

A Data Mining Methodology for Evaluating Maintainability according to ISO/IEC-9126 Software Engineering-Product Quality Standard

Antonellis P.¹, Antoniou D.¹, Theodoridis E.¹, Kanellopoulos J.², Makris C.¹, Tjortjis C.², Tsirakis N.¹.

¹Department of Computer Engineering and Informatics, University of Patras
And

²University of Manchester, UK

Abstract

This paper presents ongoing work on using data mining to evaluate a software system's maintainability according to the ISO/IEC-9126 quality standard. More specifically it proposes a methodology for knowledge acquisition by integrating 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-Means clustering 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 a very important and complex stage in software lifecycle typically consuming 50-70% of the total effort allocated to a software system according to [01], [15]. Given this fact, 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 [18], [1], [2]. According to ISO/IEC-9126 [ref International Standards Organization and International Electrotechnical Commission] maintainability is the capability of a software product to be modified [0]. Evaluating such a characteristic is a difficult process as many contradictory criteria must be considered in order to reach a decision [12]. On the other hand data mining and its capacity to deal with large volumes of data and to uncover hidden patterns has been proposed as a means to support the evaluation and assessment of the maintainability of industrial scale software systems [0], [0].

The scope of this work is to present 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 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.
- Data mining clustering is 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, 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 proposed method for extracting elements and metrics from Java source code, the assignment of relative weights by employing multicriteria analysis methods and the clustering method. 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

Developing software systems of any size which do not need to be changed is unattainable [0]. Such systems, once in use, need to be functional and flexible in order to operate correctly and fulfill their mission, as new requirements emerge. Consequently, software systems remain subject to changes and maintenance throughout their lifetime. Software inspection and evaluation is required by maintenance engineers in

order to identify problematic files or modules; and to assess their maintainability [0].

2.1 Previous Work

Data mining and its ability to deal with vast amounts of data, has been considered a suitable solution in assisting software maintenance, often resulting in remarkable results [0, [0, [21].

Data mining techniques have been used previously, for identification of subsystems based on associations (ISA methodology) [0]. This approach provides a system abstraction up to the program level as it produces a decomposition of a system into data cohesive subsystems by detecting associations between programs sharing the same files.

Clustering has also been used to support software maintenance and systems knowledge discovery. A method for grouping Java code elements together according to their similarity was proposed in [0]. It focuses on achieving a high level system understanding. The method derives system structure and interrelationships, as well as similarities among systems components, by applying cluster analysis on data extracted from source code. Hierarchical Agglomerative Clustering was employed to reveal similarities between classes and other code elements thus facilitating software maintenance and Java program comprehension.

An approach for the evaluation of dynamic clustering was presented in [0]. The scope of this solution was to evaluate the usefulness of providing dynamic dependencies as input to software clustering algorithms. This method was applied to Mozilla, a large open source software system with more than four million lines of C/C++.

All these approaches employ data mining techniques only to recover the structure of a software system. On the other hand [0] is employing clustering for predicting software modules' fault proneness and potential noisy modules. k-Means and Neural – Gas algorithms were employed in order to group together modules with similar software measurements. A software engineering expert inspected the derived clusters and labelled them as fault prone or not.

Finally [0] presents a methodology that uses clustering for both recovering the structure of a software artifact and assessing its maintainability. It does that by creating an input model which considers as program's entities' attributes both metrics (e.g.

Chidamber and Kemerer metrics suite) and elements from source code data (e.g. class name, method name, superclass etc.).

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.

ISO/IEC-9126 Maintainability

According to ISO/IEC-9126, maintainability is the capability of the software to be modified. These modifications can be corrective, adaptive or perfective in order for software to comply with new requirements and functional specifications [0]. They also can happen without scheduling (emergency). Maintainability is characterized by the following five sub-characteristics:

- Analysability, which defines the ability of software to be diagnosed for deficiencies or causes or failures or for parts to be modified [0].
- Changeability, which defines how easy, is to perform a specified modification [0].
- Stability, which shows how capable software is to remain stable after being modified [0].
- Testability, which defines the ability of software to be validated [0].
- Maintainability Compliance, which defines how easy, is for the software to comply with standards or conventions relating to maintainability [0].

This set of maintainability's sub-characteristics can be classified in a hierarchical tree which consists of the characteristic of maintainability, its sub-characteristics and metrics. Maintainability is on the tree's highest level while metrics are on the lowest. Maintainability is analysed in sub-characteristics which in turn can be evaluated by using metrics. This is depicted on the following figure.

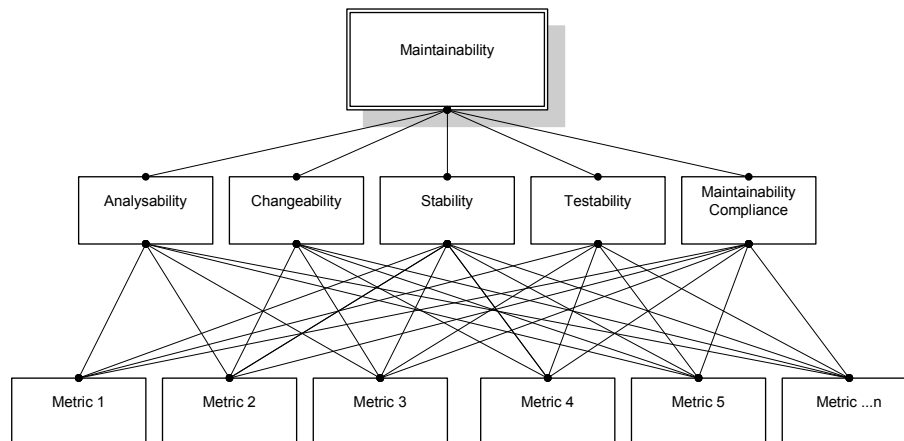


Figure 1: ISO/IEC-9126 Maintainability Hierarchy

Multicriteria Analysis

Multicriteria Analysis (MA) is a procedure aiming at supporting decision makers whose problem involves numerous and conflicting evaluations with many and diverse parameters. As a basic principle, MA highlights these conflicts and facilitates finding a way to come to a compromise in a transparent process. A very common example is the decisions made by a government. Before a new policy applies; several types of after-effects such as morals and ethics, economy and environmental protection are taken in mind. The implementation of MA using certain elements of subjectiveness makes recommendations that approach the expected results with more accuracy. The ethical point is very important; for example, when one is making a decision that seriously impact on other people as opposed to a personal decision. Some of the MA models are [4]:

- Analytic Hierarchy Process (AHP)
- Multi-Attribute Global Inference of Quality (MAGIQ)
- Goal Programming
- ELECTRE (Outranking)
- Data Envelopment Analysis

In our problem's context, evaluating the software quality parameters reduce the problem to a multicriteria problem. According to the software's objectives some notion of subjectiveness is needed, in our case study maintainability, in order to evaluate properly all the ISO/IEC-9126 software engineering quality standard. We have adopted analytic hierarchy process (AHP) [14], presented in a following section, for purposes of simplicity and accuracy of the method.

3. Description of the proposed Methodology

This section presents the proposed methodology which:

- Extracts elements and metrics from Java source code.
- Assigns weights to the selected metrics in order to reflect their importance on evaluating a system's maintainability according to ISO/IEC-9126 quality standard.
- Applies the clustering data mining technique on the derived maintainability values.

3.1 Data Extraction Process

The objective of the data extraction process is two-fold:

- To extract appropriate elements and metrics from Java source code. Elements include native source code attributes and help describing the software architecture and its characteristics. On the other hand, metrics depict quantitatively, thoroughly and more effectively the system under evaluation.
- To aggregate the extracted data, choose a refined subset of them, and store it in a relational database system for further analysis and evaluation.

Native attributes depend on the system's programming paradigm and may include:

- ids and names of packages
- class-files, definition files, names of classes
- structure blocks (e.g. loop blocks, if blocks, exception blocks, etc)

Moreover, metrics provide additional and valuable information concerning the system under evaluation. This information will help further describe the software system's entities. In §3.4 we further describe the selected metrics and we justify why we think they are important in facilitating the evaluation of a system's

maintainability according to ISO/IEC-9126. All of the above collected elements and metrics are stored permanently into appropriate structured XML files, with every XML file corresponding to a source code file.

For simplicity reasons, not all of the above described elements are analyzed. Some of them are more important and describe more effectively the characteristics of the system under evaluation, while others contain very detailed information that is not needed. For this reason, a refined subset of the extracted data is chosen to be stored for further analysis and elaboration. This subset should be fine-grained in order to provide the flexibility to easily assess the software system and on the same time to contain all the necessary information for this evaluation. Based on this requirement, only the calculated metrics and their associated class elements are chosen to be stored and further analyzed. All the detailed elements are discarded because the information that provide exists chunked in the calculated metrics.

The chosen elements and metrics need to be extracted and stored permanently for further analysis and evaluation. For this reason, a relational database system was chosen, as it provides ease of use, reliability and effective search over the stored information. As mentioned before, all of the collected elements are at first stored into XML files, with every XML file corresponding to a source code file. The extraction method must be transparent to the underlying relational database, for portability reasons. That is why it employs cutting-edge software tools that facilitate the mapping of XML elements and nodes into

any relational database. This mapping provides the ability for the proposed methodology to be independent of any underlying relational database. Figure 2 depicts the general architecture of data extraction and preparation module.

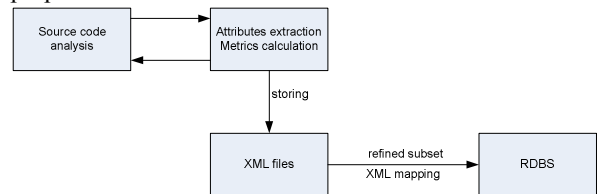


Figure 2 - Architecture of data extraction and preparation module

3.2 Metrics Description

As software demonstrates regular behaviour and trends, these can be measured [9]. The evaluation of a system's maintainability according to ISO/IEC-9126 requires the collection of such metrics in order to provide a systematic approach for code evaluation based on a set of predefined rules. This enables the stakeholders of a software product (i.e., management, software engineers, end users) to track status, control costs, and make decisions related to their tasks. These metrics can also be useful as indicators for identifying potential problematic areas.

For this work we decided to employ the following metrics suite, as it can be applied to OO programs and can be used as a predictor and evaluator of maintenance effort. The following table (Table 1) presents the set of selected metrics.

Table 1: Set of Selected Metrics

S/N	Metric	Formula	Purpose	Entity Applied
1	Weighted Methods Per Class (WMC)	WMC=S S=Sum of the cyclomatic complexity of each member method of the class	It provides a measure for predicting class's maintainability and reusability [0]. The more complex a class is the more difficult is to maintain and reuse.	Class
2	Number Of Public Methods (NPM)	NPM=A A=Number of a class's methods declared as public	It measures the size of an API provided by a package.	Class
3	Data Access Metric (DAM)	DAM=R R=Ratio of the number of private and protected member data to the total number of member data declared in a class [Error! Reference source not found.].	It reflects how well the property of encapsulation is applied on a class.	Class
4	Coupling Between Objects(CBO)	CBO=A A=Number of invocations of other class's member methods or instance variables.	It represents the number of classes coupled to a given class.	Class

S/N	Metric	Formula	Purpose	Entity Applied
5	Number of Polymorphic methods (NOP)	NOH=A A=Number of the member methods that exhibit polymorphic behavior.	It is a measure of the overridden (or virtual) methods of an Object Oriented software system.	Class
6	Depth Of Inheritance Tree (DIT)	DIT=A A=Distance from top	It provides for each class a measure of the inheritance levels from the object hierarchy top	Class
7	Number Of Children (NOC)	NOC=A A=Number of the immediate descendants of the class.	It measures the number of the immediate descendants of the class	Class
8	Lack of Cohesion in Methods (LCOM)	LCOM=A A=Number of member methods of a class C that access the same member data [0].	It measure if a class of the system has all its methods working together in order to achieve a single, well-defined purpose.	Class
9	Afferent (inward) Coupling (Ca)	Ca=A A=Number of packages depend on the package under examination	It measures the number of packages that depend on the package under examination [10].	Class

3.3 Weights Assignment

As mentioned in §2.3, 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 [14]. 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 eigenvalues are computed. These eigenvalues play the role of coefficients/weights when someone wants to evaluate the alternatives for the examined objectives.

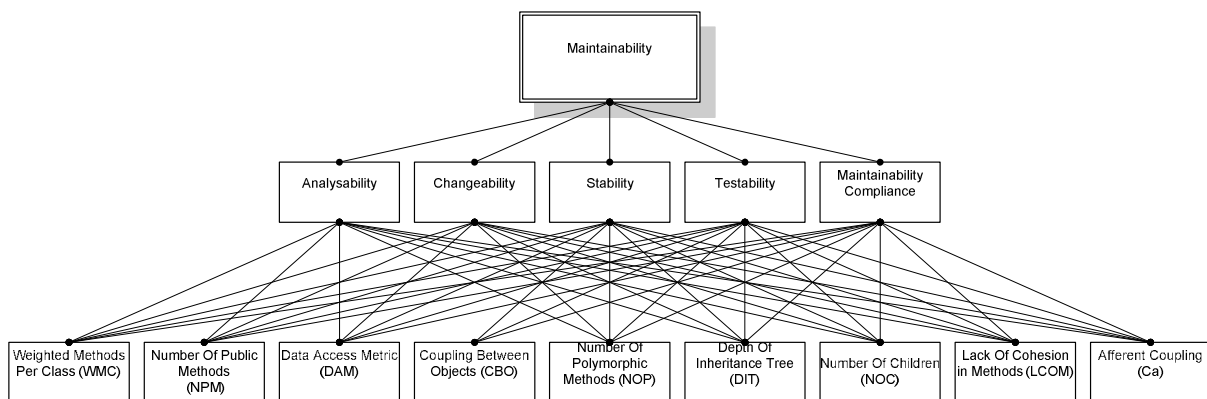


Figure 3: ISO/IEC-9126 Maintainability Metrics Hierarchy

In our case when we aim at evaluate maintainability from a set of employed metrics (see previous section and Figure 3.). 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 eigenvalues 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 eigenvalues extraction.

There are several possibilities for evaluating different systems:

- Evaluation of all the systems utilizing the same weights produced by the subjectivity of one software expert.
- Evaluation of all the systems utilizing the average weights produced by the subjectivity of several software experts.
- Evaluation of each system independently utilizing the weights produced by the subjectivity of a software expert that focuses to the characteristics of its system.

3.4 Clustering Maintainability Values

The primary objective of employing the clustering data mining technique is to facilitate a software system's evaluator to obtain a general but illuminating view of it that may lead him/her to draw useful conclusions concerning its maintainability. Clustering in general, is useful for Similarity/Dissimilarity analysis; in other words it analyzes what data points in a given dataset are close to each other. In our case, mutually exclusive groups of classes, member data or methods are created according to their similarities, and hence the time required to assess the maintainability of a software system is reduced. This is helpful especially when the system under evaluation is very large. A project manager or an engineer might only be an expert of a part of the system and cannot state of the quality of the whole of it. Another contribution of using the clustering technique is that it helps discovering programming patterns and "unusual" or outlier cases which may require attention. Clustering also facilitates the dimensionality reduction, since high dimensional data are replaced with a group (cluster) label.

For this purpose, k-Means clustering has been chosen to be implemented and performed. The particular algorithm is a simple clustering method that has the following characteristics:

- It shows optimal results.
- It is general, as it can work for any distance desired and requires no training phase.
- Finally, the algorithm's speed is very appealing in practice as well, especially in the case of a large number of variables.

k-Means clustering is a commonly used partitioning algorithm. Each cluster is represented by the mean value of the objects in the cluster. As a result, cluster similarity is measured based on the distance between the object and the mean value of the input data in a cluster. It is an iterative algorithm in which objects are moved among clusters until a desired set is reached. The steps of the algorithm can be described as follows [3]:

Given a set of n objects t_1, t_2, \dots, t_n and a number k of desired clusters

assign initial values for means m_1, m_2, \dots, m_k

repeat

assign each item t_i to the cluster with the closest mean;

calculate new cluster mean;

until means m_1, m_2, \dots, m_k do not change

The squared-error criterion is used to measure the sum of the squares of the distance between each object and the mean. The sum should be minimized in order to obtain a good clustering result. It is obvious that the smaller the sum, the more tightly the objects are clustered around the mean value (centroid), and clustering is more precise. The squared-error criterion can be expressed by the formula in Eq. (1):

$$SSE = \sum_{i=1}^k \sum_{p \in C_i} dist(c_i, p)^2$$

where dist is the standard Euclidean (L2) distance between two objects in Euclidean space; p is an object belonging to the ith cluster C_i , and c_i is the mean of the cluster. The algorithm is suitable for discovering spherical-shaped clusters in small to medium size databases. However, its main problems are that it is sensitive to noise and to the initial partitioning. As many possible initial partitions lead to many different results, the final clustering is influenced by the initial partition, which is indicated by the user input [5].

4. Results Evaluation

The evaluation of Apache Geronimo's maintainability according to ISO/IEC-9126, involved the study of 1440 classes. Figure 4 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 2 presents statistics for the derived clusters.

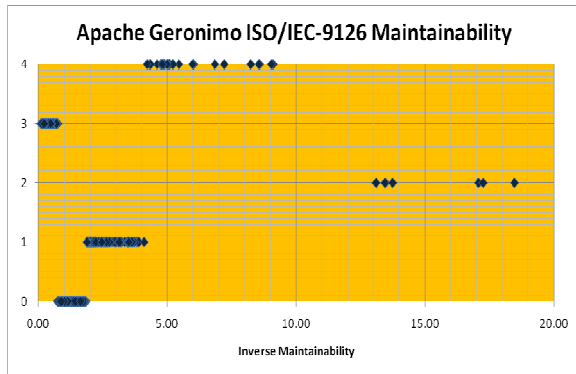


Figure 4: Apache Geronimo ISO/IEC-9126 Maintainability Clusters

Table 2: 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

Table 3: 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

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:

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 3 presents the metric values for the classes in cluster 2.

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

Table 4: 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

	Min.	Max.	Mean	Median	Stand. Dev.
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

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

This section presents conclusions drawn by evaluating the proposed methodology. Directions for future work are also discussed here.

5.1 Conclusions concerning the Methodology

The aim of this work was to present a methodology that facilitates the evaluation of a software product's maintainability according to the ISO/IEC-9126 software engineering quality standard. The first step toward that was to develop a two-level extraction process that collects appropriate elements and metrics from Java source code, aggregates them and stores a refined subset of the aggregated data in a relational database system for further analysis and evaluation. During the first step the collected data are stored permanently into appropriate structured XML files, with every XML file corresponding to a source code file. At the second step, the more important elements and metrics are chosen to form a refined subset that is mapped transparently into a relational database system.

The second step towards our goal was the adoption of the Analytic Hierarchy Process (AHP) for weights assignment to the collected metrics, in order to reflect their importance on evaluating a system's maintainability according to ISO/IEC-9126 software engineering quality standard. The weight assignment process constructs a pair wise comparison for the set of employed metrics and by applying the normalization and extraction of eigenvalues it finds the weight of each metric for calculating a maintainability score.

The last step was the application of k-Means clustering algorithm on the derived ISO/IEC-9126's

maintainability values, in order to facilitate a software system's evaluator to obtain a general but illuminating view of it that may lead him/her to draw useful conclusions concerning its maintainability. During the clustering process, mutually exclusive groups of classes, member data or methods are created according to their similarities, and hence the time required to assess the maintainability of a software system is reduced.

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 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 concerns. 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.

5.2 Future Work

We consider the following various alternatives in order to enhance the proposed methodology:

5.2.1 Systems' components clustering based on their dynamic dependencies. This research work presented the analysis of static dependencies between the components of the system. It would be of great interest to attempt to evaluate the usefulness of analysing the dynamic dependencies of a software system's artifacts.

5.2.2 Integration of more data mining algorithms. The proposed framework integrates the -Means algorithm. However it may be useful if more custom data mining algorithms were integrated in this framework. This would result in a complete system for automated program and system comprehension. An example is the integration of hierarchical clustering algorithms that they do not need the user to define the number of the desired output clusters.

5.2.3 Tune the methodology for other programming languages. The proposed methodology processes information derived only from Java source code files (.java). It is of great interest to extract information from other programming languages like C, C++, Cobol and Borland Delphi.

References

1. Arisholm E., 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
2. Bandi R., 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
3. Dunham M.H., Data Mining, Introductory and Advanced Topics, Prentice Hall, 2002.
4. Figueira, Jose; Greco, Salvatore; Ehr Gott, Matthias Multiple Criteria Decision Analysis: State of the Art Surveys Series: International Series in Operations Research & Management Science , Vol. 78 Figueira, Jose; Greco, Salvatore; Ehr Gott, Matthias (Eds.) 2005.
5. Han J. , M. Kamber, Data Mining: Concepts and Techniques, Academic Press, 2001.
6. ISO/IEC 9126-1, Software Engineering – Product Quality International Standard, Geneva 2001
7. Kanellopoulos Y., Dimopoulos T., Tjortjis C. and Makris C., "Mining Source Code Elements for Comprehending Object-Oriented Systems and Evaluating Their Maintainability" to appear at the ACM SIGKDD Explorations v8.1, Special Issue on Successful Real-World Data Mining Applications, June 2006.
8. Kanellopoulos Y., Makris C. and Tjortjis C., "An Improved Methodology on Information Distillation by Mining Program Source Code", to appear at Elsevier Data & Knowledge Engineering, 2006.
9. Lehman M.M., "Programs, Life Cycles, and Laws of Software Evolution", Proc. IEEE, Vol. 68, No 9, 1980, pp. 1060 - 1076.
10. Oca C. M. and D. L. Carver, "Identification of Data Cohesive Subsystems Using Data Mining Techniques", Proc. Int'l Conf. Software Maintenance (ICSM 98), 1998, pp.16-23.
11. Pigoski T.M., Practical Software Maintenance: Best Practices for Managing your Software Investment, Wiley Computer Publishing, 1996.
12. Roger P.S., "Software Engineering, A Practitioner's Approach", McGraw – Hill International Edition 2005.
13. Rousidis D., C. Tjortjis, "Clustering Data Retrieved from Java Source Code to Support Software Maintenance: A Case Study", Proc. IEEE 9th European Conf. Software Maintenance Reengineering (CSMR 05), 2005, pp. 276-279
14. Saaty T.. Multicriteria Decision Making: The Analytic Hierarchy Process, Vol. 1, AHP Series, RWS Publications, 502 pp., 1990
15. Sommerville, Software Engineering, 6th ed., Harlow, Addison-Wesley, 2001.
16. Spinellis D: "Code Quality: The Open Source Perspective", Addison-Wesley, 2006
17. Stamelos I., Vlahavas I., Refanidis I., Tsoukias A.: "Knowledge Based Evaluation of Software Systems: A Case Study", Elsevier Information and Software Technology, Vol. 42, No 5, April 2000, pp. 333-345
18. Sutherland J., "Business Objects in Corporate Information Systems", ACM Computing Survey, vol. 27, 1995, pp 274-276
19. Tjortjis C., L. Sinos and P.J. Layzell, "Facilitating Program Comprehension by Mining Association Rules from Source Code", Proc. IEEE 11th Int'l Workshop Program Comprehension (IWPC 03), 2003, pp. 125-132.
20. Xiao C., V. Tzerpos, "Software Clustering on Dynamic Dependencies", Proc. IEEE 9th European Conf. Software Maintenance Reengineering (CSMR 05), 2005, pp. 124-133.
21. Zhong S., T.M. Khoshgoftaar, and N. Seliya, "Analyzing Software Measurement Data with Clustering Techniques", IEEE Intelligent Systems, Vol. 19, No. 2, 2004, pp. 20-27.