Information and Software Technology 92 (2017) 105–120

Contents lists available at ScienceDirect

Information and Software Technology

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

Predicting move method refactoring opportunities in object-oriented code

Jehad Al Dallal

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

ARTICLE INFO

Article history:
Received 23 March 2017
Revised 5 July 2017
Accepted 29 July 2017
Available online 31 July 2017

Keywords:
Object-oriented design
Move method refactoring
Class quality
Logistic regression analysis

ABSTRACT

Context: Refactoring is the maintenance process of restructuring software source code to improve its quality without changing its external behavior. Move Method Refactoring (MMR) refers to moving a method from one class to the class in which the method is used the most often. Manually inspecting and analyzing the source code of the system under consideration to determine the methods in need of MMR is a costly and time-consuming process. Existing techniques for identifying MMR opportunities have several limitations, such as scalability problems and being inapplicable in early development stages. Most of these techniques do not consider semantic relationships.

Objective: We introduce a measure and a corresponding model to precisely predict whether a class includes methods in need of MMR. The measure is applicable once a class has entered the early development stages without waiting for other classes to be developed.

Method: The proposed measure considers both the cohesion and coupling aspects of methods. In addition, the measure uses structural and semantic data available within the class of interest. A statistical technique is applied to construct prediction models for classes that include methods in need of MMR. The models are applied on seven object-oriented systems to empirically evaluate their abilities to predict MMR opportunities.

Results: The results show both that the prediction models based on the proposed measure had outstanding prediction abilities and that the measure was able to correctly detect more than 90% of the methods in need of MMR within the predicted classes.

Conclusions: The proposed measure and corresponding prediction models are expected to greatly assist software engineers both in locating classes that include methods in need of MMR and in identifying these methods within the predicted classes.

1. Introduction

Refactoring refers to altering the internal structure of existing object-oriented software code without changing its external behavior [23], and it aims to improve several quality attributes of the considered code, such as maintainability and reliability [4,23,35]. Fowler [23] defined several refactoring scenarios and explained how they can be applied. One of the defined scenarios, Move Method Refactoring (MMR), is applied to move a method of interest from its current class to the class in which the method is used the most often. In practice, MMR is one of the most frequently applied refactoring activities (e.g., [10,34]).

Determining the methods in need of MMR is a labor-intensive, time-consuming task because it entails inspecting, understanding, and analyzing not only the class in which the methods of interest exist but also the other classes to which the methods are directly or indirectly related. After identifying the methods in need of MMR, software engineers can use automated refactoring tools such as JRefactory [29] and Eclipse [21] to perform the actual refactoring.

Researchers have proposed various techniques to determine or predict opportunities for MMR (e.g., [6,7,12,31,45,52]). To determine whether a method is an MMR candidate, existing techniques require an analysis of not only the class to which the method belongs but also all other classes to which the method is related. In some cases, this requirement creates two problems. The first problem is the scalability of the technique (i.e., whether the technique is easily applicable to large software systems). The second problem is that the techniques are inapplicable when the system under consideration is still under development and is not yet complete. When developing a class, software engineers may prefer to check whether the class includes methods in need of MMR without waiting for other classes to be implemented. The cost

E-mail address: j.aldallal@ku.edu.kw

http://dx.doi.org/10.1016/j.infsof.2017.07.013
0950-5849/

of performing refactoring activities is lower when these activities are performed earlier. After all of the classes related to a method that requires MMR have been developed, MMR requires that these classes be modified. The cost for these modifications can be prevented if MMR is performed long enough before other classes are developed.

The empirical evaluations reported in all of the related studies show that although most of the actual MMR opportunities are correctly detected, relatively high percentages of methods are incorrectly identified as needing MMR (Type I errors). As far as we know, none of the related research studies considered the application of statistical-based approaches, which have been found useful in predicting other refactoring opportunities. This research addresses all of these limitations.

In this paper, we propose a measure to indicate the degree to which a method in a class is in need of MMR. The measure considers three types of relations, including method-attribute relations, method invocation relations, and conceptual relations. In addition, it considers a set of preconditions that account for compilation, behavior-preservation, and coupling issues. We show how to use this measure to construct a statistical-based model that predicts whether a class includes a method that needs MMR. In this case, once a class is predicted to include a method that needs MMR, the measure can be applied to indicate which of the class's methods are potentially in need of MMR. We empirically evaluated the proposed approach by performing three studies involving seven open-source systems. The first study explored the impact of individual measures incorporated into building the prediction model on the model's classification performance. The second study investigated the effect of the distribution of weights given to the three types of relations considered when they are incorporated into the proposed measure on the ability of the corresponding model to predict the classes that contain methods in need of MMR. The third study explored the proposed measure's precision when indicating the methods in need of MMR within the predicted classes.

The major contributions of this paper are as follows:

1. Proposal of a measure to indicate which methods are in need of MMR.
2. Construction of statistical-based models that have outstanding abilities to predict whether a class includes a method that needs MMR.
3. Exploration of the proposed measure's ability to indicate which of the methods included in the predicted classes are in need of MMR.

The paper is organized as follows. Section 2 reviews related work. Section 3 proposes the measure that indicates the MMR opportunities. Section 4 explains the design of the empirical study. Sections 5–7 report and discuss three empirical studies that both validate the proposed approach and illustrate its usefulness. Section 8 lists the issues that threaten the validity of the empirical studies. Finally, Section 9 concludes the paper and discusses future work.

2. Related work

In this section, we review and discuss related research in the area of MMR refactoring opportunity identification and provide an overview of the existing class cohesion measures that are adapted in this research.

2.1. Identifying MMR refactoring opportunities

Researchers have identified four stages of typical refactoring processes [33,43,54]. In the first stage, the refactoring candidates are determined. In the second stage, the advantage and cost of performing the refactoring are analyzed. In the third stage, the resulting code modification is performed. In the fourth and last stage, the performed code modification is examined for its preservation of the external software behavior. In this paper, we are interested in the first stage, during which the candidates for MMR are predicted or determined.

Researchers have proposed numerous techniques and developed several tools to assist in predicting or identifying MMR refactoring candidates. Bois et al. [14] and Seng et al. [48] identified and applied a set of preconditions that must be satisfied by a method to be considered as an MMR candidate. Fokaefs et al. [22] proposed identifying Feature Envy bad smells by examining the distance between each method in a class and the other methods in the class (inner entities) and comparing this distance to the distance between the method and the methods in each class to which the method is coupled (outer entities). This examination has to be applied to all of the methods in a system. The ratios of the average distances of inner entities to those of outer entities are calculated. Methods with high corresponding ratios are MMR candidates. The technique is implemented in a tool called JDeodorant and evaluated using two small examples. No justification is provided for the selected distance measures, and neither preconditions for the methods considered as MMR candidates nor preconditions for the target classes are provided.

Higo et al. [26] suggested the application of a code clone detection technique to identify opportunities for several refactoring activities. They suggested applying MMR when detecting duplicated methods in different classes with no common base class. Their technique for detecting MMR is not empirically evaluated.

Tsantalis and Chatzigeorgiou [52] suggested a technique to determine MMR opportunities. They defined a set of MMR preconditions to be satisfied both by the method to be moved and by the class to which the method can be moved. For each method that satisfies the preconditions, the distance between the method and each of the classes that satisfy the preconditions is calculated. The class that exhibits the smallest Jaccard distance is identified as a target class. No justification for selecting the Jaccard distance is provided. An evaluation of whether the MMR opportunities suggested by the technique match intuition is performed using a small application of 34 classes. In addition, the prediction of MMR opportunities in class c requires not only an analysis of the relationship among the members of class c but also an analysis of the relationship among the members in each class to which class c is coupled, which can cause scalability problems and requires availability of the code of the entire system. In other words, this technique cannot be applied in early design stages when the system is not yet complete.

Pan et al. [40] used a community detection algorithm, and Pan et al. [41] applied an evolutionary algorithm to analyze the graph that represents the class structure and construct an optimized graph. In both techniques, the original and optimized graphs were compared to identify Move Method Refactoring opportunities. Lee et al. [31] introduced a technique to identify MMR opportunities by detecting code clones. Identical methods that use more attributes of a class c than attributes of their classes should be moved to class c. Al Dallal and Briand [6] proposed the Low-level design Similarity-based Class Cohesion (LSCC) cohesion measure and demonstrated its usefulness in predicting MMR opportunities. Their work is based on the assumption that a method is predicted to be moved when its removal from the class causes the LSCC value of the class to improve. The technique does not consider other quality attributes such as coupling, and therefore it suffers from high false-positive (Type I error) percentages.

To test whether a method coupled with other classes is an MMR candidate, Sales et al. [45] suggested measuring the

similarity between the method and other methods in the same class and between the method and the methods in each class with which the method is coupled. The method is determined to be an MMR candidate if the similarity between the method in the class and the methods in another class is greater than the similarity between the method in the class and other methods in the same class. Sales et al. considered an existing similarity measure and compared its results with those based on other similarity measures. They mutated classes in 14 open-source systems, applied their technique (implemented in a tool called JMove), and compared the results with those based on using the JDeodorant tool. Although the results based on JMove are better than those based on JDeodorant, the results show that the application of the proposed technique results in high false-positive (Type I error) percentages.

Bavota et al. [12] and Oliveto et al. [37] proposed measuring the similarity between each pair of methods in a system. Two types of similarities are considered: similarity in terms of percentage of shared attributes and similarity in terms of method call dependency. Two corresponding matrices are built; one includes the shared-attributes-based similarity value for each pair of methods in the system and the other matrix holds the call-dependency value for each pair of methods in the system. In addition, a term-by-document matrix is built to hold the semantic data based on comments, identifiers, and literal strings presented in a method. The Relational Topic Model (RTM) takes the three matrices as an input and produces an RTM similarity matrix. A threshold is applied to identify MMR opportunities, and a set of preconditions is considered to suggest the target classes. The study has several limitations, including the following: (1) lack of precision evaluation, (2) potential scalability problems for large systems with huge numbers of methods, (3) the fact that the selection of the similarity measures is not justified, and (4) the fact that the application of the technique requires the availability of the entire implemented system (i.e., the technique is inapplicable when the system under development is not yet complete).

Alkhalid et al. [7], Czibula and Czibula [18,19], and Serban and Czibula [49] proposed clustering-based techniques to identify MMR opportunities. In these techniques, Alkhalid et al. [7] suggested using the similarities between methods within a class and in other classes, and Czibula and Czibula [18,19] and Serban and Czibula [49] suggested using the distances between the methods and attributes within a class and in different classes as the basis for performing the proposed clustering. Al Dallal [3] reported a systematic literature review of the research on refactoring opportunity identification, and Al Dallal and Abdin [4] reported a systematic literature review of the research on the impact of refactoring on code quality.

In summary, all of the proposed techniques require the availability of the complete system to perform the required analysis. In other words, the identification of MMR opportunities is not performed locally using only the data available within the class of interest. Therefore, these techniques cannot be used during the software development stages, during which the systems are not yet complete. In addition, such an analysis for the relationship between a method and all classes to which a method is coupled may cause scalability problems when a method is coupled to many other classes. Except for the study performed by Oliveto et al. [37] and extended by Bavota et al. [12], none of the other studies considered the semantic relations in addition to the structured relations. In this paper, we propose an approach to indicate whether a class includes a method that needs MMR and which of the methods in the indicated class is potentially in need of MMR using only data available locally within the class of interest and without requiring the entire system to be fully developed.

2.2. Class cohesion measures

Several measures have been proposed in the literature to quantify the degree of cohesion among the members of a class. These measures can be classified into structural and conceptual measures. The measures that consider the structural class cohesion follow different measurement approaches. In this research, we adapted three measures: Coh, TCC, and LSCC. We selected these measures because they (1) follow different measurement approaches, (2) are well validated both empirically and theoretically [1,5,6,16,17], and (3) have been found to be useful for predicting opportunities for various refactoring activities [2,30,36]. The measures are listed and defined in Table 1.

3. Move method refactoring indicator for a class (MMRIC) measure

The MMRIC measure introduced here provides an indication of whether the class of interest has a method potentially in need of MMR. The value of the MMRIC measure ranges over the interval [0, 1], where a value of zero indicates that the class of interest potentially does not have a method that needs MMR and a value of "1" indicates that the class has the highest possibility of including a method that needs MMR. The measure accounts for three main factors: (1) the structural cohesion, (2) the conceptual cohesion, and (3) the MMR preconditions. The measures are defined as follows.

3.1. Structural cohesion

We considered two types of relations to compute the structural cohesion of a method in a class: (1) relations attributable to a method-attribute reference and (2) relations attributable to method invocations. There are three major approaches for measuring cohesion based on a method-attribute reference: (1) counting the number of attributes referenced by a method, (2) counting the number of method pairs that share at least one attribute, and (3) measuring the similarity degree between each pair of methods by considering the number of attributes shared between each pair of methods. In this research, we selected Coh [15], TCC [13], and LSCC [6] as examples of each of the three measuring approaches, respectively. These three measures are selected because they are found to adhere to all of Briand et al.'s [15] cohesion properties of a class cohesion measure [1]. To measure the cohesion of a method i to other methods in a class with m methods and n attributes, the definitions of these selected measures are adapted as follows.

A method-attribute reference (MAR) matrix represents the method-attribute relations for a class with m methods and n attributes. In a MAR matrix, the cell of row i and column j has a value MAR(i,j) of "1" if the ith method references the jth attribute; otherwise, it has a value of "0". Based on the definition of Coh, we define the Method Cohesion of method i, MCoh(i), as the proportion of the attributes accessed by method i. The measure is formally defined as follows.

$$MCoh(i) = \begin{cases} 1 & \text{if } n = 0, \\ \frac{1}{n}\sum_{a=1}^{n} MAR(i,a) & \text{otherwise.} \end{cases} \quad (1)$$

In addition, based on the definition of LSCC, we define the low-level design of Similarity-based Method Cohesion for a method i, LSMC(i), to be the average similarity between method i and each other method in the class of interest. The value of LSMC for a method i is calculated as follows:

Table 1
Definitions of the considered class cohesion measures.

Class Cohesion Measure	Definition/Formula
Coh [15]	Coh = $\lambda/(m \times n)$, where m is the number of methods, n is the number of attributes, and λ is the summation of the number of attributes referenced by each method in a class.
Tight class cohesion (TCC) [13]	TCC = Ratio of the pairs of methods that access the same attributes. An attribute accessed by a method k can be direct if it occurs within the method k or indirect if it occurs in a method directly or transitively invoked by method k.
Low-level design similarity-based class cohesion (LSCC) [6]	$LSCC(C) = \begin{cases} 0 & \text{if } m = 0 \text{ or } n = 0, \\ 1 & \text{if m} = 1, \\ \frac{\sum_{i=1}^{n} x_i(x_i - 1)}{mn(m-1)} & \text{otherwise.} \end{cases}$, where n and m are as defined above, and x_i is the number of methods that access attribute i.

$$LSMC(i) = \begin{cases} 1 & \text{if n} = 0, \\ \frac{1}{n(m-1)} \sum_{k=1,k\neq i}^{m} \sum_{a=1}^{n} MAR(i,a) \wedge MAR(k,a) & \text{otherwise.} \end{cases} \quad (2)$$

Based on TCC, a method has a cohesive relation with another method if they access a common attribute. Accordingly, Tight Method Pair Cohesion (TMPC), which indicates whether a method i has a cohesive relation with method j, is defined as follows:

$$TMPC(i,j) = \begin{cases} 1 & \text{if } \sum_{a=1}^{n} MAR(i,a) \wedge MAR(j,a) \geq 1 \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

The Tight Method Cohesion (TMC) of method i is accordingly defined as:

$$TMC(i) = \begin{cases} 1 & \text{if n} = 0, \\ \frac{1}{m-1} \sum_{k=1,k\neq i}^{m} TMPC(i,k) & \text{otherwise} \end{cases} \quad (4)$$

Method Invocation (MI) is an $m \times m$ matrix that represents the method invocation relationship among methods of a class of interest, where m is the number of methods in the class. In an MI matrix, the cell of row i and column j has a value of "1" if the ith method invokes the jth method and a value of "0" otherwise. We define the Method Invocation-based Cohesion for a method i, MIC(i), to be the average number of methods invoked by method i. The measure is formally defined as follows:

$$MIC(i) = \frac{1}{m-1} \sum_{k=1,k\neq i}^{m} MI(i,k) \quad (5)$$

3.2. Conceptual cohesion

We followed the approach proposed by Marcus et al. [32] to use Latent Semantic Indexing (LSI) [20] to measure the textual cohesion between a method and the class in which it is defined. The textual cohesion considers the code comments and identifiers. The conceptual cohesion between a method and a class is determined by the degree of similarity between the vector that includes the textual information of the method of interest and the vector that includes the textual information of the class in which the method is defined. In this case, the textual information of the class includes all comments and identifiers of all methods except those of the method of interest. The measurement of the degree of similarity between the two vectors follows the LSI approach.

3.3. MMR preconditions

We considered three preconditions to be satisfied by a method to be predicted as in need of MMR. The first precondition is that the method must not be a special method (i.e., a constructor, destructor, setter, getter, or delegate method). These methods do not provide functionalities other than creating an object, destroying an object, assigning a value to an attribute, getting the value of an attribute, or delegating responsibility to another method. The second precondition is that the method must have a coupling relation to another class inside the system boundary (e.g., not a library class). We considered several types of coupling relations, including (1) defining a local variable of a class type, (2) invoking a method of another class, (3) accessing an attribute of a class type, and (4) returning an object of another class. The third precondition is that the method must not be declared as a static method because a static method belongs to the class in which it is defined but is not an instance of it [52]. It is important to note that performing MMR requires ensuring a set of preconditions, identified by Tsantalis and Chatzigeorgiou [52], to be satisfied in the target class. In this research, we are not interested in performing MMR. Instead, we are interested in predicting the methods that are in need of MMR, and therefore, the preconditions related to target classes are ignored.

3.4. MMR indicator

The MMR Indicator (MMRI) indicates the degree to which a method is potentially in need of MMR. We calculate the MMR indicator for each method i in a class c with m methods as follows.

$$MMRI(i) = \begin{cases} 0 & \text{if method m does notsatisfy MMR preconditions,} \\ 1 - (\alpha \times MAR\_Coh(i) + \beta \times MIC(i) + \gamma \times Concep\_Coh(i)) & \text{otherwise} \end{cases} \quad (6)$$

where $\alpha + \beta + \gamma = 1$ and MAR_Coh(i) is selected among the three considered measures of MCoh, TMPC, and TMC. Based on this definition, the degree to which a method is potentially in need of MMR increases if the method has a lower degree of structural and conceptual cohesion to other methods in the class and vice versa. The MMRI indicator is set to zero if the method does not satisfy the preconditions stated in Section 3.3. Such a method must not be moved from the class in which it is defined.

The MMRIC indicates the degree to which the class has a method that needs MMR, and it is defined as follows.

$$MMRIC(c) = \begin{cases} 0 & \text{if m} \leq 1, \\ \max_{i=1}^{m}(MMRI(i)) & \text{otherwise} \end{cases} \quad (7)$$

Based on this definition, the degree to which a class has a method that needs MMR increases as the maximum MMRI value among those of all methods of the class increases and vice versa. Once a class is designed, the developer can obtain the MMRIC value of the class and decide whether to pay more attention to

Table 2
Systems used in the empirical study.

System Id	System	Version	Domain	LOC	No. of classes	No. of target classes	Percentage of target classes
S1	OmegaT [39]	2.6.3.11	Text processing	121 K	377	102	27.06%
S2	JHotDraw [28]	5.2	Software development	25 K	148	30	20.27%
S3	PDFSAM [42]	2.2.2	Office/Business	68 K	277	65	23.47%
S4	Ant [8]	8.0.5	Java building tool	254 K	1242	235	18.92%
S5	FreePlane [24]	1.3.12	Mind mapping, Knowledge and project management	111 K	926	262	28.29%
S6	Gantt [25]	2.0.10	Project scheduling	51 K	419	75	17.90%
S7	TVbrowser [53]	3.4.1.0	Entertainment	90 K	543	104	19.15%

moving a method from the class to another class if the MMRIC value exceeds a certain threshold. This threshold is set based on empirical studies, as explained in the following section.

4. Design of the empirical study

This section reviews the goals, context, hypotheses, and variables considered in the empirical studies. In addition, it describes the data collection process, the statistical techniques applied to build the prediction models using the collected data, and the evaluation criteria considered to assess the classification performance of the constructed models.

4.1. Goals, context, hypotheses, and variables

The goal of the empirical study was to investigate the ability of the proposed MMR indicator to identify classes that include one or more methods potentially in need of MMR. More specifically, the empirical study answered the following research questions:

RQ1: *Which individual cohesion measure provides the best results when applied to measure the MMRIC?*
RQ2: *What is the effect of the parameters of the proposed approach on the obtained classification results?*
RQ3: *What is the performance of the MMRI in indicating the methods that require MMR?*

The empirical study was executed by the author of this paper and his research assistant, who has a B.Sc. in computer science, eleven years of experience in software development activities, and five years of experience in refactoring activities. The study involved seven open-source applications, which were Java systems from different domains. The names, versions, domains, and sizes of the selected systems are reported in Table 2. The systems were selected based on the following restrictions: (1) they are implemented using Java, (2) they are from different domains, and (3) their source code is available.

The empirical study considered several MMR prediction models that were built using different cohesion measures and different parameter settings and compared their prediction performances. For each of the constructed prediction models, we tested the following hypotheses:

H_0 (null hypothesis): The constructed MMR prediction model is an insignificant predictor for the classes that include methods in need of MMR.

H_1 (alternative hypothesis): The constructed MMR prediction model is a significant predictor for the classes that include methods in need of MMR.

In our analysis, the dependent variable was an indicator of whether a class included a method in need of MMR, and the independent variable was the MMRIC of that class.

4.2. Data collection

To build a model based on MMRIC and explore its ability to indicate classes that include methods potentially in need of MMR, it is necessary to collect two pieces of data for each of the considered classes: (1) whether the class includes methods in need of MMR and (2) the MMRIC value of the class. The first piece of data is used to validate a prediction model that is based on the second piece of data. To determine whether a class includes methods in need of MMR, we mutated a set of classes (source classes) by moving some of their methods to other classes (target classes). The source and target classes were randomly selected. The restrictions that we placed on the choice of any method to be moved from a source class to a target class are (1) that the method is not a special method (i.e., not a setter, a getter, a constructor, or a delegation method) and (2) that the placement of the method in the target class does not violate any of the compilation and behavior-preservation preconditions stated by Tsantalis and Chatzigeorgiou [52]. To better evaluate the ability of the proposed prediction approach, we did not move more than one method to a target class. In other words, each class in the mutated system is a target class for one moved method at most. As a result, in the mutation process, the number of target classes is identical to the number of moved methods. Three months of this research project were dedicated to completing the mutation step. The number of days assigned for performing the mutation process for each application was proportional to the number of classes in that application relative to the total number of classes in all seven applications. However, the percentages of target classes, which are reported in the last column in Table 2, differ among the seven applications because the time spent in the mutation process differed from one mutation case to another based on the number and types of relations between the moved method and other methods.

In the mutation process, we applied the MMR activity steps, as suggested by Fowler [23], and we ensured that the mutation process did not alter the system functionality by re-compiling and re-executing the application. The mutation process was manually performed by the research assistant and checked by the author of this paper, who randomly selected 50% of the target classes and ensured that the mutation process was performed correctly. The last two columns in Table 2 report the numbers and percentages of target classes.

We developed our own Java tool[1] to apply the proposed approach, obtain the MMRIC values for the Java classes, and report the results in a single sheet that can be processed using Microsoft Excel. In addition to the provided source code to be analyzed, the tool takes the values of the parameters α, β, and γ, included in Formula 6, as inputs. For each class in the analyzed system, the tool calculates the MMRIC values based on each of the proposed structural measures stated in Section 3.1 (i.e., each of MCoh, LSMC, and TMC). The resulting output sheet includes the three MMRIC values for each class in the analyzed system.

[1] The code is available at http://www.isc.ku.edu.kw/drjehad/PredictingMMR.htm.

4.3. Data analysis techniques

To construct a model based on the proposed MMRIC measure to predict classes that have methods in need of MMR, we applied logistic regression [27], a standard and mature statistical method based on maximum likelihood estimation. In logistic regression, dependent variables that can only take discrete values are predicted using independent variables. The logistic regression model is referred to as univariate when it has only one independent variable and multivariate when it includes several independent variables. In our context, the dependent variable indicates whether the class has a method that needs MMR (i.e., whether the class is among the classes reported in the seventh column of Table 2). Only one independent variable is considered: MMRIC. Accordingly, only univariate models are considered in this work. The goal is to compare the abilities of the models constructed based on MMRIC values using different structural-based cohesion measures at different parameter weights, which helps us answer our research questions.

The application of univariate regression analysis results in the construction of the following equation:

$$\pi(X) = \frac{1}{1 + e^{-(C_0 + C_1 X)}}$$

In our context, π represents the probability that the class includes a method that needs MMR, X is the MMRIC measure, and the C_i coefficients are estimated by maximizing the likelihood function [27]. In practice, it is necessary to calculate the π value of each class of interest. The classes of relatively high π values are potential candidates with methods in need of MMR; therefore, they require more attention from software engineers. The setting of the threshold for π must be determined based on the resources and time available to perform the refactoring process.

In our analysis, we consider the constructed model to be a statistically significant predictor if its p-value (i.e., the probability of the coefficient C_1 being different from zero by chance) is below the typical significance threshold ($\alpha = 0.05$). To assess the logistic regression models' ability to predict the classes that have methods in need of MMR, we constructed a confusion matrix that includes four values, denoted by TP, FP, FN, and TN, where P, N, T, and F refer to positive, negative, true, and false. In our context, the TP value represents the number of classes correctly classified as including methods in need of MMR, the FP value refers to the number of classes incorrectly predicted as including methods in need of MMR, the FN value refers to the number of classes incorrectly predicted as not including methods in need of MMR, and the TN value represents the number of classes correctly predicted to not include methods in need of MMR.

4.4. Classification performance

To assess the ability of the logistic regression models in predicting the classes that have methods in need of MMR, we applied the precision and recall evaluation criteria [38]. In our context, precision is defined as the number of classes correctly categorized as including methods in need of MMR divided by the total number of classes categorized as including methods in need of MMR. This provides the percentage of classes that include methods in need of MMR and that are correctly categorized as including methods in need of MMR. Recall is defined as the number of classes correctly categorized as including methods in need of MMR divided by the actual number of classes that include methods in need of MMR. This measures the percentage of classes that include methods in need of MMR, whether correctly or incorrectly categorized as including methods in need of MMR. The precision and recall values can be formally obtained as follows [38]:

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

Fmeasure is a classification performance measure that provides a single value considering both recall and precision and indicating the overall quality of the classification [9]. Its value is formally obtained as follows:

$$Fmeasure = \frac{2 \times precision \times recall}{precision + recall}.$$

It is important to note that the values of precision and recall each depend on the threshold used in the categorization process. In our context, when evaluating the classification performance of the constructed models, we considered the actual percentage of mutated classes (i.e., the percentage of target classes reported in Table 2) as the classification threshold. Using this value is better than relying on the default threshold value of 0.5, which does not consider information available from the field and has been found to be far from the actual percentages of mutated classes.

In addition to Fmeasure, we used the receiver operating characteristic (ROC) curve [27] to assess the performance of the prediction models. The ROC curve is independent of any classification threshold. In our context, the ROC curve is a graphical plot of the ratio of classes that are correctly categorized as including methods in need of MMR versus the ratio of classes that are incorrectly categorized as including methods in need of MMR at different thresholds. The area under the ROC curve (AUC) represents the capability of the prediction model to correctly classify classes as including methods either in need or not in need of MMR. We applied Hosmer and Lemeshow's [27] general rules to evaluate the categorization performance based on the AUC value, where $AUC = 0.5$ indicates that the classification is not good, $0.5 < AUC < 0.6$ indicates that the categorization is poor, $0.6 \leq AUC < 0.7$ indicates that the categorization is fair, $0.7 \leq AUC < 0.8$ indicates that the categorization is acceptable, $0.8 \leq AUC < 0.9$ indicates that the categorization is excellent, and $AUC \geq 0.9$ indicates that the categorization is outstanding. Practical models are those with AUC values greater than or equal to 0.7 [50].

Based on the boxplot statistical technique [44], we found that none of the collected MMRIC values are considered outliers. To assess the predictive capabilities of the constructed models more realistically, we applied k-fold cross-validation. In this procedure, a training data set is divided into k subsamples. The prediction model is then built and assessed k times. Each time, k-1 subsamples are used as training data to build the prediction model, and the remaining subsample is used to evaluate the model's classification performance. In our empirical study, we applied this validation technique when building each of the considered univariate prediction models, and we set the value of k to 10.

5. Empirical evaluation of the individual proposed cohesion measures

The first empirical study explored the abilities of the proposed cohesion measures when considered individually in MMRIC measurements. The goal was to determine which cohesion measure yielded the best prediction results when applied to measure the MMRIC.

5.1. Preparation

To answer RQ1, we empirically explored the performance of the models based on each of the four proposed structural cohesion measures defined in Section 3.1. For example, to explore the performance of the model based on MCoh, in Formula 6, we set α to 1 and β and γ both to zero, and we selected MCoh to represent MAR_Coh. We applied the resulting formula to each class

Table 3
The performance of the MMRIC based on individual measures.

Measure	Performance	S1	S2	S3	S4	S5	S6	S7
LSMC	Fmeasure	0.718	0.556	0.689	0.580	0.658	0.549	0.656
	AUC	0.902	0.852	0.911	0.917	0.823	0.908	0.911
TMC	Fmeasure	0.762	**0.556**	0.757	0.651	0.673	0.640	0.699
	AUC	0.907	**0.825**	0.917	0.922	0.825	0.916	0.914
MCoh	Fmeasure	0.742	**0.556**	0.731	0.629	0.691	0.617	0.685
	AUC	0.909	**0.825**	0.922	0.922	0.844	0.92	0.913
MIC	Fmeasure	**0.553**	**0.429**	**0.469**	0.450	0.523	**0.398**	0.424
	AUC	**0.726**	**0.737**	**0.676**	0.77	0.668	**0.733**	0.703
Concep.	Fmeasure	0.553	0.327	0.412	0.424	0.489	0.381	0.432
	AUC	0.771	0.658	0.66	0.785	0.683	0.775	0.771

in each application and obtained the corresponding MMRIC value.[2] For each application, the obtained MMRIC values for the classes in the application and the corresponding binary data indicating whether the classes were mutated were used to build a logistic regression prediction model. Given that we consider five different cohesion measures and seven applications, the total number of constructed models is 35.

5.2. Analysis and interpretation of the results

The Fmeasure and AUC values of the models are reported in Table 3, where the results of the models that were found to have insignificant predictors (p-value > 0.05) are highlighted in boldface. The detailed results, including the model coefficients, p-values, TN, FP, FN, TP, precision, and recall values, are reported in Appendix A.

The results reported in Table 3 and Appendix A lead to the following observations:

1. Most of the models were found to be significant predictors for classes that included methods in need of MMR (i.e., for these models, H_0 is rejected and H_1 is accepted). In addition, most of the models constructed using MMRIC, when only MIC is considered, are found to be insignificant predictors (i.e., for these models H_0 is accepted and H_1 is rejected). The rest of the observations are only related to significant predictor models.
2. The classification performance of the models based on method-attribute relations (i.e., LSMC, TMC, and MCoh) are considerably better than those based on method invocation relations (i.e., MIC) and conceptual relations (i.e., Concep.) in terms of both Fmeasure and AUC.
3. Based on AUC results, all of the models based on method-attribute relations are considered either excellent or outstanding classifiers, whereas the models based on method invocation relations or conceptual relations are considered either fair or acceptable classifiers. Accordingly, it is potentially risky to rely exclusively on method invocation relations or conceptual relations when building models to predict the classes that include methods in need of MMR. When building the prediction models, the data based on method invocation relations or conceptual relations can be incorporated with those based on method-attribute relations to potentially improve the classification performance of the prediction models, as will be empirically shown in Section 6. Our interpretation of the observation that the models based on method-attribute relations are better than the others is that method-attribute relations were mostly found to capture more cohesion aspects than method invocations and conceptual relations [5,32]. Typically, the measures based on method invocation relations and conceptual relations are used in combination with those based on method-attribute relations to improve the abilities of the models to predict various quality aspects such as fault proneness [5,32] or support maintenance activities such as refactoring [11,12].
4. The AUC values of the models based on method-attribute relations vary slightly across the various measures, empirically indicating that the approach considered for measuring the cohesion based on method-attributes relations is potentially not the key factor affecting the classification performance of the constructed prediction models. Instead, the key factor is found to be the code artifacts considered in calculating the cohesion degree.
5. The Fmeasure values of the models range between 0.327 and 0.762, and most of them are below 0.7. This is because the selected thresholds caused a relatively high number of classes to be incorrectly classified as including methods in need of MMR. The Fmeasure values change as the threshold value changes, which is why AUC is a more reliable indicator for the classification goodness of the models.

In general, the results indicate that the models constructed based on the individual measures considered have different prediction abilities and that the models based on method-attribute relations have considerably better prediction abilities than those based on method invocation relations and conceptual relations. The next section investigates whether consideration of different types of relations together, based on Formula 6, to indicate the classes including methods in need of MMR improves the prediction abilities of the models.

6. Empirical study for MMRIC parameter setting

The second empirical study explored the effect exerted by the parameters of the proposed approach on the obtained classification results. The goal was to determine the parameter values that yielded the best prediction results.

6.1. Preparation

To answer RQ2, we obtained the MMRIC values for different value combinations of α, β, and γ in Formula 6 and explored the abilities of the corresponding prediction models to predict the classes that include methods in need of MMR. This study has two goals. The first goal is to empirically explore the impact of MMRIC parameter setting on the prediction performance of the resulting models. The second goal is to empirically determine the parameter values that result in building the model with the best ability to predict the classes that include methods in need of MMR. To obtain the data required to answer RQ2, we extended our tool to automatically vary the weights of the parameters α, β, and γ, given in Formula 6, and calculate MMRIC accordingly. The tool varied the

[2] The raw data are available at http://www.isc.ku.edu.kw/drjehad/PredictingMMR.htm.

Table 4
Considered values of α, β, and γ.

Case	α	β	γ	Case	α	β	γ	Case	α	β	γ
1	0	0	1	23	0.2	0.1	0.7	45	0.4	0.6	0
2	0	0.1	0.9	24	0.2	0.2	0.6	46	0.5	0	0.5
3	0	0.2	0.8	25	0.2	0.3	0.5	47	0.5	0.1	0.4
4	0	0.3	0.7	26	0.2	0.4	0.4	48	0.5	0.2	0.3
5	0	0.4	0.6	27	0.2	0.5	0.3	49	0.5	0.3	0.2
6	0	0.5	0.5	28	0.2	0.6	0.2	50	0.5	0.4	0.1
7	0	0.6	0.4	29	0.2	0.7	0.1	51	0.5	0.5	0
8	0	0.7	0.3	30	0.2	0.8	0	52	0.6	0	0.4
9	0	0.8	0.2	31	0.3	0	0.7	53	0.6	0.1	0.3
10	0	0.9	0.1	32	0.3	0.1	0.6	54	0.6	0.2	0.2
11	0	1	0	33	0.3	0.2	0.5	55	0.6	0.3	0.1
12	0.1	0	0.9	34	0.3	0.3	0.4	56	0.6	0.4	0
13	0.1	0.1	0.8	35	0.3	0.4	0.3	57	0.7	0	0.3
14	0.1	0.2	0.7	36	0.3	0.5	0.2	58	0.7	0.1	0.2
15	0.1	0.3	0.6	37	0.3	0.6	0.1	59	0.7	0.2	0.1
16	0.1	0.4	0.5	38	0.3	0.7	0	60	0.7	0.3	0
17	0.1	0.5	0.4	39	0.4	0	0.6	61	0.8	0	0.2
18	0.1	0.6	0.3	40	0.4	0.1	0.5	62	0.8	0.1	0.1
19	0.1	0.7	0.2	41	0.4	0.2	0.4	63	0.8	0.2	0
20	0.1	0.8	0.1	42	0.4	0.3	0.3	64	0.9	0	0.1
21	0.1	0.9	0	43	0.4	0.4	0.2	65	0.9	0.1	0
22	0.2	0	0.8	44	0.4	0.5	0.1	66	1	0	0

weight of each parameter, starting from 0 and increasing it each time by 0.1 until reaching 1. The total number of resulting combinations is 66, as given in Table 4. These are the only possible combinations that ensure that $\alpha+\beta+\gamma=1$. Therefore, for each of the three structural measures (i.e., each of MCoh, LSMC, and TMC), the tool obtained 66 MMRIC values for each class in each selected system. In other words, the tool calculated $66\times3=198$ MMRIC values for each class in each selected system.[3]

For each system, we constructed 198 logistic regression models. Each model is based on one set of MMRIC values of the classes in the system among the 198 MMRIC sets. The total number of constructed models for the seven systems was $198\times7=1386$ models. Finally, we obtained the AUC and Fmeasure values[4] for each model and represented the results in the charts given in Fig. 1. The x-axis of each chart in Fig. 1 represents the cases reported in Table 4, and the y-axis represents the AUC and Fmeasure values. For each of the seven selected systems, a chart in Fig. 1 shows how AUC and Fmeasure values vary across different settings of α, β, and γ and different selections of method-attribute relations measures.

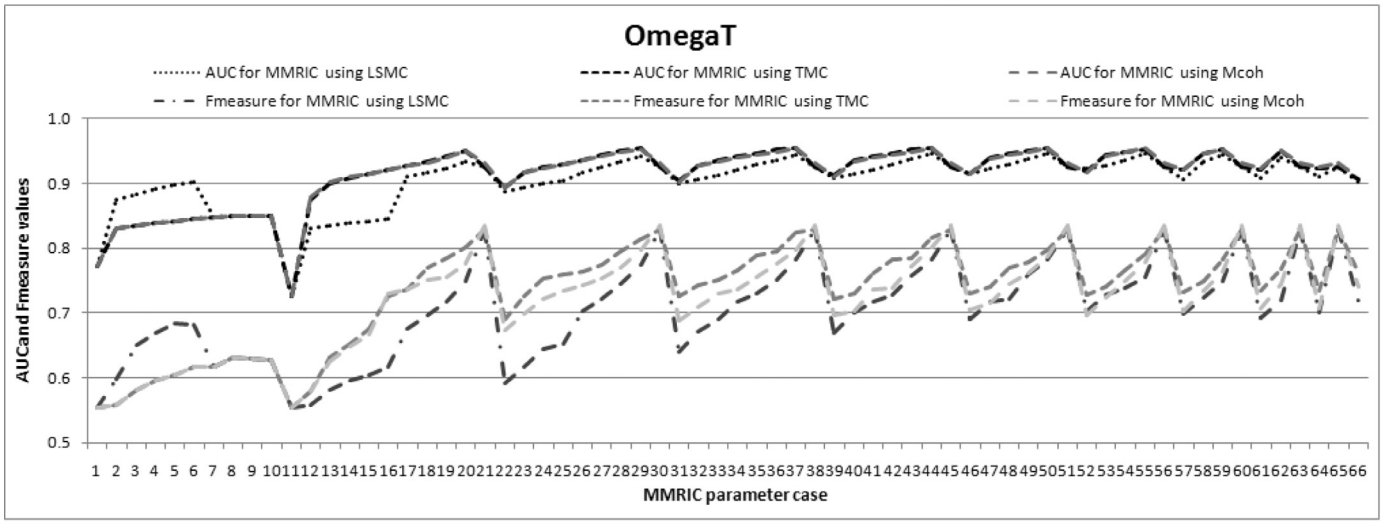
6.2. Analysis and interpretation of the results

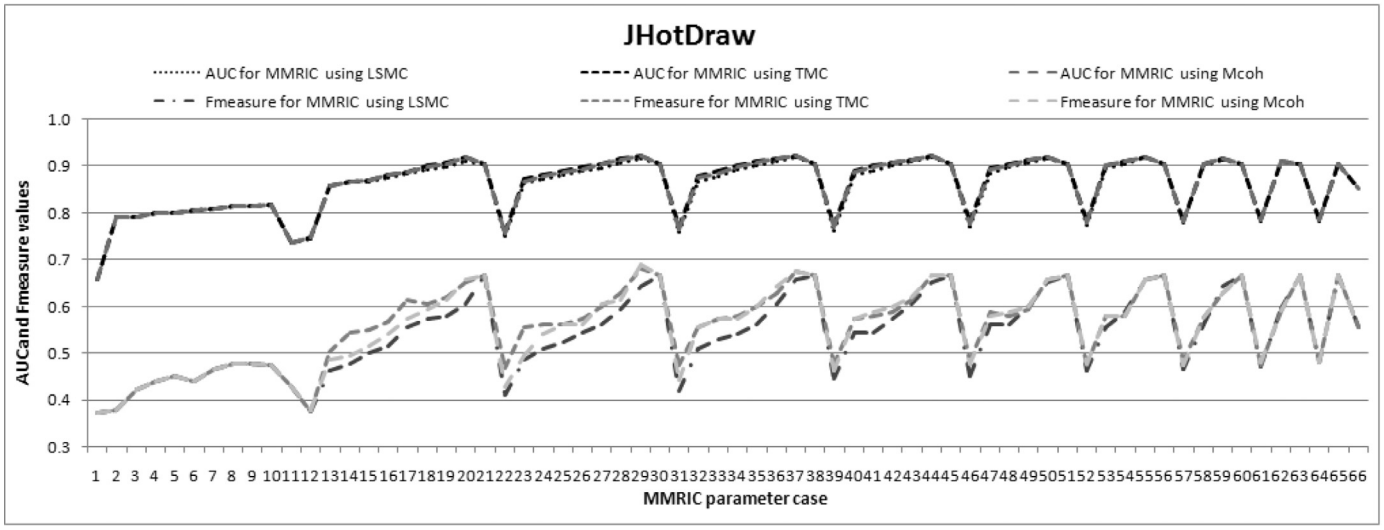
The results obtained for the constructed models and those shown in Fig. 1 lead to the following observations:

1. Most of the models (98.7%) were found to be significant predictors (p-value < 0.05) for classes that included methods in need of MMR (i.e., for most of the models, H_0 is rejected and H_1 is accepted). Only 18 out of the 1386 constructed models were found to be insignificant predictors for classes that included methods in need of MMR.
2. The values of AUC range within the following boundaries for the following systems: S1 [0.73, 0.95], S2 [0.66, 0.92], S3 [0.66, 0.96], S4 [0.77, 0.94], S5 [0.67, 0.92], S6 [0.73, 0.95], and S7 [0.7, 0.95]. The values of Fmeasure range within the following boundaries for the following systems: S1 [0.55, 0.84], S2 [0.37, 0.69], S3 [0.41, 0.84], S4 [0.42, 0.72], S5 [0.49, 0.78], S6 [0.38, 0.74], and S7 [0.42, 0.77]. This observation shows that varying the values of the parameters has a considerable impact on the predictive abilities of the models. Therefore, when applying Formula 6, it is important to carefully select the appropriate parameter values instead of applying any arbitrary values.
3. The prediction abilities of the models, in terms of AUC and Fmeasure, generally show an increasing trend as the values of α and β increase and the value of γ decreases. The only exception is when the value of γ drops from 0.1 to 0. In most cases, it is consistently found that the predictive abilities of the models with a value of 0.1 for γ are better than those of value 0. These observations indicate that when building models to predict classes that include methods in need of MMR, the highest weight must be given to the data related to method-attribute relations and method invocation relations. The data related to conceptual relations must be given relatively low weight but must not be ignored.
4. Setting one of the values of α, β, and γ to 1 and the rest to 0 causes the performance of the prediction model to degrade. This observation indicates that when building models to predict classes that include methods in need of MMR, considering combinations of method-attribute relations, method invocation relations, and conceptual relations is better than considering just one type of relation.
5. For most of the systems, the combinations of α, β, and γ that result in the best predictive abilities for the models are (0.4, 0.5, 0.1) and (0.5, 0.4, 0.1). In other words, the measures based on method-attribute relations and method invocation relations should be given almost equal weight, and the measure based on conceptual relations should be considered and given much lower weight. It was found that the models based on these two combinations consistently have outstanding classification performances (AUC > 0.9). The coefficients and details of the two models are provided in Appendix A.
6. The models based on different approaches to considering the method-attribute relations (i.e., the approaches implemented by the LSMC, TMC, and MCoh measures) sometimes have different predictive abilities. However, it was found that the first four observations are always applicable to all three considered approaches.
7. It was found that in most cases, when considering a specific parameter weight distribution, the classification perfor-

[3] The raw data are available at http://www.isc.ku.edu.kw/drjehad/PredictingMMR.htm.

[4] The contents of the confusion matrices, AUC, and Fmeasure values are available at http://www.isc.ku.edu.kw/drjehad/PredictingMMR.htm.





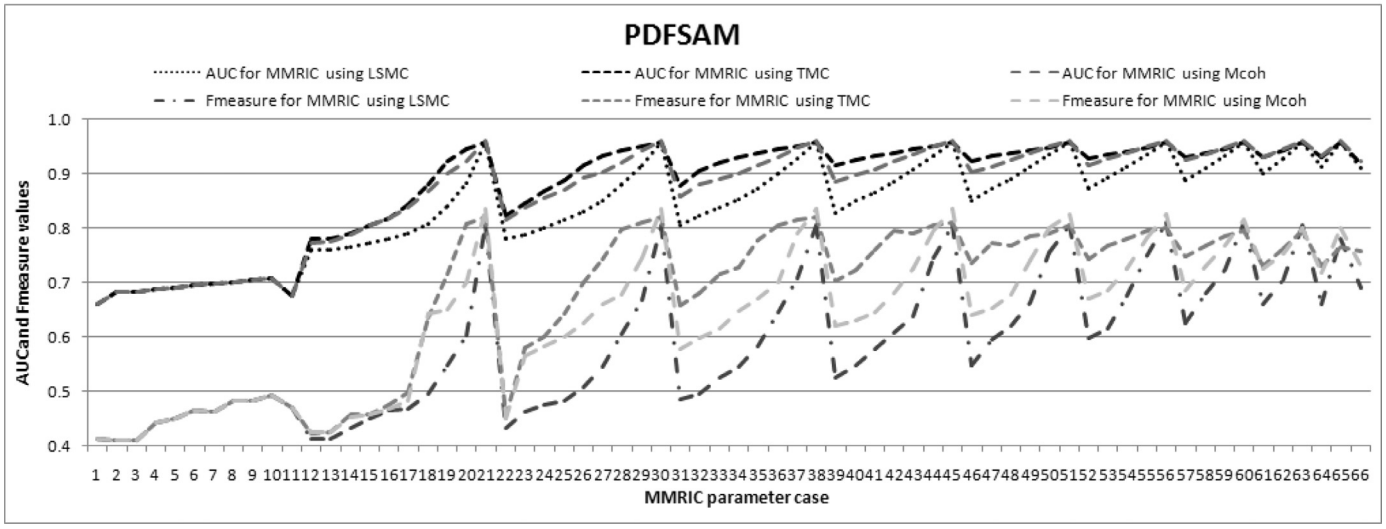
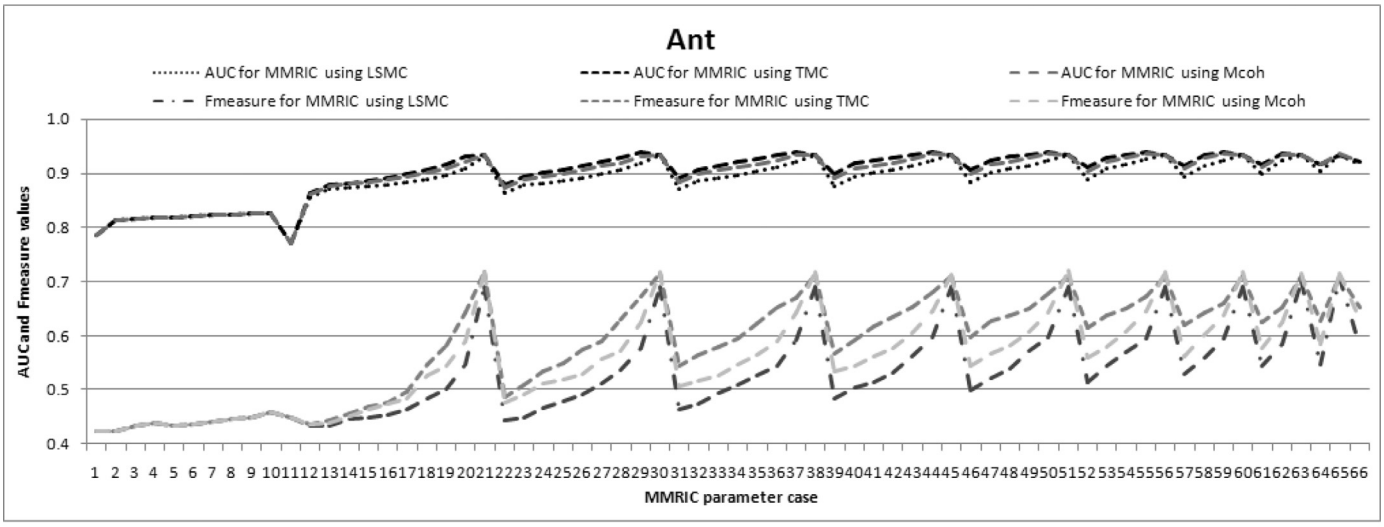
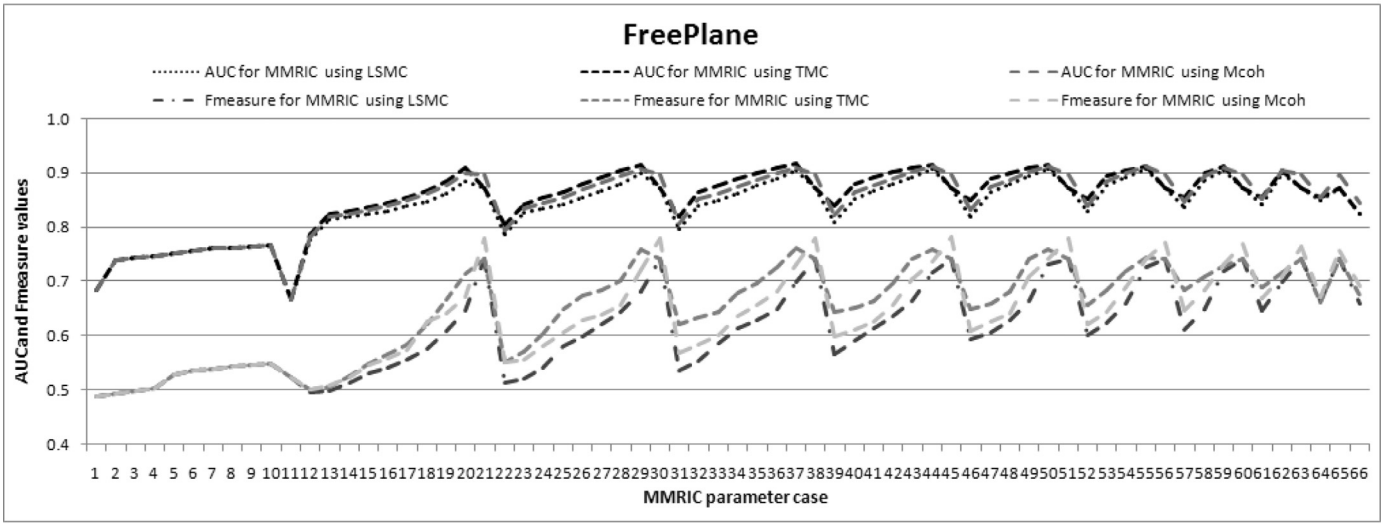


Fig. 1. AUC and Fmeasure values for the prediction models across different parameters.





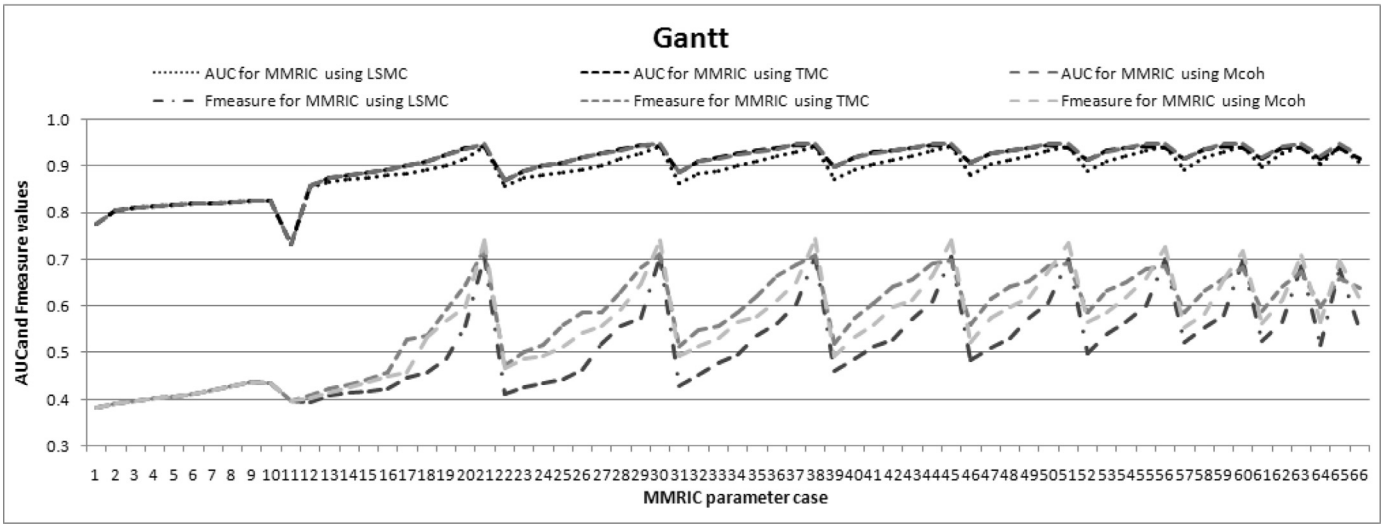


Fig. 1. Continued

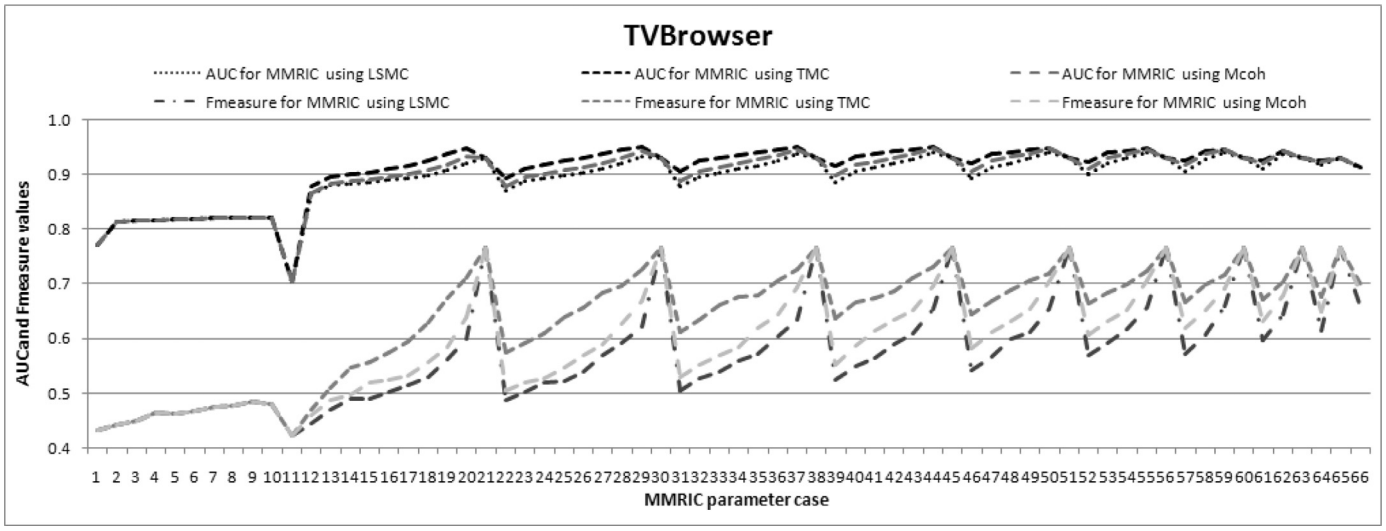


Fig. 1. Continued

mance results of the models that use the TMC measure are the best in terms of AUC and Fmeasure, whereas the models that use LSMC have the worst results. For each specific parameter weight distribution, it was found that in most cases, the difference between the AUC values across the three types of method-attribute relations measures is marginal, but the differences between the Fmeasure values are considerably different in many cases.

In conclusion, the empirical results support the recommendation to use combinations of measures that account for structural and conceptual relations when building prediction models for classes that include methods in need of MMR. The parameters in Formula 6 have to be carefully tuned to obtain the best prediction results because the distribution of the weights over the parameters has a great impact on the prediction results. The best combinations were empirically found to be (0.4, 0.5, 0.1) and (0.5, 0.4, 0.1). Both this study and the study reported in Section 5 concentrate on predicting the classes that include methods that need MMR. When predicting that a class includes a method that needs MMR, must we analyze each method to determine which method(s) need MMR? The answer is that the MMRIC value is the value of the maximum MMRI value among all methods, as formally indicated in Formula 6. Therefore, the method with the maximum MMRI is the one with the highest likelihood of needing MMR. To empirically investigate this claim, we performed the study reported in Section 7.

7. Performance of MMRI in indicating the methods in need of MMR

The third case study investigated the performance of MMRI in indicating methods that require MMR. The goal was to explore whether MMRI is a reliable indicator for methods in need of MMR.

7.1. Preparation

To answer RQ3, we empirically explored whether the method that has the maximum MMRI value in the class predicted to include a method that needs MMR is actually the method that needs MMR. In our case, we have to check whether this method is the method that we moved to the class in the mutation process. The goal of this study is to determine the performance of the proposed measure not only in predicting the class including the method that needs MMR but also in predicting the method that needs MMR within the class. Such prediction is expected to greatly conserve software engineering time because, in this case, a software engineer would not have to analyze the entire code in the class predicted to include a method that needs MMR and could instead limit his analysis to one method or certain methods within the class of interest.

To obtain the required data, we extended our tool to report the name of the last method that has the maximum MMRI for each class and for each computed MMRIC value.[5] For example, if the class has more than one method with an MMRI value of 1, the tool will report the name of the last method with an MMRI value of 1 written in the class source code. The reason for this implementation decision is that in the mutation process, we placed the moved method at the end of the code of the target class. It is important to note that the tool is implemented to support performing this empirical study. If the tool is intended to be used in practice, we must extend it to report the names of all methods with the same maximum value of MMRI. One or more of these methods are the ones predicted to need MMR. Another suggestion is to extend the tool to report the MMRI value of each method. In this case, the software engineer has to focus his analysis on those methods with the highest values of MMRI. Because of space limitations and to focus more on practical parameter weight distributions, we initially limited the analysis in this section to the two parameter combinations that mostly resulted in the best prediction performances: (0.4, 0.5, 0.1) and (0.5, 0.4, 0.1), as stated in Section 6. For each of the seven considered applications and for each of the three considered approaches to accounting for method-attribute relations, we limited our analysis to those classes correctly predicted to include methods in need of MMR. The reason for this constraint is that in practice, we are providing these classes to the software engineers to find the methods that need MMR and move them to their proper classes. Therefore, for each application and for each considered approach of accounting for method-attribute relations, we obtained the classes correctly predicted as needing MMR based on the models constructed in the analysis reported in Section 6. For each of these classes, we compared the name of the method moved during the mutation process with the one reported by the tool as being the one potentially needing MMR, and we calculated the percentage of methods correctly indicated as needing MMR.

[5] The raw data are available at http://www.isc.ku.edu.kw/drjehad/PredictingMMR.htm.

Table 5
Percentage of methods correctly predicted to need MMR.

System	LSMC			TMC			MCoh		
	(0.4, 0.5, 0.1)	(0.5, 0.4, 0.1)	(0.5, 0.5, 0)	(0.4, 0.5, 0.1)	(0.5, 0.4, 0.1)	(0.5, 0.5, 0)	(0.4, 0.5, 0.1)	(0.5, 0.4, 0.1)	(0.5, 0.5, 0)
S1	83.3%	83.3%	98.0%	84.3%	84.3%	98.0%	84.3%	84.3%	98.0%
S2	80.0%	80.0%	90.0%	80.0%	80.0%	90.0%	80.0%	80.0%	90.0%
S3	86.2%	86.2%	98.5%	86.2%	86.2%	98.5%	86.2%	86.2%	98.5%
S4	91.5%	91.1%	99.1%	92.3%	91.9%	99.1%	91.9%	91.9%	99.1%
S5	72.9%	72.9%	93.1%	72.9%	72.9%	93.1%	73.3%	73.3%	93.1%
S6	89.3%	89.3%	96.0%	89.3%	89.3%	96.0%	89.3%	89.3%	96.0%
S7	80.8%	80.8%	96.2%	82.7%	82.7%	96.2%	82.7%	82.7%	96.2%
Average	**83.4%**	**83.4%**	**95.8%**	**84.0%**	**83.9%**	**95.8%**	**84.0%**	**84.0%**	**95.8%**

7.2. Analysis and interpretation of the results

The resulting percentages of methods correctly indicated as needing MMR are reported in Table 5. For example, the table shows that when we obtain the MMRIC values based on LSMC measure for all of the classes in application S1 (i.e., OmegaT), set the parameters in Formula 6 to (0.4, 0.5, 0.1), and apply the corresponding prediction model as explained in the analysis reported in Section 6, the percentage of methods correctly indicated as needing MMR within the classes that were correctly predicted as including methods needing MMR is 83.3%. The last row in Table 5 shows the average percentages over all applications for each considered parameter combination.

The results reported in Table 5 for the two combinations (0.4, 0.5, 0.1) and (0.5, 0.4, 0.1) lead to the following observations:

1. For any specific application and selected approach, the pair results for the two combinations are mostly identical. This observation is consistent with that reported in Point 4 in Section 6, and it is attributable to the fact that the differences between the considered weights between the two combinations are marginal.
2. For any specific application and selected combination, the results across the three approaches are close to each other, with slightly higher percentages for some of the TMC and MCoh results compared to those of LSMC.
3. The results for the reported percentages considerably ranged between 72.9% (for S5) and 91.9% (for S4), with averages slightly ranging between 83.4% and 84%.

To investigate why the moved methods were not identified, we manually inspected the code for classes with methods in need of MMR that were not correctly identified. We found that the key reason for this problem is the consideration of the conceptual cohesion measure in the MMRIC formula (which was given a weight of 0.1 in the two selected combinations). The problem is detailed as follows. In most cases, we found that the class correctly predicted to include a method that needs MMR originally included a method p that does not have any structural cohesive relations to any of the methods in the class and that only has marginal, if any, conceptual relations with the class and its methods. The method moved to the class in the mutation process was incidentally found to have some wording similarity (in the comments or identifiers) (i.e., conceptual cohesive relations) with the target class, and the degree of conceptual cohesive relations was found to be higher than that of method p. Therefore, because each of the selected combinations (0.4, 0.5, 0.1) and (0.5, 0.4, 0.1) gives a weight of 10% to the conceptual cohesive relations, the method p was found to have a higher MMRI value than that of the moved method. In this case, the method p is incorrectly predicted as the one in need of MMR instead of the method moved in the mutation process. To empirically show that this stated reason is valid, we considered the parameter combination (0.5, 0.5, 0), which does not give any weight to the conceptual cohesive relations. The corresponding results are given in the fourth, seventh, and tenth columns in Table 5. These results show that the percentages of correctly identified methods in need of MMR were improved by more than 10%, comparing to the corresponding results reported for the other two parameter combinations. The results corresponding to the parameter combination (0.5, 0.5, 0) ranged between 90% and 99.1% with an average of 95.8%, which is considerably better than the results obtained for the two other parameter combinations. The coefficients and details of the model are provided in Appendix A.

Based on these results, we recommend using one of the two combinations, (0.4, 0.5, 0.1) or (0.5, 0.4, 0.1), when building the model to predict the classes including methods in need of MMR. Once the model is applied and the classes predicted to include methods in need of MMR are identified, Formula 6 with the parameter settings (0.5, 0.5, 0) must be applied to each method in each of the predicted classes. For each class, the software engineer must pay more attention to those methods with the highest MMRI values because those methods are the potential MMR candidates.

8. Threats to validity

8.1. Internal validity

Analyzing the coupling relations between the method of interest in a class and the code in other classes and determining how other classes are using the method of interest are key factors to be considered when determining whether the method of interest needs MMR. However, in this research we aimed to propose a measure and a corresponding prediction model that can predict whether a class includes methods of the need MMR and determine these methods by analyzing the code locally within the class only. The results show that the proposed models based only on localized measures have outstanding abilities to predict the classes that include methods in need of MMR. In addition, the proposed localized measures are shown to have high precision in determining the methods in need of MMR within the predicted classes.

The analysis based on mutated code is based on an assumption that the original code is of good quality and designed in such a manner that the methods are potentially placed in their proper classes. To support this assumption, we selected JHotDraw, an application widely known for its good quality, to be one of the analyzed systems. We found that the results based on the other selected systems have similar and sometimes better results than those of JHotDraw, which gave us confidence that the selected systems are also of good quality. Therefore, it is strongly expected that the classes in the selected systems are well designed, and accordingly, the methods are highly expected to be placed in their proper classes. An alternative to performing the analysis based on mutated code is to rely on external developers searching for methods in need of MMR in the selected systems and to use this knowledge in constructing the prediction models. However, we expect this alternative solution to be less reliable than the mutation approach because this alternative requires a detailed and deep understand-

ing of the analyzed code and is sometimes based on subjective decisions.

One key internal validity threat is the selection of the measures adapted to form the proposed MMRIC measure. Other cohesion measures would also be considered. However, we selected the measures that apply different approaches of accounting for structural cohesive relations and considered a widely applied conceptual-based cohesion measure. The goal of this research is to build a model to predict classes that include methods in need of MMR based on localized data. We found that the proposed models based on the selected measures using localized data have outstanding abilities to predict classes that include methods in need of MMR. Therefore, by using the selected measures, we achieved our main goal.

Finally, a prediction model based on data of a system and applied on classes of other systems may not perform as well as when it is applied to the classes of the system originally used to build the prediction model. To increase confidence that the prediction models perform well when applied to classes other than those used in building the model, we applied *k*-fold cross-validation, which is a widely acceptable and applied validation technique.

8.2. External validity

Although all of the selected systems are implemented in Java, the choice of which object-oriented system to select does not have any impact on the measures considered in this research. Restricting the selection of systems to open-source code may have an impact on the quality of the systems. However, studies (e.g., [46,47,51]) on the differences in reliability and design quality between industrial systems and open-source systems have not provided clear and reliable conclusions. The selected systems might not be representative in terms of their sizes and numbers. However, most of the results were found to be consistent over the selected systems, giving us confidence that the obtained results are adequately reliable. In addition, the number of selected systems is comparable to similar studies (e.g., [11,12,52]). To generalize the results obtained and conclusions drawn, more systems of different sizes, programming languages, and application domains, including industrial and real-life systems, should be involved in large-scale evaluations.

9. Conclusions and future work

This paper proposes a measure that indicates whether a class includes a method that needs MMR. The measure is based on three types of cohesive relations: method-attribute relations, method invocation relations, and conceptual relations. In addition, the measure accounts for a set of preconditions related to MMR. We adapted three existing method-attribute cohesion-based measures and incorporated them into the proposed measure. All of the measures and preconditions can be determined locally within the class of interest and do not require analysis of any code outside the class. We reported three empirical studies involving seven open-source systems. In the first study, we constructed prediction models based on the proposed measure when individually accounting for each type of cohesive relations and compared the results. The main conclusion was that the prediction models based on the measures that account for method-attribute relations are considerably better, in terms of classification performance, than those that account for method invocations or conceptual relations. The second study empirically explored the impact of the weight distribution of the parameters included in the proposed measure on the ability of the resulting model to predict whether a class includes a method that needs MMR. We found that the best model is that which gives a low weight to the part of the measure that accounts for conceptual relations and distributes the rest of the weight almost equally between the parts that account for method-attribute relations and method invocation relations. This model is empirically found to have outstanding classification performance. The third study investigated the proposed measure's ability to indicate the methods in need of MMR within the classes predicted to include such methods. We found that the measure that gives equal weight to the measures accounting for method-attribute relations and method invocation relations and ignores conceptual relations was able to correctly detect 90% or more of the methods in need of MMR.

In practice, we suggest two scenarios that use the proposed measure and obtained results. The first practical scenario is applicable during the software development phase and can be applied even during early design stages when a class is under development and its internal relations are defined, with no need to wait for the rest of classes to be developed. In this case, we suggest using the proposed measure with parameters set to (0.4, 0.5, 0.1) or (0.5, 0.4, 0.1) and applying the corresponding prediction model to obtain the probability that the class includes methods in need of MMR. The developer must pay more attention to the design of the class and the placement of the methods and (probably) revise them if the obtained probability value is found to be above a certain threshold. The determination of the threshold is based on the available time and resources during the development stages. Typically, a high threshold is selected when limited time and resources are available and vice versa. Based on the obtained probability values and selected threshold, if the class is predicted to include method(s) in need of MMR, the developer is directed to obtain the MMRI value for each method with parameters set to (0.4, 0.5, 0). Methods with the highest obtained MMRI values must be carefully inspected because they are the MMR candidates. In this scenario, the advantage of the proposed approach is that the proposed approach is applicable in early development stages when the entire system is not yet complete, whereas others are not. This is because none of the proposed measures require knowledge of data or relations to be extracted from outside the class of interest.

The second scenario is applicable during the maintenance phase when the software engineers are working to improve code quality by applying refactoring activities. In this case, we suggest applying the proposed measure with parameters set to (0.4, 0.5, 0.1) or (0.5, 0.4, 0.1) to all classes in the system of interest and applying the corresponding prediction model to obtain the probability of each class including methods that need MMR. Based on the available time and resources, a threshold must be set and applied to determine which classes are predicted to include methods in need of MMR. Each of these predicted classes must be addressed individually, as stated in the first scenario, beginning with obtaining the MMRI values of the methods. In this scenario, the advantage of the proposed approach is that the proposed approach is scalable and can be applied to systems with long chains of coupling relations. Unlike other approaches, the proposed approach does not require analyzing and tracing such coupling relations.

In the future, we intend to propose measures to indicate the need to apply other refactoring activities. The selection of the existing measures to be used or adapted in these cases is highly dependent on the type of refactoring activity considered and its relationship with quality attributes. In addition, replicated studies are required to prove or disprove the results obtained and conclusions drawn and investigate the impact of refactoring on different internal and external quality attributes.

Acknowledgments

The author would like to acknowledge the support of this work by Kuwait University Research Grant QI01/15. In addition, the author would like to thank Anas Abdin for assisting in collecting the quality results and performing the mutation process.

Appendix A

Table A.1.

Table A.1
Prediction models based on the selected applications.

MMRIC Case	C0	C1	*p*-value	TN	FP	FN	TP	Precision	Recall
S1: OmegaT									
LSMC (1,0,0)	−18.1	18.5	<0.0001	234	73	4	98	0.573	0.961
TMC(1,0,0)	−10.8	11.3	<0.0001	248	59	3	99	0.627	0.971
MCoh (1,0,0)	−13.0	13.5	< 0.0001	241	66	3	99	0.600	0.971
LSMC (0,1,0)	−241.7	241.2	0.992	142	165	0	102	0.382	1.000
LSMC (0,0,1)	−5.2	4.9	<0.0001	171	136	11	91	0.401	0.892
TMC(0.4,0.5,0.1)	−43.2	44.5	<0.0001	267	40	4	98	0.710	0.961
TMC (0.5,0.4,0.1)	−38.3	39.5	<0.0001	261	46	4	98	0.681	0.961
TMC (0.5,0.5,0)	−510.2	511.1	< 0.0001	265	42	0	102	0.708	1.000
S2: JHotDraw									
LSMC (1,0,0)	−373.1	372.6	<0.0001	93	48	0	30	0.385	1.000
TMC(1,0,0)	−218.2	217.7	0.992	93	48	0	30	0.385	1.000
MCoh (1,0,0)	−219.0	218.6	0.992	93	48	0	30	0.385	1.000
LSMC (0,1,0)	−124.7	123.7	0.993	61	80	0	30	0.273	1.000
LSMC (0,0,1)	−4.2	3.1	0.016	53	88	3	27	0.235	0.900
TMC(0.4,0.5,0.1)	−58.6	59.0	0.001	113	28	1	29	0.509	0.967
TMC (0.5,0.4,0.1)	−60.0	60.4	0.001	112	29	1	29	0.500	0.967
TMC (0.5,0.5,0)	−831.3	831.3	<0.0001	111	30	0	30	0.500	1.000
S3: PDFSAM									
LSMC (1,0,0)	−19.2	19.5	<0.0001	179	51	4	61	0.545	0.938
TMC(1,0,0)	−7.0	7.5	<0.0001	190	40	1	64	0.615	0.985
MCoh (1,0,0)	−15.1	15.6	<0.0001	189	41	4	61	0.598	0.938
LSMC (0,1,0)	−218.5	217.7	0.992	83	147	0	65	0.307	1.000
LSMC (0,0,1)	−3.2	2.5	<0.0001	106	124	16	49	0.283	0.754
TMC(0.4,0.5,0.1)	−20.7	21.9	<0.0001	200	30	1	64	0.681	0.985
TMC (0.5,0.4,0.1)	−16.9	18.0	<0.0001	197	33	1	64	0.660	0.985
TMC (0.5,0.5,0)	−18.9	19.8	<0.0001	200	30	1	64	0.681	0.985
S4: Ant									
LSMC (1,0,0)	−16.9	16.7	<0.0001	935	316	10	225	0.416	0.957
TMC(1,0,0)	−11.4	11.5	<0.0001	1018	233	9	226	0.492	0.962
MCoh (1,0,0)	−15.6	15.6	<0.0001	996	255	10	225	0.469	0.957
LSMC (0,1,0)	−532.3	531.4	<0.0001	676	575	0	235	0.290	1.000
LSMC (0,0,1)	−4.7	3.9	<0.0001	697	554	23	212	0.277	0.902
TMC(0.4,0.5,0.1)	−29.5	30.0	<0.0001	1043	208	7	228	0.523	0.970
TMC (0.5,0.4,0.1)	−26.0	26.4	<0.0001	1039	212	6	229	0.519	0.974
TMC (0.5,0.5,0)	−96.3	96.6	<0.0001	1061	190	1	234	0.552	0.996
S5: FreePlane									
LSMC (1,0,0)	−21.3	21.3	<0.0001	500	260	6	256	0.496	0.977
TMC(1,0,0)	−11.9	11.9	<0.0001	513	247	4	258	0.511	0.985
MCoh (1,0,0)	−16.2	16.4	<0.0001	537	223	6	256	0.534	0.977
LSMC (0,1,0)	−478.7	478.1	<0.0001	282	478	0	262	0.354	1.000
LSMC (0,0,1)	−3.3	2.8	<0.0001	348	412	44	218	0.346	0.832
TMC(0.4,0.5,0.1)	−29.1	30.0	<0.0001	607	153	8	254	0.624	0.969
TMC (0.5,0.4,0.1)	−27.9	28.8	<0.0001	609	151	9	253	0.626	0.966
TMC (0.5,0.5,0)	−922.7	923.1	<0.0001	578	182	0	262	0.590	1.000
S6: Gantt									
LSMC (1,0,0)	−17.5	17.3	<0.0001	333	110	5	70	0.389	0.933
TMC(1,0,0)	−10.0	10.0	<0.0001	365	78	3	72	0.480	0.960
MCoh (1,0,0)	−14.9	15.0	<0.0001	361	82	5	70	0.461	0.933
LSMC (0,1,0)	−230.3	229.2	0.993	216	227	0	75	0.248	1.000
LSMC (0,0,1)	−5.0	4.0	<0.0001	246	197	11	64	0.245	0.853
TMC(0.4,0.5,0.1)	−25.5	26.0	<0.0001	378	65	1	74	0.532	0.987
TMC (0.5,0.4,0.1)	−21.4	21.8	<0.0001	376	67	1	74	0.525	0.987
TMC (0.5,0.5,0)	−34.0	34.3	<0.0001	378	65	1	74	0.532	0.987
S7: TVBrowser									
LSMC (1,0,0)	−36.3	36.4	<0.0001	415	103	3	101	0.495	0.971
TMC(1,0,0)	−16.3	16.5	<0.0001	432	86	2	102	0.543	0.981
MCoh (1,0,0)	−26.5	26.7	<0.0001	428	90	3	101	0.529	0.971
LSMC (0,1,0)	−319.3	318.3	<0.0001	235	283	0	104	0.269	1.000
LSMC (0,0,1)	−6.5	5.7	<0.0001	274	244	8	96	0.282	0.923
TMC(0.4,0.5,0.1)	−42.0	42.8	<0.0001	445	73	2	102	0.583	0.981
TMC (0.5,0.4,0.1)	−−37.7	38.4	<0.0001	442	76	3	101	0.571	0.971
TMC (0.5,0.5,0)	−1397.5	1398.0	<0.0001	455	63	0	104	0.623	1.000

References

[1] J. Al Dallal, Mathematical validation of object-oriented class cohesion metrics, Int. J. Comput. 4 (2) (2010) 45–52.
[2] J. Al Dallal, Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics, Inf. Softw. Technol. 54 (10) (2012) 1125–1141.
[3] J. Al Dallal, Identifying refactoring opportunities in object-oriented code: a systematic literature review, Inf. Softw. Technol. 58 (2015) 231–249.
[4] J. Al Dallal, A. Abdin, Empirical evaluation of the impact of object-oriented code refactoring on quality attributes: a systematic literature review, IEEE Trans. Softw. Eng. (2017), doi:10.1109/TSE.2017.2658573.
[5] J. Al Dallal, L. Briand, An object-oriented high-level design-based class cohesion metric, Inf. Softw. Technol. 52 (12) (2010) 1346–1361.
[6] J. Al Dallal and L. Briand, A Precise method-method interaction-based cohesion metric for object-oriented classes, ACM Trans. Softw. Eng. Methodol., 21(2) (2012) 8:1–8:34.
[7] A. Alkhalid, M. Alshayeb, S.A. Mahmoud, Software refactoring at the class level using clustering techniques, J. Res. Pract. Inf. Technol. 43 (4) (2011) 285–306.
[8] Ant, https://github.com/apache/ant, accessed January 2017.
[9] A. Bacchelli, M. Lanza, R. Robbes, Linking e-mails and source code artifacts, in: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, 1, Cape Town, South Africa, 2010, pp. 375–384.
[10] G. Bavota, B. De Carluccio, A. De Lucia, M.D. Penta, R. Oliveto, O. Strollo, When does a refactoring induce bugs? An empirical study, in: IEEE 12th International Working Conference on In Source Code Analysis and Manipulation (SCAM), 2012, pp. 104–113.
[11] G. Bavota, M. Gethers, R. Oliveto, D. Poshyvanyk, A.D. Lucia, Improving software modularization via automated analysis of latent topics and dependencies, Trans. Softw. Eng. Methodol. 23 (1) (2014) article 4.
[12] G. Bavota, R. Oliveto, M. Gethers, D. Poshyvanyk, A. De Lucia, Methodbook: recommending move method refactorings via relational topic models, IEEE Trans. Softw. Eng. 40 (7) (2014) 671–694.
[13] 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.
[14] 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.
[15] L. Briand, J. Daly, J. Wuest, A unified framework for cohesion measurement in object-oriented systems, Empir. Softw. Eng. 3 (1) (1998) 65–117.
[16] L.C. Briand, S. Morasca, V.R. Basili, Defining and validating measures for object-based high-level design, IEEE Trans. Softw. Eng. 25 (5) (1999) 722–743.
[17] L.C. Briand, J. Wüst, H. Lounis, Replicated case studies for investigating quality factors in object-oriented designs, Empir. Softw. Eng. 6 (1) (2001) 11–58.
[18] I.G. Czibula, G. Czibula, Hierarchical clustering based automatic refactorings detection, WSEAS Trans. Electr. 5 (7) (2008) 291–302.
[19] I.G. Czibula, G. Serban, Improving systems design using a clustering approach, Int. J. Comput. Sci. Netw. Secur. 6 (12) (2006) 40–49.
[20] S.T. Dumais, Latent semantic analysis, Ann. Rev. Inf. Sci. Technol. 38 (2005) 188–230.
[21] Eclipse, http://www.eclipse.org/, accessed January 2017.
[22] M. Fokaefs, N. Tsantalis, E. Stroulia, A. Chatzigeorgiou, J. Deodorant, Identification and removal of feature envy bad smells, in: Proceedings of IEEE International Conference on Software Maintenance, 2007, pp. 467–468.
[23] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1999.
[24] FreePlane, https://sourceforge.net/projects/freeplane/, accessed January 2017.
[25] Gantt, https://sourceforge.net/projects/ganttproject/, accessed January 2017.
[26] Y. Higo, S. Kusumoto, K. Inoue, A metric-based approach to identifying refactoring opportunities for merging code clones in a Java software system, J. Softw. Maint. Evol. 20 (6) (2008) 435–461.
[27] D. Hosmer, S. Lemeshow, Applied Logistic Regression, second ed., Wiley Interscience, 2000.
[28] JHotDraw, http://sourceforge.net/projects/jhotdraw/, accessed January 2017.
[29] JRefactory, http://jrefactory.sourceforge.net/, accessed January 2017.
[30] M. Kim, D. Cai, S. Kim, An empirical investigation into the role of refactorings during software evolution, in: ICSE' 11: Proceedings of the 2011 ACM and IEEE 33rd International Conference on Software Engineering, 2011, pp. 151–160.
[31] S. Lee, G. Bae, H.S. Chae, D. Bae, Y.R. Kwon, Automated scheduling for clone-based refactoring using a competent GA, Software 41 (5) (2011) 521–550.
[32] A. Marcus, D. Poshyvanyk, R. Ferenc, Using the conceptual cohesion of classes for fault prediction in object-oriented systems, IEEE Trans. Softw. Eng. 34 (2) (2008) 287–300.
[33] T. Mens, T. Tourwé, A survey of software refactoring, IEEE Trans. Softw. Eng. 30 (2) (2004) 126–139.
[34] A. Murgia, R. Tonelli, M. Marchesi, G. Concas, S. Counsell, J. McFall, S. Swift, Refactoring and its relationship with fan-in and fan-out: an empirical study, 16th European Conference on Software Maintenance and Reengineering (CSMR), 2012.
[35] M. O'Cinneide, D. Boyle, I. Moghadam, Automated refactoring for testability, The 4th Workshop on Refactoring Tools (WRT), 2011.
[36] M. Ó Cinnéide, L. Tratt, M. Harman, S. Counsell, I.H. Moghadam, Experimental assessment of software metrics using automated refactoring, in: Proc. Empirical Software Engineering and Management (ESEM), 2012, pp. 49–58.
[37] R. Oliveto, M. Gethers, G. Bavota, D. Poshyvanyk, A. De Lucia, Identifying method friendships to remove the feature envy bad smell (NIER track), in: Proceedings of the 33rd International Conference on Software Engineering, 2011, pp. 820–823.
[38] D. Olson, D. Delen, Advanced Data Mining Techniques, first ed., Springer, 2008.
[39] OmegaT, http://sourceforge.net/projects/omegat/, accessed January 2017.
[40] W. Pan, B. Li, Y. Ma, J. Liu, Y. Qin, Class structure refactoring of object-oriented softwares using community detection in dependency networks, Front. Comput. Sci. China 3 (3) (2009) 396–404.
[41] W. Pan, J. Wang, M. Wang, Identifying the move method refactoring opportunities based on evolutionary algorithm, Int. J. Model. Ident. Control 18 (2) (2013) 182–189.
[42] PDFSAM, http://sourceforge.net/projects/pdfsam/, accessed January 2017.
[43] 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.
[44] P. Rousseeuw, I. Ruts, J. Tukey, The bagplot: a bivariate boxplot, Am. Stat. 53 (4) (1999) 382–387.
[45] V. Sales, R. Terra, L.F. Miranda, M.T. Valente, Recommending move method refactorings using dependency sets, in: IEEE 20th Working Conference on Reverse Engineering (WCRE), 2013, pp. 232–241.
[46] 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, 9th Panhellenic Conference in Informatics, 2003.
[47] I. Samoladas, G. Gousios, D. Spinellis, I. Stamelos, The SQO-OSS quality model: measurement based open-source software evaluation, in: Open Source Development, Communities and Quality, vol. 275, 2008, pp. 237–248.
[48] O. Seng, J. Stammel, D. Burkhart, Search-based determination of refactorings for improving the class structure of object-oriented systems, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, 2006, pp. 1909–1916.
[49] G. Serban, I.G. Czibula, Restructuring software systems using clustering, in: 22nd International Symposium on Computer and Information Sciences, 2007, pp. 1–6.
[50] R. Shatnawi, W. Li, J. Swain, T. Newman, Finding software metrics threshold values using ROC curves, J. Softw. Maint. Evol. 22 (1) (2010) 1–16.
[51] D. Spinellis, G. Gousios, V. Karakoidas, P. Louridas, P.J. Adams, I. Samoladas, I. Stamelos, Evaluating the quality of open source software, SQM 2008: Second International Workshop on Software Quality and Maintainability—12th European Conference on Software Maintenance and Reengineering (CSMR 2008) Satellite Event vol. 233 (2009) 5–28.
[52] N. Tsantalis, A. Chatzigeorgiou, Identification of move method refactoring opportunities, IEEE Trans. Softw. Eng. 35 (3) (2009) 347–367.
[53] TVbrowser, January 2017. accessed accessed https://sourceforge.net/projects/tvbrowser/.
[54] L. Zhao, J. Hayes, Predicting classes in need of refactoring: an application of static metrics, Workshop on Predictive Models of Software Engineering (PROMISE), 2006 associated with ICSM.

Jehad Al Dallal is a professor and chairman of the Information Science Department at Kuwait University. He received his PhD in computer science from the University of Alberta in Canada and was granted the award for best PhD researcher. Prof. Al Dallal has completed many research projects in the areas of software testing, software metrics, software refactoring, and communication protocols. In addition, he has published more than 90 papers in conference proceedings and in journals published by *ACM, IEEE, IET, Elsevier, Wiley*, and others. Prof. Al Dallal was involved in developing more than 20 software systems. He also served as a technical committee member of several international conferences and as an associate editor for several refereed journals.