Metrics-based Detection of Similar Software

Paloma Oliveira[1,2]
[1]Department of Computer, IFMG
Formiga-MG, Brazil
paloma.oliveira@ifmg.edu.br

Hudson Borges[2], Marco Tulio Valente[2]
[2]Department of Computer Science, UFMG
Belo Horizonte – MG, Brazil
mtov@dcc.ufmg.br

Heitor Augustus Xavier Costa[3]
[3]Department of Computer, UFLA
Lavras – MG, Brazil
heitor@dcc.ufla.br

***Abstract* - This paper presents a quantitative approach to identify similarity among object-oriented systems. This approach has three major contributions: a) a mechanism to derive thresholds for a specific metric, considering different class profiles; b) a mechanism to obtain a subset of similar systems from a portfolio of systems according to one software metric and using well-known clustering techniques; and c) a mechanism to obtain subsets of similar systems according to a set of software metrics and using concepts of graph theory. In this paper, we also present a tool that supports our approach, called SQComp. Using SQComp, we evaluated our approach in a corpus of 103 open-source systems, comprising more than 16 MLOC. As a result, we were able to found several groups of systems with strong indications of similarity.**

Keywords - internal quality; software metrics; thresholds.

I. INTRODUCTION

Object-oriented programming is the dominant software development paradigm. However, after more than four decades of the inception of the main OO abstractions, there is a surprisingly modest quantitative knowledge on the internal structure of object-oriented software [2]. Particularly, we still lack thresholds to metrics when evaluating the internal quality of OO systems. In other works, many metrics have been proposed to quantify internal OO properties, such as, size, coupling, cohesion, information hiding, and complexity [3]. However, we still lack a solid body of knowledge to effectively support the use such metrics to assess software quality.

Although we have many OO methodologies, design principles, and heuristics, we typically cannot answer some simple questions, such as, "How many lines of code (LOC) does the typical class have?" or "Is there such a thing as a 'typical class'?". What we would really like to know about software is "Is it good?", i. e., does it have quality attributes, such as, high modifiability, high reusability, high testability, or low maintenance costs [2].

There are some recent studies claiming that many software metrics follow heavy-tailed distributions [2, 10]. If this is true, it would have important implications on the types of empirical studies that are possible. One issue is the fact that heavy-tailed distributions do not have finite mean and variance. If this is the case, the central limit theorem cannot be applied; so, the sample mean and variance cannot be used as estimates of the population mean and variance. Therefore, making any conclusions on sample means and variances without fully understanding the distribution is questionable at best.

Software can be analyzed according to several characteristics, such as, static structure, dynamic execution, available functionality, and lifecycle evolution. This paper focuses on similarity of static structure, which includes many different measures of source code. More specifically, we present a quantitative approach based on a set of 19 software metrics to assess the internal similarity of object-oriented software. This set of metrics was selected because they are related to relevant software quality factors, such as, cohesion, coupling, and information hiding. This study involved 103 software from the Qualitas Corpus [10].

The paper is organized as follows. Section II describes the background work and concepts. Section III presents our approach. Section IV describes a case study carried out to evaluate our approach. Section V discusses related work. Concluding remarks are presented in Section VI.

II. BACKGROUND

The practical approach to verify similarity among object-oriented systems presented in this paper uses concepts like heavy-tailed distribution, clustering techniques, and graph cliques. These concepts are briefly described in this section.

A. Heavy-tailed distribution

The analysis of the interaction among modules of a system has been target of many researchers. Some studies indicate that systems, in general, are in conformity with heavy-tailed distribution. A characteristic of this distribution type is the frequency of high and low values being low and high, respectively, for the random variable. In such distribution, the mean value is not representative, and so, there is no typical value for random variables [5, 6, 9, 11].

There are several distributions with features of heavy-tailed. In this paper, the EasyFit[1] tool is used to fit the data. For each metric, we considered the values extracted for the classes of a given system as heavy-tailed when the "best-fit" distribution returned by EasyFit is Power Law, Weibull, LogNormal, Cauchy, Pareto or Exponential [11].

B. Clustering

Clustering is a technique for grouping of similar elements of a sample to form mutually exclusive clusters [12]. There are different methods of grouping, but the best known is the Kmeans algorithm. This method carries out cluster analysis though the partition of `n` observations (elements) into `k` clusters. These observations are distributed in clusters according to closeness to the mean value of each cluster, i.e., clusters are formed iteratively by rearranging the elements to minimize the cluster center (average).

[1] http://www.mathwave.com/products/easyfit.html.

C. Clique

A clique in an undirected graph, G = (V, E), is a complete sub graph of G, G' = (V', E'), such that V' ⊆ V, E' ⊆ E, and exists an edge between any two vertices in V' [4]. A maximal clique cannot be extended by including an additional vertex.

III. IDENTIFYING SIMILARITY OF OBJECT-ORIENTED SYSTEMS

The approach presented in this paper aims to contribute for establishing internal quality for object-oriented systems. Basically, thresholds of metrics for class profiles of a system are designed according to best practices. Such thresholds can be used as indicators of internal quality. For example, suppose two systems, S_1 and S_2. Suppose that S_1 was developed in accordance with the best principles/concepts of Software Engineering. Moreover, it was developed by a team of skilled developers, with high experience. Thus, the probability that S_1 has a high degree of internal quality is high. Suppose also that using the approach proposed in this paper, we identify that S_2 and S_1 are similar. Thereby, we claim that this similarity is a strong indication that S_2 also has high levels of internal quality.

Our approach has three principal mechanisms: A) A mechanism to obtain thresholds to specify software metrics, according to different profiles of classes; B) A mechanism to obtain set of similar systems from a system repository using clustering techniques and accordingly to thresholds of a specific metric; and C) A mechanism to obtain sets of similar systems, according to the thresholds of a set of metrics.

A. Obtaining Thresholds for Class Profiles

Defining threshold for software metrics at class level is not a trivial task [1, 5]. For example, to obtain a threshold to the LOC metric is complex. The main reason is that LOC follows a heavy-tailed distribution. In other words, systems have many classes with few LOC (e.g., less than a hundred lines) and few classes with many LOC (e.g., in some cases, thousands of lines). Therefore, the mean of LOC is unrepresentative, which makes it more challenging to analyze these systems.

To overcome this difficulty, we used clustering – specifically the Kmeans algorithm – to partition the classes into groups (G_1,...,G_n), automatically. The groups denote the classes' location in different positions in the curve of a heavy-tailed distribution. For example, suppose three groups, G_1, G_2, and G_3, and a metric, M_1. The classes in G_1, G_2, and G_3 have high, medium, and low values for M_1, respectively. These values represent the Euclidean distance between the value of M_1 for a class in a given group and the threshold of this group. The threshold for M_1 for each group is obtained by considering the mean value of M_1.

B. Obtaining a Set of Similar Systems for One Metric

This mechanism is used to determine sets (clusters) of systems with indications of similarity for a specific metric. Suppose a set of systems, S, and a metric, M_1. The thresholds for class profiles of the systems in S are calculated for M_1 and the groups G_1,..., G_n are generated (using the procedure described in the previous section). For example, suppose two systems, S_1 and S_2, one metric, M_1, and one profile, P. For example, both systems can have three profiles of classes ($P=3$): i) G_1: classes with high value for M_1; ii) G_2: classes with medium value for M_1; and iii) G_3: classes with low value for M_1.

In this mechanism we use the Kmeans algorithm again to obtain groups of similar systems according to the profile of their classes. In other words, classes of the systems in S are clustering according to values of M_1 in the groups generated. The determination of the number of clusters is not trivial when the Kmeans algorithm is used. Basically, we should run the algorithm several times to different amounts of clusters until the internal cohesion of the clusters exceeds 90%. Clusters of similar systems in relation to the values of M_1 are the output of this algorithm.

C. Obtaining a set of similar systems for a set of metrics

This mechanism is an extension of the Mechanism B. It is used to determine sets (clusters) of systems with similarity indications regarding a set of metrics. It combines two or more thresholds of distinct metrics as similarity criterion.

Suppose a set of systems, S, and a set of metrics, M. The Mechanism B should be used for each metric in M. Systems with similarity indications for metrics in M are represented by a graph. The nodes are the systems and the edges represent the minimum amount of metrics with similarity indications between two systems. For example, suppose two systems, S_i and S_j. Suppose also that the number of metrics for detecting similarity indications between S_i and S_j is $Q_{m(i,j)}$, and the minimum number of metrics that indicates that two systems have similarity is Q_{min}. There is an edge between S_i and S_j, if $Q_{m(i,j)} \geq Q_{min}$. Systems with similarity indications for the same set of metrics are represented by cliques. The resulting graph can have several cliques, which characterizes systems with similarity.

IV. CASE STUDY: *QUALITAS CORPUS*

We use the Qualitas Corpus (version 20101126r), which is a repository with 103 open-source Java-based systems, specially created for empirical research in software engineering [10]. These systems have around 115K classes and more than 16 MLOC.

We used 19 metrics related to classes for measuring important factors to software quality, such as, size, complexity, encapsulation, and cohesion. These metrics are: Number of Lines of Code (LoC); ii) Number of Attributes (NoA); iii) Number of Public Attributes (PubA); iv) Number of Private Attributes (PriA); v) Number of Inherited Attributes (IA); vi) Number of Methods (NoM); vii) Number of Public Methods (PubM); viii) Number of Private Methods (PriM); ix) Number of Inherited Methods (IM); x) Fan-Out; xi) Fan-In; xii) Coupling between Object Class (CBO); xiii) Number of Children (NoC); xiv) Total Number of Children (TNoC); xv) Response for a Class (RFC); xvi) Weighted Methods per Class (WMC); xvii) Number of Methods Overriden (NoMO); xviii) Lack of Cohesion in Methods (LCoM); and xix) Hierarchy Nesting Level (HNL).

A. *SQComp - A Computational Support for Comparative Evaluation of Internal Software Quality*

A tool was developed for providing a graphical user interface and to automate the proposed approach. This tool, called `SQComp` (Comparative Evaluation of Internal Software Quality), uses the Moose platform to compute the metrics values for each class of each system [8]. To cluster and to find cliques `SQComp we` used the R Platform[2].

B. *Methodology and Results:*

Mechanism A: Obtaining Thresholds for class profiles

Five groups, G_1, G_2, G_3, G_4, and G_5, were defined to represent threshold for class profiles. These groups correspond to classes with small values for metrics, classes with values for metrics ranging from small to medium, classes with medium values for metrics, classes with values for metrics ranging from medium to high, and classes with high values for metrics, respectively.

To illustrate, the thresholds of `LoC` regarding the groups of classes in the `FindBugs` system are presented in Table I. The calculated thresholds, the number of classes, and percentage of classes in each group are shown on the second, third, and fourth rows, respectively. Therefore, there is a better understanding of the class profiles of the `FindBugs` system related to number of lines of code. Approximately, 73.84% of classes are small (26.2 lines of code); on the other hand, 0.25% of classes are large (23331.7 lines of code). The thresholds of `LoC` for five other systems are presented in Table II.

TABLE I. THRESHOLDS FOR CLASS PROFILES IN THE FINDBUGS SYSTEM

	G_1	G_2	G_3	G_4	G_5
Number of LoC	26.2	135.2	359.3	787.5	2331.7
Number of Classes	878	226	63	19	3
Percentage of Classes (%)	73.84	19.00	5.30	1.60	0.25

TABLE II. THRESHOLDS FOR LOC IN TEN SYSTEMS

Systems	G_1	G_2	G_3	G_4	G_5
Ant	19.64	92.18	217.49	448.91	916.92
Aoi	48.50	213.37	523.74	1130.36	2635.20
Argouml	26.04	163.69	478.88	1813.00	9122.00
Aspectj	56.38	382.73	1078.24	2719.89	6332.00
Cayenne	17.11	82.16	197.59	400.77	973.33

Mechanism B: *Obtaining a set of similar systems for one metric*

The number of clusters of similar systems is obtained by the `kmeans(X, k)` function of the `R` system, where `X` is equal to 5 (groups) and `k` is number of clusters. This function was run several times for different number of clusters until we obtained, at least, 90% of internal cohesion of the clusters (as suggested in Section IV-B). For example, considering `LoC`, the value for `k` more appropriate was 5. This means that the systems ere classified into five groups considering similarity indications for `LoC`.

Most systems (33 systems ≈ 32.04%) were classified in the cluster #3. These systems present the following values for `LoC`: G_1 = 23.77; G_2 = 116.87, G_3 = 286.43; G_4 = 619.17; and G_5 = 1,525.36. In the other hand, three systems were classified in the cluster #2 and have following values for LOC: G_1 = 39.17; G_2 = 226.43; G_3 = 625.47; G_4 = 1,887.83; and G_5 = 6,147.33.

Mechanism C: *Obtaining a set of similar systems for a set of metrics*

In our approach, the Mechanism B relies on a single metric and the Mechanism C uses a set of metrics. Therefore, we used the Mechanism B for each metric that `SQComp` provides (19 metrics). Besides, we decided to use `k` = 5 (`kmeans(X, k)` function), because the internal cohesion was approximately 90%. Finally, we obtained a 103 x 103 matrix, where each cell has the number of times that two systems were classified in the same cluster (similar metric). A subset of this matrix with only 10 systems is shown in Table III.

TABLE III. NUMBER OF SIMILAR METRICS AMONG SYSTEMS

	Ant	Aoi	ArgoUML	AspectJ	Cayene
Ant	19	1	3	1	9
Aoi	1	19	9	10	2
Argouml	3	9	19	9	1
AspectJ	1	10	9	19	0
Cayenne	9	2	1	0	19

As it can be seen, the intersection cell between the `Ant` and `Cayenne` has the value nine, meaning that these systems have similarity indications for the same set of metrics (nine metrics). Cells in the main diagonal have the value 19, because each system is compared with itself. We decide to set Q_{min} = 9, because it represents half the number of metrics. An adjacency matrix was then used to represent pair of systems with $Q_{min} \geq 9$, where 1 and 0 represent the presence and absence of an edge, respectively. The graph corresponding to this adjacency matrix is shown in Figure . In this figure, there is one clique with three vertices (`AspectJ`, `AOI`, and `ArgoUML`) and one clique with two vertices (`ant` and `cayenne`).

Evaluating 103 systems. We conducted some experiments by changing Q_{min}. In one of these experiments, we used Q_{min} = 12 and we find out a maximum clique with three vertices (`Quilt`; `JUnit`; `JFinDateMath`) and similar metrics were `LoC`, `IA`, `NoM`, `PubM`, `FAN-OUT`, `FAN-IN`, `CBO`, `NoMO`, `TNoC`, `WMC`, `RFC`, and `LCoM`. Besides this experiment, two experiments are shown in Table IV. We can noticed that smaller the set of similar metrics (Q_{min}), the higher is the number of similar systems.

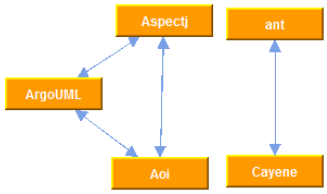

Figure 1 - Adjacency Matrix in Table III

Discussion: Software quality can be measured by external and internal factors [7]. Our approach evaluates internal software quality by considering a set of metrics that includes several features of object-oriented systems. Table IV shows the systems with indications of high similarity in their internal structure, considering coupling, cohesion, and complexity. However, the approach is not able to explain this similarity,

[2] http://www.R-project.org/

which may be attributed to development processes, programming patterns, same programmers' team, or just for being a coincidence.

TABLE IV. THREE EXPERIMENTS: MAXIMUM CLIQUE, SYSTEMS, AND SIMILAR METRICS

Q_{mim}	Systems (Vertices)	Similar Metrics (Edges)
12	Quilt; JUnit; JFinDateMath	LoC; IA; NoM; PubM; FAN-OUT; FAN-IN; CBO; NoMO; TNoC; WMC; RFC; LCoM
9	Quilt; JUnit; FitJava; JFinDateMath	LoC; NoM; PubM; CBO; NoMO; TNoC; WMC; RFC; LCoM
7	JUnit; CheckStyle; JGraph; Quilt; JFinDateMath	NoM; PubM; NoMO; FAN-OUT; CBO; RFC; LCoM

V. RELATED WORKS

In this section, we discuss work related with our approach. Baxter et. al analyzed 17 metrics in 56 Java systems for verifying their internal structure [2]. The authors reported that most metrics follow power-laws. Louridas et al. analyzed coupling metrics using 11 systems developed in multiple languages (C, Perl, Ruby, and Java) [6]. The authors concluded that most metrics are in conformity with heavy-tailed distribution, independently of programming language. Studies conducted by Taube-Schock et al. confirm such results, but for coupling metrics [9].

In the context of thresholds, a study of Alves et. al [1] used 100 systems to obtain thresholds with the aim to classify them. This study analyzed the quantile function for a set of systems and attributed weight by LOC for calculating the thresholds for each metric. Ferreira et. al [5] have proposed thresholds for 6 metrics, considering 40 systems. As result, systems were classified into 3 categories: Good, Regular, and Poor.

We can observe that there is not well-established thresholds for software metrics that reflect real practices and projects. In general, studies tend to recommend absolute thresholds, such as, "the class is good if it has at most 20 methods" [5]. However, several studies indicate that software metrics follow a heavy-tail distribution [2,5,6,9]. In other words, software practice seems to reveal that is inevitable to have software components with very high values of metrics (the tail of the distribution). In this context, our approach aims to provide thresholds respecting the heavy-tailed behavior common in software metrics distributions.

VI. CONCLUSION

A approach to verify similarity among object-oriented systems was presented in this paper. The approach includes three mechanisms: i) obtaining thresholds for class profiles; ii) obtaining a set of similar systems for a single metric; and iii) obtaining a set of similar systems for a set of metrics.

The proposed approach helps to check the existence of classes of distinct profiles in the systems analyzed in terms of size, coupling, cohesion etc. This reinforces the result of some authors that software metrics follow a heavy-tailed distribution. The use of clustering is an interesting alternative, because the mean of the elements in a cluster is more representative and can be used as threshold for different class profiles.

As future work, we suggest to use the approach in other systems repositories and to investigate why systems have a specific similarity. We also aim to investigate the existence of causality between the proposed thresholds and external software quality metrics, such as number of bugs [13] and number of warnings raised by bug finding tools [14,15].

Acknowledgments Our research is supported by CAPES, FAPEMIG, and CNPq.

REFERENCES

[1] Alves, T.; Ypma, C.; Visser, J. (2010). Deriving metric thresholds from benchmark data. In 26th *Int. Con. on Software Maintenance*, pp. 1–10.

[2] Baxter, G.; Frean, M.; Noble, J.; Rickerby, M.; Smith, H.; Visser, M.; Melton, H.; Tempero, E. (2006). Understanding the shape of Java software. *In* 21th *Int. Conf. on Object Oriented Programming, Systems, Languages and Applications*. pp. 397-412.

[3] Chidamber, S..; Kemerer, C. (1994). A metrics suite for object oriented design. *IEEE Trans. on Software Engineering.* 20(6). pp.476–493.

[4] Cormen, T.; Leiserson, C.; Rivest, R.; Cliford C. (2009). Introduction to Algorithms. 3rd edition. *MIT Press.*

[5] Ferreira, K.; Bigonha, M.; Bigonha, R.; Mendes, L.; Almeida, H. (2011). Identifying Thresholds for Object-Oriented Software Metrics. *The Journal of Systems and Software*. 85(2). pp.244-257.

[6] Louridas, P.; Spinellis, D.; Vlachos, V. (2008) Power Laws in Software. *ACM Trans. on Software Engineering and Methodology.* 18(1). pp.1-26.

[7] Meyer, B. (2000). Object-oriented Software Construction. Prentice-Hall.

[8] Oscar, N.; Stéphane, D.; Tudor, G. (2005). The Story of Moose: An Agile Reengineering Environment. *In: European Sof. Engineering Conference*. pp. 1-10.

[9] Taube-Schock, C.; Walker, R.; Witten, I.(2011). Can We Avoid High Coupling". *In. 25th European Conf. on Object-Oriented Programming*. pp. 204-228

[10] Tempero, E.; Anslow, C.; Dietrich, J.; Han, T.; Li, J.; Lumpe, M.; Melton, H.; Noble, J. (2010). The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. *In: Asia-Pacific Software Engineering Conference.* pp. 336-345.

[11] Foss, S.; Korshunov, K.; Zachary, S. (2011) An Introduction to Heavy-Tailed and Subexponential Distributions. Springer-Verlag.

[12] Tufféry, S. (2011) Association Analysis, in Data Mining and Statistics for Decision Making. *John Wiley & Sons.*

[13] Couto, C; Silva, C.; Valente, M. T.; Bigonha, R.; Anquetil, N. (2012) Uncovering Causal Relationships between Software Metrics and Bugs. *In: 16th European Conf. on Software Maintenance and Reengineering*, pp. 223-232.

[14] Araujo, J. E.; Souza, S.; Valente, M. T. (2011) Study on the Relevance of the Warnings Reported by Java Bug Finding Tools. *IET Software*, 5(4), pp. 366-374.

[15] Couto, C; Araujo J. E.; Silva, C.; Valente, M. T.. (2013) Static Correspondence and Correlation between Field Defects and Warnings Reported by a Bug Finding Tool. *Software Quality Journal*, 21(2), pp. 241-257, Springer.