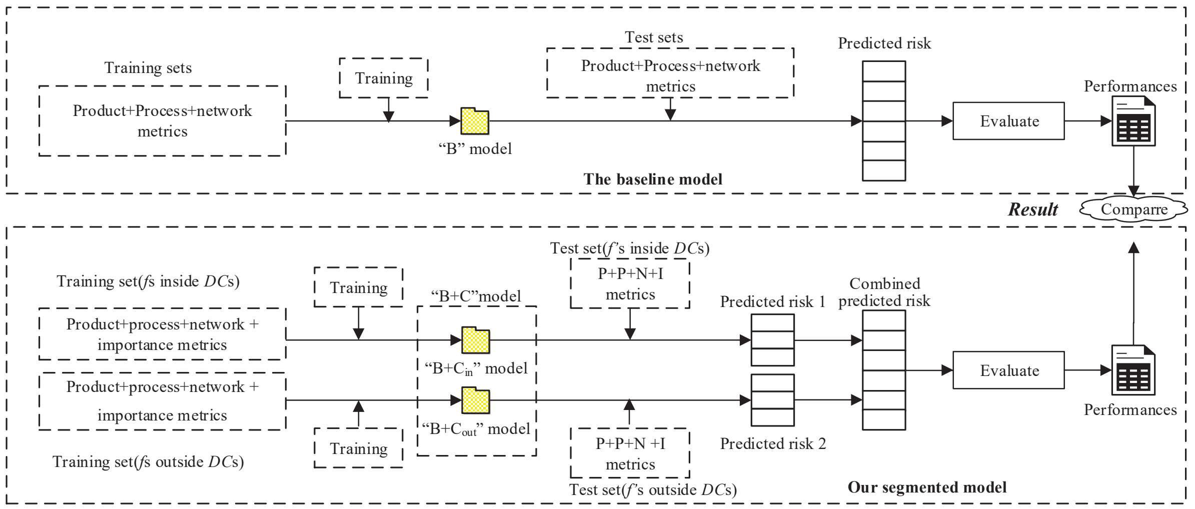

Fig. 2: Overview of the analysis method for "B" VS "B+C" in within-project and "B" VS "B+C" in cross-project

TABLE II: The dependence clusters in experimental systems

System name	# functions	# clusters	% functions Inside clusters	Size of Largest cluster
Gstreamer	3946	59	15.2	170
Glibc	5923	105	11.6	277
Gimp	19978	363	14.2	158
Gcc-core	13612	139	34.9	4083
Bash	1947	41	46.2	483

TABLE III: The importance metrics for dependence clusters

Metrics	Description
Betweenness	# shortest paths through the vertex
Centr_betw	Centrality score according to betweenness
Centr_clo	Centrality score according to the closeness
Centr_degree	Centrality score according to the degrees
Centr_eigen	Centrality score according to eigenvector
Closeness	How close to other vertices
Constraint	The Burt's constraint
Degree	# v's adjacent edges
Eccentricity	Maximum graph distance to other vertices
Page_rank	Google page rank score

TABLE IV: Baseline metrics in this study

Category	Description
Product	SLOC, FANIN, FANOUT, NPATH Cyclomatic, CyclomaticModified, CyclomaticStrict, Essential, Knots, Nesting,MaxEssentialKnots, MinEssentialKnots, n1, n2, N1, N2
Process	Added, Deleted, Modified
Network	Size, Ties, Pairs, Density, nWeakComp, pWeakComp, 2StepReach, ReachEffic, Broker, nBroker, EgoBetw, nEgoBetw, effsize, efficiency, constraint, Degree, Closeness, dwReach, Eigenvector, Betweenness, Power

B. Experimental Result

In the following, we describe the experimental results for RQ1 and RQ2, respectively.

(1) ***RQ1.*** *In the scenario of within-project defect prediction, are dependence cluster based metrics useful for non-effort-aware prediction?*

For RQ1, we use 30 times 3-fold cross-validation to evaluate the effectiveness of the prediction models. We use the same training/test set to train/test our segmented model (i.e., the "B+C" model) and the baseline model (i.e., the "B" model). On each fold, we first divide the training set into two groups: functions inside dependence clusters and functions outside dependence clusters. Then, we train the "B+C$_{in}$" model and the "B+C$_{out}$" model, respectively. We also divide the test set into two groups and subsequently use the "B+C$_{in}$" model and the "B+C$_{out}$" model to predict the probability of those functions that contain faults. After that, we combine the predicted values to derive the final predicted values to compute the performance indicators.

Based on F-measure predictive values, we use the Wilcoxon's signed-rank test to examine whether two models have a significant difference in their predictive effectiveness. Then, we use the Bonferroni correction method to adjust p-values to examine whether a difference is significant at the significance level of 0.05 [13].Furthermore, we use Cliff's δ to examine whether the magnitude of the difference between the prediction performances of two models is important from the viewpoint of practical application [14]. By convention, the magnitude of the difference is considered either trivial ($|\delta| < 0.147$), small (0.147-

0.33), moderate (0.33-0.474), or large ($|\delta| > 0.474$) [15] . From Table V, we find that the "B+C" models have larger F-measure values than the "B" model in all the five systems except in Gcc-core , and most of cliff's $|\delta|$ values are more than 0.147 except in Gcc-core .That is to say, the dependence cluster based importance metrics are useful for non-effort-aware prediction under within-project evaluation.

(2) ***RQ2.*** *In the scenario of cross-project defect prediction, are dependence cluster based metrics useful for non-effort-aware prediction?*

TABLE V: The experimental results for RQ1

Projects	"B"	"B+C"	%↑	$\|\delta\|$
Gstreamer1.0.0	0.139	**0.166**	**19.4%**	0.153✓
Glibc2.1.1	0.068	**0.180**	**164.7%**	0.997✓
Gimp2.8.0	0.065	**0.159**	**144.6%**	0.992✓
Gcc-core4.0.0	0.094	0.086	-8.5%	0.111
Bash3.2	0.187	**0.229**	**22.5%**	0.556✓
Average	0.111	**0.164**	68.5%	0.562

Cross-project defect prediction uses a predicted model trained on one project to predict defect in another projects [8]. From Table VI, we find that the "B+C" models have larger F-measure values than the "B" model, and the cliff's δ values are more than 0.147. That is to say, the dependence cluster based importance metrics are useful for non-effort-aware prediction under cross-project evaluation.

TABLE VI: The experimental results for RQ2

Source	Target	"B"	"B+C"	↑%	$\|\delta\|$
Gimp2.8.0	Glibc2.1.1	0.132	**0.135**	18.5%	**0.203**✓
	Gstreamer1.0.0	0.071	**0.075**		
	Gcc-core4.0.0	0.061	**0.067**		
	Bash3.2	0.067	0.062		
Glibc2.1.1	Gimp2.8.0	0.107	0.051		
	Gstreamer1.0.0	0.111	0.080		
	Gcc-core4.0.0	0.159	0.069		
	Bash3.2	0.027	0.012		
Gstreamer1.0.0	Gimp2.8.0	0.079	**0.159**		
	Gcc-core4.0.0	0.061	**0.106**		
	Bash3.2	0.021	**0.026**		
	Glibc2.1.1	0.132	**0.143**		
Gcc-core4.0.0	Gimp2.8.0	0.025	**0.042**		
	Bash3.2	0.011	**0.052**		
	Glibc2.1.1	0.014	**0.067**		
	Gstreamer1.0.0	0.101	0.099		
Bash3.2	Gimp2.8.0	0.021	**0.078**		
	Glibc2.1.1	0.023	**0.081**		
	Gstreamer1.0.0	0.013	**0.064**		
	Gcc-core4.0.0	0.059	**0.066**		

Overall, the above experimental results show that the non-effort-aware defect prediction capability of "B+C" model is better than that "B" model under the settings of within-project and cross-project prediction.

VI. DISCUSSION

In this section, we further discuss our findings. First, we analyze whether our conclusions will change if the potentially confounding effect of module size is excluded for the "B" and the "B+C" models. Then, we analyze whether we have similar conclusions if the class imbalanced method is used.

(1) Will our conclusions change if the potentially confounding effect of module size is excluded?

In our study, we did not take into account the potentially confounding effect of function size on the associations between those metrics with fault-proneness [16], when building a fault-proneness prediction model. Therefore, it is not readily known whether our conclusions will change if the potentially confounding effect of module size is excluded. In the following, we use the method proposed by Zhou et al. [16] to remove the confounding effect of module size and then rerun the analyses for RQ1 and RQ2. From Table VII, we find that the "B+C" models have larger F-measure values than the "B" model in all the five systems except in Gcc-core based on Within-Project Defect Prediction and most of cliff's δ values more than 0.147 except in Gcc-core. Table VIII, we find that the "B+C" models have most of larger F-measure values than the "B" model based on Cross-Project Defect Prediction ,and the cliff's δ values are more than 0.147. This indicates that "B+C" model still performs better than the "B" model.

TABLE VII: F-measure values after excluding the potentially confounding effect of module size: the "B" model vs "B+C" model based on Within-Project Defect Prediction

Projects	"B"	"B+C"	%↑	$\|\delta\|$
Gstreamer1.0.0	0.106	**0.162**	52.8%	0.875✓
Glibc2.1.1	0.070	**0.176**	151.4%	0.923✓
Gimp2.8.0	0.085	**0.155**	82.3%	0.728✓
Gcc-core4.0.0	0.084	0.072	-14.2%	0.187
Bash3.2	0.161	**0.221**	37.2%	0.421✓
Average	0.101	0.157	61.9%	0.627

TABLE VIII: F-measure values after excluding the potentially confounding effect of module size: the "B" model vs "B+C" model based on Cross-Project Defect Prediction

Source	Target	"B"	"B+C"	↑%	$\|\delta\|$
Gimp2.8.0	Glibc2.1.1	0.096	**0.123**	18.2%	0.303✓
	Gstreamer1.0.0	0.084	**0.101**		
	Gcc-core4.0.0	0.051	**0.092**		
	Bash3.2	0.043	**0.098**		
Glibc2.1.1	Gimp2.8.0	0.073	**0.091**		
	Gstreamer1.0.0	0.091	0.072		
	Gcc-core4.0.0	0.087	0.057		
	Bash3.2	0.082	**0.107**		
Gstreamer1.0.0	Gimp2.8.0	0.052	**0.087**		
	Gcc-core4.0.0	0.077	0.052		
	Bash3.2	0.042	0.031		
	Glibc2.1.1	0.081	**0.102**		
Gcc-core4.0.0	Gimp2.8.0	0.072	0.056		
	Bash3.2	0.052	**0.071**		
	Glibc2.1.1	0.081	0.067		
	Gstreamer1.0.0	0.103	0.071		
Bash3.2	Gimp2.8.0	0.041	**0.071**		
	Glibc2.1.1	0.057	**0.101**		
	Gstreamer1.0.0	0.036	**0.052**		
	Gcc-core4.0.0	0.043	**0.087**		

(2) Will our conclusions change if the class imbalanced method is used?

We did not take into account removing the imbalanced data in our study, when building a fault-proneness prediction model. Therefore, it is not readily known whether our conclusions will change if removing imbalanced data. In the following, we use the random under-sampling method proposed by Kamei et al. [17] to remove imbalanced data and then rerun the analyses for RQ1 and RQ2. From

Table IX, we find that the "B+C" models have larger F-measure values than the "B" model except in Gcc-core based on Within-Project Defect Prediction ,and most of cliff's δ values are more than 0.147 except in Gcc-core. Table X, we find that the "B+C" models have most of larger F-measure values than the "B" model, and the cliff's δ values are more than 0.147. This indicates that "B+C" model still performs better than the "B" model.

TABLE IX: F-measure values after removing the imbalanced data: the "B" model vs "B+C" model based on Within-Project Defect Prediction

Projects	"B"	"B+C"	%↑	$\lvert\delta\rvert$
Gstreamer1.0.0	0.177	**0.190**	7.34%	0.256✓
Glibc2.1.1	0.203	**0.276**	36.0%	0.421✓
Gimp2.8.0	0.168	**0.198**	17.9%	0.556✓
Gcc-core4.0.0	0.156	0.152	-2.6%	0.187
Bash3.2	0.122	**0.134**	9.8%	0.375✓
Average	0.165	**0.190**	13.7%	0.359

TABLE X: F-measure values after removing the imbalanced data: the "B" model vs "B+C" model based on Cross-Project Defect Prediction2

Source	Target	"B"	"B+C"	↑%	$\lvert\delta\rvert$
Gimp2.8.0	Glibc2.1.1	0.132	**0.207**	12.1%	0.215✓
	Gstreamer1.0.0	0.091	**0.196**		
	Gcc-core4.0.0	0.102	**0.201**		
	Bash3.2	0.097	**0.182**		
Glibc2.1.1	Gimp2.8.0	0.155	0.087		
	Gstreamer1.0.0	0.186	**0.205**		
	Gcc-core4.0.0	0.113	0.067		
	Bash3.2	0.105	**0.109**		
Gstreamer1.0.0	Gimp2.8.0	0.079	**0.127**		
	Gcc-core4.0.0	0.062	**0.102**		
	Bash3.2	0.061	**0.112**		
	Glibc2.1.1	0.131	**0.192**		
Gcc-core4.0.0	Gimp2.8.0	0.202	0.162		
	Bash3.2	0.167	0.134		
	Glibc2.1.1	0.236	0.195		
	Gstreamer1.0.0	0.232	0.195		
Bash3.2	Gimp2.8.0	0.160	**0.201**		
	Glibc2.1.1	0.203	0.171		
	Gstreamer1.0.0	0.177	**0.181**		
	Gcc-core4.0.0	0.103	**0.105**		

VII. CONCLUSION AND FUTURE WORK

In this paper, we investigate whether dependence clusters are useful for non-effort-aware defect prediction. We use the product, process, and network metrics to build the "B" model (baseline model), and use the product, process, network and dependence cluster metrics to build the "B+C" model. Our experimental results, based on five well-known open-source systems, show that the dependence clusters are useful for non-effort-aware defect prediction. In the future, we plan to build the model for dependence clusters at different granularities and examine their effectiveness.

References

[1] D. Binkley and M. Harman, "Locating dependence clusters and dependence pollution," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005, pp. 177–186.

[2] M. Harman, D. Binkley, K. Gallagher, N. Gold, and J. Krinke, "Dependence clusters in source code," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 32, no. 1, pp. 1–33, 2009.

[3] Y. Yang, M. Harman, J. Krinke, S. Islam, D. Binkley, Y. Zhou, and B. Xu, "An empirical study on dependence clusters for effort-aware fault-proneness prediction," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2016, pp. 296–307.

[4] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, and H. Leung, "Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 157–168.

[5] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2011.

[6] L. C. Briand, W. L. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," *IEEE transactions on Software Engineering*, vol. 28, no. 7, pp. 706–720, 2002.

[7] Y. Yang, Y. Zhou, H. Lu, L. Chen, Z. Chen, B. Xu, H. Leung, and Z. Zhang, "Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? an empirical study," *IEEE Transactions on Software Engineering*, vol. 41, no. 4, pp. 331–357, 2014.

[8] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the" imprecision" of cross-project defect prediction," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, 2012, pp. 1–11.

[9] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 432–441.

[10] S. Wasserman, K. Faust *et al.*, *Social network analysis: Methods and applications*. Cambridge university press, 1994, vol. 8.

[11] X. Chen, Y. Mu, Y. Qu, C. Ni, M. Liu, T. He, and S. Liu, "Do different cross-project defect prediction methods identify the same defective modules?" *Journal of Software: Evolution and Process*, 10 2019.

[12] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, 2008.

[13] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: a practical and powerful approach to multiple testing," *Journal of the Royal statistical society: series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995.

[14] 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.

[15] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys," in *annual meeting of the Florida Association of Institutional Research*, 2006, pp. 1–33.

[16] Y. Zhou, H. Leung, and B. Xu, "Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness," *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 607–623, 2009.

[17] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan, "Studying just-in-time defect prediction using cross-project models," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2072–2106, 2016.