An Empirical Study on Dependence Clusters for Effort-Aware Fault-Proneness Prediction

Yibiao Yang[1], Mark Harman[2], Jens Krinke[2], Syed Islam[3], David Binkley[4], Yuming Zhou[1]*, and Baowen Xu[1]

[1]Department of Computer Science and Technology, Nanjing University, China

[2]Department of Computer Science, University College London, UK

[3]School of Architecture, Computing and Engineering, University of East London, UK

[4]Department of Computer Science, Loyola University Maryland, USA

ABSTRACT

A dependence cluster is a set of mutually inter-dependent program elements. Prior studies have found that large dependence clusters are prevalent in software systems. It has been suggested that dependence clusters have potentially harmful effects on software quality. However, little empirical evidence has been provided to support this claim. The study presented in this paper investigates the relationship between dependence clusters and software quality at the function-level with a focus on effort-aware fault-proneness prediction. The investigation first analyzes whether or not larger dependence clusters tend to be more fault-prone. Second, it investigates whether the proportion of faulty functions inside dependence clusters is significantly different from the proportion of faulty functions outside dependence clusters. Third, it examines whether or not functions inside dependence clusters playing a more important role than others are more fault-prone. Finally, based on two groups of functions (i.e., functions inside and outside dependence clusters), the investigation considers a segmented fault-proneness prediction model. Our experimental results, based on five well-known open-source systems, show that (1) larger dependence clusters tend to be more fault-prone; (2) the proportion of faulty functions inside dependence clusters is significantly larger than the proportion of faulty functions outside dependence clusters; (3) functions inside dependence clusters that play more important roles are more fault-prone; (4) our segmented prediction model can significantly improve the effectiveness of effort-aware fault-proneness prediction in both ranking and classification scenarios. These findings help us better understand how dependence clusters influence software quality.

*Corresponding author: zhouyuming@nju.edu.cn.

ASE '16 September 3–7, 2016, Singapore, Singapore

© 2016 ACM. ISBN 123-4567-24-567/08/06.

DOI: 10.475/123_4

CCS Concepts

•**Software and its engineering → Abstraction, modeling and modularity;** Software development process management;

Keywords

Dependence clusters, fault-proneness, fault prediction, network analysis

1. INTRODUCTION

A dependence cluster is a set of program elements that all directly or transitively depend upon one another [8, 18]. Prior empirical studies found that large dependence clusters are highly prevalent in software systems and further complicate many software activities such as software maintenance, testing, and comprehension [8, 18]. In the presence of a (large) dependence cluster, an issue or a code change in one element likely has significant ripple effects involving the other elements of the cluster [8, 18]. Hence, there is a reason to believe that dependence clusters have potentially harmful effects on software quality. This suggests that the elements inside dependence clusters have relatively lower quality when compared to elements outside any dependence cluster. Given this observation, dependence clusters should be useful in fault-prediction. However, few empirical studies have investigated the effect of dependence clusters on fault-proneness prediction.

This paper presents an empirical study of the relationships between dependence clusters and fault-proneness. The concept of a dependence cluster was originally introduced by Binkley and Harman [8]. They treat program statements as basic units, however, they note that dependence clusters can be also defined at coarser granularities, such as at the function-level [7]. For a given program, the identification of function-level dependence clusters consists of two steps. The first step generates a function-level System Dependence Graph for all functions of the program. In general, these graphs involve two types of dependencies between functions: call dependency (i.e., one function calls another function) and data dependency (e.g., a global variable defined in one function is used in another function). In the System Dependence Graphs used in our study, nodes denote functions and directed edges denote the dependencies between these functions. In the second step, a clustering algorithm is used to calculate all the maximal strongly connected components found in the System Dependence Graph (SDG). In function-level dependence clusters, functions are regarded as the basic units and each cluster consists of at least two functions. According to Binkley et al. [7], the function-level dependence clusters can offer an effective proxy for the more expensive statement-level dependence clusters.

Based on this observation, we investigate dependence clusters at the function-level. Our main contributions are the following:

1) We investigate whether the qualities of dependence clusters are influenced by their size. Our results show that larger dependence clusters tend to be more fault-prone.

2) We examine whether functions inside dependence clusters are more fault-prone than functions outside dependence clusters. The results show that the proportion of faulty functions inside dependence clusters is significantly greater than that of functions outside all dependence clusters.

3) We examine whether functions playing more important roles inside dependence clusters are more fault-prone. Our empirical results show that importance metrics are positively correlated with fault-proneness.

4) Finally, we propose a segmented prediction model for fault-proneness prediction. More specifically, we build two different fault-proneness prediction models respectively for functions inside and functions outside dependence clusters. The empirical results show that our segmented prediction model can significantly improve the prediction performance in effort-aware evaluations.

The rest of this paper is organized as follows. In Section 2, we summarize related work. We present our research questions in Section 3. In Section 4, we describe the experimental setup, including the subject systems and the data collection method used. In Section 5, we describe the research method and report the detailed experimental results with respect to each of the research questions. Section 6 discusses our findings. In Section 7, we examine threats to validity. Finally, Section 8 concludes the paper and outlines directions for future work.

2. RELATED WORK

This section summarizes related work on dependence clusters and dependence analysis in fault-proneness prediction.

2.1 Dependence Clusters

Binkley and Harman [8] originally introduced the concept of dependence clusters based on program slicing at the statement level. They proposed a “same slice size” approach to identifying dependence clusters using the SDG. Later, Harman et al. [18] extended this initial study to include a larger set of programs. Their empirical results showed that the “same slice size” approach was extremely accurate. In addition, they found that large dependence clusters were surprisingly commonplace and consumed from more than 10% of the program to, in some cases, 80% of the whole program. Islam et al. [20, 21] introduced the concept of coherent dependence clusters. In a coherent dependence cluster, all elements depend upon the same set of elements and also affect a common set of elements. They used coherent dependence clusters to identify logical functionality within programs.

Binkley and Harman [10, 9] introduced a method to measure the effect of an SDG vertex or an edge on the formation of dependence clusters and then used this method to identify linchpins, which effectively hold a dependence cluster together. Their results showed that only a few vertices and edges act as linchpins. After that, Binkley et al. [10] introduced a simple transformation-based analysis algorithm to identify the impact of global variables on the presence of dependence clusters. Their results showed that over half of the studied programs include a global variable that was responsible for the formation of a dependence cluster.

Beszédes et al. [5, 6] conducted an empirical study into the properties of SEA-based dependence clusters. Such cluster are defined at the function-level and are based on the static execute after (SEA) relation. Their empirical results showed that SEA-based dependence clusters occur frequently in programs regardless of their domain and size. However, the SEA-based relation only considers call structure information. In other words, data dependencies are not considered in their study. In contrast, we take the data dependency between functions into account in our study.

Binkley et al. [7] compared the following two types of dependence clusters: slice-based dependence clusters at the statement-level and SEA-based dependence clusters at the function-level. They found that the less expensive SEA-based dependence clusters could be used as an effective proxy for the more expensive slice-based dependence clusters. Unlike the above studies, we investigate dependence clusters from the perspective of software quality. More specifically, we investigate whether dependence clusters have practical value in effort-aware fault-proneness prediction.

2.2 Dependence Analysis in Fault-Proneness Prediction

Zimmermann and Nagappan [41] calculated network metrics based on a dependence graph and used them to predict faults. More specifically, they first generate a SDG at the function level. Two kinds of dependencies between functions are then taken into account: call dependencies and data dependencies. They then lift this graph up to binary level since the defects were at the binary level. They considered the presence of dependencies without considering the multiplicity of dependencies. After that, they compute network measures on the dependence graph and then evaluated their fault-proneness prediction performance on Windows Server 2003. Their results show that the recall of the model built from network measures was 10% higher than the model built from complexity measures.

Ma et al. [23] conducted an empirical study to examine the effectiveness of network measures in the context of effort-aware fault-proneness prediction, taking into account the effort required to inspect predicted faulty module. They investigated dependence graphs at the file-level and did not consider the multiplicity of dependencies between files. They found that most network measures were of practical value in the context of effort-aware evaluations. Unlike these two studies, our SDGs are finer grained (i.e., function-level vs binary/file-level). In addition, we take into account the multiplicity of dependencies between functions.

Cataldo et al. [13] compared the relative impact of the syntactic, logical, and work dependencies on fault-proneness prediction. Syntactic dependencies are code dependencies (e.g., control and data dependencies). Logical dependencies focus on deducing dependencies between source code files that are changed together [16]. Finally, work dependencies account the human and organization of information [17, 19, 27, 29, 32]. Their work showed that the logical dependencies explained most of the variance in fault proneness while work flow dependencies had more impact than code dependencies. In our study, we only investigate syntactic dependencies, but do so at a finer granularity.

Oyetoyan et al. [31] studied the impact of cyclic dependencies on fault-proneness prediction. They found that most defects and defective components were concentrated in *cyclic dependent components*. The cyclic dependent components are those in call cycles in call dependence graph at the class level. These structures can also be viewed as dependence clusters. Our study is different from their study mainly with respect to the following aspects: 1) our dependence clustering is at a finer granularity (i.e., function level) while their study is at the file/class level; 2) we take into account more types of dependencies, including both call and data dependencies; 3) we study the fault-proneness prediction model in effort-aware evaluations with respect to ranking and classification scenarios; 4) we propose a segmented fault-proneness prediction model and compare our models with the traditional fault-proneness prediction models.

3. RESEARCH QUESTIONS

In this section we discuss our research questions and use the example SDG shown in Figure 1 to illustrate the questions. In Figure 1, the nodes (e.g., *f1* and *f2*) are functions and the directed edges are dependencies between functions, depicting data dependencies (labeled "d") and function call dependencies (labeled "c"), respectively. In this dependence graph, there are 15 functions and 3 dependence clusters (i.e., dc_1, dc_2, and dc_3). In Figure 1, dc_1, dc_2, and dc_3 are separate clusters since they are maximal strongly connected subgraphs. The functions are divided into two groups: functions inside dependence clusters and functions outside dependence clusters. Functions inside dependence clusters and functions outside dependence clusters form the subgraphs SubG$_{in}$ and SubG$_{out}$, respectively.

First, because a code change to one element of a dependence cluster likely ripples to the others elements of the cluster, our first research question (RQ1) investigates the relationship between the size of dependence clusters and fault-proneness:

RQ1. Are larger dependence clusters more fault-prone?

Second, functions in Figure 1 are classified into two groups: functions inside and outside dependence clusters. Our second research question (RQ2) focuses on the quality of functions in these two groups.

RQ2. Are functions inside dependence clusters more fault-prone than functions outside dependence clusters?

Third, functions inside dependence clusters form a sub-dependence graph (e.g., SubG$_{in}$ of Figure 1). Different functions play different roles in this sub-graph. Thus, we set up RQ3 for functions inside dependence clusters as follows:

RQ3. Are functions playing more important roles inside dependence clusters more fault-prone?

Finally, we aim to examine the usefulness of dependence clusters for fault-proneness prediction. Therefore, our last research question (RQ4) is set up as follows:

RQ4. Are dependence clusters useful in fault-proneness 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. Little is currently known on this subject. Our study attempts to fill this gap.

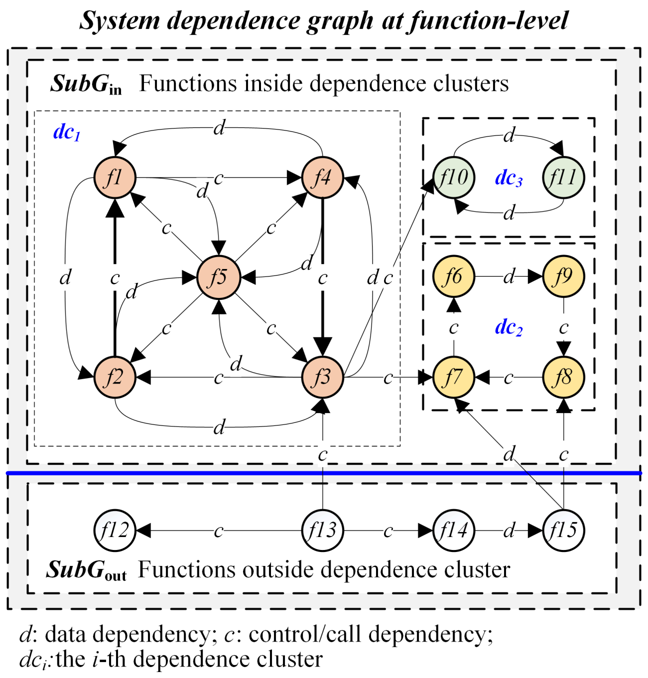

System dependence graph at function-level
SubG_in Functions inside dependence clusters
dc1
dc2
dc3
f1
f2
f3
f4
f5
f6
f7
f8
f9
f10
f11
f12
f13
f14
f15
d
c
SubG_out Functions outside dependence cluster
d: data dependency; c: control/call dependency;
dc_i: the i-th dependence cluster

Figure 1: An SDG with dependence clusters

4. EXPERIMENTAL SETUP

This section first introduces the systems studied before describing the procedure used to collect the experimental data.

4.1 Studied Projects

Table 1 summarizes the subjects used in the study. The first column is the system name. We use five well-known open-source projects as subject systems: Bash (BASH), gcc-core (GCC), GIMP (GIMP), glibc (GLIB), and GStreamer (GSTR). Bash is a command language interpreter, gcc-core is the GNU compiler collection, GIMP is the GNU Image Manipulation Program, glibc is the GNU Project's implementation of the C standard library and GStreamer is a multimedia framework. We chose these five projects as subjects for two reasons. First, they are well-known open source projects with a publicly available bug-fix history. In particular, the bug-fixing releases do not add any new features to the corresponding systems, thus allowing us to collect accurate fault data at the function level. For instance, gcc distribution website states "Note that starting with version 3.3.4, we provide bug releases for older release branches for those users who require a very high degree of stability". Second, they are non-trivial software systems belonging to several different domains. In Table 1, the second to the seventh columns are respectively the version number, the release date, the total source lines of code in the subject release, the number of functions, the number of faulty functions, and the percentage of faulty functions. The eighth and the ninth columns are the version number and the release date of the previous version used for computing the process metrics in Section 5.4. The last two columns are the version number and the release date of the fixing release.

Table 1: The subject systems

System	Subject release						Previous release		Fixing release	
	Version	Release date	Total SLoC	# functions	# faulty functions	% faulty functions	Version	Release date	Version	Release date
Bash	3.2	2006-10-11	49 608	1 947	68	3.49%	3.1	2005-12-08	3.2.57	2014-11-07
Gcc-core	4.0.0	2005-04-21	422 182	13 612	430	3.16%	3.4.0	2004-04-20	4.0.4	2007-01-31
Gimp	2.8.0	2012-05-12	557 436	19 978	818	4.10%	2.7.0	2009-08-15	2.8.16	2015-11-21
Glibc	2.1.1	1999-05-24	172 559	5 923	417	7.04%	2.0.1	1997-02-04	2.1.3	2000-02-25
Gstreamer	1.0.0	2012-09-24	75 985	3 946	146	3.70%	0.11.90	2011-08-02	1.0.10	2013-08-30

The subject projects are moderate to large-scale software systems (from 49 to 557 KSLOC). They have only a small number of faulty functions (from approximately 3% to 7% of all functions). Furthermore, on average, the fixing release comes out approximately 3 years after the subject version is released. We believe 3 years is sufficiently long for the majority of faulty functions to be identified and fixed.

4.2 Data Collection Procedure

We collected data from the above mentioned five projects. For each subject system, we obtained the fault data and identified dependence clusters for further analysis using the following steps. At the first step, we determined the faulty or not faulty label for each function. As mentioned before, any of the bug-fixing releases did not add any new features to the corresponding system. For each of the subject systems, we compared these versions with the latest bug-fixing releases (identified by the last two columns of Table 1) and determined which functions were changed. If a function was changed, it was marked as a faulty. Otherwise, it was marked as not-faulty. This method has been used to determine faulty functions before [42].

Our second step, collected the dependence clusters for each system using the Understand[1] tool and an R package igraph[2]. For each subject system, we first generated an Understand database. Then, we extracted the call and data dependencies for all functions from the generated database. In this way we obtained the SDG of the subject system. After that, we used the function *cluster* in igraph package to identify all dependence clusters. Each system's functions are divided into two groups: functions inside and functions outside dependence clusters.

Table 2: The dependence clusters in subject systems

System	# functions	# clusters	% functions inside clusters	Size of largest cluster
BASH	1 947	41	46.2	483
GCC	13 612	139	34.9	4083
GIMP	19 978	363	14.2	158
GLIB	5 923	105	11.6	277
GSTR	3 946	59	15.2	170

Table 2 describes the clusters in the subject projects. The third to the fifth columns respectively show the number of clusters, the percentage of functions inside clusters, and the size of the largest cluster in each subject project. From Table 2, we can see that there exist many dependence clusters (from 41 to 363) in these projects. Furthermore, from 11.6% to 46.2% of the total functions are found inside dependence clusters. Additionally, the size of the largest cluster in these projects varied from 158 to 4083. Of these five projects, GCC has the largest dependence cluster (that includes 4083 functions). This, to a certain extent, indicates that GCC is more complex than the other systems.

[1]https://scitools.com
[2]http://igraph.org/r/

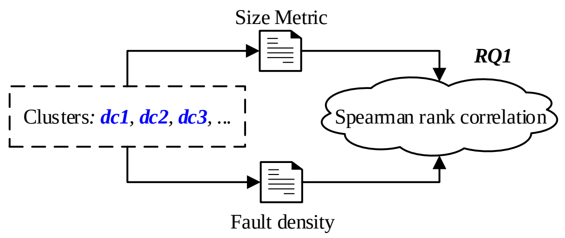


Figure 2: Overview of the analysis method for *RQ1*

5. METHODOLOGY AND RESULTS

In the section, we describe the research method and report the experimental results in detail with respect to each of the research questions.

5.1 RQ1. Are larger dependence clusters more fault-prone?

In the following, we describe the research method used and report the experimental result to address RQ1.

5.1.1 Research method

Figure 2 provides an overview of the analysis method used to address RQ1. As can be seen, in order to answer RQ1, we use Spearman's rank correlation to investigate the relationship between the size of dependence clusters and the fault density of dependence clusters. Here, fault density refers to the percentage of faulty functions in the dependence clusters. There are two basic metrics to measure the size of a graph: Size and Ties. Size is the number of functions within dependence clusters while Ties is the number of edges between functions in dependence clusters. In this study, we first use igraph to compute these two metrics for all dependence clusters in each subject system. We choose Spearman's rank correlation rather than Pearson's linear correlation since the former is a non-parametric method and makes no normality assumptions on variables [30]. According to Ott and Longnecker [30], for correlation coefficient rho, the correlation is considered either weak ($|rho| \leq 0.5$), moderate ($0.5 < |rho| < 0.8$), or strong ($0.8 \leq |rho| \leq 1.0$).

5.1.2 Experimental result

In the following, we describe the empirical results used to answer RQ1. Table 3 summarizes the Spearman correlation coefficients relating the size metrics with fault density of dependence clusters. In Table 3, the second column is the number of dependence clusters in each subject system. The third and the fifth columns respectively present the correlation coefficients for the Size and the Ties metrics from Spearman's singed-rank correlation. The correlation coefficients which are not statistically significant at the significance level of $\alpha = 0.05$ are marked in gray.

Table 3: **Spearman correlation for dependence clusters size and fault density ($RQ1$)**

System	# clusters	Size *rho*	Size *p*	Ties *rho*	Ties *p*
BASH	41	0.230	0.148	0.315	0.045
GCC	139	0.299	< 0.001	0.233	0.006
GIMP	363	0.150	0.004	0.195	< 0.001
GLIB	105	0.092	0.350	0.113	0.249
GSTR	59	0.345	0.007	0.295	0.023

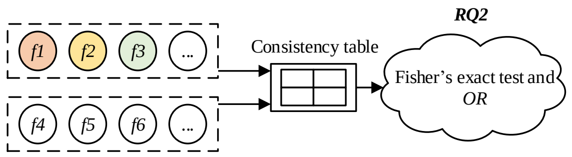


Figure 3: Overview of the analysis method for *RQ2*

In Table 3, we see that all the absolute values of the correlation coefficients are less than 0.5. This indicates that there is only a weak correlation between these two size metrics (i.e., Size and Ties) with fault density of dependence clusters. However, all the correlation coefficients are larger than 0 and most of them are statistically significant at the significance level of $\alpha = 0.05$. This indicates that these size metrics are positively correlated with fault density. In other words, larger dependence clusters tend to be more fault-prone. Thus, large dependence clusters are likely more harmful and hence should be avoided, advice that is consistent with prior studies [8, 18].

5.2 RQ2. Are functions inside dependence clusters more fault-prone than functions outside dependence clusters

In the following, we describe the research method and the experimental result answering RQ2.

5.2.1 *Research method*

Figure 3 provides an overview of the data analysis method for addressing RQ2. As can be seen, in order to answer RQ2, we use Fisher's exact test and the odds ratio (OR) to examine whether the proportion of faulty functions inside dependence clusters is statistically significantly different from the proportion of faulty functions outside dependence clusters. Fisher's exact test is a statistical significance test used in the analysis of contingency tables [36]. The contingency table is a matrix that displays the frequency distribution of variables. In our study, the contingency table has four types of functions: (1) functions inside dependence clusters that have faults; (2) functions inside dependence clusters that have no faults; (3) functions outside dependence clusters that have faults; and (4) functions outside dependence clusters that have no faults. The OR indicates the likelihood that an event (e.g., that a function is faulty) occurs [36]. Assume p is the proportion of faulty functions inside dependence clusters and q is the proportion of faulty functions outside dependence clusters. Then, OR is defined as $\frac{p/(1-p)}{q/(1-q)}$. Thus $OR > 1$ indicates that faults are more likely to occur inside dependence clusters. $OR = 1$ indicates an equal probability.

5.2.2 *Experimental result*

Table 4 summarizes the results of the comparison of the proportions of faulty functions inside and outside dependence clusters. In Table 4, the second and the third columns respectively represent the proportion of faulty functions inside and outside dependence clusters. The fourth and the fifth columns respectively show the Bonferroni adjusted p-value from Fisher's exact test and OR.

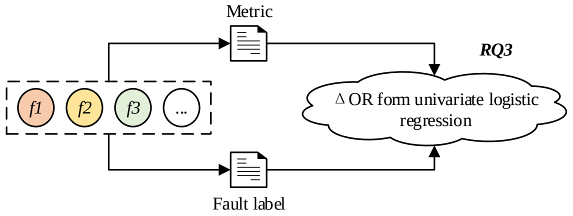


Figure 4: Overview of the analysis method for *RQ3*

Table 4: The proportion of faulty functions inside vs. outside dependence clusters ($RQ2$)

System	% functions is faulty inside	% functions is faulty outside	Fisher's exact test	OR
BASH	5.44%	1.82%	< 0.001	3.115
GCC	6.08%	1.59%	< 0.001	4.004
GIMP	4.47%	4.03%	1.000	1.115
GLIB	14.54%	6.06%	< 0.001	2.638
GSTR	10.20%	2.54%	< 0.001	4.361

From Table 4, we can see that the proportion of faulty functions inside dependence clusters is larger than the proportion of faulty functions outside dependence clusters in all cases, and significantly larger in all but one case. All the p-values are less than 0.05 except in GIMP which indicates statistically significant at the significance level of $\alpha = 0.05$. This indicates that the proportions of faulty functions between these two groups are significantly different. Meanwhile, all the ORs are substantially greater than 1, two are even greater than 4, which confirms the results from Fisher's exact test.

Overall, Fisher's exact test and the ORs consistently indicate that functions inside dependence clusters are more fault-prone than functions outside dependence clusters.

5.3 RQ3. Are functions playing more important roles inside dependence clusters more fault-prone?

In the following, we describe the corresponding research method and the experimental results that address RQ3.

5.3.1 *Research method*

Figure 4 provides an overview of the analysis method for RQ3. Functions inside dependence clusters form an independent dependence graph (e.g., SubG_{in} in Figure 1). In order to answer RQ3, we first use this graph to compute the importance metrics as described in Table 5 for the functions inside dependence clusters in the sub-dependence graph. The metrics in Table 5 are widely used networks metrics [37] that measure the extent to which these functions contribute to the sub-dependence graph. For example, the *Betweenness* metric for a vertex measures how many shortest paths pass through the vertex for all pairs of vertices of the subgraph. Thus, vertices with large Betweenness indicates a large importance. Note that some of these importance metrics can be computed by one the following three methods: "IN", "OUT", and "ALL". The "IN" method concerns all incoming edges. The "OUT" method concerns all outgoing edges. While the "ALL" method treats the graph as an undirected graph. In this study, we only compute the metrics using the "OUT" method.

Table 5: Summarization of the importance metrics

Metric	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

After that, we build univariate logistic regression models for each of these metrics with fault-proneness. Similar to prior studies [11, 38], we use ΔOR, the odds ratio associated with one standard deviation increase, to quantify the effect of these metrics on fault-proneness. ΔOR is defined as follows: $\Delta OR = e^{\beta \times \sigma}$. Here, β and σ are respectively the regression coefficient from the univariate logistic regression and the standard deviation of the variable. $\Delta OR > 1$ indicates that the corresponding metric is positively associated with fault-proneness while $\Delta OR < 1$ indicates a negative association.

5.3.2 Experimental result

Table 6 summarizes the ΔORs from univariate logistic regression analysis for the metrics of functions inside dependence clusters. In Table 6, the second and the third rows respectively show the number of functions and faulty functions inside dependence clusters for each subject system. After each ΔORs, "×" indicate the ΔORs is not statistically significant at a significance level of $\alpha = 0.05$. Note that, all the p-values are corrected by the Bonferroni correction method.

Table 6: Results from univariate analysis for the importance metrics of functions inside dependence clusters in terms of ΔOR ($RQ3$)

Metric	BASH	GCC	GIMP	GLIB	GSTR
N	900	4752	2839	688	598
# faulty functions	49	289	127	100	61
Betweenness	1.394	1.159	1.097 ×	0.876 ×	1.079 ×
Centr_betw	1.431	1.194	1.108 ×	0.949 ×	1.080 ×
Centr_clo	1.034 ×	1.257	0.957 ×	1.315	1.101 ×
Centr_degree	1.425	1.227	1.051 ×	1.314	1.223
Centr_eigen	1.013 ×	1.004 ×	1.106 ×	0.340 ×	0.947 ×
Closeness	1.035 ×	1.277	0.958 ×	1.310	1.102 ×
Constraint	0.716	0.705	1.039 ×	0.779	0.775 ×
Degree	1.425	1.227	1.051 ×	1.314	1.223
Eccentricity	0.901 ×	1.068 ×	0.998 ×	1.030 ×	0.963 ×
Page_rank	1.264 ×	1.037 ×	1.246	0.845 ×	1.127 ×

In Table 6, we see that the ΔORs of the Centr_degree and Degree metrics are larger than 1.0 in all systems. For other metrics, the ΔORs are larger than 1.0 in most systems. This indicates that they are positively associated with fault-proneness. Overall, this result indicates that functions that play a more important role in dependence clusters tend to be more fault-prone.

5.4 RQ4. Are dependence clusters useful in fault-proneness prediction?

In the following, we describe the research method and present the experimental results for RQ4.

5.4.1 Research method

Figure 5 provides an overview of the analysis method for RQ4. In order to address RQ4, we use AIC 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 and (2) the "B+C" model. The logistic regression is a standard statistical modeling technique in which the dependent variable can take on only one of two different values [3]. It is suitable and widely used for building fault-proneness prediction models [34, 33]. We choose the forward rather than the backward variant because the former is less time consuming on stepwise variable selection especially on a large number of independent metrics. AIC is a widely used variable selection criteria [33].

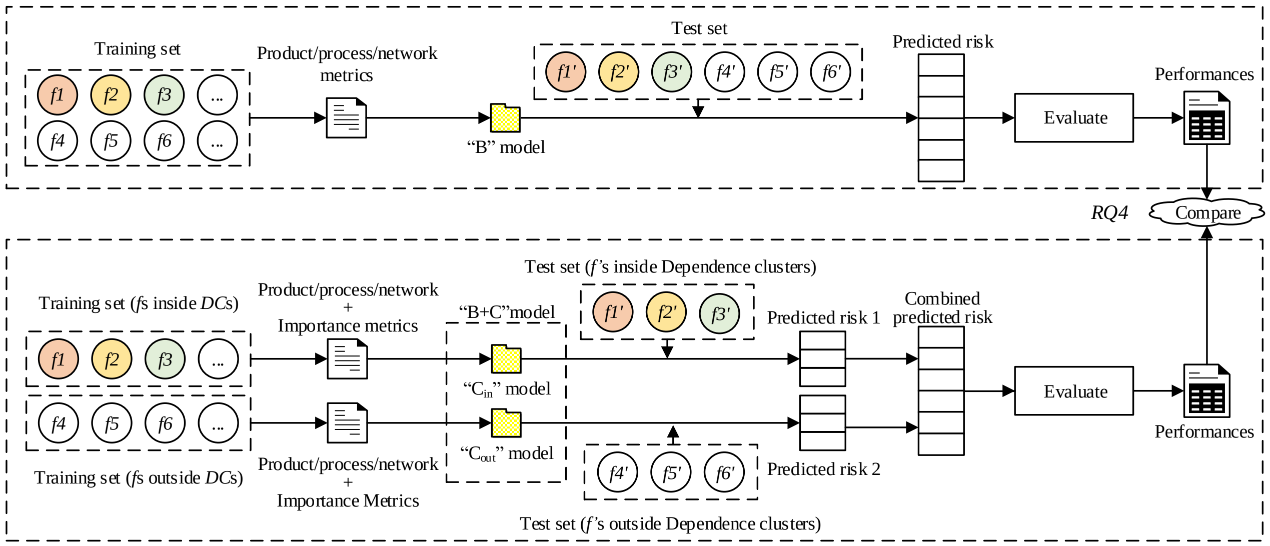
Table 7: The most commonly used product, process, and network metrics in this study

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

Table 8: Description of the studied network metrics

Metric	Description
Size	# alters that ego is directly connected to
Ties	# ties in the ego network
Pairs	# pairs of alters in the ego network
Density	% possible ties that are actually present
nWeakComp	# weak components in the ego network
pWeakComp	# weak components normalized by size
2StepReach	# nodes ego can reach within two steps
ReachEffic	2StepReach normalized by sum of alters' size
Broker	# pairs not directly connected to each other
nBroker	Broker normalized by the number of pairs
EgoBetw	% all shortest paths across ego
nEgoBetw	normalized EgoBetween (by ego size)
Effsize	# alters minus the average degree of alters
Efficiency	effsize divided by number of alters
Constraint	The extent to which ego is constrained
Degree	# nodes adjacent to a given node
Closeness	sum of the shortest paths to all other nodes
dwReach	# nodes that can be reached
Eigenvector	The influence of node in the network
Betweenness	# shortest paths through the vertex
Power	The connections of nodes in one's neighbors

(1) *The "B" model.* The "B" model is used as the baseline model, which is built with the most commonly used product, process, and network metrics. In this study, the product metrics consist of 16 metrics, including one code size metric, 11 complexity metrics, and 4 software science metrics. The process metrics consist of 3 code churn metrics [28]. The description for the product and the process metrics can be found in [38]. The network metrics consist of 21 network metrics, which are described in Table 8. We choose these metrics as the baseline metrics for the following reasons. First, the network analysis metrics are also computed from dependence graphs [41]. Second, these metrics are widely used and considered as useful indicators for fault-proneness prediction [25, 26, 28, 41]. Third, they can be cheaply collected from source code for large software systems.



Training set
f1
f2
f3
...
f4
f5
f6
...
Product/process/network metrics
"B" model
Test set
f1′
f2′
f3′
f4′
f5′
f6′
Predicted risk
Evaluate
Performances
RQ4
Compare
Training set (fs inside DCs)
Product/process/network + Importance metrics
"B+C"model
"C_in" model
"C_out" model
Test set (f′s inside Dependence clusters)
Predicted risk 1
Combined predicted risk
Evaluate
Performances
Training set (fs outside DCs)
Product/process/network + Importance Metrics
Predicted risk 2
Test set (f′s outside Dependence clusters)

Figure 5: Overview of the analysis method for *RQ4*

(2) *The "B+C" model.* The "B+C" model is our segmented model which consists of two independent models, i.e., the "B+C$_{in}$" model and the "B+C$_{out}$" model. The "B+C$_{in}$" and the "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 the most commonly used product/process/network metrics and the importance metrics described in Table 5. For the "B+C$_{in}$" model, the importance metrics are computed on the sub-dependence graph for functions inside dependence clusters (e.g., SubG$_{in}$ of Figure 1). While for the "B+C$_{in}$" model, the importance metrics are computed on the sub-dependence graph for functions outside dependence clusters (e.g., SubG$_{out}$ of Figure 1).

Note that, as mentioned in Section 5.3.1, some of the importance and the network metrics can be computed by the "IN", "OUT", or the "ALL" method. For the sake of simplicity, we only use the "OUT" method.

After building the "B" and the "B+C" models, we compare the prediction performance of the "B" model and the "B+C" model with respect to ranking and classification scenarios. In the following, we describe the performance indicators and the prediction settings, respectively.

(1) Performance indicators. In recent years, effort-aware performance measures have been widely used for evaluating the fault-proneness prediction models. The reason is that effort-aware measures take into account the effort required to inspect the predicted "faulty" functions and thus can provide a more realistic evaluation than non-effort-aware measures [24]. In this study, we thus compare the "B" and the "B+C" models in effort-aware evaluations. In greater detail, the predictive effectiveness is evaluated in the following two different scenarios: ranking and classification. In the ranking scenario, the functions are ranked in a descending order by the degree of their predicted relative risk. With such a ranking in hand, software project managers can easily select as many "high-risk" functions for inspecting or testing as available resources will allow. In the classification scenario, the functions are first classified into two categories according to their predictive relative risk: high-risk and low-risk. The functions that are predicted as high risk will be focused on for software quality enhancement. Following previous work [38], we also use SLOC in a function f as the proxy of the effort required to inspect or test the function and define the relative risk of function f as $R(f) = Pr/SLOC(f)$, where Pr is the predicted probability of function f being faulty. In other words, $R(f)$ represents the predicted fault-proneness per SLOC. In the following, we describe the effort-aware predictive performance indicators used in this study with respect to ranking and classification.

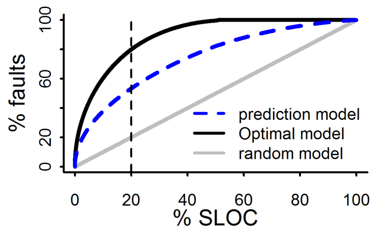


Figure 6: SLOC-based Alberg diagram

Ranking. We use CE, which is the cost-effectiveness measure proposed by Arisholm et al. [2] to evaluate the effort-aware ranking effectiveness of a fault-proneness prediction model. The CE measure is based on the concept of the "SLOC-based" Alberg diagram. In this diagram, the x-axis and y-axis are respectively the cumulative percentages of SLOC of the functions and the cumulative percentage of faults found in selected from the function ranking list. Figure 6 is an example "SLOC-based" Alberg diagram showing the ranking performances of a prediction model m (in our context, the prediction model m could be the "B" model and the "B+C" model). To compute CE, we also include two additional curves: the "random" model and the "optimal" model. In the "random" model, functions are randomly selected to inspect or test. In the "optimal" model, functions are sorted in decreasing order according to their actual fault densities. Based on this diagram, the effort-aware ranking effectiveness of the prediction model m is defined as follows [2]:

$$CE_{\pi}(m) = \frac{Area_{\pi}(m) - Area_{\pi}(random)}{Area_{\pi}(optimal) - Area_{\pi}(random)}$$

Here, $Area_{\pi}(m)$ is the area under the curve corresponding to model m for a given top 100% percentage of SLOC. The cut-off value π varies between 0 and 1, depending on the amount of available resource for inspecting functions.

As aforementioned, practitioners are more interested in the ranking performance of a prediction model at the top fraction. In this study, we use the CE at the cut-off $\pi = 0.2$ (indicated as $CE_{0.2}$) to evaluate the effort-aware ranking performance of a model.

Classification. We use Effort Reduction in Amount (ERA), a classification performance indicator adapted from the "ER" measure used by Zhou et al. [40], to evaluate the effort-aware classification effectiveness of a fault-proneness prediction model. In the classification scenario, only those functions predicted to be high-risk will be inspected or tested for software quality enhancement. The ERA measure denotes the amount of the reduced SLOC (i.e., the amount of effort reduction) to be inspected by a model m compared with the random model that achieves the same recall of faults. Therefore, the effort-aware classification effectiveness of the prediction model m can be formally defined as follows:

Here, Effort(m) is the ratio of the total SLOC in those predicted faulty functions to the total SLOC in the system. Effort(random) is the ratio of SLOC to inspect or test to the total SLOC in the system that a random selection model needs to achieve the same recall of faults as the prediction model m. In this paper, for the sake of simplicity, we use $ERA_{0.2}$ to evaluate the effort-aware classification performance. In order to compute $ERA_{0.2}$, we first use the predicted fault-proneness by the model to rank the modules in descending order. Then, we classify the top 20% modules into the fault-prone category and the other 80% modules into the defect-free category. Finally, we compute the effort-aware classification performance ERA as $ERA_{0.2}$. Here, we use 20% as the cut-off value because many studies show that the distribution of fault data in a system generally follow the Pareto principle [1, 15]. The Pareto principle, also known as the 20-80 rule, states that for many phenomena, 80 percent of the consequences stem from 20 percent of the causes [22]. In our context, this means that by inspecting these 20% predicted fault-prone functions, we expect that almost 80% of faulty modules in a system will be found.

(2) Prediction settings. To obtain a realistic comparison, we evaluate the prediction performance under 30 times 3-fold cross-validation. We choose 3-fold cross-validation rather than 10-fold cross-validation due to the small percentage of faulty function in the data sets. At each 3-fold cross-validation, we randomize and then divide the data set into 3 parts of approximately equal size. Then, we test each part by the prediction model built with the remainder of the data set. This process is repeated 30 times to alleviate potential sampling bias. Note that, for each fold of the 30 times 3-fold cross-validation, 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 these predictive effectiveness values, we use the Wilcoxon's signed-rank test to examine whether two models have a significant difference in their predictive effectiveness.

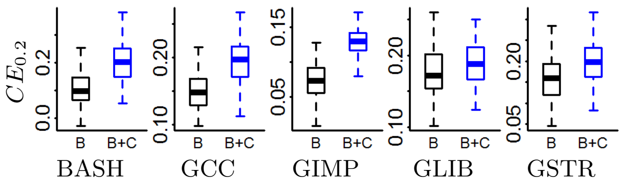


Figure 7: Ranking performance comparison for the "B" and the "B+C" model in terms of $CE_{0.2}$

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 [4]. 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 [2]. Cliff's δ is widely used for median comparison. By convention, the magnitude of the difference is considered either trivial ($|\delta| < 0.147$), small ($0.147 \sim 0.33$), moderate ($0.33 \sim 0.474$), or large (> 0.474) [35].

5.4.2 Experimental result

This section presents the results with respect to ranking and classification scenarios to answer RQ4.

(1) *Ranking performance comparison*

Figure 7 employs the box-plot to describe the distributions of $CE_{0.2}$ obtained from 30 times 3-fold cross-validation for the "B" and the "B+C" models with respect to each of the subject systems. For each model, the box-plot shows the median (the horizontal line within the box), the 25th percentile (the lower side of the box), and the 75th percentile (the upper side of the box). In Figure 7, a blue box indicates that (1) the corresponding "B+C" model performs significantly better than the "B" model according to the p-values from Wilcoxon signed-rank test; and (2) the magnitude of the difference between the corresponding "B+C" model and the "B" is not trivial according to Cliff's δ (i.e. $|\delta| \geq 0.147$).

Table 9: Ranking comparison in terms of $CE_{0.2}$: the "B" model vs the "B+C" model

System	B	B+C	%↑	$\|\delta\|$
BASH	0.098	**0.201**	104.90%	0.688 √
GCC	0.148	**0.197**	33.00%	0.714 √
GIMP	0.073	**0.130**	78.70%	0.938 √
GLIB	0.172	**0.188**	9.40%	0.194 √
GSTR	0.160	**0.198**	24.00%	0.426 √
Average	0.130	**0.183**	50.00%	0.592

From Figure 7, it is obvious that the "B+C" model performs substantially better than the "B" model in each of the subject systems.

Table 9 presents median $CE_{0.2}$ for the "B" and the "B+C" models. In Table 9, the second and the third columns present the median $CE_{0.2}$" respectively for the B and the "B+C" model. The fourth and the fifth column are respectively the percentage of the improvement for the "B+C" model over the "B" model and the effect sizes in terms of the Cliff's δ. In the last column, "√" indicates that the "B+C" model has significantly larger median $CE_{0.2}$ than the "B" model by the Wilcoxon's signed-rank test. The last row in Table 9 shows the average values for the five projects.

From Table 9, we have the following observations. For all systems, the "B+C" model has a larger median $CE_{0.2}$ than the "B" model in terms of the median $CE_{0.2}$. On average, the "B+C" model leads to about 50.0% improvement over

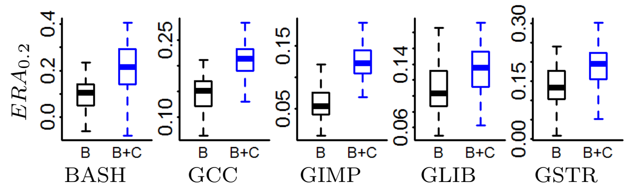


Figure 8: Classification performance comparison for the "B" and the "B+C" model in terms of $ERA_{0.2}$

the "B" model in terms of the median $CE_{0.2}$. The Wilcoxon signed-rank test p-values are very significant (< 0.001). Furthermore, the effect sizes are moderate to large except in GLIB where the effect size is small. The core observation is that, from the viewpoint of practical application, the "B+C" model has a substantially better ranking performance than the "B" model.

(2) ***Classification performance comparison***

Figure 8 employs box-plots to describe the distributions of $ERA_{0.2}$ obtained from 30 times 3-fold cross-validation for the "B" and the "B+C" models with respect to each of the subject systems. From Figure 8, we can find that the "B+C" models are also substantially better than the "B" model.

Table 10: Classification comparison in term of $ERA_{0.2}$: the "B" model vs the "B+C" model

System	B	B+C	%↑	$\|\delta\|$
BASH	0.098	**0.211**	115.20%	0.644 ✓
GCC	0.148	**0.210**	41.80%	0.816 ✓
GIMP	0.059	**0.124**	112.60%	0.940 ✓
GLIB	0.110	**0.135**	22.80%	0.462 ✓
GSTR	0.132	**0.189**	43.30%	0.551 ✓
Average	0.109	**0.174**	67.10%	0.683

Table 10 presents the classification performance for the "B" and the "B+C" models in terms of $CE_{0.2}$. For all systems, the "B+C" model has a larger median $ERA_{0.2}$ than the "B" model in terms of the median $ERA_{0.2}$. On average, the "B+C" model leads to about 67.1% improvement over the "B" model. The p-values are very significant (< 0.001). Besides, the effect sizes are moderate to large. The core observation is that, from the viewpoint of practical application, the "B+C" model has a substantially better classification performance than the "B" model.

Overall, the above observations suggest that the "B+C" model outperforms the "B" model in effort-aware fault-proneness prediction under both ranking and classification scenarios. This indicates that dependence clusters are actually useful in effort-aware fault-proneness prediction.

6. 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 multiplicity of dependencies is not considered.

6.1 Will our conclusions change if the potentially confounding effect of module size is excluded?

In our study, when building a fault-proneness prediction model, we did not take into account the potentially confounding effect of function size on the associations between those metrics with fault-proneness [14, 39]. 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. [39] to remove the confounding effect of module size and then rerun the analyses for RQ4.

Table 11: Ranking comparison in terms of $CE_{0.2}$ after excluding the potentially confounding effect of module size: the "B" model vs the "B+C" model

System	B	B+C	%↑	$\|\delta\|$
BASH	0.117	0.094	-0.194	0.224
GCC	0.150	**0.174**	0.160	0.520 ✓
GIMP	0.073	**0.131**	0.799	0.928 ✓
GLIB	0.183	**0.188**	0.025	0.041
GSTR	0.155	**0.187**	0.209	0.399 ✓
Average	0.136	**0.155**	0.200	0.333

Table 12: Classification comparison in terms of $ERA_{0.2}$ after excluding the potentially confounding effect of module size: the "B" model vs the "B+C" model

System	B	B+C	%↑	$\|\delta\|$
BASH	0.128	0.109	-15.40%	0.136
GCC	0.148	**0.196**	31.80%	0.728 ✓
GIMP	0.059	**0.128**	118.80%	0.948 ✓
GLIB	0.112	**0.135**	20.70%	0.446 ✓
GSTR	0.128	**0.171**	33.40%	0.444 ✓
Average	0.115	**0.148**	37.90%	0.486

Table 11 and Table 12 respectively present the median $CE_{0.2}$ and $ERA_{0.2}$ for the "B" and the "B+C" models after excluding the potentially confounding effect of module size. From Table 11 and Table 12, we find that the "B+C" models have both larger median $CE_{0.2}$ and median $ERA_{0.2}$ than the "B" model in all the five subject systems except in BASH. This indicates that our proposed model still performs better than the baseline model in both of the ranking and classification scenarios in most cases.

Overall, after excluding the potentially confounding effect of function size, our conclusion on RQ4 is mostly the same.

6.2 Will our conclusions change if the multiplicity of dependencies is ignored?

As mentioned before, in our study we take into account the multiplicity of dependencies between functions. The multiplicity information is used as the weight of dependencies in the SDG. However, prior studies [23, 31, 41] ignored this information. Therefore, it is not readily answerable whether our conclusions will change if the multiplicity of dependencies is also ignored. Next, we ignore the multiplicity of dependencies and rerun the analysis for RQ4.

Table 13 and Table 14 respectively summarize the median $CE_{0.2}$ and the median $ERA_{0.2}$ for the "B" and the "B+C" models when the multiplicity of dependencies is not considered. From Table 13 and Table 14, we observe that the "B+C" models have substantially larger median $CE_{0.2}$ and median $ERA_{0.2}$ than the "B" model in all the five subject systems. This indicates that our proposed model still performs substantially better than the baseline model in both of the ranking and classification scenarios.

Overall, the above observations show that our conclusions on RQ4 remain unchanged if the multiplicity of dependencies is not considered.

Table 13: Ranking comparison in terms of $CE_{0.2}$ when the multiplicity of dependencies is not considered: the "B" model vs the "B+C" model

System	B	B+C	%↑	$\|\delta\|$
BASH	0.142	**0.187**	32.20%	0.313 √
GCC	0.159	**0.196**	23.30%	0.576 √
GIMP	0.070	**0.134**	92.60%	0.986 √
GLIB	0.172	**0.179**	4.50%	0.167 √
GSTR	0.160	**0.192**	20.60%	0.378 √
Average	0.140	**0.178**	34.60%	0.484

Table 14: Classification comparison in terms of $ERA_{0.2}$ when the multiplicity of dependencies is not considered: the "B" model vs the "B+C" model

System	B	B+C	%↑	$\|\delta\|$
BASH	0.157	**0.202**	28.80%	0.261 √
GCC	0.163	**0.203**	24.50%	0.605 √
GIMP	0.058	**0.135**	132.90%	0.994 √
GLIB	0.100	**0.122**	22.20%	0.402 √
GSTR	0.131	**0.184**	40.10%	0.501 √
Average	0.122	**0.169**	49.70%	0.552

7. THREATS TO VALIDITY

This section analyzes the most important threats to the construct, internal, and external validity of our study.

7.1 Construct Validity

There are two potential threats to the construct validity of our study. The first threat concerns the fault data. In our study, we collected fault data by comparing the latest bug-fixing version with the investigated version for each system. Bug-fixing version did not add new features to the corresponding systems. Thus, the construct validity of the fault data can be considered acceptable. The second threat concerns the method we used to compute the importance and the network metrics. In our study, we use the "OUT" method to compute those metrics which only concerns the outgoing degree. In order to address this threat, we recomputed those metrics by using the other two methods and reran the analysis for RQ3, and RQ4. We found that the results were very similar.

7.2 Internal Validity

There are three possible threats to the internal validity of our study. The first threat is the unknown effect of the method to define the relative risk of a function in RQ4. In our study, we use the ratio of the predicted value from a naïve logistic regression model to the functions SLOC as the relative risk for each function. However, in the literature, most studies use the predicted value from the naïve logistic regression model as the relative risk of a function. In order to eliminate this threat, we reran the analysis for RQ4 by using the predicted value from the naïve logistic regression model as the relative risk. We found that the relative performance of the "B" model and the "B+C" model is not changed. That is to say, the "B+C" model is still significantly better than the "B" model. The second threat is from the specific cut-off value used for the performance indicator (i.e. the CE and ERA). In our study, 0.2 is used as the cut-off value for the computation of CE and ERA. To eliminate this potential threat, we rerun all the analyses using the following typical cut-off values: 0.10 and 0.30. We found our conclusion remains unchanged. The third threat is the unknown effect of the method for the stepwise variable selection in RQ4. In our study, we use AIC as the criteria to perform the stepwise variable selection. BIC is also a widely used method to perform stepwise variable selection [12]. We reran the analysis for RQ4 using BIC as the criteria to perform variable selection and found the results to be very similar.

7.3 External Validity

Our experiments are based on five long-lived and widely used open-source C systems. The most important threat to the external validity of this study is that our findings may not be generalized to other systems, especially closed-source systems. The second threat to the external validity of this study is that our findings are restricted to only one language. These external threats are ever present in any empirical study concerning program analysis, we hope that researchers will replicate our study across a wide variety of systems in the future.

8. CONCLUSIONS AND FUTURE WORK

In this study, we perform an extensive study to examine the relationships between function-level dependence clusters and fault-proneness. Our findings from five widely used industrial-size systems show that (1) larger dependence clusters tend to be more fault-prone; (2) functions inside dependence clusters tend to be more fault-prone than functions outside dependence clusters; (3) functions that play more important roles in dependence clusters are more fault-prone; (4) our segmented prediction model can significantly improve the performance in effort-aware fault-proneness prediction. These results consistently suggest that (1) large dependence clusters in software systems should be avoided; (2) when performing code refactoring, we should pay more attention to dependence clusters. These results provide valuable data for better understanding the properties of dependence clusters and its effect on software quality.

This study focuses on function-level dependence cluster for open-source C software systems. As future work, we plan to replicate our experiments for dependence clusters at different granularities (e.g., statement level) and on systems written in other languages and other programming paradigms.

9. REPEATABILITY

We provide all data sets and R scripts that used to conduct this study at http://ise.nju.edu.cn/yangyibiao/dc.html

10. ACKNOWLEDGMENTS

This work is supported by the National Key Basic Research and Development Program of China (2014CB340702), the National Natural Science Foundation of China (61432001, 91418202, 61272082, 61300051, 61321491, 61472178, and 91318301), the Natural Science Foundation of Jiangsu Province (BK20130014), and the program A for Outstanding PhD candidate of Nanjing University. Harman and Krinke are part-supported by the EPSRC DAASE programme grant. Binkley is supported by NSF grant IIA-1360707 and a J. William Fulbright award.

11. REFERENCES

[1] C. Andersson and P. Runeson. A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems. *IEEE Transactions on Software Engineering*, 33(5):273–286, May 2007.

[2] 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*, 83(1):2–17, Jan. 2010.

[3] V. Basili, L. Briand, and W. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, Oct. 1996.

[4] 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)*, 57(1):289–300, Jan. 1995.

[5] Á. Beszédes, L. Schrettner, B. Csaba, T. Gergely, J. Jász, and T. Gyimóthy. Empirical investigation of SEA-based dependence cluster properties. In *Proceedings of the 2013 IEEE International Working Conference on Source Code Analysis and Manipulation*, SCAM '12, pages 1–10, Sept. 2013.

[6] Á. Beszédes, L. Schrettner, B. Csaba, T. Gergely, J. Jász, and T. Gyimóthy. Empirical Investigation of SEA-based Dependence Cluster Properties. *Sci. Comput. Program.*, 105(C):3–25, July 2015.

[7] D. Binkley, Á. Beszédes, S. Islam, J. Jász, and B. Vancsics. Uncovering dependence clusters and linchpin functions. In *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution*, (ICSME' 15, pages 141–150, Sept. 2015.

[8] D. Binkley and M. Harman. Locating dependence clusters and dependence pollution. In *Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005. ICSM'05*, pages 177–186, Sept. 2005.

[9] D. Binkley and M. Harman. Identifying 'Linchpin Vertices' That Cause Large Dependence Clusters. In *Proceedings of the 2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*, SCAM '09, pages 89–98, Washington, DC, USA, 2009. IEEE Computer Society.

[10] D. Binkley, M. Harman, Y. Hassoun, S. Islam, and Z. Li. Assessing the impact of global variables on program dependence and dependence clusters. *Journal of Systems and Software*, 83(1):96–107, Jan. 2010.

[11] L. C. Briand, J. Wüst, J. W. Daly, and D. Victor Porter. Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of Systems and Software*, 51(3):245–273, May 2000.

[12] K. P. Burnham and D. R. Anderson. Multimodel Inference Understanding AIC and BIC in Model Selection. *Sociological Methods & Research*, 33(2):261–304, Nov. 2004.

[13] M. Cataldo, A. Mockus, J. Roberts, and J. Herbsleb. Software Dependencies, Work Dependencies, and Their Impact on Failures. *IEEE Transactions on Software Engineering*, 35(6):864–878, Nov. 2009.

[14] K. El Emam, S. Benlarbi, N. Goel, and S. Rai. The confounding effect of class size on the validity of object-oriented metrics. *IEEE Transactions on Software Engineering*, 27(7):630–650, July 2001.

[15] N. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering*, 26(8):797–814, Aug. 2000.

[16] H. Gall, K. Hajek, and M. Jazayeri. Detection of Logical Coupling Based on Product Release History. In *Proceedings of the International Conference on Software Maintenance*, ICSM '98, pages 190–, Washington, DC, USA, 1998. IEEE Computer Society.

[17] R. E. Grinter, J. D. Herbsleb, and D. E. Perry. The Geography of Coordination: Dealing with Distance in R&D Work. In *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work*, GROUP '99, pages 306–315, New York, NY, USA, 1999. ACM.

[18] M. Harman, D. Binkley, K. Gallagher, N. Gold, and J. Krinke. Dependence Clusters in Source Code. *ACM Trans. Program. Lang. Syst.*, 32(1):1:1–1:33, Nov. 2009.

[19] J. D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, 29(6):481–494, June 2003.

[20] S. Islam, J. Krinke, D. Binkley, and M. Harman. Coherent clusters in source code. *Journal of Systems and Software*, 88:1–24, Feb. 2014.

[21] S. S. Islam, J. Krinke, D. Binkley, and M. Harman. Coherent Dependence Clusters. In *Proceedings of the 9th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, PASTE '10, pages 53–60, New York, NY, USA, 2010. ACM.

[22] J. M. Juran. Quality control handbook. In *Quality control handbook*. McGraw-Hill, 1962.

[23] W. Ma, L. Chen, Y. Yang, Y. Zhou, and B. Xu. Empirical analysis of network measures for effort-aware fault-proneness prediction. *Information and Software Technology*, 69:50–70, Jan. 2016.

[24] T. Mende and R. Koschke. Effort-Aware Defect Prediction Models. In *Proceedings of the 2010 14th European Conference on Software Maintenance and Reengineering*, CSMR '10, pages 107–116, Washington, DC, USA, 2010. IEEE Computer Society.

[25] T. Menzies, J. Greenwald, and A. Frank. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, Jan. 2007.

[26] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener. Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engineering*, 17(4):375–407, May 2010.

[27] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, Apr. 2000.

[28] N. Nagappan and T. Ball. Use of Relative Code Churn Measures to Predict System Defect Density. In *Proceedings of the 27th International Conference on*

Software Engineering, ICSE '05, pages 284–292, New York, NY, USA, 2005. ACM.

[29] N. Nagappan, B. Murphy, and V. Basili. The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 521–530, New York, NY, USA, 2008. ACM.

[30] R. Ott and M. Longnecker. *An Introduction to Statistical Methods and Data Analysis*. Cengage Learning, Dec. 2008.

[31] T. D. Oyetoyan, D. S. Cruzes, and R. Conradi. A study of cyclic dependencies on defect profile of software components. *Journal of Systems and Software*, 86(12):3162–3182, Dec. 2013.

[32] M. Pinzger, N. Nagappan, and B. Murphy. Can Developer-module Networks Predict Failures? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, pages 2–12, New York, NY, USA, 2008. ACM.

[33] F. Rahman and P. Devanbu. How, and Why, Process Metrics Are Better. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 432–441, Piscataway, NJ, USA, 2013. IEEE Press.

[34] 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*, FSE '12, pages 61:1–61:11, New York, NY, USA, 2012. ACM.

[35] 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's d for evaluating group differences on the nsse and other surveys. In *annual meeting of the Florida Association of Institutional Research*, pages 1–33, 2006.

[36] D. J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures: Third Edition*. CRC Press, Aug. 2003.

[37] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, Nov. 1994.

[38] 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*, 41(4):331–357, Apr. 2015.

[39] 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*, 35(5):607–623, 2009.

[40] Y. Zhou, B. Xu, H. Leung, and L. Chen. An In-depth Study of the Potentially Confounding Effect of Class Size in Fault Prediction. *ACM Trans. Softw. Eng. Methodol.*, 23(1):10:1–10:51, Feb. 2014.

[41] T. Zimmermann and N. Nagappan. Predicting Defects Using Network Analysis on Dependency Graphs. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 531–540, New York, NY, USA, 2008. ACM.

[42] T. Zimmermann, R. Premraj, and A. Zeller. Predicting Defects for Eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, PROMISE '07, pages 9–, Washington, DC, USA, 2007. IEEE Computer Society.