STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 1, 2017
DOI: 10.24193/subbi.2017.1.05

PRELIMINARY MEASUREMENTS IN IDENTIFYING DESIGN FLAWS

CAMELIA ŞERBAN, ANDREEA VESCAN, AND HORIA F. POP

Abstract. Software metrics are of great importance in object-oriented design assessment. They quantify various aspects of design entities and play an important role in predicting design quality. Despite the fact that software metrics have become increasingly useful, they raise several issues. Among them, relevant to our research are the issue of setting threshold values and the problem of measurement results interpretation. Fuzzy clustering analysis is used to overcome the limitations of the existing approaches that are using threshold values for metrics and to provide a better interpretation of the obtained measurement results.

This paper focuses on metrics-based design flaw detection in object-oriented design. A new metric, Design Flaw Entropy which measures the distribution of design flaws into the analyzed system is introduced. To validate the proposed approach, a case study was also proposed.

1. INTRODUCTION

Over an extended period of time software systems are often subject to a process of evolution, applications becoming very large and complex. They undergo repeated modifications in order to satisfy any requirement regarding a business change. The result is that the code deviates from its original design and the system becomes unmanageable. A minor change in one of its parts may have unpredictable effects in completely other parts [1]. To avoid such risk a high quality design should be preserved throughout the system life cycle. This can be achieved by repeatedly assessing the system design, aiming to identify in due course those design entities that do not comply with the rules, principles and practices of a good design, and suggesting possible refactorings or improvements to be performed.

Received by the editors: April 25, 2017.

2000 *Mathematics Subject Classification.* 68N30, 68T37.

1998 *CR Categories and Descriptors.* code D.2.8 [**Software Enginnering**]: *Metrics – Product metrics*; code D.1.5 [**Pattern recognition**]: *Clustering – Algorithms*.

Key words and phrases. Software metrics, object oriented design, fuzzy clustering.

As a consequence of the above detailing, our previous work [10] was focused on developing a methodology for quantitative evaluation of object-oriented design. The proposed methodology is based on static analysis of the source code and is described by a framework of four abstraction layers. A new method for measurements results interpretation, based on fuzzy clustering technique, is also contained in this framework.

The above mentioned proposed methodology for design assessment is completed in this paper with a new metric, Design Flaw Entropy (DFE), which offers preliminary measurements in identifying those parts of the system design that suffers from degradation. In other words, the DFE metric measures the dispersion or the distribution of a specified design flaw among the analyzed design entities. The proposed metric is based on fuzzy clustering analysis method which aims to overcome the limitations of existing approaches that use thresholds values for metrics used.

The rest of the paper is organized as follows. The description and relevance of the problem of object-oriented design assessment briefly emphasizing the main layers of the above mentioned framework is presented in Section 2. The Design Flaw Entropy metrics is introduced in Section 3, by giving the definition, representative examples and the properties. To validate this metric in Section 4 a case study that aims to identify those classes from a software system that are affected by "God Class" design flaw is presented. Section 5 presents some metric-based related approaches for solving the object-oriented assessment problem and also discuses their limitations. Finally, Section 6 summarizes the contributions of this work and outlines directions for further research.

2. OBJECT-ORIENTED DESIGN ASSESSMENT PROCESS

In this section we aim at presenting the problem of object-oriented assessment and its relevance, as well as the main steps of the software assessment process.

The main steps needed to be performed in any software assessment process require a clear specification of *entities that are evaluated* (the assessment domain) and of *the assessment objectives*, as well as the identification of *methods* and *techniques* that offer a *relevant interpretation of the assessment results obtained* (a thorny issue, insufficiently explored so far in the literature).

All the above mentioned elements are described in a formal manner in our previous proposed methodology for object-oriented design assessment [10], defining the contextual background for the proposed metric, metric that completes our previous work and adds more relevance to the interpretation of the assessment results.

Therefore, in what follows we briefly describe these steps of the software assessment process:

(1) Domain Assessment identification.
(2) Setting the assessment objectives.
(3) Computing the metrics values. Fuzzy partition determination. Assessment results analysis.

2.1. **Domain Assessment identification.** The proposed methodology for a quantitative assessment of object oriented design [10] uses static analysis of the source code. Therefore, the assessed domain should capture only those elements that define the structure of an object-oriented system: the *design entities* (e.g. classes, packages) that are relevant for the analysis, together with their *properties* (e.g. the visibility level of attributes) and the *relationships* (e.g. methods access attributes) that exist between them. Marinescu [1] gathers these elements into a model for object-oriented design.

Our previous work [10] has defined in a formal manner, using terms of algebraic sets and relations, the above mentioned elements, introducing a new background used to formally define metrics and to establish the assessment objectives.

In what follows, the 3-tuple:

$$D(S) = (E, Prop(E), Rel(E))$$

represents the assessment domain corresponding to a software system S, where: E represents the design entities set of S; $Prop(E)$ defines the properties of the elements from E, and $Rel(E)$ represents the relations between the design entities of the set E.

2.2. **Setting the assessment objectives.** The main objective of an object-oriented software assessment is that of verifying whether the built system meets quality factors such as *maintainability, extensibility, scalability and reusability.* Fenton's axiom [15] states that good internal structure should provide good external quality. Consequently, the assessment objectives are reduced to verifying if there is conformity between the software system internal structure and the principles and heuristics of good design. According to Marinescu [1] these principles and heuristics of good design are related with the internal quality attributes such as coupling, cohesion, complexity and data abstraction.

A design feature that indicates deviations from good design principles is named "design flaw" [1]. In recent years, the literature displays various forms of descriptions for bad or flawed design such as bad-smells [12]. The community of researchers [1, 12, 14, 5] was interested in setting a relation between the principles of good design with the design flaws. They wanted to seek what

the violated principles or rules were for a certain design flaw or vice versa, what were the design flaws that could propagate in code if a design principle was violated. These design flaws or principles are than correlated with metrics to quantify these aspects and to automate the assessment process. According to these, the assessment objectives are reduced at identifying a list of design entities, called "suspect" which are affected by a specified design flaw. More detailed, being given a list of design entities AE_p that are evaluated with respect to a specified design flaw p, we have to establish its corresponding design principles and for each of these principles some relevant software metrics M_p. Based on the metrics values, computed on a given design entities set, we aim to identify the suspect entities.

2.3. **Computing the metrics values. Fuzzy partition determination.** After establishing the assessment objectives, the next step is metrics computation. In order to automatically compute these metric and to obtain the fuzzy partition, we have developed a tool, called *Metrics* written in C#. Metrics is divided in five components, Metrics Worker, Parser, Design Entities Model, Metric Definitions and Fuzzy Analysis.

The results of the assessment, done in an automatically manner, are therefore, the values of selected metrics
$M_p = \{m_1, m_2, ..., m_{noM_p}\}$, computed on each design entity from the assessed design entities set AE_p.

To overcome the limitation encountered when a metric based approach is used, that of setting the thresholds for the metrics values, the fuzzy clustering analysis is used. Thus, an entity may be placed in more that one group, having different membership degree, obtaining a fuzzy partition defined as in Definition 1:

Definition 1. *([10]) **Fuzzy partition of the design entities.***
A set $U_{AE_p,M_p} = \{U_1, U_2, ..., U_c\}$ is called a fuzzy partition of the design entities set $AE_p = \{e_1, e_2, ..., e_n\}$, entities characterized by the values of metrics the $M_p = \{m_1, m_2, ..., m_{noM_p}\}$ iff:

- $U_i = (u_{i1}, u_{i2}, ..., u_{in})$, $1 \leq i \leq c$;
- $u_{ij} \in [0..1]$, $1 \leq i \leq c$, $1 \leq j \leq n$, u_{ij} – *represents the membership degree of the design entity e_j to cluster i;*
- $\sum_{i=1}^{c} u_{ij} = 1$, $1 \leq j \leq n$ – *the sum of each column of U is constrained to the value 1.*

If $c = 2$ then U is called a *binary fuzzy partition.*

The fuzzy partition U best represents the cluster substructure of the data set AE_p, i.e. objects of the same class should be as similar as possible (the

difference between any two metrics values of these objects is very close to 0 value), and objects of different classes should be as dissimilar as possible. The measure used for discriminating objects (classes) can be any *metric* or *semimetric* function (d). In our approach we have used the *Euclidian distance* metric.

Fuzzy Divisive Hierarchic Clustering (FDHC) algorithm [11] was applied to determine a fuzzy partition. The FDHC algorithm produce a binary tree hierarchy that provides an in-depth analysis of the data set, by deciding on the optimal number of clusters and the optimal cluster substructure of the data set. The leaves of the binary tree hierarchy determine an optimal fuzzy partition of the assessed entities.

Based on previously obtained optimal fuzzy partition, we have to decide which design clusters contain suspect entities and which of them require further investigation. This decision is influenced by the distribution of entities per clusters (how many entities have dominant membership degree in that cluster), being also the distribution of the analyzed design flaw into the system. It is obvious the fact that a uniform distribution of entities per cluster highlights a difficult re-factorization. For example, if we have a system with sixty design entities divided into two clusters with thirty entities on each of them, it will be hard to make a decision to redesign thirty entities out of sixty. Conversely, if the two clusters contain eight and fifty two entities, it is much easier to take a decision. In the following, we'll introduce a metric to provide us with information on defect design entities distribution. This metric measures also the effort needed in order to restructure the system design. A high value of this metric suggesting that the system is compromised and would require a redesign from scratch.

The next section introduces this metric, discusses some representative examples and identifies its main properties.

3. Design Flaw Entropy Metric

As we have mentioned before, our goal is to define a metric (Design Flaw Entropy - DFE) which could provide an in-depth analysis regarding the distribution of an analyzed design flaw (the degree of its spread into the system) in order to converge through an optimal decision regarding the set of "suspect" design entities.

DFE is defined considering the notion of *entropy* adapted from communication information theory of Shannon [23]. Starting from this concept many researchers [19, 20, 21, 22] have developed new measures for the assessment of software products.

3.1. **Design Flaw Entropy metric definition.** Let us consider a fuzzy partition $U_{AE_p,M_p} = \{U_1, U_2, \ldots, U_c\}$ of design entities $AE_p = \{e_1, e_2, ..., e_n\}$, entities characterized by the values of metrics $M_p = \{m_1, m_2, \ldots, m_{noM_p}\}$, metrics selected in order to quantify a specified design flaw p.

Definition 2. *We say that an entity $e_j \in AE_p$, $1 \leq j \leq n$, have dominant membership degree to cluster U_i, $1 \leq i \leq c$, if $u_{ij} = \max\{u_{rj} | r = \overline{1,c}\}$.*

Definition 3. *The relative frequency of occurrence or the probability of a cluster $U_i \in U_{AE_p,M_p}$, denoted by $p(U_i)$, represents the ratio between the number of entities from AE_p that have dominant membership degree to cluster U_i and the total number of entities from AE_p.*

We will denote by $P_{U_{AE_p,M_p}} = \{p(U_1), p(U_2), \ldots, p(U_c)\}$ the probability distribution per clusters of the partition U_{AE_p,M_p}.

Definition 4. *A measure of the information (self-information) contained in a cluster $U_i \in U_{AE_p,M_p}$ is defined as $I(U_i) = -\log_2 p(U_i)$.*

In the context of the previous definitions and notations, we can now introduce the definition of the proposed metric.

Definition 5. *Design Flaw Entropy (DFE) corresponding to fuzzy partition U_{AE_p,M_p} is defined as the average of the self-information associated to each cluster $U_i \in U_{AE_p,M_p}$. Formally:*

$$DFE : FP(AE_p, M_p) \to [0..\infty],$$

$$DFE(U_{AE_p,M_p}) = \sum_{i=1}^{c} p(U_i) \cdot I(U_i)$$

where $FP(AE_p, M_p)$ is the set of all fuzzy partitions of the design entities set AE_p, entities characterized by the values of metrics $M_p = \{m_1, m_2, \ldots, m_{noM_p}\}$, metrics selected in order to quantify a specified design principle or design flaw p.

The definition of this metric can be shortly described as follows: for each cluster c of the analyzed fuzzy partition, we compute its probability distribution as a ratio between the number of entities that have dominant membership degree to that cluster and the total number of analyzed entities. We also compute a measure of the information (self-information) contained in that cluster that is next used in the definition of the DEF metric: the average of the self-information associated to each cluster.

3.2. **Further analysis of DFE metric. Representative examples.** For a given set of entities, the value of DFE metric depends on the number of clusters of the obtained fuzzy partition and on the entities distribution per clusters. In order to identify the properties of this metric and to better emphasize its meaning, several representative examples are discussed in what follows. We compute the value of DFE metric for nine different fuzzy partitions on a set of sixty design entities. The meaning of these values are also discussed.

The first considered partition $U_1 = \{U_{1,1}\}$ has one cluster ($c = 1$) with the probability distribution per clusters $P_{U_1} = (60/60)$, all design entities having dominant membership degree on the same cluster. The DFE metric value being in this case:

$$DFE(U_1) = -p(U_{1,1}) \cdot \log_2 p(U_{1,1}) = -1 \cdot 0 = 0. \tag{1}$$

The meaning of such a situation is that all entities are equally affected by the analyzed design flaw, a case almost impossible to meet.

The second partition $U_2 = \{U_{2,1}, U_{2,2}\}$ has two clusters ($c = 2$) with the probability distribution per clusters $P_{U_2} = (1/60, 59/60)$, one design entity having dominant membership degree on the first cluster and 59 entities on the second one. The DFE metric value being:

$$\begin{aligned} DFE(U_2) &= -(p(U_{2,1}) \cdot \log_2 p(U_{2,1}) + p(U_{2,2}) \cdot \log_2 p(U_{2,2})) \\ &= -(1/60 \cdot \log_2 1/60 + 59/60 \cdot \log_2 59/60) = 0.12. \end{aligned} \tag{2}$$

In this example one entity need to be reviewed.

The third partition $U_3 = \{U_{3,1}, U_{3,2}\}$ has two clusters ($c = 2$) with the probability distribution per clusters $P_{U_3} = (2/60, 58/60)$, two design entities having dominant membership degree on the first cluster and 58 entities on the second one. The DFE metric value being in this case:

$$\begin{aligned} DFE(U_3) &= -(p(U_{2,1}) \cdot \log_2 p(U_{2,1}) + p(U_{2,2}) \cdot \log_2 p(U_{2,2})) \\ &= -(2/60 \cdot \log_2 2/60 + 58/60 \cdot \log_2 58/60) = 0.21. \end{aligned} \tag{3}$$

The U_3 partition is very similar with U_2, identifying two entities to be reviewed.

The fourth partition $U_4 = \{U_{4,1}, U_{4,2}\}$ has two clusters ($c = 2$) with the probability distribution per clusters $P_{U_4} = (10/60, 50/60)$, 10 design entities having dominant membership degree on the first cluster and 50 entities on the second one. The DFE metric value being in this case:

$$DFE(U_4) = -(10/60 \cdot \log_2 10/60 + 50/60 \cdot \log_2 50/60) = 0.65. \tag{4}$$

Now, we can observe that once the entities distribution per clusters tends to be uniform, the DFE metric value increases. This means that the number of entities that need to be reviewed is higher.

The fifth partition, $U_5 = \{U_{5,1}, U_{5,2}\}$ has two clusters ($c = 2$) with an equiprobable distribution per clusters $P_{U_5} = (30/60, 30/60)$. The DFE metric value being in this case:

$$DFE(U_5) = -(30/60 \cdot \log_2 30/60 + 30/60 \cdot \log_2 30/60) = 1. \tag{5}$$

In such a case the analyzed design flaw is spread on half of the system design or even more, at least fifty percents of design entities are affected.

The sixth partition, $U_6 = \{U_{6,1}, U_{6,2}, U_{6,3}\}$ has three clusters ($c = 3$) with the probability distribution per clusters $P_{U_5} = (1/60, 1/60, 58/60)$, one design entity having dominant membership degree on the first cluster, one have on the second cluster and 58 entities on the third one. The DFE metric value being in this case:

$$\begin{aligned} DFE(U_6) = -(&1/60 \cdot \log_2 1/60 + 1/60 \cdot \log_2 1/60, \\ &58/60 \cdot \log_2 58/60) = 0.45. \end{aligned} \tag{6}$$

A case very similar with the second one, but with three clusters.

The seventh partition, $U_7 = \{U_{7,1}, U_{7,2}, U_{7,3}\}$ has three clusters ($c = 3$) with the probability distribution per clusters $P_{U_7} = (1/60, 2/60, 57/60)$, one design entity having dominant membership degree on the first cluster, two have on the second cluster and 57 entities on the third one. The DFE metric value being in this case:

$$\begin{aligned} DFE(U_7) = -(&1/60 \cdot \log_2 1/60 + 2/60 \cdot \log_2 2/60, \\ &57/60 \cdot \log_2 57/60) = 0.53. \end{aligned} \tag{7}$$

The eighth partition, $U_8 = \{U_{8,1}, U_{8,2}, U_{8,3}\}$ has three clusters ($c = 3$) with the probability distribution per clusters $P_{U_8} = (10/60, 20/60, 30/60)$, 10 design entities having dominant membership degree on the first cluster, 20 have on the second cluster and 30 entities on the third one. The DFE metric value being in this case:

$$\begin{aligned} DFE(U_8) = -(&10/60 \cdot \log_2 10/60 + 20/60 \cdot \log_2 20/60, \\ &30/60 \cdot \log_2 30/60) = 1.45. \end{aligned} \tag{8}$$

Here, we can observe again that, once the probability distribution per clusters tends to be equiprobable (uniformity) the analyzed design flaw is spread almost over the entire system.

TABLE 1. Representative examples of DFE metric computed on 9 partitions of 60 design entities

U_k	c	$P_{U_k} = (p(U_{k,1}), p(U_{k,2}), ..., p(U_{k,c}))$	$DFE(U_k)$
$U_1 = \{U_{1,1}\}$	1	$(60/60)$	0
$U_2 = \{U_{2,1}, U_{2,2}\}$	2	$(1/60, 59/60)$	0.12
$U_3 = \{U_{3,1}, U_{3,2}\}$	2	$(2/60, 58/60)$	0.21
$U_4 = \{U_{4,1}, U_{4,2}\}$	2	$(10/60, 50/60)$	0.65
$U_5 = \{U_{5,1}, U_{5,2}\}$	2	$(30/60, 30/60)$	1
$U_6 = \{U_{6,1}, U_{6,2}, U_{6,3}\}$	3	$(1/60, 1/60, 48/60)$	0.45
$U_7 = \{U_{7,1}, U_{7,2}, U_{7,3}\}$	3	$(1/60, 2/60, 47/60)$	0.53
$U_8 = \{U_{8,1}, U_{8,2}, U_{8,3}\}$	3	$(10/60, 20/60, 30/60)$	1.45
$U_9 = \{U_{9,1}, U_{9,2}, U_{9,3}\}$	3	$(20/60, 20/60, 20/60)$	1.58

The ninth partition $U_9 = \{U_{9,1}, U_{9,2}, U_{9,3}\}$ has three clusters, $(c = 3)$ with an equiprobable distribution per clusters $P_{U_9} = (20/60, 20/60, 20/60)$. The DFE metric value being in this case:

$$DFE(U_9) = -(20/60 \cdot \log_2 20/60 + 20/60 \cdot \log_2 20/60, \\ 20/60 \cdot \log_2 20/60) = 1.58. \tag{9}$$

Having all these representative examples into account, we can identify the properties of the proposed metric. They are very important in order to draw a conclusion regarding the meaning of DFE metric value. The next section describes these properties.

3.3. **Properties of the Design Flaw Entropy metric.** Analyzing the information contained in Table 1 (that contains the above presented representative examples) we can conclude the following properties of DFE metric:

(1) $DFE(U_{AE_p,M_p}) = 0 \Leftrightarrow c = 1$, $\forall U_{AE_p,M_p} \in FP(U_{AE_p,M_p})$; This property states that design flaw entropy is zero if and only if all entities are placed on the same cluster. In this case the variety of the analyzed design flaw p is *minimal*, meaning that all entities are affected on the same measure by the analyzed design flaw.

On the other hand, the possible *maximum entropy* occurs when each entity is placed in a separate cluster. In such a case the number of clusters equals to the number of entities:

$$DFE(U_{AE,M}) = -\log_2 1/n \Leftrightarrow c = n \wedge p(U_i) = 1/n.$$

(2) Let us consider the set of all partitions from $FP(AE_p, M_p)$ which have the number of clusters equals to c,

$$FP_c(AE_p, M_p) = \{U_{AE_p,M_p} \in FP(AE_p, M_p)$$

$$|U_{AE_p,M_p} = \{U_1, U_2, ..., U_c\}\}.$$

In this case two values are important to be discussed: the *minimum* and the *maximum* values of DFE metric:

- A *minimum value* of DFE metric is attained for the following distribution of entities per clusters: $(1/n, ..., 1/n, (n-c)/n)$, where n is the number of entities. In what follows we denote these values by $MinDFE(n, c)$.
- On the other extreme, as the probabilities associated with each cluster will have values closer together, the entropy will have a higher value. At the limit, DFE reaches a *maximum value* for an equiprobable distribution of elements per clusters: $U_{AE_p,M_p} = \{U_1, U_2, ..., U_c\}$, $p(U_i) = \frac{1}{c}, 1 \leq i \leq c$ we have:
  $DFE(V_{AE_p,M_p}) \leq DFE(U_{AE_p,M_p})$,
  $(\forall)V_{AE_p,M_p} \in FP(AE_p, M_p)$,
  $V_{AE_p,M_p} = \{V_1, V_2, ..., V_c\}$ we denote these values by $Max\ DFE(n, c)$.
  In this case, the higher the entropy becomes, the more difficult is to identify those design fragments that need to be reviewed.

(3) For an equiprobable distribution of elements per clusters, once the number of clusters increases, the entropy will have a higher value:
$(\forall)\ U_{AE_p,M_p}, V_{AE_p,M_p} \in FP(AE_p, M_p)$, $U_{AE_p,M_p} = \{U_1, U_2, ..., U_c\}$, $V_{AE_p,M_p} = \{V_1, V_2, ..., V_c, V_{c+1}\}$ such that $p(U_i) = \frac{1}{c}, p(V_j) = \frac{1}{c+1}, (1 \leq i \leq c), (1 \leq j \leq c+1)$ we have:

$$DFE(U_{AE,M}) \leq DFE(V_{AE,M}).$$

4. Experimental evaluation

In this section we present an empirical validation of our proposed metric. This validation is based on a case study which aims to evaluate the design of an open source object-oriented software system, called log4net [13]. It consists of 214 classes grouped in 10 packages.

4.1. **God Class suspect identification - a fuzzy based approach.** The objective of the proposed assessment is to identify those design entities affected by "God Class" [12] design flaw. An instance of God Class does most of the operation tasks, leaving only minor details to a series of trivial classes; it

TABLE 2. The distribution per clusters of the class design entities

Cluster	No. of members with dominant membership degree
1.1.1	19
1.1.2	23
1.2.1	42
1.2.2	106
2.1	14
2.2	10
Total number of entities = 214	

also uses the data from other classes. Briefly, *God Class* design flaw refers to those classes "which tend to centralize the intelligence of the system" [1]. As a consequence, the principle of manageable complexity is violated, as god classes tend to capture more than one abstraction. Another shortcoming of these pathological classes is their tendency towards non-cohesion. If we consider the quality attributes, god classes also have a negative impact on the reusability and understandability of that part of the system they belong to.

Marinescu [1] correlates this design flaw with the metrics: Weighted Methods per Class (WMC) [6], Tight Class Cohesion (TCC) [7], Access to Foreign Data (ATFD) [1]. Analyzing the definitions of these metrics, we can conclude that *a possible "God Class" suspect will have high WMC and ATFD metric values and low TCC metric values.* As we mentioned earlier, due to the fact that is hard to establish a threshold for metrics values, we have proposed a new approach based on fuzzy clustering analysis.

The assessed entities, $AE_{GodClass}$, are the set of classes from our "log4net" application. After computing the metrics values for each class design entity, we apply the $FDHC$ algorithm in order to obtain the optimal fuzzy partition, denoted as $U_{AE_{God\ Class},M_{God\ Class}} = \{1.1.1, 1.1.2, 1.2.1, 1.2.2, 2.1, 2.2\}$. Table 2 contains the distribution of entities per clusters. This is a filtered information needed for DFE metric computation. However, the complete data of this partition is required for further analysis in order to establish the final list of suspect entities.

An important aspect regarding this partition, is that of isolated data points. These entities defined a new cluster for the partition denoted by $U_{AE_{God\ Class},M_{God\ Class}}$.

In order to study the distribution of God Class design flaw into our system, we compute the DFE metric, introduced in Section 3. The obtained value of DFE metric, $DFE(U_{AE_{God\ Class},M_{God\ Class}}) = 2.22$. is then compared with the minimum (MinDFE(214,7)=0.32) and the maximum (MaxDFE(214,7)=2.81) values of this metric, computed on a set of 214 entities distributed in 7 clusters.

The probability distribution of the corresponding partition for minimum and maximum values of DFE metric being:

$$(1/214, 1/214, 1/214, 1/214, 1/214, 1/214, 198/214),$$

$$(30/214, 30/214, 30/214, 30/214, 30/214, 30/214, 34/214).$$

By analyzing the value of the DFE metric, we can observe that the values are very close to the maximum value, "saying" that many parts of the software systems must be revised. The computation of DFE metric gives us preliminary information regarding the extent to which our system design is affected by God Class flaw. Further analysis is needed in order to identify those classes that need to be refactored.

5. Related approaches

This section presents the current state of art regarding the entropy metric as a measure of object-oriented design quality and analyzes the differences compared with our present approach.

Object-oriented design assessment are traditionally done in metrics-centric manner. Using the notion of entropy from communication information theory, new measures [19] were developed for the assessment of software designs. The metric is computed using information available in class definitions. The new complexity measure of classes is correlated with traditional complexity measures such as McCabe's cyclomatic metric and the number-of-defects metric. The defined entropy metric is shown to be a reliable measure in predicting the implementation complexity of classes, given that the design of the classes does not change substantially during implementation. *In relation to this existing approach, our proposal uses the same "initial" definition of entropy but applied to measure design flaw in an already developed object-oriented system.*

An approach was proposed in [9]. The authors propose the use of entropy as defined in Information Theory [23], to evaluate the initial status of an object-oriented design as well as its status after the addition of new functionality. The difference between the entropy of the two systems provides insight to the quality of the design in terms of how flexible it has been during the enhancement of its functionality. *Our approach aims to achieve goals similar to* [9]*, to differentiate "good"from "bad" designs by the use of an information theoretic entropy metric. We argue that our model differs by the fact that DFE metric can be applied for any design alteration type not only for the extension of the system's functionality.*

Another type of metrics based assessment of an object-oriented system is defined using only the class definition [20] and not information from the class implementation. The proposed metric is a true design metric that can be

calculated during the design phase of the software maintenance or development life cycles, before any code has been implemented. The metric could provide a better early indication (in the design phase rather than in the implementation phase) of the design complexity than has been available before. *Our model retrieves information for metric computation by parsing the source but also it could be applied at the design phase, but the amount of information beeng limited at this stage (ex. methods' complexity).*

By replicating and extending previous studies on the entropy of software systems, in [24] the authors defined three entropies as global metrics at the system level, in terms of three CK metrics computed at the class level. They extended the empirical analysis also to RFC and CBO since these CK metrics have been shown to be correlated with fault-proneness of OO class. With the aim of finding a global metric for describing software quality in terms of code degradation and reduced maintainability during time, they correlated such metrics to the variation in time of the total number of defects of the system a good measure of defect proneness. *Our current approach considers a fuzzy partition obtained starting from any object oriented metric that is selected to quantify features related to asessed design entities.*

As a conclusion, in relation to this existing approach, our also uses the notion of entropy from communication information theory [23], but applied to measure design flaw in an object-oriented system, either for an already developed object-oriented system or for an extension of the systems functionality. The information for metric computation is obtained by parsing the source code, and regarding the threshold criteria our approach overcomes the limitation impose by it, using a fuzzy clustering method.

6. Conclusion and Future Work

Software metrics are considered of great importance in software quality assurance. In spite of this fact, there is a gap between the things measured and the ones really important in terms of quality characteristics. This discontinuity is due mainly to the fact that the methods currently used for interpreting metrics results are at a low level of abstraction, incomplete and sometimes irrelevant. The current paper proposes a new metric - Design Flaw Entropy (DFE) - which completes our previous framework for object oriented design assessment, being very useful in measurements results interpretation.

To highlight the relevance of our proposed approach, a case-study has been used which aims to identify those classes from a software system that are affected by "God Class" design flaw. The source code of the open source object-oriented software system, called log4net [13] was used.

For future work, we intend to focus our research in the following directions:

- to apply the proposed evaluation framework on more case studies;
- to repeat the evaluation for a new version of the software system, after some suggested refactorings were applied;
- to automate the task of establishing the list of suspect entities and the list of refactorings that could be applied.

References

[1] R. Marinescu. *Measurement and quality in object-oriented design*, Ph.D. thesis, Faculty of Automatics and Computer Science, Politehnica University of Timisoara, 2003.

[2] S. Mazeiar, Li. Shimin, and T. Ladan. *A Metric-Based Heuristic Framework to Detect Object-Oriented Design Flaws* Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC06), 2006.

[3] 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, 92-101, 2005.

[4] L. Tahvildari, and K. Kontogiannis. *Improving design quality using meta-pattern transformations: A metric-based approach*, Journal of Software Maintenance and Evolution: Research and Practice, 16, 331-361, 2004.

[5] A.J. Riel. *Object-Oriented Design Heuristics*, Addison-Wesley, 1996.

[6] S. Chidamber, and C. Kemerer. *A metric suite for object-oriented design*, IEEE Transactions on Software Engineering, 20(6), 476–493, 1994.

[7] J.M. Bieman, and B.K. Kang. *Cohesion and Reuse in an Object-Oriented System*, ACM Symposium on Software Reusability, 1995.

[8] M. O'Keeffe, and M.Ó. Cinnéide. Search-based refactoring: an empirical study, *Journal of Software Maintenance and Evolution: Research and Practice*, 20, 345–364, 2008.

[9] A. Chatzigeorgiou, and G. Stephanides. Entropy as a Measure of Object-Oriented Design Quality, *1st Balkan Conference on Informatics (BCI'2003)*, 21–23, 2003.

[10] C. Serban. A Conceptual Framework for Object-oriented Design Assessment. *Computer Modeling and Simulation, UKSim Fourth European Modelling Symposium on Computer Modelling and Simulation*, 90–95, 2010.

[11] D. Dumitrescu. Hierarchical pattern classification, *Fuzzy Sets and Systems* 28, 145–162, 1988.

[12] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.

[13] Open source project: log4net, http://logging.apache.org/log4net.

[14] R. Martin. Design Principles and Patterns: http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf, 2006.

[15] N. Fenton. *Software measurement: A necessary scientific base*, IEEE Transactions on Softw. Engineering, 20(3), 1994.

[16] J. Han, and M. Kamber. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, (2001).

[17] A. Jain, and R. Dubes. Algorithms for Clustering Data. Prentice Hall, Englewood Cliffs, New Jersey, 1998.

[18] A. Jain, M.N. Murty, and P. Flynn. Data clustering: A review. ACM Computing Surveys, 31(3):264–323, (1999).

[19] J. Bansiyav, C. Davis, L. Etzkorn. An entropy-based complexity measure for object-oriented designs. Theory an Practice of Object Systems. 5(2):111–118, 1999.
[20] L. Etzkorn, S. Gholston, and W.E. Hughes. A semantic entropy metric. Journal of Software Maintenance: Research and Practice. 14(4):293–310, 2002.
[21] A. Marcus, M. Boxall, and S. Araban. Interface Metrics for Reusability Analysis of Components. Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04), 2004.
[22] K. Kim, Y. Shin, and C. Wu. Complexity Measures for Object-Oriented Program Based on the Entropy. In Proceedings of the Second Asia Pacific Software Engineering Conference, 1995.
[23] C.E. Shannon, and W. Weaver. The Mathematical Theory of Communication. Urbana, IL, University of Illinois Press, 1949.
[24] I. Turnu, G. Concas, M. Marchesi, and R. Tonelli. Entropy of some CK metrics to Assess Object-Oriented Software Quality. International Journal of Software Engineering and Knowledge Engineering, 23(3), 2013.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: {camelia,avescan,hfpop}@cs.ubbcluj.ro