New Conceptual Cohesion Metrics: Assessment for Software Defect Prediction

Diana-Lucia Miholca
Faculty of Mathematics and Computer Science
Babeş-Bolyai University
400084, Cluj-Napoca, Romania
Email: diana.miholca@ubbcluj.ro

***Abstract*—High cohesion and low coupling is a desirable software design principle as it significantly and positively impacts software quality. While the majority of existing cohesion metrics are structural, this paper proposes a set of new object-oriented metrics capturing the conceptual cohesion of classes. They are derived from the semantic information contained in the source code and involve using Doc2Vec, an artificial neural network based unsupervised model. Two case studies that compare the proposed metrics with an extensive set of relevant existing ones are presented. The empirical validation has been performed on three open source software systems. It includes the assessment of the proposed metrics versus preexisting ones from the perspective of their utility for software defect prediction. The experimental results confirm that the metrics we propose capture additional aspects of cohesion when compared to existing cohesion metrics, while also providing better software defect prediction performance than preexisting related conceptual cohesion metrics.**

Keywords*—*software cohesion; semantic cohesion; conceptual cohesion; software defect prediction; machine learning; Doc2Vec;

I. Introduction

Software Defects Prediction (SDP) consists in identifying defective software components. It assists measuring project evolution, supports process management, predicts software reliability, streamlines testing and guides code review. All these allow to significantly reduce the costs involved in developing and maintaining software products [1], [2]. Despite its importance and extensive applicability, SDP remains a difficult problem, especially in large-scale complex systems, and thus a very active research area [3].

Software defects are consequences of poor software quality, while software coupling and software cohesion are properties that strongly relate to the quality of the software design. They are both code quality metrics [4]. High cohesion in conjunction with low coupling is a desirable design principle [5]. Design flaws make the software more defects-prone [6]. Consequently, there is a connection between software coupling and cohesion, which may indicate design flaws, and software defects [7]. More generally, coupling and cohesion have significant impact on software quality [8], [9], by affecting multiple software development and maintenance activities including SDP, impact analysis, program comprehension and software re-usability [2].

While coupling expresses the degree of dependence between software entities, cohesion is defined as the extent of relatedness among a software entity's components. Measuring cohesion is difficult, mainly because such a definition, even if very intuitive, is equally ambiguous, thus leaving room for interpretation. Consequently, there is no one cohesion metric or cohesion metrics suite that is accepted as a standard. Instead, different metrics correspond to different interpretations [5], [2].

Currently, there is plethora of interest in object-oriented (OO) software metrics [4]. In the rising OO paradigm [2], cohesion is usually measured at class level [5]. The cohesion of a class indicates the degree to which the class provides a sole functionality to the software as a whole [9] and is computed based on the magnitude of relatedness among its members, that is, its methods and attributes [8], [2]. So, cohesion is decreased if the methods in a class are insufficiently related. Low cohesion complicates the understanding, development and maintenance of a software [10] and thus, as a side effect, may generate software defects [1], [9], [11].

As well as software coupling, software cohesion is generally computed based on structural information extracted from the source code [7], such as attribute references in methods [12], [8]. *Structural* cohesion metrics include: Lack of Cohesion Of Methods (LCOM1) [13], LCOM2 [13], LCOM3 [14], LCOM4 [14], LCOM5 [15], Information-Flow-based Cohesion (ICH) [16], Similarity-based Class Cohesion (SCC) [10], Low-Level Design Class Cohesion (LSCC) [8], Yet Another Lack of Cohesion in Methods (YALCOM) [4], etc. These cohesion metrics express the degree to which the members of a class belong together structurally, but do not inform whether the class is cohesive conceptually [5].

The most desirable form of cohesion, however, is *model* cohesion [17], also called *semantic* cohesion or *conceptual* cohesion, which implies that a class represents an unique and semantically meaningful concept. Despite this, there are significantly fewer conceptual cohesion metrics in the literature. Of these, to the best of our knowledge, specific to the OO paradigm and based on semantic information extracted from the source code are only the following: the Logical Relatedness of Methods (LORM) [18], the Lack of Conceptual Cohesion in Methods (LCSM) [5], Conceptual Cohesion of Classes (C3) [12], Conceptual Lack of Cohesion on Methods (CLCOM5) [19] and Maximal Weighted Entropy (MWE). More about these metrics will be presented in Section II. They are computed using knowledge-based systems, statistical and count-based information retrieval (IR) methods. In particular, the underlying IR technique used for computing LCSM, C3 and CLCOM5 is Latent Semantic Indexing (LSI) [20], MWE utilizes Latent Dirichlet Allocation (LDA), while LORM is calculated with the help of a knowledge-based system.

Motivated by the above-described high yet still little investigated potential of semantic cohesion in assessing software defect proneness, the prime aim of this paper is to propose new Machine Learning (ML) based conceptual cohesion metrics for supporting SDP, thus complementing the conceptual coupling based metrics we have introduced in previous studies [21], [22]. The underlying information retrieval technique we use for this purpose is Doc2Vec [23], which is shown in the literature to better capture the semantics than statistical, count-based information retrieval methods. This is an element of novelty, given that no software cohesion metrics based on Doc2Vec have been proposed yet. The proposed metrics were validated theoretically and empirically. For the empirical evaluation, we considered three open source software systems and performed multiple analyzes, including from supervised SDP perspective.

The rest of the paper is structured as follows. Section II reviews existing related work. The new cohesion metrics are defined and theoretically validated in Section III. Their empirical assessment is detailed in Section IV. The results are presented and discussed in Section V. Finally, Section VI draws the conclusions and indicates directions for future work.

II. Related work

In the last two decades, a notable amount of research studies focused on the reliance of coupling and cohesion for predicting various software quality parameters [2]. If until relatively recently the studies concentrated on evaluating coupling and cohesion metrics included in traditional suites (such as the Chidamber and Kemerer (C&K) [13] metrics suite), the latest research concerns aim at updating, extending and complementing them, by proposing new relevant coupling and cohesion measures [21], [22], [24], [25] thus confirming the subject of this paper as belonging to an emerging research field. A recent study involving participants from the software development community, conducted by Sharma and Spinellis [4], has confirmed that the increasing scientific interest in proposing new cohesion metrics comes in response to a practical need in this regard.

Tiwari and Rathore [2] have performed a systematic mapping study on coupling and cohesion metrics for OO software. From the 137 relevant papers selected, 17% proposed coupling metrics, 8% proposed cohesion metrics, while 24% proposed both coupling and cohesion metrics. The remaining studies (51%) are focused on evaluating existing metrics suites. The authors signaled that, due to lack of empirical validation, more papers are needed to assess the practical relevance of the coupling and cohesion metrics. They also deduced that coupling and cohesion are most extensively evaluated from the perspective of their relevance for SDP.

Subsequently, Rathore and Kumar [1] have conducted a review on existing SDP approaches, with emphasis on the metrics, data quality issues, techniques and performance evaluation metrics. The survey revealed that most of the studies (39%) validated OO metrics. The authors motivated the heavy use of OO metrics for SDP through the inability of traditional metrics to capture OO features including coupling and cohesion, that underlie modern software development practices. They also concluded that more research works proposing and validating new metrics suites are needed.

In the following, given their relatedness with the metrics proposed in the current paper, we mainly review the existing OO conceptual cohesion metrics. When possible, we discuss them in the context of SDP.

A first semantically-derived cohesion metric, LORM, has been introduced by Etzkom and Delugach [18]. LORM measures the conceptual similarity of the methods in a class, as determined by representing the methods by a semantic network of conceptual graphs. The proposed metric incorporate OO features such as polymorphism and inheritance. To calculate this metric, a semantic analysis by a knowledge based expert system (called the Program Analysis Tool for Reuse (PATRicia) system) is applied to each class.

Two complementary semantic class cohesion measures named Conceptual Cohesion of Classes (C3) and Lack of Conceptual Similarity between Methods (LCSM) have been introduced by Marcus and Poshyvanyk [5]. LCSM is an inverse measure of cohesion. The semantic information based on which C3 and LCSM are computed is extracted from the source code using LSI. An experimental case study on WinMerge 2.0.2 has been performed in order to test the correlation of the newly proposed metrics with the traditional structural ones. The results confirmed that C3 may capture aspects of cohesion similar to those captured by existing metrics (ICH, LCOM5), but they refuted any significant correlation between LCSM and the existing structural cohesion metrics.

In a subsequent study, Marcus et al. [12] have investigated C3 versus existing cohesion metrics from the perspective of their utility for predicting software defects. As experimental case study, they considered Mozilla software (versions 1.6 and 1.7), which is mostly implemented in C++. The experimental results have revealed that combining C3 with existing structural cohesion metrics improves the predictive performance.

The LCSM metric introduced by Marcus and Poshyvanyk [5] has been considered as the semantic cohesion which, along with a structural cohesion metric, form the core of a cohesion-based refactoring technique proposed by De Lucia et al. [26].

Újházi et al. [19] have proposed Conceptual Lack of Cohesion on Methods (CLCOM5), a conceptual cohesion metric based on the structural metric LCOM5. Together with a conceptual coupling metric, Conceptual Coupling between Object classes (CCBO), CLCOM5 has been empirically validated in the context of SDP, by considering, once again, Mozilla as a case study. The conclusions of the study are that the proposed metrics produce a similar accuracy when compared to existing structural metrics, while combining them with preexisting ones leads to statistically significant improvements.

There are multiple other studies that have evaluated cohesion in the context of SDP, possibly along with coupling, but more frequently as structural metrics. For instance, Kharta et al. [7] have proposed using coupling in conjunction with cohesion for SDP. The coupling and cohesion metrics considered are: Coupling Between Objects (CBO) [27], Afferent Coupling (CA), Cohesion Among Methods (CAM), LCOM [13] and C3 [5]. The experimental evaluation has been performed on only 7 classes, but the results uncovered that a defect proneness is probable in the case of a less cohesive class that is coupled with another class. Chowdhury and Zulkernine [9] have also evaluated structural complexity, coupling and cohesion metrics for software vulnerability prediction, confirming their relevancy for this aim. Besides them, Dallal and Briand [10] have assessed cohesion metrics for early stage SDP. The results obtained revealed that low cohesion leads to more defects.

Other recent works related to cohesion include the empirical study of Zhao et al. [28] who have investigated the significance of considering client usage context in package cohesion for SDP at package level or the paper of Chen et al. [11] who have proposed, among other measures, Number of Topics (NT), which is a static metric expressing the cohesion of a source file using topics, computed using LDA. An additional cohesion metric based on the analysis of latent topics embedded the source code, utilizing LDA, is MWE, which has been proposed by Liu et al. [29].

Reiterating the conclusions of related studies, proposing new OO cohesion measures is a emergent [2], necessary [1] and promising [8], [9], [19], [30] research concern.

III. The proposed conceptual cohesion metrics

Let's consider an OO software system $\mathcal{S}yst$ as consisting of a set of nc classes: $\mathcal{S}yst = \{c_1, c_2, \dots, c_{nc}\}$. Each class $c_i \in \mathcal{S}yst$ $(1 \leq i \leq nc)$ is composed of a set of nm_i methods: $c_i = \{m_{i1}, m_{i2}, \dots, m_{inm_i}\}$.

A. Learning of the conceptual vectors

In our proposal, the methods are represented as *conceptual vectors* of numeric values corresponding to a set of conceptual features $\mathcal{S} = \{s_1, s_2, \dots, s_l\}$ that are unsupervisedly learned from the source code using Doc2Vec [23]. Accordingly, a method m_{ij} $(1 \leq i \leq nc,\ 1 \leq j \leq nm_i)$ is represented as a l-dimensional vector $m_{ij} = (m_{ij1}, m_{ij2}, \cdots, m_{ijl})$, where m_{ijk}, $(1 \leq k \leq l)$ denotes the value of the conceptual feature s_k computed for the method m_{ij}.

Paragraph Vector, or Doc2Vec, is a Multilayer Perceptron based model proposed by Le and Mikolov [23]. It allows expressing variable-length textual information as a fixed-length dense numeric vector, thus being an alternative to common models such as bag-of-words and bag-of-n-grams. The main advantage of Doc2Vec over traditional models is that it considers the semantic distance between words [23]. When compared to LSI [20], both are models aimed to represent texts of variable lengths as fixed-length numeric vectors capturing semantic characteristics, but Doc2Vec is a prediction-based model trained using backpropagation together with stochastic gradient descent, while LSI is a statistical, count-based model.

B. Definitions of the proposed metrics

Given the representation described above, we define a set of new metrics expressing the cohesion of classes in OO systems. The intuition behind is that the strength of conceptual similarities among the methods of a class (i.e., the similarities among their conceptual vectors) are able to convey to what extent they are related conceptually and thus to what degree a class represents a single concept in the problem or solution domain, thereby being cohesive.

Definition 3.1 (COSM: Conceptual Similarity between Methods)

We define the ***Conceptual Similarity between*** *two* ***Methods (COSM)*** m_{ij} *and* m_{ik} $(m_{ij}, m_{ik} \in c_i \in \mathcal{S}yst)$ *as being the similarity between their conceptual vectors,* $m_{ij} = (m_{ij1}, m_{ij2}, \cdots, m_{ijl})$ *and* $m_{ik} = (m_{ik1}, m_{ik2}, \cdots, m_{ikl})$*, in the* l*-dimensional semantic space learned by Doc2Vec.*

We mention that for expressing the similarity of two high dimensional vectors, both *cosine* (see Formula (1)) and *euclidean* (see Formula (2)) similarities can be used. We chose to define metrics using both of them since in a previous study [31] we concluded that they could be somewhat complementary in defining conceptual coupling since using them together boosted the SDP performance.

$$COSM^{cos}(m_{ij}, m_{ik}) = \frac{\sum\limits_{p=1}^{l} (m_{ijp} \cdot m_{ikp})}{\sqrt{\sum\limits_{p=1}^{l} (m_{ijp} \cdot m_{ijp})} \cdot \sqrt{\sum\limits_{p=1}^{l} (m_{ikp} \cdot m_{ikp})}} \tag{1}$$

$$COSM^{euc}(m_{ij}, m_{ik}) = \frac{1}{1 + \sqrt{\sum\limits_{p=1}^{l} (m_{ijp} - m_{ikp})^2}} \tag{2}$$

Definition 3.2 (ACOSM: Average Conceptual Similarity of Methods in a class)

The ***Average Conceptual Similarity of Methods (ACOSM) in a class*** $c_i \in \mathcal{S}yst$ *is defined as:*

$$ACOSM^{cos/euc}(c_i) = \frac{\sum_{p=1}^{\binom{nm_i}{2}} COSM^{cos/euc}(m_{ij}, m_{ik})}{\binom{nm_i}{2}} \tag{3}$$

, where $m_{ij}, m_{ik} \in c_i, 1 \leq j < k \leq nm_i$*,* $\binom{nm_i}{2}$ *being the number of combinations of two distinct methods of the class* c_i *containing a total of* nm_i *methods.*

Due to space limitations, we defined $ACOSM^{cos}$ and $ACOSM^{euc}$ through a single formula, by superscripting ACOSM and COSM with $^{cos/euc}$. Of course, $ACOSM^{cos}$ is defined based on $COSM^{cos}$, while $ACOSM^{euc}$ is defined based on $COSM^{euc}$. For the same reason and with the same interpretation, we will continue to use the superscript $^{cos/euc}$ in the current paper.

From our perspective, $ACOSM^{cos/euc}(c_i)$ expresses the degree to which the methods of class c_i relate conceptually and thus can act as a foundation for computing the class's conceptual cohesion.

Definition 3.3 (COCC: Conceptual Cohesion of Classes)

We define the ***Conceptual Cohesion of Classes (COCC)*** *for a class* $c_i \in \mathcal{S}yst$ *as:*

$$COCC^{cos/euc}(c_i) = \begin{cases} ACOSM^{cos/euc}(c_i), ACOSM^{cos/euc}(c_i) > 0 \\ 0, otherwise \end{cases} \tag{4}$$

The previous definition which is, in turn, parameterized with the desired similarity function, ensures the positivity of the conceptual cohesion. Additionally, the maximum value for COSM is 1 and, therefore, the same is for ACOSM, resulting in COCC being at most 1. The interpretation of the COCC value is as follows. The cohesion of a class $c_i \in \mathcal{S}yst$ increases as $COCC(c_i)$ approaches 1. If $COCC(c_i)$ is very close to one, implying that the methods of c_i are firmly conceptually related, then c_i is highly cohesive. Contrariwise, a class c_i lacks cohesion if its methods are poorly conceptually related, which means that they most likely contribute to the implementation of different concepts in the solution domain of the software project. Being an average measure, COCC is agnostic of the relative influence of highly conceptually related versus unrelated pairs of methods (for instance, we could have both in a class c_i having $COCC(c_i) \approx 0.5$). This is why, inspired by Marcus and Poshyvanyk [5], who raised the same issue for C3, we define in the following a complementary, inverse measure of cohesion.

Definition 3.4 (LCOSM: Lack of Conceptual Similarity between Methods)

Let the set $M_{ij}^{cos/euc}$ be defined as $M_{ij}^{cos/euc} = \{m_{ik}|, COSM^{cos/euc}(m_{ij}, m_{ik}) > ACOSM^{cos/euc}(c_i), 1 \leq j < k \leq nm_i\}$ be the set of methods of class c_i with which method m_{ij} of class c_i has an above-average COSM value.

From the sets $M_{ij}^{cos/euc}, 1 \leq i \leq nc, 1 \leq j \leq nm_i$, we derive two additional sets. Let's the set $P_i^{cos/euc}$ be defined as: $P_i^{cos/euc} = \{(M_{ij}^{cos/euc}, M_{ik}^{cos/euc})|M_{ij}^{cos/euc} \cap M_{ik}^{cos/euc} = \emptyset\}$ and the set $Q_i^{cos/euc}$ be defined as $Q_i^{cos/euc} = \{(M_{ij}^{cos/euc}, M_{ik}^{cos/euc})|M_{ij}^{cos/euc} \cap M_{ik}^{cos/euc} \neq \emptyset\}$.

*Eventually, we define the **Lack of Conceptual Similarity between Methods (LCOSM)** for a class $c_i \in \mathcal{S}yst$ as:*

$$LCOSM^{cos/euc}(c_i) = \begin{cases} \frac{|P_i^{cos/euc}|-|Q_i^{cos/euc}|}{\binom{nm_i}{2}}, |P_i^{cos/euc}| > |Q_i^{cos/euc}| \\ 0, otherwise \end{cases} \quad (5)$$

As well as LCSM [5] or LCOM2 [13], LCOSM is an inverse measure of cohesion (lower values denoting higher cohesion) intended to consider the influence of highly semantically similar methods of a class with low COCC and thus to complement COCC.

C. Theoretical validation

Both COCC and LCOSM comply the top three most important [5] mathematical properties, defined by Briand et al. [32], that provide a supportive underlying theory for class cohesion metrics: *non-negativity*, *normalization* and *null value*.

The *non-negativity* property states that cohesion metrics should be non-negative. Due to their definitions (see Formulae 4 and 5), both COCC and LCOSM have as the minimum possible value for a given class the value 0.

Normalization, an even more important property [32] facilitates interpretation and comparison by imposing the independence of cohesion value interval on the class's size. For any class $c_i \in \mathcal{S}yst$, $COCC^{cos/euc}(c_i)$ cannot exceed 1, which is the maximum value for $ACOSM^{cos/euc}(c_i)$. The maximum is reachable only when all the $\binom{nm_i}{2}$ pairs of methods of class c_i are maximally related (e.g., they are clone of each other), entailing that $COSM^{cos/euc}(m_{ij}, m_{ik}) = 1, \forall m_{ij}, m_{ik} \in c_i, 1 \leq j < k \leq nm_i$. The same upper limit stands for LCOSM. In accordance with Formula 5, given any class $c_i \in \mathcal{S}yst$, $LCOSM^{cos/euc}(c_i)$ would reach its maximum value when the difference $|P_i^{cos/euc}| - |Q_i^{cos/euc}|$ would be as large as possible. This would occur when $|P_i^{cos/euc}| = \binom{nm_i}{2}$, while $Q_i^{cos/euc} = \emptyset$. As a result, neither COCC nor LCSOM can exceed the value 1, regardless of the size of the class. We mention that, unlike LCOSM, the related LCSM metric [5] is not normalized.

The *null value* property holds if in the case of no cohesive relatedness among the methods of a class its cohesion is null. Considering any class $c_i \in \mathcal{S}yst$, if its methods are completely unrelated conceptually and thus $COSM^{euc}(m_{ij}, m_{ik}) = 0, \forall m_{ij}, m_{ik} \in c_i, 1 \leq j < k \leq nm_i$ then $ACOSM^{cos/euc}(c_i) = 0$ which, according to Formula 4, leads to $COCC^{cos/euc}(c_i) = 0$. As respects LCOSM, which is an inverse measure of cohesion, if $COSM^{euc}(m_{ij}, m_{ik}) = 0, \forall m_{ij}, m_{ik} \in c_i, 1 \leq j < k \leq nm_i$ and, as a result, $ACOSM^{cos/euc}(c_i) = 0$ then $|P_i^{cos/euc}| - |Q_i^{cos/euc}| = \binom{nm_i}{2} - 0 = \binom{nm_i}{2}$, which causes $LCOSM^{cos/euc}(c_i)$ to be 1. A maximum value for LCOSM means null cohesion, so the *null value* property is also satisfied by LCOSM.

The fourth property formulated by Briand et al. [32], even if in terms of structural relatedness among methods, is *monotonicity*. Reformulating it for conceptual cohesion, *monotonicity* holds that addition of conceptual relationships inside a class cannot descrease its cohesion. Intuitively, if the conceptual connection between two methods m_{ij} and m_{ik} of a class $c_i \in \mathcal{S}yst$ would be strengthened then both $COSM^{cos}(m_{ij}, m_{ik})$ and $COSM^{euc}(m_{ij}, m_{ik})$ would increase. This would obviously increase $ACOSM^{cos/euc}(c_i)$, since it is obtained by averaging these values, and, consequently, $COCC^{cos/euc}(c_i)$ would rise too. Therefore, we intuit that COCC satisfies even this fourth property.

IV. Assessment of the proposed cohesion metrics

A. Case studies

For assessing the relevance of the proposed cohesion metrics, we considered together the following criteria: their mathematical properties, their relationships with preexisting metrics and their discrimination power as an indicator of an external quality attribute (i.e., the defectiveness). The mathematical properties are discussed in Section III-C. In the current section, the remaining two criteria are addressed. They have been framed in the following two research questions:

- RQ1: Do COCC and LCOSM capture dimensions of class cohesion that are not captured by already existing structural or conceptual cohesion metrics?
- RQ2: Do COCC and LCOSM provide better results in predicting software defects than existing related conceptual cohesion metrics?

We addressed each of the research question through one dedicated case study.

The first case study has as its goal to determine whether COCC and LCOSM capture additional aspects of class cohesion when compared to preexisting cohesion metrics.

The second case study evaluates the proposed conceptual cohesion metrics versus other existing conceptual cohesion metrics from the perspective of their usefulness for SDP.

B. Design of the case studies

1) Software systems: For the empirical assessment of the proposed cohesion metrics, we have considered three open source Java software systems: Ant (version 1.7), Tomcat (version 6.0) and JEdit (version 4.2). Their size and defectiveness are described in Table I.

Software system	Number of defective classes	Number of non-defective classes	Percentage of non-defective classes
Ant	166	575	22.4%
Tomcat	77	726	9.6%
JEdit	48	307	13.5%

TABLE I: Description of the considered software systems

2) Cohesion metrics: Along the first evaluation criteria, we analyzed the relationships of the proposed conceptual cohesion metrics with preexisting cohesion metrics, both structural and conceptual. We have selected the following structural metrics to compare our metrics with: LCOM1 [13], LCOM2 [13], LCOM3 [14], LCOM4 [14], LCOM5 [15] and YALCOM [4]. LCOM1-LCOM5 have been extensively studied in the literature [8], [12], while YALCOM is the state-of-the-art variant of LCOM, being proposed by Sharma and Spinellis at the end of 2020 [4]. In the paper that introduces it, YALCOM is shown to perform superior (from the perspective of its ability to accurately capture structural cohesion) compared to the previous LCOM metrics. From the cohesion class the metrics we propose belong to, namely that of conceptual cohesion, we chose, naturally, the metrics proposed by Marcus and Poshyvanyk [5], namely C3 and LCSM, due to their relatedness with COCC and LCOSM, respectively. They are defined similarly, excepting that the underlying mechanism is LSI, instead of Doc2Vec, only cosine similarity is used and, unlike LCOSM, LCSM does not comply with the normalization [32] property.

3) Classification labels: To enable the comparative evaluation along the second criteria (i.e., in the frame of SDP), we extracted information regarding the defectiveness of the classes from the PROMISE repository [33]. While originally labeled with the number of defects, we labeled each class with a binary label denoting whether it is defective or not thus formulating a binary classification problem. In our experiments, we considered all classes for which both the source code and the label were available. Their number is given in Table I.

4) Settings of the case studies: We computed the conceptual cohesion metrics COCC and LCOSM by applying the following methodology. Firstly, for constructing the corpora on which to train Doc2Vec, we extracted from the source code of each software system all methods, including the afferent comments. We filtered the tokens in order to keep only the ones presumably carrying semantic meaning. So, operators, special symbols, English stop words or Java keywords have been eliminated. Secondly, conceptual features are unsupervisedly learned from the corpora, after 50 training epochs, using the Doc2Vec model parameterized so as to produce vector representations of length 10. Thirdly, we computed the conceptual similarities between all pairs of methods (i.e., COSM) in each class as the cosine and euclidean similarities between their conceptual vectors (according to the Formulae 1 and 2, respectively) and their average value (i.e., ACOSM), according to Formula 3. Finally, we derived from these, by applying Formulae 4 and 5, the values for the proposed cohesion metrics: COCC and LCOSM, respectively.

A similar methodology have been applied for computing C3 and LCSM. The corpora have been built identically, but, for extracting the conceptual vectors, for which we kept the same length of 10, Doc2Vec has been replaced by LSI. Also, we applied exactly the formulas accompanying the definitions introduced in [5] for C3 and LCSM. This implies that, as already mentioned, LCSM is not normalized by its original definition.

All structural cohesion metrics considered for comparison (LCOM1-LCOM5 and YALCOM) have been collected using the replication package provided by Sharma and Spinellis [4].

C. First case study

With the aim of confirming that the proposed metrics are indeed capturing additional aspects of cohesion when compared to existing cohesion metrics, we performed a correlation analysis. To test if there is significant linear or non-linear connection between our metrics and existing ones, we computed Pearson correlations (which measure only linear monotonic relations) and non-parametric Spearman correlations (which measure non-linear monotonic relations) among them.

D. Second case study

The goal of this case study was to determine whether there is empirical evidence that COOCC and LCOSM are better than or at least complementary to existing, comparable conceptual cohesion metrics. In this study, we rely on the widely accepted and used assumption that a software metric that predicts defects more precisely is expected to be a better quality indicator. To this effect, we firstly performed a *difficulty* analysis and then an experimental comparison from a supervised ML perspective.

1) Difficulty analysis: The *difficulty* [34] of a SDP data set is computed as the ratio of defective instances for which the nearest neighbor is a non-defective instance. It is a measure of how challenging SDP is when starting from a given set of features. Therefore, by computing the difficulty values considering different sets of cohesion metrics, one can assess their relative relevance for SDP. When the difficulty of a defect data set is higher, it is harder to discriminate between its defective and non-defective instances. Given that LCSM is not normalized [5], we normalized it here using min-max scaling.

2) Supervised learning analysis: The comparison among the conceptual cohesion metrics has been completed by a study performed from a supervised SDP perspective (i.e., the binary classification problem of deciding whether a class characterized by given conceptual cohesion values is defective or defectless). We selected two different ML models: k-Nearest Neighbors (kNN, an instance based model) and Random Forest (RF, a tree-based model). We used their Orange [35] implementations, with the following parameterizations. For kNN, we selected euclidean distance and considered 10 neighbors, while opting for 10 trees in the case of RF. The evaluation methodology we applied is *leave-one-out* (LOO), while choosing the *Area under the ROC curve* (AUC) as performance indicator since it is considered in the literature to be suitable for evaluating the SDP performance [36].

V. Results and discussion

A. Results of the first case study

The heatmaps illustrating the resulted Pearson and Spearman correlations among the twelve cohesion metrics are depicted in Figure 1 and Figure 2, respectively.

Size of correlation	Interpretation
0.9 to 1 (-1 to -0.9)	Very high positive (negative) correlation
0.7 to 0.9 (-0.9 to -0.7)	High positive (negative) correlation
0.5 to 0.7 (-0.7 to -0.5)	Moderate positive (negative) correlation
0.3 to 0.5 (-0.5 to -0.3)	Low positive (negative) correlation
0 to 0.3 (-0.3 to 0)	Negligible correlation

TABLE II: Rule of thumb for interpreting the size of a correlation coefficient [37]

For interpreting the strength of correlation coefficients we used the rule described in Table II. To reach the goal of the case study, we will only focus on the correlations with COCC and LCOSM which allowed formulating the following conclusions.

- No more than negligible or low Pearson correlation was found with any of the considered structural cohesion metrics. The same goes for Spearman, excepting only one moderate negative correlation (-0.52) between LCOSM^{euc} and LCOM1, for Ant.
- COCC and LCOSM are at most moderately linearly correlated with C3 and at most low linearly correlated with LCSM. Contrariwise, the Spearman correlations of COCC and LCOSM with C3 are all negligible or low, but, for Tomcat only, high positive correlation (0.76) was found between LCOSM and LCSM. A possible explanation would be the similarity of the counting procedure. For the other software systems, however, this correlation is low or moderate.
- The only high Pearson and Spearman correlations within our metrics set were found between LCOSM^{cos} and LCOSM^{euc}, which is not surprising, given the usage of the same conceptual information and the common counting mechanism.

The conclusions of the first case study confirm that COCC and LCOSM are able to capture dimensions of cohesion that are not already captured by existing metrics, given their predominantly negligible, low or moderate correlations with LCOM1-5, YALCOM, C3 and LCSM. Consequently, the the

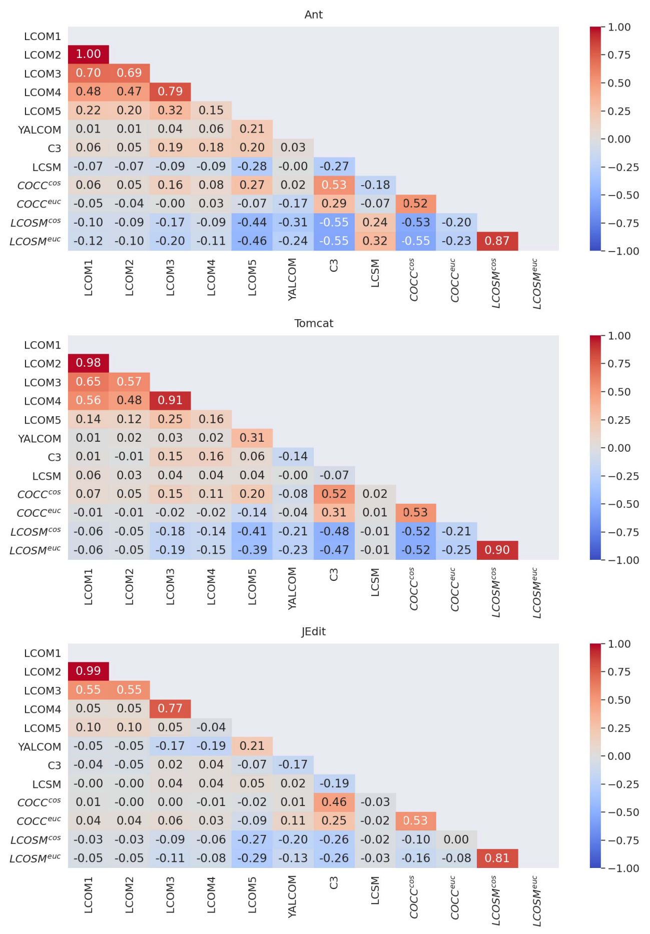

Fig. 1: Pearson correlation heatmaps

first research question, RQ1 (see Section IV-A), is answered affirmatively. Moreover, excepting LCOSM^{euc} and LCOSM^{euc}, our metrics are not even highly inter-correlated.

B. Results of the second case study

1) Difficulty analysis: The results of the difficulty analysis, approximated to three decimal places, are given in Table III. Based on these, we formulate the following observations:

- By replacing the C3 [5] related conceptual cohesion metric with COOC^{cos} (which just involves substituting LSI with Doc2Vec for learning conceptual vectors) the difficulty of the SDP data sets is reduced with 8.21% for Ant, 8.95% for Tomcat and 16.28% for JEdit.
- An even slightly higher reduction in difficulty is obtained when LSCM is replaced with LCSOM^{cos} as well and thus the conceptual cohesion metrics proposed in [5] are substituted by our metrics defined using the cosine similarity function (as in [5]).
- Excepting Ant, the difficulty is further reduced (reaching a reduction of 25.58% for JEdit) by considering the entire set of metrics we propose, including the ones in the alternative definition using euclidean similarity.

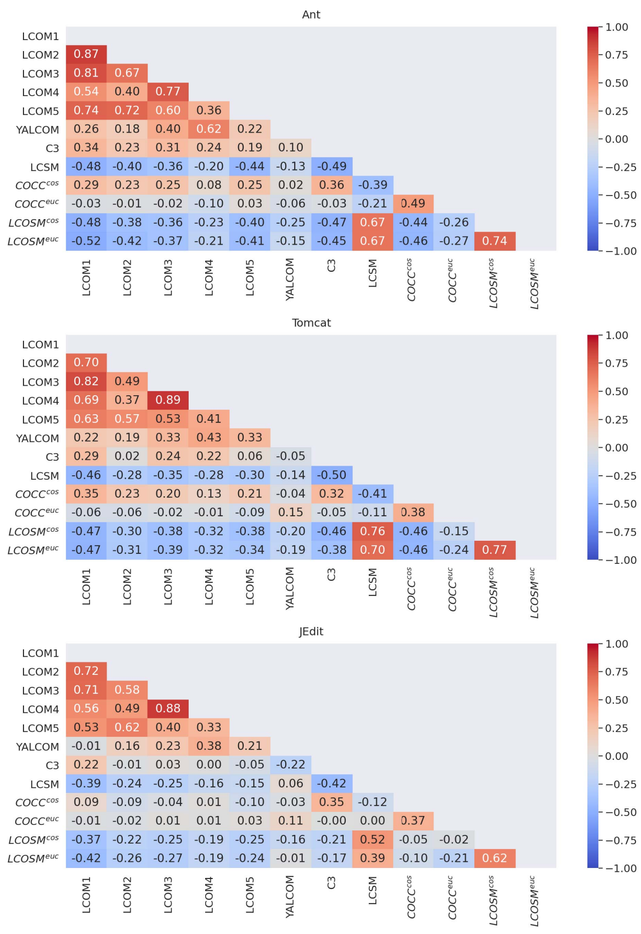

Fig. 2: Spearman correlation heatmaps

- In the case of Ant and Tomcat, combining all our metrics with the C3 and LCSM leads to the best (i.e., lowest) difficulty. For JEdit, the best difficulty is obtained when considering all our four metrics alone.

So, we can conclude that, according to the difficulty analysis, COCC and LCOSM facilitate the SDP task by considerably reducing the difficulty of distinguishing the defective classes from the others.

Cohesion metrics considered as input features for SDP	Ant	Tomcat	JEdit
{C3}	0.807	0.883	0.896
{COCC^{cos}}	0.741	0.804	0.750
{C3, LCSM}	0.801	0.883	0.896
{COCC^{cos}, LCOSM^{cos}}	0.729	0.804	0.750
{COCC^{cos}, COCC^{euc}, LCOSM^{cos}, LCOSM^{euc}}	0.735	0.792	**0.667**
{C3, LCSM, COCC^{cos}, LCOSM^{cos}}	0.747	0.740	0.708
{C3, LCSM, COCC^{cos}, COCC^{euc}, LCOSM^{cos}, LCOSM^{euc}}	**0.663**	**0.701**	0.792

TABLE III: SDP data sets' difficulty

2) Supervised learning analysis: The AUC values obtained when applying kNN and RF on labeled metric data are given in Table IV and Table V, respectively. They empirically confirm that:

- COCC^{cos} is mostly a better defect predictor than C3 since it averagely increased the AUC with 18.78%.
- The same stands for the partial or entire set of conceptual cohesion metrics we propose (i.e., {COCC^{cos}, LCOSM^{cos}} or {COCC^{cos}, COCC^{euc}, LCOSM^{cos}, LCOSM^{euc}}), when compared to the set of conceptual cohesion metrics introduced in [5] (i.e., {C3, LCSM}). Replacing {C3, LCSM} with {COCC^{cos}, COCC^{euc}, LCOSM^{cos}, LCOSM^{euc}} leads to an average improvement in AUC of 33.14%.
- Furthermore, in most cases, combining the four metrics introduced in the current paper (that are COCC and LCSOM defined using both cosine and euclidean similarities) provides the best AUC values. In the rest of the cases, combining them with the metrics introduced in [5] increases the AUC values even more.

Cohesion metrics considered as input features for SDP	Ant	Tomcat	JEdit
{C3}	0.571	0.624	0.519
{COCC^{cos}}	0.644	0.620	0.725
{C3, LCSM}	0.601	0.631	0.517
{COCC^{cos}, LCOSM^{cos}}	0.656	0.622	0.729
{COCC^{cos}, COCC^{euc}, LCOSM^{cos}, LCOSM^{euc}}	**0.758**	0.714	**0.762**
{C3, LCSM, COCC^{cos}, LCOSM^{cos}}	0.673	0.702	0.740
{C3, LCSM, COCC^{cos}, COCC^{euc}, LCOSM^{cos}, LCOSM^{euc}}	0.688	**0.740**	**0.762**

TABLE IV: AUC values obtained using kNN

Cohesion metrics considered as input features for SDP	Ant	Tomcat	JEdit
{C3}	0.514	0.524	0.507
{COCC^{cos}}	0.592	0.627	0.639
{C3, LCSM}	0.552	0.523	0.493
{COCC^{cos}, LCOSM^{cos}}	0.587	0.631	0.591
{COCC^{cos}, COCC^{euc}, LCOSM^{cos}, LCOSM^{euc}}	**0.728**	**0.718**	0.705
{C3, LCSM, COCC^{cos}, LCOSM^{cos}}	0.624	0.686	0.700
{C3, LCSM, COCC^{cos}, COCC^{euc}, LCOSM^{cos}, LCOSM^{euc}}	0.659	0.701	**0.711**

TABLE V: AUC values obtained using RF

The results of the supervised learning analysis confirm the ones of the difficulty analysis, while together affirmatively answering the second research question, RQ2, formulated in Section IV-A.

VI. Conclusions and Future Work

This paper proposed a new set of Doc2Vec based metrics for expressing the conceptual cohesion of classes in OO software systems. Through two case studies on three open source software systems, they have been empirically shown to capture additional dimensions of cohesion when compared to preexisting structural and conceptual metrics and to be better software defect predictors than related conceptual cohesion metrics. The underlying mechanism for computing the proposed metrics is general enough to adapt to different programming languages and granularities (e.g., package-level or system-level cohesion).

So, a first direction of future work would be to extend their empirical assessment, for instance by validating that they are indeed language-agnostic. A second direction would be to include the proposed conceptual cohesion metrics, along with structural cohesion metrics, in aggregated cohesion metrics. Additionally, since coupling and cohesion are complementary in assessing software design quality, other future research ideas would be to try understanding the trade-off between them, in the context of SDP, and to unite their power by developing a new extensive metrics suite for SDP, composed of new metrics derived from aggregated coupling and aggregated cohesion.

Acknowledgements

This work was supported by a grant of the Ministry of Research, Innovation and Digitization, CNCS/CCCDI – UEFISCDI, project number PN-III-P4-ID-PCE-2020-0800, within PNCDI III.

References

[1] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," *Artificial Intelligence Review*, vol. 51, no. 2, pp. 255–327, 2019.

[2] S. Tiwari and S. S. Rathore, "Coupling and cohesion metrics for object-oriented software: A systematic mapping study," in *Proceedings of the 11th Innovations in Software Engineering Conference*, ser. ISEC '18. New York, USA: Association for Computing Machinery, 2018.

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

[4] T. Sharma and D. Spinellis, "Do we need improved code quality metrics?" *CoRR*, vol. abs/2012.12324, 2020. [Online]. Available: https://arxiv.org/abs/2012.12324

[5] A. Marcus and D. Poshyvanyk, "The conceptual cohesion of classes," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*, 2005, pp. 133–142.

[6] M. D'Ambros, A. Bacchelli, and M. Lanza, "On the impact of design flaws on software defects," in *2010 10th International Conference on Quality Software*, 2010, pp. 23–31.

[7] G. P. Kartha, C. Anjali, R. V. Nair, and S. Venkateswari, "Prediction of defect susceptibility in object oriented software," in *2017 International Conference on Networks Advances in Computational Technologies (NetACT)*, 2017, pp. 467–472.

[8] J. Al Dallal and L. C. Briand, "A precise method-method interaction-based cohesion metric for object-oriented classes," *ACM Trans. Softw. Eng. Methodol.*, vol. 21, no. 2, Mar. 2012.

[9] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, 2011.

[10] J. Al Dallal and L. C. Briand, "An object-oriented high-level design-based class cohesion metric," *Information and Software Technology*, vol. 52, no. 12, pp. 1346–1361, 2010.

[11] T.-H. Chen, W. Shang, M. Nagappan, A. E. Hassan, and S. W. Thomas, "Topic-based software defect explanation," *J. Syst. Softw.*, vol. 129, no. C, p. 79–106, Jul. 2017.

[12] A. Marcus, D. Poshyvanyk, and R. Ferenc, "Using the conceptual cohesion of classes for fault prediction in object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 287–300, 2008.

[13] S. R. Chidamber and C. F. Kemerer, "Towards a metrics suite for object oriented design," *SIGPLAN Not.*, vol. 26, no. 11, pp. 197–211, Nov. 1991.

[14] M. Hitz and B. Montazeri, "Measuring coupling and cohesion in object-oriented systems," in *Proceedings of International Symposium on Applied Corporate Computing*, 1995, pp. 25–27.

[15] M. West, "Object-oriented metrics: Measures of complexity, by brian henderson-sellers, prentice hall, 1996 (book review)," *Softw. Test. Verification Reliab.*, vol. 6, pp. 255–256, 1996.

[16] M. Hitz and B. Montazeri, "Measuring coupling and cohesion in object-oriented systems," in *Proceedings of International Symposium on Applied Corporate Computing*, 1995, pp. 25–27.

[17] J. Eder, G. Kappel, and M. Schrefl, "Coupling and cohesion in object-oriented systems," 1992.

[18] L. Etzkorn and H. Delugach, "Towards a semantic metrics suite for object-oriented design," in *Proceedings. 34th International Conference on Technology of Object-Oriented Languages and Systems - TOOLS 34*, 2000, pp. 71–80.

[19] B. Újházi, R. Ferenc, D. Poshyvanyk, and T. Gyimóthy, "New conceptual coupling and cohesion metrics for object-oriented systems," in *2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*, 2010, pp. 33–42.

[20] H. Chen, B. Martin, C. Daimon, and S. Maudsley, "Effective use of latent semantic indexing and computational linguistics in biological and biomedical applications," *Frontiers in Physiology*, vol. 4, p. 8, 2013.

[21] D.-L. Miholca, G. Czibula, and V. Tomescu, "COMET: A conceptual coupling based metrics suite for software defect prediction," *Procedia Computer Science*, vol. 176, pp. 31–40, 2020, knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 24th International Conference KES2020.

[22] D. Miholca and Z. Onet-Marian, "An analysis of aggregated coupling's suitability for software defect prediction," in *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE Computer Society, sep 2020, pp. 141–148.

[23] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," *CoRR*, vol. abs/1405.4053, 2014.

[24] S. Husein and A. Oxley, "A coupling and cohesion metrics suite for object-oriented software," in *2009 International Conference on Computer Technology and Development*, vol. 1, 2009, pp. 421–425.

[25] M. O. Elish and M. Al-Khiaty, "A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software," *Journal of Software: Evolution and Process*, vol. 25, 2013.

[26] A. De Lucia, R. Oliveto, and L. Vorraro, "Using structural and semantic metrics to improve class cohesion," in *2008 IEEE International Conference on Software Maintenance*, 2008, pp. 27–36.

[27] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[28] Y. Zhao, Y. Yang, H. Lu, J. Liu, H. Leung, Y. Wu, Y. Zhou, and B. Xu, "Understanding the value of considering client usage context in package cohesion for fault-proneness prediction," *Automated Software Engg.*, vol. 24, no. 2, p. 393–453, Jun. 2017.

[29] Y. Liu, D. Poshyvanyk, R. Ferenc, T. Gyimothy, and N. Chrisochoides, "Modeling class cohesion as mixtures of latent topics," in *2009 IEEE International Conference on Software Maintenance*, 2009, pp. 233–242.

[30] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimóthy, "Using information retrieval based coupling measures for impact analysis," *Empirical Softw. Engg.*, vol. 14, no. 1, pp. 5–32, Feb. 2009.

[31] D. Miholca and Z. Onet-Marian, "An analysis of aggregated coupling's suitability for software defect prediction," in *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE Computer Society, sep 2020, pp. 141–148.

[32] L. Briand, S. Morasca, and V. Basili, "Property-based software engineering measurement," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 68–86, 1996.

[33] S. Sayyad and T. Menzies, "The PROMISE Repository of Software Engineering Databases," School of Information Technology and Engineering, University of Ottawa, Canada, 2015. [Online]. Available: http://promise.site.uottawa.ca/SERepository

[34] D. Zhang, J. Tsai, and G. Boetticher, "Improving credibility of machine learner models in software engineering," in *Advances in Machine Learning Applications in Software Engineering*, 2007, pp. 52–72.

[35] J. Demšar, T. Curk, A. Erjavec, Črt Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, and B. Zupan, "Orange: Data mining toolbox in python," *Journal of Machine Learning Research*, vol. 14, pp. 2349–2353, 2013.

[36] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006.

[37] M. Mukaka, "Statistics corner: A guide to appropriate use of correlation coefficient in medical research," *Malawi medical journal : the journal of Medical Association of Malawi*, vol. 24, pp. 69–71, 09 2012.