PAPER

Assessing the Bug-Prediction with Re-Usability Based Package Organization for Object Oriented Software Systems

Mohsin SHAIKH[†], **Ki-Seong LEE**[†], ***Nonmembers***, ***and*** **Chan-Gun LEE**[†a)], ***Member***

SUMMARY Packages are re-usable components for faster and effective software maintenance. To promote the re-use in object-oriented systems and maintenance tasks easier, packages should be organized to depict compact design. Therefore, understanding and assessing package organization is primordial for maintenance tasks like Re-usability and Changeability. We believe that additional investigations of prevalent basic design principles such as defined by R.C. Martin are required to explore different aspects of package organization. In this study, we propose package-organization framework based on reachable components that measures re-usability index. Package re-usability index measures common effect of change taking place over dependent elements of a package in an object-oriented design paradigm. A detailed quality assessment on different versions of open source software systems is presented which evaluates capability of the proposed package re-usability index and other traditional package-level metrics to predict fault-proneness in software. The experimental study shows that proposed index captures different aspects of package-design which can be practically integrated with best practices of software development. Furthermore, the results provide insights on organization of feasible software design to counter potential faults appearing due to complex package dependencies.

key words: *package reuse, software quality, fault-proneness prediction*

1. Introduction

Software development typically requires intensive human expertise to develop techniques and tools that promote quality of applications. Package organization in Object Oriented (OO) design follows decomposition of relevant source code to simplify the development and maintenance. In order to understand the OO software, flexible design with well-connected constituent components is highly demanded for accommodating future changes and requirements. Most non-trivial software systems are modularized on different levels of abstraction employing different coding techniques. Over the years classes were considered as basic structural source code units for specified tasks. However, due to increasing complexity of classes and their inherent structural deteriorating nature, packages may serve as integral and functional components of software applications. Commercial applications are developed using large chunk of code that require exhaustive maintenance, re-engineering and refactoring effort. Conception of functional units within the software systems have reformed due to inclusive integration of new design principles. To facilitate software understanding and maintenance, interrelated classes should be allocated into groups called packages. In accordance with modern design principles [1], a package is to be formulated based on highly inter-related (cohesive) source code entities. A highly cohesive package is likely to be easier to understand, modify and maintain in comparison to less cohesive package. Thus, highly cohesive packages having an effect over re-usability and maintainability of software systems show the great need of further research [2]. On the contrary, even for highly cohesive packages, effect of change propagation should be managed properly taking into account package dependencies. As the software evolves over the time, modification, addition and removal of classes may influence inter-package dependencies in an adverse manner. Consequently, design quality gradually drifts due to misplacement of classes within a package and re-structuring of packages [3], [4]. In an OO design scenario, there are complicated structural dependencies among and within the packages. Thus, external dependencies attributed towards the packages also account for any significant change in modularization. Prediction of fault-prone entities in software systems is an important process to asses the quality of source code [5]. Despite the importance of package-level design paradigms and its effect on software quality, there has not been much effort to evaluate this subject quantitatively. Nevertheless, the conceptual foundation of package modularization provided by Sarkar *et al.* and Abdeen *et al.* is healthy motivation of research in this area [6], [7]. And more recently, empirical analysis of package-modularization and its implications over software fault-proneness prediction by Zhao *et al* is indeed in line with our research direction [8].

This research study is composed of two parts. The first part proposes a Package Re-usability *PkgReuse* index that follows the principles of good software modularity as explained by Martin [9]. This index measures strength of software re-usability in an OO design, essentially classifying the effect of change propagation over a package. Furthermore, proposed index adopts two-dimensional measurement of components within the package; Intra-package dependencies are calculated to measure the extent to which classes and interfaces within a packages are related to each other, Inter-package dependencies are calculated to measure the extent to which all dependent packages are affected due to internal change. In the second part, analysis of fault-proneness prediction with our Package Re-usability index metric and other recognized package-level metrics defined

Manuscript received May 2, 2016.
Manuscript revised August 24, 2016.
Manuscript publicized October 7, 2016.

[†]The authors are with School of Computer Science and Engineering, Chung-Ang University, 221 Heukseok, Dongjak, Seoul 156–756 South Korea.
a) E-mail: cglee@cau.ac.kr (Corresponding author)

DOI: 10.1587/transinf.2016EDP7186

by Martin [9] is carried out. The results indicate that studied metrics are better design quality indicators in terms of their ability to predict the faults in corresponding packages of software system. The major implications of this research are as follow:

1. Describe characteristics of design quality with changes taking place due to dependencies within and among the packages.
2. Determine impact of re-usable components in package based design scenario.
3. Evaluate the effectiveness of fault-proneness prediction using information obtained from re-usable components.

In the following, Sect. 2 provides related work. Section 3 underlines the package design principles and their pros and cons. We define a set of definitions and theoretical formulations of Package Re-usability index in Sect. 4. Section 5 presents experimental work as a standard case study followed by Threats to Validity in Sect. 6. We conclude our study with future dimensions in Sect. 7.

2. Related Work

There has been few notable efforts to measure quality aspects of package organization in the past [10], [11], but, there are still research opportunities to explore others dimensions of packages for characterizing the object oriented systems on distinct criteria [12]. There exists a lot of work in the literature proposing metrics for OO software. The majority of these works centered on characterizing a single class as the criteria of high cohesion, low coupling and structural organization [13], [14]. According to a comprehensive survey, there exist almost 16 standard forms of cohesion metrics proposed for object oriented system at different level of abstraction [6]. There have been on-going and continuous attempts to redefine the cohesion metrics with broader scope by bringing into consideration collaborating relationships of elements outside a single component. Counsell *et al*. [15] mathematically analyzed traditional definition proposed in [16]. Result of this study was useful enough to understand behavior of class cohesion from the point of view of distance between the elements and depth of inheritance in software system.

In particular, new principles of Information Hiding, assuming role of module as a service provider and implications of communication among different components have been acknowledged as contributing factors for measuring package based modular quality [7]. These efforts are aimed at redefining the modularization of software component with confined internal functioning and well-defined tasks. Sarkar *et al*. [6] have proposed API-aware cohesion metrics with primary focus to shape the internal and external collaborations of software elements with the notion of segregation and encapsulation. Principle of Information Hiding introduced by Parnas [17] deals with specific aspect to hide the design decisions inside the modules. However, Information Hiding later on became one of the main principle of OO paradigm [18]. Recently, there has been considerable advancement in defining explicit package based modularization principles that impact design and its consequences. Martin [9] introduced package design principles: The *Reuse-Release equivalence principle* (REP), *Common-Reuse Principle* (CRP) and the *Common-Closure principle* (CCP). These principles propose that package reflects the granule of reuse, release and change. One of motivating factor in this direction is package cohesion studied by Albatah *et al.* [19]. Albatah *et al.* proposed cohesion metric considering directed and un-directed dependencies coming into and going out of package. A careful analysis suggests that change propagation can be pivotal factor of re-usability and design restructuring. Hence, there is genuine need of incorporating new principles and methodologies towards software modularization.

In summary, various principles exist on how to modularize software systems. They focus on different aspects of software engineering design like, the principle of low coupling and high cohesion looks at the source code, information hiding OR takes the development process into account. Martin's [9] study reflects the domain that software is part of (identification of classes) or considers the ecosystem around the software (common-reuse principle). It is reasonable to assume that these principles are not independent but connected with each other. A consistent theory or framework to integrate these principles for assessing stable package organization, however, is required.

3. Package Organization Framework

As defined above, a package contains a set of classes and interfaces leading to a compact structure of OO module design. At a particular hierarchical level, package organization can be viewed as set of elements (classes and interfaces) and relations between pairs of these elements with other modules in the system. It is important to build precise assessment, for which a package may conform to principles of CRP and CCP [9]. We have formulated a solution by measuring incoming external dependencies and outgoing external dependencies which eventually insure re-usability of module and change propagation influence. Since the package design should maximize cohesion and minimize the coupling, there is need to integrate both dimensions of relationships to compose re-usability based metric. For specific analysis, Two concepts proposed in this work are: *Package-Package Dependency Analysis (PPDA)* to asses common reuse of packages and *Class-Package Dependency Analysis (CPDA)* to assess common closure of package components. *PPDA* metric is developed on the basis of CCP principle and it considers elements within package to be dependency determinant. It is derived from the ideal design guideline of Martin [9], that the classes of package should depend on same package for allowing uniform re-factoring process in

maintenance phase. And more importantly, common re-use can be achieved, if classes have maximum similar out-going package dependencies. For this purpose, Package Reachability ($\mathcal{PR}$) of each element of package is calculated by finding set of packages which are dependent upon the class under the analysis. *CPDA* metric measures the common reuse of package based on internal incoming dependencies. Intra-package dependencies of a package through *CPDA* are essentially judged considering fact, elements within a package are to serve uniform task and share the responsibility of any change. So, we introduce concept of Class Reachability ($\mathcal{CR}$) sets to determine reuse coverage of classes.

3.1 Notations

In our context, modularized OO design can be formally represented as $\mathcal{MD} =< \mathcal{P}, \mathcal{D} >$, which is basically combination of packages and dependencies among them. In terms of programming terminologies, package is container object for elements, i.e., classes and interfaces at highest level abstraction. Our domain of analysis focus is dependencies involved in change propagation for a package. Furthermore, $\mathcal{C}$ is set of classes, $\mathcal{P}$ is set of all packages, $\mathcal{I}$ is set of all interfaces. Package can be described as collection of elements and relationships among them, i.e, in formal way: $\mathcal{P} = \langle E, \mathcal{R} \rangle$. E represents set of package elements (classes and interfaces), i.e., $E(p) = \{C(p) \bigcup I(p)\}$ and $\mathcal{R}$ is relationship set among elements in $\mathcal{MD}$. For a packages $p \in \mathcal{P}$, $C(p)$ denotes the set of classes in package, $\mathcal{I}(p)$ represents set of interfaces in a package p and $E(p)$ represents set of all elements in package, *DependsOn* defines relationships (i.e., inheritance, association, composition, use dependencies, abstract implementation) among the elements in overall $\mathcal{MD}$. For elements $e, e' \in E$, $DependsOn(e, e')$ represents dependency of e' on e among and within the elements of packages in $\mathcal{MD}$. In other words, $DependsOn(e, e')$ determines, if there is dependency from e' pointing to e.

In order to evaluate the *PPDA*, package dependencies of interfaces $\mathcal{I}(p)$ and classes $C(p)$ are measured in the overall $\mathcal{MD}$. Having established a formal background of our contextual $\mathcal{MD}$, then $\mathcal{PR}$ set accumulates their elements, if there are direct dependent packages p_1 to p_2 or there are indirect dependent packages p_1 to p_m in sequence of inter-dependent packages $\{p_1, p_2, \ldots, p_m\}$. Similarly, *CPDA* is computed through internal dependencies of all classes $C(p)$ in the package p. Therefore, $\mathcal{CR}$ set accumulates their elements, if there is direct dependent elements e_1 to e_2 or there are indirect dependent elements from e_1 to e_m in sequence of inter-dependent elements $\{e_1, e_2, \ldots, e_m\}$ within a package p.

$$\mathcal{PR}(e) = \{p' \in \mathcal{P} \mid DependsOn(e,e') \wedge (p \neq p') \quad where\ e \in E(p),\ e' \in E(p')\}$$

$$\mathcal{CR}(e) = \{e' \in E(p) \mid DependsOn(e,e') \quad where\ e \in E(p)\}$$

3.2 Definitions

On the basis of notations defined in previous section, each type of metric is computed. We consider Package re-usability index as composite value that can be obtained from *PPDA* and *CPDA*.

Definition 1 *PPDA can be defined as a "ratio for similarity of purpose among all the Package Reachability sets in package p".*

$$PPDA(p) = \frac{|\bigcup_{e1,e2 \in E(p)} PR(e1) \cap PR(e2)|}{|\bigcup_{e \in E(p)} PR(e)|}$$

Definition 2 *CPDA can be defined as a "ratio for similarity of purpose among all the Class Reachability sets in pacakge p".*

$$CPDA(p) = \frac{|\bigcup_{e1,e2 \in E(p)} CR(e1) \cap CR(e2)|}{|\bigcup_{e \in E(p)} CR(e)|}$$

3.3 Method of Calculating the Package's Normalized Distance

Formation of composite metric follows Martin's concept of graph based approach for computing package metric [20]. A package can be represented as point in coordinate systems having format: (Instability, Abstractness), i.e., Instability along *X-axis* and Abstractness along the *Y-axis*. The core idea of abstractness in object oriented design paradigm is ability of package to be extended by other interacting packages. Therefore, any modification in a package shall eventually cause a cascading effect on its dependent packages along dependency relationship path (a situation known as ripple effect). In order to determine consistency of change during the ripple effect in a package, PPDA metric is proposed. Packages at location (0,1) have maximum abstractness and at location (0,0) have minimum abstractness.

On the contrary, instability relates to a design in which package is more concrete and resistant to any external change. In other words, CPDA measures ripple effect consistency of elements within a package. Packages at location (1,0) are most concrete and internally stables and at location (0,0), packages are maximally instable. In real applications, packages vary in their degrees of abstractness and instability; not all the packages are located at extreme points, i.e., (0,1) and (1,0). The cohesive balance between Instability and Abstractness is measured using distance metric. Therefore, a package at (0,0) indicates bad design (worst case) and at (1,1) depicts the optimal case. The combined dependency values shall constitute the re-usability index, which is defined as follows:
$PkgReuse(p) = \frac{\sqrt{2}-D}{\sqrt{2}}$, where D is the distance between the two metrics of package dependencies, and it is given by: $D = \sqrt{(1 - PPDA)^2 + (1 - CPDA)^2}$. This approach is widely used to form composite metric [9], [19].

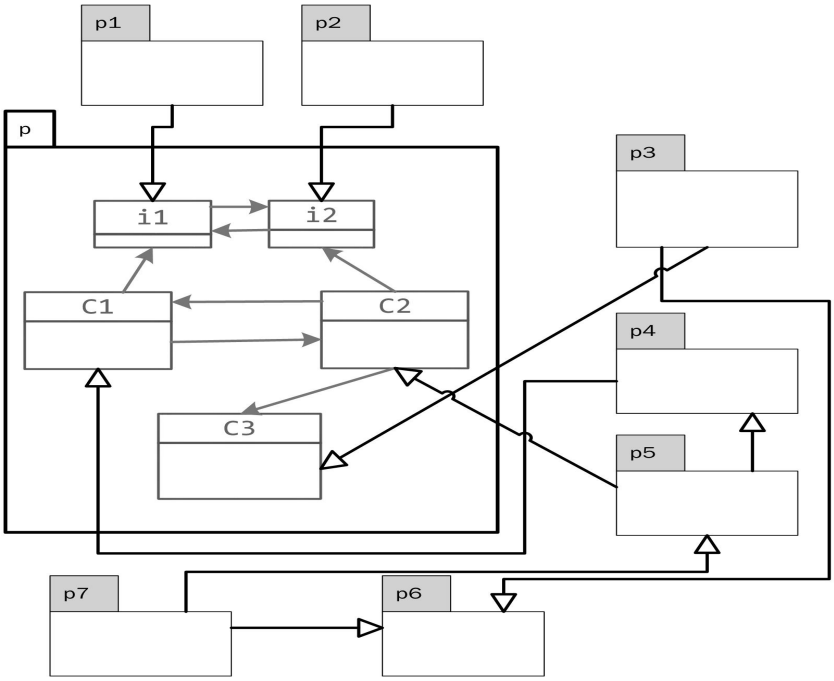


Fig. 1 Package modularization interactive scenario

3.4 Working Example

To elaborate the notion of package re-usability practically, consider a well-engineered code in which a package p has certain service-oriented inter-linkages with the seven other packages, i.e., *p1, p2, . . . ,p7* as shown in Fig. 1. Internal package dependencies are represented through solid arrows inside package p while closed sharp arrows show external dependencies causing an overall change-propagation effect in modularization design for package p. Peeping into design details, we see that dependencies of package p are extended to linked packages through interfaces: $i1, i2$ and classes: $c1, c2$ and $c3$. It is evident that any change in class $c1$ shall trigger change in all direct and indirect dependent packages: *p4, p5, p7* forming *PR(c1)* set. Further design analysis shall also derive what internal dependencies are involved in change effect for classes or elements of a package p, like $c3$ has internal dependence on classes: $c1, c2$ that compose *CR(c3)* set.

Our objective is to capture this aspect of usage in overall modularization measures for a package. To illustrate the idea further from the Fig. 2, which shows depends relationships for a package p in an OO design. In practice, classes can utilize functionality of other packages, while those packages have dependencies in turn on various classes. Table 1 shows Package Reach-ability and Class Reach-ability analysis for package p concretely. From Table 1, we can compute normalized metric values using the definitions as:

$$CPDA(p) = \frac{\left|\bigcup_{e1,e2 \in E(p)} CR(e1) \cap CR(e2)\right|}{\left|\bigcup_{e \in E(p)} CR(e)\right|} = \frac{|\{c1,c2\}|}{|\{c1,c2,c3,i1,i2\}|} = 2/5$$

$$PPDA(p) = \frac{\left|\bigcup_{e1,e2 \in E(p)} PR(e1) \cap PR(e2)\right|}{\left|\bigcup_{e \in E(p)} PR(e)\right|} = \frac{|\{p4,p5,p7\}|}{|\{p1,p2,p3,p4,p5,p6,p7\}|}$$

Table 1 Dependency analysis of a package

$E(p)$	$\mathcal{CR}$	$\mathcal{PR}$
$C1$	$\{c2\}$	$\{p4, p5, p7\}$
$C2$	$\{c1\}$	$\{p4, p5, p7\}$
$C3$	$\{c1, c2\}$	$\{p3, p4, p5, p7\}$
$i1$	$\{c1, c2, i2\}$	$\{p1, p2, p4, p5, p7\}$
$i2$	$\{c1, c2, i1\}$	$\{p1, p2, p4, p5, p7\}$

$= 3/7$ and $PkgReuse(p) = 0.55$.

Theoretically, we follow the theme of definitions as proposed by Albatah *et al.* [19]. However, our approach differ in calculating the scope of dependencies which is to determine effect of change propagation in package organization. Two dimensional reach-ability analysis has its main focus to retrieve the dependency information which impact the package design in terms of re-useability. Similarly, Our proposed index bear an implicit validation as it's computation and concept correspond to cohesion metrics discussed in the literature [7], [13], [19].

4. Empirical Investigation

We present an empirical evaluation on different subsequent releases of open source software systems. An empirical methodology was adopted to evaluate our approach using state-of-the-art mechanism of quality assurance. In the subsequent sections, quantitative results are presented and explained through comprehensive graphical analysis as well.

4.1 Research Objectives

As explained in earlier section, our goal is to determine

the impact of proposed software modularization index over fault-proneness prediction of software systems. Our hypothesis is that, given the nature of the package modularization, it should capture different aspects of software quality. More specifically, we build following different prediction models and statistical analysis.

(a) Magnitude of association between metrics and post-release faults.

(b) Intra-package prediction models with three different classification techniques.

(c) Intra-package prediction models with Robert Martin metrics *RM* and *PkgReuse*.

(d) Inter-package prediction models with Robert Martin metrics *RM* and *PkgReuse*.

(e) Intra-package effort-aware prediction analysis with Zhao *et al.* studied model and *PkgReuse*.

4.2 Experimental Methodology

In this section, we provide a brief overview of statistical techniques and mechanism of their application in our study.

4.2.1 Multivariate Logistic Regression

Logistic regression is standard statistical modeling technique in which the dependent variable can take on one of two different values: 0 and 1. A multivariate logistic regression model is based on the following relationship equation:

$$Pr(Y = 1|X_1, X_2, \ldots X_n) = \frac{e^{\alpha+\beta_1 X_1+\beta_2 X_2+\ldots\beta_n X_n}}{1+e^{\alpha+\beta_1 X_1+\beta_2 X_2+\ldots\beta_n X_n}}$$

Where $X_1, X2 \ldots, X_n$ are independent variables, i.e., characteristics describing the source code (package level metrics), $Pr(Y = 1|X_1, X_2, \ldots X_n)$ represents the probability that the dependent variable $Y = 1$, i.e., the extent of package predicted as faulty.

4.2.2 Random Forest

RF averages the predictions obtained from several decision trees where each tree is fully grown and is based on sampled values. Random forest is known to be robust techniques for noise reduction in prediction models.

4.2.3 Support Vector Machine

SVM is a machine learning technique that tries to maximize margin of hyperplane separating different classifications.

4.2.4 Correlation Analysis

The correlation analysis aims to determine relationship among variables. The correlation coefficient is measure of association strength between two variables. For this purpose, Spearman's rank correlation is widely performed over software metrics having nonparametric nature. The significance of correlation is tested at different levels of confidence interval, i.e., 95%, 90%.

4.2.5 Evaluation

All our prediction models output probabilities of fault-proneness of package entities. To classify a package as faulty, varying thresholds on probability are utilized. Thus, different choices of threshold will produce varying rates of *false* positives/negatives (FP/FN) and *true* positives/negatives (TP/TN).

- *Accuracy (Acc.)*: measures the proportion of correct predictions. Accuracy is defined as: $Acc = \frac{TP+TN}{TP+TN+FP+FN}$
- *Precision (Pr.)*: measure of exactness, defines probabilities of true faulty packages to the number of package predicted as faulty. Precision is defined as $Pr = \frac{TP}{TP+FP}$
- *Recall (Rec.)*: measure of completeness, defines the probabilities of true faulty packages in comparison to total number of faulty packages. Recall is defined as $Rec = \frac{TP}{TP+FN}$
- *F-measure (F1.)*: measures harmonic mean of precision and recall of predicted model. $F1 = \frac{2*Pr.*Rec.}{Pr.+Rec.}$

4.3 Data Collection

For constructing different prediction models, 4 different open source software were used. Eclipse[†]: An Integrated Development Environment (IDE) for software development in collaborative working groups. jEdit[††]: A mature programmer's text editor, written in java and an extensible plug-in architecture. JDT Core[†††]: An incremental java compiler and API for navigating the java element tree. Ant[††††]: It is a library and command-line tool to drive processes described in build files. These systems encompass different application domains and are frequently used in software quality research [21]. Table 2 provides descriptive structural information regarding all these systems. Each data-set comprises of six traditional package-level metric described in Table 3 by Martin [1] and one *PkgReuse(p)* metric. Post-release fault data of subject systems was obtained from public repositories, i.e., Eclipse Bug Data[†††††] and PROMISE[††††††]. All the required metrics were computed by our own scripts developed through Understand-Perl API[*].

[†]https://eclipse.org/
[††]http://www.jedit.org/
[†††]http://www.eclipse.org/jdt/core/index.php
[††††]http://ant.apache.org/
[†††††]https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/
[††††††]http://openscience.us/repo/issues/bugfiles.html
[*]https://scitools.com/feature/api/

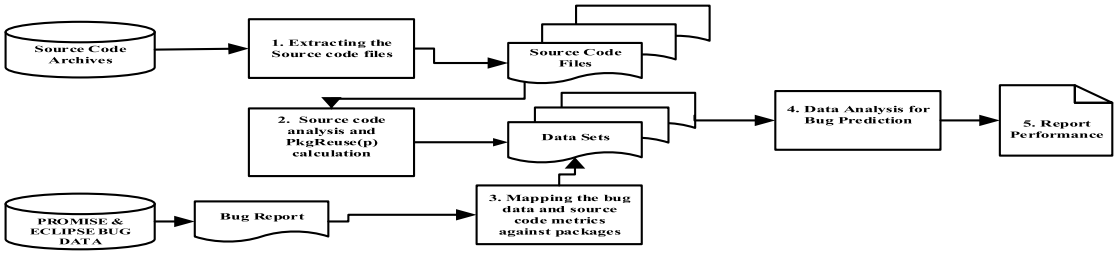


Fig. 2 Data processing steps involved in empricial study

Table 2 Structural information of subject systems

System	Versions	Number of Packages	Total Number of defects	Defective Packages	Percentage of Defective packages
Eclipse	2.0	377	917	190	50%
Eclipse	2.1	428	657	193	45%
Eclipse	3.0	642	1523	307	47%
Ant	1.6	74	107	15	20%
Ant	1.7	80	316	30	38%
JDTCore	2.0	37	144	23	62%
JDTCore	2.1	38	63	18	47%
JDTCore	3.0	42	318	34	80%
jEdit	4.2	28	14	12	42%
jEdit	4.3	37	19	6	16%

Table 3 Description of Martin metric suite

Metric	Definition
N	Class entities: The number of concrete, abstract classes and interfaces in the package.
Ca	Afferent Coupling: The number of other packages that depend upon classes within a package.
Ce	Efferent Coupling: The number of other packages that other class in a package depend upon.
A	Abstractness: The ratio of abstract classes in package to total number classes in a package.
I	Instability: The ratio of Efferent Coupling to total Coupling, $I = \frac{Ce}{(Ce+Ca)}$.
D	Distance: The distance from the main sequence: $D = \|A + I - 1\|$.

Table 4 Magnitude of association with post-release faults in larger datasets

Metric	Eclipse-2.0	Eclipse-2.1	Eclipse-3.0	Ant-1.6	Ant-1.7
PkgReuse(p)	0.17^{***}	0.15^{**}	0.12^{**}	0.049^{*}	0.054^{*}
N	0.62^{***}	0.66^{***}	0.54^{***}	0.76^{***}	0.12
A	−0.0025	−0.034	−0.012	0.052	0.019
D	−0.02	−0.015	−0.02	−0.15	0.034
Ce	0.14^{**}	0.20^{***}	0.24^{***}	0.76^{***}	0.16
Ca	0.26^{***}	0.31^{***}	0.27^{***}	0.68^{***}	0.23^{*}
I	−0.004	−0.015	0.03	0.21	0.18

Table 5 Magnitude of association with post-release faults in smaller datasets

Metric	JEdit-4.2	jEdit-4.3	JDTCore-2.0	JDTCore-2.1	JDTCore-3.0
PkgReuse(p)	0.15^{*}	0.12^{*}	0.47^{**}	0.71^{**}	0.3
N	0.12	0.54^{***}	0.33^{**}	0.37^{***}	0.38^{***}
A	0.021	−0.012	0.018	0.019	0.03
D	−0.26	−0.02	−0.022	−0.004	0.0039
Ce	0.24	0.24^{***}	0.34^{***}	0.37^{***}	0.41^{***}
Ca	0.18	0.27^{***}	0.41^{**}	0.39^{**}	0.42^{**}
I	−0.015	0.30	0.21	0.04	0.03

4.4 Data Processing

In this section, an overview of data processing steps are introduced in Fig. 2. The first step is to mine the source code from repositories and parsing source code files through Understand[†]: A commercial static analysis tool for re-engineering and maintaining the software. Source code analysis is carried out by querying the Understand database to extract the required package level metrics as second step. Step three is mapping of post-release faults collected from open-source repositories and metrics against package entities of corresponding source code file to compose the data-sets. In fourth step, data analysis techniques are used to build prediction models. Finally, fifth step reports performance of models.

4.5 Result Analysis

In this section, we present experimental analysis using subject systems. We have essentially highlighted potential of data to predict the fault-prone packages through different statistical models. All the computations are performed using R[††]. Additional information related to experiments has been made public on project site[†††].

[†]https://scitools.com/

[††]http://www.r-project.org/

[†††]https://github.com/Analyzer2210cau/IEICE-Data

4.5.1 Association with Faults

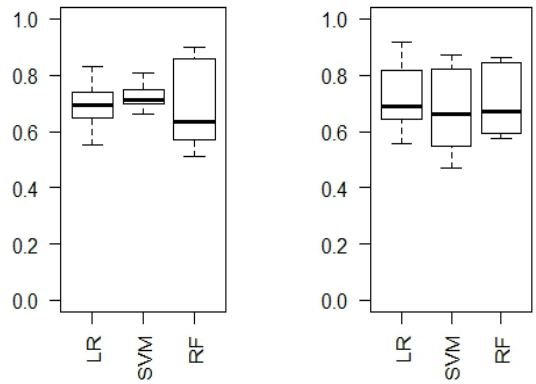
To analyze the association between metrics under investigation and post-release faults, we used Spearman's correlation test at different levels. The results obtained by this test are shown for larger and smaller data-sets in Tables 4 and 5 respectively. In both tables, each cell with correlation coefficient value means that metric in corresponding data set is associated with number of faults at significant levels of 0.001 (denoted by ***), 0.05 (denoted by **) and 0.01 (denoted by *). In case of larger data-sets, most of metrics have shown statistical significance except *D, A* and *I*. However, they still have developed weak negative and positive correlation with number of post-release faults as shown in Table 4. While in case of smaller data-sets, statistical significance for some of Martin metrics, like, *N, Ca, Ce* and *PkgReuse* is observed frequently as shown in Table 5. Interestingly, *Ca* and *Ce* are seen to posses strong positive statistically significant association with number of post-release faults. Such findings lead to following salient indications for ranking the metrics against post-release faults. First, presence of post-release faults can be extensively found in software systems with large number of packages. Second, Afferent and Efferent package coupling (*Ca* and *Ce* metrics) can influence post-release faults to considerable extent.

Table 6 10-Fold Cross-validation analysis using different modeling techniques

System	LR			SVM			RF		
	Acc. (mean)	**Pr.** (mean)	**Rec.** (mean)	**Acc.** (mean)	**Pr.** (mean)	**Rec.** (mean)	**Acc.** (mean)	**Pr.** (mean)	**Rec.** (mean)
Eclipse-2.0	0.65	0.67	0.64	0.70	0.43	0.52	0.62	0.63	0.66
Ecipse-2.1	0.69	0.72	0.68	0.75	0.70	0.75	0.57	0.69	0.63
Eclipse-3.0	0.71✓	0.75	0.69	0.70	0.72	0.52	0.56	0.71	0.66
Ant-1.7	0.70	0.86	0.78	0.81✓	0.84	0.89	0.9 ✓	0.81	0.92
Ant-1.6	0.69	0.72	0.65	0.71	0.83	0.81	0.74✓	0.80	0.74
JDTCore-3.0	0.70	0.85	0.79	0.77✓	0.84	0.90	0.86	0.78	0.93
JDTCore-2.1	0.55	0.61	0.51	0.70	0.57	0.52	0.51	0.54	0.66
JDTCore-2.0	0.83✓	0.93	0.90	0.75	0.72	0.75	0.89✓	0.88	0.81
jEdit-4.2	0.74	0.67	0.62	0.71	0.59	0.57	0.62	0.56	0.60
jEdit-4.3	0.79✓	0.65	0.59	0.66	0.58	0.52	0.65	0.53	0.63

4.5.2 Prediction Performance

We employed three different types of prediction techniques, i.e., Logistic Regression (LR), Random Forest (RF) and Support Vector Machine (SVM) to discover possible relationships between values of collected metrics and fault-proneness of packages. These methods have been proven effective in fault-proneness prediction research literature [22]–[24]. In order to analyze our data with reliability and confidence, 10 times 10-fold cross validation analysis was applied in all of three techniques. This approach essentially separates the data-set in training and testing segments, then adopts classification approach to predict the package to be faulty or not. Cross-validation mechanism of prediction has sound computational importance and is widely used to evaluate the model from different dimensions [25]. Table 6 shows performance evaluation of all described techniques using confusion matrix paradigm. Certain interesting findings are to be mentioned; First, LR and RF have relatively produced satisfactory results than SVM which demonstrate that prediction result is dependent on efficacy of technique used. Second, fault-prediction accuracy is seen to be improving with increase in number of packages in corresponding releases of data-sets, e.g., Ant-1.7, JDTCore-3.0, Eclipse-3.0. It can be well inferred that fault-prediction capability with *pkgReuse* becomes more effective as software systems evolve. Third, despite low percentage of defective packages in smaller data-sets like, jEdit-4.3, JDTCore-3.0, significant accuracy of fault-prediction is observed. Fourth, RF has generated effective prediction models in terms of accuracy. For detailed assessment of performances of different techniques, the accuracy and F-measure of different techniques are compared graphically in Fig. 3. From the Table 6 and Fig. 3 (a), we can observe that LR's overall prediction accuracy ranges between 0.83 and 0.55 where as, that of SVM falls in range of 0.81 to 0.66 and RF is found between 0.56 and 0.9. However, mean value of accuracy with LR and SVM is recorded almost 0.65 which is higher than RF. It asserts the fact that LR is a reliable techniques for predicting the faults accurately. As of generalized finding indications, we are able to correctly predict



(a) Accuracy analysis (b) F-Measure analysis

Fig. 3 Graphical representation of performance measure with different modeling techniques

almost 70% of fault-prone packages, with varying false positive rate and prediction accuracy of 55% as minimum with and 90% as maximum. Notably, these reasonable prediction values are obtained with rigorous experimental setup of 10-times 10-fold cross validation and default parameter settings of the learning methods (indicated with ✓). F-measure evaluates prediction performance considering harmonic mean of recall and precession which is recognized in many machine learning studies [26], [27]. Figure 3 (b) compares F-measure of LR, RF, SVM, with LR and RF having almost same level of measure. LR has delivered significantly sound prediction by acquiring 0.69 median value. On the other hand, lowest F-measure score of prediction, i.e., 0.47 is observed using SVM. It implies that correctness and soundness of prediction result using LR and RF technique is better than SVM. Apparently, SVM has not produced the confidence in term of F-measure performance, but, prediction accuracy value is still significant in general perspective. Therefore, if objective is to correctly predict a higher percentage of defective packages, then LR remains preferred technique. Table 7 summarizes detailed information of fault-prediction model built on the basis of intra-release test and train data-sets. Results obtained from 10 times 10-fold cross-validation logistic regression (LR) for two models using Martin metrics

Table 7 Intra-release prediction models: comparison

System	*RM*		*PkgReuse(p)+RM*		Improved
	Acc. (mean)	**F1.** (mean)	**Acc.** (mean)	**F1.** (mean)	**%age**
Eclipse-2.0	0.60	0.59	0.65✓	0.65	10%
Eclipse-2.1	0.59	0.49	0.61✓	0.69	3%
Eclipse-3.0	0.64	0.61	0.66✓	0.71	3%
Ant-1.7	0.72	0.75	0.81✓	0.78	12 %
Ant-1.6	0.64	0.65	0.69✓	0.68	8 %
JDTCore-3.0	0.70	0.78	0.72 ✓	0.81	2%
JDTCore-2.1	0.55	0.52	0.55	0.55	–
JDTCore-2.0	0.82	0.86	0.82	0.91	–
jEdit-4.2	0.71	0.74	0.74✓	0.64	5.7%
jEdit-4.3	0.77	0.35	0.79✓	0.61	2.5%

Table 8 Inter-release prediction models: comparison

System	*RM*		*RM+PkgReuse(p)*		Improved
	Acc. (mean)	**F1.** (mean)	**Acc.** (mean)	**F1.** (mean)	**%age**
Eclipse-2.0(Train), Eclipse-2.1(Test)	0.65	0.58	0.71✓	0.61	3%
jEdit-4.2(Train), jEdit-4.3(Test)	0.57	0.62	0. 63✓	0.64	9%
Ant-1.6(Train), Ant-1.7(Test)	0.60	0.64	0.59	0.6	–
JDTCore-2.0(Train), JDTCore-2.1(Test)	0.58	0.57	0.582	0.59	–

Table 9 Intra-release ER models: comparison

System	*RM*+M	*RM +PkgReuse(p)*+M
(a) ER-MFM		
Eclipse-2.0	0.51 ± 0.13	0.53 ± 0.1 ✓
Eclipse-2.1	0.4 ± 0.1	0.43 ± 0.13 ✓
Eclipse-3.0	0.53 ± 0.11	0.55 ± 0.14 ✓
Ant-1.6	0.79 ± 0.12	0.80 ± 0.13✓
Ant-1.7	0.77± 0.1	0.81± 0.09 ✓
JDTCore-2.0	0.63 ± 0.13	0.61± 0.3 ×
JDTCore-2.1	0.58 ± 0.2	0.57± 0.23 ×
JDTCore-3.0	0.51 ± 0.11	0.51± 0.12–
jEdit-4.2	0.61± 0.21	0.51± 0.22 ×
jEdit-4.3	0.72 ± 0.27	0.74 ± 0.21✓
Win\Tie \Loss	—	6 \1 \3
(b)ER-BPP		
Eclipse-2.0	0.31 ± 0.08	0.32 ± 0.07 ✓
Eclipse-2.1	0.47 ± 0.06	0.48 ± 0.07 ✓
Eclipse-3.0	0.34 ± 0.07	0.34 ± 0.07 –
Ant-1.6	0.64± 0.01	0.62 ± 0.03 ×
Ant-1.7	0.77± 0.1	0.81± 0.09 ✓
JDTCore-2.0	0.61 ± 0.1	0.63 ± 0.09 ✓
JDTCore-2.1	0.60 ± 0.09	0.57 ± 0.07 ×
JDTCore-3.0	0.47 ± 0.12	0.43 ± 0.13 ×
jEdit-4.2	0.59 ± 0.07	0.56 ± 0.1 ×
jEdit-4.3	0.67 ± 0.09	0.64± 0.08 ×
Win\Tie \Loss	—	4 \1 \5

RM and *PkgReuse+RM* are presented in Table 7. It can be clearly observed that there is substantial accuracy improvement while classifying the *PkgReuse+RM* model against *RM*(Baseline) model. From Table 7, we make following two observations. First, intra-release prediction models for data-sets with large number of packages have easily outperformed baseline approach of fault-prediction which is the case with 3 versions of Eclipse (2.0, 2.1, 3.0) and 2 versions of Ant (1.6, 1.7). Second, Although accuracy in data-sets with smaller number of packages has not won over *RM* model, however, their F-measure score is still satisfactory. Mean improved accuracy of successful models in Table 7 has been seen in range of 0.61 − 0.81 (indicated with ✓). Overall *PkgReuse+RM* obtained 12% of maximum accuracy improvement in larger data-sets and 5.7% of maximum accuracy improvement in smaller data-sets. Table 8 presents summary of fault-prediction model using inter-release frame which is more rigorous approach of developing train and test data-sets. Table 8 mainly shows values of accuracy and F1 measure for fault-prediction model across releases of Eclipse, Ant, JDTCore and jEdit using Logistic Regression. Clearly, prediction results with *PkgReuse+RM* achieved competitive accuracy in 2 cases (indicated with ✓) as shown in Table 8 and is recorded with 9% of maximum improvement. Again, data-sets with less improved accuracy have managed to gain improved F-measure score in inter-release prediction as shown for JDTCore(2.0-2.1) and Ant(1.6-1.7) in Table 8. It justifies that application of *PkgReuse+RM* is not redundant with baseline approach.

4.5.3 Effort Aware Performance

To evaluate the performance of models, there has been emphasis on effort-reduction (ER) while predicting fault-prone entities using classification. Zhao *et al.* mainly presented comparative analysis of effort-aware fault-prediction models between Martin metrics and modularization metrics by Sarkar *et al.* [6], [8]. Zhao *et al.* termed these models as "T" (*RM* model in our context of study) and "M+T" respectively and concluded that "M+T" model is more effective in fault-proneness prediction. Zhao *et al.* utilized certain benchmarks for computing ER models, like, Balanced-Metric (BPP), Maximum-F-measure (MFM). In an experimental scenario, all described benchmarks define threshold for model prediction as prerequisite. BPP method leverages Receiver Operating Curve (ROC) for prediction model and sets maximum "balance" as classification threshold. MFM method chooses the threshold that has maximum F-measure score for prediction model. ER based models are generally perceived to bear cost-effectiveness against traditional models. To further illustrate utility of *pkgReuse*, comparative analysis between models: "M+*RM*" and "M+*RM*+*pkgReuse*" is explained in Table 9.

Table 9 summarizes mean and standard deviation values of different ER metrics acquired from 10-times 10-fold cross validation mechanism. It is evident that *PkgReuse+RM*+M either out performs *RM*+M model or exhibits substantial improvement in most of cases (indicated with ✓) using ER-MFM criterion. However, using ER-BPP, *PkgReuse+RM*+M has delivered competitive performance against *RM*+M. Combining the results from Sects. 4.5.2 and 4.5.3, we conjecture that proposed *PkgReuse* index has a better ability to predict fault-proneness of packages when used in combination with traditional metrics. Additionally, *PkgReuse* index is not redundant with *RM*+M model evalu-

ated by Zhao et al. This summary of results is consistent with intuition that impact of managing re-usable components extensively improves software design.

5. Threats to Validity

In this section, most important threat to construct, internal and external validity of our study are discussed. Construct validity generally refers to measurement accuracy of variables. Internal validity of study is insured on the basis of conclusion drawn from concept to experimental work. External validity is the extent to which generalization can be obtained from the scope of study and other research settings.

The most important threat to construct validity is measurement accuracy of dependent and independent variables. We employed our source code analysis procedure with incremental testing to collect the metrics (independent variables) reliably using *Understand* tool which is already utilized in recent empirical studies [8]. Additionally, we calculated metrics manually for test project to insure their validity as satisfactory as possible. The dependent variable in our study is binary variable indicating package being faulty or non-faulty. In our study, fault data for open source software system was obtained from public ally available resources, like *Promise* repository. Also, the data for dependent variables has been already utilized for many fault-prediction studies [28], [29] which can be considered reliable for research specific study.

A potential threat to internal validity comes from testing effort employed. Since, prediction methodology employs that each package that is predicted as faulty, incurs effort roughly proportional to its size (SLOC). Therefore, we also developed effort-aware prediction models to enhance its cost effectiveness. Beside, we also believe that SLOC may not have large influence on our results as SLOC has been used as effort proxy in previous research literature [30], [31]. Although, internal validity could have been questioned in case of worsen performance by our proposed *pkgReuse* against traditional model. However, results lead to draw meaningful conclusion. We applied cross-validation evaluation methodology over open-source java systems. Therefore, generalization of conclusion to software systems developed in other particular language may not be possible. This is common problem in most of empirical studies where researchers do not have free access to software systems developed or operational in industrial environment. Software systems used in our study have reasonable size and specific domain which also account for threats to external validity. To mitigate this threat, we aim to further refine our approach on larger data sets with effective computational model. Apart from the complications discussed above, we believe that study still holds theoretical and experimental importance.

6. Conclusions

Our core emphasis was focused on determining quality impact of package organization and metrics related to it, which are assumed to have good explanation and predictive strength. We have presented an assessment with empirical evidence that describes effectiveness of re-usability based package organization towards fault-proneness prediction. Also, in this study, we investigated the efficacy of applying package level metrics to automatically predict the fault-prone entities in software systems. We proposed a *PkgReuse* index which is derived from the context of re-usability and changeability of package organization. We presented comparative analysis using three statistical learning techniques to build bug predictors in an empirical study of open-source software systems. Fault-prediction models constructed through our *PkgReuse* metric and traditional metrics in intra-release and inter-release frame of experiment were able to obtain improved accuracy against traditional metrics. Following conclusion can be conjectured based on our study: First, re-usability is one of of key determinants for insuring bug-free maintenance in object oriented software. However, re-factoring process adversely affects structural organization of source code. Consequently, faults can not be underestimated due to these changes considering operational importance of software. Predicting fault-prone entities in the context of package re-use and change captures new and complementary dimensions of its structure; allowing developers to prioritize their restructuring objectives.

Second, our results show that statistical models deployed for package level fault-prediction have diverse computational techniques and fluctuate in their performance. It can be deduced that package-level predictors of faults are context-sensitive to particular type of models and this research dimension can be explored further. Our results also indicate that base-line approach despite its significant performance in fault-prediction requires additional information at package-level to acquire better performance in terms of avoiding false positives.

Third, consolidated relationship between theory and implementation is essential. Complication of measuring variables and analysis framework proposed in our approach require broader verifications and recognitions. We also observed that validations performed over open source software systems vary due to certain restrictive requirements in source code and applications of fault-prediction approach.

In today's software industry, application development takes place under competitive and rapid approaches that may raise threats of faults in source code packages having complex dependencies. Thus, maximum possible estimation of faults is desirable to insure error free code and avoid functional failures. Such approaches help in optimizing and restructuring development models to fix the faults as early as possible, eventually increasing productivity and economic value of software systems.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant (NRF-2014R1A2A2A01005519)

and the ITRC (Information Technology Research Center) support program (IITP-2015-H8501-15-1012) funded by the Korea government.

References

[1] R.C. Martin, "Design principles and design patterns," Object Mentor, vol.1, pp.1–34, 2000.
[2] T. Zhou, B. Xu, L. Shi, Y. Zhou, and L. Chen, "Measuring package cohesion based on context," 2008 IEEE International Workshop on Semantic Computing and Systems, WSCS'08, pp.127–132, IEEE, 2008.
[3] Y. Lee, B. Liang, S. Wu, and F. Wang, "Measuring the coupling and cohesion of an object-oriented program based on information flow," Proc. International Conference on Software Quality, Maribor, Slovenia, pp.81–90, 1995.
[4] G. Gui and P.D. Scott, "Coupling and cohesion measures for evaluation of component reusability," Proc. 2006 International Workshop on Mining Software Repositories, pp.18–21, ACM, 2006.
[5] D. Rodríguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz, "Detecting fault modules applying feature selection to classifiers," 2007 IEEE International Conference on Information Reuse and Integration, IRI 2007, pp.667–672, IEEE, 2007.
[6] S. Sarkar, A.C. Kak, and G.M. Rama, "Metrics for measuring the quality of modularization of large-scale object-oriented software," IEEE Trans. Softw. Eng., vol.34, no.5, pp.700–720, 2008.
[7] H. Abdeen, S. Ducasse, and H. Sahraoui, "Modularization metrics: Assessing package organization in legacy large object-oriented software," 2011 18th Working Conference on Reverse Engineering (WCRE), pp.394–398, IEEE, 2011.
[8] Y. Zhao, Y. Yang, H. Lu, Y. Zhou, Q. Song, and B. Xu, "An empirical analysis of package-modularization metrics: Implications for software fault-proneness," Information and Software Technology, vol.57, pp.186–203, 2015.
[9] R.C. Martin, Agile software development: principles, patterns, and practices, Prentice Hall PTR, 2003.
[10] E.B. Allen, T.M. Khoshgoftaar, and Y. Chen, "Measuring coupling and cohesion of software modules: an information-theory approach," Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International, pp.124–134, IEEE, 2001.
[11] T.J. Emerson, "A discriminant metric for module cohesion," Proc. 7th International Conference on Software Engineering, pp.294–303, 1984.
[12] R. Martin, "Oo design quality metrics-an analysis of dependencies," Proc. Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, 1994.
[13] L.C. Briand, J.W. Daly, and J. Wüst, "A unified framework for cohesion measurement in object-oriented systems," Empirical Software Engineering, vol.3, no.1, pp.65–117, 1998.
[14] J.M. Bieman and B.K. Kang, "Cohesion and reuse in an object-oriented system," ACM SIGSOFT Software Engineering Notes, pp.259–262, ACM, 1995.
[15] S. Counsell, S. Swift, and J. Crampton, "The interpretation and utility of three cohesion metrics for object-oriented design," ACM Transactions on Software Engineering and Methodology (TOSEM), vol.15, no.2, pp.123–149, 2006.
[16] J. Bansiya, L. Etzkorn, C. Davis, and W. Li, "A class cohesion metric for object-oriented designs," J. Object-Oriented Programming, vol.11, no.8, pp.47–52, 1999.
[17] D.L. Parnas, "On the criteria to be used in decomposing systems into modules," Communications of the ACM, vol.15, no.12, pp.1053–1058, 1972.
[18] B. Meyer, Object-oriented software construction, Prentice Hall New York, 1988.
[19] W. Albattah and A. Melton, "Package cohesion classification," 2014 5th IEEE International Conference on Software Engineering and Service Science (ICSESS), pp.1–8, IEEE, 2014.
[20] S.A. Almugrin, Definitions and Validations of Metrics of Indirect Package Coupling in an Agile, Object-Oriented Environment, Ph.D. Thesis, Kent State University, 2015.
[21] S.L. Abebe, V. Arnaoudova, P. Tonella, G. Antoniol, and Y. Gueheneuc, "Can lexicon bad smells improve fault prediction?," 2012 19th Working Conference on Reverse Engineering (WCRE), pp.235–244, IEEE, 2012.
[22] H.M. Olague, L.H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes," IEEE Trans. Softw. Eng., vol.33, no.6, pp.402–419, 2007.
[23] L.C. Briand, W.L. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," IEEE Trans. Softw. Eng., vol.28, no.7, pp.706–720, 2002.
[24] E.J. Weyuker, T.J. Ostrand, and R.M. Bell, "Comparing the effectiveness of several modeling methods for fault prediction," Empirical Software Engineering, vol.15, no.3, pp.277–295, 2010.
[25] J.L. Schafer, Analysis of incomplete multivariate data, CRC press, 1997.
[26] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," Information and Software Technology, vol.54, no.3, pp.248–256, 2012.
[27] J. Ren, K. Qin, Y. Ma, and G. Luo, "On software defect prediction using machine learning," Journal of Applied Mathematics, vol.2014, 2014.
[28] Y. Zhao, Y. Yang, H. Lu, J. Liu, H. Leung, Y. Wu, Y. Zhou, and B. Xu, "Understanding the value of considering client usage context in package cohesion for fault-proneness prediction," Automated Software Engineering, pp.1–61, 2016.
[29] K.O. Elish and M.O. Elish, "Predicting defect-prone software modules using support vector machines," Journal of Systems and Software, vol.81, no.5, pp.649–660, 2008.
[30] S. Kim, T. Zimmermann, E.J. Whitehead Jr, and A. Zeller, "Predicting faults from cached history," Proc. 29th International Conference on Software Engineering, pp.489–498, IEEE Computer Society, 2007.
[31] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," 2007 International Workshop on Predictor Models in Software Engineering, PROMISE'07: ICSE Workshops 2007, pp.9–9, IEEE, 2007.

Mohsin Shaikh received B.E. and M.S. degrees from Mehran University, Pakistan and Hanyang University, Korea in 2006 and 2010, respectively. From 2012 to 2013, he worked as Assistant Professor in Qaid-e-Awam University Pakistan. Currently, he is Ph.D. Candidate in Dept. of Computer Science and Engineering at Chung-Ang University. His research interests include software engineering, software architecture and software quality assurance.

Ki-Seong Lee is a post-doctoral research fellow in Dept. of Computer Science and Engineering at Chung-Ang University, Seoul, Korea. He earned the Bachelor's degree in 2005 from the College of Liberal Arts at Sung Kyun Kwan University. He worked as a software engineer for Internet Service Company OnNet from 2006 to 2008. He received the M.S. and Ph.D. degrees in Computer Science and Engineering from Chung-Ang University. His research interests include software architecture recovery and software quality.

Chan-Gun Lee received the B.S., M.S., and Ph.D. degrees in Computer Science from Chung-Ang University, KAIST, and University of Texas at Austin, in 1996, 1998, and 2005, respectively. From 2005 to 2007, he was a senior software engineer at Intel. Currently, he is an Associate Professor of Computer Science and Engineering at Chung-Ang University, Seoul, Korea. His research interests include software engineering and real-time systems.