STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LII**, Number 1, 2007

ON EVALUATING SOFTWARE SYSTEMS DESIGN

GABRIELA ŞERBAN AND ISTVÁN GERGELY CZIBULA

Abstract. We have previously introduced in [1, 2] clustering approaches for identifying refactorings in order to improve the structure of software systems. For this purpose, we have defined in [2] a *semi-metric* function in order to express the dissimilarity between the entities from a software system. In this paper we aim at giving a theoretical validation for this *semi-metric*. In other words, we are focussing on proving that this function illustrates the cohesion between the entities from a software system and can be used in order to obtain appropriate refactorings of it.

Keywords: software engineering, refactoring, distance metric, clustering.

1. Introduction

Improving the quality of a software system design is the most important issue during the evolution of object oriented software systems.

Refactoring is the process of improving the design of software systems. It improves the internal structure of the system, but without altering the external behavior of the code ([5]).

During the software development cycle, there is a continuous alternance between adding new tests and functionalities for a software system, and refactoring the code in order to improve its internal consistency and clarity.

We have previously introduced in [1, 2] clustering approaches for identifying refactorings in order to improve the structure of software systems. For this purpose, we have defined in [2] a *semi-metric* function in order to express the dissimilarity between the entities from a software system.

Received by the editors: April, 20.

2000 *Mathematics Subject Classification.* 68N99, 62H30.

1998 *CR Categories and Descriptors.* D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement –*Restructuring, reverse engineering, and reengineering*; I.5.3 [**Computing Methodologies**]: Pattern Recognition – *Clustering*.

The main contribution of this paper is to give a theoretical validation for this *semi-metric* function, and, consequently, a theoretical validation for the clustering approaches.

The rest of the paper is structured as follows. The main aspects related to the clustering approach for systems design improvement (previously introduced in [1, 2]) are exposed in Section 2.1. The theoretical validation of the *semi-metric* distance d used in the clustering approach from [2] is given in Section 3. Section 4 provides an experimental validation of d. Conclusions and further work are outlined in Section 5.

2. Background

2.1. **Clustering approach for refactorings determination.** In this section we briefly describe the clustering approach for improving software systems design (*CARD*) previously introduced in [1, 2].

In [1], a software system S is viewed as a set $S = \{s_1, s_2, \ldots, s_n\}$, where $s_i, 1 \leq i \leq n$, may be an application class, a method from a class, or an attribute from a class. *CARD* consists of three steps:

- **Data collection.** The existent software system is analyzed in order to extract from it the relevant entities: classes, methods, attributes and the existent relationships between them.
- **Grouping.** The set of entities extracted at the previous step are re-grouped in clusters using a partitioning algorithm (like *kRED* algoritm in [1] or *PAMRED* algorithm in [2]). The goal of this step is to obtain an improved structure of the existing software system.
- **Refactorings extraction.** The newly obtained software structure is compared with the original software structure in order to provide a list of refactorings which transform the original structure into an improved one.

A more detailed description of *CARD* is given in [1].

At the **Grouping** step of *CARD*, the software system S has to be re-grouped. This re-grouping is represented as a ***partition*** $\mathcal{K} = \{K_1, K_2, \ldots, K_v\}$ of S. K_i is the i-th *cluster* of $\mathcal{K}$, and an element s_i from S is referred as an *entity*. A cluster K_i from the partition $\mathcal{K}$ represents an application class in the new structure of the software system.

2.2. **A semi-metric dissimilarity function.** In the clustering approaches from [1, 2], the objects to be clustered are the entities from the software system S, i.e.,

$\mathcal{O} = \{s_1, s_2, \ldots, s_n\}$. Our focus is to group similar entities from S in order to obtain high cohesive groups (clusters).

In [1], we have adapted the generic cohesion measure introduced in [6] that is connected with the theory of similarity and dissimilarity. In order to express the dissimilarity degree between any two entities from the software system S, we have considered the distance $d(s_i, s_j)$ between two entities s_i and s_j as expressed in Equation (1) ([1]).

$$d(s_i, s_j) = \begin{cases} 1 - \frac{|p(s_i) \cap p(s_j)|}{|p(s_i) \cup p(s_j)|} & if \quad p(s_i) \cap p(s_j) \neq \emptyset \\ \infty & otherwise \end{cases}, \tag{1}$$

where, for a given entity $e \in S$, $p(e)$ represents a set of relevant properties of e, defined as:

- If e is an attribute, then $p(e)$ consists of: the attribute itself, the application class where the attribute is defined, and all methods from S that access the attribute.
- If e is a method, then $p(e)$ consists of: the method itself, the application class where the method is defined, and all attributes from S accessed by the method.
- If e is a class, then $p(e)$ consists of: the application class itself, and all attributes and methods defined in the class.

We have chosen the distance between two entities as expressed in Equation (1) because it emphasizes the idea of cohesion. As illustrated in [7], "*Cohesion refers to the degree to which module components belong together*".

Based on the definition of distance d given in Equation (1) it can be easily proved that d is a semi-metric function.

3. Theoretical validation

In this section we are focusing on giving a theoretical validation of the *semi-metric* dissimilarity function d described in Subsection 2.2. We aim at proving that our distance, as defined in Equation (1), highlight the concept of cohesion, i.e., entities with low distances are cohesive, whereas entities with higher distances are less cohesive. This theoretical validation of d will give a validation of the clustering approach from [2], also.

Let us consider that e, α and β are three entities from the software system S, $e \neq \alpha \neq \beta$. In Lemma 1 we give a necessary and sufficient condition in order to illustrate that entity e is more distant from entity β than from entity α.

We consider, in the following, the definition of the distance function d given in Equation (1).

Lemma 1. *If e, α and β are three entities from the software system S, and $p(e) \cap p(\alpha) \neq \emptyset$, then*

$$d(e, \alpha) < d(e, \beta) \tag{2}$$

iff

$$\frac{|p(e) \cap p(\alpha)|}{|p(e)| + |p(\alpha)|} > \frac{|p(e) \cap p(\beta)|}{|p(e)| + |p(\beta)|}. \tag{3}$$

Proof.

Since $p(e) \cap p(\alpha) \neq \emptyset$, we have that

$$|p(e) \cap p(\alpha)| > 0. \tag{4}$$

From (4) we can deduce that:

$$d(e, \alpha) = 1 - \frac{|p(e) \cap p(\alpha)|}{|p(e) \cup p(\alpha)|}. \tag{5}$$

First, we prove implication "⇒" from Lemma 1.

Let us assume that Inequality (2) holds. We have to prove that Inequality (3) holds, also.

We have two situations in which Inequality (2) holds:

1. $d(e, \beta) = \infty$.

   This means (from Equation (1)) that entities e and β are unrelated and have no common relevant properties. Consequently we have that $p(e) \cap p(\beta) = \emptyset$. It follows that

$$|p(e) \cap p(\beta)| = 0. \tag{6}$$

   From (6) and (4) we have that Inequality (3) holds. So, implication "⇒" from Lemma 1 is proved.
2. $d(e, \beta) < 1$.

   This means (from Equation (1)) that entities e and β are related and have common relevant properties. Consequently, using (5), it follows that:

$$1 - \frac{|p(\alpha) \cap p(e)|}{|p(\alpha) \cup p(e)|} < 1 - \frac{|p(\beta) \cap p(e)|}{|p(\beta) \cup p(e)|}. \tag{7}$$

From (7) we have that:

$$\frac{|p(\beta) \cap p(e)|}{|p(\beta) \cup p(e)|} < \frac{|p(\alpha) \cap p(e)|}{|p(\alpha) \cup p(e)|}. \tag{8}$$

If A and B are two sets, it is well known that Equality (9) holds.

$$|A \cup B| = |A| + |B| - |A \cap B|. \tag{9}$$

From (8) and (9) it follows that:

$$|p(\beta) \cap p(e)| \cdot (|p(\alpha)| + |p(e)| - |p(\alpha) \cap p(e)|) < \\ |p(\alpha) \cap p(e)| \cdot (|p(\beta)| + |p(e)| - |p(\beta) \cap p(e)|). \tag{10}$$

Using (10) we have that:

$$|p(\beta) \cap p(e)| \cdot (|p(\alpha)| + |p(e)|) < |p(\alpha) \cap p(e)| \cdot (|p(\beta)| + |p(e)|). \tag{11}$$

Inequality (11) implies Inequality (3). So, implication "⇒" from Lemma 1 is proved.

Now, we prove implication "⇐" from Lemma 1.

Let us assume that Inequality (3) holds. We have to prove that Inequality (2) holds, also.

We have two situations in which Inequality (3) holds:

1. $p(\beta) \cap p(e) = \emptyset$.

   This means (from Equation (1)) that entities e and β are unrelated and have no common relevant properties. Consequently we have that $d(e, \beta) = \infty$. It follows that Inequality (2) holds and implication "⇐" from Lemma 1 is proved.
2. $p(\beta) \cap p(e) \neq \emptyset$.

   This means (from Equation (1)) that entities e and β are related and have common relevant properties. Consequently it follows that:

$$d(e, \beta) = 1 - \frac{|p(e) \cap p(\beta)|}{|p(e) \cup p(\beta)|}. \tag{12}$$

From (3) we have that:

$$|p(\alpha) \cap p(e)| \cdot (|p(\beta)| + |p(e)|) > |p(\beta) \cap p(e)| \cdot (|p(\alpha)| + |p(e)|). \tag{13}$$

Consequently, we can deduce that:

$$|p(\alpha) \cap p(e)| \cdot (|p(\beta)| + |p(e)|) - |p(\alpha) \cap p(e)| \cdot |p(\beta) \cap p(e)| > \\ |p(\beta) \cap p(e)| \cdot (|p(\alpha)| + |p(e)|) - |p(\alpha) \cap p(e)| \cdot |p(\beta) \cap p(e)|. \tag{14}$$

From (14) we have that:

$$|p(\alpha) \cap p(e)| \cdot (|p(\beta)| + |p(e)| - |p(\beta) \cap p(e)|) > \\ |p(\beta) \cap p(e)| \cdot (|p(\alpha)| + |p(e)| - |p(\alpha) \cap p(e)|). \tag{15}$$

From (15) and (9) it follows that:

$$|p(\alpha) \cap p(e)| \cdot |p(\beta) \cup p(e)| > |p(\beta) \cap p(e)| \cdot |p(\alpha) \cup p(e)| \tag{16}$$

Using (16) we can deduce that:

$$\frac{|p(\alpha) \cap p(e)|}{|p(\alpha) \cup p(e)|} > \frac{|p(\beta) \cap p(e)|}{|p(\beta) \cup p(e)|}. \tag{17}$$

Consequently, we have that:

$$1 - \frac{|p(\alpha) \cap p(e)|}{|p(\alpha) \cup p(e)|} < 1 - \frac{|p(\beta) \cap p(e)|}{|p(\beta) \cup p(e)|}. \tag{18}$$

From (18), (5) and (12) it follows that Inequality (2) holds and implication "$\Leftarrow$" from Lemma 1 is proved.

As both implications "$\Rightarrow$" and "$\Leftarrow$" from Lemma 1 were proved, Lemma 1 is also proved.

Let us consider that α and β are two entities of the software system S that are situated in different application classes, and e is an entity of S that has to be disposed in one of the application classes (corresponding to α or β).

In this situation, Inequality (3) from Lemma 1 expresses that the number of elements that α has in common with e with respect to the total number of elements from α and e is greater than the number of elements that β has in common with

e with respect to the total number of elements from β and e. This is very likely to express that e is more cohesive with α than with β.

Intuitively, condition (3) is very probable a necessary and suficient condition to indicate that e belongs to the same application class with α and not with β. This statement cannot be rigurously proved, because the decision that an entity to be disposed in an application class or another is very complex and cannot be quantified using rigurous mathematical measures. In practice, the developers decide wheather or not an entity is disposed into an application class, and the decision can be a subjective one. Still, in Subsection 3.1 we give an experimental justification for this statement.

Consequently, we can consider that Proposition 1 is valid.

Proposition 1. *If α is an entity of S situated in application class A and β is an entity of S that is situated in application class B ($B \neq A$), and e is an entity of S that has to be disposed in one of the application classes A or B, then e belongs to A and does not belong to B iff Inequality (3) holds.*

From Lemma 1 and Proposition 1 results the mathematical validation of the semi-metric distance function d, i.e., the facts that:

(1) Entities with low distances are cohesive, whereas entities with higher distances are less cohesive.
(2) The distances between less cohesive entities are greater than the distances between cohesive entities.

This theoretical validation of d is given in Lemma 2.

Lemma 2. *If α is an entity of S situated in application class A and β is an entity of S that is situated in application class B ($B \neq A$), and e is an entity of S that has to be disposed in one of the application classes A or B, then e belongs to A and does not belong to B iff $d(e, \alpha) < d(e, \beta)$.*

From Lemma 2 we can conclude that the decision about putting an entity e from S into an application class or another is based on the distances between e and the entities from the corresponding application classes.

Consequently, a clustering approach that uses the semi-metric d for expressing the dissimilarity between the entities from the software system is very appropriate, and can be used in order to recondition the class structure of a software system, because it expresses the cohesion between the entities from it.

3.1. **Example.** Let us consider the software system S given by the Java code example shown in Figure 1.

```
public class Class_A {
  public static int attributeA1;
  public static int attributeA2;

  public static void methodA1(){
      attributeA1 = 0;
      methodA2();
  }

  public static void methodA2(){
          attributeA2 = 0;
          attributeA1 = 0;
    }

  public static void methodA3(){
          attributeA2 = 0;
          attributeA1 = 0;
          methodA1();
          methodA2();
  }
}
```

```
public class Class_B {
  private static int attributeB1;
  private static int attributeB2;

  public static void methodB1(){
          Class_A.attributeA1=0;
          Class_A.attributeA2=0;
          Class_A.methodA1();
    }

  public static void methodB2(){
          attributeB1=0;
          attributeB2=0;
    }

  public static void methodB3(){
          attributeB1=0;
          methodB1();
          methodB2();
  }
}
```

FIGURE 1. Code example for *Move Method* refactoring

Analyzing the code presented in Figure 1, it is obvious that the method **methodB1()** has to belong to **class_A**, because it uses features of **class_A** only. This means, according to Proposition 1, that Inequality (3) holds for $e =$ **methodB1()**, $\forall \alpha \in$ **class_A**, and $\forall \beta \in$ **class_B**.

Analyzing the code from Figure 1 we can observe that all other entities e from both classes **class_A** and **class_B**, excepting **methodB1()** are correctly disposed in their application classes *App*. This means, according to Proposition 1, that Inequality (3) holds for e, $\forall \alpha \in App$, and $\forall \beta \in C$ ($C \in Class(S)$, $C \neq App$).

We have verified Proposition 1 $\forall e, \alpha, \beta \in S$ that satisfy its hypothesis and we have concluded that the proposition is valid.

For lack of space, we will illustrate in Table 1 the validity of Proposition 1 for e=**methodB1()** and in Table 2 the validity of Proposition 1 for e=**methodB2()**. We aim at illustrating that:

(i) **methodB1()** is more cohesive with entities from **class_A**, consequently the *Move Method* refactoring **methodB1()** from **class_B** to **class_A** will be determined by the clustering approach from [2].

(ii) **methodB2()** is more cohesive with entities from **class_B**, consequently its position in **class_B** is correct.

Items (i) and (ii) are expressed in Tables 1 and 2 by giving in column **Class** the class in which entity $e \in \{$**methodB1()**, **methodB2()**$\}$ should be disposed. The correct application class in which **methodB1()** should be disposed is **class_A** and the correct application class in which **methodB2()** should be disposed is **class_B**.

The results illustrated in Tables 1 and 2 validate experimentally Proposition 1 for the considered software system, according to the considerations in this subsection.

4. Experimental Validation

An experimental validation of the *semi-metric d* (Subsection 2.2) is given in [2]. In [2] we have introduced, based on the *semi-metric d*, a *k-medoids* like clustering algorithm (*PAMRED*) for identifying refactorings in order to improve the design of a software system. *PAMRED* algorithm can be used in the **Grouping** step of *CARD*.

PAMRED algorithm is evaluated on the open source case study JHotDraw ([8]) and a comparison with previous related approaches is also given. This comparison illustrates that *CARD* with *PAMRED* algorithm is better than other similar approaches existing in the literature in the field of refactoring.

5. Conclusions and Future Work

We have given in this paper a theoretical validation of the *semi-metric* function d previously introduced in [1] and used in the clustering approach introduced in [2]. In fact, we have validated our previous approach from [2], approach that can be used to determine refactorings in order to improve the structure of a software system. As a future work we intend:

- To give theoretical validation for other distance functions that were used in our previous clustering approaches for refactorings determination ([1]).
- To develop other distance metrics to be used in clustering approaches for refactorings determination and to give theoretical validations for them.

e	α	β	$\frac{\lvert p(e)\cap p(\alpha)\rvert}{\lvert p(e)\rvert+\lvert p(\alpha)\rvert} - \frac{\lvert p(e)\cap p(\beta)\rvert}{\lvert p(e)\rvert+\lvert p(\beta)\rvert}$	**Class**
methodB1()	methodA1()	methodB2()	0.11111111	Class_A
methodB1()	methodA1()	methodB3()	0.02222222	Class_A
methodB1()	methodA1()	attributeB1()	0.11111111	Class_A
methodB1()	methodA1()	attributeB2()	0.097222224	Class_A
methodB1()	methodA1()	Class_B	0.055555552	Class_A
methodB1()	methodA2()	methodB2()	0.11111111	Class_A
methodB1()	methodA2()	methodB3()	0.02222222	Class_A
methodB1()	methodA2()	attributeB1	0.11111111	Class_A
methodB1()	methodA2()	attributeB2	0.097222224	Class_A
methodB1()	methodA2()	Class_B	0.055555552	Class_A
methodB1()	methodA3()	methodB2()	0.16161618	Class_A
methodB1()	methodA3()	methodB3()	0.07272728	Class_A
methodB1()	methodA3()	attributeB1	0.16161618	Class_A
methodB1()	methodA3()	attributeB2	0.14772728	Class_A
methodB1()	methodA3()	Class_B	0.10606061	Class_A
methodB1()	methodB2()	attributeA1	-0.16161618	Class_A
methodB1()	methodB2()	attributeA2	-0.08888889	Class_A
methodB1()	methodB2()	Class_A	-0.1388889	Class_A
methodB1()	methodB3()	attributeA1	-0.07272728	Class_A
methodB1()	methodB3()	attributeA2	0.0	Class_A
methodB1()	methodB3()	Class_A	-0.049999997	Class_A
methodB1()	attributeA1()	attributeB1	0.16161618	Class_A
methodB1()	attributeA1()	attributeB2	0.14772728	Class_A
methodB1()	attributeA1()	Class_B	0.10606061	Class_A
methodB1()	attributeA2()	attributeB1	0.08888889	Class_A
methodB1()	attributeA2()	attributeB2	0.075	Class_A
methodB1()	attributeA2()	Class_B	0.03333333	Class_A
methodB1()	attributeB1()	Class_A	-0.1388889	Class_A
methodB1()	attributeB2()	Class_A	-0.125	Class_A
methodB1()	Class_A	Class_B	0.08333333	Class_A

TABLE 1. Validation of Proposition 1 for entity e=methodB1().

REFERENCES

[1] Czibula, I.G., Serban, G.: Improving Systems Design Using a Clustering Approach. IJCSNS International Journal of Computer Science and Network Security, VOL.6, No.12 (2006) 40–49

e	α	β	$\frac{\lvert p(e)\cap p(\alpha)\rvert}{\lvert p(e)\rvert+\lvert p(\alpha)\rvert} - \frac{\lvert p(e)\cap p(\beta)\rvert}{\lvert p(e)\rvert+\lvert p(\beta)\rvert}$	**Class**
methodB2()	methodA1()	methodB1()	-0.11111111	Class_B
methodB2()	methodA1()	methodB3()	-0.33333334	Class_B
methodB2()	methodA1()	attributeB1	-0.375	Class_B
methodB2()	methodA1()	attributeB2	-0.42857143	Class_B
methodB2()	methodA1()	Class_B	-0.36363637	Class_B
methodB2()	methodA2	methodB1()	-0.11111111	Class_B
methodB2()	methodA2	methodB3()	-0.33333334	Class_B
methodB2()	methodA2	attributeB1	-0.375	Class_B
methodB2()	methodA2	attributeB2	-0.42857143	Class_B
methodB2()	methodA2	Class_B	-0.36363637	Class_B
methodB2()	methodA3	methodB1()	-0.11111111	Class_B
methodB2()	methodA3	methodB3()	-0.33333334	Class_B
methodB2()	methodA3	attributeB1	-0.375	Class_B
methodB2()	methodA3	attributeB2	-0.42857143	Class_B
methodB2()	methodA3	Class_B	-0.36363637	Class_B
methodB2()	methodB1	attributeA1	0.11111111	Class_B
methodB2()	methodB1	attributeA2	0.11111111	Class_B
methodB2()	methodB1	Class_A	0.11111111	Class_B
methodB2()	methodB3	attributeA1	0.33333334	Class_B
methodB2()	methodB3	attributeA2	0.33333334	Class_B
methodB2()	methodB3	Class_A	0.33333334	Class_B
methodB2()	attributeA1	attributeB1	-0.375	Class_B
methodB2()	attributeA1	attributeB2	-0.42857143	Class_B
methodB2()	attributeA1	Class_B	-0.36363637	Class_B
methodB2()	attributeA2	attributeB1	-0.375	Class_B
methodB2()	attributeA2	attributeB2	-0.42857143	Class_B
methodB2()	attributeA2	Class_B	-0.36363637	Class_B
methodB2()	attributeB1	Class_A	0.375	Class_B
methodB2()	attributeB2	Class_A	0.42857143	Class_B
methodB2()	Class_A	Class_B	-0.36363637	Class_B

TABLE 2. Validation of Proposition 1 for entity e=methodB2().

[2] Serban, G., Czibula, I.G.: A New Clustering Approach for Systems Designs Improvement. 2007 International Conference on Software Engineering Theory and Practice, SETP-07, Orlando, USA (2007) to appear

[3] Simon, F., Steinbruckner, F., Lewerentz, C.: Metrics based refactoring. In: Proc. European Conf. Software Maintenance and Reengineering. IEEE Computer Society Press (2001) 30-38

[4] Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers (2001)
[5] Fowler, M.: Improving the design of existing code. Addison-Wesley, New-York (1999)
[6] Simon, F., Loffler, S., Lewerentz, C.: Distance based cohesion measuring. In Proceedings of the 2nd European Software Measurement Conference (FESMA) 99, Technologisch Instituut Amsterdam (1999)
[7] Bieman, J.M., Kang, B.-K.: Measuring Design-Level Cohesion. In: IEEE Transactions on Software Engineering **24** No. 2 (1998)
[8] JHotDraw Project: http://sourceforge.net/projects/jhotdraw (1997)

Department of Computer Science, Babeş-Bolyai University 1, M. Kogalniceanu Street, Cluj-Napoca, Romania,
E-mail address: gabis@cs.ubbcluj.ro

Department of Computer Science, Babeş-Bolyai University, 1, M. Kogalniceanu Street, Cluj-Napoca, Romania,
E-mail address: istvanc@cs.ubbcluj.ro