Alqmase M, Alshayeb M, Ghouti L. Threshold extraction framework for software metrics. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 34(5): 1063–1078 Sept. 2019. DOI 10.1007/s11390-019-1960-6

Threshold Extraction Framework for Software Metrics

Mohammed Alqmase, Mohammad Alshayeb*, and Lahouari Ghouti

Information and Computer Science Department, King Fahd University of Petroleum and Minerals Dhahran 31261, Saudi Arabia

E-mail: {g201531270, alshayeb, lahouari}@kfupm.edu.sa

Received October 13, 2018; revised March 23, 2019.

Abstract Software metrics are used to measure different attributes of software. To practically measure software attributes using these metrics, metric thresholds are needed. Many researchers attempted to identify these thresholds based on personal experiences. However, the resulted experience-based thresholds cannot be generalized due to the variability in personal experiences and the subjectivity of opinions. The goal of this paper is to propose an automated clustering framework based on the expectation maximization (EM) algorithm where clusters are generated using a simplified 3-metric set (LOC, LCOM, and CBO). Given these clusters, different threshold levels for software metrics are systematically determined such that each threshold reflects a specific level of software quality. The proposed framework comprises two major steps: the clustering step where the software quality historical dataset is decomposed into a fixed set of clusters using the EM algorithm, and the threshold extraction step where thresholds, specific to each software metric in the resulting clusters, are estimated using statistical measures such as the mean (μ) and the standard deviation (σ) of each software metric in each cluster. The paper's findings highlight the capability of EM-based clustering, using a minimum metric set, to group software quality datasets according to different quality levels.

Keywords metric threshold, expectation maximization, empirical study

1 Introduction

Software metrics are used to measure different characteristics of software. Metric thresholds are needed to understand different levels of software quality. Thresholds can be used in the software quality assessment process[1]. They can also be used to detect bad smell code[2]. As a result, many thresholds have been proposed according to the personal experiences of subject matter experts (SMEs)[3−6]. However, these thresholds cannot be generalized due to the variability in personal experiences and the subjectivity of opinions. This variation may lead to different thresholds for the same metric. The main challenge is to provide a method for identifying thresholds systematically that can be representative of the variety of software systems. For these reasons, many attempts have been made in the literature to define scientifically sound thresholds proportional to a specific level of quality for widely used software metrics[1,7]. In these attempts, statistical measures, such as the mean and standard deviation of each software metric in each cluster, are used to extract thresholds from software quality historical data with the assumption that the metrics follow a normal distribution[1,7]. However, some studies show that many object-oriented metrics do not follow a normal distribution; rather, some follow power law distribution[8,9] and other metrics follow Pareto or log-normal distributions[10]. To avoid the assumption that metrics should be normally distributed, French[11] used the same statistical measures (mean and standard deviation) combined with Chebyshev's inequality theorem. Other attempts are also reported in the literature where the historical data and extracted thresholds are analyzed using machine learning (ML) and data mining (DM) algorithms[12−14]. However, the existing methods for threshold extraction are restricted by the maximum number of thresholds associated with the quality levels for any software metric. In most cases, the extracted thresholds cannot exceed four bins.

Regular Paper

*Corresponding Author

Defining multiple threshold levels can help to reduce the time and effort associated with refactoring since it can help the designer to focus only on the worst code. Sometimes, the designer needs to focus only on the very critical parts of the code and limit the number of redesign entities. It can help test engineers to distribute the load of testing effort efficiently where more test cases can be applied on those that have a very high risk. It is also useful for maintenance as it can help to concentrate on the worst code and reduce the range of maintenance as much as possible.

This paper proposes a solution to alleviate this restriction using a new clustering framework based on well-established ML algorithms. The proposed clustering algorithm, using the expectation maximization (EM) algorithm[15], groups software quality historical data into related clusters and derives quality thresholds for a simplified metric set. The simplified metric set, attributed to He *et al.*[16], consists of the LCOM, CBO, and LOC metrics only. This simplified metric set, used to build defect prediction models, is constructively exploited in our framework to split any software quality historical data into different clusters with variable quality thresholds.

Clustering groups data objects into a variable number of clusters or classes according to a specific similarity/distance measure[17]. Similar data objects are assigned to the same cluster in two different ways: 1) hard assignments; 2) soft assignments. In the former approach, commonly known as the K-means algorithm[18], objects are assigned to one specific cluster with a probability of 1. In the latter, soft assignments, based on the EM algorithm, are carried out where each data object is assigned to different clusters with varying membership probabilities. In this way, the EM-based clustering maximizes the overall likelihood assignment by iterating through its two main steps of expectation (E-step) and maximization (M-step) after which the EM algorithm is named. In the E-step, data objects are associated with an initial guess of the model parameters where a probability distribution is created. Then, this initial model is refined using a likelihood maximization procedure in the M-step. The final clustering decisions of the EM algorithm are obtained when the model distribution becomes stable and does not change anymore between the E-step and the M-step[15,17,18].

Following approaches similar to the ones described above, the proposed framework consists of two distinct steps: 1) software quality historical data is first split into different clusters where entities or classes with similar characteristics are grouped in the same cluster; 2) for each software metric, thresholds are extracted from each cluster using statistical measures including the mean and the standard deviation. The proposed framework is characterized by its flexibility to extract any number of thresholds for the software metric considered where these thresholds are guaranteed to reflect a specific level of quality. It is noteworthy to mention the practical usefulness of these thresholds in assessing software product quality. In addition, our proposed framework is language-agnostic and based on object-oriented (OO) characteristics and properties.

The benchmarking approach, adopted in our framework, ensures a seamless integration with software product quality tools as an assisting component and its effectiveness is highlighted through automation. In fact, our framework is fully automated and deployed as a web application which can be used in practice to extract any number of thresholds associated with software quality levels. This web application is developed using the Oracle Application Development Framework (Oracle-ADF) in conjunction with the Weka machine learning tool. The threshold extraction framework (TEFSW①) can be used with any software metric that is correlated with the software defect attribute as demonstrated by the minimum set of metrics, commonly used to predict defects (LOC, LOM, CBO)[16], used in our framework.

The remainder of this paper is organized as follows. Section 2 discusses the work related to the identification of metric thresholds. The research methodology is discussed in Section 3. In Section 4, we introduce the experiment planning and setup. Validation and analysis of the results are provided in Section 5. Finally, Section 6 concludes the paper with suggested directions for future work.

2 Literature Review

In this section, we review the work related to this research and survey different approaches and techniques to derive, identify and validate metric thresholds.

2.1 Experience-Based Threshold

Many software engineering experts define metric thresholds depending on their personal experiences. For example, McCabe[3] proposed the McCabe metric as a complexity measurement which counts the number of possible execution paths, and a threshold value

①TEFSW. http://www.qumasi.com/TEFSW/, May 2019.

of 10 was suggested. In [4], Nejmeh proposed the NPATH metric to measure software complexity using the number of possible execution paths. Similar to McCabe, a threshold value of 200 is recommended by Nejmeh. Henderson-Sellers[5] quantified thresholds associated with software metrics to classify classes into safe, flag, and alarm categories. A 3-range threshold, attributed to Coleman *et al.*[6], is assigned to the maintainability index (MI) metric. In Coleman *et al.*'s multivalued threshold study, a threshold value smaller than 65 is assigned to software products which are difficult to maintain. Moderately maintainable software products are associated with threshold values 65 and 85. Threshold values higher than 85 are used to represent highly maintainable software products. The thresholds proposed in [3] are based on personal experiences and may be affected by subjective opinions, which limits their generalization capability to other software metrics and thresholds associated with varying quality levels.

2.2 Statistics-Based Threshold

Other existing solutions are based on software metrics and data analysis to derive thresholds in which the mean and standard deviation measures are employed to extract thresholds from project data[1,7] with the assumption that the metric values are normally distributed. The mean and the standard deviation using Chebyshev's inequality theorem are used to avoid the normality assumption in the data[11]. However, this approach is sensitive to a large number of outliers. Chidamber and Kemerer used histograms to characterize and analyze data to identify the metric distribution and outliers[19]. Vale and Figueiredo proposed another method that derives thresholds in software product line context using a benchmark of 33 SPLs②[20]. This method was assessed using recall and precision and two code smells (God Class and Lazy Class) detection strategies were used[20]. However, these methods do not show how we can guarantee that the proposed thresholds reflect a specific level of quality. In addition, these approaches do not have the flexibility of generating any number of thresholds for each metric in which each threshold reflects a specific level of quality. Furthermore, these approaches do not provide a clear roadmap on how the findings can be applied in practice for software quality assessments. The proposed approach tends to be flexible enough to generate any number of thresholds for each metric in which each threshold reflects a specific level of quality.

2.3 Metric-Defect Correlation Threshold

Some studies identify thresholds by examining the correlation between defects and metrics. Benlarbi *et al.*[21] investigated the relation between metric thresholds and software failures for a subset of Chidamber and Kemerer (C&K)[19] metrics using linear regression. They evaluated the effect of the threshold to detect faults that lead to failures. Their results show that there is no threshold effect. Emam *et al.*[22] also showed that there is no empirical evidence for the threshold of class size to predict faults. However, these results are based on specific metrics and specific defect prediction models that the authors used. Other models and metrics might give different outcomes. Shatnawi *et al.*[12] conducted an empirical study to identify threshold values using receiver operating characteristic (ROC)curve analysis which finds the relation between metrics and errors in the Eclipse system. They performed an experiment using C&K metrics[19] and applied the technique to three releases of Eclipse. They conducted statistical analysis to find specific metric values that could classify the Eclipse modules (classes) into four different error categories (no error, low-impact error, medium-impact error, and high-impact error). After identifying the threshold values using ROC analysis, they were validated by the classification performance of ROC. As a result, the study identified useful threshold values for the ordinal category. However, for the binary category, the study could not obtain practical and useful threshold values.

Since the information on defects cannot be easily collected and sometimes is not available, we use a minimum set of metrics that is correlated with defects as proposed by He *et al.*[16] These metrics can be collected and calculated automatically. We use the defect information in our approach only for validation. Therefore, the proposed method is applicable and can be extended to extract thresholds for a wide range of metrics that have a correlation with defects.

In comparison with the existing work, the proposed approach is different in many aspects.

1) It is flexible enough to generate any number of thresholds for each metric in which each threshold reflects a specific level of quality. In contrast, existing approaches extract either one or a fixed number of threshold levels for each metric.

②http://labsoft.dcc.ufmg.br/doku.php?id=%20about:spl_list, May 2019.

2) Since defect information is not easily available, our approach uses a minimum set of metrics that is correlated with defects instead of directly using the defect information. This ensures the flexibility of the proposed approach in clustering the dataset from different perspectives (LOC, LOM, and CBO). Using defect information directly restricts the use of the extraction method to only datasets that have defect information.

3) Our approach allows the use of other dependent variables to build the EM cluster model, and the proposed approach is not static to the three metrics that are used in this paper. This will open the door for future research to select and use other metrics.

4) Our approach provides evidence that clustering using the EM algorithm can be used to group a software dataset into different levels of quality.

5) We build online and available web tools that allow thresholds to be extracted using another dataset and provide an easy and simple way to analyze and explore the extracted thresholds.

6) The framework and the tool can be used to extract thresholds for all metrics that are related to the defect under the research constraint.

3 Research Methodology

To use software metrics in practice for software quality assessment, we need to derive different thresholds for different levels of quality for each metric, which allows the evaluator to evaluate the software and rank it into ordinal categorization. The main reason for categorizing the datasets into different levels (categories) is to differentiate high-risk (error-prone) classes from low-risk (error-prone) classes. Therefore, the data that will be used to extract the threshold should be qualified first into different levels of quality as needed. Thus, entities with a high level of quality should be in the same cluster and entities with low quality should be in the same cluster. As a result, any threshold extracted from the cluster of high-quality entities will reflect a high level of quality. To achieve this goal, we use defect density as a standard measure of software quality to help cluster the data and then use the appropriate method to extract the threshold from each cluster. However, the main obstacle of using defect density as a standard measure of software quality is the difficulty in finding sufficient information about defects. To address this challenge and instead of using the number of defects as a main attribute to cluster and classify the entities or the data, we use metrics that have a high correlation with defects, thereby the clustering algorithm can cluster the data depending on these metrics. This leads to another challenge that is what the simple set of metrics that can identify defect density to cluster the data are. We use the minimum set of metrics proposed by He *el al.*[16] that can predict defected components. These metrics are LOC, LCOM and CBO.

The framework we propose uses machine-learning techniques and a minimum metric set (LOC, LCOM, and CBO) to identify different clusters. This methodology to define the framework has three steps as shown in Fig.1.

Step 1. This step is adopted from [16], in which the authors used the CfsSubsetEval evaluator and GreedyStepwise search algorithm in Weka to select features from the original datasets automatically. Their findings help us in our context to cluster the data into different levels of quality using their suggested minimum metric set (CBO, LCOM, AND LOC) as shown in step 1 of Fig.1. The output of this step in our experiment is the minimum set of metrics (CBO, LCOM, AND LOC) suggested by [16]. These three metrics can be used by the EM clustering algorithm to cluster the dataset into different clusters where each cluster reflects a specific level of quality, as shown in step 2.

Step 2. In this step, we use the EM clustering algorithm which assigns a probability distribution to each observation. Cross-validation can be used to decide how many clusters to generate or to specify how many clusters to create. We find this algorithm appropriate for the intended outcome and our results show that this algorithm successfully predicts different quality clusters where each cluster has classes or entities with the same characteristics. The EM clustering algorithm uses the minimum metric set that was selected by the first step to predict defect-prone classes as shown in step 1 of Fig.1.

The input of the EM clustering algorithm is the dataset of open source projects. The proposed approach is based on OO characteristics, which means it is entity-based but not project-based. Therefore, each instance in the dataset contains information about each class, component or entity. This information includes these three metrics (LOC, LCOM, and CBO) and information about the other metrics whose thresholds we want to extract. The EM clustering algorithm will take the dataset and cluster it into different clusters using the minimum metric set (LOC, LCOM, and CBO). The output of step 2 is different clusters where each cluster contains many instances and each instance contains

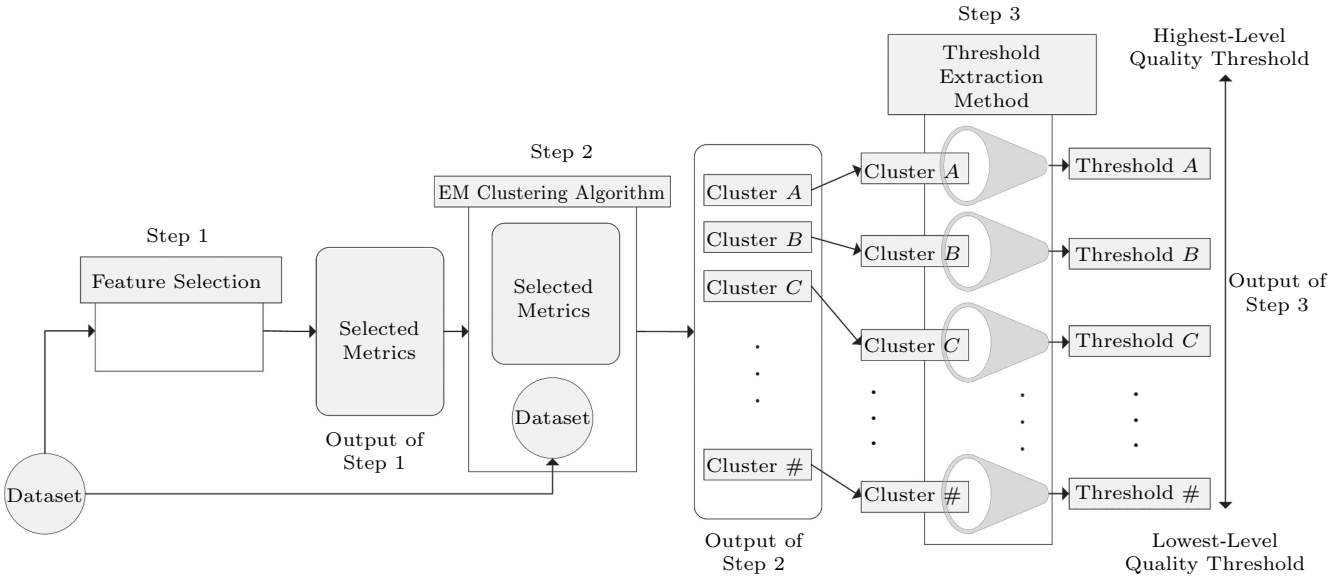

Fig.1. Architecture for the threshold extraction framework for software metrics.

information about a specific entity or class. In other words, the output should cluster the dataset into different clusters where the data of each cluster reflects a specific level of quality. As a result, any threshold that is extracted from any cluster will reflect the cluster's level of quality. Note the flexibility of generating any number of thresholds by specifying the number of clusters where each threshold represents the quality of each cluster. Different methods can be used to establish the appropriate number of clusters, such as cluster-elbow. This proposed framework is designed to be flexible so that users can select their desired number of clusters based on their experience or use other methods such as cluster-elbow.

Step 3. After clustering the dataset into different clusters successfully, the threshold can be extracted from each cluster using any preferable method. We use the mean and standard deviation as a method to extract the threshold from each cluster. However, any other method can be used to do this. Step 3 of Fig.1 shows how the thresholds are extracted from each cluster using the mean and standard deviation.

The input of step 3 is the clusters that are generated by the EM algorithm in step 2. In this step, the mean is calculated for each metric of each cluster in which the value of the mean of each metric in each cluster is the threshold of that metric in that cluster. Therefore, the output of this step is the thresholds of each metric in each cluster. For example, if we need to generate five thresholds for each metric, we need to cluster the dataset into five clusters and then we extract the threshold of each metric from each cluster using the mean method or any other method. The EM algorithm allows the number of clusters that we want to generate to be specified.

4 Experiment Planning

The goal of this study is to validate the feasibility of using machine learning techniques and the EM clustering algorithm using a simplified metric set to derive different threshold levels of software metrics systematically, in which each threshold reflects a specific level of quality.

4.1 Research Hypothesis

Because of the clustering concern with grouping instances that are similar to each other[15,17,18], we assume that the EM clustering algorithm will group the instances that have characteristics of high risks into the same cluster. While we can cluster the dataset into many clusters, we assume that each cluster contains instances with a specific level of risk, which result in the extraction of different threshold levels.

To validate this assumption, we define the following research question.

RQ. Do the metric thresholds derived from the datasets clustered using the EM clustering algorithm built with a minimum metric set (LOC, LCOM, and CBO) reflect a specific level of quality where the defect is used as the quality measure?

To answer the research question, we define the following hypothesis.

H_1. If the EM clustering algorithm is built with a simplified metric set (LOC, LCOM, and CBO), then the derived thresholds from each cluster reflect a specific level of quality for all metrics that are correlated with the defect.

H_0. If the EM clustering algorithm is built with a simplified metric set (LOC, LCOM, and CBO), then the derived thresholds from each cluster do not reflect a specific level of quality for all metrics that are correlated with the defect.

4.2 Research Variables

Twenty-one static code metrics are used in this research. All these metrics are used after clustering the datasets to extract three threshold levels for each metric. These metrics are listed in Table 1 and their description is given in [23]. The metrics of the selected minimum set (LOC, LCOM, and CBO) are used as dependent variables and will be used to build the clustering model as shown in Fig.2.

As a result, the thresholds for the other variables will be extracted after the EM clustering algorithm is built with the three dependent variables to cluster the data into different clusters.

4.3 Dataset

We use a dataset of 16 open-source Java-based projects. The projects include more than 6 500 classes and 1 903 267 lines of code. The descriptive statistics of the projects are shown in Table 2. #instances/classes, #LOC, #buggy instances, % of buggy instances and #defects are the number of instances or classes, the number of lines of code, the number of buggy instances, the percentage of buggy instances and the number of defects respectively. Each instance represents a class file that contains 20 static code metrics (e.g., CBO, WMC, RFC, LCOM) which present all the variables involved in our study. We adjust the dataset by adding a new attribute. We transform the defect proneness attribute into two different classes (bug and no bug). This leads to the binary classification problem where only binary labels are required. Therefore, we transform the label values into 0 and 1, where a label of 1 represents data instances with one or more bugs. All the datasets were collected by Jureczko and Madeyski[24], and Jureczko and Spinellis[25] using BugInfo and Ckjm tools.

Table 1. 21 Static Code Metrics

Metric Suite (Number of Metrics)	Metric Acronym	Metric Full Name
CK suite (6)	WMC	Weighted method per class
	DIT	Depth of inheritance tree
	LCOM	Lack of cohesion in methods
	RFC	Response for a class
	CBO	Coupling between object classes
	NOC	Number of children
Martins metrics (2)	CA	Afferent couplings
	CE	Efferent couplings
QMOOM suite (5)	DAM	Data access metric
	NPM	Number of public methods
	MFA	Measure of functional abstraction
	CAM	Cohesion among methods
	MOA	Measure of aggregation
Extended CK suite (4)	IC	Inheritance coupling
	CBM	Coupling between methods
	AMC	Average method complexity
	LCOM3	Normalized version of LCOM
McCabe's CC (2)	AVG_CC	Mean values of methods within the same class
	MAX_CC	Maximum values of methods in the same class
Others (2)	LOC	Lines of code
	BUG	Non-buggy or buggy

Ckjm③ is used to collect static code metrics④ while BugInfo, a method commonly used in many other research studies[16,24,25], is used for defect information; however, unfortunately, the URL of the tool is no longer accessible. The dataset used in this paper can be found in the PROMISE repository⑤, a repository for software engineering research datasets. The dataset was publicly available during the data collection in this study and was used by Jureczko *et al.*[24,25]

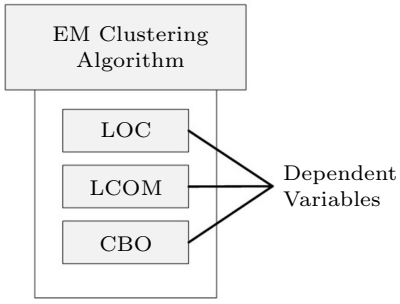

Fig.2. Selected dependent variables.

Table 2. Details of the 16 Java Open-Source Projects

No.	Project	#Instances/Classes	#LOC	#Buggy Instances	% of Buggy Instances	#Defects
1	Ant-1.7	745	208 653	166	22.3	338
2	Camel-1.6	965	113 055	188	19.5	500
3	Ivy-2.0	352	87 796	40	11.4	56
4	Jedit-4.3	492	202 363	11	2.2	12
5	Lucene-2.4	340	102 859	203	59.7	632
6	Poi-3.0	442	129 327	281	63.6	500
7	Synapse-1.2	256	53 500	86	33.6	145
8	Velocity-1.6	229	57 012	78	34.1	190
9	Xalan-2.6	885	411 737	411	46.4	1 213
10	Xerces-1.4	588	141 180	437	74.3	1 596
11	Tomcat-6.0.3	895	300 674	77	8.6	114
12	log4j-1.2	206	38 191	180	87.4	498
13	forrest-0.8	32	6 540	2	6.3	6
14	e-learning-1.0	64	3 639	5	7.8	9
15	Berek-1.0	43	32 320	16	37.2	70
16	Zuzel-1.0	29	14 421	13	44.8	22

5 Validation and Analysis

In this section, we validate the feasibility of using machine learning techniques and the EM clustering algorithm to derive different thresholds of software metrics.

5.1 Descriptive Statistics

Tables 3–9 summarize the estimated statistics including the mean, median, standard deviation, maximum, minimum, skewness and kurtosis for each metric in each cluster, respectively. First, the mean and median measures are given in Table 3 and Table 4 where we characterize the central tendency of the data being used. The standard deviation measure, reported in Table 5, provides a description of the dispersion in the data. Table 6 and Table 7 give a summary of the dynamic range for each metric using the maximum and the minimum measures respectively. Finally, we calculate the skewness and the kurtosis that measure the degree of asymmetry and peakedness or flatness of the distribution as shown in Table 8 and Table 9, respectively. The size of each cluster is represented in Fig.3.

Table 3. Mean Measure for Each Metric in Each Cluster with 3-Level Threshold

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	62.8	4.9	2.2	3.9	11.0	2.0	2.1	0.5
Cluster 3	2	302.6	10.7	47.4	12.0	34.1	4.6	4.9	1.0
Cluster 1	3	1 340.2	36.0	931.5	39.2	93.8	23.9	11.3	2.4

③http://www.spinellis.gr/sw/ckjm/, May 2019.

④http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metric.html, May 2019.

⑤http://promise.site.uottawa.ca/SERepository/, July 2019.

Table 4. Median Measure for Each Metric in Each Cluster with 3-Level Treshold

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	39.0	4.0	1.0	3.0	8.0	1.0	1.0	0.0
Cluster 3	2	232.0	9.0	27.0	11.0	29.0	2.0	3.0	0.0
Cluster 1	3	943.0	23.0	374.0	33.0	79.0	6.0	6.0	1.0

Table 5. Standard Deviation Measure for Each Metric in Each Cluster with 3-Level Treshold

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	65.1	3.5	2.8	3.0	9.2	2.5	2.5	1.0
Cluster 3	2	272.0	8.4	55.0	6.8	23.2	6.8	6.4	1.7
Cluster 1	3	1 466.0	46.4	2 494.5	31.5	74.5	45.9	17.8	5.2

Table 6. Maximum Measure for Each Metric in Each Cluster with 3-Level Treshold

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	283.0	17.0	12.0	39.0	78.0	17.0	22.0	17.0
Cluster 3	2	1 383.0	45.0	276.0	83.0	178.0	44.0	85.0	24.0
Cluster 1	3	13 175.0	499.0	41 713.0	351.0	540.0	498.0	209.0	62.0

Table 7. Minimum Measure for Each Metric in Each Cluster with 3-Level Treshold

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Cluster 3	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Cluster 1	3	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0

Table 8. Skewness Measure for Each Metric in Each Cluster with 3-Level Treshold

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	1.2	0.8	1.4	3.4	1.3	2.3	3.0	4.8
Cluster 3	2	1.4	1.2	1.7	1.9	1.3	2.7	4.2	4.4
Cluster 1	3	3.1	4.5	10.1	3.3	2.0	4.8	5.5	5.3

Table 9. Kurtosis Measure for Each Metric in Each Cluster with 3-Level Treshold

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	3.7	3.2	4.0	27.7	5.7	9.3	14.3	45.9
Cluster 3	2	4.9	4.2	5.5	12.3	5.2	10.9	33.0	35.5
Cluster 1	3	18.7	34.4	138.9	24.6	9.4	37.7	46.8	42.3

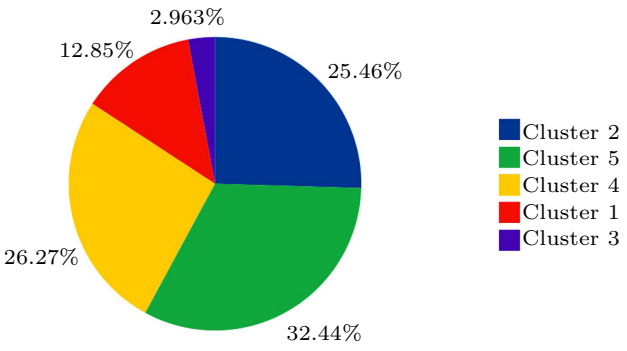

Fig.3. Cluster size.

5.2 Validation Using Statistical Analysis

Several studies investigated the relationship between lines of code (LOC) and defects[26,27]. Their results show that the relationship between them is linear. Since our dataset has information about defects (BUG) and LOC, we can use them to validate that each cluster reflects a specific level of quality. We use defects to measure software quality and the LOC as a cluster level indicator. The information about each cluster is sorted increasingly using LOC as shown in Table 10 and Table 11. Table 10 shows three cluster models and Table 11 shows five cluster models (the details of the 5-level thresholds are shown in Appendix). Both tables contain the mean for each cluster for LOC and BUG as thresholds. Table 10 and Table 11 demonstrate the linear relationship between LOC and BUG for each cluster. Maintaining this relationship means that each cluster represents a specific level of quality. Therefore, the confusion matrix can be constructed using the fol-

lowing information.

For each generated cluster:

• if the current cluster has a lower value than the previous cluster in both BUG and LOC attributes, then add 1 into the true positive (TP);

• if the current cluster does not have a lower value than the previous cluster in both BUG and LOC attributes, then add 1 into the false positive (FP);

• if the current cluster has a higher value than the next cluster in both the BUG and the LOC attributes, then add 1 into the true negative (TN);

• if the current cluster does not have a higher value than the next cluster in both BUG and LOC attributes, then add 1 into the false negative (FN).

Table 10. 3-Level Clustering Result for LOC Metric and Bug Measure

Cluster	Threshold	LOC	BUG
Cluster 2	1	62.8	0.5
Cluster 3	2	302.6	1.0
Cluster 1	3	1 340.2	2.4

Table 11. 5-Level Clustering Result for LOC Metric and Bug Measure

Cluster	Threshold	LOC	BUG
Cluster 5	1	22.3	0.4
Cluster 1	2	119.5	0.7
Cluster 3	3	283.8	0.9
Cluster 4	4	768.0	1.7
Cluster 2	5	2 359.9	3.6

Table 10 shows the clustering result for LOC and BUG, where each cluster represents a specific level of quality from the defect perspective. We choose LOC to relate with defect density to examine the relation between them. Maintaining the correlation between them means that the clustering algorithm is successful in clustering the dataset into different levels of quality. After calculating the mean and the standard deviation for both, the mean of all clusters demonstrates a line correlation. For example, the mean of cluster 3 has the lowest value for both LOC and BUG while the mean of cluster 2 has the highest values for both compared with the other clusters. This confirms that the EM clustering algorithm has the ability to group the high-level risk entities in the same cluster (e.g., cluster 2), the low-level risk entities in cluster 3, and the others in between.

In Table 12, we compare each cluster with two other clusters, the previous and the next cluster. Since we have three clusters, each cluster is compared twice, once with the next cluster and once with the previous cluster, except for the first and last clusters which are compared only once. Table 12 illustrates the results of this comparison.

Table 12. Confusion Matrix

Comparing Each Cluster with Adjacent Cluster	Lower Value of Defects	Higher Value of Defects
Lower value of LOC compared with the next	TP (2)	FN (0)
Higher value of LOC compared with the previous	FP (0)	TN (2)

To evaluate and validate the effectiveness of the derived thresholds, a statistical and a comparative analysis are performed. For the statistical analysis, precision, recall and $F1$ score ($F1$) measures are computed. These measures are usually used to assess the accuracy and validate the thresholds derived from clustering the datasets. These thresholds, based on the EM clustering algorithm, reflect a specific level of quality. In our case, the obtained result is significant, which confirms the research questions and the hypothesis.

$$Precision = \frac{TP}{TP + FP} = \frac{2}{2+0} = 1,$$
$$Recall = \frac{TP}{TP + FN} = \frac{2}{2+0} = 1,$$
$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{2 \times 1 \times 1}{1+1} = 1.$$

In the next equation, we use the correlation coefficient (also called Pearson's product moment coefficient) for correlation analysis.

$$r_{A,b} = \frac{\sum_{i=1}^{n}(a_i - \mu_A)(b_i - \mu_B)}{(a_i - \mu_B)\sigma_A\sigma_B} = 0.996,$$

where n is the number of clusters (as tuples), and μ_A and μ_B are the respective means of A (LOC) and B (Bug) respectively, σ_A and σ_B are the respective standard deviation of A and B respectively. The attribute LOC used in Table 10 is considered as A in the above equation while the attribute BUG is considered as B.

In order to assess whether a given correlation is statistically significant, we use the p-value. We obtain the p-value using the regression procedure in the data analysis ToolPak in Excel.

The correlation P-value in Table 10 for three clusters is 0.05 and the correlation P-value in Table 11 for five clusters is 0.000 28. This shows that the correlation is statistically significant, especially in Table 11.

As we see, the result shows a strong correlation which gives further evidence that the clustering algorithm used is appropriate for extracting different level thresholds in which each threshold reflects the specific level of quality. For clustering quality measures, Table 13 outlines the clustering quality using the purity and $F1$ score measures.

$$\text{purity}_i = \frac{1}{n_i} \max_{j=1}^{k} \{n_{ij}\},$$
$$\text{purity} = \sum_{i=1}^{r} \frac{n_i}{n} \text{purity}_i = 0.621.$$

The $F1$ score usedto measure the clustering quality is defined as follows:

$$F_i = \frac{2 \times n_{ij_i}}{n_i + m_{j_i}},$$
$$F1 = \frac{1}{r} \sum_{i=1}^{r} F_i = 0.434.$$

From Table 13, we observe that the highest-quality threshold 1 and the lowest quality threshold 3 have the highest purity (0.70) and (0.61) respectively. On the other hand, threshold 2 which has a moderate quality level, has the lowest purity. In the highest-quality cluster (threshold 1), most of the data points are unbuggy; thus, purity is high from this perspective. For the lowest-quality cluster (threshold 3), most of the data points are buggy; thus, purity is high from this perspective. For the in-between clusters, purity decreases from both sides. This result confirms the hypothesis and gives further evidence that the clustering algorithm used is appropriate for extracting different level thresholds in which each threshold reflects a specific level of quality.

In Subsection 5.3, we further validate the significance of our findings by comparing the obtained results with the method proposed by Ferreira *et al.*[28]

5.3 Validation Using Comparison Analysis

In this subsection, we compare the thresholds extracted by the proposed method with the thresholds derived by the method proposed by Ferreira *et al.*[28] The comparison process has four steps. 1) The values of each metric are collected for every system and grouped into a unique file. 2) A weight ratio function is computed for each value and each value with its weight becomes unique. 3) The values are sorted in ascending order and the values of entities are then aggregated. 4) We select two thresholds (the range of the labels) that define three different labels called good, regular, and bad, respectively.

To conduct a fair comparison, we need to satisfy the following criteria.

• We must derive the thresholds for both approaches using the same dataset. We implement Ferreira *et al.*'s method[28] using the TDTool tool. Then, we derive the thresholds using the same dataset. Table 14 shows the thresholds derived using our proposed method and Table 15 demonstrates the thresholds extracted using Ferreira *et al.*'s method on the same dataset. The TDTool implements four methods: Alves *et al.*'s method[29], Ferreira *et al.*'s method[28], Oliveira *et al.*'s method[30] and the method of Vale and Figueiredo[20]. The implementation is done by Veado *et al.*[31] and is available online.

• The number of levels must be the same. Ferreira *et al.*'s method extracts two-level thresholds to generate three categories (good, regular and bad). Therefore, we cluster the dataset using the proposed method into only two clusters. Ferreira *et al.*'s method has three categories (good, regular and bad), as shown in Table 15 and the proposed method represents two clusters (two levels) (cluster 2 as level 1 and cluster 1 as level 2), as shown in Table 14 and Table 16.

• The size of each level must be the same. The size of each cluster must be the same when using Ferreira *et*

Table 13. Clustering Quality Measures Using Purity and $F1$ Score with 3-Level Treshold

Threshold	Cluster	#Buggy Data Points	#Unbuggy Data Points	Count	Max	Purity	$F1$ Score
1	Cluster 2	1 000	2 321	3 321	2 321	0.698 886	0.628 401 245
2	Cluster 3	1 309	1 483	2 792	1 483	0.531 160	0.432 487 606
3	Cluster 1	408	262	670	408	0.608 955	0.240 921 169
Total		2 717	4 066	6 783	4 212	0.620 964	0.433 936 673

Table 14. Two Thresholds Extracted Using the Proposed Method

Cluster	Size (%)	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	75.76	1	113.1	6.4	9.2	5.8	16.5	2.6	2.7	0.6
Cluster 1	24.23	2	833.3	22.9	435.9	26.1	66.7	13.4	8.6	1.7

Table 15. Two-Level Thresholds Extracted Using the Method Proposed by Ferreira *et al.*[28]

Label	Rang	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Good	0–49	1	< 101	< 6	< 5	< 6	< 17	< 2	< 2	< 0
Regular	49–90	2	101–699	6–21	5–154	6–23	17–66	2–10	2–10	0–2
Bad	90–100	3	> 699	> 21	>154	> 23	> 66	> 10	> 10	> 2

Table 16. Two-Level Thresholds Extracted Using Our Method to Get Three Groups

Label	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Good	1	< 113	< 6	< 9	< 6	< 17	< 3	< 3	< 1
Regular	2	113–833	6–23	9–435	6–26	17–67	3–13	3–9	1–2
Bad	3	> 833	> 23	> 435	> 26	> 67	> 13	> 9	>2

al.'s method when clustering the data into three clusters as in our proposed method (as shown in Fig.4). The second column in Table 15 illustrates the size of each level for each method.

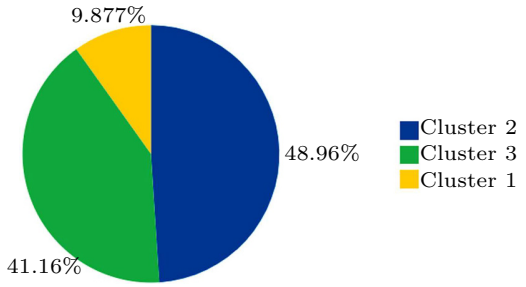

Fig.4. Size of clusters in the proposed method.

Table 15 and Table 16 summarize the results of extracting two-level thresholds of both methods with three categories of quality for both.

After conducting the comparison, we find that the thresholds extracted by our method are within the range of Ferreira *et al.*'s method[28]. They are close in the range for each category. The gap between both methods is small. For example, the first threshold (threshold level 1) for the metric LOC which represents a very high quality is 113 in Table 16 is close to the threshold level 1 (101) of Ferreira *et al.*'s method[28], as shown in Table 15. The threshold level 2 for LOC in our method is 833 whereas the threshold level 2 in Ferreira *et al.*'s method is 699 and so on. Therefore, each metric in each level threshold extracted by our method is close to the range of the thresholds derived by Ferreira *et al.*'s method except for the thresholds of level 2 for the metric LCOM, which is more when using our approach. The key advantage of the proposed approach compared with Ferreira *et al.*'s method[28] is that the proposed approach has the flexibility to generate one or more thresholds while Ferreira *et al.*'s method is restricted to two thresholds to identify three categories (good, regular and bad). Identifying more than one threshold can help in categorizing software entities into several risk levels. If one threshold helps in differentiating high-risk classes from low-risk classes, then more than one threshold can help in distinguishing different risk levels, as shown in [7, 20, 29]. This would help in many software areas, for example, it can help to reduce the time and effort of refactoring and bad smell detection as it allows the designer to focus only on the worst code to limit the number of redesign entities. In testing, it can help to distribute the load of testing effort efficiently where more test cases can be applied on those parts that have higher risks. In maintenance, it can help to concentrate on the worst code and reduce maintenance effort. In our future work, we plan to conduct experiments to show how we can use different threshold levels to propose a software quality assessment model that assesses and evaluates software projects and ranks them to specific quality levels. The proposed method will be implicitly used as a quality measure to rank the software project into different levels of quality.

The result of the comparison illustrates the significance of our findings and is the evidence that the metric thresholds derived from the datasets clustered using the EM clustering algorithm, reflect a specific level of quality and provides further evidence to accept the research hypothesis.

5.4 Tool Support

We implemented a web application tool using Oracle ADF integrated with Weka. Fig.5 shows the main page of the tool. The tool simplifies the extraction of thresholds for any software metric. It provides a visualization tab to visualize and analyze the thresholds. It has been deployed and hosted in the site[⑥].

[⑥]TEFSW. http://www.qumasi.com/TEFSW/, May 2019.

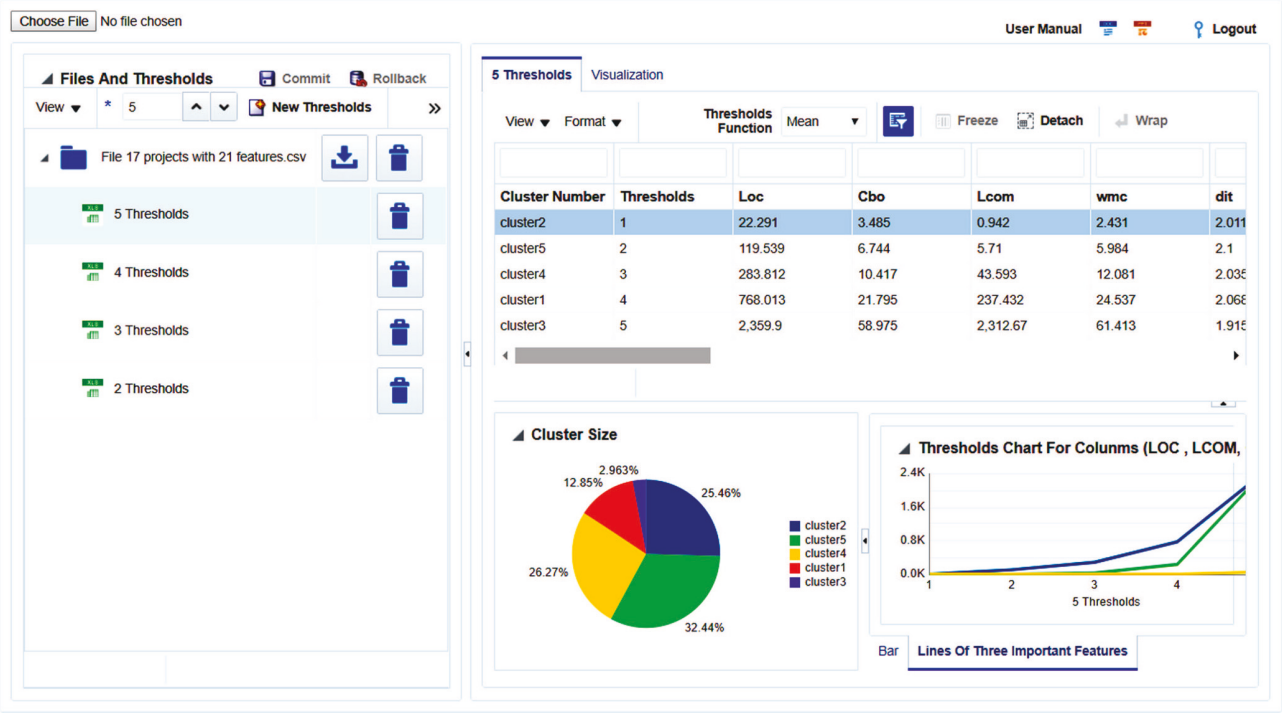

Fig.5. Main page of the implemented tool.

5.5 Threats to Validity

While we obtain significant results to answer the research question, there are potential threats that may affect the validity of this work. Threats to construct validity concern the relationship between theory and observation. These threats are primarily related to the static code metrics and the tool used to extract the metrics. There are many tools to extract software metrics, but each tool might calculate specific metrics differently and produce different values for specific metrics[32]. This might lead to different threshold values for specific metrics. In this research, all the datasets were collected by Jureczko and Madeyski[24], and Jureczko and Spinellis[25] using Ckjm and BugInfotools. Ckjm⑦ is used to collect static code metrics while BugInfo is used to identify defects and can be found in the PROMISE repository⑧.

For threats to internal validity, we cluster the dataset using the EM clustering algorithm built with the simplest metric set (LOC, LCOM, and CBO). Building the EM clustering algorithm using other features might not give the same result. However, we adopt these metrics as they were proposed by He *et al.*[16] using selection feature techniques. This means that the metric set may depend on the dataset. However, we use the same dataset that was used by He *et al.* to select these three metrics. In this stage, our work may be considered as a foundation that opens the door for future research.

A threat to external validity concerns the dataset used by the proposed approach. It contains only Java open-source projects; other languages might exhibit different results. Another possible threat to external validity concerns the tool that implements our approach to extract thresholds easily. We develop a web application to extract thresholds easily, but maybe with some errors. However, we perform extensive testing, implement the clustering models in Weka and integrate them with the tool.

6 Conclusions

Many software metrics have been proposed to measure different attributes of the software. For these metrics to be effectively used to measure and improve software quality, metric thresholds should be identified. These threshold values can guide the assignment of soft-

⑦http://www.spinellis.gr/sw/ckjm/, May 2019.

⑧All software datasets along with description are available at http://openscience.us/repo/, May 2019.

ware quality to a variable set of quality levels (such as good, regular, and bad levels). In this paper, we proposed a machine learning based framework to extract any number of thresholds associated with varying quality levels. The proposed framework relies on a robust clustering scheme derived using the expectation-maximization (EM) algorithm. First, this framework selects a minimum metric set, commonly used in defect prediction models, to assign historical software quality datasets to different clusters. Each cluster consists of entities or classes with similar characteristics. Then, for each metric, thresholds are extracted from each cluster using statistical measures based on the mean, standard deviation, median or mode of each software metric in each cluster. The resulting clusters are characterized with different threshold levels for the metrics in a systematic fashion. Each extracted threshold reflects a specific software quality level. To assess the performance of the proposed framework, historical datasets, consisting of 16 open-source Java-based projects with more than 6 500 classes and 1 903 267 lines of code, were used. Based on the results reported in this paper, the EM-based clustering algorithm, built with the minimum metric set (LOC, LCOM, and CBO), can cluster historical software quality datasets into different levels of quality from the perspective of defects as a quality measure. Each cluster is characterized with the extracted thresholds associated with a specific quality level. To evaluate and validate the effectiveness of the derived thresholds, statistical and comparative analysis was performed. Performance in the statistical analysis is based on precision, recall, F-measure, and p-value. Our proposed framework attained values of 1 for the first three measures and 0.000 2 for the p-value on five clusters. In addition, the thresholds extracted by our framework are well aligned and in agreement with those proposed in the literature as reported in the comparative analysis study carried out in this paper. Furthermore, the proposed framework, unlike existing ones, can extract a variable number of threshold levels, which provides more flexibility and makes this framework more appropriate for practical software quality assessment tools and products.

In future research, we aim to find a minimum metric set for other external attributes, such as performance, reliability and security. In addition, we will adapt/adjust/alter our framework to generate thresholds of metrics that are not correlated with defects. Moreover, we will conduct repeated research with other datasets, other subsets of metrics and other clustering algorithms.

Acknowledgement The authors acknowledge the support of King Fahd University of Petroleum and Minerals in the development of this work.

References

[1] Erni K, Lewerentz C. Applying design metrics to object-oriented frameworks. In *Proc. the 3rd IEEE International Software Metrics Symposium*, March 1996, pp.64-74.

[2] Abílio R, Padilha J, Figueiredo E, Costa H. Detecting code smells in software product lines — An exploratory study. In *Proc. the 12th International Conference on Information Technology — New Generations*, April 2015, pp.433-438.

[3] McCabe T J. A complexity measure. *IEEE Transactions on Software Engineering*, 1976, SE-2(4): 308-320.

[4] Nejmeh B A. NPATH: A measure of execution path complexity and its applications. *Commun. ACM*, 1988, 31(2): 188-200.

[5] Henderson-Sellers B. Object-Oriented Metrics: Measures of Complexity. Prentice Hall, 1995.

[6] Coleman D, Lowther B, Oman P. The application of software maintainability models in industrial software systems. *Journal of Systems and Software*, 1995, 29(1): 3-16.

[7] Lanza M, Marinescu R. Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. Springer, 2006.

[8] Wheeldon R, Counsell S. Power law distributions in class relationships. In *Proc. the 3rd IEEE International Workshop on Source Code Analysis and Manipulation*, September 2003, pp.45-54.

[9] Concas G, Marchesi M, Pinna S, Serra N. Power-laws in a large object-oriented software system. *IEEE Transactions on Software Engineering*, 2007, 33(10): 687-708.

[10] Baxter G, Frean M, Noble J *et al.* Understanding the shape of Java software. In *Proc. the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, October 2006, pp.397-412.

[11] French V. Establishing software metric thresholds. In *Proc. the 9th International Workshop on Software Measurement*, September 1999, Article No. 7.

[12] Shatnawi R, Li W, Swain J, Newman T. Finding software metrics threshold values using ROC curves. *Journal of Software Maintenance and Evolution: Research and Practice*, 2010, 22(1): 1-16.

[13] Catal C, Alan O, Balkan K. Class noise detection based on software metrics and ROC curves. *Information Sciences*, 2011, 181(21): 4867-4877.

[14] Herbold S, Grabowski J, Waack S. Calculation and optimization of thresholds for sets of software metrics. *Empirical Software Engineering*, 2011, 16(6): 812-841.

[15] Do C B, Batzoglou S. What is the expectation maximization algorithm? *Nature Biotechnology*, 2008, 26: 897-899.

[16] He P, Li B, Liu X, Chen J, Ma Y. An empirical study on software defect prediction with a simplified metric set. *Information and Software Technology*, 2015, 59: 170-190.

[17] Sharma N, Bajpai A, Litoriya M R. Comparison the various clustering algorithms of Weka tools. *International Journal of Emerging Technology and Advanced Engineering*, 2012, 2(5): 73-80.

[18] Hill T, Lewicki P. Statistics: Methods and Applications; A Comprehensive Reference for Science, Industry, and Data Mining. StatSoft, 2006.

[19] Chidamber S R, Kemerer C F. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 1994, 20(6): 476-493.

[20] Vale G A D, Figueiredo E M L. A method to derive metric thresholds for software product lines. In *Proc. the 29th Brazilian Symposium on Software Engineering*, September 2015, pp.110-119.

[21] Benlarbi S, Emam K E, Goel N, Rai S. Thresholds for object-oriented measures. In *Proc. the 11th International Symposium on Software Reliability Engineering*, October 2000, pp.24-39.

[22] Emam K E, Benlarbi S, Goel N, Melo W, Lounis H, Rai S N. The optimal class size for object-oriented software. *IEEE Transactions on Software Engineering*, 2002, 28(5): 494-509.

[23] Spinellis D, Jureczko M. Metric descriptions. http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metric.html, December 2018.

[24] Jureczko M, Madeyski L. Towards identifying software project clusters with regard to defect prediction. In *Proc. the 6th International Conference on Predictive Models in Software Engineering*, September 2010, Article No. 9.

[25] Jureczko M, Spinellis D. Using object-oriented design metrics to predict software defects. In *Proc. the 5th International Conference on Dependability of Computer Systems*, June 2010, pp.69-81.

[26] Zhang H. An investigation of the relationships between lines of code and defects. In *Proc. the 25th IEEE International Conference on Software Maintenance*, September 2009, pp.274-283.

[27] Lipow M. Number of faults per line of code. *IEEE Transactions on Software Engineering*, 1982, SE-8(4): 437-439.

[28] Ferreira K A M, Bigonha M A S, Bigonha R S, Mendes L F O, Almeida H C. Identifying thresholds for object-oriented software metrics. *Journal of Systems and Software*, 2012, 85(2): 244-257.

[29] Alves T L, Ypma C, Visser J. Deriving metric thresholds from benchmark data. In *Proc. the 26th IEEE International Conference on Software Maintenance*, September 2010, Article No. 44.

[30] Oliveira P, Valente M T, Lima F P. Extracting relative thresholds for source code metrics. In *Proc. the 2014 IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering*, February 2014, pp.254-263.

[31] Veado L, Vale G, Fernandes E, Figueiredo E. TDTool: Threshold derivation tool. In *Proc. the 20th International Conference on Evaluation and Assessment in Software Engineering*, June 2016, Article No. 24.

[32] Lincke R, Lundberg J, Löwe W. Comparing software metrics tools. In *Proc. the 2008 International Symposium on Software Testing and Analysis*, July 2008, pp.131-142.

Mohammed Alqmase received his M.S. degree in computer science from King Fahd University of Petroleum and Minerals, Dhahran, in 2019, and his B.S. degree in information technology (IT) from King Abdul-Aziz University, Jeddah, in 2013. He worked as a content management system analyst for Hippo CMS in 2017. He also worked as an instructor in Sana'a Community College, Sana, Yemen, from 2013 to 2015. His research interests include sentiment analysis, natural language processing, machine learning, algorithms and software engineering.

Mohammad Alshayeb received his M.S. and Ph.D. degrees in computer science and a Certificate of Software Engineering from the University of Alabama in Huntsville in 2000, 2002 and 1999, respectively. He received his B.S. degree in computer science from Mutah University, Jordan, in 1995. He worked as a senior research associate in the Information Technology and Systems Center, Huntsville, Alabama, USA. Currently, he is working as an associate professor at the Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahram, a position he has held since 2003. Dr. Alshayeb's research interests include software quality and quality improvements, software measurement and metrics, and empirical studies in software engineering.

Lahouari Ghouti received his Ph.D. degree in computer science from Queen's University of Belfast, Belfast, UK, in 2005. He holds his M.Sc. degree in electrical engineering from King Fahd University of Petroleum and Minerals (KFUPM), Dhahran. Dr. Ghouti worked as a research fellow in Queen's University of Belfast, Belfast. In September 2007, he was appointed as an assistant professor of Computer Science at KFUPM, Dhahran. In May 2016, he became an associate professor. He received the Best Paper Award from IEEE AHS 2006 Conference. He is also the recipient of the Silver Medal of the John Wiley Breast Journal in 2011. Dr Ghouti was selected as the KFUPM Best Teacher and Best Academic Advisor in 2015 and 2016. His current research interests include artificial intelligence, machine and deep learning.

Appendix

Tables A1–A7 show the five-level thresholds for each metric generated by the proposed approach and tool, and Table A8 shows the clustering quality measures using purity and F-measure for the five-level thresholds for each metric generated by the proposed approach and tool.

Table A1. Mean Measure for Each Metric in Each Cluster

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	22.3	3.5	0.9	2.4	5.6	1.7	1.1	0.4
Cluster 5	2	119.5	6.7	5.7	6.0	18.4	2.5	3.2	0.7
Cluster 4	3	283.8	10.4	43.6	12.1	33.9	4.3	4.9	0.9
Cluster 1	4	768.0	21.8	237.4	24.5	64.5	12.1	8.2	1.7
Cluster 3	5	2 359.9	59.0	2 312.7	61.4	139.2	43.3	16.5	3.6

Table A2. Median Measure for Each Metric in Each Cluster

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	17.0	3.0	1.0	2.0	5.0	1.0	1.0	0.0
Cluster 5	2	107.0	6.0	4.0	5.0	17.0	1.0	2.0	0.0
Cluster 4	3	242.0	9.0	35.0	11.0	31.0	2.0	3.0	0.0
Cluster 1	4	677.0	17.0	187.0	24.0	60.0	4.0	5.0	1.0
Cluster 3	5	1 863.0	35.5	1 319.5	59.0	131.0	13.0	8.0	1.5

Table A3. Standard Deviation Measure for Each Metric in Each Cluster

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	20.9	2.4	1.2	1.5	4.3	2.0	1.0	0.8
Cluster 5	2	82.8	4.3	5.8	3.4	10.7	3.2	3.8	1.4
Cluster 4	3	214.1	7.4	37.8	5.5	20.0	6.0	6.2	1.6
Cluster 1	4	610.5	18.5	229.1	14.0	42.9	17.8	10.2	3.3
Cluster 3	5	2 116.8	73.3	4 225.6	45.6	101.8	73.6	26.6	7.0

Table A4. Maximum Measure for Each Metric in Each Cluster

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	85.0	11.0	4.0	13.0	23.0	11.0	12.0	7.0
Cluster 5	2	357.0	20.0	22.0	39.0	78.0	19.0	85.0	17.0
Cluster 4	3	939.0	34.0	160.0	58.0	117.0	32.0	79.0	21.0
Cluster 1	4	2 891.0	90.0	1 044.0	122.0	346.0	88.0	143.0	40.0
Cluster 3	5	13 175.0	499.0	41 713.0	351.0	540.0	498.0	209.0	62.0

Table A5. Minimum Measure for Each Metric in Each Cluster

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Cluster 5	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Cluster 4	3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Cluster 1	4	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0
Cluster 3	5	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0

Table A6. Skewness Measure for Each Metric in Each Cluster

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	0.9	0.5	1.1	1.3	0.9	1.7	3.5	3.1
Cluster 5	2	0.6	0.6	1.0	3.0	0.9	2.2	6.2	4.7
Cluster 4	3	0.7	0.8	1.0	1.3	0.8	2.2	4.1	4.3
Cluster 1	4	0.9	1.1	1.2	1.0	1.2	2.0	4.3	5.5
Cluster 3	5	2.0	2.9	5.9	2.2	1.2	3.1	4.2	4.5

Table A7. Kurtosis Measure for Each Metric in Each Cluster

Cluster	Threshold	LOC	CBO	LCOM	WMC	RFC	CA	MAX_CC	BUG
Cluster 2	1	2.9	2.7	2.9	6.7	3.2	5.8	24.5	17.6
Cluster 5	2	2.8	2.7	3.1	20.6	4.5	8.2	103.7	40.0
Cluster 4	3	2.7	2.9	3.4	8.9	3.4	7.8	30.5	32.7
Cluster 1	4	3.5	4.0	4.2	7.3	6.7	6.2	41.1	46.0
Cluster 3	5	9.0	14.2	47.9	12.7	5.1	15.3	24.6	29.9

Table A8. Clustering Quality Measures Using Purity and F-Measure ($F1$ Score)

Threshold	Cluster	#Buggy Data Points	#Unbuggy Data Points	Count	Max	Purity	F-Measure
1	Cluster 5	459	1 279	1 738	1 279	0.735 903	0.440 730 531
2	Cluster 1	786	1 430	2 216	1 430	0.645 307	0.455 269 023
3	Cluster 3	845	92	1 767	922	0.521 788	0.316 132 350
4	Cluster 4	485	37	864	485	0.561 343	0.270 874 058
5	Cluster 2	142	56	198	142	0.717 172	0.097 427 101
Total		2 717	4 066	6 783	4 258	0.627 746	0.316 086 612