Employing Clustering for Assisting Source Code Maintainability Evaluation according to ISO/IEC-9126

Panagiotis Antonellis[1], Dimitris Antoniou[1], Yiannis Kanellopoulos[1, 2], Christos Makris[1], Evangelos Theodoridis[1], Christos Tjortjis[2], Nikos Tsirakis[1]

[1] Universiry of Patras, Computer Engineering and Informatics Department, Greece
[2] The University Of Manchester, School Of Computer Science, U.K.

Abstract

This paper elaborates on how to use clustering for the evaluation of a software system's maintainability according to the ISO/IEC-9126 quality standard. More specifically it proposes a methodology that combines clustering and multicriteria decision aid techniques for knowledge acquisition by integrating groups of data from source code with the expertise of a software system's evaluators. A process for the extraction of elements from source code and Analytical Hierarchical Processing for assigning weights to these data are provided; k-Attractors clustering algorithm is then applied on these data, in order to produce system overviews and deductions. The methodology is evaluated on Apache Geronimo, a large Open Source Application Server; results are discussed and conclusions are presented together with directions for future work

1. Introduction

Software maintenance is considered as the most difficult stage in software lifecycle. According to the National Institute of Standards and Technology (NIST), it costs the U.S. economy $60 billion per year [12]. Given this high cost, maintenance processes can be considered as an area of competitive advantage. There are several studies for evaluating a system's maintainability and controlling the effort required to carry out maintenance activities [2], [14], [18]. According to ISO/IEC-9126, maintainability is the capability of a software product to be modified. Evaluating such a characteristic is a difficult process as many contradictory criteria must be considered in order to reach a decision.

This paper presents a methodology that facilitates the evaluation of a software product's maintainability according to the ISO/IEC-9126 software engineering quality standard. The intuition of this methodology is to integrate groups of measurement data extracted from source code's elements with the expertise of a system's evaluators by providing them the ability to define a number of attributes suitable for such evaluation. For this reason:

- Metrics are extracted from elements of system's source code.
- Relative weights are assigned to these metrics by employing the Analytical Hierarchy Process, reflecting their importance on evaluating maintainability.
- The k-Attractors clustering algorithm [4] is then applied on the derived ISO/IEC-9126's maintainability values, in order to provide the evaluator with a quick and rough grasp of the system.

We attempt to evaluate the usefulness of this methodology by employing as test-bed, Geronimo 1.0, an open source application server used in real life industrial applications. The remaining of this paper is organized as follows: Section 2 reviews existing work in the area of data mining and software evaluation. Section 3 outlines the logic behind the main parts of the proposed methodology. Section 4 assesses the accuracy of the output of the proposed framework, analyses its results and outlines deductions from its application. Finally, conclusions and directions for future work are presented in Section 5.

2. Background

Data mining [3], is the process which extracts implicit, previously unknown, and potentially useful information from data, by searching large volumes of them for patterns and by employing techniques such as classification, association rules mining, and clustering. It is a quite complex topic and has links with multiple core fields such as computer science and adds value to rich seminal computational techniques from statistics, information retrieval, machine learning and pattern recognition. Its ability to deal with vast amounts of data has been considered a suitable solution in assisting software maintenance, often resulting in remarkable results [1], [7], [8]], [10], [20]. As previous studies have shown, data mining is capable to obtain useful knowledge about the structure of large systems.

Sartipi et al. used data mining for architectural design recovery [16]. They proposed a model for the evaluation of the architectural design of a system based on associations among system components and used system modularity measurement as an indication of design quality and its decomposition into subsystems. Besides association rules, the clustering data mining technique has been used to support software maintenance and software systems knowledge discovery [21], [15]. The work in [15] proposes a methodology for grouping Java code elements together, according to their similarity and focuses on achieving a high level system understanding.

Understanding low/medium level concepts and relationships among components at the function, paragraph or even line of code level by mining C and COBOL legacy systems source code was addressed in [19]. For C programs, functions were used as entities, and attributes defined according to the use and types of parameters and variables, and the types of returned values. Then clustering was applied to identify sub-sets of source code that were grouped together according to custom-made similarity metrics [19]. An approach for the evaluation of dynamic clustering is presented in [22]. The scope of this solution is to evaluate the usefulness of providing dynamic dependencies as input to software clustering algorithms. Finally, Clustering over a Module Dependency Graph (MDG) [9] uses a collection of algorithms which facilitate the automatic recovery of the modular structure of a software system from its source code. The method creates a hierarchical view of system architecture into subsystems, based on the components and the relationships between components that can be detected in source code.

Recently, [8] presented an approach that examines the evolution of code stored in source control repositories. This technique identifies *Change Clusters*, which can help managers to classify different code change activities as either software maintenance or a new development. On the other hand [20] analyzes whether some change coupling between source code entities is significant or only minor textual adjustments have been checked in, as reflect the changes to the source code entities. An approach for analyzing and classifying change types based on code revisions has been developed. Finally, in [4] language processing techniques are applied to extend human judgment into situations where obtaining direct human judgment is impractical due to the volume of information that must be considered.

The value of this work that differentiates it from what presented above, is that we don't cluster raw software measurement data. Instead, we provide the evaluator the ability to employ a Multicriteria Analysis (MA) method, the Analytical Hierarchy Process (AHP), for assigning relative weights to the extracted metrics in order to reflect their importance on evaluating maintainability. This helps incorporating the evaluator's domain expertise with the measurement data extracted from source code, which may lead to more accurate and interesting clustering results.

3. Description of the Methodology

The proposed methodology is supported by the Code4Thought tool [24]. Our main purpose when implementing this tool was to use open source and portable technologies. Thus, we decided to use the Java programming language for implementing the main functionality of our tool, the MySQL database for storing our data the PHP scripting language for designing the user interface of our tool.

This section presents the logic behind the following modules that constitute the Code4Thought tool:

- Data extraction and preparation
- Weights assignment
- Data analysis

3.1. Data Extraction and Preparation

The objective of data extraction and preparation is two-fold:

- At first to collect appropriate elements that describe the software architecture and its characteristics. These elements include native source code attributes and metrics.
- Then to analyze the collected elements, choose a refinement subset of them and store them in a relational database system for further analysis.

Native attributes include Definition files, classes, Structure blocks etc. Metrics, on the other hand, provide additional system information and describe more effectively the system's characteristics and behaviour.

All the metrics are associated with a native source code attribute, e.g. the lack of cohesion is associated with a class member method. All of the above collected attributes and metrics are stored into appropriate structured XML files. We have chosen XML because of its interoperability and its wide acceptance as a de facto standard for data representation and exchange. Storing the metrics in XML files enables further processing and analysis with a variety of tools.

For simplicity, we chose to analyse a refinement subset of the most important collected elements. This subset should be small enough in order to be easily analyzed and large enough to contain all the necessary system information. Based on this requirement, we stored and further analyzed only the metrics and their associated native attributes.

The elements chosen need to be extracted from the XML files and stored permanently in a relational database. For this reason we used tools that map XML elements and nodes into any relational database, keeping the extraction method transparent from the underlying database.

Figure 1 depicts the general architecture of data extraction and preparation module.

3.2 Weights Assignment

As mentioned above, we have adopted the analytic hierarchy process (AHP) for the weights assignment. AHP is a decision making technique that allows consideration of both qualitative and quantitative aspects of decisions [25]. It reduces complex decisions to a series of one-on-one comparisons and then synthesizes the results. Compared to other techniques, like ranking or rating techniques, AHP emulates the human ability to compare single properties of alternatives. It not only helps decision makers choose the best alternative, but also provides a clear rationale for the choice.

In a systematic way AHP compares a list of objectives or alternatives. When used in the systems engineering process, AHP can be a powerful tool for comparing alternative design concepts. Assuming that a set of objectives has been established; and that we are trying to establish a normalized set of weights to be used when comparing alternatives using these objectives. AHP forms a pairwise comparison matrix A, where the number in the i-th row and j-th column gives the relative importance of objective O(i) as compared with O(j). Values that usually are used are in a 1–9 scale, with a(i,j) = 1 if the two objectives are equal in importance, a(i,j) = 3 if O(i) is weakly more important than O(j), a(i,j) = 5 if O(i) is strongly more important than O(j), a(i,j) = 7 if O(i) is very strongly more important than O(j), and a(i,j) = 9 if O(i) is absolutely more important than O(j). After this procedure the comparison matrix is normalized and its eighenvalues are

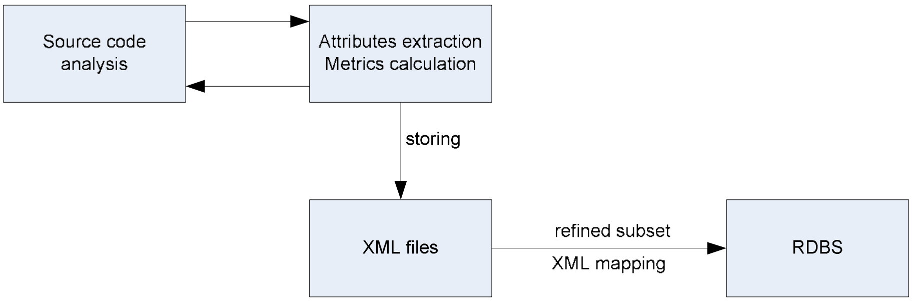

Figure 1. Architecture of data extraction and preparation module

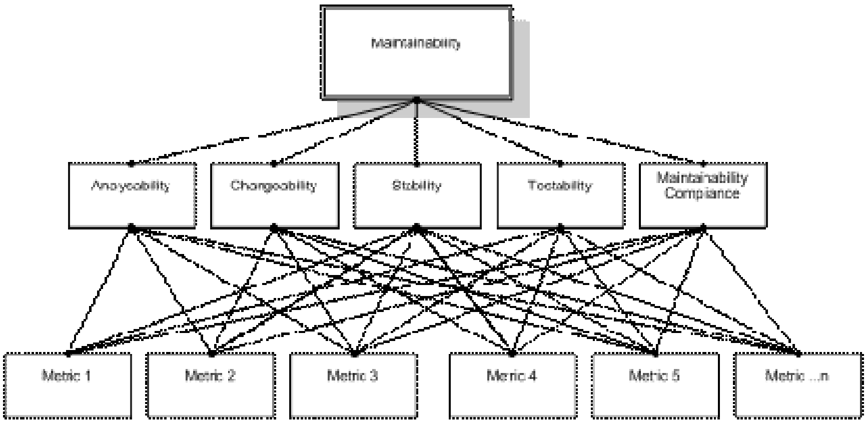

Figure 2: Weights Assignment Hierarchy

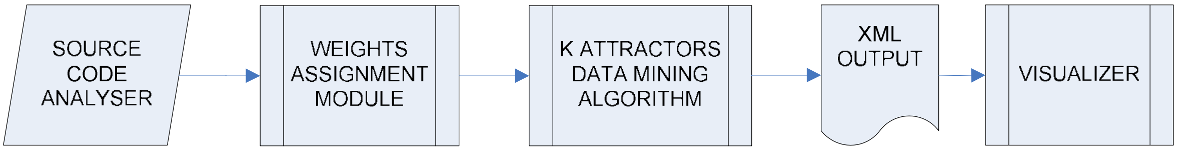

Figure 3. Data analysis module

computed. These eighenvalues play the role of coefficients/weights when someone wants to evaluate the alternatives for the examined objectives.

In our case when we aim at evaluate maintainability (see Figure 2) from a set of employed metrics, we apply AHP procedure in each level of the maintainability metrics hierarchy. At the first level we evaluate the characteristics (analyzability, changeability, etc) from the extracted metrics and at the second level we evaluate maintainability from the characteristics applying AHP procedure again. So at first level we construct a pairwise comparison table for each one of the characteristics reflecting the expert's knowledge of how much each metric influences each characteristic. Then by applying the normalization and extraction of eighenvalues upon each matrix we find the weight of each metric for calculating a score for each characteristic. At the higher level a pairwise comparison table is constructed too reflecting the expert's knowledge of how much each characteristic influences maintainability; and the weights are calculated by normalization and eighenvalues extraction.

3.3. Data Analysis

As depicted in the Figure 3, the k-Attractors algorithm, accepts data from the source code analyzer, by performing queries on the database, wherethe data reside. The outcome of the analysis is stored in XML files, in order to be visualized by the corresponding module.

In the case of software maintainability evaluation, clustering produces overviews of systems by creating mutually exclusive groups of classes, member data or methods, according to their similarities in terms of technical (source code) measurements [16]. This helps reducing the time required to understand and evaluate the overall system. Another contribution of clustering is that it helps discovering programming patterns and "unusual" or outlier cases which may require attention.

For this purpose the k-Attractors algorithm was employed which is tailored for numerical data such as measurements from source code **Error! Reference source not found.**. The main characteristics of k- Attractors are:

- It defines the desired number of clusters (i.e. the number of k), without user intervention.
- It locates the initial attractors of cluster centers with great precision.
- It measures similarity based on a composite metric that combines the Hamming distance and the inner product of transactions and clusters' attractors.

The k-Attractors algorithm employs the maximal frequent itemset discovery and partitioning in order to define the number of desired clusters and the initial attractors of the centers of these clusters. The intuition is that a frequent itemset in the case of software metrics is a set of measurements that occur together in a minimum part of a software system's classes. Classes with similar measurements are expected to be on the same cluster. The term attractor is used instead of centroid, as it is not determined randomly, but by its frequency in the whole population of a software system's classes.

4. Application - Results Evaluation

The evaluation of Apache Geronimo's maintainability according to ISO/IEC-9126, involved the study of 1440 classes. Figure 3 depicts the clusters derived from clustering the maintainability values of Geronimo's classes. The higher the values on axis X the less maintainable the classes are. Table 1 presents statistics for the derived clusters.

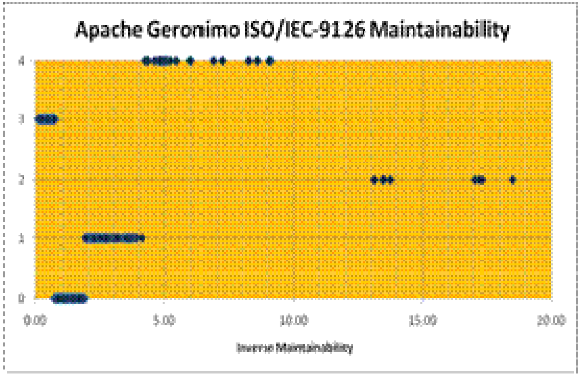

Figure 3: Apache Geronimo ISO/IEC-9126 Maintainability Clusters

Table 1: Clusters Statistics

S/N	Population	Percentage	Mean	Standard Deviation
0	419	29%	1.10	0.29
1	130	9%	2.45	0.60
2	7	0.004%	13.75	2.27
3	856	59%	0.39	0.16
4	28	1.996%	5.02	1.55

Cluster 3, which has the biggest population, contains classes that their maintainability values range between 0 and 0.9. This shows that the vast majority of Geronimo's classes are highly maintainable. Furthermore, clusters 0, 1 and 4 contain classes that their maintainability values range from 0.9 – 2, 2 - 4 and 4 – 9.2 respectively, which can be considered good in terms of maintainability.

However, outliers are detected in cluster 2, which consists of only seven (7) classes that have the lowest maintainability values. These classes are:

1. KernelManagementHelper.java, a class of 1024 Lines Of Code (LOC).
2. TradeDirect.java, a class of 2312 LOC.
3. ClientApp.java, a class of 1633 LOC.
4. CdrInputStream.java, a class of 1569 LOC.
5. CdrOutputStream.java, a class of 1241 LOC.
6. ASN1Encodable.java, a class of only 62 LOC.
7. DERobject.java, a class of only 38 LOC.

Table 2 presents the metric values for the classes in cluster 2. A further study on these values indicates that the classes in cluster 2 are grouped in two categories:

- The first category includes the first five classes that have the following characteristics:
- They don't follow the principle of low coupling/high cohesion. On the contrary they exhibit low cohesion and high coupling.
- They are highly complex.
- All of them have polymorphic methods; which indicates that encapsulation is not applied in these classes.
- The second category includes the classes ASN1Encodable and DERObject that are difficult to maintain for different reasons. More specifically these two classes have the following characteristics:
    - Interestingly they are not complex, and their size is very small unlike the classes on the first category. They also follow the principle of low coupling/high cohesion.
    - They have an excessive number of children. This indicates probably that these classes are fundamental elements of Apache Geronimo's structure.
    - The number of classes depending on them (Ca) is big.

Table 3 presents statistics for the metrics of Apache Geronimo's classes in clusters 0, 1, 3 and 4.

This table indicates that:

- The lower the metric values the higher the probability of low maintainability.
- There is limited use of inheritance as shown by the low DIT and NOC values.
- The majority of the classes follow the low coupling/high cohesion principle.
- Most of the classes exhibit low complexity.
- The design property of encapsulation is applied to most of the classes.

5. Conclusions and Future Work

The application of the proposed methodology has been proved to be time and performance efficient. The extraction process, which is the most time-consuming part of this methodology, analyzed the 1440 classes of Apache Geronimo 1.0 and stored the corresponding metrics and elements in a limited amount of time. A domain expert previewed the stored metrics and assigned easily and efficiently the corresponding weights, according to his priorities and concernings. After clustering application, the resulted clusters proved to be representative of the code artifacts, helping the domain expert to identify relations between specific metrics and global maintainability as well as spot individual outlier classes that may need reconsideration.

As future work, we intend to enhance our extraction method by calculating metrics from other languages like C++, C and COBOL which were used for the development of the majority of legacy systems, a category of software systems which is very interesting in terms of program comprehension and maintainability evaluation.

Acknowledgements

This research work has been partially supported by the Greek General Secretariat for Research and Technology (GSRT) and Dynacomp S.A. within the program "P.E.P. of Western Greece Act 3.4"

Table 2: Cluster 2 Metrics

S/N	WMC	NPM	DAM	CBO	POM	DIT	NOC	LCOM	Ca
1	9.15	11.13	1.62	17.40	40.00	0.72	0.00	42.69	0.00
2	11.58	4.52	1.62	35.65	30.00	0.72	0.00	45.21	0.51
3	10.68	0.32	1.62	2.99	2.50	0.72	0.00	81.97	2.53
4	18.38	11.45	1.62	14.37	20.00	0.72	0.00	64.96	9.61
5	14.77	11.29	1.62	13.14	12.50	0.72	0.00	47.82	9.61
6	0.42	0.81	0.00	0.33	0.00	0.72	149.49	0.27	26.30
7	0.28	0.48	0.00	0.00	0.00	1.44	76.27	0.18	52.10

Table 3: Cluster 0, 1, 3 and 4 Metrics Statistics

	Min.	Max.	Mean	Median	Stand. Dev.
WMC	0.07	12.55	0.96	0.55	1.20
NPM	0.00	8.71	0.98	0.65	1.17
DAM	0.00	1.62	1.00	1.62	0.76
CBO	0.00	16.54	0.95	0.41	1.54
POM	0.00	37.50	0.93	0.00	2.88
DIT	0.72	3.60	1.00	0.72	0.49
NOC	0.00	70.17	0.85	0.00	3.87
LCOM	0.00	26.84	0.81	0.11	2.43
Ca	0.00	81.94	0.93	0.00	3.28

References

[1] N. Anquetil and T. C. Lethbridge, “Experiments with Clustering as a Software Remodularization method”, Proc. 6th Working Conf. Reverse Engineering (WCRE 99), IEEE Comp. Soc. Press, (1999) 235-255.

[2] Erik Arisholm, Lionel C. Briand, Audun Foyen, “Dynamic Coupling Measurement for Object-Oriented Software”, IEEE Transactions on Software Engineering, vol. 30, No. 8, August 2004, pp. 491-506.

[3] Dunham, M. H. *Data Mining: Introductory and Advanced Topics*. Prentice Hall PTR, 2002.

[4] Kanellopoulos Y., Antonellis P. Tjortjis C., Makris C., “k-Attractors, A Clustering Algorithm for Software Measurement Data Analysis”, In Proceedings of IEEE 19th International Conference on Tools for Artificial Intelligence (ICTAI 2007), IEEE Computer Society Press 2007

[5] Kan, S. H. *Metrics and Models in Software Quality Engineering*. Addison-Wesley. Second Edition. 2002.

[6] Jay Kothari, Ali Shokoufandeh, Spiros Mancoridis, Ahmed E. Hassan, "Studying the 1Evolution of Software Systems Using Change Clusters," *ICPC*, pp. 46-55, 14th IEEE International Conference on Program Comprehension (ICPC'06), 2006.

[7] T. Kunz and J. P. Black, “Using Automatic Process Clustering for Design Recovery and Distributed Debugging”, IEEE Transactions on Software Engineering, 21(6), (1995) 515-527.

[8] Dawn J. Lawrie, Henry Feild, David Binkley, "Leveraged Quality Assessment using Information Retrieval Techniques," *ICPC*, pp. 149-158, 14th IEEE International Conference on Program Comprehension (ICPC'06), 2006.

[9] S. Mancoridis, B.S. Mitchell, Y. Chen and E.R. Gansner, “Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures”, *Proc. Int'l Conf. Software Maintenance (ICSM 99)*, IEEE Comp. Soc. Press, (1998) 50-59.

[10] O. Maqbool, H.A. Babri, A. Karim, and M. Sarwar, “Metarule-guided association rule mining for program understanding, Software”, IEEE Proceedings, 152(6) (2005) 281- 296.

[11] Storey Margaret-Anne: “Theories, Methods and Tools in Program Comprehension: Past, Present and Future”, *Proc. IEEE 13th Int’l Workshop Program Comprehension* (IWPC 2005), 2005.

[12] National Institute of Standards and Technology (NIST), “The Economic Impacts of Inadequate Infrastructure for Software Testing.”, Washington D.C. 2002.

[13] C. M. de Oca and D. L. Carver, “Identification of Data Cohesive Subsystems Using Data Mining Techniques”, Proc. Int'l Conf. Software Maintenance (ICSM 98), IEEE Comp. Soc. Press, (1998) 16-23.

[14] Rajendra K. Bandi, Vijay K. Vaishnavi, Daniel E. Turk, “Predicting Maintenance Performance Using Object Oriented Design Complexity Metrics”, IEEE Transactions on Software Engineering, vol. 29, No. 1, January 2003, pp. 77-87.

[15] D. Rousidis and C. Tjortjis, “Clustering Data Retrieved from Java Source Code to Support Software Maintenance: A Case Study”, *Proc IEEE 9th European Conf. Software Maintenance and Reengineering (CSMR 05)*, IEEE Comp. Soc. Press, (2005) 276-279.

[16] K. Sartipi, K. Kontogiannis and F. Mavaddat, “Architectural Design Recovery Using Data Mining Techniques”, *Proc. 2nd European Working Conf. Software Maintenance Reengineering (CSMR 00)*, IEEE Comp. Soc. Press, (2000) 129-140.

[17] Spinellis D: “Code Quality: The Open Source Perspective“, Addison-Wesley, 2006.

[18] Yong Tan, Vijay S. Mookerjee, “Comparing Uniform and Flexible Policies for Software Maintenance and Replacement”, IEEE Transactions on Software Engineering, vol. 31, No. 3, March 2005, pp. 238-255.

[19] C. Tjortjis, N. Gold, P.J. Layzell and K. Bennett, ”From System Comprehension to Program Comprehension”, *Proc. IEEE 26th Int’l Computer Software Applications Conf. (COMPSAC 02)*, IEEE Comp. Soc. Press, (2002) 427-432.

[20] C. Tjortjis C., L. Sinos and Layzell P.J., “Facilitating Program Comprehension by Mining Association Rules from Source Code”, *Proc. IEEE 11th Int’l Workshop Program Comprehension (IWPC 03)*, IEEE Comp. Soc. Press, (2003) 125-132.

[21] V. Tzerpos and R. Holt, “Software Botryology: Automatic Clustering of Software Systems”, *Proc. 9th Int'l Workshop Database Expert Systems Applications (DEXA 98)*, IEEE Comp. Soc. Press, (1998) 811-818.

[22] C. Xiao and V. Tzerpos, “Software Clustering on Dynamic Dependencies”, *Proc. IEEE 9th European Conf. Software Maintenance and Reengineering (CSMR 05)*, IEEE Comp. Soc. Press, (2005) 124-133.

[23] http://geronimo.apache.org/downloads.htm

[24] http://www.code4thought.org

[25] Saaty T.. Multicriteria Decision Making: The Analytic Hierarchy Process, Vol. 1, AHP Series, RWS Publications, 502 pp., 1990