Interpretation of Source Code Clusters in Terms of the ISO/IEC-9126 Maintainability Characteristics

Yiannis Kanellopoulos[1,2] Ilja Heitlager[3]
Christos Tjortjis[1] Joost Visser[3]

[1]*University of Manchester, School of Computer Science, UK*
[2]*University of Patras, Computer Engineering and Informatics Department, Greece*
[3]*Software Improvement Group, The Netherlands*
Yiannis.Kanellopoulos@postgrad.manchester.ac.uk i.heitlager@sig.nl
Christos.Tjortjis@manchester.ac.uk j.visser@sig.nl

Abstract

Clustering is a data mining technique that allows the grouping of data points on the basis of their similarity with respect to multiple dimensions of measurement. It has also been applied in the software engineering domain, in particular to support software quality assessment based on source code metrics. Unfortunately, since clusters emerge from metrics at the source code level, it is difficult to interpret the significance of clusters at the level of the quality of the entire system. In this paper, we propose a method for interpreting source code clusters using the ISO/IEC 9126 software product quality model. Several methods have been proposed to perform quantitative assessment of software systems in terms of the quality characteristics defined by ISO/IEC 9126. These methods perform mappings of low-level source code metrics to high-level quality characteristics by various aggregation and weighting procedures. We applied such a method to obtain quality profiles at various abstraction levels for each generated source code cluster. Subsequently, the plethora of quality profiles obtained is visualized such that conclusions about different quality problems in various clusters can be obtained at a glance.

1. Introduction

Reliable assessment of the quality of large software-intensive systems is of prime importance to IT management for making decisions about planning, budgeting, and controlling the processes of developing, delivering, maintaining, and operating these systems. The main challenge of methods and techniques for such software product quality assessment is to translate technical findings on the level of source code artifacts into meaningful appraisals of characteristics of the system as a whole. For example, if a certain percentage of the source code lines are part of a method with cyclomatic complexity over 50, then what does this mean for the testability aspect of the system's overall quality? Though it may seem tempting to look for a single number to capture system quality as a whole, a more practical approach is to provide an abstract, but informative characterization of weak and strong parts of the software for a selected number of high-level quality aspects.

A breakdown of the concept of software product quality into sub-characteristics is offered by the ISO/IEC 9126 international standard for software product quality [5]. The first part of this standard provides a quality model that breaks software product quality up into six characteristics, which are then further decomposed into a total of 27 sub-characteristics. This consensual identification and definition of software quality characteristics provides a useful frame of reference and standardized terminology which facilitates communication and thought concerning software quality.

Several methods have been proposed to perform assessment of quality aspects according to ISO/IEC 9126 for a given software system, i.e. to come to a judgment regarding these characteristics at the system level from concrete findings at the technical level via a repeatable and objective process [14], [10], [16], [12]. Many of these methods focus on the *maintainability* characteristic and its sub-characteristics [4], [1], [2]. Some methods are tailored to the object-oriented paradigm [1], while others are paradigm-agnostic [4]. Each of these methods employs more or less sophisticated instruments for aggregating and summarizing technical data. In all cases, the final quality judgments are presented in terms of the characteristics and sub-characteristics of the ISO/IEC 9126 quality model.

Apart from judgments on the level of an entire system, software quality assessment may call for a

course-grained subdivision of the system into parts with different quality profiles. Though structural subdivision is the bread-and-butter of the software quality engineer, it is not the most appropriate instrument for quality assessment for several reasons. Firstly, structural subdivisions may group together elements that are strongly dissimilar in terms of their quality. Secondly, along the lifetime of a system, its structure often degrades significantly, to the point of losing its usefulness for system comprehension. In fact, a software quality engineer may need to invest substantial effort to discover system structure at all.

As an alternative to structural subdivision, *clustering* can be used. This is a data mining technique that allows the grouping of data points on the basis of their similarity with respect to multiple dimensions of measurement [6]. It is a discovery-driven data analysis technique, rather than a verification-driven technique, which makes it particularly useful in problems where there is little prior information available about the data. It can be applied in the context of software quality assessment to group the elements of a software system according to their similarities in terms of technical (source code) measurements [17].

Unfortunately, the source code clusters that are discovered by the application of a clustering algorithm may be meaningful at the level of the source code metrics from which they are computed, but they are difficult to interpret at the level of quality aspects. For example, when a set of Java classes are grouped together on the basis of their values for a suite of object-oriented design metrics, what does their similarity in terms of these values mean for their changeability or testability?

In this paper, we explore the possibility of attaching meaning to source code clusters using a method for software quality assessment in terms of ISO/IEC 9126. In particular, we will apply a recent improvement of the k-Means clustering algorithm, called the k-Attractors algorithm [8], to derive clusters from source code metrics. Subsequently, we apply a method for software quality assessment – an improvement over the methods of [4] and [1] – to interpret these metrics at the level of source code properties and finally at the level of ISO/IEC 9126 maintainability sub-characteristics. We have applied and validated the proposed approach in an industrial case study.

The paper is organized as follows. Section 2 presents a formal statement of the problem under investigation. Section 3 discusses the necessary background information on the various ingredients of our proposal, such as cluster analysis, analytical hierarchical processing and the ISO/IEC 9126 quality model. Section 4 discusses our proposed approach that combines these ingredients. Section 5 presents empirical results from the analysis of a case study and discusses the benefits and risks of our methodology. Section 6 presents related work in the area of software quality evaluation. Finally, section 7 concludes with a summary of our work and indicates directions for future work.

2. Formal Problem Statement

The challenge addressed in this paper is to interpret technical measurement data about the units of a large software system into high-level appraisals of the quality characteristics of its main parts. Typically, the units are individual methods or functions in the system. The main parts, in our approach, will typically be a handful of groups of such units.

A concise formalization of our problem is given by the following function signature:

$$appraisal : v^{M \times U} \rightarrow n^{B \times Q \times G}$$

where:

U: units in the system
M: measurements performed on units
G: groups of units
Q: high-level quality characteristics
B: bins in a quality profile

Thus, the input of our problem is a two-dimensional matrix of measurement values. For each unit in *U* and each measurement *M* there is a measurement value *v*. Measurements are typically source code metrics such as lines of code or cyclomatic complexity. Units can be methods, classes, etc.

The output is a three dimensional matrix of counts. These three dimensions are best understood when we look at the output as a two-dimensional matrix of vectors:

$$n^{B \times Q \times G} = (n^{B})^{Q \times G}$$

Each vector n^B is what we call a "quality profile". Each quality profile is a vector of counts that summarizes the quality of a given group *G* of units for a given quality characteristic *Q*. In this paper, we will use histograms as quality profiles, and by *B* we denote the bins of the histogram.

Given this formalization of the problem, we will proceed to discuss the individual techniques that we will combine to assemble the overall translation.

3. Background

To resolve the problem formalized above, we have adopted and combined several techniques,

including a 2-step mapping of source code metrics to ISO/IEC 9126 quality characteristics, analytical hierarchical processing, and clustering.

3.1 k-Attractors Clustering

Clustering is a technique known from data mining. On the basis of the similarity of data points with respect to multiple dimensions of measurement, clustering allows grouping of these points [6]. Clustering is a discovery-driven data analysis technique, rather than a verification-driven technique, which makes it particularly useful in problems where there is little prior information available about the data.

In the case of software quality 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 [17]. This helps reducing the time required to understand and evaluate the overall system. Another contribution of this method 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 like measurements from source code [8]. 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.

Clustering can be formalized by the following function signature:

$$cluster : v^{M \times U} \rightarrow v^{M \times U \times G}$$

Thus, a matrix of measurement values is partitioned into a list of such matrices. Each matrix represents a cluster of items that are similar in terms of their measurement values.

3.2 Analytic Hierarchical Processing

AHP is a decision-making technique that reduces complex multi-criterion decisions to a series of one-on-one comparisons [15]. Compared to other techniques, like ranking or rating techniques, AHP leverages the human ability to compare single properties of alternatives. From the result of a large number of one-on-one comparisons, a smaller number of weights for each criterion is synthesized.

At first let us assume that a set of objectives has been established. Then we are trying to establish a normalized set of weights to be used when comparing alternatives using these objectives. AHP forms a pair wise 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 objectives under examination.

A concise formalization of AHP is given by the following function signature:

$$ahp : a^{O \times O} \rightarrow w^{O}$$

Here, *O* is the number of criteria or objectives. Thus, a square matrix of relative importance scores *a* for each pair of criteria is transformed into a vector of weights *w*. One weight is obtained for each criterion.

After weights have been obtained by AHP, they can be applied to aggregate input criteria into a weighted score, with the following function:

$$score : w^{O} \times x^{O} \rightarrow y$$
$$score([w_1,..,w_O],[x_1,..,x_O]) = w_1 x_1 + \ldots + w_O x_O$$

Here, criteria *x* are mapped to a score *y*.

Below, we will use AHP to obtain high-level quality characteristics from low-level measurement data in a weighted manner.

3.3 Quality appraisal based on ISO/IEC 9126

Part 1 of the ISO/IEC 9126 international standard for software product quality [5] describes a model that dissects the notion of software product quality into

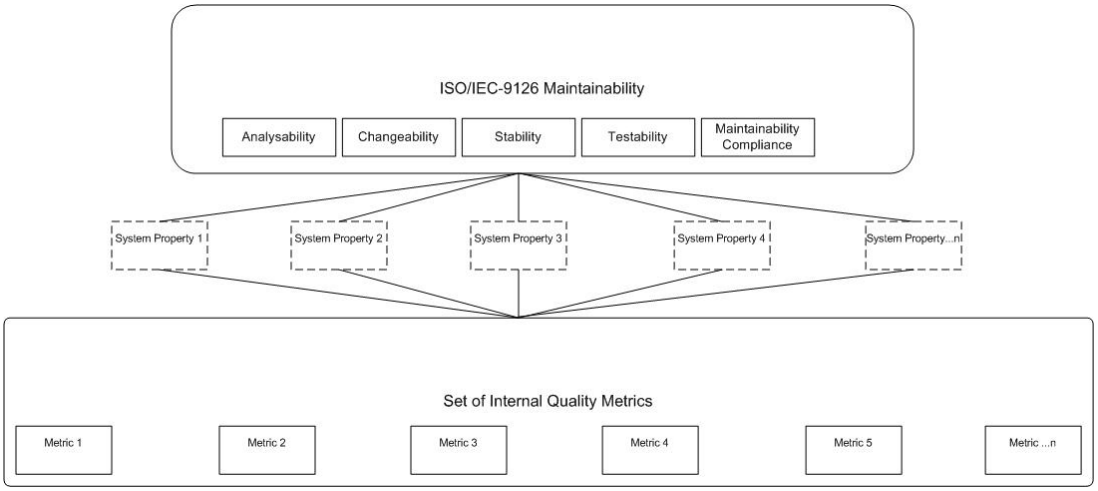

Figure 1, ISO/IEC-9126 Maintainability Model according to [4]and [1]

six main characteristics: functionality, reliability, usability, efficiency, maintainability, and portability. These main characteristics are further subdivided into a total of 27 sub-characteristics.

Though the ISO/IEC 9126 model defines a useful terminological framework for thought and communication about software quality, it does not provide an operational instrument for quality evaluation.

In parts 2 and 3 of the model, measures are suggested for evaluating the quality sub-characteristics defined in the model of part 1. However, these measures are not limited to observations of the software product itself, but for instance involve comparison of implemented features to required features or measurement of the activities of software engineers.

The Software Improvement Group (SIG) has developed a pragmatic operationalization of the ISO/IEC 9126 model, which has been described in simplified form by Heitlager et al [4]. The method is used to support IT management in activities such as vendor management, outsourcing, product selection, quality improvement programs, and strategic risk mitigation.

The SIG method maps source code measurements to quality sub-characteristics in two steps, with so-called system properties as intermediate layer. For the maintainability characteristic, the mapping between system properties and quality sub-characteristics is depicted in Figure 1. We discuss each step in turn.

3.3.1 Mapping code measures to system properties

For each system property, one or two source code measures are defined, together with an aggregation and scoring method. We discuss two examples.

Duplication: The system property "duplication" is calculated as the percentage of all code that occurs more than once in equal code blocks of at least 6 lines. When comparing code lines, leading spaces are ignored. For scoring, the following table is used:

Table 1: Duplication Scores

rank	duplication
++	0-3%
+	3-5%
o	5-10%
-	10-20%
--	20-100%

Thus, a well-designed system should have no more than 5% code duplication, while only systems with lower than 3% duplication are considered excellent.

Complexity: The system property "complexity" is measured using the well-known cyclomatic complexity metric of McCabe [11]. This metric is computed for each unit. To aggregate the measurements for all units in a meaningful way, the units are first categorized according to the following table, published by the Software Engineering Institute:

Table 2: Complexity Levels

MCC value	risk level
1-10	simple, without much risk
11-20	more complex, moderate risk
21-50	complex, high risk
> 50	untestable, very high risk

With this table, the risk level of each unit can be determined. Subsequently, aggregation is performed by counting for each risk level what percentage of code lines falls within units at that level. Finally, the following table is used to arrive at a complexity score at the system level:

Table 3: Complexity on System Level

rank	Maximum relative LOC Moderate risk	High risk	very high risk
++	25%	0%	0%
+	30%	5%	0%
o	40%	10%	0%
-	50%	15%	5%
--	-	-	-

Thus, to be rated as excellent for instance, a system can have not more than 25% of code with moderate risk, and no code with higher than moderate risk.

3.3.2 Mapping system properties to quality sub-characteristics

The mapping of system properties to quality sub-characteristics as defined by the ISO/IEC 9126 quality model is given for the maintainability characteristic by the following table:

Table 4: System Properties Mapping

	volume	complexity per unit	duplication	unit size	unit testing
analyzability	x		x	x	
changeability		x	x		
stability					x
testability		x		x	x

Thus, for example, the sub-characteristic of changeability is influenced by the system properties of complexity and duplication.

3.3.3 Formalization

A concise formalization of the 2-step mapping of SIG is given by the following two function signatures:

$$map_1 : v^M \rightarrow p^P$$
$$map_2 : p^P \rightarrow q^Q$$

The first function maps source code measurement values v onto system properties p. The second function translates system properties to quality characteristics. The number of system properties in the model is represented by P. As in the formalization of the problem statement, M and Q represent the number of measurements and the number of quality characteristics.

AHP can be applied to introduce weights into the ISO/IEC-9126 appraisals described above. Since the translation from source code measurements to quality characteristics is done in two steps, there are two levels at which AHP can be applied: to obtain property scores from measurements, and to obtain quality scores from properties. At each of these layers, multiple properties and quality scores exist, so, in fact, the scoring function is applied many times:

$$score_{\text{properies}} : w^M \times m^M \rightarrow p^P$$
$$score_{\text{characteristics}} : w^P \times p^P \rightarrow q^Q$$

Thus, each scoring function computes weighted sums of the values at one level to obtain a vector of scores on the next level.

Note that in the SIG model the source code measurements v are already aggregated from single units to the entire system.

$$aggregate : v^{M \times U} \rightarrow v^M$$

When using clustering, we are not interested in performing aggregation so early in the process. Instead, we apply the scoring functions per unit and per cluster:

$$score_{\text{properies}} : w^M \times m^{M \times U \times G} \rightarrow p^{P \times U \times G}$$
$$score_{\text{characteristics}} : w^Q \times p^{P \times U \times G} \rightarrow q^{Q \times U \times G}$$

Thus, after applying clustering and the two scoring functions in sequence, we obtain a three-dimensional matrix that contains a quality score for each ISO/IEC-9126 maintainability sub-characteristic of each unit in each cluster.

3.4 Quality profiles

Since clusters can contain large numbers of units, a separate quality score per unit is not a convenient way to report the cluster's quality. A good summary can be obtained by aggregating these separate scores into a so-called quality profile. This means that a limited number of bins are defined, together with criteria for assigning units to these bins. The counts of units assigned to each bin provide a concise summary. A particular case of such a quality profile is a histogram.

Quality profiles are formalized by the following function signature:

$$profile : x^U \rightarrow n^B$$

We will apply quality profiling to each cluster and each quality characteristic:

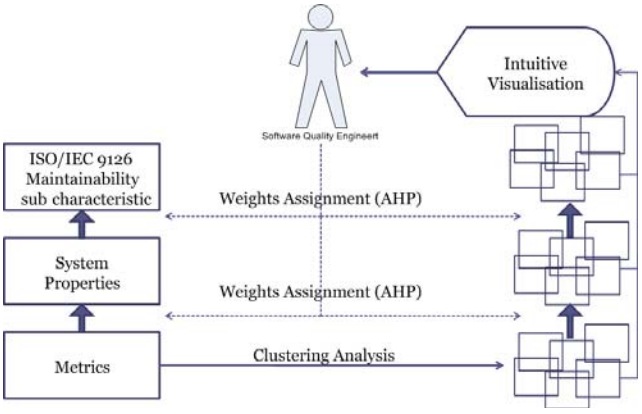

Figure 2, Steps of the Methodology

$$profile : q^{Q \times U \times G} \rightarrow n^{B \times Q \times G}$$

Thus, we obtain a series of histograms that together provide an overview of the quality of the system.

4. Method Description

We have combined AHP, clustering, ISO/IEC-9126 appraisal, and quality profiling into a single method for software product quality assessment. An overview of the overall method is provided in Figure 2. The left column in the figure indicates the application of the ISO/IEC-9126 appraisal using AHP-generated weights to the system as a whole, i.e. without clustering. The right column in the figure indicates what happens if clustering is applied first. In that case, the ISO/IEC-9126 appraisal is applied to each cluster separately. The result of the left column is one quality profile per quality characteristic. The result of the right column is one profile per combination of cluster and quality characteristic.

We will explain the various steps of the methodology in turn.

4.1 Set the assessment parameters

The first step of the methodology consists in choosing the values of its various parameters. These parameters include the following:

- *M:* the types of measures applied to source code units.
- *P:* the system properties to which the measures M are aggregated, using weights from AHP.
- *Q:* the quality sub-characteristics to which the system properties P are aggregated, using weights from AHP.
- *B:* the number of bins employed to create quality profiles.

4.2 Elicit one-on-one comparisons

The second step of the methodology is to use AHP to configure the various scoring functions. The pair-wise comparisons to be elicited from the experts are determined by the choice of measures *M*, properties *P*, and quality characteristics *Q*.

4.3 Metrics extraction

For each unit of the system under study, source code measures are performed to obtain a vector v^M of measurement values.

In general, the extracted metrics are numerical data with different ranges. Data normalization is performed in order to fit the measurement values into the range [0…1]. This makes the data suitable for clustering and for scoring with AHP.

4.4 Clustering Parameters

The matrix of measurement value vectors $v^{M \times U}$ is divided into groups using the k-Attractors clustering algorithm. The following table presents the input parameters of the algorithm.

Table 5: Clustering Parameters

Parameter	Description
Support *s*	Defines the required support for the discovery of initial attractors
Hamming Distance power *h*	Defines the similarity metric's sensitivity to Hamming distance [8]
Inner Product power *i*	Defines the similarity metric's sensitivity to the Inner product [8]
Number of Attractors *k*	Defines the maximum number for the derived attractors [8]

After setting these parameters, the k-Attractors can be applied.

4.5 ISO/IEC 9126 appraisal

After clustering, the scoring functions are applied to derive the appraisals in terms of ISO/IEC 9126 sub-characteristics.

4.6 Visualization

The histogram representation was employed for visualization purposes as it enhances the understanding of parameters of interest [7]. It is a graphic representation of frequency counts of a population. The X-axis lists the unit intervals of a quality aspect (i.e. ISO/IEC-9126 maintainability sub-characteristic, system property, software metric) ranked in ascending order from left to right, and the Y-axis contains the frequency counts. The purpose of the histogram is to show the distribution characteristics of a quality dimension such as overall shape, central tendency, dispersion, and skewness.

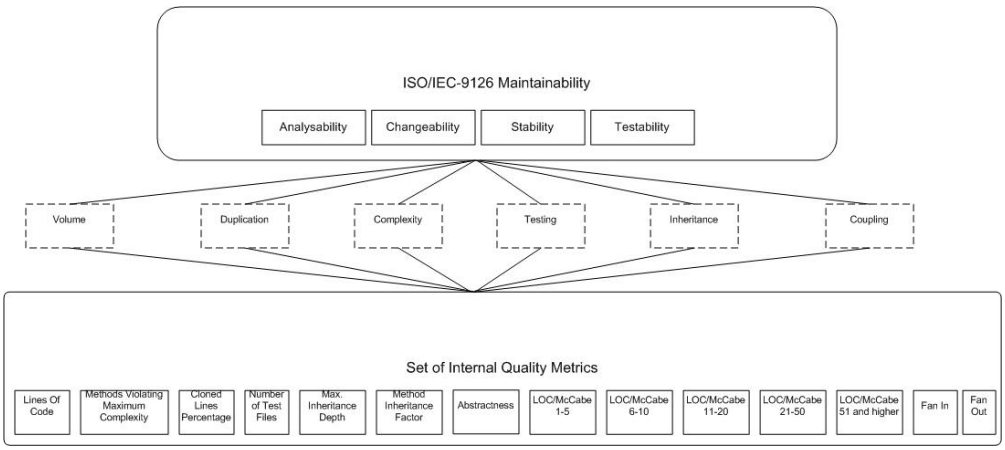

Figure 3, Employed Assessment Parameters

5. Experimental Results

We have evaluated our proposed method by applying it to a number of case studies taken from the IT management consultancy practice of the Software Improvement Group. Here we report on one of these case studies.

The Software Improvement Group specializes in providing IT management consultancy grounded in source code analysis of the software systems of its clients. SIG's main services are Software Risk Assessments [3] in which a single snapshot of a system is studied in depth, and Software Portfolio Monitoring [9] in which one or several systems are studied over time. Currently, SIG analyses about 50 industrial-size, mission-critical software systems per year, developed in a wide variety of technologies.

The case study reported on in this section, concerns a 3-year old administrative system for a regional government. The system is developed in Java. The Java code amounts to about 400.000 lines of code, divided over about 5000 compilation units.

5.1 Assessment parameters

Figure 3 presents the selected parameters for the performed assessment. The assessment focused on the maintainability aspect of software product quality, which ISO/IEC 9126 defines in terms of the following sub-characteristics:

- *Analysability:* how easy is it to understand where in the system changes need to be made?
- *Changeability:* how easy is it to actually make change?
- *Stability:* after making a change, how easy is it to bring the overall system back into a consistent state?
- *Testability:* how easy is it to determine whether the change made has been made correctly?

The number of bins B of the quality profiles has been set to 24.

5.2. Weight assignment by AHP

The pair-wise comparisons used as input for the AHP scoring were elicited from senior consultants of SIG. The weight elicitation was done independently of the system assessment itself, as the consultants had no prior knowledge of the system under study. The elicited pair-wise comparisons gave rise to the following weights w^P for scoring of quality sub-characteristics:

Table 6: System Properties Weights

	Analysa -bility	Stability	Change-Ability	Testability
Volume	0.32	0.05	0.07	0.08
Duplication	0.21	0.15	0.31	0.07
Complexity	0.11	0.08	0.17	0.24
Testing	0.15	0.35	0.12	0.37
Inheritance	0.08	0.11	0.09	0.08
Coupling	0.14	0.27	0.23	0.15

Weights were also given in order to reflect the importance of metrics in relation to source code properties. The following tables present the weights w^M concerning the metrics relevance to the source code properties of complexity, inheritance and coupling respectively.

Table 7: Complexity Weights

	Complexity
Methods violating maximum complexity	0.36
LOC of units with McCabe 1-5	0.05
LOC of units with McCabe 6-10	0.08
LOC of units with McCabe 11-20	0.12
LOC of units with McCabe 21-50	0.22
LOC of units with McCabe 51 and higher	0.16

Table 8: Inheritance Weights

	Inheritance
Maximum Inheritance Depth	0.54
Method Inheritance Factor	0.30
Abstractness	0.16

Table 9: Coupling Weights

	Coupling
Fan In	0.30
Fan Out	0.30
Afferent Coupling	0.17
Efferent Coupling	0.17
Instability	0.06

5.3. Clustering

Clustering was performed on the measurement data using the k-Attractors algorithm. The values of input parameters values are in the following table. The support factor *s* for the frequent itemsets discovery was chosen to be 0.1 (10%) because the value of each class metric has a large range, thus it is more difficult to find frequent itemsets with a greater support. Additionally we have chosen to give more weight to the Hamming distance in relation to the inner product distance, because of the similar values of every module metric which result in a low Hamming distance for the most data items. The number of clusters in k-Attractors was derived by

applying the graph partitioning algorithm (kMetis). In our case, kMetis partitioned the graph into 7 parts, thus the initial number of clusters was 7.

Table 10: Employed Parameters

Parameter	Description
Support *s*	0.10
Hamming Distance power *h*	5
Inner Product power *i*	3
Number of Attractors *k*	7

The algorithm derived 6 clusters of various sizes:

Table 11: Obtained Clusters

Cluster No	Size %
1	0.52%
2	73.71%
3	4.64%
4	0.52%
5	5.15%
6	15.46%

5.4 Results evaluation

Inspection of the visualized quality profiles helps to obtain a quality appraisal of the various clusters discovered in the system. To this end, the various histograms are plotted side-by-side for easy comparison. For example, Figure 4 shows the histograms for the changeability sub-characteristic for all clusters. In once glance, one can deduce that the worst cluster in terms of changeability is cluster number 6.

Figure 4: Clusters Changeability

Alternatively, a quality engineer can inspect all histograms for a single cluster. For example, figure 5 shows all four sub-characteristic histograms for cluster number 6. At a glance, one can deduce that this cluster scores well in terms of testability, but poorly in terms of changeability.

Figure 5: Cluster 6 overview

Histograms for lower levels can be used to zoom in on the root causes of problems recognized at higher levels. For example, figure 6 shows the histograms for system properties for the same cluster. These histograms show that the poor changeability is caused by the high duplication and complexity of this cluster.

Figure 6: Cluster 6 Source Code Properties

5.5 Discussion

Using our methodology for interpreting source code clusters in terms of ISO/IEC 9126 quality characteristics, we were able to quickly obtain high-level, but informative information about the quality of the system. The method has a number of attractive aspects. Firstly, the method compares well with the most naïve technique for interpreting source code metrics, being the identification of outliers for individual metrics. Such outliers are important to identify, but they only point to local quality problems. They do not say much about the quality of the system as a whole.

Secondly, the method compares well with existing methods for translating source code measurements into high-level quality characteristics. The introduction of clustering into the process ensures that aggregation is done on relatively homogenous groups of units. Without such homogeneity, the high-level aggregation is necessarily less informative.

Thirdly, the use of AHP, rather than other weighting schemes turned out to scale up nicely to the high number of metrics and system properties addressed in the case study.

6. Related Work

Plosh et al. [14] presented the EMISQ method, which stands for "Evaluation Method for Internal Software Quality" and it consists of a quality model, an assessment model, a set of documents and a tool named S.P.Q.R. (Software Product Quality Reporter). The EMISQ's quality model is based on the ISO/IEC-9126 quality model, enhanced with input from the SATC and CMM quality models.

Liang et al. [10] present a five-step practical procedure, which also combines the ISO/IEC 9126 with a sophistication of AHP named fuzzy AHP (FAHP) in the context of the ERP selection problem optimization. The first step is the project initiation and requirements identification. Then, the feasible software search and selection criteria are extracted whereas the third step is the construction of hierarchy concerning the ERP software. Computing the values by employing the FAHP approach and selecting the optimal ERP software are the last two final steps of this procedure.

Svanhberg et al. [16] present an empirical study of a method that enables quantification of the importance different quality attributes have for

different software architecture patterns (e.g. layered, pipes and filters, blackboard etc.). For this study the quality characteristics of ISO/IEC 9126 were employed in order to cover a wide range of a software system's aspects. AHP on the other hand was used as the decision method that will enable the quantitative comparison between different quality attributes and architecture candidates.

Broy et al. [2] have independently developed a similar model of maintainability in which maintenance activities are strictly separated from facts about the system being maintained. Both activities and facts are organized into hierarchical trees whose leaves are related through a (weighted) matrix that indicates which atomic facts influence each atomic activity. The decomposition of activities is based on the IEEE 1219 standard maintenance process, and the decomposition of facts has been developed in concert with industrial partners. A simplified part of the full model is presented only. Specific rating guidelines or weights are not presented.

Oman et al. [12] have proposed a hierarchical structure of measurable maintainability attributes, based on a review of 35 publications. They attach specific software metrics to the leaves of the tree and propose a formula for combining them into a single index. No specific weights are proposed to instantiate that formula. A much larger number of metrics is proposed than in [13], and they include not only source code metrics, but also metrics about e.g. changes, defects found, and documentation.

Moreover [17] 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 labeled them as fault prone or not.

The value of our work is the integration of an improved model of quantitative quality assessment with the k-Attractors clustering algorithm; into a methodology for interpreting the derived clusters in terms of the ISO/IEC 9126 quality characteristics. Moreover, a method for visualizing the quality profiles associated to generated clusters using word-sized histograms, allows drawing conclusions about the variation of quality scores among generated clusters.

7. Conclusions and Future Work

We briefly summarize our contributions, and we provide an outlook for possible elaborations and improvements of our ideas.

7.1 Contributions

We have taken various pre-existing techniques for data mining and software quality analysis and combined them into a single method for obtaining a high-level quality assessment for a software system from low-level measurements of its source code. The following contributions have been made:

- We have shown how Analytic Hierarchical Processing can be used to improve our previous approach to mapping source code measurements to ISO/IEC 9126 quality sub-characteristics. We have shown that AHP can be used to incorporate expert knowledge into the weighting process in a structured way.
- We have shown that clusters of source code units discovered by unsupervised clustering can be interpreted in terms of system-wide quality characteristics. We have shown that the improved mapping of source code measurements to ISO/IEC 9126 sub-characteristics can be used to provide such interpretation.
- We have demonstrated that histograms or quality profiles are an effective means for summarizing the quality appraisal information obtained at system level.
-

7.2 Future work

Many directions for future work are worth exploring.

- We have opted to use the k-Attractors clustering algorithm for grouping source code units. Other grouping methods can be used. For example, the grouping of source code into packages, layers, or modules, as performed by the designers and developers can be used instead of clustering. It remains to be seen in what circumstances such alternative groupings are feasible and perhaps more informative than clustering.
- We have focused on the maintainability characteristic of the ISO/IEC 9126 quality model. It will be worthwhile to explore the inclusion of further characteristics, such as reliability, efficiency, and usability, into the analysis.
- We have made a particular selection of source code metrics and system properties to operationalize the ISO/IEC-9126 quality model. Experimentation is needed with other sets of metrics and properties.
- We have reported on a single case study in this paper. Other case studies are underway. We are especially interested in applying the approach to even larger systems, and systems with different technology footprints.

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".

REFERENCES

[1] P. Antonellis, D. Antoniou, Y. Kanellopoulos, C. Makris, E. Theodoridis, C. Tjortjis, and N.Tsirakis, "A data mining methodology for evaluating maintainability according to ISO/IEC-9126 software engineering – product quality standard," in Special Session on System Quality and Maintainability (SQM), 2007.

[2] M. Broy, F. Deissenboeck, and M. Pizka, "Demystifying maintainability," in Fourth International Workshop on Software Quality Assurance (SOQUA 2007). ACM, 2007.

[3] Arie van Deursen, Tobias Kuipers: Source-Based Software Risk Assessment. ICSM 2003: 385-388

[4] I. Heitlager., T. Kuipers, and J. Visser, A Practical Model for Measuring Maintainability, In proceedings of the 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007), IEEE Computer Society Press, 2007.

[5] ISO/IEC 9126-1, Software Engineering – Product Quality Int'l Standard Quality Model, Geneva 2003.

[6] A.K Jain, M.N. Murty, and P.J. Flynn, "Data Clustering: A Review", ACM Computing Surveys, ACM, Vol. 31, No 3, September 1999, pp. 264-323.

[7] S.H. Kan. "Metrics and Models in Software Quality Engineering". Addison-Wesley, 2003.

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

[9] Tobias Kuipers, Joost Visser: A Tool-based Methodology for Software Portfolio Monitoring. Software Audit and Metrics 2004: 118-128

[10] S. Liang, C. Lien, "Selecting the Optimal ERP Software by Combining the ISO 9126 Standard and Fuzzy AHP Approach", Journal of Contemporary Management Research, Pages 23-44, Vol. 3, No. 1, March 2007

[11] T.J. McCabe, "A complexity measure." IEEE Trans. Software Eng., vol. 2, no. 4, pp. 308–320, 1976.

[12] P. Oman and J. Hagemeister, "Metrics for assessing a software system's maintainability," in Proceedings of Conference on Software Maintenance, 1992, Nov. 1992, pp. 337–344.

[13] P. Oman and J. R. Hagemeister, "Construction and testing of polynomials predicting software maintainability." Journal of Systems and Software, vol. 24, no. 3, pp. 251–266, 1994.

[14] R. Plösch, H. Gruber, A. Hentschel, Ch. Körner, G. Pomberger, S. Schiffer, M. Saft, S. Storck: The EMISQ Method - Expert Based Evaluation of Internal Software Quality, Proc. 3rd IEEE Systems and Software Week, 2007, Baltimore, USA, IEEE Computer Society Press, 2007.

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

[16] S. Svahnberg, C. Wohlin, "An Investigation of a Method for Identifying a Software Architecture Candidate with Respect to Quality Attributes", Journal of Empirical Software Engineering, pages 140-181, Vol. 10, 2005

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