UML class diagram object-oriented metrics: algorithms of calculation

Olga Deryugina [1, *]

[1]Moscow Technological University (MIREA), 119454 Moscow, Russia

Abstract. The paper proposes algorithms of the object oriented UML class diagram metrics calculation. Metrics include Average CBO, Average DIT, Average NOC, Average DAC, Average NLM, Average NOM, DAC2, SIZE2, and DSC. The proposed algorithms have become a part of the UML Refactoring tool providing UML class diagram analysis and transformation.

1 Introduction

Model-Driven Architecture (MDA) approach [1], which was developed by the Object-Management Group (OMG), introduces a new software development paradigm. This paradigm proposes a design process, which starts with the creation of the PIM (Platform Independent Model). Then the PIM is automatically transformed into the PSM (Platform Specific Model).

MDA approach proposes a set of standards including MOF (Meta-Object Facilities) [2], UML (Unified Modelling Language) [3], XMI (XML Metadata Interchange) [4], etc., which provide model design, transformation, validation and exchange.

The problem of model refactoring is tightly connected with the MDA approach: model refactoring is less time consuming than code refactoring. In addition, models can help a developer to look at the software system from a perspective view.

In the MDA approach, software models are designed using UML diagrams. UML class diagrams describe main classes, interfaces of the system and the relations between them.

With the software systems of the large size, it becomes harder to analyze UML class diagrams manually. Software tools of the automated UML class diagrams refactoring have been developed: Dearthóir [5], Darwin [6], CODe-Imp [7], Bunch tool [8], UML Refactoring [9], etc.

These tools use a fitness function for the refactoring process. Nowadays a number of studies have investigated several means of the object oriented metrics [10-14]. Some of them can be calculated for the UML class diagrams too [15] (see Fig. 1).

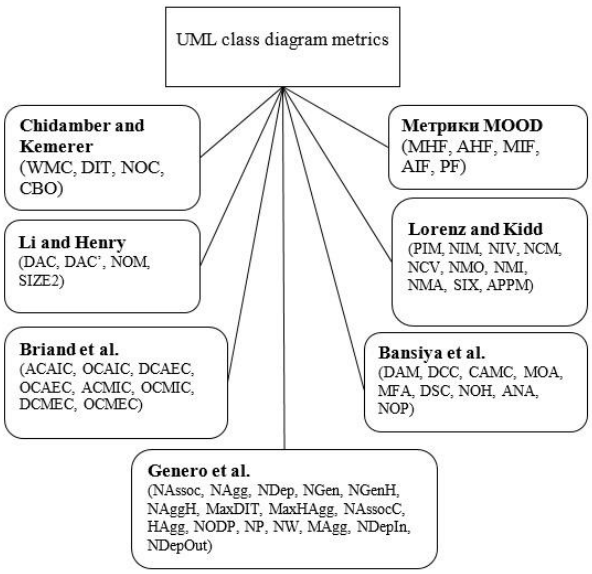

Fig. 1. UML class diagram metrics

The aim of this paper is to propose algorithms of several object oriented UML class diagram metrics calculation based on the abstract data structure UML Map [9].

The rest of this paper is organized as follows: First, the UML Map abstract data structure, which stores elements of a UML class diagram in hash maps, is introduced in Section 2. In Section 3, the algorithms of object oriented UML class diagram metrics calculation including Avg. CBO, Avg. DIT, Avg. NOC, Avg. DAC, Avg. NLM, Avg. NOM, DAC2, SIZE2, and DSC [9] are proposed. In addition, the experimental results of applying these algorithms to class diagrams of different size are presented. In Section 4, the role of the proposed algorithms in the UML Refactoring software tool is introduced before concluding in Section 5.

* Corresponding author : o.a.derugina@yandex.ru

2 UML Map abstract data structure

UML Map abstract data structure provide UML model storage, analysis and transformation. It is based on the hash maps of UML model elements. The reduced UML class diagram of the UML Map abstract data structure is shown in Fig. 2.

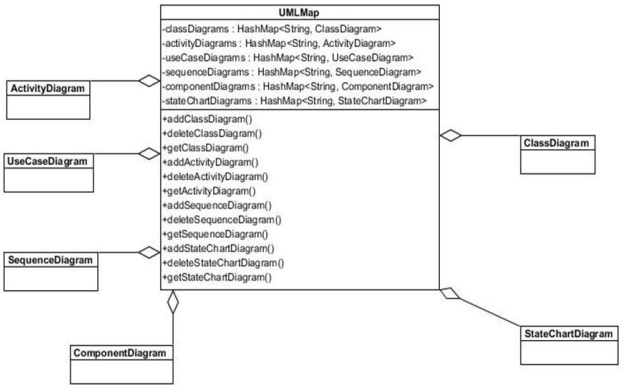

Fig. 2. Reduced class diagram of the UML Map abstract data structure

The detailed UML class diagram of the classes, providing analysis and transformation of UML class diagrams is shown in Fig. 3.

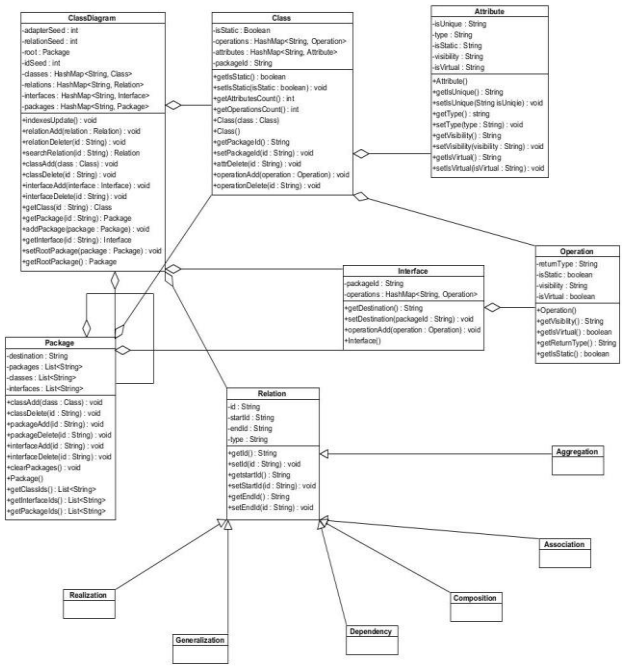

Fig. 3. UML class diagram of the UML Map classes providing UML class diagram storage, analysis and transformation

For example, the complexity of searching a class with the xmi:id = "X" in an XMI text document is evaluated as $O(n)$, while the complexity of searching in a hash map storing pairs of keys and values {xmi:id, Class} is evaluated as $O(1)$.

Special Translator has been developed to translate an XMI file to the UML Map and from the UML Map to an XMI file.

With the UML Map, it is possible to calculate metrics, search elements where transformation application is convenient, automatically transform diagrams, etc.

3 Algorithms of the object-oriented UML class diagram metrics calculation

The Average CBO metric can be calculated as follows:

```
1.public double calculate(ClassDiagram classDiagram) {
2.  double result = getAverageCBO(classDiagram);
3.  this.value = new SimpleDoubleProperty(result);
4.  return result;
5.}
6.public double getAverageCBO( ClassDiagram classDiagram) {
7.  int classCount = classDiagram.getClasses().size();
8.  double sumCBO = 0;
9.  for (Entry<String,Class> entry :
10.classDiagram.getClasses().entrySet()) {
11.    sumCBO += getClassCBO(entry.getKey(),classDiagram);
12.  }
13.  return  roundUp(sumCBO/classCount,3);
14.}
15.//Coupling Between Objects for a class
16.public Integer getClassCBO(String classID, ClassDiagram
classDiagram) {
17.  Integer classCBO=0;
18.  for (Entry<String, Relation> entry :
19.(classDiagram.getRelations()).entrySet()) {
20.    Relation relation = entry.getValue();
21.    if (!relation.getType().equals("generalization")) {
22.      if (relation.getEndId().equals(classID) ||
relation.getStartId().equals(classID) ) {
23.        classCBO++;
24.      }
25.    }
26.  }
27.  return classCBO;
28.}
```

A graph depicting the execution time of the algorithm depending on the UML class diagram size is shown in Fig. 4. The value of the execution time was calculated as the average execution time measured during 10000 runs for each UML class diagram size.

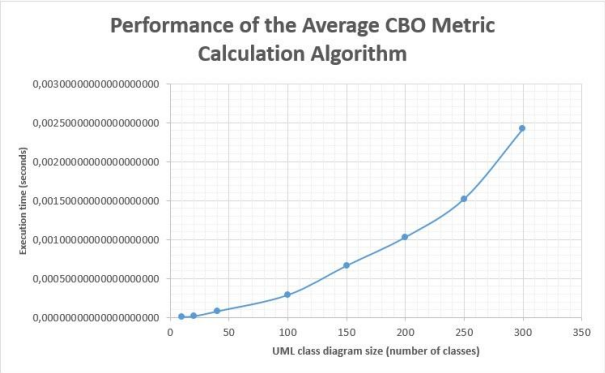

Fig. 4. Performance of the Average CBO Metric Calculation Algorithm

The Average DIT metric can be calculated as follows:

```
1.public double getAverageDIT( ClassDiagram classDiagram ) {
2.  int classCount = classDiagram.getClasses().size();
3.  double sumDIT = 0;
4.  for (Entry<String,Class> entry :
classDiagram.getClasses().entrySet()) {
5.    sumDIT += getClassDIT(entry.getKey(),classDiagram);
6.  }
7.  return  roundUp(sumDIT/classCount,3);
8.}
9.//Depth of Inheritance Tree
10.public Integer getClassDIT (String classID, ClassDiagram
classDiagram ){
11.  int classDIT = 0;
12.  List<String> classParentIds = new ArrayList<String>();
```

```
13.  for (Entry<String, Relation> entry :
(classDiagram.getRelations()).entrySet()) {
14.    Relation relation = entry.getValue();
15.    if (relation.getType().equals("generalization")) {
16.      if (relation.getStartId().equals(classID)) {
17.        classParentIds.add(relation.getEndId());
18.      }
19.    }
20.  }
21.  for (int i=0; i<classParentIds.size();i++) {
22.    int classDITTemp =
getClassDIT(classParentIds.get(i),classDiagram);
23.    if (classDIT < classDITTemp+1) {
24.      classDIT = classDITTemp+1;
25.    }
26.  }
27.  return classDIT;
28.}
```

A graph depicting the execution time of the algorithm depending on the UML class diagram size is shown in Fig. 5. The value of the execution time was calculated as the average execution time measured during 10000 runs for each UML class diagram size.

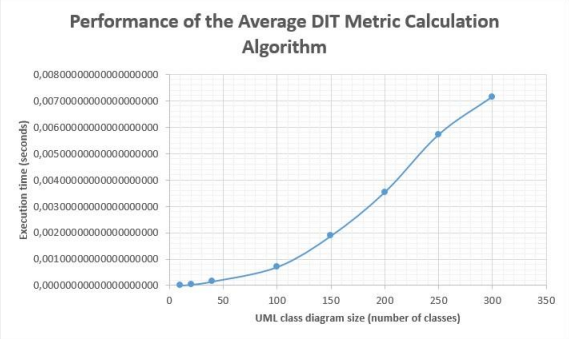

Fig. 5. Performance of the Average DIT Metric Calculation Algorithm

The Average NOC metric can be calculated as follows:

```
1.public double getAverageNOC(ClassDiagram classDiagram) {
2.  int classCount = classDiagram.getClasses().size();
3.  double sumNOC = 0;
4.  for (Entry<String,Class> entry :
classDiagram.getClasses().entrySet()) {
5.    sumNOC += getClassNOC(entry.getKey(),classDiagram);
6.  }
7.  double averageNOC = roundUp(sumNOC/classCount,3);
8. return  averageNOC;
9.}
10.//Number of Children  - number of immediate classes
11.public  Integer getClassNOC (String classId,ClassDiagram
classDiagram){
12.  int classNOC = 0;
13.  for (Entry<String, Relation> entry :
(classDiagram.getRelations().entrySet())) {
14.    Relation relationTemp = entry.getValue();
15.    if (relationTemp.getType().equals("generalization"))
{
16.      if (relationTemp.getEndId().equals(classId)) {
17.        classNOC++;
18.      }
19.     }
20.  }
21.  return classNOC;
22.}
```

A graph depicting the execution time of the algorithm depending on the UML class diagram size is shown in Fig. 6. The value of the execution time was calculated as the average execution time measured during 10000 runs for each UML class diagram size.

The Average DAC metric can be calculated as follows:

```
1.public double getAverageDAC(ClassDiagram classDiagram) {
2.  double averageDAC = 0;
3.  int sumDAC = 0;
4.  for (Entry<String,Class> entry :
classDiagram.getClasses().entrySet()) {
5.    Class classTemp = entry.getValue();
6.    sumDAC += getDAC(classTemp,classDiagram);
7.  }
8.  int classCount = classDiagram.getClasses().size();
9.  averageDAC = roundUp(sumDAC/classCount,3);
10.  return  averageDAC;
11.}
12.public int getDAC(Class classTemp, ClassDiagram
classDiagram) {
13.  int DAC = 0;
14.  for (Entry<String,Attribute> entry:
classTemp.getAttributes().entrySet()) {
15.    Attribute attributeTemp = entry.getValue();
16.    if ( searchClassByName(attributeTemp.getType(),
classDiagram)) {
17.      DAC++;
18.    }
19.  }
20.  return DAC;
21.}
```

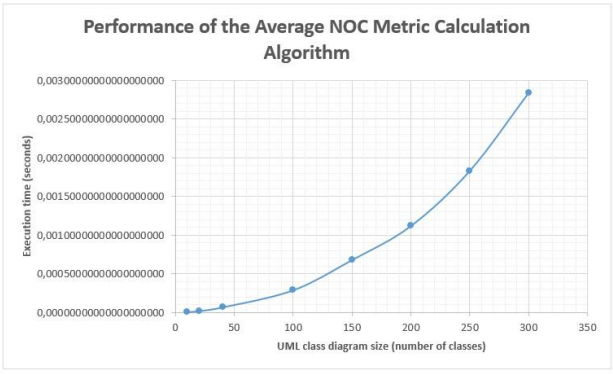

Fig. 6. Performance of the Average NOC Metric Calculation Algorithm

A graph depicting the execution time of the algorithm depending on the UML class diagram size is shown in Fig. 7. The value of the execution time was calculated as the average execution time measured during 10000 runs for each UML class diagram size.

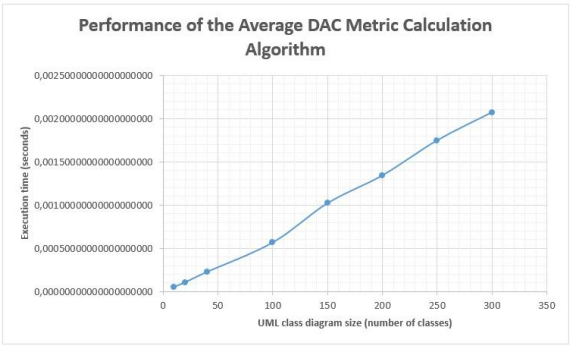

Fig. 7. Performance of the Average DAC Metric Calculation Algorithm

The Average NLM metric can be calculated as follows:

```
1.public double getAverageNLM(ClassDiagram classDiagram) {
2.  int classCount = classDiagram.getClasses().size();
3.  double sumNLM = 0;
4.  for (Entry<String,Class> entry :
classDiagram.getClasses().entrySet()) {
5.    sumNLM += getNLM(entry.getKey(),classDiagram);
6.  }
7.  double averageNLM = roundUp(sumNLM/classCount,3);
8.  return  averageNLM;
9.}
```

```
10.public int getNLM(String classId, ClassDiagram
classDiagram) {
11.  int NLM = 0;
12.  Class classTemp = classDiagram.getClass(classId);
13.  for (Entry<String,Operation> entry :
classTemp.getOperations().entrySet()) {
14.    Operation operationTemp = entry.getValue();
15.    if (!operationTemp.getVisibilty().equals("public")) {
16.      NLM++;
17.    }
18.  }
19.  return NLM;
20.}
```

A graph depicting the execution time of the algorithm depending on the UML class diagram size is shown in Fig. 8. The value of the execution time was calculated as the average execution time measured during 10000 runs for each UML class diagram size.

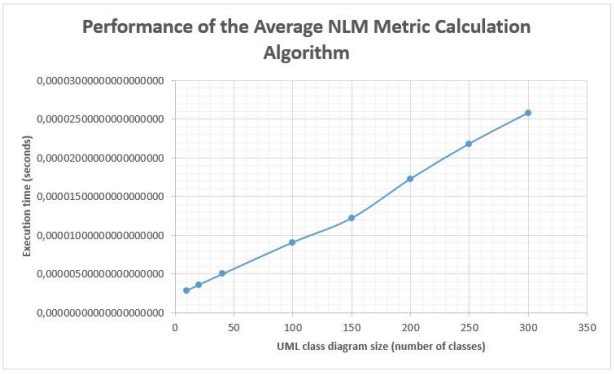

Fig. 8. Performance of the Average NLM Metric Calculation Algorithm

The Average NOM metric can be calculated as follows:

```
1.public double getAverageNOM(ClassDiagram classDiagram) {
2.  int classCount = classDiagram.getClasses().size();
3.  double sumNOM = 0;
4.  for (Entry<String,Class> entry :
classDiagram.getClasses().entrySet()) {
5.    Class classTemp = entry.getValue();
6.    sumNOM += getNOM(classTemp);
7.  }
8.  double averageNOM = roundUp(sumNOM/classCount,3);
9.  return  averageNOM;
10.}
11.public int getNOM(Class classTemp) {
12.  return classTemp.getOperations().size();
13.}
```

A graph depicting the execution time of the algorithm depending on the UML class diagram size is shown in Fig. 9. The value of the execution time was calculated as the average execution time measured during 10000 runs for each UML class diagram size.

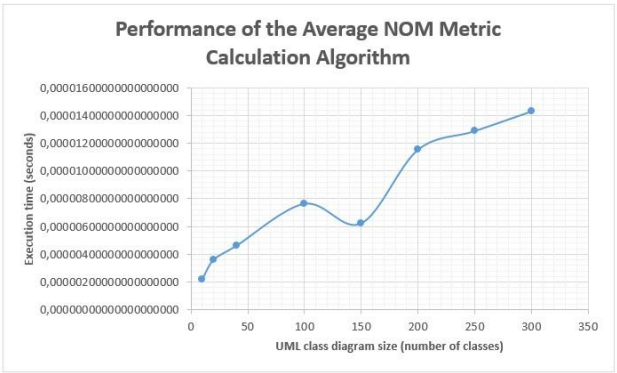

Fig. 9. Performance of the Average NOM Metric Calculation Algorithm

The Average DAC2 metric can be calculated as follows:

```
1.public int getDAC2(ClassDiagram classDiagram) {
2.  int DAC2 = 0;
3.  for (Entry<String,Class> entry :
classDiagram.getClasses().entrySet()) {
4.    Class classTemp = entry.getValue();
5.    if (getDAC2(classTemp,classDiagram)) {
6.    DAC2++;
7.    }
8.  }
9.  return  DAC2;
10.}
11.public boolean getDAC2(Class classTemp, ClassDiagram
classDiagram) {
12.  for (Entry<String,Attribute> entry:
classTemp.getAttributes().entrySet()) {
13.    Attribute attributeTemp = entry.getValue();
14.    if ( searchClassByName(attributeTemp.getType(),
classDiagram)) {
15.      return true;
16.    }
17.  }
18.  return false;
19.}
```

A graph depicting the execution time of the algorithm depending on the UML class diagram size is shown in Fig. 10. The value of the execution time was calculated as the average execution time measured during 10000 runs for each UML class diagram size.

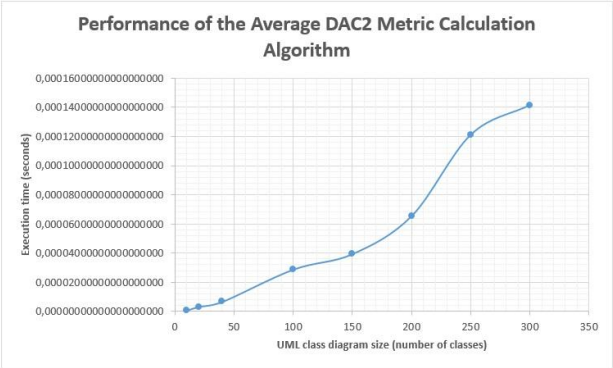

Fig. 10. Performance of the Average DAC2 Metric Calculation Algorithm

The Average SIZE2 metric can be calculated as follows:

```
1.public double getAverageSIZE2(ClassDiagram classDiagram) {
2.  int classCount = classDiagram.getClasses().size();
3.  int sumSIZE2 = 0;
4.  for (Entry<String,Class> entry :
classDiagram.getClasses().entrySet()) {
5.    sumSIZE2 += getSIZE2(entry.getKey(),classDiagram);
6.  }
7.  double averageSIZE2 = roundUp(sumSIZE2/classCount,3);
8.  return  averageSIZE2;
9.}
10.public double getSIZE2(String classId, ClassDiagram
classDiagram) {
11.  Class classTemp = classDiagram.getClass(classId);
12.  DACMetric dac = new DACMetric();
13.  double SIZE2 = classTemp.getAttributes().size() +
dac.calculate(classId, classDiagram);
14.  return SIZE2;
15.}
```

A graph depicting the execution time of the algorithm depending on the UML class diagram size is shown in Fig. 11. The value of the execution time was calculated as the average execution time measured during 10000 runs for each UML class diagram size.

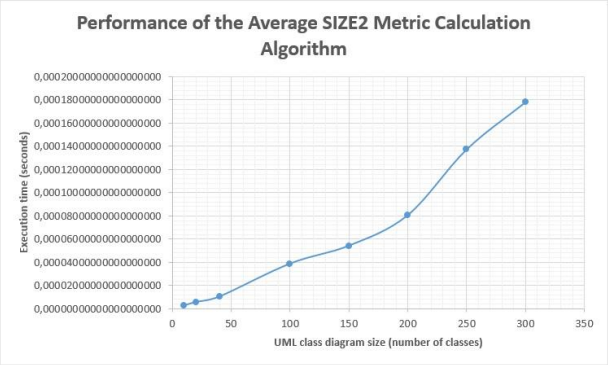

Fig. 11. Performance of the Average SIZE2 Metric Calculation Algorithm

The DSC metric can be calculated as follows:

```
1.public double getComplexity(ClassDiagram classDiagram) {
2.  int c = classDiagram.getClasses().size(); //classes
count
3.  int i = classDiagram.getInterfaces().size(); //
interfaces count
4.  int r = classDiagram.getRelations().size(); // relations
count
5.  //count attributes
6.  int a = 0; // attributes count
7.  for (Entry<String, Class> entry :
classDiagram.getClasses().entrySet()) {
8.    Class c0 = entry.getValue();
9.    a += c0.getAttributes().size();
10.  }
11.  //count class methods
12.  int m1 = 0;
13.  for (Entry<String, Class> entry :
classDiagram.getClasses().entrySet()) {
14.    Class c0 = entry.getValue();
15.    int m0 = c0.getOperations().size();
16.    int m00 = 0;
17.    //search interfaces, realized by this class
18.    for (Entry<String, Relation> relEntry :
(classDiagram.getRelations()).entrySet()) {

19.      Relation rel = relEntry.getValue();
20.      if (rel.getStartId().equals(c0.getId())) {
21.        if (rel.getType().equals("realization")) {
22.          Interface interTemp =
classDiagram.getInterfaces().get(rel.getEndId());
23.          m00 += interTemp.getOperations().size();
24.        }
25.      }
26.    }
27.    m1 += m0 - m00;
28.  }
29.  //count interface methods
30.  int m2 = 0;
31.  for (Entry<String, Interface> entry :
classDiagram.getInterfaces().entrySet()) {
32.    Interface i0 = entry.getValue();
33.    m2 += i0.getOperations().size();
34.  }
35.  double complexity = k1*c + k2*i + k3*r + k4*a +
k5*m1+k6*m2;
36.  complexity = roundUp(complexity,3);
37.  return complexity;
38.}
```

A graph depicting the execution time of the algorithm depending on the UML class diagram size is shown in Fig. 12. The value of the execution time was calculated as the average execution time measured during 10000 runs for each UML class diagram size.

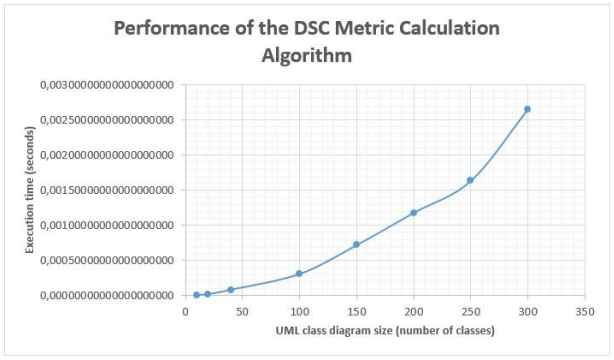

Fig. 12. Performance of the DSC Metric Calculation Algorithm

6 UML Refactoring tool

The introduced algorithms have become a part of the automated UML class diagram refactoring framework UML Refactoring [9], which can be downloaded from the official site of the project: www.uml-refactoring.ru.

This tool calculates various object-oriented metrics (such as Avg. DIT, Avg. CBO, DAC, SIZE2, etc.) and proposes a list of transformations (Interface Insertion, Façade, Strategy), which minimize a fitness function value. In addition, the tool applies the chosen transformation to the diagram. Framework can import/export projects from/to the XMI format.

The main window of the UML Refactoring tool is shown in Fig. 13.

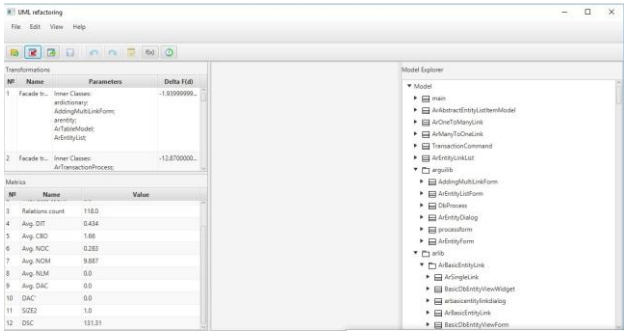

Fig. 13. Performance of the DSC Metric Calculation Algorithm

The fitness function selection window is shown in Fig. 14.

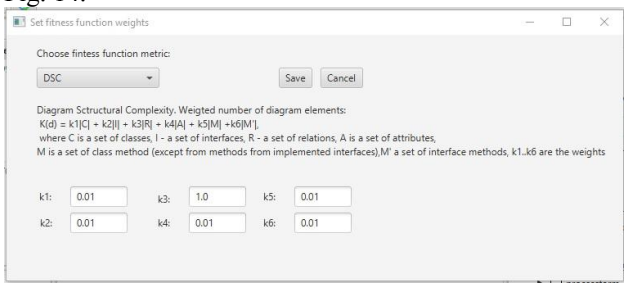

Fig. 14. Performance of the DSC Metric Calculation Algorithm

7 Conclusions

The proposed UML Map abstract data structure provides convenient UML class diagram analysis and transformation.

The proposed algorithms of the object oriented UML class diagram metrics calculation (including Average CBO, Average DIT, Average NOC, Average DAC, Average NLM, Average NOM, DAC2, SIZE2, and DSC are proposed) have become a part of the UML Refactoring software tool.

References

[1] OMG Model Driven Architecture (http://www.omg.org/mda)

[2] OMG Meta Object Facility Core Specification. Version 2.5.1. (http://www.omg.org/spec/MOF/2.5.1)

[3] OMG Unified Modelling Language UML. Version 2.5 (http://www.omg.org/spec/UML/2.5)

[4] XML Metadata Interchange (XMI) Specification. Version 2.4.2 (http://www.omg.org/spec/XMI/2.4.2)

[5] M. O'Keeffe and M. Ó Cinnéide, *Towards automated design improvements through combinatorial optimization* (W2S, 2004)

[6] S. Hadaytullah, Vathsavayi, O. Räihä, and K. Koskimies, *Tool support for software architecture design with genetic algorithms* (IEEE CS Press, 2010)

[7] M. O'Keeffe and M. Ó Cinnéide, *Search-based software maintenance* (CSMR, 2006)

[8] S. Mancoridis, B.S. Mitchell, C. Rorres, Y.F. Chen, and E.R. Gansner, *Using automatic clustering to produce high-level system organizations of source code* (IWPC, 1998)

[9] E. Nikulchev, O. Deryugina, International Journal of Advanced Computer Science and Applications, **7**, 76 (2016)

[10] S. R. Chidamber and C. F. Kemerer, IEEE Transaction on Software Engineering, **20**, 476 (1994)

[11] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics: A Practical Guide*, Prentice Hall, Englewood Cliffs, New Jersey, (1994)

[12] F. Brito e Abreu, L. Ochoa and M. Goulao, *The MOOD metrics set* (INESC/ISEG, 1998)

[13] L. Briand, W. Devanbu and W. Melo, *An investigation into coupling measures for C++* (ICSE, 1997)

[14] M. Marchesi, *OOA Metrics for the Unified Modeling Language* (Euromicro Conference on Software Maintenance and Reengineering, 1998)

[15] M. Genero, M. Piattini and C. Calero, L´Objet, **6**, 4, 489 (2001)