Information and Software Technology 54 (2012) 1125–1141

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics

Jehad Al Dallal *

Department of Information Science, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait

A R T I C L E I N F O

Article history:
Received 1 January 2012
Received in revised form 24 April 2012
Accepted 29 April 2012
Available online 7 May 2012

Keywords:
Object-oriented design
Extract subclass refactoring
Class quality
Class cohesion
Class coupling
Logistic regression analysis

A B S T R A C T

Context: Refactoring is a maintenance task that refers to the process of restructuring software source code to enhance its quality without affecting its external behavior. Inspecting and analyzing the source code of the system under consideration to identify the classes in need of extract subclass refactoring (ESR) is a time consuming and costly process.
Objective: This paper explores the abilities of several quality metrics considered individually and in combination to predict the classes in need of ESR.
Method: For a given a class, this paper empirically investigates, using univariate logistic regression analysis, the abilities of 25 existing size, cohesion, and coupling metrics to predict whether the class is in need of restructuring by extracting a subclass from it. In addition, models of combined metrics based on multivariate logistic regression analysis were constructed and validated to predict the classes in need of ESR, and the best model is justifiably recommended. We explored the statistical relations between the values of the selected metrics and the decisions of the developers of six open source Java systems with respect to whether the classes require ESR.
Results: The results indicate that there was a strong statistical relation between some of the quality metrics and the decision of whether ESR activity was required. From a statistical point of view, the recommended model of metrics has practical thresholds that lead to an outstanding classification of the classes into those that require ESR and those that do not.
Conclusion: The proposed model can be applied to automatically predict the classes in need of ESR and present them as suggestions to developers working to enhance the system during the maintenance phase. In addition, the model is capable of ranking the classes of the system under consideration according to their degree of need of ESR.

1. Introduction

Refactoring refers to the process of changing an existing object-oriented software code to improve its internal structure while preserving its external behavior [42]. If applied well, refactoring has several benefits such as enhancing the code's understandability, maintainability, and testability [42,62]. Fowler [42] identified when and described how to perform several refactoring scenarios. In one of these scenarios, a class of interest is split into a subclass and a superclass. The motivation behind applying this refactoring is that some of the features of the class are used in some instances and not in others. In this case, an extract subclass refactoring (ESR) activity is applied to split the features of the class.

Identifying the classes in need of ESR requires careful code understanding, review, inspection, and analysis, not only of the classes of interest, but also of the classes that use the classes of interest directly or indirectly by inheritance or by instantiation. Therefore, this identification of classes is a time consuming and labor intensive task. Once the classes in need of ESR are identified, automated refactoring tools (e.g., [53,38] can be applied to perform the required refactoring.

Several techniques are proposed in the literature to predict or identify opportunities to performing different refactoring activities such as moving a method from one class to another (e.g., [13,75]), extracting a method from an existing method (e.g., [76]), introducing polymorphism [77], and extracting a class with an association relation to an existing class (e.g., [82,54,10]. Some of these techniques are based on new metrics that have to be applied to identify the refactoring opportunities (e.g., [13,75–77,10]. Some others are based on weighted formulas that combine sets of existing metrics (e.g., [82]. Except for the work proposed by Kosker et al. [54], as far as we know, none of the existing approaches is based on empirical analysis of the correlation between the need for refactoring and the values obtained from existing quality metrics to predict the refactoring opportunities. In addition, none of the existing

* Tel.: +965 24983309.
E-mail address: j.aldallal@ku.edu.kw

0950-5849/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2012.04.004

techniques aims to predict the classes that require ESR. Kosker et al. [54] used a model of complexity metrics constructed using machine learning statistical techniques to predict the classes in need of refactoring. Their prediction model is limited to the use of complexity metrics as predictors, and it does not predict classes in need of specific types of refactoring. That is, when applying the model, the software engineer will not know which type of refactoring is to be applied to the predicted classes.

The relation between refactoring and quality metrics has been addressed in two other research areas. The first is applying metrics to perform refactoring activities (e.g., [57,37,41,16,17]. The other area is investigating the impact of particular refactoring activities on the quality values obtained using quality metrics (e.g., [20,23,21,11]. In both areas, the pieces of code in need of refactoring are manually identified.

In this paper, we report and discuss the results of several empirical studies that investigated the abilities of 25 size, coupling, and cohesion metrics, considered individually and in combination, to predict the classes in need of ESR. We selected six Java open source systems and mutated the code of selected classes with inheritance relations. These classes were the ones considered to be in need of ESR, as implicitly suggested by the developers of the systems. We discuss how to apply logistic regression analysis [49], a widely applied statistical technique in experiment-based research, in the context of predicting refactoring opportunities. The analysis is applied to investigate the ability of each quality metric to predict the classes in need of ESR (i.e., the mutated classes). In addition, the analysis investigated the general relation between ESR and the size, cohesion, and coupling quality factors. Finally, we constructed, compared, and validated several prediction models based on combinations of metrics.

The results indicate that ESR improves the general quality of classes in terms of size, cohesion, and coupling. In addition, the results demonstrated that some of the considered metrics have outstanding abilities to predict the presence of classes in need of ESR. Considering the metrics in combination showed that some of the metrics were complementary to predicting classes that require ESR. Thus, considering certain combinations of metrics resulted in enhancing the abilities of the metrics to predict the classes in need of ESR. Some of the constructed models showed outstanding abilities to classify the classes into those that require ESR and those that do not.

The major contributions of this paper are as follows:

1. Explanation of how to apply logistic regression analysis in the context of predicting the classes in need of ESR.
2. Investigation of the relationship between size, cohesion, and coupling quality attributes and ESR.
3. Exploration of the abilities of several size, cohesion, and coupling metrics, considered individually, to predict classes in need of ESR.
4. Construction of models, based on combinations of metrics, that have outstanding abilities to predict classes in need of ESR.

The paper is organized as follows: Section 2 reviews related work. Section 3 explains the design of the empirical study. Sections 4 and 5 report and discuss the univariate and multivariate regression analyses and results. Section 6 lists validity threats to the empirical studies. Finally, Section 7 concludes the paper and discusses future work.

2. Related work

In this section, we review and discuss related work in the refactoring area. In addition, we provide an overview of several existing quality class metrics for object-oriented systems and other related work in the area of measuring software quality.

2.1. Refactoring opportunities

Refactoring the code of a software system is typically performed in four stages: (1) identifying the refactoring opportunities, (2) analyzing the cost and benefit of performing the refactoring, (3) performing the corresponding code modification [82], and (4) ensuring that the code modification preserves behavior [59]. Piveta [66] has provided a detailed process for refactoring. Several reports have appeared in the literature that study and explore each of these stages. In this paper, we are concerned with the first stage, in which the refactoring opportunities are identified or predicted.

Several tools and techniques have been developed and applied to identify or predict refactoring opportunities. Atkinson and King [13] proposed a tool that detects some types of refactoring opportunities in C# programs. They chose the refactoring types that do not require complex code analysis to show that predicting some refactoring types can be easily automated. The tool uses code syntactic data such as simple code metrics.

Zhao and Hayes [82] suggested applying several size and complexity metrics to the classes of the system under consideration, calculating the weighted average of the obtained size and complexity values, and ranking the classes accordingly. Classes of high rank are to be given a high priority for refactoring. One of the key limitations of the technique is that it does not differentiate between different refactoring types. That is, once the class is predicted to be in need of refactoring, the software engineer has to analyze the code to decide what types of refactoring are to be performed. The other limitation is related to the selection of only size and complexity metrics as predictors of the classes in need of refactoring whereas other factors that may affect the refactoring decisions, such as cohesion and coupling, were not considered.

O'Keeffe and O'Cinneide [63] proposed a refactoring tool that uses search-based algorithms to find refactoring candidates. The tool performs certain refactoring types and evaluates the modified code using eleven weighted size, cohesion, coupling, and inheritance metrics. If the quality of the code is found to be enhanced by the performed refactoring, the refactoring will be accepted; otherwise, the refactoring will be rejected.

Kosker et al. [54] proposed a set of class complexity metrics and a machine learning technique called Naïve Bayes to predict refactoring opportunities. They used several versions of a system to build their prediction model. They mutated some classes in the first version using common refactoring types such as Extract Interface, Push Members Down, and Extract Superclass. They compared the code of the classes across different versions to determine the actual refactored classes. In addition, they used the data on refactored classes in the machine learning process, which included the complexity values of the classes, to build a classification model. This proposal shares the same limitations as those discussed above for the system of Zhao and Hayes [82].

Tsantalis and Chatzigeorgiou [75] used the notion of a distance between the members of a class to identify Move Method refactoring opportunities. In addition, they proposed a technique based on code slicing to identify opportunities for Extract Method refactoring and perform the corresponding refactoring activity [76].

Piveta et al. [67] proposed a formal approach for representing refactoring opportunities. The resulting representation can be used to help researchers and developers search for the required refactoring types and to find approaches, tools, and quality metrics to perform these refactoring types.

Moha et al. [60] proposed a method to specify and detect several code and design smells. They automatically generated the corresponding detection algorithms and validated them using 11 open-source systems. Independent engineers were involved to assess whether the detection algorithms correctly detected the code and design smells.

Given a class already identified as in need of Extract Class refactoring, Bavota et al. [17] proposed an approach that parses the class and constructs a weighted graph representing the structural and semantic cohesion relations between methods. This graph is used to automatically perform the Extract Class refactoring. Bavota et al. [16] proposed a similar approach that relies on the structural and semantic cohesion relations between methods to perform the Extract Class refactoring, but using a matrix-based representation.

Al Dallal and Briand [10] proposed a cohesion metric and discussed and analyzed how the metric can guide the detection of

Table 1
Definitions of the considered class coupling metrics.

Class coupling metric	Definition/formula
Response for a class (RFC) [35]	Number of methods directly invoked by the methods of class A
Message Passing Coupling (MPC) [56]	The total number of method invocations in a class
Data Abstraction Coupling (DAC1) [56]	The number of attributes that their types are of other classes
DAC2 [56]	The number of distinct classes used as types of the attributes of the class
OCMEC [26]	Number of distinct classes used as types of the parameters of the methods in the class

Table 2
Definitions of the considered class cohesion metrics (modified form [5]).

Class cohesion metric	Definition/formula
Part 1	
Lack of Cohesion of Methods (LCOM1) [34]	LCOM1 = Number of pairs of methods that do not share attributes
LCOM2 [35]	P = Number of pairs of methods that do not share attributes Q = Number of pairs of methods that share attributes. $\text{LCOM2} = \begin{cases} P-Q & \text{if } P-Q \geqslant 0 \\ 0 & \text{otherwise} \end{cases}$
LCOM3 [56]	LCOM3 = Number of connected components in the graph that represents each method as a node and the sharing of at least one attribute as an edge
LCOM4 [48]	Similar to LCOM3 and additional edges are used to represent method invocations
LCOM5 [46]	LCOM5 = $(a-kl)/(l-kl)$, where l is the number of attributes, k is the number of methods, and a is the summation of the number of distinct attributes accessed by each method in a class
Coh [24]	Coh = a/kl, where a, k, and l have the same definitions above
Tight Class Cohesion (TCC) [19]	TCC = Relative number of directly connected pairs of methods, where two methods are directly connected if they are directly connected to an attribute. A method m is directly connected to an attribute when the attribute appears within the method's body or within the body of a method invoked by method m directly or transitively
Loose Class Cohesion (LCC) [19]	LCC = Relative number of directly or transitively connected pairs of methods, where two methods are transitively connected if they are directly or indirectly connected to an attribute. A method m, directly connected to an attribute j, is indirectly connected to an attribute i when there is a method directly or transitively connected to both attributes i and j
Degree of Cohesion-Direct (DC_D) [14]	DC_D = Relative number of directly connected pairs of methods, where two methods are directly connected if they satisfy the condition mentioned above for TCC or if the two methods directly or transitively invoke the same method
Degree of Cohesion-Indirect (DC_I) [14]	DC_I = Relative number of directly or transitively connected pairs of methods, where two methods are transitively connected if they satisfy the same condition mentioned above for LCC or if the two methods directly or transitively invoke the same method
Part 2	
Class Cohesion (CC) [22]	CC = Ratio of the summation of the similarities between all pairs of methods to the total number of pairs of methods. The similarity between methods i and j is defined as:$\text{Similarity}(i,j) = \frac{\lvert I_i \cap I_j \rvert}{\lvert I_i \cup I_j \rvert}$, where I_i and I_j are the sets of attributes referenced by methods i and j, respectively
Class Cohesion Metric (SCOM) [40]	SCOM = Ratio of the summation of the similarities between all pairs of methods to the total number of pairs of methods. The similarity between methods i and j is defined as:$\text{Similarity}(i,j) = \frac{\lvert I_i \cap I_j \rvert}{\min(\lvert I_i \rvert, \lvert I_j \rvert)} \cdot \frac{\lvert I_i \cup I_j \rvert}{l}$, where l is the number of attributes
Low-level design Similarity-based Class Cohesion (LSCC) [10]	$\text{LSCC}(C) = \begin{cases} 0 & \text{if } k = 0 \text{ or } l = 0, \\ 1 & \text{if } k = 1, \\ \frac{\sum_{i=1}^{l} x_i(x_i-1)}{lk(k-1)} & \text{otherwise.} \end{cases}$, where l is the number of attributes, k is the number of methods, and x_i is the number of methods that reference attribute i
Cohesion Based on Member Connectivity (CBMC) [31]	$\text{CBMC}(G) = F_c(G) \times F_s(G)$, where $F_c(G) = \lvert M(G) \rvert / \lvert N(G) \rvert$, $M(G)$ = the number of glue methods in graph G, N(G) = the number of non-special methods represented in graph G, $F_s(G) = \left[\sum_{i=1}^{n} \text{CBMC}(G^i)\right]/n$, n = the number of child nodes of G, and glue methods is the minimum set of methods for which their removal causes the class-representative graph to become disjointed
Improved Cohesion Based on Member Connectivity (ICBMC) [79,80]	$\text{ICBMC}(G) = F_c(G) \times F_s(G)$, where $F_c(G) = \lvert M(G) \rvert / \lvert N(G) \rvert$, $M(G)$ = the number of edges in the cut set of G, N(G) = the number of non-special methods represented in graph G multiplied by the number of attributes, $F_s(G) = \left[\sum_{i=1}^{2} \text{ICBMC}(G^i)\right]/2$, and cut set is the minimum set of edges such that their removal causes the graph to become disjointed
OL_n [81]	OL_n = The average strength of the attributes, wherein the strength of an attribute is the average strength of the methods that reference that attribute. The strength of a method is initially set to 1 and is computed, in each iteration, as the average strength of the attributes that it references, where n is the number of iterations that are used to compute OL
Path Connectivity Class Cohesion (PCCC) [7]	$\text{PCCC}(C) = \begin{cases} 0 & \text{if } l = 0 \text{ and } k > 1, \\ 1 & \text{if } l > 0 \text{ and } k = 0, \\ \frac{\text{NSP}(G_c)}{\text{NSP}(FG_c)} & \text{otherwise.} \end{cases}$ where NSP is the number of simple paths in graph G_c, FG_c is the corresponding fully connected graph, and a simple path is a path in which each node occurs once at most

candidates for Move Method and Extract Class refactoring. The metric was successfully applied by O'Cinneide et al. [62] to detect refactoring opportunities used in exploring the relation between refactoring and testability.

Several researchers have addressed the relation between the types of refactoring and quality attributes. Bois and Mens [21] and Bois et al. [20] analyzed the impact of performing several types of refactoring, including the Extract Method, Encapsulate Field, the Pull Up Method, Extract Class, the Move Method, the Replace Method with Method Object, and Replace Data Value with Object, on cohesion, coupling, and other quality attributes. This effect was empirically investigated by Bryton and Abreu [30], Alshayeb [11], and Boshnakoska and Mišev [23] with different sets of refactoring types and quality metrics. They all concluded that refactoring does not necessarily improve the quality of the code.

None of the proposed techniques aims at detecting or performing ESR. Typically, this type of refactoring requires a complex analysis of the classes of interest and the classes that directly or indirectly use the classes of interest. In addition, none of the studies reviewed here uses logistic regression analysis to construct models to predict refactoring opportunities, although this statistical analysis technique has been successfully applied to detect other software quality aspects, such as fault proneness. This paper shows how to use logistic regression analysis to explore the ability of quality metrics, considered individually or in combination, to predict classes in need of ESR.

2.2. Class quality metrics

Several metrics have been proposed in the literature to measure the size of a class. The most commonly used are lines of code (LOC), number of local methods (NOM), and number of local attributes (NOA). By choosing these three measures, we studied three aspects of a class's size: the amount of functionality that a class provides (NOM), the amount of data necessary for it to function (NOA), and the sheer size in terms of statements (LOC).

Class coupling refers to the extent to which the class is coupled with other classes [25]. Several coupling metrics have been proposed in the literature to assess class coupling (e.g., [26,34,35,56,55]. A few existing coupling metrics measure class coupling locally (i.e., they require data available within the class of interest), and most coupling metrics require data available in the classes that are coupled with the class of interest. In this paper, we consider five local coupling measures, RFC, MPC, DAC1, DAC2, and OCMEC, which are defined in Table 1. Nonlocal measures were not included because they do not satisfy our goal of predicting the classes in need of ESR in advance, i.e., after the class is developed but before the entire system is completed. For example, Coupling Between Object Classes (CBO) [34] requires counting the classes, excluding the inherited ones, to which the class of interest is coupled, where a class A is coupled to another class B if the methods of A use attributes or methods of B, or vice versa. Therefore, to compute CBO, data must be collected from all classes that are coupled to the class of interest, which makes CBO a nonlocal measure. The theoretical validation of these coupling measures was studied by [25].

Class cohesion refers to the relatedness of class members (i.e., methods and attributes) [24]. Many class cohesion metrics have been proposed in the literature (e.g., [24,25,36,9,10,2,7,34,35,48,19,33,14,78,40,56,84,83,32,39]). In this paper, we consider 17 cohesion metrics, including LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, Coh, TCC, LCC, DC_D, DC_I, CC, SCOM, LSCC, CBMC, ICBMC, OL_n, and PCCC, as defined in Table 2. All these metrics are applicable during low-level design or the implementation phase, and they have been theoretically and empirically studied [27,29,3–8].

3. Design of the empirical study

The goal of the empirical study was to investigate the abilities of several selected quality metrics, considered individually or in combination, to predict classes in need of ESR. To perform the case study, we selected six Java open source systems of different domains. Table 3 reports the names, versions, domains, and sizes of the systems. The considered classes in the empirical study were all the classes in the selected systems except abstract classes and interfaces. Abstract classes and interfaces do not have available code for abstract methods, which prevents collection of the required data to measure the class internal quality with most of the considered metrics. The restrictions that were placed on the choice of these systems were that they (1) be implemented using Java, (2) be from different domains, and (3) have available source code. One of the selected systems is JHotDraw, which is a framework developed as an exercise for using design patterns and is well known for its good design [37]. We selected this system to be able to generally comment on the design quality of the other selected systems.

To investigate the relation between the quality metrics and the ESR opportunities, two pieces of data are required for each considered class: (1) whether the class is in need of ESR and (2) the values of different quality attributes of the class. The former data are required to validate a prediction model that is based on the latter data. Deciding whether a class is in need of ESR (i.e., the former data to be collected) is not a straightforward and systematic task. This decision requires not only inspection of the code of the class of interest but also careful inspection and analysis of how the other classes are using the features of the class of interest. Manually performing this analysis for every class of interest is time consuming and labor intensive. In addition, unless this code inspection is performed by the developer of the code, the inspector has to spend considerable time and effort to understand the semantics of the code before performing the required analysis. To overcome these problems, we automatically searched for and mutated each class that had a single subclass. By implementing this inheritance relation, the developer of the system is implicitly claiming that the features of only the subclass are used in some instances. Therefore, if the superclass and subclass are merged together, the merged class will be in need of ESR. This claim is supported by the facts that the

Table 3
Systems used in the empirical study.

System Id	System	Version	Domain	LOC (K)	No. of originally considered classes	No. of kept mutated classes	No. of removed mutated classes	No. of remaining classes
s_1	Art of Illusion [50]	2.5	3D studio system	88	463	16	11	452
s_2	FreeMind [43]	0.8.0	Hierarchical editing system	64	378	55	53	325
s_3	GanttProject [44]	2.0.5	Project scheduling application	39	435	22	15	420
s_4	JabRef [51]	2.3 beta 2	Graphical application	48	540	9	7	533
s_5	Openbravo [65]	0.0.24	Point-of-sale application	36	428	6	6	422
s_6	JHotDraw [52]	5.2	Java graphics framework	18	137	8	8	129

Table 4
Mean values of the considered metrics before and after mutation.

Metric	Mean values before mutation						Mean values after mutation					
	s_1	s_2	s_3	s_4	s_5	s_6	s_1	s_2	s_3	s_4	s_5	s_6
NOM	11.63	9.01	5.06	4.51	6.20	8.74	(+)12.36	(+)9.52	(+)5.53	(+)4.6	(+)6.29	(+)10.34
NOA	8.53	3.17	2.63	3.39	3.04	2.60	(+)9.08	(+)3.55	(+)2.8	(+)3.45	(+)3.09	(+)2.73
LOC	182.77	86.35	57.13	64.40	69.31	65.82	(+)190.49	(+)88.58	(+)59.92	(+)65.29	(+)70.22	(+)76.72
LCOM1	87.30	76.86	15.11	16.00	24.62	64.93	(+)119.99	(+)93.33	(+)21.44	(+)17.26	(+)26.13	(+)118.84
LCOM2	53.52	70.49	11.46	12.41	18.21	55.66	(+)82.24	(+)86.44	(+)17.43	(+)13.51	(+)19.5	(+)108.64
LCOM3	1.56	1.61	1.55	1.64	1.74	1.75	(+)1.59	(+)1.96	(+)1.6	(+)1.66	(−)**1.73**	(+)1.78
LCOM4	1.55	1.61	1.55	1.63	1.73	1.71	(+)1.59	(+)1.96	(+)1.6	(+)1.65	(−)**1.73**	(+)1.74
LCOM5	0.73	0.79	0.72	0.81	0.82	0.84	(+)0.73	(−)**0.75**	(+)0.73	(+)0.82	(−)**0.82**	(−)**0.8**
LSCC	0.32	0.28	0.49	0.63	0.50	0.39	(−)0.31	**(+)0.31**	(−)0.47	(−)0.63	(−)0.5	(−)0.38
CC	0.36	0.31	0.52	0.65	0.53	0.43	(−)0.36	**(+)0.34**	(−)0.51	(−)0.65	(−)0.53	(−)0.42
SCOM	0.44	0.32	0.56	0.69	0.56	0.42	(−)0.44	**(+)0.36**	(−)0.54	(−)0.69	(−)0.55	(−)0.41
Coh	0.47	0.41	0.63	0.71	0.62	0.49	(−)0.46	**(+)0.45**	(−)0.61	(−)0.71	(−)0.62	(−)0.47
TCC	0.53	0.38	0.47	0.52	0.45	0.46	**(+)0.53**	**(+)0.43**	(−)0.46	(−)0.52	**(+)0.45**	**(+)0.47**
LCC	0.62	0.43	0.51	0.55	0.49	0.51	**(+)0.63**	**(+)0.47**	(−)0.5	(−)0.55	**(+)0.5**	**(+)0.51**
DC_D	0.54	0.38	0.51	0.53	0.43	0.43	**(+)0.54**	**(+)0.43**	(−)0.5	(−)0.52	**(+)0.43**	**(+)0.44**
DC_I	0.67	0.44	0.57	0.57	0.50	0.49	**(+)0.68**	**(+)0.49**	(−)0.56	(−)0.56	**(+)0.5**	**(+)0.5**
CBMC	0.23	0.22	0.47	0.54	0.45	0.23	(−)0.22	**(+)0.25**	(−)0.45	(−)0.54	(−)0.45	(−)0.22
ICBMC	0.22	0.21	0.46	0.53	0.43	0.22	(−)0.21	**(+)0.24**	(−)0.44	(−)0.53	(−)0.43	(−)0.21
OL_2	0.23	0.22	0.47	0.54	0.45	0.23	(−)0.22	**(+)0.25**	(−)0.45	(−)0.54	(−)0.45	(−)0.22
PCCC	0.32	0.30	0.61	0.63	0.53	0.30	(−)0.32	**(+)0.34**	(−)0.59	(−)0.62	(−)0.53	(−)0.28
MPC	78.41	32.71	22.25	31.97	31.91	28.83	(+)82.01	(+)34.92	(+)23.27	(+)32.4	(+)32.36	(+)34.64
RFC	31.70	18.42	11.85	14.30	13.54	16.74	(+)33.1	(+)19.75	(+)12.38	(+)14.43	(+)13.72	(+)18.93
DAC	4.14	2.60	2.02	2.84	2.51	1.46	(+)4.4	(+)2.82	(+)2.16	(+)2.88	(+)2.55	(+)1.49
DAC2	2.75	2.08	1.57	1.65	1.55	1.19	(+)2.92	(+)2.16	(+)1.68	(+)1.67	(+)1.58	(+)1.21
OCMEC	4.91	3.10	2.23	2.00	2.25	3.07	(+)5.06	(+)3.29	(+)2.33	(+)2.01	(+)2.26	(+)3.28

JHotDraw system is well known for its good design and that the qualities of the other selected systems are not particularly different from those of JHotDraw, as generally depicted in Table 4. It is important to note that we did not merge any subclass with its superclass unless the superclass had only a single subclass. The motivation behind having a superclass with multiple subclasses was that the subclasses have common code that can be pulled up to a superclass to reduce code redundancy. This refactoring is called extract superclass refactoring, and its motivation is different than the motivation of ESR. Therefore, we excluded these classes from being determined to be in need of ESR.

We developed a Java tool to detect the classes that had single subclasses. Because the number of instances of these classes was not large (as shown in the seventh and eighth columns of Table 3), we manually applied sequences of the pull up method and pull up field refactoring activities, as suggested by Fowler [42], to the methods and attributes of the subclasses until they became empty. Following the justified recommendations by Beyer et al. [18], in the case of having overriding attributes or methods in the subclass, the overriding members were pulled up to the superclass, and the overridden members were removed. After performing the required sequence of pull up refactorings, the resulting empty subclasses were removed from the systems. Whenever a class is removed, the other classes that use the removed class have to be modified to refer to the superclass instead of the subclass. However, we did not perform this modification because we were using only the quality metrics that measure the quality of the class of interest with only the data available within the class. Therefore, the metrics were unaffected by this modification. The only case in which we did not remove the empty subclass was when it was inherited by only one class. In this case, this subclass, s_1, had to be mutated because it was a superclass of another class, s_2. We did not mutate class s_1 from its empty version. Instead, we mutated the original version of the class because mutating the empty version would have resulted in classes s_1 (the mutated version) and s_2 (the original version) being identical, which would violate our goal of having the classes considered to be in need of ESR contain the code of both the subclass and superclass. The last column of Table 3 shows the number of remaining classes in each system. For example, after we excluded the interfaces and abstract classes, the Art of Illusion system had 463 classes. We found that there were 16 superclasses with single subclasses (as reported in the seventh column of Table 3). Eleven of these subclasses were not inherited by any class, and therefore, after performing the pull up refactoring, these classes became empty and were removed (as reported in the eighth column of Table 3). Thus, there were 452 (i.e., 463-11) remaining classes (as reported in the last column of Table 3). After removing the empty classes and performing the mutation process, the total number of classes in all systems considered in this empirical study was 2281.

The empirical study considered three quality attributes, namely size, cohesion, and coupling. These quality attributes were somehow related to refactoring activities and other software attributes of interest to software practitioners (e.g., [25,29,11,9,10]. The restrictions that were placed on the choice of metrics to measure each quality attribute were that they (1) be implemented without analyzing the control structure of the code and (2) be localized within the class of interest. Unless automated, metrics based on analyzing the control structure of the code are not practical for software practitioners whereas our goal was to suggest a practical prediction model that relies on easily collected data. Metrics that require collecting data that are available outside the class of interest were not selected because they violate our goal of using the data collected only from the class itself to predict whether the class is in need of ESR. The selected metrics are listed in Table 4, and they were summarized in Section 2. It is important to note that the selected metrics do not represent all of the existing metrics in the literature. These metrics are example metrics of different internal quality attributes used to investigate the general relationship between the internal quality attributes and the existence of classes in need of ESR. For each class, all of the selected metrics locally quantify the corresponding internal quality attributes, which explains the relatively low number of the selected coupling measures. As indicated in Section 2.2, most of the existing coupling measures (e.g., CBO) require data available in all classes coupled to the class of interest. Therefore, a software engineer cannot use these metrics in a model that predicts the classes is in need of ESR in advance without waiting for other classes to be completed.

We developed our own Java tool to automate the size, cohesion, and coupling measurement process. For each of the considered classes, the tool analyzed the Java source code, extracted the required data, calculated the size, cohesion, and coupling values using the 25 considered metrics, and reported the results in an excel spreadsheet. Some of the selected metrics had undefined values for some classes. For example, LCOM1 was originally undefined when the class of interest had a single method. For all such cases, the tool set the metric value according to the recommendations proposed by Al Dallal [4], which resulted in modifying the metrics to always be applicable. This modification allowed us to apply the empirical study to all of the considered classes, which consequently made the results more general. The considered classes in the empirical study were the classes remaining after the mutation and class removal processes (i.e., the classes reported in the last column of Table 3). However, we also used the tool to collect data on the original classes before mutation (i.e., the classes reported in the sixth column of Table 3). The results of this version of classes were used to comment on the quality of the considered applications by comparing their results with those of JHotDraw. In addition, the results of the original classes were compared with the results of the mutated version to comment on the impact of ESR on the size, cohesion, and coupling quality attributes. That is, the results were provided to investigate whether the overall quality of the systems after applying the ESR (i.e., the systems before mutation) are better than those before applying the ESR (i.e., the mutated systems).

Because of space limitations, for each considered system, Table 4 shows only the mean of each considered metric. Appendix A reports the remaining descriptive statistics for the classes considered in the empirical study using each metric, including the minimum, 25% quartile, median, 75% quartile, maximum, and standard deviation values. For most of the metrics, the descriptive statistics show that most of the considered systems are generally better than JHotDraw in terms of quality. For example, the average NOM of the classes in most of the considered systems was less than that of JHotDraw. In addition, considering LCOM3, LCOM5, and PCCC, all systems had better cohesion than JHotDraw. Better quality means smaller size, coupling, and lack of cohesion values, as well as higher cohesion values. Considering the selected 25 metrics, systems s_1, s_2, s_3, s_4, and s_5 were found to be better than JHotDraw using 10, 4, 21, 21, and 17 metrics, respectively, among the selected 25 metrics. This observation suggests that the qualities of the considered systems are close, and frequently better, than the quality of JHotDraw, which is known for its good quality [57,17].

The signs between parentheses in Table 4 indicate whether the mean value was increased or decreased when the selected classes were mutated. The reported mean values were rounded to two places after the decimal point. As a result, some of the values appear to be the same, but before rounding, they were different. For example, the SCOM mean value for s_1 looks to be the same before and after mutation. However, the actual value (without rounding) after mutation was less than the value after mutation. This difference is indicated by the minus sign between the parentheses.

The expectation is that the mutated versions of the systems have classes in need of refactoring, and therefore, their quality is worse than that of the original version. In Table 4, we highlighted the values in boldface that did not match the expectations. Table 4 shows that all of the results of the size and coupling metrics matched the expectations. In addition, the results for most of the cohesion metrics matched the expectations. This observation gave a general indication that ESR improves the size, cohesion, and coupling quality attributes, as will be confirmed in the univariate regression analysis reported in Section 4. Table 3 shows that, except for the FreeMind system, low percentage of classes in each system were mutated, which indicates that, given classes of a system, it will be expected that a low percentage of classes will be in need of ESR. However, this observation does not mean that the problem considered in this paper is negligible and does not affect the overall quality of the systems under consideration. We used the paired z-test [47] statistical technique to compare the mean values, given in Table 4, of the considered metrics before and after mutation. The results show that the mean values of the considered quality metrics were significantly changed for systems s_3, s_4, and s_5. In addition, we analyzed the results of the mutated classes alone and applied the two columns z-test (Hines et al., 2003) statistical technique to compare the quality metric values of the classes before and after mutation. The results show that the values of all of the considered quality metrics were significantly changed for the classes of all of the considered systems.

Investigating the correlations among the considered metrics is beyond the scope of this paper. The correlations among most of the considered metrics were previously explored in [9,10,7].

To explore the relationship between the collected values of the metrics and the extent to which a class is in need of ESR, we applied logistic regression [49]. This standard and mature statistical method is based on a maximum likelihood estimation to predict the probability of occurrence of an event by fitting the data of other related variables. It has been widely and successfully applied to predict fault-prone classes [25,29,45,58], and to the best of our knowledge, it has not yet been applied to predict refactoring opportunities. Applying other analysis methods, such as the methods discussed by Briand and Wust [28], Subramanyam and Krishnan [74], and Arisholm et al. [12], and comparing the results with those obtained using logistic regression analysis is beyond the scope of this paper. In logistic regression, explanatory or independent variables are used to explain and predict dependent variables. A dependent variable can only take discrete values. The logistic regression model is called univariate when it includes only one explanatory variable and multivariate when it features several explanatory variables. In our context, the dependent variable can only take binary values and indicates whether the class is in need of ESR (i.e., whether the class is among the classes reported in the seventh column of Table 3). The explanatory variables were the size, cohesion, and coupling metrics.

Univariate regression is applied to study the capability of each metric, considered separately, to predict the classes in need of ESR. More specifically, our goals here included (1) investigating whether each of the considered metrics was useful in practice to predict classes in need of ESR, (2) comparing the considered metrics in terms of their abilities to predict classes that require ESR, and (3) drawing and empirically supporting general conclusions regarding the relation between each of the size, cohesion, and coupling quality attributes and the presence of classes in need of ESR. Multivariate logistic regression is applied to investigate the relationship between the values of the collected metrics considered in combination and the extent to which a class is in need of ESR. The goal of this study is constructing a practical model based on an optimized combination of the selected metrics to predict the classes in need of ESR.

4. Univariate logistic regression analysis

In univariate regression analysis, several resulting factors can be considered to obtain observations and draw conclusions. The estimated regression coefficient (ERC) is one of these factors. The larger the absolute value of the coefficient, the stronger the impact (positive or negative, according to the sign of the coefficient) of the metric on the probability that a class will require ESR. The sign of the ERC indicates the relation between the value of the metric and the probability of the class requiring ESR. That is, a positive sign indicates that the probability of the class requiring ESR increases as the value of the metric increases; a negative sign indicates the inversely proportional relation. The considered metrics had significantly different standard deviations, as shown in Appendix A. Therefore, to help compare the coefficients, we standardized the explanatory variables by subtracting the mean and dividing by the standard deviation and, as a result, they all had an equal variance of 1. In this way, we obtained the standardized coefficients, which are reported in Tables 5–7. In this case, the reported ERC values represent the variation in standard deviations in the dependent variable when there is a change of one standard deviation in their corresponding independent variable. The *p*-value is another factor to be considered in regression analysis, as it represents the probability of the coefficient being different from zero by chance. It also indicates the accuracy of the coefficient estimate. We used a typical significance threshold ($\alpha = 0.05$) for the *p*-value to determine whether a metric was a statistically significant refactoring predictor. That is, if the *p*-value was less than 0.05, we rejected the null hypothesis that the metric was not a significant ESR predictor.

To evaluate the prediction accuracy of logistic regression models, the traditional precision and recall evaluation criteria [64] can be used. In the context of this study, precision is defined as the number of classes correctly classified as in need of ESR divided by the total number of classes classified as in need of ESR. It measures the percentage of classes in need of ESR correctly classified as in need of refactoring. Recall is defined as the number of classes correctly classified as in need of ESR divided by the actual number of classes in need of ESR. It measures the percentage of classes in need of ESR correctly or incorrectly classified as being in need of ESR. Both precision and recall are dependent on the value of the probability threshold selected for classification. To evaluate the performance of a prediction model regardless of any particular threshold, we instead used the receiver operating characteristic (ROC) curve [49]. In this study, the ROC curve is a graphical plot of the ratio of classes correctly classified as in need of ESR versus the ratio of classes incorrectly classified as in need of ESR at different thresholds. The area under the ROC curve (AUC) shows the ability of the model to correctly rank classes as in need or not in need of ESR. A 100% ROC area represents a perfect model that correctly classifies all classes. The larger the ROC area, the better the model is in terms of classifying the classes. We decided to use the value of the AUC to evaluate the performance of the metrics in predicting classes in need of ESR because it is often considered a better evaluation criterion than standard precision and recall, as selecting a threshold is always somewhat subjective. We applied the following general rules to assess the classification performance according the AUC value [49]: AUC = 0.5 means that the classification is not good, $0.5 < \text{AUC} < 0.6$ means that the classification is poor, $0.6 \leqslant \text{AUC} < 0.7$ means that the classification is fair, $0.7 \leqslant \text{AUC} < 0.8$ means that the classification is acceptable, $0.8 \leqslant \text{AUC} < 0.9$ means that the classification is excellent, and $\text{AUC} \geqslant 0.9$ means that the classification is outstanding. Thresholds based on the ROC analysis for the selected metrics are considered practical if they fall at least within the acceptable range [72]. It is important to note that a metric might be found to be a statistically significant refactoring predictor (*p*-value < 0.05), but according to the AUC, it could be determined to be an impractical predictor.

We applied the boxplot statistical technique [69] on the collected quality data to detect outliers. A few outliers were detected for the nonnormalized metrics. However, we did not exclude any of the collected data because we found that the removal of the outliers did not lead to significant differences in the final regression analysis results. Tables 5–7 report the univariate logistic regression results for the size, cohesion, and coupling metrics, respectively. Because of space limitations, the tables only show the ERC and AUC values. The rest of the results are reported in Appendix B. In Tables 5–7, values of metrics found to be significant refactoring predictors (*p*-value < 0.05) are highlighted in boldface. To obtain

Table 5
Univariate logistic regression results for the considered size metrics.

Metric	s_1		s_2		s_3		s_4		s_5		s_6		All systems	
	ERC	AUC	ERC	AUC	ERC	AUC	ERC	AUC	ERC	AUC	ERC	AUC	ERC	AUC
NOM	**0.90**	**0.90**	**0.58**	**0.88**	**1.01**	**0.84**	**0.67**	**0.82**	**0.68**	**0.84**	**1.01**	**0.93**	**0.67**	**0.87**
NOA	**0.47**	**0.77**	**0.35**	**0.77**	**0.43**	**0.73**	0.08	0.60	0.27	0.80	0.34	0.76	**0.22**	**0.71**
LOC	**0.37**	**0.80**	0.18	0.74	**0.46**	**0.73**	0.24	0.77	**0.35**	**0.80**	**0.85**	**0.88**	**0.23**	**0.75**

Table 6
Univariate logistic regression results for the considered cohesion metrics.

Metric	s_1		s_2		s_3		s_4		s_5		s_6		All systems	
	ERC	AUC	ERC	AUC	ERC	AUC	ERC	AUC	ERC	AUC	ERC	AUC	ERC	AUC
LCOM1	**0.66**	**0.91**	0.07	0.90	**1.19**	**0.87**	**0.29**	**0.83**	**0.45**	**0.86**	**0.85**	**0.93**	**0.38**	**0.89**
LCOM2	**0.70**	**0.88**	0.08	0.91	**1.42**	**0.89**	**0.26**	**0.79**	**0.42**	**0.87**	**0.87**	**0.94**	**0.40**	**0.90**
LCOM3	**0.44**	**0.76**	**0.88**	**0.86**	**0.47**	**0.74**	0.28	0.60	−0.77	0.58	0.29	0.67	**0.42**	**0.75**
LCOM4	**0.44**	**0.76**	**0.88**	**0.86**	**0.47**	**0.74**	0.28	0.61	−0.76	0.58	0.26	0.66	**0.42**	**0.76**
LCOM5	0.30	0.68	**0.73**	**0.79**	0.31	0.68	0.30	0.62	0.08	0.55	0.42	0.68	**0.31**	**0.69**
LSCC	**−3.20**	**0.80**	**−4.54**	**0.86**	**−2.26**	**0.82**	−0.54	0.66	−3.13	0.79	−26.69	0.86	**−2.35**	**0.84**
CC	**−1.82**	**0.76**	**−3.82**	**0.86**	**−1.98**	**0.81**	−0.50	0.66	−1.96	0.76	−9.77	0.86	**−2.16**	**0.83**
SCOM	**−2.51**	**0.82**	**−4.79**	**0.87**	**−2.12**	**0.83**	−0.58	0.68	−2.04	0.79	−20.82	0.86	**−2.38**	**0.85**
Coh	**−1.99**	**0.81**	**−3.46**	**0.88**	**−1.47**	**0.82**	−0.52	0.66	**−1.66**	**0.82**	**−3.68**	**0.86**	**−1.94**	**0.85**
TCC	**−0.66**	**0.67**	**−2.82**	**0.78**	**−0.71**	**0.60**	−0.41	0.58	−0.61	0.58	−0.73	0.64	**−1.07**	**0.66**
LCC	−0.27	0.62	**−2.18**	**0.78**	**−0.50**	**0.58**	−0.20	0.58	−0.14	0.52	−0.62	0.63	**−0.80**	**0.65**
DC_D	**−0.71**	**0.71**	**−2.95**	**0.81**	**−1.03**	**0.71**	−0.51	0.64	−0.71	0.61	−0.65	0.64	**−1.18**	**0.71**
DC_I	−0.32	0.64	**−2.29**	**0.81**	**−0.81**	**0.68**	−0.27	0.60	−0.21	0.53	−0.56	0.62	**−0.90**	**0.69**
CBMC	−135.48	0.69	−40.18	0.66	**−1.12**	**0.67**	**−1.04**	**0.73**	−1.75	0.62	−25.56	0.57	**−1.63**	**0.66**
ICBMC	−508.75	0.69	−327.08	0.66	**−1.09**	**0.67**	**−1.10**	**0.73**	−1.68	0.61	−73.53	0.57	**−1.62**	**0.66**
OL_2	−307.29	0.69	−61.99	0.66	**−1.12**	**0.67**	**−1.06**	**0.73**	−2.28	0.62	−28.13	0.57	**−1.65**	**0.66**
PCCC	−443.69	0.83	**−7.56**	**0.82**	**−1.12**	**0.72**	**−1.31**	**0.79**	−2.70	0.77	−245.75	0.74	**−1.82**	**0.77**

Table 7
Univariate logistic regression results for the considered coupling metrics.

Metric	s_1		s_2		s_3		s_4		s_5		s_6		All systems	
	ERC	AUC	ERC	AUC	ERC	AUC	ERC	AUC	ERC	AUC	ERC	AUC	ERC	AUC
MPC	0.23	0.72	**−0.68**	**0.54**	0.18	0.64	0.30	0.68	0.09	0.61	**0.71**	**0.83**	0.10	0.61
RFC	**0.45**	**0.73**	**−0.72**	**0.59**	**0.34**	**0.66**	**0.43**	**0.67**	0.27	0.62	**0.75**	**0.84**	**0.21**	**0.60**
DAC	**0.37**	**0.73**	**0.37**	**0.78**	**0.46**	**0.75**	0.06	0.59	0.26	0.83	0.06	0.55	**0.18**	**0.73**
DAC2	**0.56**	**0.73**	**0.48**	**0.78**	**0.51**	**0.75**	0.16	0.58	0.33	0.69	0.11	0.54	**0.36**	**0.73**
OCMEC	**1.02**	**0.84**	−0.06	0.52	**0.54**	**0.62**	**0.50**	**0.69**	0.52	0.66	**0.98**	**0.87**	**0.43**	**0.65**

more general results with enough data, we combined the classes of all of the considered systems and reported the corresponding univariate regression results in the last two columns of Tables 5–7. Our criterion for analyzing the results was to ignore the insignificant results. These results cannot be relied onto reject the null hypothesis. For the rest of the results, we relied on the AUC value to determine whether the metric was a practical classifier to determine classes that require or do not require ESR.

4.1. Univariate regression results for size metrics

The results of the univariate regression analysis performed on the considered size metrics are reported in Table 5, and they lead to the following observations:

1. Considering the significant results (highlighted in boldface), all of the considered size metrics were found to be practical refactoring predictors for both cases when the systems were considered individually or in combination.
2. As expected, the positive signs of ERC for the size metrics consistently showed that the probability that a class is in need of ESR increases as the size increases and vice versa.
3. On the basis of both ERC and AUC values, the NOM metric was found to be the best ESR predictor among the size metrics. It was found to have an outstanding classification power for some of the selected systems and excellent classification power for the other systems. On the basis of the AUC values of the significant results, LOC and NOA were found to be the second and third best predictors, respectively.
4. In the mutation process, some classes are merged together. Therefore, the size of the mutated classes are larger than the size of the classes before being mutated. However, this fact does not imply that these classes will be necessary predicted to be in need of ESR. This is because the size of some classes, even after being merged with others in the mutation process, can be less than the average size of the classes and they can have prediction probabilities that are lower than the selected threshold value. For example, the average LOC of the classes considered in the empirical study (including the mutated and nonmutated classes) is 93.9. The number of classes that their LOC are less than the average LOC is 1644 classes. Among these classes, 49 classes (3%) are mutated ones, and therefore, they are flagged to be in need of ESR. However, if the selected prediction threshold that is based on the LOC value was selected to be 0.05 (i.e., the value found to be the prediction probability of the class of 93 lines of code), none of the 49 mutated classes will be detected to be in need of ESR. We found that each of the 49 classes were mutated by merging two relatively small sized classes (e.g., one of them was formed by merging a class of 13 LOC with its superclass which is of 22 LOC, which resulted in forming the class that has less LOC than the average LOC). As a result, it is not necessary that the mutated classes will have sizes that are larger than the average size. The size metrics were found to be good predictors not because of the mutation process, but due to the nature of the classes selected to be mutated. These classes are composed of source code of classes that inherit each other, and therefore, the *chances* that all or most of the features of these merged classes used by each of the classes that use the merged classes is low. For example, it was found that the number of classes that their LOC is higher than the average LOC is 637. Among these classes, 67 classes (10.5%) are mutated ones. Using the same prediction threshold (0.05), the 67 classes will be detected to be in need of ESR.

4.2. Univariate regression results for cohesion metrics

The results of the univariate regression analysis performed on the considered cohesion metrics are reported in Table 6, and they lead to the following observations:

1. LCOM1, LCOM2, and Coh were the only cohesion metrics that were found to be practical and significant ESR predictors for *most* of the considered systems. The remaining cohesion metrics were found to be insignificant predictors of the classes of at least half of the considered systems. However, when all of the data were combined, all the cohesion metrics became significant ESR predictors.
2. For the significant results, LCOM1, LCOM2, LCOM3, LCOM4, LSCC, CC, SCOM, Coh, DC_D, and PCCC were found to always lead to practical classification of classes. The rest of the cohesion metrics were found to be practical classifiers for only one of the considered systems, which indicates that, in practice, it is risky to rely on these metrics alone to predict classes that require ESR.
3. Considering the significant results only, the sign of ERC for each of the considered cohesion metrics consistently indicated that the probability for a class to be in need of ESR increases as the cohesion value decreases and vice versa.
4. On the basis of the AUC values, alternately, LCOM1 and LCOM2 were found to be the best ESR predictors among the considered cohesion metrics. Both metrics showed an outstanding classification power for the classes in some systems.
5. It is difficult to relate the cohesion measuring approach to the ability of the metric in predicting the ESR opportunities. For example, both Coh and LCOM5 followed the same approach of counting the number of distinct attributes accessed by each method whereas Coh was found to lead to a practical classification, and LCOM5 was a risky metric to rely on. LCOM1, LCOM2, TCC, LCC, DC_D, and DC_I followed the approach of counting the number of method pairs that share at least one attribute whereas LCOM1 and LCOM2 were among the best predictors, and TCC, LCC, and DC_I were among the risky metrics. All of LCOM3, LCOM4, CBMC, ICBMC, OL_2, and PCCC were connectivity-based metrics whereas only LCOM3, LCOM4, and PCCC were found to always lead to practical classification of classes. The only approach whose metrics were found to always lead to practical classification of classes was the approach of finding the similarities between each pair of methods in terms of their access to the class attributes. This approach is used by the LSCC, CC, and SCOM metrics.

4.3. Univariate regression results for coupling metrics

The results of the univariate regression analysis performed on the considered coupling metrics are reported in Table 7, and they lead to the following observations:

1. For the significant results, only DAC and DAC2 were found to always lead to practical (acceptable) classification of classes. Each of the rest of the coupling metrics was found to be a practical classifier for only one or two of the considered systems, which indicates that, in practice, it is risky to rely on these metrics alone to predict classes that require ESR.
2. Except for the impractical results reported for some coupling metrics applied to classes of system s_2, expectedly, the positive signs of ERC for the coupling metrics consistently indicated that the probability of a class to be in need of ESR increases as the coupling value increases and vice versa.
3. It is difficult to draw a conclusion as to which coupling metric is the best ESR predictor because there was no single system in which the results of all coupling metrics were significant. However, on the basis of the combined data, DAC and DAC2 were found to be the best predictors among the considered coupling metrics.

4.4. Univariate analyses: discussion

The univariate regression results showed that all of the size metrics and some of the cohesion and coupling metrics were significant and practical predictors of classes in need of ESR. The abilities of the metrics to predict these classes varied from one metric to another. The results support the hypothesis that classes in need of ESR have relatively low quality (i.e., relatively high size, coupling, and lack of cohesion values as well as relatively low cohesion values).

More specifically, a class that instantiates a large class potentially do not use all its attributes and methods. As a result, expectedly, large classes (in terms of NOM, NOA, or LOC) include features that are used in some instances and not in others, and therefore, such large classes are potential candidates of ESR. In addition, classes of low cohesion include lowly related components. Typically, a instantiating class uses features of an instantiated class that are related somehow to the features of the instantiating class. When the instantiated class exhibits low cohesion level, and therefore, it has lowly related features, these features are expected to be used in some instances and not in others, because it is highly unexpected that all of these unrelated features are related to the features of the instantiating class. As a result, classes of low cohesion level are expected to be in need of ESR. More specifically, the potential unrelated features in the classes that are in need of ESR cause (1) the number of methods that do not share common attributes to be relatively high (i.e., the LCOM1 and LCOM2 values to be relatively high), (2) the relative number of methods that share common attributes to be relatively low (i.e., the TCC, LCC, DC_D, and DC_I values to be relatively low), (3) the similarities between the methods in terms of their access to the common attributes to be relatively low (i.e., the LSCC, CC, and SCOM values to be relatively low), (4) the relative number of attributes accessed by the methods to be relatively low (i.e., the Coh value to be relatively low and the LCOM5 value to be relatively high), and (5) the connectivity of the graph that represents the cohesive relations among the features of the class to be relatively low (i.e., the CBMC, ICBMC, OL_2, and PCCC values to be relatively low and the LCOM3 and LCOM4 values to be relatively high).

Finally, a class that features high coupling with other classes is expected to perform different functionalities related to different classes. When such a class is instantiated, the instantiating class is expected to use only the functionalities of the instantiated class that are somehow related to the functionalities of the instantiating class but not the others. As a result, features of a class that exhibits high coupling, via method invocations (measured by RFC and MPC), types of attributes (measured by DAC1 and DAC2), or types of parameters (measured by OCMEC), are expected to be used in some instances and not in others, and therefore, such a class is potentially in need of ESR.

In the next section, we report the multivariate regression analysis that builds prediction models by taking into account multiple quality aspects and dimensions. Therefore, such multivariate models can be more accurate, precise, and general in predicting classes in need of ESR compared with univariate models.

5. Multivariate logistic regression analysis

To explore the relationship between the values of the collected metrics considered in combination and the extent to which a class is in need of ESR, we applied multivariate logistic regression [49]. The main difference between this analysis and the univariate one was that the multivariate analysis considered several explanatory variables whereas the univariate analysis considered only an explanatory variable. The goal of this analysis was to construct a practical and optimized model that included multiple quality metrics to predict the classes in need of ESR. Such a model is more general than the model including a single metric because the former model considers different aspects of quality that were found to be related to the problem of interest, and not all of these aspects are captured by a single metric. Typically, well-constructed multivariate models perform better than univariate models in terms of the ability to predict the dependent variable. The analysis explores how well a model of measures used in combination can predict classes in need of ESR.

Including all of the metrics in the model can produce the best result in terms of the AUC value. However, this strategy results in a model with a relatively high estimated standard error, which means that the model is highly dependent on the data set. This result contradicts our goal of having a general model, which is expected to perform well given any data set. In addition, the model that includes all of the considered metrics is difficult to use in practice because it exhausts the software engineer's time by applying all of the metrics. To solve these problems, a set of metrics must be selected to construct an optimized model. These metrics have to be selected in such a way that reduces the multicollinearity in the model, i.e., the existence of highly correlated metrics [49]. As a result, the constructed model does not necessary include the metrics that were found to be the best individual predictors, because, depending on the correlations between them, the existence of these metrics together may cause the multicollinearity in the model to increase. Two general approaches are followed to construct such an optimized model: forward selection and backward selection [49]. In the forward selection approach, the model construction process starts with no variables in the model, and the variables are added one by one if they are statistically significant to the prediction model. Conversely, in the backward selection approach, the model construction process starts with a model that includes all of the independent variables, and the statistically insignificant variables are removed from the model one by one. Different researchers have used different statistical criteria to select the variables to be added or deleted.

In this research, we tried both construction approaches. The first experiment was based on the backward selection approach. We selected the *p*-value to be the criterion for removing metrics from the model. In each step, we applied multivariate regression analysis and removed the metric that had the highest p-value from the model. The process continued until each of the remaining metrics had a p-value above an adjusted α. A typical value used for α is 0.05. However, to avoid the typical problem of inflation of type-I error rates in the context of multiple tests, we used an adjusted significance threshold using the Bonferroni adjustment procedure: α/n, where n is the number of variables that currently exist in the model [1]. This backward-based experiment does not take the AUC values into account. The second experiment was based on the forward selection approach. First, we ordered the metrics in a descending fashion according to the AUC reported in Section 4 (last column in Tables 5–7). On the basis of this order, in each step, a metric was added to the model and the regression analysis was performed again using the metrics that existed in the model at that moment. If the added metric was found to have a p-value greater than the adjusted α, the metric would be removed from the model. In addition, if the added metric caused a metric already in the model to become insignificant (p-value > adjusted α), the metric would be deleted from the model. We used $\alpha = 0.05$ and adjusted it as explained above. Because our forward-based model-construction process considered the AUC values of the individual metrics, we noticed that the resulting models had AUC values that were better than those of the models constructed using the backward approach. At the same time, the multicollinearities in the forward models were relatively low due to the consideration of the p-values while constructing the models. Therefore, because of space limitations, in this section we only report, discuss, and compare the results of the forward-based models.

We tested the multicollinearity in the models by obtaining the variance inflation factor (VIF) [61], a widely used measure of the multicollinearity of a variable with other variables in the model. We used the rule of thumb that a value of four indicated a multicollinearity problem.

We considered four different scenarios using size metrics only, cohesion metrics only, coupling metrics only, and all of the considered metrics. The goal was to investigate whether the quality attributes were complementary in predicting the classes in need of ESR. In the first scenario, the number of considered size metrics was low, which allowed us to try all possible combinations of size metrics to find the best model. In the other three scenarios, we performed both the backward and forward-based experiments and selected the best model. Because of space limitations, in this section we only report and discuss the results of the fourth scenario. The results of the first three scenarios are provided in Appendices C, D, and E.

Once a model was constructed, multivariate regression analysis resulted in building the following equation:

$$\pi(X_1, X_2, \ldots, X_n) = \frac{1}{1 + e^{-(C_0 + C_1X_1 + C_2X_2 + \cdots + C_nX_n)}}$$

In our context, π represented the probability that the class is in need of ESR, X_is were the quality metrics, and the C_i coefficients were estimated through maximization of a likelihood function (i.e., obtained using logistic regression analysis) [49]. In practice, the π of each class of interest has to be calculated. The software engineer has to pay more attention to the classes of relatively high π because these classes are candidates for ESR. A threshold needs to be set to determine which classes have to be tested for refactoring and which ones can be ignored. The choice of the threshold depends on the time and resources provided to perform the refactoring process. That is, a relatively high threshold can be applied in situations where time and resources are limited. Typically, this threshold results in reducing the size of the set that includes the classes predicted to be in need of refactoring. A relatively high percentage of this set of classes includes classes that are actually in need of refactoring. However, a relatively high percentage of classes in need of refactoring is not included in this set. As a result, in this case, the software engineer will not waste much time by examining predicted classes that are actually not in need of refactoring. On the other hand, the software engineer will miss the refactoring of the classes that are actually in need of refactoring because they were not predicted. Alternatively, a relatively low threshold enlarges the set of classes predicted to be in need of refactoring. This set of classes includes a relatively high percentage of classes that are actually in need of refactoring. However, a relatively high percentage of the classes in this set is not in need of refactoring. In this case, the software engineer will waste much of his time in examining classes that are not actually in need of refactoring. However, by examining all of the classes in the set, the software engineer will have relatively high confidence that a high percentage of the classes that are in need of refactoring were predicted.

Once the threshold is set, a confusion matrix can be constructed to show four values, denoted as TN, FP, FN, and TP, where T, F, N, and P refer to true, false, negative, and positive. In our context, the TN value represents the number of classes correctly predicted to not be in need of ESR, the FP value represents the number of classes incorrectly predicted as in need of ESR, the FN value represents the number of classes incorrectly predicted as not in need of ESR, and the TP value refers to the number of classes correctly classified as in need of ESR. In this case, the precision and recall criteria, which are defined in Section 4, can be formally obtained as follows [64]:

$$\text{precision} = \frac{TP}{TP + FP} \text{ and recall} = \frac{TP}{TP + FN}.$$

For example, given a set of 100 classes and a threshold of a model with 40% precision and 70% recall, these figures indicate that if the model predicts a set of 20 classes to be in need of refactoring, the predicted set will include 40% of the classes in need of refactoring among the 100 classes. Additionally, 70% of the predicted classes ($20 \times 0.7 = 14$ classes) are actually in need of refactoring, and the other six classes are not in need of refactoring. It is important to note that there is an inverse relationship between precision and recall factors. That is, a selected threshold that causes one of them to increase results in a decrease in the other.

Aside from the problem of being dependent on the selected classification threshold, the precision and recall factors only evaluate the ability of the model to predict the classes in need of refactoring. They do not represent the ability of the model to predict the classes not in need of refactoring. That is, the precision and recall factors are not affected by the percentage of classes correctly predicted as not in need of refactoring. In practice, ignoring the classes that are correctly predicted as not in need of refactoring saves the software engineer time and effort. To capture these aspects, inverse-precision and inverse-recall factors are calculated as follows [68]:

$$\text{inverse\_precision} = \frac{TN}{TN + FN} \text{ and inverse\_recall} = \frac{TN}{TN + FP}.$$

To consider both direct and inverse factors, the weighted average precision (WAP) and weighted average recall (WAR) are calculated as follows:

$$\text{WAP} = \frac{\text{precision} * (FN + TP) + \text{inverse\_precision} * (TN + FP)}{TN + FP + FN + TP} \text{ and}$$

$$\text{WAR} = \frac{\text{recall} * (FN + TP) + \text{inverse\_recall} * (TN + FP)}{TN + FP + FN + TP}.$$

The values of the WAP and WAR are still dependent on the selected threshold for the model, but they are more representative of the values of the confusion matrix than the traditional precision and recall values.

Table 8
Quality-based forward multivariate regression model.

Metric	Coefficient	Std. err.	p-Value	VIF
Intercept	−1.95	0.24	<0.0001	
NOM	0.09	0.02	<0.0001	2.05
SCOM	−3.69	0.63	<0.0001	2.27
PCCC	−2.03	0.54	0.0002	1.74
LCOM4	0.28	0.06	<0.0001	1.72
DC_D	−2.36	0.63	0.0002	1.64
MPC	−0.02	0.00	<0.0001	1.55

To obtain models that are more general than those based on the individual considered systems, we combined the collected data sets of all of the classes in the six considered systems. We used the Mahalanobis Distance [15] to detect outliers in the models that include multiple independent variables. However, we found that the removal of outliers did not lead to significant differences in the final multivariate regression analysis results.

To obtain a more realistic assessment of the predictive capacities of the constructed models, we used the V-cross-validation, a procedure in which the data set is partitioned into k sub-samples. The regression model is then built and evaluated k times. Each time, a different sub-sample is used to evaluate the WAP, WAR, and AUC of the model, and the remaining sub-samples are then used as training data to build the regression model.

5.1. Size, cohesion, and coupling-based multivariate regression model

We built two models based on all of the considered quality metrics, one using the forward selection approach and the other using the backward selection approach. The later model had an AUC of 0.934 (outstanding) and included seven metrics: NOM, LCOM1, LCOM4, Coh, DC_I, PCCC, and MPC. In the model, the NOM metric had a VIF of 6.05, which indicated the existence of a multicollinearity problem (VIF > 4). The forward-based model had a slightly better AUC of 0.935 (outstanding), but it was better than the backward-based model in terms of the number of included metrics and multicollinearity, as shown in Table 8. The three considered quality attributes, size, cohesion, and coupling, were represented in both the forward and backward models. The contents of the confusion matrices for the forward-based model and the related precision, recall, WAP, and WAR values at different classification thresholds are shown in Table 9. The results highlighted in boldface in Tables 9, 17, 19, and 21 are for the thresholds that result in having the number of classes classified as in need of ESR roughly equal to the number of classes that are actually in need of ESR [25]. Typically, this threshold results in optimizing the balance between the precision and recall values. The results demonstrate that the model based on the three quality attributes was the best in terms of all of the considered performance factors comparing to the results reported in Appendices C, D, and E. According to their needs and available time and resources, software engineers can use the values reported in Table 9 as a guide to determine which threshold to apply. Generally, the recommended threshold to be applied when using this model is 0.27 because, typically, this threshold results in a balance between the precision and recall values.

5.2. Other multivariate regression models

We built all possible size-based models, where each model included a possible combination of the considered size metrics. The results of the best model among all possible size-based models, in terms of AUC, are reported in Appendix C. The results show that the precision and recall values for all of the selected thresholds were low whereas the WAP and WAR values were high. This

Table 9
Classification results at different sample thresholds for the model described in Table 8.

π	TN	FP	FN	TP	Precision	Recall	WAP	WAR
0.1	1956	209	22	94	0.810	0.310	0.980	0.873
0.2	2077	88	38	78	0.672	0.470	0.966	0.934
0.27	**2118**	**47**	**45**	**71**	**0.612**	**0.602**	**0.961**	**0.959**
0.3	2130	35	47	69	0.595	0.663	0.959	0.968
0.4	2145	20	66	50	0.431	0.714	0.943	0.977
0.5	2152	13	89	27	0.233	0.675	0.923	0.978
0.6	2156	9	96	20	0.172	0.690	0.917	0.980
0.7	2157	8	104	12	0.103	0.600	0.911	0.976
0.8	2159	6	110	6	0.052	0.500	0.906	0.972
0.9	2161	4	111	5	0.043	0.556	0.905	0.976

observation indicated that the model was bad at predicting ESR and excellent at predicting the classes not in need of ESR.

We built two models based on the considered cohesion metrics, one using the forward selection approach and the other using the backward selection approach. The later model had an AUC of 0.913 (outstanding) and included five metrics: LCOM4, LCOM5, Coh, DC_I, and PCCC. The forward-based model had a slightly better AUC of 0.915 (outstanding) and included five metrics: LCOM4, LCOM5, Coh, DC_D, and PCCC. Both models were free of multicollinearity problems. The results of the forward-based model are reported in Appendix D. Some of the thresholds had either precision or recall values that were much better than those of the size-based model. The WAP and WAR values were slightly higher than those of the size-based model. These observations indicated that the cohesion-based model was better than the size-based model at predicting both the classes in need and the classes not in need of ESR. To obtain better results, the software engineer who uses the cohesion-based model must apply a threshold that is higher than the one to be applied when using the size-based model, as can be noticed when comparing the values reported in Appendix D with those reported in Appendix C.

Both forward and backward selection strategies resulted in building the same optimized model of coupling metrics. The model had an AUC of 0.743 (acceptable), was free of multicollinearity problems, and included three metrics, as shown in Appendix E. The results show that the coupling-based model was worse than the size- and coupling-based models in terms of all of the considered performance factors.

5.3. Multivariate analyses: discussion

The results reported in this section indicate that the size, coupling, and cohesion quality attributes are complementary in predicting the classes in need of ESR because they were all represented in the best optimized model. The optimized model based on the metrics that measure these quality attributes had an outstanding classification ability in determining classes that required or did not require ESR. Applying several metrics that require different pieces of data increases the effort put forth by the software engineer to predict the classes in need of ESR. However, in comparison to the results of the models based on the individual metrics, the results of the multi-attribute model (reported in Section 5.1) indicate that it is worthwhile to make this additional effort. For example, we studied, in more detail, the univariate regression model based on LCOM2, which was found to be the best univariate model in terms of AUC when all classes were considered, as indicated in Table 6. We found that the threshold that resulted in an optimized balance of the precision and recall values was 0.0555. The corresponding precision and recall values were 0.28 and 0.29, respectively, which were much worse than those of the multi-attribute model.

6. Threats to validity

6.1. Internal validity

Analyzing how other related classes are using the class of interest plays a role in precisely predicting whether the class is in need of ESR. However, our goal was to investigate whether we could predict the classes in need of ESR with metrics that analyze the class only locally. Our results show that with only the localized metrics, we were able to construct models that had outstanding classification abilities in determining the classes that were in need or not in need of ESR. One of the internal validity threats is that our analysis relies on the claim that the selected systems are of good quality and that the developers of the systems correctly designed the inheritance relations among the classes in the systems. However, we supported our claim by selecting a system widely known for its good quality, and we compared the other systems to it in terms of the considered quality attributes. The comparison results gave us confidence that the selected systems were of good quality, and thus, it is strongly expected that the inheritance relations in the systems were designed well. In addition, an alternative to building the dependent data required for the logistic regression is to have external developers search for the classes in need of ESR in the original code. We expect this alternative to be less reliable than our mutating approach because, typically, the developers of systems understand their systems better than external developers. Another internal validity threat is related to the metrics selected to predict the refactoring opportunities. Many more metrics are proposed in the literature to measure the size, cohesion, and coupling of a class. However, including all of these metrics was not our goal. Instead, we selected several metrics that use different measuring approaches to determine whether there was evidence that locally-based quality metrics are able to predict classes in need of ESR, which typically involves analyzing other related classes. Our results demonstrate that the selected metrics were able to achieve the intended purpose. Finally, the validity of the constructed multivariate regression models was determined using the same data set used in building the models. Therefore, the constructed models may not perform as well as they performed on the original data set. However, the V-cross-validation used in our empirical study is a widely applied validation technique, which suggests that the model will behave well when applied to other data sets.

6.2. External validity

Several external factors may restrict the generality and limit the interpretation of our results. The first factor is that all six of the considered systems were implemented in Java. Other programming languages, such as C++, allow for multiple inheritance relations. However, in this case, the version of the system that includes the mutated classes is expected to have a larger size, less cohesion, and higher coupling mean values than that constructed in the case of single inheritance. Therefore, in the case of a language that allows for multiple inheritance relations, it is expected that the selected metrics will be better able to predict the mutated classes than in the case of languages with single inheritance relations. The second factor is that all of the considered systems were open-source systems that may not be representative of all industrial domains. However, this practice is common in the research community. Though differences in design quality and reliability between open source systems and industrial systems have been investigated (e.g., [70,71,73]), there is not yet a clear, general result we can rely on. The third factor is that, although they are not artificial examples, the selected systems may not be representative in terms of the number and sizes of classes. However, in our empirical analyses, we paid considerable attention to factors related to the significance of the collected data and results, as discussed in Sections 4 and 5. To generalize the results, different systems written in different programming languages, selected from different domains, and including real-life, large-scale software should be taken into account in similar large-scale evaluations.

7. Conclusions and future work

This paper demonstrated how to apply logistic regression analysis to predict classes in need of ESR. The analysis showed a strong relation between the internal quality attributes of a class and its

Table A.1
The descriptive statistics of the considered metrics applied on Art of Illusion and FreeMind systems.

Metric	Art of illusion						FreeMind					
	Min	Max	25%	Med	75%	Std. dev.	Min	Max	25%	Med	75%	Std. dev.
NOM	1	102	4.00	8.00	15.00	13.58	1	126	4.00	7.00	12.00	11.01
NOA	0	114	3.00	7.00	12.00	10.47	0	53	1.00	2.00	4.00	4.62
LOC	6	2767	57.75	105.00	215.00	270.96	7	1042	38.00	70.00	99.00	95.07
LCOM1	0	4716	2.00	15.00	65.25	427.14	0	7875	2.00	18.00	55.00	491.80
LCOM2	0	4281	0.00	2.00	32.25	363.47	0	7875	0.00	11.00	53.00	488.00
LCOM3	0	15	1.00	1.00	1.00	1.74	0	15	1.00	1.00	2.00	1.91
LCOM4	0	15	1.00	1.00	1.00	1.74	0	15	1.00	1.00	2.00	1.91
LCOM5	0	2	0.56	0.78	0.89	0.37	0	2	0.57	0.83	0.98	0.42
LSCC	0	1	0.05	0.14	0.46	0.35	0	1	0.01	0.09	0.53	0.40
CC	0	1	0.12	0.22	0.50	0.33	0	1	0.03	0.16	0.60	0.39
SCOM	0	1	0.13	0.32	0.70	0.36	0	1	0.01	0.16	1.00	0.41
Coh	0	1	0.22	0.39	0.67	0.30	0	1	0.10	0.33	0.75	0.36
TCC	0	1	0.22	0.50	0.97	0.36	0	1	0.02	0.33	1.00	0.41
LCC	0	1	0.33	0.72	1.00	0.38	0	1	0.02	0.40	1.00	0.43
DC_D	0	1	0.28	0.58	0.81	0.33	0	1	0.03	0.33	0.75	0.39
DC_I	0	1	0.41	0.78	1.00	0.35	0	1	0.03	0.48	1.00	0.41
CBMC	0	1	0.00	0.00	0.15	0.40	0	1	0.00	0.00	0.50	0.42
ICBMC	0	1	0.00	0.00	0.06	0.40	0	1	0.00	0.00	0.22	0.42
OL_2	0	1	0.00	0.00	0.11	0.40	0	1	0.00	0.00	0.39	0.42
PCCC	0	1	0.00	0.00	1.00	0.45	0	1	0.00	0.03	1.00	0.44
MPC	0	1739	12.00	40.00	91.00	143.81	0	438	10.00	18.00	41.00	47.76
RFC	0	413	8.75	21.00	43.25	39.75	0	175	8.00	12.00	28.00	20.18
DAC	0	55	1.00	2.00	5.00	6.36	0	47	1.00	2.00	4.00	3.76
DAC2	0	19	1.00	2.00	4.00	3.26	0	26	1.00	2.00	3.00	2.15
OCMEC	0	25	2.00	4.00	7.00	4.06	0	25	2.00	3.00	4.00	2.34

need for ESR. Without analyzing the external relations between the class of interest and other classes, the models constructed using the considered quality metrics showed high abilities to segregate classes into those that were in need and those that were not in need of refactoring. In practice, the models can be used in different scenarios. One of the scenarios is applicable when a class or a set of classes is modified. In this case, the software engineer applies the proposed equation of the model to the set of related classes. Classes with a high probability of being candidates for refactoring have to be carefully inspected. The second scenario is applicable when a software engineer is given a large number of classes to review and perform the required refactoring activities wherever applicable. Inspecting the whole code and analyzing the relations among the classes to identify the classes in need of ESR would be time consuming and labor intensive. Instead, the software engineer can apply the proposed optimized model to all of the classes, given that the metrics are automated, and rank the classes according to their probability of needing ESR. The software engineer can set a

Table A.2
The descriptive statistics of the considered metrics applied on GanttProject and JabRef systems.

Metric	GanttProject						JabRef					
	Min	Max	25%	Med	75%	Std. dev.	Min	Max	25%	Med	75%	Std. dev.
NOM	1	40	2.00	4.00	7.00	5.41	1	54	2.00	3.00	5.00	5.25
NOA	0	44	1.00	2.00	4.00	4.04	0	135	0.00	1.00	3.00	8.51
LOC	6	616	27.00	43.00	69.00	60.97	7	1498	18.00	36.00	75.00	100.01
LCOM1	0	753	0.00	3.00	13.00	73.53	0	1304	0.00	1.00	7.00	72.68
LCOM2	0	726	0.00	1.00	9.00	68.25	0	1177	0.00	1.00	4.00	65.57
LCOM3	0	12	1.00	1.00	2.00	1.45	0	20	1.00	1.00	2.00	1.82
LCOM4	0	12	1.00	1.00	2.00	1.44	0	20	1.00	1.00	2.00	1.82
LCOM5	0	2	0.33	0.72	0.98	0.54	0	2	0.00	0.78	1.33	0.70
LSCC	0	1	0.07	0.33	1.00	0.42	0	1	0.15	1.00	1.00	0.43
CC	0	1	0.13	0.33	1.00	0.41	0	1	0.19	1.00	1.00	0.41
SCOM	0	1	0.13	0.50	1.00	0.41	0	1	0.29	1.00	1.00	0.39
Coh	0	1	0.33	0.58	1.00	0.34	0	1	0.40	1.00	1.00	0.35
TCC	0	1	0.00	0.34	1.00	0.43	0	1	0.00	0.51	1.00	0.45
LCC	0	1	0.00	0.47	1.00	0.44	0	1	0.00	0.75	1.00	0.46
DC_D	0	1	0.10	0.50	1.00	0.39	0	1	0.00	0.50	1.00	0.43
DC_I	0	1	0.10	0.53	1.00	0.42	0	1	0.00	0.71	1.00	0.45
CBMC	0	1	0.00	0.00	1.00	0.49	0	1	0.00	1.00	1.00	0.49
ICBMC	0	1	0.00	0.00	1.00	0.49	0	1	0.00	1.00	1.00	0.49
OL_2	0	1	0.00	0.00	1.00	0.49	0	1	0.00	1.00	1.00	0.49
PCCC	0	1	0.01	1.00	1.00	0.47	0	1	0.02	1.00	1.00	0.46
MPC	0	363	4.00	10.00	26.00	39.73	0	583	5.00	13.00	35.00	56.09
RFC	0	111	3.00	8.00	16.00	14.61	0	122	4.00	9.00	19.00	16.54
DAC	0	27	0.00	1.00	3.00	2.95	0	131	0.00	1.00	3.00	7.87
DAC2	0	16	0.00	1.00	2.00	2.07	0	22	0.00	1.00	2.00	2.71
OCMEC	0	15	1.00	2.00	3.00	1.80	0	15	1.00	2.00	3.00	1.71

Table A.3
The descriptive statistics of the considered metrics applied on Openbravo and JHotDraw systems.

Metric	Openbravo						JHotDraw					
	Min	Max	25%	Med	75%	Std. dev.	Min	Max	25%	Med	75%	Std. dev.
NOM	1	41	2.00	4.00	8.00	5.93	1	80	3.00	6.00	11.00	12.79
NOA	0	53	0.00	1.00	3.00	5.10	0	17	1.00	2.00	3.00	2.95
LOC	7	1007	19.00	39.00	87.00	93.85	7	687	24.00	46.00	85.00	98.13
LCOM1	0	653	1.00	4.00	21.00	73.64	0	3055	2.00	14.00	45.00	405.45
LCOM2	0	624	0.00	3.00	13.00	60.16	0	2950	0.00	8.00	35.00	395.28
LCOM3	0	15	1.00	1.00	2.00	1.74	0	11	1.00	1.00	2.00	1.89
LCOM4	0	15	1.00	1.00	2.00	1.74	0	11	1.00	1.00	2.00	1.82
LCOM5	0	2	0.50	0.82	1.17	0.58	0	2	0.63	0.88	1.00	0.45
LSCC	0	1	0.05	0.33	1.00	0.44	0	1	0.02	0.14	1.00	0.42
CC	0	1	0.10	0.40	1.00	0.42	0	1	0.05	0.26	1.00	0.41
SCOM	0	1	0.13	0.50	1.00	0.41	0	1	0.02	0.20	1.00	0.42
Coh	0	1	0.26	0.60	1.00	0.36	0	1	0.13	0.38	1.00	0.38
TCC	0	1	0.00	0.36	1.00	0.42	0	1	0.00	0.42	1.00	0.39
LCC	0	1	0.00	0.48	1.00	0.43	0	1	0.00	0.50	1.00	0.40
DC_D	0	1	0.00	0.34	0.80	0.39	0	1	0.10	0.40	0.67	0.36
DC_I	0	1	0.00	0.50	1.00	0.42	0	1	0.10	0.50	1.00	0.38
CBMC	0	1	0.00	0.00	1.00	0.48	0	1	0.00	0.00	0.00	0.40
ICBMC	0	1	0.00	0.00	1.00	0.49	0	1	0.00	0.00	0.00	0.40
OL_2	0	1	0.00	0.00	1.00	0.48	0	1	0.00	0.00	0.00	0.40
PCCC	0	1	0.00	0.50	1.00	0.47	0	1	0.00	0.00	0.50	0.42
MPC	0	841	3.00	10.00	33.00	68.41	0	485	7.00	18.00	36.00	59.41
RFC	0	165	2.00	7.00	16.00	18.35	0	187	6.00	13.00	22.00	24.20
DAC	0	45	0.00	1.00	3.00	4.42	0	16	0.00	1.00	2.00	2.22
DAC2	0	16	0.00	1.00	2.00	1.92	0	9	0.00	1.00	2.00	1.46
OCMEC	0	11	1.00	2.00	3.00	1.76	0	13	2.00	3.00	4.00	2.24

threshold according to the available time and resources to determine the set of classes to be analyzed. Then, the classes whose prediction probabilities are above the threshold will be analyzed.

In the future, we plan to apply a similar analysis to construct models that predict other types of refactoring activities, such as extract class and extract superclass refactoring. Each of these refactoring activities has its unique motivation, which is different than that of ESR. Therefore, although we expect that the selected metrics are potential predictors for these other types of refactoring activities, the metrics are expected to have different abilities of prediction than those of ESR. Consequently, the multivariate prediction models for the other types of refactoring activities are expected to include different metrics and feature different classification thresholds than those of the prediction models of ESR. In the future, we plan to incorporate other quality attributes into the models, including complexity and inheritance. Instead of mutating the classes for regression analysis, the classes of a system could be given to a set of researchers to manually search for the classes in need of refactoring. This set of data will be considered as the dependent variable in a logistic regression analysis.

Acknowledgments

The author would like to acknowledge the support of this work by Kuwait University Research Grant WI06/09. In addition, the author would like to thank Anas Abdin for assisting in collecting the quality results.

Appendix A. The descriptive statistics of the considered systems

Tables A.1–A.4.

Table A.4
The descriptive statistics of the considered metrics applied on all systems.

Metric	All systems					
	Min	Max	25%	Med	75%	Std. dev.
NOM	1	126	2.00	5.00	9.00	9.49
NOA	0	135	1.00	2.00	5.00	7.46
LOC	6	2767	25.00	51.00	101.00	153.00
LCOM1	0	7875	1.00	5.00	27.00	291.74
LCOM2	0	7875	0.00	1.00	16.00	269.66
LCOM3	0	20	1.00	1.00	2.00	1.75
LCOM4	0	20	1.00	1.00	2.00	1.75
LCOM5	0	2	0.50	0.79	1.00	0.54
LSCC	0	1	0.05	0.25	1.00	0.43
CC	0	1	0.10	0.33	1.00	0.41
SCOM	0	1	0.11	0.43	1.00	0.41
Coh	0	1	0.25	0.50	1.00	0.36
TCC	0	1	0.00	0.42	1.00	0.42
LCC	0	1	0.00	0.55	1.00	0.43
DC_D	0	1	0.06	0.47	1.00	0.39
DC_I	0	1	0.07	0.60	1.00	0.41
CBMC	0	1	0.00	0.00	1.00	0.47
ICBMC	0	1	0.00	0.00	1.00	0.48
OL_2	0	1	0.00	0.00	1.00	0.47
PCCC	0	1	0.00	0.25	1.00	0.48
MPC	0	1739	5.00	16.00	44.00	83.33
RFC	0	413	4.00	10.00	24.00	25.04
DAC	0	131	0.00	1.00	3.00	5.54
DAC2	0	26	0.00	1.00	3.00	2.52
OCMEC	0	25	1.00	2.00	4.00	2.73

Appendix B. The univariate regression analysis results

Tables B.1 and B.2.

Appendix C. The size-based multivariate regression model results

Tables C.1 and C.2.

Appendix D. The cohesion-based multivariate regression model results

Tables D.1 and D.2.

Table B.1
The univariate regression analysis results for Art of Illusion, FreeMind, GanttProject, and JabRef systems.

Metric	Art of illusion			FreeMind			GanttProject			JabRef		
	Std. err.	*p*-Value	Odd ratio	Std. err.	*p*-Value	Odd ratio	Std. err.	*p*-Value	Odd ratio	Std. err.	*p*-Value	Odd ratio
NOM	0.16	<0.0001	2.46	0.17	0.001	1.79	0.17	<0.0001	2.74	0.18	0.000	1.95
NOA	0.15	0.001	1.60	0.14	0.014	1.42	0.14	0.002	1.53	0.23	0.726	1.08
LOC	0.13	0.006	1.44	0.12	0.156	1.19	0.14	0.001	1.58	0.13	0.069	1.28
LCOM1	0.15	<0.0001	1.93	0.12	0.532	1.08	0.24	<0.0001	3.27	0.12	0.015	1.33
LCOM2	0.18	<0.0001	2.01	0.12	0.513	1.08	0.28	<0.0001	4.12	0.11	0.018	1.30
LCOM3	0.13	0.001	1.55	0.16	<0.0001	2.41	0.13	0.000	1.60	0.14	0.055	1.32
LCOM4	0.13	0.001	1.55	0.16	<0.0001	2.41	0.13	0.000	1.60	0.14	0.054	1.32
LCOM5	0.23	0.195	1.35	0.17	<0.0001	2.07	0.21	0.133	1.36	0.33	0.356	1.35
LSCC	1.40	0.023	0.04	1.21	0.000	0.01	0.77	0.003	0.10	0.34	0.116	0.58
CC	0.77	0.018	0.16	0.90	<0.0001	0.02	0.59	0.001	0.14	0.34	0.138	0.61
SCOM	0.85	0.003	0.08	1.12	<0.0001	0.01	0.61	0.001	0.12	0.33	0.076	0.56
Coh	0.59	0.001	0.14	0.61	<0.0001	0.03	0.34	<0.0001	0.23	0.32	0.103	0.60
TCC	0.29	0.023	0.52	0.60	<0.0001	0.06	0.28	0.010	0.49	0.36	0.257	0.67
LCC	0.25	0.276	0.77	0.41	<0.0001	0.11	0.24	0.041	0.61	0.34	0.548	0.82
DC_D	0.28	0.011	0.49	0.58	<0.0001	0.05	0.31	0.001	0.36	0.37	0.165	0.60
DC_I	0.24	0.175	0.72	0.41	<0.0001	0.10	0.26	0.002	0.44	0.34	0.416	0.76
CBMC	0.00	N.A.	N.A.	4367.48	0.993	N.A.	0.38	0.003	0.33	0.49	0.035	0.35
ICBMC	0.00	N.A.	N.A.	0.00	N.A.	N.A.	0.38	0.004	0.34	0.53	0.037	0.33
OL_2	0.00	N.A.	N.A.	8160.41	0.994	N.A.	0.38	0.003	0.33	0.50	0.034	0.35
PCCC	484.18	0.359	0.00	2.79	0.007	0.00	0.31	0.000	0.33	0.52	0.013	0.27
MPC	0.14	0.099	1.26	0.31	0.026	0.51	0.16	0.253	1.20	0.17	0.079	1.34
RFC	0.15	0.004	1.57	0.26	0.005	0.48	0.15	0.026	1.41	0.20	0.030	1.54
DAC	0.15	0.012	1.44	0.15	0.016	1.45	0.14	0.001	1.58	0.24	0.797	1.06
DAC2	0.16	0.001	1.75	0.17	0.004	1.61	0.14	0.000	1.67	0.27	0.541	1.18
OCMEC	0.18	<0.0001	2.77	0.16	0.717	0.94	0.15	0.000	1.72	0.19	0.008	1.65

Table B.2
The univariate regression analysis results for Openbravo, JHotDraw, and all systems.

Metric	Openbravo			JHotDraw			All systems		
	Std. err.	*p*-Value	Odd ratio	Std. err.	*p*-Value	Odd ratio	Std. err.	*p*-Value	Odd ratio
NOM	0.20	0.001	1.97	0.30	0.001	2.75	0.07	<0.0001	1.96
NOA	0.20	0.172	1.32	0.26	0.188	1.40	0.06	0.000	1.25
LOC	0.17	0.041	1.42	0.26	0.001	2.34	0.06	<0.0001	1.26
LCOM1	0.14	0.002	1.57	0.33	0.010	2.33	0.07	<0.0001	1.46
LCOM2	0.14	0.002	1.52	0.35	0.012	2.40	0.08	<0.0001	1.49
LCOM3	0.95	0.416	0.46	0.26	0.279	1.33	0.06	<0.0001	1.52
LCOM4	0.95	0.421	0.47	0.27	0.323	1.30	0.06	<0.0001	1.52
LCOM5	0.41	0.838	1.09	0.38	0.270	1.53	0.09	0.001	1.37
LSCC	2.22	0.159	0.04	16.98	0.116	0.00	0.37	<0.0001	0.10
CC	1.13	0.083	0.14	5.04	0.053	0.00	0.30	<0.0001	0.12
SCOM	1.12	0.068	0.13	13.24	0.116	0.00	0.31	<0.0001	0.09
Coh	0.73	0.024	0.19	1.63	0.024	0.03	0.20	<0.0001	0.14
TCC	0.51	0.228	0.54	0.46	0.108	0.48	0.14	<0.0001	0.34
LCC	0.42	0.737	0.87	0.41	0.128	0.54	0.12	<0.0001	0.45
DC_D	0.54	0.187	0.49	0.45	0.147	0.52	0.15	<0.0001	0.31
DC_I	0.42	0.612	0.81	0.41	0.170	0.57	0.12	<0.0001	0.41
CBMC	1.26	0.166	0.17	3823.22	0.995	N.A.	0.29	<0.0001	0.20
ICBMC	1.26	0.183	0.19	8705.97	0.993	N.A.	0.30	<0.0001	0.20
OL_2	2.00	0.256	0.10	3914.00	0.994	N.A.	0.30	<0.0001	0.19
PCCC	2.24	0.227	0.07	347.41	0.479	0.00	0.28	<0.0001	0.16
MPC	0.31	0.780	1.09	0.27	0.007	2.03	0.07	0.132	1.10
RFC	0.25	0.287	1.31	0.27	0.005	2.12	0.07	0.002	1.23
DAC	0.21	0.205	1.30	0.33	0.858	1.06	0.06	0.004	1.19
DAC2	0.24	0.175	1.39	0.32	0.740	1.11	0.06	<0.0001	1.43
OCMEC	0.28	0.060	1.68	0.29	0.001	2.67	0.06	<0.0001	1.54

Table C.1
Best size-based multivariate regression model.

Metric	Coefficient	Std. err.	*p*-Value	VIF
Intercept	−3.75	0.15	<0.0001	
NOM	0.18	0.02	<0.0001	1.96
LOC	−0.01	<0.01	<0.0001	1.96

Table C.2
Classification results at different sample thresholds for the model described in Table C.1.

π	TN	FP	FN	TP	Precision	Recall	WAP	WAR
0.1	2045	120	62	54	0.466	0.310	0.945	0.912
0.14	**2091**	**74**	**75**	**41**	**0.353**	**0.357**	**0.934**	**0.935**
0.2	2121	44	90	26	0.224	0.371	0.922	0.949
0.3	2139	26	97	19	0.164	0.422	0.916	0.959
0.4	2146	19	101	15	0.129	0.441	0.913	0.963
0.5	2149	16	104	12	0.103	0.429	0.911	0.964
0.6	2157	8	107	9	0.078	0.529	0.908	0.973
0.7	2158	7	108	8	0.069	0.533	0.907	0.973
0.8	2159	6	110	6	0.052	0.500	0.906	0.972
0.9	2161	4	112	4	0.034	0.500	0.904	0.973

Table D.1
Cohesion-based forward multivariate regression model.

Metric	Coefficient	Std. err.	*p*-Value	VIF
Intercept	1.79	0.97	0.064	
Coh	−5.08	0.68	<0.0001	1.73
PCCC	−1.60	0.55	0.003	1.69
LCOM4	0.32	0.05	<0.0001	1.47
DC_D	−3.79	0.71	<0.0001	2.94
LCOM5	−2.62	0.92	0.004	2.49

Table D.2
Classification results at different sample thresholds for the model described in this table.

π	TN	FP	FN	TP	Precision	Recall	WAP	WAR
0.1	1915	250	28	88	0.759	0.260	0.974	0.853
0.2	2065	100	44	72	0.621	0.419	0.961	0.927
0.27	**2109**	**56**	**54**	**62**	**0.534**	**0.525**	**0.953**	**0.951**
0.3	2130	35	58	58	0.500	0.624	0.949	0.966
0.4	2149	16	78	38	0.328	0.704	0.933	0.978
0.5	2156	9	101	15	0.129	0.625	0.913	0.977
0.6	2158	7	107	9	0.078	0.563	0.908	0.975
0.7	2159	6	110	6	0.052	0.500	0.906	0.972
0.8	2162	3	111	5	0.043	0.625	0.905	0.980
0.9	2163	2	115	1	0.009	0.333	0.902	0.965

Appendix E. The coupling-based multivariate regression model results

Tables E.1 and E.2.

Table E.1
Coupling-based multivariate regression model.

Metric	Coefficient	Std. err.	*p*-Value	VIF
Intercept	−3.72	0.15	<0.0001	
DAC2	0.15	0.04	<0.0001	1.56
OCMEC	0.21	0.04	<0.0001	1.60
MPC	−0.01	0.00	0.001	1.62

Table E.2
Classification results at different sample thresholds for the model described in this table.

π	TN	FP	FN	TP	Precision	Recall	WAP	WAR
0.1	2041	124	94	22	0.190	0.151	0.917	0.902
0.11	**2068**	**97**	**99**	**17**	**0.147**	**0.149**	**0.913**	**0.914**
0.2	2140	25	107	9	0.078	0.265	0.908	0.952
0.3	2151	14	109	7	0.060	0.333	0.906	0.960
0.4	2159	6	112	4	0.034	0.400	0.904	0.967
0.5	2164	1	112	4	0.034	0.800	0.904	0.989
0.6	2165	0	114	2	0.017	1.000	0.903	1.000
0.7	2165	0	116	0	0.000	Undefined	0.901	Undefined
0.8	2165	0	116	0	0.000	Undefined	0.901	Undefined
0.9	2165	0	116	0	0.000	Undefined	0.901	Undefined

References

[1] H. Abdi, Bonferroni and Sidak corrections for multiple comparisons, in: Neil Salkind (Ed.), Encyclopedia of Measurement and Statistics, Sage, Thousand Oaks, CA, 2007, pp. 1–9.
[2] J. Al Dallal, A design-based cohesion metric for object-oriented classes, International Journal of Computer Science and Engineering 1 (3) (2007) 195–200.
[3] J. Al Dallal, Mathematical validation of object-oriented class cohesion metrics, International Journal of Computers 4 (2) (2010) 45–52.
[4] J. Al Dallal, Improving the applicability of object-oriented class cohesion metrics, Information and Software Technology 53 (9) (2011) 914–928.
[5] J. Al Dallal, Measuring the discriminative power of object-oriented class cohesion metrics, IEEE Transactions on Software Engineering 37 (6) (2011) 788–804.
[6] J. Al Dallal, The impact of accounting for special methods in the measurement of object-oriented class cohesion on refactoring and fault prediction activities, Journal of Systems and Software 85(5) (2012) 1042–1057.
[7] J. Al Dallal, Fault prediction and the discriminative powers of connectivity-based object-oriented class cohesion metrics, Information and Software Technology 54 (4) (2012) 396–416.
[8] J. Al Dallal, Incorporating transitive relations in low-level design-based class cohesion measurement, Software—Practice & Experience, in press, http://dx.doi.org/10.1002/spe.2127.
[9] J. Al Dallal, L. Briand, An object-oriented high-level design-based class cohesion metric, Information and Software Technology 52 (12) (2010) 1346–1361.
[10] J. Al Dallal, L. Briand, A precise method-method interaction-based cohesion metric for object-oriented classes, ACM Transactions on Software Engineering and Methodology (TOSEM) 21(2) (2012) 8:1–8:34.
[11] M. Alshayeb, Empirical investigation of refactoring effect on software quality, Information and Software Technology 51 (9) (2009) 1319–1326.
[12] E. Arisholm, L. Briand, E. Johannessen, A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, Journal of Systems and Software 83 (1) (2010) 2–17.
[13] D.C. Atkinson, T. King, Lightweight detection of program refactorings, in: The 12th Asia–Pacific software engineering conference (APSEC), Taiwan, 2005.
[14] L. Badri, M. Badri, A proposal of a new class cohesion criterion: an empirical study, Journal of Object Technology 3 (4) (2004) 145–159.
[15] V. Barnett, T. Lewis, Outliers in Statistical Data, third ed., John Wiley and Sons, 1994. 584.
[16] G. Bavota, A. De Lucia, A. Marcus, R. Oliveto, A two-step technique for extract class refactoring, in: International Conference On Automated Software Engineering (ASE), Belgium, 2010, pp. 151–154.
[17] G. Bavota, A. De Lucia, R. Oliveto, Identifying extract class refactoring opportunities using structural and semantic cohesion measures, Journal of Systems and Software 84 (3) (2011) 397–414.
[18] D. Beyer, C. Lewerentz, F. Simon, Impact of inheritance on metrics for size, coupling, and cohesion in object oriented, in: The 10th International Workshop on Software Measurement: New Approaches in Software Measurement, Berlin, 2001, pp. 1–17.
[19] J. Bieman, B. Kang, Cohesion and reuse in an object-oriented system, in: Proceedings of the 1995 Symposium on Software reusability, Seattle, Washington, United States, 1995, pp. 259–262.
[20] B. Bois, S. Demeyer, J. Verelst, Refactoring – improving coupling and cohesion of existing code, in: The 11th Working Conference on Reverse Engineering (WCRE), Netherlands, 2004, pp. 144–151.
[21] B. Bois, T. Mens, Describing the impact of refactoring on internal program quality, in: The International Workshop on Evolution of Large-Scale Industrial Software Applications (ELISA), The Netherlands, 2003.
[22] C. Bonja, E. Kidanmariam, Metrics for class cohesion and similarity between methods, in: Proceedings of the 44th Annual ACM Southeast Regional Conference, Melbourne, Florida, 2006, pp. 91–95.
[23] D. Boshnakoska, A. Mišev, Correlation between object-oriented metrics and refactoring, ICT Innovations, Communication in Computer and Information Science (CCIS) 83 (2) (2011) 226–235.
[24] L. Briand, J. Daly, J. Wuest, A unified framework for cohesion measurement in object-oriented systems, Empirical Software Engineering – An International Journal 3 (1) (1998) 65–117.
[25] L. Briand, J. Daly, J. Wuest, A unified framework for coupling measurement in object-oriented systems, IEEE Transactions on Software Engineering 25 (1) (1999) 91–121.
[26] L. Briand, P. Devanbu, W. Melo, An investigation into coupling measures for C++, in: Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, United States, 1997, pp. 412–421.
[27] L. Briand, S. Morasca, V.R. Basili, Defining and validating measures for object-based high-level design, IEEE Transactions on Software Engineering 25 (5) (1999) 722–743.
[28] L. Briand, J. Wust, Empirical studies of quality models in object-oriented systems, Advances in Computers, Academic Press, 2002. 97–166.
[29] L. Briand, J. Wüst, H. Lounis, Replicated case studies for investigating quality factors in object-oriented designs, Empirical Software Engineering 6 (1) (2001) 11–58.
[30] S. Bryton, F. Abreu, Strengthening refactoring: towards software evolution with quantitative and experimental grounds, in: The 4th International Conference on Software Engineering Advances, Porto, 2009, pp. 570–575.
[31] H.S. Chae, Y.R. Kwon, D. Bae, A cohesion measure for object-oriented classes, Software—Practice & Experience 30 (12) (2000) 1405–1431.
[32] H.S. Chae, Y.R. Kwon, D. Bae, Improving cohesion metrics for classes by considering dependent instance variables, IEEE Transactions on Software Engineering 30 (11) (2004) 826–832.
[33] Z. Chen, Y. Zhou, B. Xu, A novel approach to measuring class cohesion based on dependence analysis, in: Proceedings of the International Conference on Software Maintenance, 2002, pp. 377–384.
[34] S.R. Chidamber, C.F. Kemerer, Towards a metrics suite for object-oriented design, object-oriented programming systems, languages and applications (OOPSLA), Special Issue of SIGPLAN Notices 26(10) (1991) 197–211.
[35] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Transactions on Software Engineering 20 (6) (1994) 476–493.
[36] S. Counsell, S. Swift, J. Crampton, The interpretation and utility of three cohesion metrics for object-oriented design, ACM Transactions on Software Engineering and Methodology (TOSEM) 15 (2) (2006) 123–149.
[37] I. Czibula, G. Serban, Improving systems design using a clustering approach, IJCSNS International Journal of Computer Science and Network Security 6 (12) (2006) 40–49.
[38] Eclipse, <http://www.eclipse.org/> (accessed July 2011).
[39] L. Etzkorn, S. Gholston, J. Fortune, C. Stein, D. Utley, P. Farrington, G. Cox, A comparison of cohesion metrics for object-oriented systems, Information and Software Technology 46 (10) (2004) 677–687.
[40] L. Fernández, R. Peña, A sensitive metric of class cohesion, International Journal of Information Theories and Applications 13 (1) (2006) 82–91.
[41] M. Fokaefs, N. Tsantalis, A. Chatzigeorgiou, J. Sander, Decomposing object-oriented class modules using an agglomerative clustering technique, in: International Conference on Software Maintenance (ICSM 2009), 2009, pp. 93–101.
[42] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1999.
[43] FreeMind, <http://freemind.sourceforge.net/> (accessed July 2011).
[44] GanttProject, <http://sourceforge.net/projects/ganttproject/> (accessed July 2011).
[45] T. Gyimothy, R. Ferenc, I. Siket, Empirical validation of object-oriented metrics on open source software for fault prediction, IEEE Transactions on Software Engineering 3 (10) (2005) 897–910.
[46] B. Henderson-sellers, Object-Oriented Metrics Measures of Complexity, Prentice-Hall, 1996.
[47] W. Hines, D. Montgomery, D. Goldsman, C. Borror, Probability and Statistics in Engineering, fourth ed., John Wiley & Sons Inc., 2003.
[48] M. Hitz, B. Montazeri, Measuring coupling and cohesion in object oriented systems, in: Proceedings of the International Symposium on Applied Corporate, Computing, 1995, pp. 25–27.
[49] D. Hosmer, S. Lemeshow, Applied Logistic Regression, second ed., Wiley Interscience, 2000.
[50] Illusion, <http://sourceforge.net/projects/aoi/> (accessed July 2011).
[51] JabRef, <http://sourceforge.net/projects/jabref/> (accessed July 2011).
[52] JHotDraw, <http://sourceforge.net/projects/jhotdraw/> (accessed July 2011).
[53] JRefactory, <http://jrefactory.sourceforge.net/> (accessed July 2011).
[54] Y. Kosker, B. Turhan, A. Bener, An expert system for determining candidate software classes for refactoring, Expert Systems with Applications 36 (6) (2009) 10000–10003.
[55] Y. Lee, B. Liang, S. Wu, F. Wang, Measuring the coupling and cohesion of an object-oriented program based on information flow, in: Proceedings of International Conference on Software Quality, Maribor, Slovenia, 1995, pp. 81–90.
[56] W. Li, S.M. Henry, Object-oriented metrics that predict maintainability, Journal of Systems and Software 23 (2) (1993) 111–122.
[57] A. De Lucia, R. Oliveto, L. Vorraro, Using structural and semantic metrics to improve class cohesion, in: IEEE International Conference on Software Maintenance, 2008, pp. 27–36.
[58] A. Marcus, D. Poshyvanyk, R. Ferenc, Using the conceptual cohesion of classes for fault prediction in object-oriented systems, IEEE Transactions on Software Engineering 34 (2) (2008) 287–300.
[59] T. Mens, T. Tourwé, A survey of software refactoring, IEEE Transactions on Software Engineering 30 (2) (2004) 126–139.
[60] N. Moha, Y. Guéhéneuc, L. Duchien, A. Meur, DECOR: a method for the specification and detection of code and design smells, IEEE Transactions on Software Engineering 36 (1) (2010) 20–36.
[61] R. O'Brien, A caution regarding rules of thumb for variance inflation factors, Quality and Quantity 41 (5) (2007) 673–690.
[62] M. O'Cinneide, D. Boyle, I. Moghadam, Automated refactoring for testability, in: The 4th Workshop on Refactoring Tools (WRT), Berlin, 2011.
[63] M. O'Keeffe, M. O'Cinneide, Search-based software maintenance, in: The 10th European Conference on Software Maintenance and Reengineering (CSMR), Italy, 2006, pp. 249–260.
[64] D. Olson, D. Delen, Advanced Data Mining Techniques, first ed., Springer, 2008.
[65] Openbravo, <http://sourceforge.net/projects/openbravopos> (accessed July 2011).
[66] E. Piveta, Improving The Search For Refactoring Opportunities on Object-Oriented And Aspect-Oriented Software, Ph.D. thesis, Univeridade Federal Do Rio Grande Do Sul, Porto Alegre, 2009.
[67] E. Piveta, M. Pimenta, J. Araujo, A. Moreira, P. Guerreiro, R. Price, Representing refactoring opportunities, in: The Symposium on Applied Computing (SAC), New York, 2009, pp. 1867–1872.

[68] D. Powers, Evaluation: form precision, recall and F-factor to ROC, School of Informatics and Engineering, Flinders University, Technical report SIE-07-001, 2007.
[69] P. Rousseeuw, I. Ruts, J. Tukey, The bagplot: a bivariate boxplot, The American Statistician 53 (4) (1999) 382–387.
[70] I. Samoladas, S. Bibi, I. Stamelos, G.L. Bleris. Exploring the quality of free/open-source software: a case study on an ERP/CRM system, in: 9th Panhellenic Conference in Informatics, Thessaloniki, Greece, 2003.
[71] I. Samoladas, G. Gousios, D. Spinellis, I. Stamelos, The SQO-OSS quality model: measurement based open-source software evaluation, Open Source Development, Communities and Quality 275 (2008) 237–248.
[72] R. Shatnawi, W. Li, J. Swain, T. Newman, Finding software metrics threshold values using ROC curves, Journal of Software Maintenance and Evolution: Research and Practice 22 (1) (2010) 1–16.
[73] D. Spinellis, G. Gousios, V. Karakoidas, P. Louridas, P.J. Adams, I. Samoladas, I. Stamelos, Evaluating the quality of open source software, Electronic Notes in Theoretical Computer Science 233 (2009) 5–28.
[74] R. Subramanyam, M. Krishnan, Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects, IEEE Transactions on Software Engineering 29 (4) (2003) 297–310.
[75] N. Tsantalis, A. Chatzigeorgiou, Identification of move method refactoring opportunities, IEEE Transactions on Software Engineering 35 (3) (2009) 347–367.
[76] N. Tsantalis, A. Chatzigeorgiou, Identification of extract method refactoring opportunities, in: The 13th European Conference on Software Maintenance and Reengineering, Germany, 2009, pp. 119–128.
[77] N. Tsantalis, A. Chatzigeorgiou, Identification of refactoring opportunities including polymorphism, Journal of Systems and Software 83 (3) (2010) 391–404.
[78] J. Wang, Y. Zhou, L. Wen, Y. Chen, H. Lu, B. Xu, DMC: a more precise cohesion measure for classes, Information and Software Technology 47 (3) (2005) 167–180.
[79] B. Xu, Y. Zhou, Comments on 'A cohesion measure for object-oriented classes' by H.S. Chae, Y.R. Kwon, D.H. Bae (Softw. Pract. Exper. 30 (2000) 1405–1431), Software—Practice & Experience 31(14) (2001) 1381–1388.
[80] B. Xu, Y. Zhou, More comments on 'A cohesion measure for object-oriented classes' by H.S. Chae, Y.R. Kwon, D.H. Bae (Softw. Pract. Exper. 30 (2000) 1405–1431), Software—Practice & Experience 33(6) (2003) 583–588.
[81] X. Yang, Research on Class Cohesion Measures, M.S. Thesis, Department of Computer Science and Engineering, Southeast University, 2002.
[82] L. Zhao, J. Hayes, Predicting classes in need of refactoring: an application of static metrics, in: Workshop on Predictive Models of Software Engineering (PROMISE), associated with ICSM 2006, 2006.
[83] Y. Zhou, J. Lu, H. Lu, B. Xu, A comparative study of graph theory-based class cohesion measures, ACM SIGSOFT Software Engineering Notes 29 (2) (2004) 1–13.
[84] Y. Zhou, B. Xu, J. Zhao, H. Yang, ICBMC: an improved cohesion measure for classes, in: Proceedings of International Conference on Software, Maintenance, 2002, pp. 44–53.