Short Paper

Guidelines for the application of Data Envelopment Analysis to assess evolving software

Alexander Chatzigeorgiou

Department of Applied Informatics, University of Macedonia
54006 Thessaloniki, Greece
Email: achat@uom.gr

Abstract. The assessment of software evolution in terms of quality poses significant challenges as different metrics have to be combined and normalized over the size of each examined version. Data Envelopment Analysis (DEA), a non-parametric technique from production economics, can offer a unified view of several design properties providing insight into global evolutionary trends. In this paper a set of practical guidelines for the application of DEA and the interpretation of the extracted results is proposed, with a focus on open source software, where limited information and documentation might be available.

Keywords: software evolution, software metrics, Data Envelopment Analysis

1 Introduction

Continuous metric tracking and change monitoring of evolving software is a key factor to ensure high quality [6] and to identify possible symptoms of software aging and degrading design. One of the most important challenges that quality assurance has to confront is the *combination* of several metrics that express different aspects of code and design quality into a single unified measure that captures the global trend in the evolution of software. In other words, even if a certain set of metrics (such as complexity and coupling) is agreed to serve as quantifiable properties expressing higher level qualitative properties (such as analyzability and changeability) [7] it is non-trivial to combine the different metric values into a single level or rank.

Data Envelopment Analysis (DEA) [2] is a non-parametric technique, usually employed in production economics to rank different companies based on their performance as captured by financial indicators. In the context of economics, the parameters for the analysis can be either outputs of the production process, such as profit and sales or inputs, such as personnel and raw materials. Beyond the ability to provide a single numerical value for the efficiency of each company, DEA offers the advantage of normalizing the extracted efficiency scores over the input variables.

We have previously shown that DEA can be effectively ported to the domain of software quality assessment and can be useful for comparing the design of open source libraries and application software [3] and for assessing multiple versions of the same project [4]. In this paper a number of guidelines are suggested, which according

adfa, p. 1, 2011.

to our experience facilitate the application of DEA for software evolution analysis and investigate the impact of various decisions or missing data. The involved concepts are exemplified through the results on an open-source chart library, namely JFreeChart, of which 22 successive versions have been analyzed (from 0.9.0 to 0.9.21). It should be stressed that the results are indicative and emphasis is not placed on their interpretation for this particular project but rather on the advantages that DEA can offer. The proposed guidelines throughout the paper are marked in bold and italics.

2 Brief Introduction to DEA

Let us consider two companies *A* and *B* which are assessed by their annual profit (output of the process) and the personnel employed by each company (input). Assuming that the profit of *B* is larger than the profit of *A,* one cannot simply claim that *B* is more efficient than *A*. The reason is that, for example, the higher profits of *B* might have been achieved with ten times the number of employees of *A*. Efficiency could be trivially obtained in this case as the ratio of output over input. However, efficiency computation becomes a non-trivial issue when numerous factors should be considered simultaneously. DEA calculates a so-called "efficient frontier" which is a mathematical space formed by all efficient units. The degree of inefficiency of all other units is obtained by their distance from the efficient frontier [2].

In a software context, the goal is to compare and rank different software versions according to their design quality. Each version can be assessed by a number of metrics, serving as outputs of the design process under evaluation. In other words, design metrics are the outputs that the designers wish to maximize. However, the size properties (or the offered functionality) of each version should also be considered. The reason can be stated through an example: if we assume that two versions s_i and s_{i+1} achieve the same value for a particular metric (e.g. complexity), one would consider the larger one better-designed, in the sense that despite the larger code base or number of offered features, the designers still managed to keep complexity at the same level.

To summarize, we can employ DEA to rank successive software versions simply by providing as outputs the design metrics of interest and as input(s) the size characteristics of each version. Eventually, DEA extracts a single efficiency score for each version. An efficiency score equal to 1 indicates that a version is a good as it could be, according to the selected criteria. Efficiency scores less than 1 imply that there is room in improving metric values or that the version achieved certain levels for the design metrics but is rather small in size, compared to other versions. Results of the application of DEA on actual open-source and industrial systems can be found in [4].

3 Normal DEA Application

Let us assume that metric values characterizing the design qualities of interest are available for a number of consecutive software versions that we wish to analyze. For our example we have used as outputs, metric values concerning cohesion, fan-in and fan-out, retrieved without any modification from [5] in order to emphasize that DEA

can be applied in any experimental setting. Since no information regarding quantitative measures of the implemented functionality in each version (such as Function Points) is available, we have used as substitute input metric the number of classes (NOC). The evolution of these metrics for the examined versions is shown in Fig. 1.

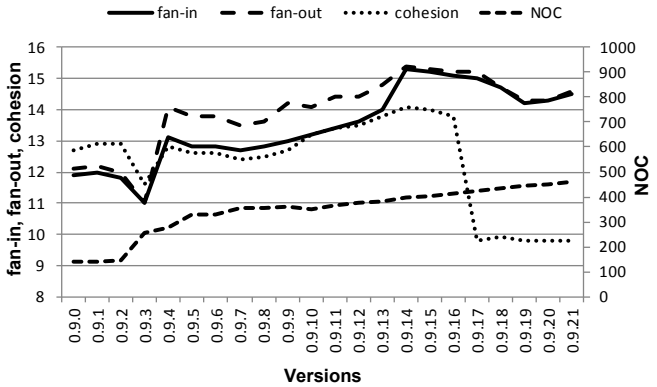

Fig. 1. Evolution of selected software metrics for JFreeChart

When the trends of individual metrics are not similar during periods of the software history it becomes difficult to derive a single representative trend. The problem is analogous to the extraction of a single trend of stock exchange progress when the trends of several individual stocks have to be considered. We have shown [4] that the added value of DEA depends on how much the series of metrics data are convoluted:

Guideline 1. DEA is suitable when the trends of individual metrics are convoluted

To run DEA one should simply provide the metric values for each version, designating the inputs (I) or outputs (O). In the usual setting, DEA assumes that the goal is to maximize each output. However, for many metrics it is desirable to minimize their values (such as cohesion and fan-out). To allow for a proper handling of the corresponding variables by DEA, the simplest solution is to invert the corresponding values:

Guideline 2. When the goal is to minimize an output (metric) the corresponding values should be inverted before the application of DEA

The application of DEA extracts an efficiency score for each version allowing a full characterization of software evolution. For JFreeChart the results are shown in Fig. 2.

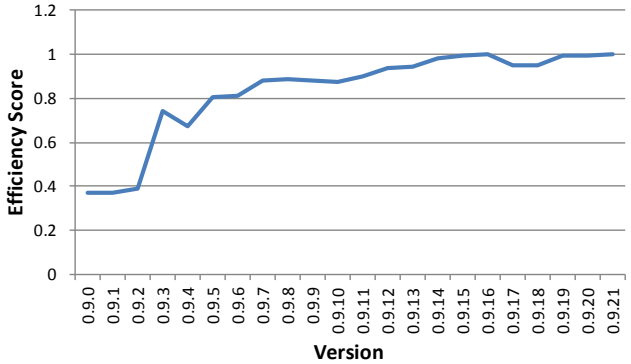

Fig. 2. Evolution of DEA efficiency score (basic model)

The application of the simple DEA model might be not ideal in some cases when the approach extracts multiple efficient units (i.e. units with an efficiency score equal or very close to 1). In these cases it would be harder to discriminate among the examined versions. For example, in the evolution of DEA efficiency scores for JFreeChart shown in Fig. 2, it appears that several versions obtain a score that is very close to 1.
Under these circumstances an alternative is to use the super-efficiency DEA model [1] which is capable of providing a full ranking by differentiating between the efficient units. This is achieved by excluding the efficient unit under evaluation from the efficient frontier. The effect of this is to shrink the frontier, allowing the efficient unit to become super-efficient since it now has a score greater than unity. The user simply has to select the corresponding model in the tool that he employs:
Guideline 3. When the application of the basic DEA model leads to multiple versions with an efficiency score equal or close to one, the super-efficiency DEA model should be used instead.
The application of the super-efficiency DEA model on the examined versions of JFreeChart leads to the efficiency scores shown in Fig. 3 (to illustrate the effect of selecting a different model the DEA results from the basic model are also shown).

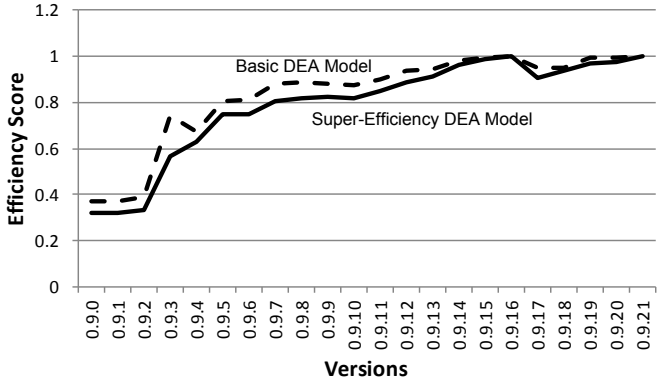

Fig. 3. Evolution of DEA efficiency score: Super-efficiency vs. basic model

4 Selection of Outputs

It is reasonable to assume that the selection of metrics plays an extremely important role in the assessment of software quality. The selection of output variables unavoidably impacts the extracted efficiency scores obtained by DEA and as a result the overall trend that is depicted as evolution of software quality. To illustrate this and to emphasize the importance of selecting the most appropriate metrics, a 3-stage experiment has been performed, where one of the three output metrics is excluded from the set of outputs at each stage. The results are shown in Fig. 4 (the super-efficiency DEA model is employed). To allow a comparison to the results obtained when all output metrics are considered, the initial efficiency scores of Fig. 2 are also shown.

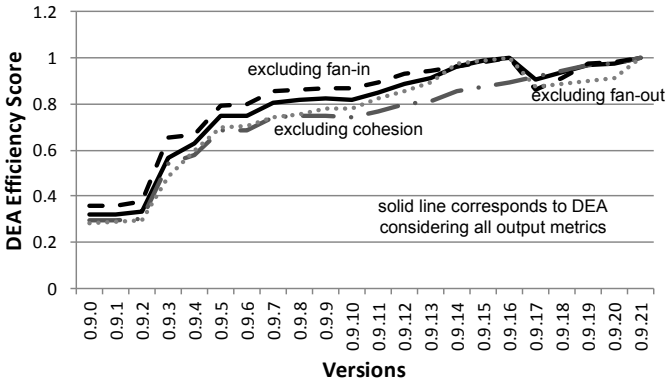

Fig. 4. Impact of excluding output metrics on the DEA efficiency scores

As it can be observed, the overall trend and ranking of the examined versions remains rather unaltered when excluding one out of the three considered output metrics from the evaluation. However, subtle differences can be observed and since the application of DEA is simple and effortless one should possibly investigate the various options by experimentation:

Guideline 4. The output variables can have varying impact on the extracted DEA scores. When not sure about the necessity to consider a metric, experimentation could highlight whether its impact is significant or not.

In general, DEA can handle any number of outputs. However, as the number of considered outputs increases, the discriminative power of the approach becomes weaker. Consequently, the consideration of a large set of metrics would not lead to a sharp discrimination among the examined versions. The aforementioned guideline might be valuable for reducing the set of examined metrics.

5 Application in case of no inputs

As already mentioned, DEA has the ability to "normalize" the efficiency over the size characteristics of each version. However, the impact of considering size might be large when there are large differences between versions, shadowing the effect of other metrics. Moreover, the use of substitute size measures instead of the amount of offered functionality might not always be the right choice as size metrics, such as number of classes or methods are also dependent on the design decisions. In these cases it might be preferable to ignore inputs and this can simply be done in DEA by providing a constant input to all examined versions, zeroing the effect of input variables to the model. In Fig. 5 the obtained DEA efficiency scores are shown when a constant input with value 1 is assumed for each version. To allow the comparison to the results obtained when input is considered, the initial DEA efficiency scores are also depicted.

The impact of the input on the obtained scores and the ranking of versions is significant, justifying the use of DEA as an approach for software evolution analysis:

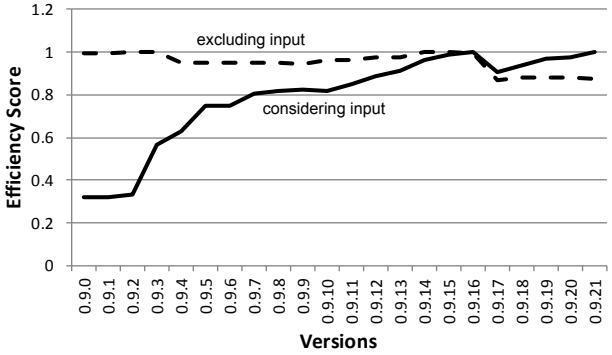

Fig. 5. Impact of excluding input on the DEA efficiency scores

Guideline 5. Inputs can be neglected from the application of DEA by setting a constant value to the input variables. However the exclusion of inputs might have a significant impact on the obtained evolutionary trends.

6 Conclusions

Data Envelopment Analysis offers a promising alternative for the assessment of software evolution by combining several metrics that serve as quality indicators and by normalizing the obtained scores over the size characteristics of each examined version. In this paper the impact of several alternatives in the application of DEA is examined resulting to a set of practical guidelines.

Acknowledgements. This work has been partially funded by the Research Committee of the University of Macedonia, Greece.

7 References

1. Andersen, P., Petersen, N.C. (1993). A procedure for ranking efficient units in Data Envelopment Analysis. *Management Science*. 39, pp. 1261-1264.
2. Charnes, A., Cooper, W.W., Rhodes, E. (1978) Measuring the efficiency of decision making units. *European Journal of Operational Research*. 2 (6), pp. 429-444.
3. Chatzigeorgiou, A. and Stiakakis, E. (2011) Benchmarking library and application software with Data Envelopment Analysis. *Software Quality Journal*. 19(3), pp. 553-578.
4. Chatzigeorgiou, A. and Stiakakis, E. (2012) Combining Metrics for Software Evolution Assessment by Means of Data Envelopment Analysis. *Journal of Software: Evolution and Process*. published online.
5. Lee, Y., Yang, J., Chang, KH. (2007) Metrics and evolution in open source software. *7th International Conference on Quality Software* (QSIC'2007). Portland, Oregon, 191-197
6. Lehman, M. M and Ramil, J. F. (2001). Rules and tools for software evolution planning and management. *Annals of Software Engineering*. 11(1), pp.15-44.
7. Samoladas, I., Gousios, G., Spinellis, D., Stamelos, I. (2008) The SQO-OSS quality model: Measurement based open source software evaluation. *Proc. of 4th International Conference on Open Source Systems* (OSS'2008). Milan, Italy, 237-248.