Journal of Software: Evolution and Process

JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
J. Softw.: Evol. and Proc. 2013; **25**:303–324
Published online 31 January 2012 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/smr.584

Combining metrics for software evolution assessment by means of Data Envelopment Analysis

Alexander Chatzigeorgiou*,† and Emmanouil Stiakakis

Department of Applied Informatics, University of Macedonia, 156 Egnatia St, 54006 Thessaloniki, Greece

SUMMARY

Research and practice in software engineering have led to an extensive set of metrics for the evaluation of almost every aspect of software development. One of the major challenges for any quality model is the combination of metrics, which are complementary to each other. In this paper, we propose the use of Data Envelopment Analysis (DEA), a non-parametric technique employed in economics, as a means of providing a unified view of selected design metrics. The proposed application of DEA aims at assessing the overall trend of quality during the evolution of software systems, by considering releases of a project as units to be ranked. An important benefit derived from the use of DEA is the ability to "normalize" the evaluation over the size characteristics of the examined systems, which is vital when comparing projects of different scale. Results are presented for successive versions of two open-source, one industrial and one research project, whereas validation, whereas validation is performed by comparing the findings with the results obtained by Analytic Hierarchy Process, which is an acknowledged multi-criteria decision analysis approach. According to the results, DEA enables the perception of global trends in qualitative characteristics, which would be otherwise difficult to recognize and interpret. Copyright © 2012 John Wiley & Sons, Ltd.

Received 23 February 2011; Accepted 9 December 2011

KEY WORDS: software evolution; software metrics; Data Envelopment Analysis; Analytic Hierarchy Process

1. INTRODUCTION

Software metrics are widely considered as a means to capture objectively the quality of software products, processes or projects [1]. A direct consequence of this perspective is the multitude of metrics that have been proposed in the literature for the quantification of almost every aspect of software products and software development lifecycle, as well as the support provided by all Computer-Aided Software Engineering tools for performing measurements. A number of studies provide empirical evidence supporting the role of metrics in determining high-level quality attributes [2], such as software defects [3, 4], maintenance effort [5, 6], reusability, and flexibility [7]. However, it should be noted that despite their long history in software engineering, the value that metrics can offer is frequently brought in question [8].

One of the major challenges when analyzing the quality of software projects is the problem of unifying the values of different metrics, aiming at the assessment of closely related but different notions of a quality attribute. In other words, it is non-trivial to reveal the relationships between high-level quality properties, such as maintainability and internal quality attributes [9, 10]. As an example, according to ISO 9126 [11], the main quality characteristic referring to maintainability is

*Correspondence to: Alexander Chatzigeorgiou, Department of Applied Informatics, University of Macedonia, 156 Egnatia St, 54006 Thessaloniki, Greece.
†E-mail: achat@uom.gr

assessed in terms of four sub-characteristics, namely analyzability, changeability, stability, and testability. Each sub-characteristic can be supposedly assessed in an objective manner employing appropriate software metrics [7]. This is the point at which most quality frameworks are differentiated because various complementary measures exist for the quantification of a given sub-characteristic, and the selection of them depends on the perspective of the modelers. For example, according to the SQO-OSS quality model [12], changeability for open-source software (incl. procedural code) depends on the average size of statements, the vocabulary frequency, the number of unconditional jumps, the number of nested levels, the coupling between objects, the lack of cohesion, and the depth of the inheritance tree.

However, even if a software quality model is adopted and the metrics for the quantification of each sub-characteristic are established, one key problem is that software metrics have to be combined in order to provide a single view and allow the ranking of the examined project or module at a certain level (we use the term *combination* to refer to the unified consideration of several metrics for the same artifact in contrast to *aggregation*, which refers to the employed mechanism for unifying measurements from the micro-level of individual artifacts, for example, methods, classes, and packages to the macro-level of the entire software system [13]).

The second challenge related to the assessment of the quality of a software artifact concerns the consideration only of "output" variables neglecting the fact that software projects may be completely different in terms of size. This is particularly important when metrics are used to compare different software projects or, as in the case of this paper, successive versions of the same project to assess its evolution. Even if metrics are normalized over some global system measure, when two systems are assessed by means of metrics, one of the two might be significantly larger in terms of delivered functionality. It is generally acknowledged that system's size and relevant complexity affect design properties and thus should not be neglected when comparing software systems of unequal dimensions. Comparing and especially benchmarking software systems or versions neglecting their size might lead to situations where apples are compared with oranges.

The aforementioned issues become extremely important when the goal is to assess the evolution of a software system observing selected metric values over a number of successive versions [9, 14]. According to the recommendations proposed by Lehman and Ramil for software evolution planning [15], metric tracking and monitoring changes in evolutionary trends is a desirable management aid. It is now understood that organized collections of software repositories can offer a rich source of information regarding software quality and previous design decisions [16], because they grant access to historical versions of the source code [17]. An entire field of research, namely the Mining of Software Repositories [18] has focused on the exploitation of past software related data to support the maintenance of software systems, improve software design/reuse, and empirically validate novel ideas and techniques. The correct interpretation of metrics in the context of evolution and a reliable approach to combine metrics reflecting complementary aspects of quality are therefore essential.

The approach presented in this paper draws ideas, which have been widely applied in production economics, where researchers have developed elaborated techniques that allow the benchmarking of different companies that have varying size characteristics and whose efficiency is a nonlinear function of a number of inputs and outputs. When comparing the efficiency of two companies, it cannot be claimed that a company A is more efficient than a company B, simply because the profit of the first is larger. It might be the case that the larger profit of A is achieved at the cost of 10 times the amount of investment. Efficiency can be trivially obtained for a single input/output process as the ratio of output over input. However, efficiency computation becomes a challenging and non-trivial issue when numerous factors should be considered simultaneously and when there is a nonlinear relationship between inputs and outputs [19].

The analogy between the benchmarking of companies with different "input" characteristics assessing various "output" measures to the comparison of successive software versions forms the inspiration of the proposed approach. Software design metrics, such as coupling and cohesion are the means to assess the output of a design process in analogy to economic factors such as profit and sales which are measured when assessing the productivity of a company. One of the most successful approaches for providing a single view of various economic factors is the well-studied Data Envelopment Analysis (DEA), originally proposed by Charnes *et al.* [19]. DEA employs linear programming methods to measure the performance of so-called decision-making units (DMUs) with respect to each other. Furthermore, DEA

DOI: 10.1002/smr

accounts for the inputs to each assessed company, such as employees, investments, resources, loans, and so on. This makes DEA an ideal choice for comparing the evolution among different software versions, which differ in terms of size, delivered functionality, or number of state variables.

In this paper, we illustrate the suitability of DEA as a standard, simple, and automatable approach for assessing the evolution of quality over a number of successive software versions. The obtained combined view of selected software metrics, the choice of which does not affect the application of the approach, and the inherent normalization over the input characteristics provides an objective way to study the evolution of certain aspects of design quality. The fact that the compared systems are of the same nature is another factor that underpins the suitability of DEA as the compared units should be of the same nature [20].

In a previous study, we presented the use of DEA as a means of benchmarking software projects belonging to two different categories, namely libraries (APIs) and applications [21]. The results indicated that libraries exhibit a superior design quality compared with applications, according to a set of selected metrics. The major threat to the validity of that study is the fact that the examined systems belonged to different sub-categories (e.g., the application software consisted of various kinds of projects, such as games, editors, scientific software, etc.), whereas DEA is ideal for benchmarking systems of the same type. This is completely ensured in the current study because the goal is to rank versions of the same system eliminating the risk of comparing unrelated projects.

Nevertheless, it should be mentioned that the application of DEA in order to assess the evolution of quality over a number of software versions is preliminary. The limitations of the proposed approach, as well as a number of threats to the validity of the results are recognized, imposing the need for further validation on a larger set of open-source and industrial projects.

The rest of the paper is organized as follows: Section 2 provides an overview of approaches employed for the combination of metrics. A brief introduction of DEA is given in Section 3 along with a discussion of its use in the software design context and the added value that it can offer. The application of Analytic Hierarchy Process (AHP) is presented in Section 4. Results for four case studies are discussed in Section 5, whereas limitations and threats to validity are given in Section 6. Finally, we conclude in Section 7.

2. RELATED WORK

Several approaches are used in the software engineering community to confront the issue of combining software metrics. The most classic approach is to obtain an aggregate figure as a weighted average of individual metric results. In this case, a quantitative estimate for the attribute of interest is computed as the sum of the products of the weight and each measure value [22]. A typical example is the Maintainability Index [23], which combines several code measures into a polynomial that can be used as an indicator of maintainability. Beyond the immediate drawback related to the subjective selection of weights, it is questionable whether weighing and summing are legal operations for metrics of different scales and units [22]. However, the weighted sum is by far the most widely used approach for the combination of metrics.

An alternative approach is profile based evaluation [24], where a profile vector refers to the minimum (or maximum) values that metrics should have so that they can be combined at a certain level. As an example for analyzability, a cyclomatic complexity lower than 4, a number of declarations lower than 10, a comment ratio higher than 0.5, and an average size of declarations lower than 4 would lead to an assessment ranked "Excellent" [25]. The selection of the appropriate threshold values that constitute the profile vector (in this case, the vector [4, 10, 0.5, 4]) is obviously the aspect that introduces subjectivity into this approach.

Various measures can also be combined in the context of *multivariate regression analysis* employing techniques, such as Principal Component Analysis to select the metrics that should be included in a model based on their explanatory strength [26]. However, because the models are constructed using past data for training, the model is perfectly suited to the examined data but the usual threats to its ability for generalization are valid. Multiple measures can also be combined into a single view employing visualization techniques, but these strategies suffer from the need to rely on human expertise.

DOI: 10.1002/smr

Despite the obvious importance of metric combination for assessing the overall quality of a given system, the unification of multiple metrics is also a major challenge in search-based determination of refactorings [27]. Search-based refactoring aims at extracting a single sequence of refactorings that leads to an optimum system. Search exploration is usually guided by a single metric that measures the overall quality of the system and is used as the fitness function that evaluates each solution. Evaluation functions consisting of multiple metric values have also been used by O'Keeffe and O'Cinnéide [28] to drive the search techniques for performing automated refactorings. The combination of metrics is necessary in case the corresponding formulation is as a single-objective search problem. According to Harman and Tratt [27], if weights are used to combine multiple metrics into a single one, problems can arise when the choice of weight coefficients is unclear and when the various metric values are not independent.

The combination of multiple metrics is also a requirement for treating the allocation of methods and attributes in an object-oriented system as an optimization problem where the goal is to minimize by means of genetic algorithms certain metrics, such as cohesion and coupling [29]. The alternative would be to employ multiple complementary measures forming the more complex multi-objective genetic algorithm approach [30]. However, a multi-objective algorithm leads inevitably to a large number of alternative, non-dominated solutions that have to be inspected by the designer. It should also be mentioned that multi-objective approaches have a significantly higher computational cost, which can be prohibitive in terms of time even for a small-sized system.

3. APPLICATION OF DATA ENVELOPMENT ANALYSIS

3.1. Brief description of Data Envelopment Analysis

Data Envelopment Analysis is a linear programming methodology that evaluates the relative efficiency of a number of DMUs, which utilize multiple inputs to produce multiple outputs [19]. The units that attain the maximum efficiency score form a mathematical space, which envelops all the other less efficient units. This space is the so called "efficient frontier" and the degree of inefficiency of all other units is evaluated in relation to their distance from the efficient frontier. The relative efficiency of a DMU is calculated by forming the ratio of a weighted sum of outputs to a weighted sum of inputs. The weights for both outputs and inputs are calculated in a manner that gives the maximal efficiency measure of each DMU, subject to the constraint that no DMU can have an efficiency score greater than unity [31, 32]. A more detailed illustration of how DEA works by means of a simplified example is provided in Section 3.2.

One of the primary advantages of DEA is that each input and output can be measured in its physical units. There is no need for a weighting system that reduces those units into a single unit measure [33]. Another strength of DEA is that it allows each DMU to select the weights that maximize its own efficiency. DEA can also be used as a multi-criteria decision making (MCDM) tool, if inputs and outputs are considered as criteria for evaluating DMUs, with minimization of inputs and/or maximization of outputs as the associated objectives of this methodology [34]. A multi-criteria usage of DEA is ranking the DMUs in terms of their efficiency, that is, the ability to convert inputs into outputs leading to the most classical application of DEA, which refers to the benchmarking of companies [31]. Beyond the primary domain of production economics, DEA has been successfully applied to a wide range of contexts, such as education, health care, banking, auditing, sports, market research, and so on. A few characteristic examples are given next [35]. A noteworthy application of DEA was the evaluation of the performance of medical and surgical departments of a number of acute hospitals; this kind of studies is very complex because of the amount of input and output information needed to describe the hospitals' services and the patients' trajectories. In the field of engineering, DEA has been applied in electric power systems, semiconductor assembly, flexible manufacturing systems, and so on. These examples are indicative of the diffusion of the DEA methodology into many different disciplines and contexts; however, to our knowledge, DEA has not been applied to evaluate successive versions of the same system in general and particularly in the evolution of a software system.

DOI: 10.1002/smr

3.2. Software context

As already mentioned, DEA is employed in our approach in order to rank successive versions of the same software project according to a unified view of selected design metrics and considering size characteristics of the examined projects. Obviously, a software project cannot have a direct analogy to a business unit, which is the usual target of evaluation by DEA. However, design metrics, such as coupling, cohesion, and complexity can be considered as the "outputs" of the software design process, which the developers seek to optimize (minimize or maximize depending on the particular metric). Considering two successive software versions s_i and s_{i+1}, with the same size characteristics, the version with the better metric values can be considered as more efficient, from a design quality perspective. DEA estimates a single efficiency score and thus it handles the unification of the different selected output metrics.

However, under realistic conditions, it would be unfair to compare two software versions with different size characteristics, that is, with different functionality. As an example, comparing versions s_i and s_{i+1} and assuming that the latter version has been significantly enhanced in terms of functionality (which would be reflected in an increased LOC count, increased number of classes, methods, and attributes) simply on the basis of their design metrics (even if they are normalized over some global size measure) would be misleading. A larger system that achieves the same average class complexity, coupling or cohesion with a substantially smaller one, would be generally considered as better designed. This perspective imposes the need to consider "inputs" to the design process along with the "output" metrics for the examined system.

An ideal and code independent measure of size would be the requested system functionality from the user's point of view as measured by function points [36]. However, because of the lack of documented function point measures for most open-source projects, we relaxed the specification of "inputs" to indirect size measures that can be extracted from the source code, such as the number of methods, attributes, or classes. These size metrics are not ideal candidates as inputs to an efficiency evaluation approach because their value also depends on the design decisions taken by the developers. However, considering that in a typical object-oriented analysis and design methodology most classes, methods, and attributes have been discovered when the final design phase is reached, the corresponding metrics can be treated as inputs to that phase. This relaxed view is graphically illustrated in Figure 1. Therefore, the evaluation of the final design phase where major design decisions are taken, class relationships are finalized, design principles are enforced, and design patterns eventually are implemented, can be performed based on these size measures as inputs and the selected software metrics as outputs. This is similar to a business unit that, for given input resources, strives for the best (financial) results.

A major difference compared with the conventional application of DEA in an economic context, is that within the aforementioned perspective, inputs to the design phase are not negotiable. Obviously, for a given functionality, there are several alternatives in terms of methods, attributes, and classes that can implement the requested functional requirements. However, when benchmarking software versions of the same project, it would be irrational to suggest an artificial increase to the number of classes/methods/attributes just in order to improve the efficiency of a version. Consequently, we view the system with the higher input values (larger size) as better than another one with smaller

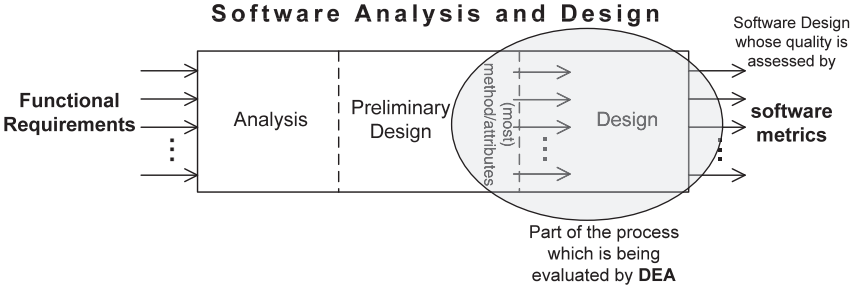

Figure 1. Relaxed view of the software design process forming the context of DEA.

DOI: 10.1002/smr

size (that has the same output design metric values), but we do not treat the inputs as variables that can be adjusted.

To illustrate how DEA works, let us assume a software system of eight versions ($v.1$, $v.2$, $v.3$, ..., $v.8$) where each version needs to be assessed by means of one input and two outputs (we employed in this case only two outputs and one input in order to be able to depict graphically the solution of the system). For this example, we used the Number of Operations (NOO) as input, and the average fan-in coupling and average class cohesion as outputs. Sample data are given in Table I.

The efficiency score of each version (DMU), for example of version $v.4$, is given by maximizing the weighted ratio of all outputs over all inputs, subject to the constraints that the efficiency scores for each DMU must be less than or equal to unity:

$$\max_{u_{1v.4},u_{2v.4},v_{v.4}} \frac{u_{1v.4}\cdot y_{1v.4} + u_{2v.4}\cdot y_{2v.4}}{v_{v.4}\cdot x_{v.4}}$$

subject to

$$\begin{aligned} \frac{u_{1v.4}\cdot y_{1i} + u_{2v.4}\cdot y_{2i}}{v_{v.4}\cdot x_i} &\leq 1 \quad \forall i \in \{v.1, \ldots, v.8\} \\ u_{1v.4},\ u_{2v.4},\ v_{v.4} &\geq 0 \end{aligned}$$

where $u_{1v.x}$, $u_{2v.x}$ are weights for the two outputs of version $v.x$ and $v_{v.x}$ is the weight for the input of version $v.x$.

Because this problem has an infinite number of solutions, a reformulation of the DEA model is required aiming to convert the objective function to a linear function. The resulting model is the so called "*multiplier form*", which has a corresponding dual model that is usually easier to be solved (duality theory in linear programming problems). The solution of the latter model, so called "*envelopment form*", leads to the calculation of a single efficiency score for each DMU.

To illustrate graphically the essence of DEA, we show in Figure 2, the points that represent each of the examined versions. In order to be able to illustrate all data on a two-axis plane, we normalized the outputs over the employed input, that is, axis x corresponds to *fan-in coupling/NOO* whereas axis y corresponds to *cohesion/NOO*. The results of that normalization process are presented in Table II.

The efficiency score of version $v.4$ (point A in the diagram) is calculated by the radial measure $efficiency_{v.4} = \frac{0A}{0A'}$. The efficient frontier is defined essentially by versions $v.1$ and $v.2$ and the intersection with the two axes. As it can be observed, the efficiency of version $v.4$ can be improved either by increasing cohesion or fan-in coupling to move the DMU closer to the efficient frontier as shown by the dashed arrows or by a combination of both actions. The distance of each version from the efficient frontier determines the ranking of each version compared with the others. For example, according to these metrics, versions $v.1$ and $v.2$ are efficient and ranked first, whereas version $v.8$ is ranked last.

The model described earlier is the basic DEA model. Many other DEA models have been proposed to deal with specific problems. An interesting variant is the super-efficiency DEA model [37]. This model was introduced with the objective to provide a full ranking of all DMUs when the basic DEA model produces several efficient units (as in the earlier example where versions $v.1$ and $v.2$ have an efficiency score equal to one). Because the full ranking of the successive versions of a software project is indispensable in order to evaluate its progress, the super-efficiency DEA model was chosen to be applied in our analysis. This model excludes the efficient DMU under evaluation from

Table I. Sample data for DEA example.

	$v.1$	$v.2$	$v.3$	$v.4$	$v.5$	$v.6$	$v.7$	$v.8$
NOO (x)	130	150	200	230	300	320	400	450
Fan-in coupling (y_1)	10.6	11.2	13	13.6	15	15.1	15.8	14
Cohesion (y_2)	9	12.5	10.5	11	12	10	9	13

DOI: 10.1002/smr

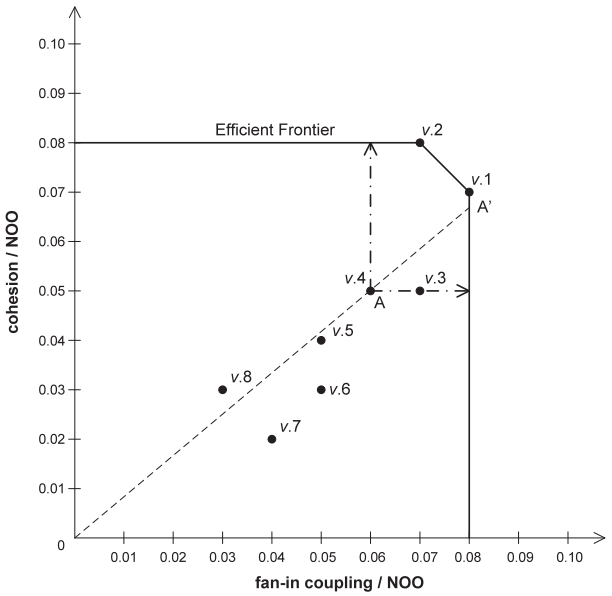

Figure 2. Efficient frontier for the examined versions of the sample software project.

Table II. Normalized data for DEA example.

	v.1	*v*.2	*v*.3	*v*.4	*v*.5	*v*.6	*v*.7	*v*.8
NOO (*x*)	1	1	1	1	1	1	1	1
Fan-in coupling (y_1)	0.08	0.07	0.07	0.06	0.05	0.05	0.04	0.03
Cohesion (y_2)	0.07	0.08	0.05	0.05	0.04	0.03	0.02	0.03

the efficient frontier. The effect of this is to shrink the frontier, allowing the efficient DMU to become super-efficient because it now has a score greater than unity.

If, for example, version *v*.2 is excluded from the efficient frontier, point *v*.1 would be directly linked to the vertical axis (see the displacement of the efficient frontier in Figure 3). As a result, point *v*.2 would be above the efficient frontier resulting in an efficiency score greater than one. In this case, which essentially applies the super-efficiency DEA model, the efficiency score of version *v*.2 (point B in the diagram) is calculated by the radial measure $efficiency_{v.2} = \frac{0B}{0B'} > 1$. This process will be repeated for all units whose efficiency score is equal to unity according to the initial efficient frontier (in this case, version *v*.1). By applying the super-efficiency DEA model, a full ranking of all DMUs becomes feasible because, after the application of this process, versions *v*.1 and *v*.2 will have a distinct efficiency score.

3.3. Added value of Data Envelopment Analysis

As already mentioned, DEA offers an automated and objective means of obtaining the overall picture when assessing the trends of multiple metrics, during the evolution of a software system. The added value of DEA essentially depends on how much the series of metrics data are convoluted. Highly convoluted trends of metrics (i.e., when their trends are not similar during periods of the software history) are difficult to assess, as no single representative trend can be derived.

The set of metrics and size values over a number of versions essentially constitutes a time series. According to Javed *et al*. [38], the perception of multiple, concurrent time series can be non-trivial and depends heavily on the extent of visual clutter. Several similarity measures have been proposed in the literature of data mining in order to capture the similarity between two time series $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_n$ to assess, for example, whether two stocks have similar movements during a selected

DOI: 10.1002/smr

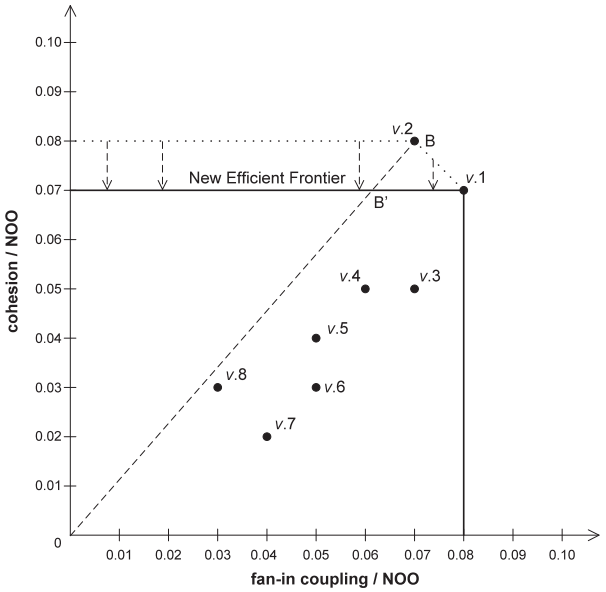

Figure 3. Displacement of the efficient frontier in case of using the super-efficiency DEA model.

period [39]. An effective and simple similarity measure for time series that can be represented by a piecewise linear approximation is the trend similarity measure [40]. The time series formed by metric values of successive software versions are inherently piecewise linear and each line, corresponding to the transition from one version to the next, is characterized by its trend, which can have one of the following three statuses, namely increasing, constant, and decreasing, depending on the slope of each line segment. Formally, the trend of each line segment can be defined as [40, 41]:

$$trend(slope) = \begin{cases} 1 & slope > 0 \\ 0 & slope = 0 \\ -1 & slope < 0 \end{cases}$$

whereas the distance between the trends of two line segments is obtained as [40]:

$$td(slope_1, slope_2) = \begin{cases} 0 & trend(slope_1) = trend(slope_2) \\ 1 & trend(slope_1) \neq trend(slope_2) \end{cases}$$

Finally, the piecewise similarity of two time series is obtained as the complement of their average distance considering all line segments:

$$trendSim(TS_1, TS_2) = 1 - trendDist(TS_1, TS_2) = 1 - \frac{1}{N}\sum_{i=1}^{N} td(slope_{1i}, slope_{2i})$$

where:

TS_1 and TS_2 refer to the analyzed time series,
N is the number of line segments (i.e., the number of examined versions minus one), and $slope_{1i}$ and $slope_{2i}$ correspond to the slope of line segment i for the first and second time series, respectively, whereas *trendDist* refers to the distance between the two series.

It should be noted that the definitions in [40] have been slightly adapted to allow for an intuitive interpretation of similarity where a larger absolute value corresponds to higher similarity.

DOI: 10.1002/smr

The definition of the trend similarity measure ensures that the similarity is normalized in the range [0, 1]. The higher the value is, the more similar the time series are. To quantify the extent of cluttering for several time series, we calculate the average trend similarity, considering all pairs of time series:

$$AverageTrendSimilarity = \frac{1}{noPairs} \sum_{\substack{i,j=1 \\ i \neq j}}^{noPairs} trendSim\left(TS_i, TS_j\right)$$

where the number of time series pairs is equal to the number of their combinations.

An average trend similarity close to one implies a large similarity between the trends of the examined metrics, allowing a relatively easier interpretation of the evolution. On the other hand, highly cluttered trends lead to a low average trend similarity, prohibiting software stakeholders from extracting straightforward conclusions from the examination of metric values. It is in these cases that DEA helps the most by providing an overall picture for the evolution of quality, based on the selected metrics. In all case studies that will be presented in Section 5, the average trend similarity of the metrics is provided to indicate how difficult it would be to obtain an overall picture of the evolution without the use of DEA.

4. APPLICATION OF ANALYTIC HIERARCHY PROCESS

4.1. Brief description of Analytic Hierarchy Process

The Analytic Hierarchy Process, developed by Saaty [42], has been applied in a wide variety of MCDM problems. The AHP methodology consists of the following steps: (i) constructing the hierarchy of the components of the problem, which comprises a goal at the topmost level, criteria (and subcriteria in some cases) at the intermediate levels and the alternatives (or options) at the lowest level; (ii) priority setting of the criteria by pairwise comparisons, using an ordinal scale from 1 (equal importance) to 9 (extreme importance of a criterion compared to the other criterion in the pair); (iii) rating the options by pairwise comparisons of them on each criterion; and (iv) obtaining an overall relative score for each option. The main strengths of the methodology could be summarized to the following lines: a decision problem is decomposed through AHP into its constituent parts, stressing the importance of each criterion. The ability of the method to check inconsistencies and its convenience are also advantages of AHP over other multi-criteria methods. According to many researchers (e.g., [43]), AHP is one of the most reliable MCDM methods. Among the weaknesses of the method, which should be mentioned, is the substantial number of pairwise comparisons required ($\frac{mn(n-1)}{2}$, where m and n is the number of criteria and options respectively), which usually makes the application of AHP a lengthy task. AHP has been criticized as it does not adhere to rank reversal situations. A rank reversal is likely to occur when an irrelevant alternative is added to or removed from the set of existing options. However, several researchers (e.g., [44, 45]) have successfully addressed rank reversal situations, without causing a change in ranks of the options being evaluated. Broad areas where AHP has been successfully employed include the selection of one alternative from a set of alternatives, ranking, resource allocation, forecasting, quality management, business process reengineering, and so on.

4.2. Software context

In a similar but less subjective manner, AHP can also be applied with the goal of ranking successive versions of the same software project, according to the values of "output" design metrics and "input" size characteristics. AHP has also been used in previous approaches in order to assist metrics-based decision making [46]. However, it has not been applied in the context of evolution analysis while input and output metrics have been equally treated. The proposed procedure of using AHP to rank successive software versions is illustrated in Figure 4.

In order to use AHP as a means of validation against the results of DEA, we applied AHP using the same criteria, that is, the same set of input and output metrics as for DEA. However, as already

DOI: 10.1002/smr

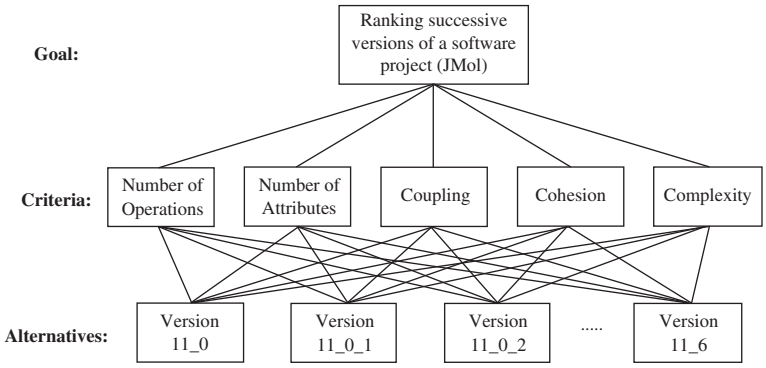

Figure 4. The procedure of AHP in the context of various versions of a software system.

mentioned, AHP requires the priority setting of the criteria by assigning weights to each one. To avoid any possible bias, priority weights have been obtained from a study relying on the opinion of an expert [46] for the first two case studies discussed in Sections 5.2.1 and 5.2.2. These priority weights indicate, for example, that complexity is regarded (according to that expert) as moderately more important than coupling, while size is regarded as slightly more important than coupling. For the research project discussed in Section 5.2.3, for which changeability was the feature of interest, the priority weights have been obtained from the authors of the SQO-OSS quality model [25]. For example, they conveyed that open-source developers consider cohesion as less important than coupling and depth of inheritance tree. For the industrial modules examined in Section 5.2.4, the priority weights were given by the lead software engineer involved in their maintenance. Given that the quality has been assessed by the type and number of faults, it was considered that design faults are more important than programming faults.

The Analytic Hierarchy Process requires also the rating of the alternatives (which, in our case, are the different versions of the examined project) for each criterion (which, in our case, are the selected metrics) by pairwise comparisons. Usually this rating scheme depends on the expertise of the evaluator. However, in the proposed study, it is possible to avoid the further use of subjective ratings because the alternatives, which are compared by AHP, are different versions of the same system. Therefore, an objective rating of the values can be obtained by uniform quantization over the range of values that each metric has for all of the examined versions. In other words, the version exhibiting the lowest (maximum) value for a specific metric obtains the lowest (highest) possible rating and all other versions are rated proportionally. For example, the complexity for the examined versions of JMol (presented in Section 5.2.1) ranges from 3.41 (lowest value) to 3.81 (highest value). As a result, for a nine-point measurement scale, which is usually employed in AHP, the *quantization step size* is equal to $\Delta = \frac{\max - \min}{9} = 0.044$.[1] Given that 3.41 is the optimum observed value, other values within the same step obtain a rating of "1", while values in the other end of the spectrum, which essentially differ drastically from 3.41, obtain a rating of "9". The formula for obtaining the index of the quantized value, for an observed value x is given by $Q_i[x] = 1 + \left\lfloor \frac{x - \min}{\Delta} \right\rfloor$. Because input metrics are treated under a different perspective than the output metrics, as already mentioned, the rating scheme for input metrics is the inverse for that of the output metrics. For example, the Number of Methods for the examined versions of JMol ranges from 4686 to 6461. Now, the highest value of 6461 is the optimum observed one, corresponding to a rating of "1".

Eventually, AHP considers both the priority setting of each criterion and the pairwise comparisons of the ratings for the examined software versions and extracts an overall score for each software version, which are essentially the alternatives in the context of AHP.

[1]A minor issue stems from the fact that uniform scalar quantization assumes that the entire range is inclusive of the lower bound and exclusive of the upper bound, i.e., range = [min, max). To avoid having the max value be assigned to the 10th region, the original max value is slightly increased by adding a very small value.

DOI: 10.1002/smr

5. CASE STUDIES AND DISCUSSION

5.1. Context

To investigate the suitability of DEA as a means for assessing the evolution of software projects employing several software metrics, we applied the aforementioned non-parametric technique to four different case studies, namely two open-source projects, one academic research project, and one industrial telecommunications system (results are given for two modules of the latter system). It should be noted that, in all cases, the super-efficiency DEA model has been applied.

The open-source systems on which DEA has been applied are JMol and JFreeChart. JMol is a free, open-source molecule viewer for students, educators, and researchers in chemistry and biochemistry, which has been constantly evolving since 2004. The selected project versions range from 11.0 to 11.6. In total, 26 successive project versions are included within this range. JFreechart is a rather mature open-source chart library, which has been constantly evolving since 2000. The selected project versions range from 0.9.0 to 0.9.21 (22 successive project versions).

The aspects that have been selected for the analysis of JMol include coupling (measured by Message Passing Coupling, MPC [47]), cohesion (measured by Lack of Cohesion of Methods, LCOM* [48–50] that extracts the percentage of methods that do not access a specific attribute of a class averaged over all attributes in that class), and complexity (Cyclomatic Complexity per Method [51]). The size characteristics have been captured by the NOO and number of attributes (NOA). These metrics have been selected because coupling, cohesion, and complexity are the most standard means for assessing the design quality of an object-oriented system. However, it should be stressed that the goal is to illustrate the suitability of DEA whose application is independent of the kind of metrics used.

To the same end, the metrics, which have been used for the analysis of JFreeChart are different and have been retrieved without any modification from [52] in order to further emphasize that DEA can be applied in any experimental setting regarding the selection of metrics. For JFreeChart, the output metrics are cohesion, fan-in and fan-out coupling [52], whereas the input metric reflecting the size characteristics is only the number of classes.

The research project that has been employed in the third case study is JDeodorant, which is an Eclipse plug-in allowing the detection and automatic resolution of code smells in object-oriented systems. It is the outcome of 4 years of development in the Software Engineering Group at the Department of Applied Informatics, University of Macedonia, Greece. Twenty revisions have been selected for the analysis reflecting the major changes in terms of functionality that have occurred in the project's history.

The aspect that has been selected in order to assess the evolution of JDeodorant with the passage of generations is changeability, which is one of the sub-characteristics defined in the ISO/IEC 9126 hierarchical quality model [53] for the characteristic of maintainability. Changeability refers to those attributes that reflect the effort required for software modification. According to a study that proposed a quality model targeting open-source software [10, 25] based on the opinions of open-source developers, changeability of object-oriented code can be assessed considering coupling, cohesion, and the depth of the inheritance tree for each of the examined classes. Therefore, in the context of software quality evolution analysis, we employed CBO [54], LCOM* [48], and Depth of Inheritance Tree [54] as measures of the corresponding internal qualities. In analogy to the first case study, the NOO and the NOA have been employed to capture the size characteristics of each examined version.

The fourth case study refers to the evaluation of two modules from a large-scale telecommunications software system implementing a digital switching product for public telecommunications networks. The system under study is being developed for more than 30 years and provides a multi-level platform supporting a range of applications including the provisioning of mobile telephony services. The particular modules that have been considered perform functions relevant to the handling of mobile application part operations exchange between nodes of a mobile telephony network. Each module communicates with other system modules within the platform by means of well-defined signal interfaces. These interfaces are associated with corresponding pieces of code that execute either after signal reception (in order to process the received message) or before signal sending (in order to prepare the message to be sent). The amount of functionality carried out by each module can therefore be accurately captured by the number of such interfaces.

DOI: 10.1002/smr

After contacting the team involved in the development and maintenance of these modules and explaining the capabilities of DEA, the software engineers have shown interest in assessing the evolution of quality in terms of the number of faults encountered in each module. Obviously, the number of faults cannot be assessed in isolation and should be normalized over the amount of functionality offered by each module. Because the employed issue tracking system enables the classification of faults found during testing, the goal was to assess the evolution of quality considering the presence of two types of faults, namely design and programming faults. Design faults are considered as more severe than programming faults because they are associated with the violation of design rules and/or incorrect or missing actions and might entail corrective actions in other system modules.

A rough rule of thumb regarding the application of DEA is that the number of DMUs should be equal to or greater than $\max\{n \times m, 3 \times (n+m)\}$, where n and m the number of inputs and outputs, respectively [20]. Because of the limited number of available versions for the two examined telecommunications modules (10 and 12, respectively) and considering the aforementioned rule, one input and two outputs have been selected for the application of DEA (three inputs/outputs require at least nine DMUs, and therefore the number of available versions is sufficient). As input, the number of interfaces for each version is considered, whereas the number of faults belonging to each type constitutes each of the two outputs. In this context, a module in a version might have y_1 faults of the first type and y_2 faults of the second type. However, as noted by Waldo [55], treating each quality level (e.g., design and programming fault categories) as a separate output dimension in the application of DEA, does not allow the distinction of DMUs. For example, version $v1$ with 10 design and 0 programming faults would not be distinguished from another version $v2$ exhibiting 0 design and 10 programming faults. To handle this problem, we adopt the approach by Olesen and Petersen [56] and use the cumulative number of faults. In other words, the cumulative number for programming faults includes the number of design faults, because a design fault is more important than a programming one. Referring to the previous example, version $v1$ would be recorded as having 10 design and 10 programming faults, whereas version $v2$ would be recorded as having only 10 programming faults, and thus a better quality (in terms of faults, assuming no change in the input characteristics of the two versions).

To validate the findings, which are extracted by DEA, which is a purely quantitative approach, we also used AHP, which principally has qualitative properties in terms of weights assigned to each criterion. The entire material required to perform DEA and AHP calculations (primary metric values used as inputs and outputs for DEA, relative importance of the criteria for AHP, and metric ranges for the AHP rating scheme) is available at [57].

5.2. DEA and AHP results

5.2.1. Case study 1: open-source project JMol. The results from the application of the super-efficiency DEA model to the examined versions of JMol are shown in Figure 5 (third diagram). In the same figure, the evolution of the employed input and output metrics is also illustrated to highlight that the graphical interpretation of multiple metrics with simultaneous changes in the size characteristics of the examined systems is a rather cumbersome task (top two diagrams). The visual clutter is also evident from the relatively low average trend similarity between the time series formed by the metrics' evolution, which is equal to 0.36. The bottom diagram in the figure displays the evolution of scores derived from AHP to enable a validation of DEA findings.

As it can be observed from Figure 5, there are five major time points at which drastic changes in the metric values and correspondingly, in the extracted combined measures, take place (indicated by vertical dashed lines). In the first point, corresponding to version 11.0.3 of project JMol, all output metric values are getting worse. Despite the fact that the size of the system also increases in this version, the overall picture obtained by the selected metrics indicates deterioration and this is captured vividly by both the DEA efficiency score, as well as the corresponding AHP score. From version 11.2.0 up to version 11.2.14, all input and output metrics and also the corresponding DEA and AHP scores are relatively stable. On the transition from version 11.2.14 to 11.4.0, cohesion and complexity are slightly improved; the size of the systems in terms of number of methods and

DOI: 10.1002/smr

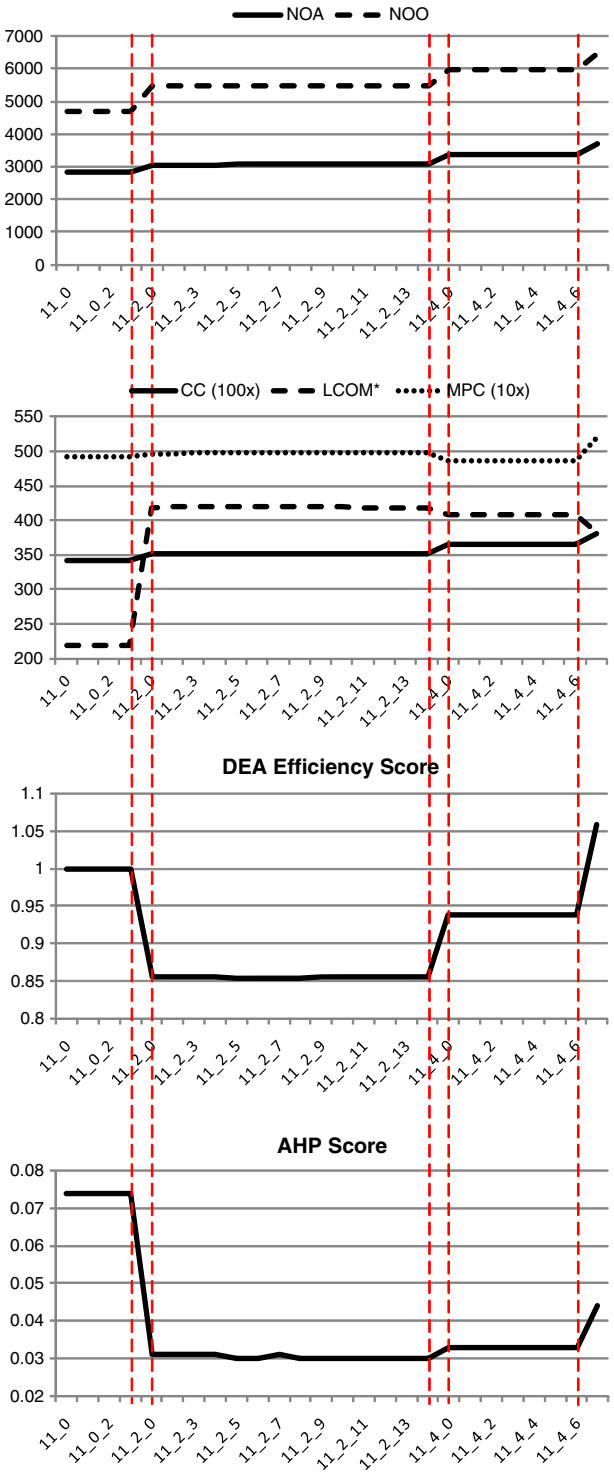

Figure 5. Evolution of software metrics, DEA and AHP scores for JMol.
*DEA efficiency scores, which appear to be equal to unity are in fact slightly different from each other.
**DEA efficiency scores have been obtained with DEA-Solver Pro v6.0 [58].
***Metric values have been obtained with Borland Together 2006 [59].
****AHP scores have been obtained with Expert Choice v11.5 [60].

attributes increases and only the complexity metric becomes worse. The latter factor is the only one that has a negative impact on the overall assessment of quality, whereas all the others are considered positive and therefore both the DEA efficiency score and AHP score record an improvement in this

DOI: 10.1002/smr

transition. A further improvement can be noticed in the last transition, which is caused by the increase of cohesion and of the input size measures.

Because the application of DEA and AHP leads to rankings of the examined versions, the rankings can be compared for their similarity. The most suitable measure for comparing the similarity between permutations of the same set (as in our case, where the set of elements contains the versions under study) is Spearman's footrule distance [61]. For two permutations σ_1 and σ_2 of the same set of elements S, Spearman's footrule distance is computed as $Fr^{|S|}(\sigma_1, \sigma_2) = \sum_{i=1}^{|S|} |\sigma_1(i) - \sigma_2(i)|$, where $\sigma(i)$ denotes the rank of the element i and $|S|$ denotes the size of set S. Ties are handled by assigning the mid position to the tied items. If, for example, there is a tie on the second, third, and fourth place, all versions are assigned the same rank "3" [62]. A normalized value for Spearman's footrule distance can be obtained by dividing the sum by its maximum value, which is equal to $\frac{1}{2}|S|^2$ when $|S|$ is even and $\frac{1}{2}(|S|+1)(|S|-1)$ when $|S|$ is odd. The resulting normalized measure is equal to 0 when the two rankings are identical and equal to 1 when lists appear in opposite order. Spearman's footrule distance between the rankings obtained by DEA and AHP for project JMol is equal to 0.237, whose low value provides a further validation for the applicability of DEA.

5.2.2. Case study 2: open-source project JFreeChart. The results from the application of the super-efficiency DEA model to the examined versions of JFreeChart are shown in Figure 6. The top diagram illustrates the evolution of the three output and the single input metric values, the central diagram displays the evolution of scores derived from DEA, whereas the bottom provides for validation the scores derived from AHP.

Whereas changes in the employed metric values are much more frequent in this case, major differentiations that affect the trends are indicated by the vertical dashed lines. The combined evaluation of the three output and the single input metric values is rather complicated in this case, hindering the human evaluator from reaching unambiguous conclusions regarding the overall trend of quality. The average trend similarity between the examined metrics is equal to 0.30 indicating an even higher degree of cluttering than for the previous project. As an example, in the first significant change corresponding to version 0.9.2, cohesion, which we want to reduce, starts to decrease (positive change), fan-out, which we want to reduce, starts also to decrease (positive change), fan-in, which we want to increase, also starts to descend (negative change), and at the same time the number of classes drastically increases, which in the context of ranking software versions should be interpreted as a positive sign (to put it simply, provides an excuse for the deterioration of output metrics). The combined view of all these effects, as a careful observer would notice, is rather an improvement in the design quality (captured by the selected metrics) and this is reflected by the change in the DEA efficiency score between the first two horizontal lines. The AHP score also captures the improvement, however as less intense, because weights assigned to each metric play an important role. The overall trends depicted by the evolution of the DEA and AHP scores are analogous. For JFreeChart, Spearman's footrule distance between the rankings obtained by DEA and AHP is even lower than for JMol (equal to 0.165), signifying a strong correlation between the results of the two methods.

5.2.3. Case study 3: research project JDeodorant. The obtained scores from the application of DEA and AHP to the 20 examined revisions of JDeodorant are shown in Figure 7 (bottom figure) along with the evolution of the examined size metrics and the metrics corresponding to the sub-characteristic of changeability (upper figure). The degree of visual cluttering among the individual trends is lower in this case, because the average trend similarity between the examined metrics is equal to 0.508. However, it is still difficult to interpret the individual trends and extract an overall conclusion regarding the evolution of changeability.

From the evolution of the DEA efficiency score over the examined revisions, it becomes evident that this particular aspect of quality, related to changeability, has been continuously improving. Given that the authors had direct contact with the developers of this project, this comes as no surprise, because the development team strived for design optimization by applying refactoring actions regularly. The evolution of the scores derived from the application of AHP on the examined set of data validated this continuous improvement. The Spearman's footrule distance between the rankings obtained by

DOI: 10.1002/smr

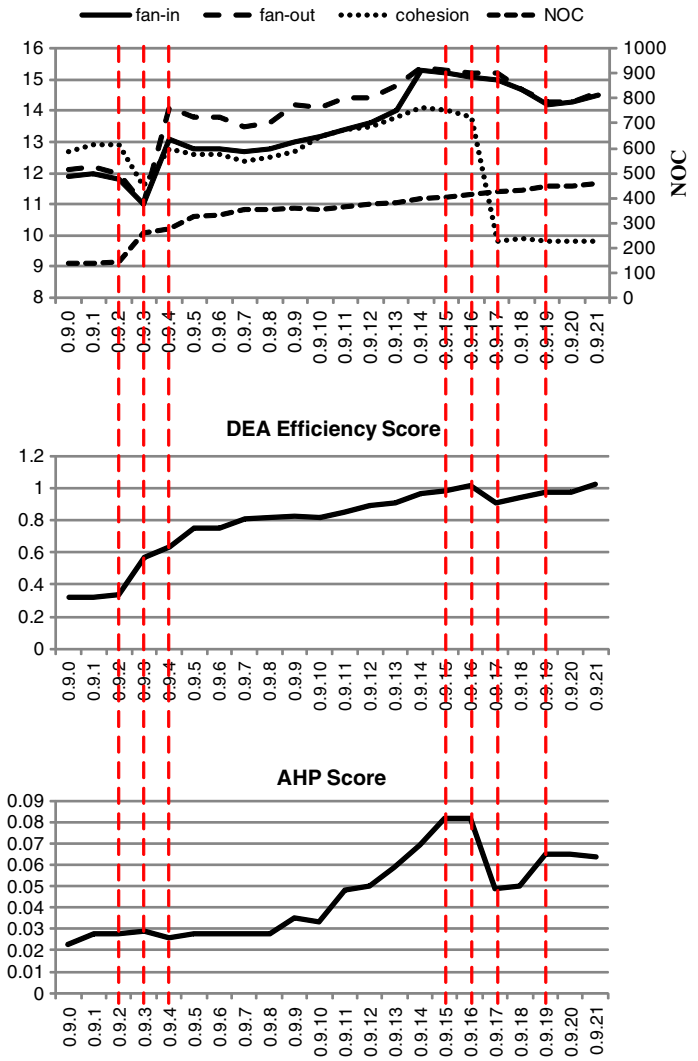

Figure 6. Evolution of software metrics, DEA and AHP scores for JFreeChart.

DEA and AHP for JDeodorant is equal to 0.195, indicating that DEA and AHP lead to very similar assessment of the overall evolution of changeability.

If a metrics combination approach is successful in capturing several metrics, which are associated with the same quality property (such as changeability in this case), then it should be demonstrated that the combined value is highly correlated to the desired property. However, as already mentioned, quality sub-characteristics are not directly quantifiable; that is why a number of metrics are required to assess them.

To confront this problem, we employed secondary measures, which are related to some extent to the quality property of interest. In this way, it becomes possible to investigate whether the evolution of efficiency scores derived by the proposed DEA approach matches the evolution of these measures and by inference, the evolution of the examined property.

In discussions with the development team of JDeodorant, it was agreed that a secondary measure, which can provide insight into the evolution of changeability (without resorting to the aforementioned metrics), is the number of added/modified lines of code to fix a single bug. This measure would be ideal if bugs were of exactly the same type. To mitigate this threat, only bugs whose fix concerned a single class have been selected. Figure 8 shows the evolution of the number of added/modified lines of code to fix a single bug (trendline).

The preceding trend provides further confidence into the validity of the extracted DEA scores, which indicate that changeability increased gradually with the passage of versions. The same trend is depicted

DOI: 10.1002/smr

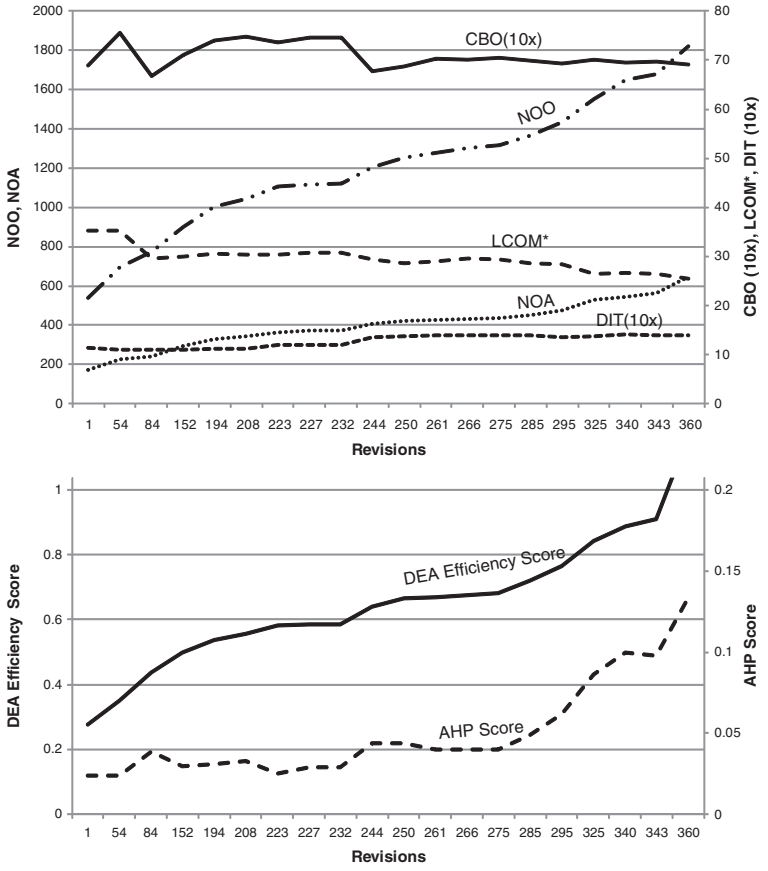

Figure 7. Evolution of software metrics, DEA and AHP scores for JDeodorant.
*Metric values have been obtained with Understand 2.6 [63].

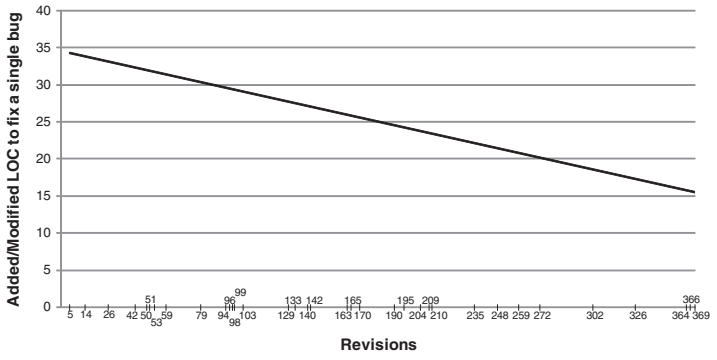

Figure 8. Number of added/modified LOC to fix a single bug (Trendline).

from the fact that the number of lines that have to be added or changed in order to fix a single bug in JDeodorant, has been reduced significantly over time.

5.2.4. Case study 4: industrial telecommunications system. To enable the presentation of the results for the two examined telecommunication modules (without referring to actual module names or version numbers), the first module will be denoted as *A* (having 12 versions) and the second as *B* (having 10 versions). The evolution of input and output measures (number of interfaces and design/programming faults) for module *A* is shown in the upper plot of Figure 9, whereas the scores

DOI: 10.1002/smr

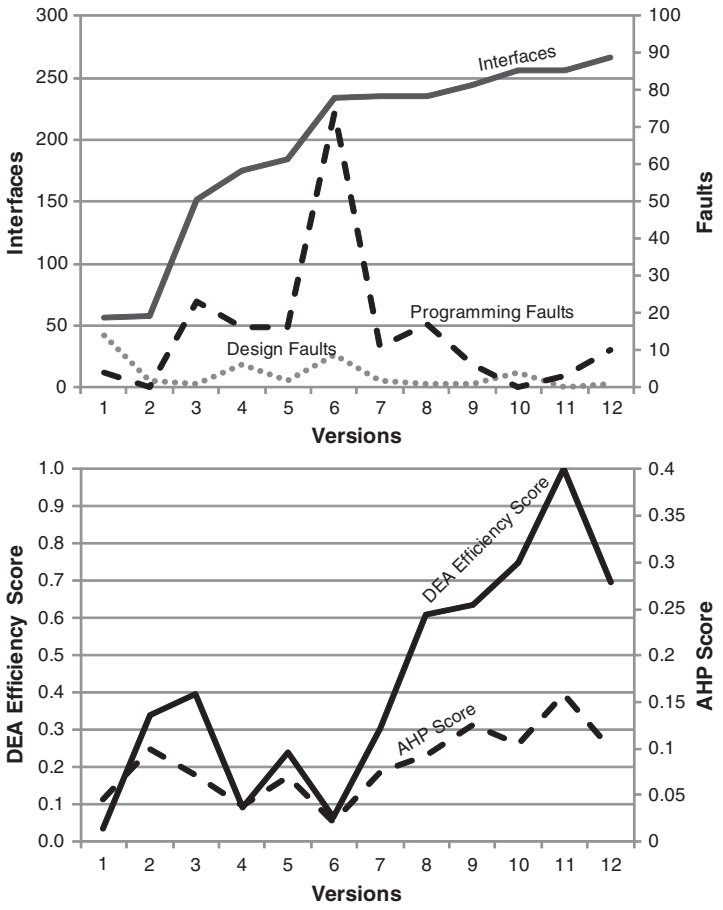

Figure 9. Evolution of input/output measures, DEA and AHP scores for module *A*.

derived from the application of DEA and AHP are shown in the bottom plot. The corresponding results for module *B* are shown in Figure 10.

The degree of visual cluttering among the individual trends is higher in the first module. In the second module, the number of interfaces and the number of faults appear to increase with the passage of versions enabling a rather easier interpretation of the overall trend in quality. This is validated by the average trend similarity, which is equal to 0.364 and 0.667 in the first and the second module, respectively.

The evolution of the DEA efficiency score over the examined versions reveals an improvement of quality for the first module and a deterioration of quality for the second one. For example, in module *A*, the last examined versions exhibit a relatively low number of programming and design faults, which, in combination with the significantly larger size in terms of delivered functionality, yield the observed large efficiency scores. On the contrary, in the second module *B*, the number of design and programming faults is constantly increasing in the last versions, leading to lower efficiency scores. The simultaneous concurrent increase in delivered amount of functionality is not sufficient to justify an improvement or stabilization of quality. These observations are in agreement with the opinion of the software engineers involved in the development and maintenance of the corresponding products regarding the evolution of quality considering the number of faults as indicators.

The application of AHP on the examined modules validates the findings derived by DEA as the corresponding score evolution matches closely the evolution of DEA efficiency scores, leading to almost identical rankings of the examined versions. The Spearman's footrule distance between the rankings obtained by DEA and AHP is equal to 0.222 and 0.08 for modules *A* and *B*, respectively.

Because there is no single property that can be related to the number of design and programming faults, the engineers suggested to assess the validity of the extracted scores by means of the number of faults having a priority beyond a certain level. Trouble reports, in the context of that company,

DOI: 10.1002/smr

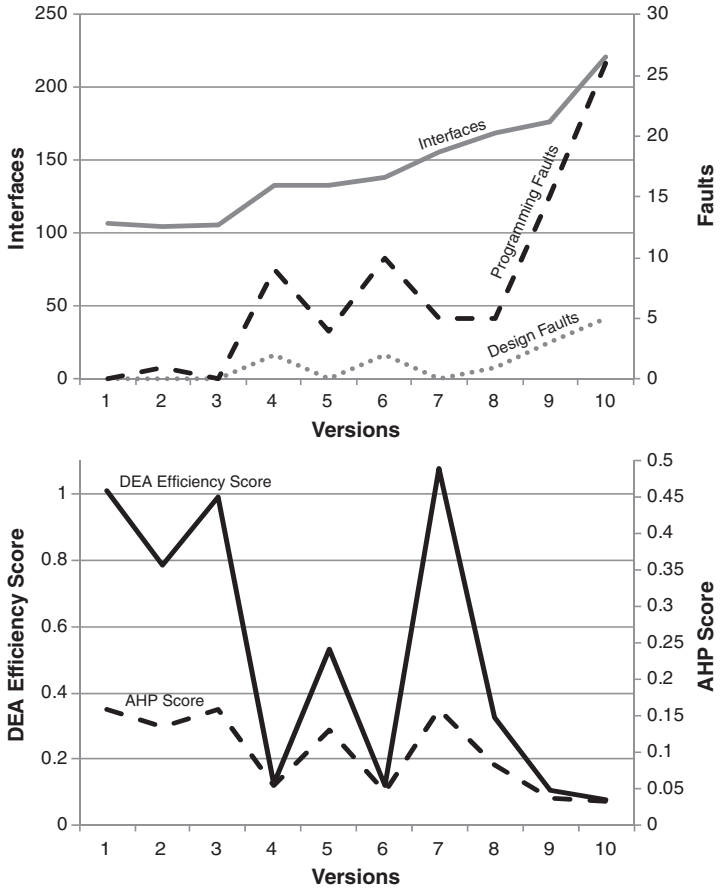

Figure 10. Evolution of input/output measures, DEA and AHP scores for module *B*.

are initially given one of three priority levels (*A*, *B* or *C*), with *A* representing the higher priority. The priority level is set based on the impact of that problem. Because only *A* and *B* priority levels have to be handled immediately (*C* level priority problems can be resolved in a future version), the engineers suggested that the number of problems with priority *A* or *B* should be used as a secondary measure of quality improvement, related to the original measures of design and programming faults.

Figure 11 shows the evolution of *A* and *B* priority level trouble reports over the examined versions (trendline) of module *A*. As it can be observed, the number of faults having a priority level of *A* or *B* is gradually decreasing over time. The fact, that the overall trend for the combined view of quality

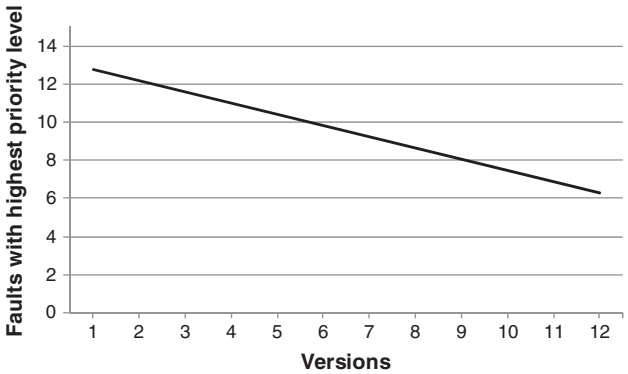

Figure 11. Number of faults with two highest priority levels (Trendline).

DOI: 10.1002/smr

(as extracted by DEA, shown in Figure 9) approaches the trend of high-priority faults (in an inverse manner), provides a partial validation for the aforementioned DEA findings.

According to the results for all examined case studies, DEA appears to be a promising alternative for evaluating and ranking several versions in the evolution of the same software project. The unification of several metrics in a single view and the consideration of size characteristics, without the need to resort to experts, provide a fair and informative way of observing whether the design quality of a system improves or deteriorates.

6. LIMITATIONS – THREATS TO VALIDITY

6.1. Limitations

The application of DEA in order to assess the evolution of quality over a number of successive software versions is subject to the following limitations, some of which are caused by the nature of the approach and the others are related to the application domain. As already mentioned, to extract reliable results from DEA, a minimum number of DMUs should be considered. In the context of software, this means that very short evolution periods, with a limited number of available versions, cannot be assessed effectively. Concerning the number of inputs and outputs to the approach, DEA can handle any number of them. However, as the number of inputs/outputs increases, the discriminative power of DEA becomes weaker. In other words, the consideration of a large set of design metrics as outputs would not lead to a sharp discrimination among the examined versions.

Concerning the application of DEA in the context of software, non-functional requirements cannot be handled, because DEA requires a quantitative measure for each input. Consequently, differences between software versions in the number of implemented non-functional requirements might get unnoticed. However, different versions of the same system are expected to exhibit less drastic changes in the set of implemented non-functional requirements than different software systems.

6.2. Threats to validity

The validity of the application of DEA in the field of design quality evaluation is subject to the following threats, including threats to construct internal and external validity [64].

Concerning the extent to which the model on which DEA operates actually reflects the real conditions (construct validity), the use of size measures as a surrogate for the amount of offered functionality (provided by measures, such as function points), might provide an inaccurate estimation of inputs. As explained in Section 3.2, size metrics can only be considered as reasonable inputs to the final design phase, when most of the behavior and state information of the system have been consolidated. The fact that non-functional requirements cannot be used as inputs to DEA might also affect the accuracy by which changes between different software versions are captured in the model.

Regarding factors that may affect the relationships that we are trying to investigate (internal validity), a valid threat is that important inputs and most probably outputs might have been ignored. The selection of a limited number of metrics certainly does not capture all aspects of the design quality of the examined versions. However, depending on the quality assessment strategies employed in each project, a small number of selected metrics is usually sufficient for evaluating a specific sub-characteristic (such as analyzability or testability) of a hierarchical quality model. Another threat is related to the fact that DEA does not impose user-selected weights on any of the metrics, whereas quality monitoring might put higher emphasis on some aspects of the design. However, according to the fundamental principles behind DEA, one of the conditions that must be satisfied to ensure meaningful results is the selection of outputs, which do not have tremendous differences in terms of their importance for the assessed DMUs.

Finally, the results from the application of DEA suffer from the usual threats to the external validity, that is, the factors that limit the ability to generalize the DEA findings. However, the goal of the proposed work is not to provide an indisputable assessment for the evolution of quality of the examined projects but rather to illustrate the application of DEA in a software context.

DOI: 10.1002/smr

7. CONCLUSIONS

In this paper, we have illustrated the use of DEA in order to rank various versions of a software system based on the values of design metrics and size characteristics. The main benefit obtained from the application of DEA, which is normally employed in an economic context, is the ability to provide a unified view of several metrics and the normalization of the derived scores over the size properties of the examined systems. Eventually, DEA leads to an efficiency score for each examined project enabling the ranking of various versions and the analysis of the evolution.

Results for two open source, one research and two industrial case studies, for which different design/quality metrics and input characteristics have been employed, indicate that DEA and particularly the super-efficiency model provides a promising alternative for the combination of metrics. The findings from another widely used multi-criteria decision analysis approach, namely AHP, validate the efficiency scores obtained by DEA. However, in contrast to AHP, DEA does not rely on weights for the employed criteria, allowing a more objective and automated means of evaluation.

REFERENCES

1. Fenton N, Pfleeger SL. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Company: Boston, 1997.
2. Harrison R, Counsell SJ, Nithi RV. An evaluation of the MOOD set of object-oriented software metrics. *IEEE Transactions on Software Engineering* 1998; **24**(6):491–496.
3. Subramanyam R, Krishnan MS. Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects. *IEEE Transactions on Software Engineering* 2003; **29**:297–310.
4. Gyimóthy T, Ferenc R, Siket I. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering* 2005; **31**(10): 897–910.
5. Kitchenham BA, Pickard LM, Linkman SJ. An evaluation of some design metrics. *Software Engineering Journal* 1990; **5**(1): 50–58.
6. Alshayeb M, Li W. An empirical validation of object-oriented metrics in two different iterative software processes. *IEEE Transactions on Software Engineering* 2003; **29**(11):1043–1049.
7. Bansiya J, Davis CG. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering* 2002; **28**(1):4–17.
8. Fenton N. New Directions for Software Metrics. CIO Symposium on Software Best Practices. Keynote Talk. September 2006. http://www.eecs.qmul.ac.uk/~norman/papers/new_directions_for_sw_metrics.pdf [25 January 2011].
9. Kozlov D, Koskinen J, Sakkinen M, Markkula J. Assessing maintainability change over multiple software releases. *Journal of Software Maintenance and Evolution: Research and Practice* 2008; **20**(1):31–58.
10. Samoladas I. Methods for Maintainability and Survival Analysis of Open-Source Software. Ph.D. Thesis, Department of Computer Science, Aristotle University of Thessaloniki, 2011.
11. Correia JP, Kanellopoulos Y, Visser J. *A survey-based study of the mapping of system properties to ISO/IEC 9126 maintainability characteristics*. 25th IEEE International Conference on Software Maintenance (ICSM'2009). Edmonton, Canada. September 2009; 61–70.
12. Spinellis D, Gousios G, Karakoidas V, Louridas P, Adams PJ, Samoladas I, Stamelos I. Evaluating the quality of open source software. *Electronic Notes in Theoretical Computer Science* 2009; **233**:5–28.
13. Serebrenik A, van den Brand M. *Theil index for aggregation of software metrics values*. 26th IEEE International Conference on Software Maintenance (ICSM'2010). Timisoara, Romania. September 2010; 1–9.
14. Samoladas I, Stamelos I, Angelis L, Oikonomou A. Open source software development should strive for even greater code maintainability. *Communications of the ACM* 2004; **47**(10):83–87.
15. Lehman MM, Ramil JF. Rules and tools for software evolution planning and management. *Annals of Software Engineering* 2001; **11**(1):15–44.
16. Gîrba T, Ducasse S, Lanza M. *Yesterday's weather: guiding early reverse engineering efforts by summarizing the evolution of changes*. 20th IEEE International Conference on Software Maintenance (ICSM'04). Chicago, IL. September 2004; 40–49.
17. Chatzigeorgiou A, Manakos A. *Investigating the evolution of bad smells in object-oriented code*. 7th International Conference on the Quality of Information and Communications Technology (QUATIC'2010). Porto, Portugal. September–October 2010; 106–115.
18. Kagdi H, Collard ML, Maletic JI. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice* 2007; **19**(2):77–131.
19. Charnes A, Cooper WW, Rhodes E. Measuring the efficiency of decision making units. *European Journal of Operational Research* 1978; **2**(6):429–444.
20. Cooper WW, Seiford LM, Tone K. *Introduction to Data Envelopment Analysis and its Uses: With DEA-Solver Software and References*. Springer: New York, 2005.

DOI: 10.1002/smr

21. Chatzigeorgiou A, Stiakakis E. Benchmarking library and application software with data envelopment analysis. *Software Quality Journal* 2011; **19**(3):553–578.
22. Benestad HC, Anda B, Arisholm E. Assessing software product maintainability based on class-level structural measures. 7th International Conference on Product-Focused Software Process Improvement (PROFES'2006). Amsterdam, Netherlands. June 2006.
23. Oman P, Hagemeister J. Construction and testing of polynomials predicting software maintainability. *Journal of Systems and Software* 1994; **24**:251–266.
24. Morisio M, Stamelos I, Tsoukias A. Software product and process assessment through profile based evaluation. *International Journal of Software Engineering and Knowledge Engineering* 2003; **13**:495–512.
25. Samoladas I, Gousios G, Spinellis D, Stamelos I. *The SQO-OSS quality model: measurement based open source software evaluation*. 4th International Conference on Open Source Systems (OSS'2008). Milan, Italy. September 2008; 237–248.
26. Briand L, Wüst CJ, Daly JW, Porter DV. Exploring the relationship between design measures and software quality in object-oriented systems. *Journal of Systems and Software* 2000; **51**:245–273.
27. Harman M, Tratt L. *Pareto optimal search based refactoring at the design level*. 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'2007). London, UK. July 2007; 1106–1113.
28. O'Keeffe M, O'Cinnéide M. Search-based software maintenance. *Journal of Systems and Software* 2008; **81**(4):502–516.
29. Basdavanos M, Chatzigeorgiou A. Placement of entities in object-oriented systems by means of a single-objective genetic algorithm. 5th International Conference on Software Engineering Advances (ICSEA'2010). Nice, France. August 2010.
30. Bowman M, Briand L, Labiche Y. *Multi-objective algorithms to support class responsibility assignment*. 23rd IEEE International Conference on Software Maintenance (ICSM'2007). Paris, France. October 2007; 124–133.
31. Charnes A, Cooper WW, Lewin AY, Seiford LM. *Data Envelopment Analysis: Theory, Methodology, and Application*. Kluwer Academic Publishers: Norwell, MA, 1996.
32. Coelli TJ, Prasada Rao DS, O'Donnell CJ, Battese GE. *An Introduction to Efficiency and Productivity Analysis* (2nd edn). Springer: New York, 2005.
33. Korpela J, Lehmusvaara A, Nisonen J. Warehouse operator selection by combining AHP and DEA methodologies. *International Journal of Production Economics* 2007; **108**:135–142.
34. Stewart TJ. Relationships between data envelopment analysis and multicriteria decision analysis. *Journal of the Operational Research Society* 1996; **47**(5):654–665.
35. Cooper WW, Seiford LM, Zhu J. *Handbook on Data Envelopment Analysis*. Kluwer Academic Publishers: New York, 2004.
36. Albrecht AJ. Measuring application development productivity. Proceedings of IBM Applications Development Symposium. Monterey, CA. October 1979.
37. Andersen P, Petersen NC. A procedure for ranking efficient units in data envelopment analysis. *Management Science* 1993; **39**:1261–1264.
38. Javed W, McDonnel B, Elmqvist N. Graphical perception of multiple time series. *IEEE Transactions on Visualization and Computer Graphics* 2010; **16**:927–934.
39. Gunopoulos D, Das G. Time series similarity measures and time series indexing. Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD'01). Santa Barbara, CA. May 2001.
40. Li G, Wang Y, Zhang L, Zhu X. *Similarity measure for time series based on piecewise linear approximation*. Proceedings of the International Conference on Wireless Communications & Signaling Processing (WCSP 2009). Nanjing, China. November 2009; 1–4.
41. Holzhüter C, Hadlak S, Schumann H. Multi-level visualization for the exploration of temporal trends in simulation data. Winter Simulation Conference (WSC 2010). Baltimore, MD. December 2010.
42. Saaty T. A scaling method for priorities in hierarchical structures. *Journal of Mathematical Psychology* 1977; **15**(3):234–281.
43. Forman EH, Gass SI. The Analytic Hierarchy Process – an exposition. *Operations Research* 2001; **49**(4):469–486.
44. Wang YM, Elhag TMS. An approach to avoiding rank reversal in AHP. *Decision Support Systems* 2006; **42**(3):1474–1480.
45. Forman EH. Facts and fictions about the Analytic Hierarchy Process. *Mathematical and Computer Modelling* 1993; **17**(4–5):19–26.
46. Ahmad N, Laplante PA. *Employing expert opinion and software metrics for reasoning about software*. 3rd IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC'2007). Columbia, MD. September 2007; 119–124.
47. Li W, Henry S. Object-oriented metrics that predict maintainability. *Journal of Systems and Software* 1993; **23**(2):111–122.
48. Henderson-Sellers B, Constantine LL, Graham IM. Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design). *Object-Oriented Systems* 1996; **3**(3):143–158.
49. Briand LC, Daly J, Porter V, Wüst J. *A comprehensive empirical validation of design measures for object-oriented systems*. Proceedings of the 5th International Symposium on Software Metrics (METRICS'98). Bethesda, MD. March 1998; 246–257.
50. Briand L, Daly J, Wüst J. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering Journal* 1998; **3**(1):65–117.

DOI: 10.1002/smr

51. McCabe TJ. A complexity measure. *IEEE Transactions on Software Engineering* 1976; **SE-2**(4):308–320.
52. Lee Y, Yang J, Chang KH. *Metrics and evolution in open source software*. 7th International Conference on Quality Software (QSIC'2007). Portland, OR. October 2007; 191–197.
53. International Organization for Standardization/International Electrotechnical Commission. ISO/IEC 9126 – Software Engineering – Product Quality, Geneve, 2001.
54. Chidamber SR, Kemerer CF. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 1994; **20**(6):476–493.
55. Waldo S. Efficiency in public education. Nationalekonomiska Institutionen, Lund University, Working Paper Series, 2002. http://www.nek.lu.se/publications/workpap/Papers/WP02_10.pdf [27 May 2011].
56. Olesen O, Petersen N. Incorporating quality into data envelopment analysis: a stochastic dominance approach. *International Journal of Production Economics* 1995; **39**:117–135.
57. DEA_AHP Material. http://java.uom.gr/OOD/DEA_AHP_Material.pdf [2 October 2011].
58. SAITECH Inc. – Products – DEA Solver PRO. http://www.saitech-inc.com/Products/Prod-DSP.asp [20 September 2010].
59. Software Architecture Design. Visual UML & Business Process Modeling. http://www.borland.com/us/products/together/index.aspx [20 September 2010].
60. Robust Decision-Support and Analysis Platform – Expert Choice. http://www.expertchoice.com/products-services/expert-choice-115 [20 September 2010].
61. Bar-Ilan J, Mat-Hassan M, Levene M. Methods for comparing rankings of search engine results. *Computer Networks* 2006; **50**(10):1448–1463.
62. Bar-Ilan J. Rankings of information and library science journals by JIF and by h-type indices. *Journal of Informetrics* 2010; **4**:141–147.
63. Understand your Code. http://www.scitools.com/ [15 May 2011].
64. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. *Experimentation in Software Engineering: An Introduction*. Kluwer: Boston, MA, 2000.

AUTHORS' BIOGRAPHIES

Alexander Chatzigeorgiou is an assistant professor of Software Engineering in the Department of Applied Informatics at the University of Macedonia, Thessaloniki, Greece. He received the Diploma in Electrical Engineering and the PhD degree in Computer Science from the Aristotle University of Thessaloniki, Greece, in 1996 and 2000, respectively. From 1997 to 1999, he was with Intracom, Greece, as a telecommunications software designer. His research interests include object-oriented design, software maintenance and metrics. He is a member of the IEEE.

Emmanouil Stiakakis is a lecturer in Digital Economics at the Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece. He holds a BSc in Mechanical Engineering from the Aristotle University of Thessaloniki, an MSc in Manufacturing Systems Engineering from Cranfield University, UK, and a PhD in Applied Informatics from the University of Macedonia. His research interests include production and operations management, Total Quality Management, e-business, and digital economy. His research has been published in international journals and conference proceedings.

DOI: 10.1002/smr