Clustering Software Metric Values Extracted from C# Code for Maintainability Assessment

Shumail Arshad
School of Computer Science,
The University of Manchester,
PO Box 88,
Manchester M60 1QD, U.K
+44 161 306 3304
shumailarshad@gmail.com

Christos Tjortjis
School of Science & Technology,
International Hellenic University,
14th km Thessaloniki - Moudania
57001 Thermi, Greece
+30 2310807576
c.tjortjis@ihu.edu.gr

ABSTRACT

This paper proposes an automated approach for supporting software maintenance using software metrics and data mining. We gather metric values from C# source code elements, such as projects, files, namespaces, classes, and interfaces. These elements are clustered together, based on their similarity, with regards to these metrics, in order to identify problematic, complex classes that might be error prone. We applied this approach to two open source software systems in C#. Results show that it supports identification of potentially problematic code parts, which require further examination and proactive maintenance.

CCS Concepts

• Information systems → Information systems applications → Data mining → Clustering
• Software and its engineering → Software creation and management → Software post-development issues → Maintaining software

Keywords

Clustering; Data mining; Software Metrics; Software Maintenance

1. INTRODUCTION

Maintenance is a complex part of the software lifecycle. Evaluating maintainability is difficult, as one has to consider many contradicting aspects. Metrics are used to measure software characteristics and support software quality and maintainability assessment, as well as “complexity hotspot” identification [2]. However, collecting and analyzing metric values for large software systems can be very hard and time consuming [4].

Data mining can be used to analyze software metrics for maintenance purposes, as it can extract information and discover hidden patterns in them [3], [7], [16]. As data mining can deal with large amounts of data in the absence of any prior domain knowledge, it is considered to be a suitable solution for large, unfamiliar software systems [6].

SETN ’16, May 18 - 20, 2016, Thessaloniki, Greece
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3734-2/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2903220.2903252.

This paper proposes an automated approach for supporting software maintenance using software metrics and data mining. The primary objective of this work is to facilitate maintenance engineers in identifying source code “complexity hotspots”. This is done by extracting Object Oriented (OO) software metric values from code and storing these in a database. We then apply K-Means to extract useful patterns. These patterns facilitate pinpointing exceptional and potentially problematic parts of the code, which require further examination and perfective maintenance. We experimented by applying this approach to two open source software written in C#, and produced very promising results.

The remaining of the paper is organized as follows: Section 2 introduces key concepts on software metrics, data mining and clustering; Section 3 discusses the proposed approach; Section 4 presents experimental results and Section 5 discusses and evaluates results. Finally, Section 6 completes the paper with conclusions and directions for further work.

2. BACKGROUND

This paper uses core concepts from software metrics, data mining, machine learning and clustering. We briefly discuss these concepts in the following sub-sections.

2.1 Software Metrics

Metrics are quantitative measures that enable software engineers and managers to gain understanding of a software system. This work emphasizes on OO metrics and follows up C# code mining work we have reported in [9] and Java code and metric mining we reported in [7]. Given the promising results when clustering elements derived from code, particularly metrics derived from Java systems, we decided to follow a similar line of attack when addressing C# systems. We used the following metrics: Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number Of Children (NOC), Coupling Between Object classes (CBO), Lack of Cohesion Of Methods (LCOM2), Response For a Class (RFC), and Number of Public Methods (NPM) [1], [5], [13].

2.2 Clustering

Clustering is useful when there is little prior information about the data. Entities can be grouped together according to the similarity of their attributes; often, distance measures are used to this effect [6]. Cluster creation can be performed in a number of ways. The purposed method uses *K-Means*, one of the most popular partitional clustering algorithms [12]. K-Means has a number of weaknesses as it requires the user to specify the total number of clusters, it works with numerical data and cannot handle noise [7].

2.3 Data Mining for Software Engineering

Data mining algorithms have been used to support software engineering tasks. Tribus et al. [19] used classifiers for knowledge discovery and troubleshooting software in C. Menzies et al. used data mining to predict errors and assist large project management [14]. They used metrics such as McCabe's Cyclomatic Complexity and classifiers such as C4.5 for C code. Prasad et al. proposed an approach for source code evaluation by knowledge engineering [17]. It discovers weaknesses and errors in code using the frequency of words or symbols in C++ code.

Data mining has also been proposed as a potential technology for supporting and enhancing our understanding of software metrics and their relationship to software quality [3], [7]. Large amounts of measurement data are produced when developing software. According to following general steps, software data can be used for extracting useful information, thus better understanding their processes and products [3].

- Define the goal of the data mining process.
- Assemble the dataset by extracting data from a software system.
- Data cleaning and preprocessing.
- Select the data mining task (clustering in this case).
- Select the data mining algorithm (we use K-Means).
- Apply data mining and evaluate the results.
- Consolidate results with prior knowledge, review and use the knowledge extracted.

3. PROPOSED APPROACH

This work aims to facilitate program comprehension by assisting maintenance engineers to focus on possible errors and problematic classes in the source code.

3.1 Systems analysis

In this work, we analyse OO systems. We use classes, methods and member data to calculate OO software metric values which act as input to the clustering process.

3.2 Data extraction process

After creating a suitable input data model, the next step is to extract data from the source code. For this purpose, our system uses MetaSpec Parser Library which is fully compliant with ECMA-334 and ECMA-335 standards. OO metric values are calculated from the data extracted by the parser.

3.3 Data mining process

We use a custom implementation of K-means clustering on OO metric values extracted from C# source code. As clustering has also the potential to discover programming patterns and ''unusual'' or outlier cases which may require further attention, our system generates graphs showing potential problematic, highly complex classes that might be error prone.

4. EXPERIMENTAL RESULTS

To evaluate our approach, we have analyzed the following two open source applications:

- *Sharp Develop* is an open source IDE written in C# that contains 3634 classes [18].
- *NUnit* is an open source unit testing framework, written in C# that contains 698 classes [15].

Table 1 details various metrics and aspects of the two systems under examination. NUnit contains fewer classes than Sharp Develop. NUnit also demonstrates lower average values than Sharp Develop, for all metrics except LCOM2. For NUnit the average WMC is approximately 8, which shows that classes are not very complex and can be easier to understand than these of Sharp Develop, having on average WMC approximately 13.

Table 1 Metrics for the two systems analyzed

	NUnit		***Sharp Develop***	
	Avg.	**Max.**	**Avg.**	**Max.**
WMC	7.9985	216	12.9785	1057
DIT	1.7450	6	2.2642	6
NOC	0.3138	18	0.5410	218
CBO	3.2957	37	4.1835	110
LCOM2	0.0064	0.9103	0.0012	0.9908
RFC	16.1948	673	18.4408	1422
NPM	4.7894	212	5.5894	317

Similarly, for NUnit the average DIT is 1.745, which shows that most of the classes are at the top of the inheritance tree, given that C# classes have a minimum DIT equal to 1. This indicates that the program is simple, although Sharp Develop fairs also well with DIT about 2.26 on average. Average NOC for NUnit is 0.3138 which shows that classes have overall few children classes, allowing maintainers to amend these without affecting other classes. Average CBO for NUnit is about 3, which shows that most of the classes have limited coupling with the other classes and that if one changes one class they will have to test approximately 4 classes in total. LCOM2 in NUnit is greater than Sharp Develop which shows that classes in Sharp Develop are more cohesive than in NUnit and possibly some classes in NUnit could be broken into subclasses. Average RFC is almost equal in both systems; their values show that classes do not have very complex response sets. Maximum values for each metric show the most complex classes of the system.

5. EVALUATION

We produce two types of results: a) statistical analysis of the metric values across classes, presented as metric value graphs, discussed in 5.1 and b) results after clustering classes, where these are treated as entities and metric values as attributes, discussed and visualized as groups (clusters) and their respective plots against each one of the metrics used in 5.2.

5.1 Metric Value Graphs

We produced graphs for the metric for each of the two systems we examined (NUnit and Sharp Develop) and will discuss the most indicative ones for illustration purposes.

5.1.1 WMC:

This graph helps maintainers appreciate the complexity of the whole system. As shown in Fig. 1, more than 130 classes have WMC=2 and more than 90 have WMC= 1 which means that 220 classes have very low complexity in NUnit.

5.1.2 NOC:

This graph helps maintainers' to view the breadth of the inheritance tree of all the classes. As shown in Fig. 2, some 620 classes have NOC = 0 in NUnit.

5.2 Metrics Cluster Graphs

We used classes as entities for clustering, their seven attributes being the respective metric values. These graphs show the clusters created by K-Means for k=5, a value known to be a sensible choice [16]. By examining these clusters, a maintainer can identify outliers (classes with exceptional metric values) and may consider improving or even refactoring them. The x-Axis shows clusters 1 to 5, and the y-axis shows the metric values present in each respective cluster. We show indicative metric cluster graphs for NUnit and Sharp Develop.

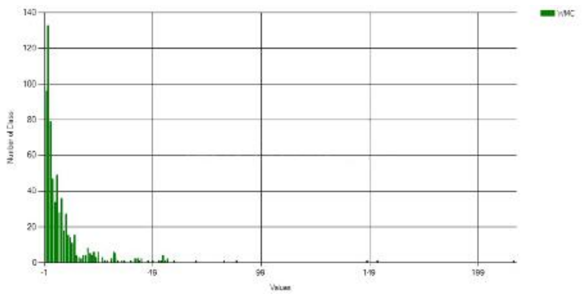

Figure 1: WMC value graph for NUnit

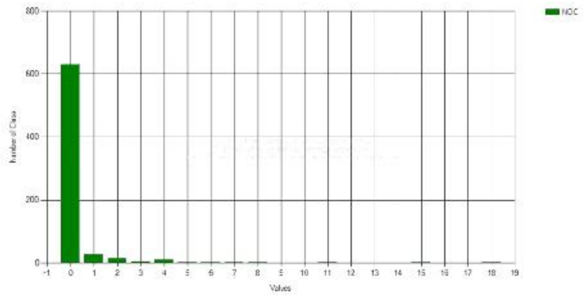

Figure 2: NOC value graph for NUnit

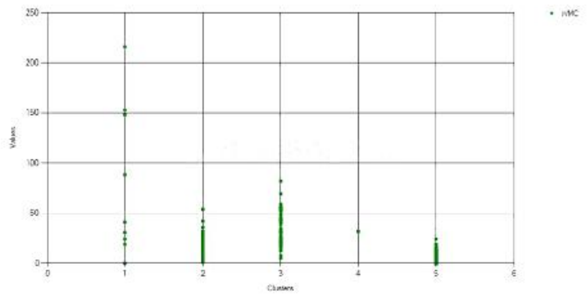

Figure 3: WMC cluster graph for NUnit

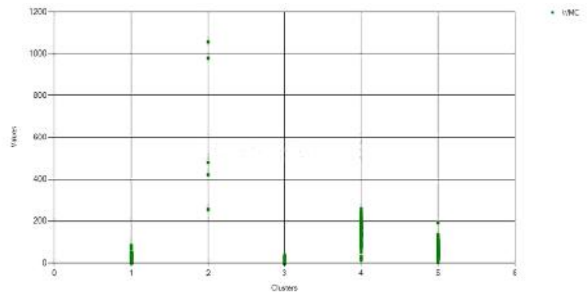

Figure 4: WMC cluster graph for Sharp Develop

5.2.1 WMC
In Fig. 3 for NUnit, clusters 2, 3, 4 and 5 consist of 93% of the total class population and have low WMC values. These classes are not very complex. Cluster 1 contains classes with high WMC values which should be considered for closer scrutiny. Similarly, in Fig. 4 for Sharp Develop, clusters 1, 3, 4 and 5 contain 95% of the classes and have low WMC. Cluster 2 contains classes with high WMC that can be considered for redesign.

5.2.2 NOC
In Fig. 5 for NUnit, clusters 1, 2, 3 and 4 (88% of total population) contain classes with fewer children than these of classes in cluster 5, which means these classes are easier to maintain. Similarly, in Fig. 6 for Sharp Develop, clusters 1, 2, 4 and 5 contain classes with fewer children than these of classes in cluster 3. Classes with high NOC values can potentially become maintenance bottlenecks.

5.3 Discussion
By examining results in 5.1, we observe how the generated metric value graphs can assist in identifying class level complexity hotspots. One can look for classes with high complexity, coupling, breadth and depth of inheritance and so on, either in isolation or comparing various systems. For instance there are two classes in NUnit with WMC values around 150 and one with WMC over 200. Even worse, there are two classes in Sharp Develop with WMC values over 400 and one with WMC over 1050. Similarly it can be observed that outliers can be found also for NOC. Although there may be good reasons for designing such classes, in most cases, this should raise the alarm for perfective maintenance. Similarly, one can look at the clustering results in 5.2, which confirm findings from 5.1.

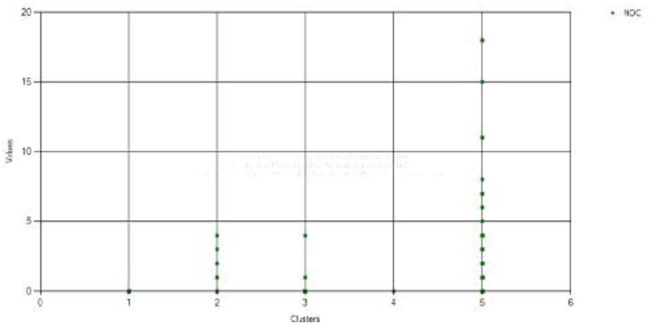

Figure 5: NOC cluster graph for NUnit

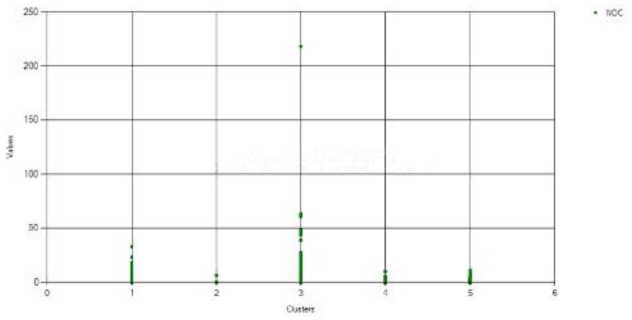

Figure 6: NOC cluster graph for Sharp DevelopIt is clear for instance, that classes in cluster 1 for NUnit and in cluster 2 for Sharp Develop require attention, as they exhibit higher than average WMC values (Fig. 3, 4). However, one can observe that cluster 5 for NUnit and cluster 3 for Sharp Develop exhibit higher than average NOC values (Fig. 5, 6), even though their respective WMC values are normal (Fig. 3, 4).

Similar work, employing K-Means and Neural – Gas for clustering modules with similar software measurements (LOC, Cyclomatic Complexity etc.), in order to predict their fault proneness and potential noisy modules was presented in [21]. Our work here extends this by employing object oriented metrics, which are more suitable for modern software systems. External, alas subjective and hard to acquire for open source systems, validation is a useful extension for our work.

6. Conclusions and Future Work

In this work we proposed and evaluated an approach for clustering C# classes based on a number of selected OO metrics. The idea is based on the notion that complex, difficult to maintain classes might score low across a number of dimensions, although in theory one could design classes which only "fail" on a single criterion. The approach follows up work we have done in mining source code elements as well as metric values extracted from Java code [7]. Given these early promising results on clustering elements derived from code as well as metrics, we followed a similar line of attack when addressing C# systems here.

Given the explorative nature of this work, we combined existing technologies wherever possible, particularly for parsing and pre-processing source code and for visualizing results. At the heart of our approach lies K-Means which we tailored to suit this particular domain. We experimented with two modest open source systems which have adequately demonstrated the viability of the concept. However, proper validation is required in order to assess our key findings and the value these add to software maintenance. We plan to conduct this validation in the future along with further work to optimize the approach.

We also recognize further threats to the validity of our approach related with very nature of software metrics, such as the perceived immaturity and unprofessionalism of the field metrics or the high cost of metrics programs [4]. In some cases, metrics programs are rejected because they "do more harm than good" [10]. It is also known that existing OO software metric tools interpret and implement the definitions of metrics differently [11].

6.1 Future Work

Apart from a more comprehensive validation we also plan to experiment by varying the focus of our approach. So far we monitored complexity and maintainability at the class level, but we could alternatively focus on the project, file, namespace, interfaces, member data, or even at the method and parameters level. Other OO properties, such as cohesion, coupling, code clones, and rule violations need to be considered in the future.

Also, even though K-means was shown to be a sensible choice for clustering data extracted from source code [8], there are promising alternatives available. For instance constraint K-Means avoids creating empty clusters the way K-means does, by introducing an extra step defining constraints for the algorithm at the beginning [20]. Also weighted K-Means could be used as it calculates each cluster's centroid not by simply calculating the mean of all attribute values but by assigning weights to these [11]. This way each metric can be assigned weights using expert opinion. Finally we can avoid having to define the number of clusters in advance, and to depend on initial centre definition, if we employ clustering algorithms automating this process [7].

7. REFERENCES

[1] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design", *IEEE Trans. Software Engineering*, Vol. 20, No. 6, pp. 476-493, 1994.

[2] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using Metrics to Evaluate Software System Maintainability", *IEEE Computer*, Vol. 27 No. 8, pp. 44-49, 1994.

[3] S. Dick, A. Meeks, M. Last, H. Bunke, and A. Kandel, "Data mining in software metrics databases", *Fuzzy Sets and Systems*, Vol. 145, No. 1, pp. 81–110, 2004.

[4] N.E. Fenton and M. Neil, "Software metrics: successes, failures and new directions", *Journal of Systems and Software*, Vol. 47, No. 2-3, pp. 149-157, 1999.

[5] R. Ferenc, I. Siket, and T. Gyimothy, "Extracting Facts from Open Source Software", *Proc. 20th IEEE Int'l Conf. Software Maintenance*, p.60-69, 2004.

[6] J. Han, and M. Kamber, *"Data Mining, Concepts and Techniques"*, 3rd Ed., Morgan Kaufmann, 2011.

[7] Y. Kanellopoulos, P. Antonellis, C. Tjortjis, and C. Makris, "k-Attractors: A Clustering Algorithm for Software Measurement Data Analysis", *Proc. 19th IEEE Int'l Conf. on Tools with Artificial Intelligence* (ICTAI 07), pp. 358-365, 2007.

[8] Y. Kanellopoulos, T. Dimopoulos, C. Tjortjis and C. Makris, "Mining Source Code Elements for Comprehending Object-Oriented Systems and Evaluating Their Maintainability", *ACM SIGKDD Explorations, Vol. 8 No. 1*, pp. 33-40, 2006.

[9] Y. Kanellopoulos, C. Makris and C. Tjortjis, "An Improved Methodology on Information Distillation by Mining Program Source Code", *Data & Knowledge Engineering*, Vol. 61, No 2, pp. 359-383, 2007.

[10] C. Kaner, W.P. Bond, "Software Engineering Metrics: What do they measure and how do we know", *10th Int'l Software Metrics Symposium*, 2004.

[11] K. Kerdprasop, N. Kerdprasop and P. Sattayatham, "Weighted K-Means for Density-Biased Clustering", *Data Warehousing and Knowledge Discovery*, *Lecture Notes in Computer Science*, Vol. 3589, pp. 488-497, 2005.

[12] J. B. MacQueen: "Some Methods for classification and Analysis of Multivariate Observations", *Proc. of 5th Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, 1:281-29, 1967.

[13] McCabe, "A Complexity Measure", *IEEE Transactions Software Engineering*, Vol. SE-2, No. 4, p.p. 308- 320, 1976.

[14] T. Menzies, J. Greenwald, A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors", *IEEE Transactions on Software Engineering*, Vol. 32, No. 11, pp. 2-13, January 2007.

[15] NUnit http://www.nunit.org/ (last access 12/15).

[16] D. Papas and C. Tjortjis, "Combining Clustering and Classification for Software Quality Evaluation", *SETN 2014, LNCS*, Vol. 8445, pp. 273-286, 2014.

[17] A.V.K. Prasad, S.R. Krishna, "Data Mining for Secure Software Engineering-Source Code Management Tool Case Study", *Int'l Journal of Engineering Science and Technology*, Vol. 2 (7), pp. 2667-2677, 2010.

[18] SharpDevelop www.icsharpcode.net/OpenSource/SD/ (last access 12/15).

[19] H. Tribus, I. Morrigl, S. Axelsson, "Using Data Mining for Static Code Analysis of C", *Proc. 8th Int'l Conf. Advanced Data Mining and Applications* (ADMA 2012), LNAI 7713, pp. 603-614, 2012.

[20] K. Wagstaff, C. Cardie, S. Rogers and S. Schroedl, "Constrained k-means clustering with background knowledge", *Proc. 18th Intl. Conf. on Machine Learning*, pp. 577–584, 2001.

[21] S. Zhong, T.M. Khoshgoftaar, and N. Seliya, "Analyzing Software Measurement Data with Clustering Techniques", *IEEE Intelligent Systems*, Vol. 19, No. 2, pp. 20-27, 2004.