STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 2, 2008

SOFTWARE QUALITY ASSESSMENT USING A FUZZY CLUSTERING APPROACH

CAMELIA SERBAN AND HORIA F. POP

Abstract. Metrics have long been studied as a way to assess the quality and complexity of software, and recently this has been applied to object-oriented software as well. However one of their shortcomings is the lack of relevant result interpretation. Related to this, there is an aspect that has a decisive influence on the accuracy of the results obtained: the issue of software metrics threshold values.

In this paper we propose an alternative approach based on fuzzy clustering analysis for the problem of setting up the software metrics threshold values. Measurements are used to evaluate the conformance of an object oriented model to well established design heuristics.

1. Introduction

In time, software systems become very large and complex due to repeated modifications and updates, needed to meet the ever changing requirements of the business. The code becomes more complex and drifts away from its original design. The result is that the system becomes practically unmanageable. A small change in one part of it may have unforeseen effects in completely other parts, leading to potential disasters. In order to prevent this, we need proper quantification means in order to assess the quality of software design during its development lifecycle.

A good object-oriented design needs design rules, principles and practices that must be known and used [11]. In this way, software metrics are very useful being a mean for quantifying these aspects and identifying those design entities that capture deviations from good design principles and heuristics.

Although a large number of metrics have been proposed by researchers to assess object-oriented design quality, they pose some problems of their own, the most important being the ability to give relevant interpretation of the

Received by the editors: October 10, 2008.

2000 *Mathematics Subject Classification.* 68N19, 68T37.

1998 *CR Categories and Descriptors.* D.2.8 [**Software Engineering**]: Metrics – *Performance measures*; I.5.3 [**Pattern recognition**]: Clustering – *Algorithms*.

Key words and phrases. Software quality, Software metrics, Fuzzy clustering.

measurement results which in turn is due to the fact that threshold values for the metrics are difficult to set. This problem is far from being new and characterizes intrinsically any metrics-based approach. A threshold divides the space of a metric value into regions. Depending on the region of the metric value, we may make an informed assessment about the measured entity. For example, if we measure the reusability of a design entity with possible values in the [0..1] range and we define 0.7 as being the threshold with good reusability, then all measured components whose reusability values are above that threshold may be quantified as being reusable. This simple example raises a set of questions: how did we come up with a threshold of 0.7 in the first place? Why not 0.5? And, is a component with a reusability value of 0.68 not reusable compared to a component having a reusability value of 0.7? Would such a threshold still be meaningful in a population where the largest reusability value is 0.5?

As a conclusion, the accuracy of the results obtained is questionable. In order to overcome this limitation, we propose an alternative approach for the problem of setting up the software metrics threshold values using fuzzy clustering analysis. This allows us to place an object in more than one group, with different membership degrees.

The remainder of this paper is organized as follows. Section 2 describes the theoretical background for an object-oriented design quality assessment system while Section 3 presents the fuzzy-clustering approach used in the quality evaluation of a system design. Section 4 describes in details our proposed approach for detecting design flaws in an object-oriented system. Section 6 presents and discusses the experimental results obtained by applying the proposed approach on an open source application, called *log4net* [3]. Section 7 reviews related works in the area of detection design flaws. Finally, Section 8 summarizes the contributions of this work and outlines directions for further research.

2. Theoretical framework

Object oriented design quality evaluation implies identification of those design entities that are relevant for the analysis of their properties and of the relationships that exist between them and the software metrics that best emphasize the aspects (design principle/heuristics) that we want to quantify. So, our theoretical framework, consists of three groups of elements:

- a meta-model for the object-oriented systems;
- design principles/heuristics;
- relevant suites of software metrics.

Thus, our object oriented design quality evaluation system will be associated with a 3-tuple, $ES = (MModel, Aspects, Metrics)$. In what follows, all the above mentioned elements will be briefly described.

2.1. **A meta-model for object-oriented systems.** A meta-model for object-oriented systems consists of design entities together with their properties and the relations between them [11]. Thus, a meta-model is a 3-tuple $MModel = (E, P, R)$ where,

- $E = \{E_1, E_2, ..., E_n\}$, represents the set of *design entities* of the software system, E_i, $1 \le i \le n$ may be a class, a method from a class, an attribute from a class, a parameter from a method or a local variable declared in the implementation of a method. We also will consider that:

- $Class(E) = \{C_1, C_2, ..., C_l\}$, $Class\,(E) \subset E$ is a set of entities that are classes;
- Each class, C_i, $1 \le i \le l$ has a set of methods and attributes, i.e. $C_i = \{m_{i1}, m_{i2}, ..., m_{ip_i}, a_{i1}, a_{i2}, ..., a_{ir_i}\}$, $1 \le p_i \le n$, $1 \le r_i \le n$, where $m_{ij} (\forall j, 1 \le j \le p_i)$ are methods and a_{ik} $(\forall k, 1 \le j \le r_i)$ are attributes from C_i;
- $Meth(E) = \bigcup\limits_{i=1}^{l} \bigcup\limits_{j=1}^{r_i} m_{ij}, Meth(E) \subset E$, is a set of methods from all classes of the software system;
- Each method m_{ij}, $1 \le i \le l$, $1 \le j \le p_i$, has a set of parameters and local variables, i.e., $m_{ij} = \{p_{ij1}, p_{ij2}, ..., p_{ijp_{ij}}, v_{ij1}, v_{ij2}, ..., v_{ijv_{ij}}\}$ $1 \le p_{ij} \le n$, $\quad 1 \le v_{ij} \le n$, where $p_{ijk} (\forall k,\ 1 \le k \le p_{ij})$ are parameters and $v_{ijs} (\forall s,\ 1 \le s \le v_{ij})$ are local variables;
- $Param(E) = \bigcup\limits_{i=1}^{l} \bigcup\limits_{j=1}^{r_i} \bigcup\limits_{k=1}^{p_{ij}} p_{ijk}, Param(E) \subset E$;
- $LocVar(E) = \bigcup\limits_{i=1}^{l} \bigcup\limits_{j=1}^{r_i} \bigcup\limits_{s=1}^{v_{ij}} v_{ijs}, LocVar(E) \subset E$;
- $Attr(E) = \bigcup\limits_{i=1}^{l} \bigcup\limits_{j=1}^{r_i} a_{ij}, Attr(E) \subset E$, is the set of attributes from all classes of the software system.

- P represents the set of *properties* of the aforementioned design entities, $P = ClassP \bigcup MethP \bigcup\ AttrP \bigcup ParamP \bigcup LocVarP$. Where,

- $ClassP$ represents the properties of all classes in E (e.g. abstraction, visibility, reusability);
- $MethP$ represents the properties of all methods in E (e.g. visibility, kind, instantiation, reuse, abstraction, binding);
- $AttrP$ represents the properties of all attributes in E (e.g. visibility);

- $ParamP$ represents the properties of all parameters in E (e.g. type, aggregation);
- $LocVarP$ represents the properties of all local variables in E (e.g. type, aggregation);

- R represents the set of *relations* between the entities of the set E. These relations are described in detail in [11].

2.2. **Design principles and heuristics.** The main purpose of our evaluation is to identify those design entities that capture deviations from good design principles and heuristics. Object-oriented design principles are mostly extensions of general design principles in software systems (*e.g.*, abstraction, modularity, information hiding). Samples of principles for good design in software systems are: *high coupling*, *low cohesion*, *manageable complexity*, *proper data abstraction*. Design heuristics [17] are stated as the rules of thumb or guidelines for good design. These rules are based on design principles and their ultimate goal is to improve quality factors of the system and avoid occurrence of design flaws. These rules recommend designers and developers to "do" or "do not" specific actions or designs. A sample of such heuristics is "minimize the number of messages in a class".

A literature survey showed a constant and important preoccupation for this issue: several authors were concerned with identifying and formulating design principles [14, 12] and heuristics [17, 9]. Riel [17] presents a set of heuristic design guidelines and discusses some of the flawed structures that result if these guidelines are violated. In the recent years, we found various forms of descriptions for bad or flawed design in the literature such as *bad-smells* [7]. In the same manner, Martin [12] discusses the main design principles of object-orientation and shows that their violation leads to a *rotting design*.

2.3. **A catalog of design metrics.** As we mentioned earlier, the quantification of object-oriented design principle needs a relevant metrics catalog. Thus, the third element of the proposed framework is the set of design metrics. These metrics have to be selected based on the definitions and classification rules of each design principle/heuristics. We do not intend to offer an exhaustive list of design metrics in this section, but to emphasize their relevance in quantifying some rules related to good object oriented design.

Thus, in the following we make a short survey of the most important object-oriented metrics defined in the literature. These metrics capture characteristics that are essential to object-orientation including *coupling, complexity and cohesion.*

Coupling Metrics. We selected Coupling Between Objects(CBO) [6] as the primitive metric for coupling. CBO provides the number of classes to which a given class is coupled by using their member functions and/or instance variables. Other metrics related with CBO are Fan - Out [19], Data Abstraction Coupling(DAC) [1] and Access To Foreign Data(ATFD) [11]. A second way of measuring coupling is: when two classes collaborate, count the number of distinct services accessed (the number of distinct remote methods invoked). One measure that counts the number of remote methods is RFC (Response For A Class)[6]. Another important aspect that has to be taken into account when measuring coupling is the access of a remote method from different parts of the client class, each access being counted once. This is the approach taken by Li and Henry in defining the Message Passing Coupling(MPC) metric, which is the number of send statements defined in a class [1] (also proposed in [10]). A similar type of definition is used by Rajaraman and Lyu in defining coupling at the method level. Their method coupling MC measure [16] is defined as the number of non-local references in a method.

Cohesion Metric. LCOM (Lack of Cohesion in Methods) [6] is not a significant cohesion indicator as discussed in [8, 5]. In [5] the authors propose two cohesion measures that are sensitive to small changes in order to evaluate the relationship between cohesion and reuse. The two measures are TCC (Tight Class Cohesion) and LCC (Loose Class Cohesion) TCC is defined as the relative number of directly connected methods. Two methods are directly connected if they access a common instance variable of the class. TCC refers the relative number of directly connected methods in a given class. LCC is the relative number of directly or indirectly connected methods. Two methods are considered to be indirectly connected if they access a common instance variable through the invocation of other methods.

Complexity Metric. In order to measure the structural complexity for a class, instead of counting the number of methods, the complexities of all methods must be added together. This is measured by WMC (Weighted Method per Class) metric [6]. WMC is the sum of the complexity of all methods for a class, where each method is weighted by its cyclomatic complexity. The number of methods and the complexity of the methods involved is a predictor of how much time and effort is required to develop and maintain the class.

Several studies have been conducted to validate these metrics and have shown that they are useful quality indicators [20].

After computing the metrics values, the next step is to give a relevant interpretation of the obtained measurements results. Following a classical approach we have to set threshholds values for metrics that we use. As we mentioned before, the problem of setting up the threshholds is not simple and the accuracy of the results obtained is questionable. In order to overcome this limitation, we propose an alternative approach based on fuzzy clustering analysis for the problem of setting up the software metrics threshold values. Thus, an object may be placed in more that one group, having different membership degree.

3. Fuzzy clustering analysis

Clustering is the division of data set into subsets (clusters) such that, similar objects belong to the same cluster and dissimilar objects to different clusters. Many concepts found in real world do not have a precise membership criterion, and thus there is no obvious boundary between clusters. In this case fuzzy clustering is often better, as objects belong to more that one cluster with different membership degrees.

Fuzzy clustering algorithms are based on the notion of fuzzy set that was introduced in 1965 by Lotfi A. Zadeh [21] as a natural generalization of the classical set concept. Let X be a data set composed of n data items. A fuzzy set on X is a mapping $A : X \rightarrow [0, 1]$. The value $A(x)$ represents the membership degree of the data item $x \in X$ to the class A. Fuzzy clustering algorithms partition the data set into overlapping groups based on similarity amongst patterns.

3.1. **Fuzzy Clustering Analysis – formalization.**

Let $X = \{O_1, O_2, ..., O_n\}$ be the set of n objects to be clustered. Using the vector space model, each object is measured with respect to a set of m initial attributes $A_1, A_2, ..., A_m$ (a set of relevant characteristics of the analyzed objects) and is therefore described by a m-dimensional vector $O_i = (O_{i1}, O_{i2}, ..., O_{im})$, $O_{ik} \in \Re$, $1 \leq i \leq n$; $1 \leq k \leq m$;

Our aim is to find a fuzzy partition matrix $U = (C_1, C_2, ..., C_c)$, $C_i = (u_{i1}, u_{i2}, ..., u_{in})$, $1 \leq i \leq c$, that best represents the cluster substructure of the data set X., i.e. objects of the same class should be as similar as possible, and objects of different classes should be as dissimilar as possible. The fuzzy partition matrix, U has to satify the following constraints:

- *membership degree:* $u_{ik} \in [0..1]$, $1 \leq i \leq c$, $1 \leq k \leq n$, u_{ik} represents the membership degree of the data object O_k to cluster i;
- *total membership:* the sum of each column of U is constrained to the value $1(\sum\limits_{i=1}^{c} u_{ik} = 1)$.

The fuzzy clustering generic algorithm, named Fuzzy c-means clustering, is described in [4]. This algorithm has the drawback that the optimal number of classes corresponding to the cluster substructure of the data set, is a data entry. As a result in this direction, hierarchical clustering algorithms, produce not only the optimal number of classes (based on the needed granularity), but also a binary hierarchy that show the existing relationships between the classes. In this paper we use the Fuzzy Divisive Hierarchic Clustering algorithm (FDHC) [22].

4. Our Approach

The main objective of this paper is to use fuzzy clustering technique in order to offer an alternative solution to the problem of setting up the software metrics thresholds values, metrics applied for object-oriented design quality investigation. In other words, we aim at identification of those design entities that violate a specified design principle, heuristics or rule. These entities are affected by some design flaw. Thus, our problem can be reduced at identification of those design flaws that violate a specified design principle or heuristic. In fact, design flaws are violations of these heuristics/principles.

Let us consider the theoretical framework proposed in Section 3. In addition, we adopt the following notations:

- DP denotes the set of design principles, heuristics or rules that we want to quantify;
- DF denotes the set of design flaws that violate the entitites from DP;
- $R \subseteq DP \times DF$, the associations set between DP and DF;

Definition 1. *The 3-tuple $GPF = (DP, DF, R)$ is a bipartite graph, called principles-design flaws.*

For each element from the DP or DF set we have to identify a set of relevant metrics. The set of all these metrics will be denoted by M. Let also consider R_1 to be the set of associations between the entities from DP and their corresponding metrics from M and R_2 to be the set of associations between the entities from DF and their coresponding metrics from M.

Definition 2. *The 3-tuple $GPM = (DP, M, R_1)$ is a bipartite graph, called principle metrics.*

Definition 3. *The 3-tuple $GFM = (DF, M, R_2)$ is a bipartite graph, called flaw metrics.*

With these considerations our problem stated in Section 2 can be rephrased as follows: given an element, p, from DP or DF set, its associated metrics set M_p and a subset of design entities from E, we have to identify (using a fuzzy clustering approach) those design entities that capture deviations from a specified principle/heuristic or are affected by a specified design flaw. In this way, for each entity implied in the evaluation, we obtain a set of metrics values.

We may apply now the FDHC algorithm referred in Section 3. The design entities implied in the evaluation correspond to objects from the fuzzy clustering algorithm and the metrics values to the attributes of these objects.

After applying this algorithm each assessed entity is placed into a cluster having a membership degree. This approach offers a better interpretation of measurements results than the thresholds values-based interpretation.

5. Case study

In order to validate our approach we have used the following case study. The object oriented system proposed for evaluation is *log4net* [3], an open source application. It consists of 214 classes. The elements of the meta-model defined in Section 2.1 (design entities, their properties and the relations between them) ware identified using our own dveloped tool.

The objective of this case-study is to identify those entities affected by "God Class" [17] design flaws. So, the objects considered for fuzzy clustering algorithm are classes from the analyzed system.

The first step in this evaluation is to construct (from the graph principles-design flaws defined in Section 5) the subgraph that contains the node "God Class" and its related "heuristics/rules". As it is known, an instance of a god-class performs most of the work, delegating only minor details to a set of trivial classes and using the data from other classes. This has a negative impact on the reusability and the understandability of that part of the system. This design problem may be partially assimilated with Fowlers Large Class bad-smell. In this case we will start from a set of two heuristics found in Riels book [17]:

- Distribute system intelligence horizontally as uniformly as possible;
- Beware of classes with much non-communicative behavior.

The second step is to select proper metrics that best quantify each of the identified heuristics/rules. This means identifying the subgraph obtained by keeping the nodes corresponding with these heuristics and their corresponding metrics that we want to take into account.

In our case the first rule refers to a uniform distribution of intelligence among classes, and thus it refers to *high class complexity.* The second rule speaks about the level of intraclass communication; thus it refers to the *low cohesion* of classes. Therefore, we chose the following metrics:

- Weighted Method per Class (WMC) is the sum of the statical complexity of all methods in a class [6]. We considered the McCabes cyclomatic complexity as a complexity measure [13].
- Tight Class Cohesion (TCC) is the relative number of directly connected methods [5].
- Access to Foreign Data (ATFD) represents the number of external classes from which a given class accesses attributes, directly or via accessor-methods [11]. The higher the ATFD value for a class, the

higher the probability that the class is or is about to become a god-class.

As a remark, a possible suspect of "God Class" will have high values for the WMC and ATFD metrics and low values for the TCC metric.

Taking into account the metrics mentioned above each class from our system, c_i, will be identified by a vector of three elements, $c_i = (m_1, m_2, m_3)$, corresponding to the metrics values applied for class c_i.

The next step is to apply the FDHC algorithm described in Section 3. The objects from the algorithm are classes from our system and the features are the computed values of the metrics corresponding to these classes. The classification tree and the final binary partition produced by FDHC algorithm are represented in Figure 1. By interpreting the results obtained we may conclude that the algorithm has identified a list of suspects, those from class 1 and a list of the objects that do not need further investigation, class 2 of objects. The list of suspects from class 1 are further partitioned according to the values of the three metrics. For example, in class 1.1.1.1.1.1. the list of suspects have the value 0 of the TCC and ATFD metrics and low value for the WMC metric.

Due to space restrictions, we include in this paper only a subset of objects, containing a list of suspects. These objects are described in Figure 2. All other numerical data are available from the authors by request.

6. Related work

During the past years, various approaches have been developed to address the problem of detecting and correcting design flaws in an object-oriented software system using metrics. Marinescu [11] defined a list of metric-based detection strategies for capturing around ten flaws of object-oriented design at method, class and subsystem levels as well as patterns. However, how to choose proper threshold values for metrics and propose design alternatives to correct the detected flaws are not addressed in his research.

Mihancea et al. [15] presented an approach to establish proper threshold values for metrics-based design flaw detection mechanism. This approach, called tuning machine, is based on inferring the threshold values based on a set of reference examples, manually classified in flawed, respectively good design entities.

Trifu [18] introduced correction strategies based on the existing flaw detection and transformation techniques. This approach serves as reference descriptions that enable a human-assisted tool to plan and perform all necessary steps for the removal of detected flaws. Consequently, it is a methodology that can be fully supported.

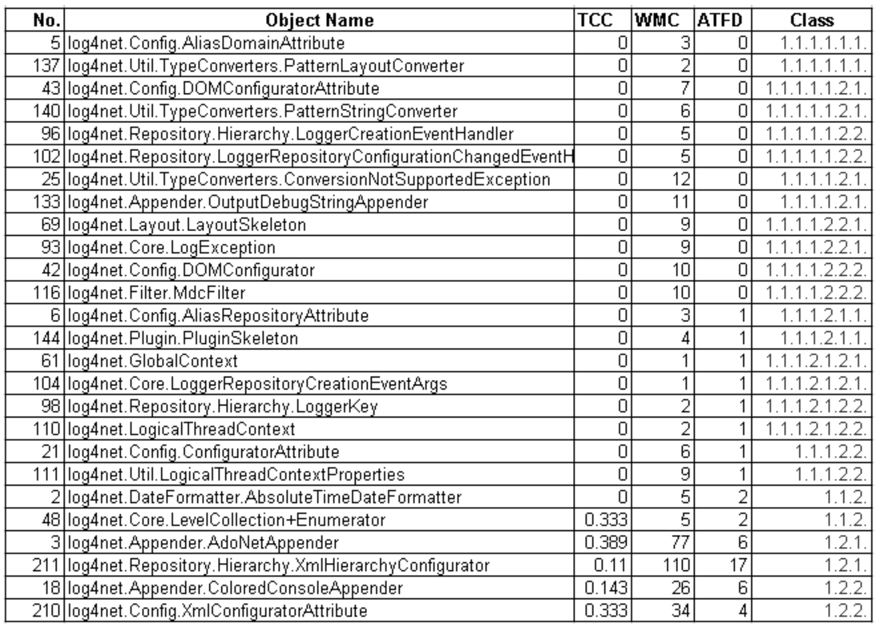
Class	Members
1.1.1.1.1.1.	1 5 15 24 35 46 137 155 183 198
1.1.1.1.1.2.1.	43 140 151
1.1.1.1.1.2.2.	41 96 102 103 105 106 207
1.1.1.1.2.1.	25 97 133 173
1.1.1.1.2.2.1.	69 86 91 93 152
1.1.1.1.2.2.2.	42 52 116 124
1.1.1.2.1.1.	6 26 27 57 71 83 84 115 129 144 166 170 199
1.1.1.2.1.2.1.	61 95 104 108
1.1.1.2.1.2.2.	98 110 191
1.1.1.2.2.	21 44 58 89 111 122 128 164 171
1.1.2.	2 14 28 45 47 48 49 66 88 113 120 121 123 136 146 160 161 162 178 187 192 193 194 197 201 206 208 213
1.2.1.	3 11 12 16 38 55 63 72 94 99 107 109 112 138 142 172 179 186 209 211
1.2.2.	18 19 37 77 87 114 1 26 132 134 135 139 163 167 168 169 180 190 210
2.	4 7 8 9 10 13 17 20 22 23 29 30 31 32 33 4 36 39 40 50 51 53 54 56 59 60 62 64 65 67 68 70 73 74 75 76 78 79 80 81 82 85 90 92 100 101 117 118 119 125 127 130 131 141 143 145 147 148 149 150 153 154 156 157 158 159 165 174 175 176 177 181 182 184 185 188 189 195 196 200 202 203 204 205 212 214

FIGURE 1. Classification tree and final partition for the set of 214 objects

M. Frenţiu and H.F.Pop [2] presented an approach based on fuzzy clustering to study dependencies between software attributes, using the projects written by second year students as a requirement in their curriculum. They have observed that there is a strong dependency between almost all considered attributes.

7. Conclusions and Future Work

We have presented in this paper a new approach that address the issue of setting up the software metrics threshold values, approach based on fuzzy clustering techniques. In order to validate our approach we have used a case study, presented in Section 5. Further work can be done in the following directions:



No.	Object Name	TCC	WMC	ATFD	Class
5	log4net.Config.AliasDomainAttribute	0	3	0	1.1.1.1.1.1.
137	log4net.Util.TypeConverters.PatternLayoutConverter	0	2	0	1.1.1.1.1.1.
43	log4net.Config.DOMConfiguratorAttribute	0	7	0	1.1.1.1.1.2.1.
140	log4net.Util.TypeConverters.PatternStringConverter	0	6	0	1.1.1.1.1.2.1.
96	log4net.Repository.Hierarchy.LoggerCreationEventHandler	0	5	0	1.1.1.1.1.2.2.
102	log4net.Repository.LoggerRepositoryConfigurationChangedEventH	0	5	0	1.1.1.1.1.2.2.
25	log4net.Util.TypeConverters.ConversionNotSupportedException	0	12	0	1.1.1.1.2.1.
133	log4net.Appender.OutputDebugStringAppender	0	11	0	1.1.1.1.2.1.
69	log4net.Layout.LayoutSkeleton	0	9	0	1.1.1.1.2.2.1.
93	log4net.Core.LogException	0	9	0	1.1.1.1.2.2.1.
42	log4net.Config.DOMConfigurator	0	10	0	1.1.1.1.2.2.2.
116	log4net.Filter.MdcFilter	0	10	0	1.1.1.1.2.2.2.
6	log4net.Config.AliasRepositoryAttribute	0	3	1	1.1.1.2.1.1.
144	log4net.Plugin.PluginSkeleton	0	4	1	1.1.1.2.1.1.
61	log4net.GlobalContext	0	1	1	1.1.1.2.1.2.1.
104	log4net.Core.LoggerRepositoryCreationEventArgs	0	1	1	1.1.1.2.1.2.1.
98	log4net.Repository.Hierarchy.LoggerKey	0	2	1	1.1.1.2.1.2.2.
110	log4net.LogicalThreadContext	0	2	1	1.1.1.2.1.2.2.
21	log4net.Config.ConfiguratorAttribute	0	6	1	1.1.1.2.2.
111	log4net.Util.LogicalThreadContextProperties	0	9	1	1.1.1.2.2.
2	log4net.DateFormatter.AbsoluteTimeDateFormatter	0	5	2	1.1.2.
48	log4net.Core.LevelCollection+Enumerator	0.333	5	2	1.1.2.
3	log4net.Appender.AdoNetAppender	0.389	77	6	1.2.1.
211	log4net.Repository.Hierarchy.XmlHierarchyConfigurator	0.11	110	17	1.2.1.
18	log4net.Appender.ColoredConsoleAppender	0.143	26	6	1.2.2.
210	log4net.Config.XmlConfiguratorAttribute	0.333	34	4	1.2.2.

FIGURE 2. A list of "God Class" design flaw suspects

(1) To apply this approach for more case studies;
(2) Comparison with others approaches regarding the issue of threshold values;
(3) To develop a tool that emphasizes the approach presented in this paper.

8. ACKNOWLEDGEMENT

This research has been supported by CNCSIS - the Romanian National University Research Council, through the PNII-IDEI research grant ID_550/2007.

REFERENCES

[1] W. Li and S. Henry. Maintenance Metrics for the Object Oriented Paradigm. IEEE Proc. First International Software Metrics Symp., pages 5260, may 1993.
[2] M. Frentiu and H.F. Pop. A study of dependence of software attributes using data analisys techniques. Studia Univ. Babes-Bolyai, Series Informatica, 2 (2002), 53–66.
[3] Project log4net.: http://logging.apache.org/log4net.
[4] Bezdek, J.: Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum Press, New York, 1981.
[5] Bieman, J. and Kang, B.: Cohesion and reuse in an object-oriented system. Proc. ACM Symposium on Software Reusability, apr (1995).
[6] Chidamber, S. and Kemerer, C.: A metric suite for object- oriented design. IEEE Transactions on Software Engineering, 20(6):476–493, June (1994).

[7] Fowler, M. Beck, K. Brant, J. Opdyke, W. and Roberts, D.: Refactoring: Improving the Design of Existing Code. Addison-Wesley, (1999).
[8] Henderson-Sellers, B.: Object-Oriented Metrics: Measures of Complexity. Prentice-Hall, (1996).
[9] Johnson, R. and Foote, B.: Designing reuseable classes. Journal of Object-Oriented Programming, 1(2):22–35, June (1988).
[10] Lorenz, M. and Kidd, J.: Object-Oriented Software Metrics. Prentice-Hall Object-Oriented Series, Englewood Cliffs, NY, (1994).
[11] R. Marinescu, Measurement and quality in object-oriented design. Ph.D. thesis in the Faculty of Automatics and Computer Science of the Politehnica University of Timisoara, 2003.
[12] R. Martin, Design Principles and Patterns. Object Mentor,http://www.objectmentor.com, 2000
[13] McCabe, T.: A complexity measure. IEEE Transactions on Software Engineering, 2(4):308–320, dec (1976).
[14] Meyer, B.: Object-Oriented Software Construction. International Series in Computer Science. Prentice Hall, Englewood Cliffs, (1988).
[15] P.F. Mihancea and R.Marinescu. Towards the optimization of automatic detection of design flaws in object-oriented software systems. In Proc. of the 9th European Conf. on Software Maintenance and Reengineering (CSMR), 92–101, (2005).
[16] Rajaraman, C. and Lyu, M.: Some coupling measures for c++ programs. Prentice-Hall Object-Oriented Series, In Proceedings of TOOLS USA92, Prentice-Hall, Englewood Cliffs, NJ, (1992).
[17] Riel, A.J.: Object-Oriented Design Heuristics. Addison-Wesley, (1996).
[18] Tahvildari, L. and Kontogiannis, K.: Improving design quality using meta-pattern transformations : A metric-based approach. Journal of Software Maintenance and Evolution : Research and Practice, 4-5(16):331–361, October (2004).
[19] D.Tegarden and S.Sheetz: Object-oriented system complexity: an integrated model of structure and perceptions. In OOPSLA92 Workshop on Metrics for Object-Oriented Software Development(Washington DC), (1992).
[20] Basili, V., Briand, L., and Melo, W.: A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering 22(10), 751-761, (1996).
[21] Zadeh L. A.: Fuzzy sets, Inf. Control, 8, 338–353, (1965).
[22] Dumitrescu, D.: Hierarchical pattern classification, Fuzzy Sets and Systems 28, 145–162, (1988).

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: camelia@cs.ubbcluj.ro, hfpop@cs.ubbcluj.ro