Knowledge Based Evaluation of Software Systems:

a Case Study[1]

Ioannis Stamelos[+], Ioannis Vlahavas[+], Ioannis Refanidis[+] and Alexis Tsoukiàs[++]

[+] Dept of Informatics, Aristotle University of Thessaloniki, 54006, Thessaloniki, GREECE

[vlahavas,stamelos,yrefanid]@csd.auth.gr

[++] LAMSADE-CNRS, Universite Paris-IX, Dauphine, 75775, Paris Cedex 16, FRANCE

tsoukias@lamsade.dauphine.fr

Abstract

Solving software evaluation problems is a particularly difficult software engineering process and many contradictory criteria must be considered to reach a decision. Nowadays, the way that decision support techniques are applied suffers from a number of severe problems, such as naive interpretation of sophisticated methods and generation of counter-intuitive, and therefore most probably erroneous, results. In this paper we identify some common flaws in decision support for software evaluations. Subsequently, we discuss an integrated solution through which significant improvement may be achieved, based on the Multiple Criteria Decision Aid methodology and the exploitation of packaged software evaluation expertise in the form of an intelligent system. Both common mistakes and the way they are overcomed are explained through a real world example.

Keywords: automated software evaluation, intelligent systems, multiple criteria decision aid

---

1. Introduction

The demand for qualitative and reliable software, compliant to international standards and easy to integrate in existing system structures is increasing continuously. On the other hand, the cost of software production and software maintenance is raising dramatically as a consequence of the increasing complexity of software systems and the need for better designed and user friendly programs. Therefore, the evaluation of such software aspects is of paramount importance. We will use the term *software evaluation* throughout this paper to denote evaluation of various aspects of software systems.

Probably the most typical problem in software evaluation is the selection of one among many software products for the accomplishment of a specific task. However, many other problems may arise, such as the decision whether to develop a new product or acquire an existing commercial one with similar requirements. On the other hand, software evaluation may have different points of view and may concern various parts of the software itself, its production process and its maintenance. Consequently, software evaluation is not a simple technical activity, aiming to define an "objectively good software product", but a decision process where subjectivity and uncertainty are present without any possibility of arbitrary reduction.

In the last years, research has focused on specific software characteristics, such as models and methods for the evaluation of the quality of software products and software production process [2, 6, 7, 9, 11, 12, 23]. The need for systematic software evaluation throughout the software life cycle has been recognised and well defined procedures have already been proposed [22] while there are on-going standardisation activities in this field [7].

However, although the research community has sought and proposed solutions to specific software evaluation problems, the situation in this field, from the point of view of decision support, is far from being satisfactory. Various approaches have been proposed for the implementation of decision support for specific software evaluation problems. Typically, the evaluator is advised to define a number of attributes, suitable for the evaluation problem at hand. In certain cases [7, 14], a set of attributes is already proposed. The evaluator, after defining the alternative choices, has to assign values and, sometimes, relative weights to the attributes and ultimately combine (*aggregate*) them to produce the final result. The problem is that, in practice, this is frequently accomplished in a wrong way, as it will be exemplified in this paper.

The authors have participated extensively in software system evaluations, either providing decision aid to the evaluators or as evaluators themselves. After reviewing the relative literature, it seems that the most commonly used method is the *Weighted Average Sum* (*WAS*), known also as *Linear Weighted Attribute* (*LWA*), or similar techniques, such as *Weighted Sum* (*WS*) (see [11]).

Another frequently used method is *Analytical Hierarchy Process* (AHP, see [17,23]) where priorities are derived from the eigenvalues of the pairwise comparison matrix of a set of elements when expressed on ratio scales. WAS and AHP fall into a class of techniques known under the name *Multiple-Criteria Decision Aid* (*MCDA*, [16, 20]). This methodology is applied in the evaluation problems where the final decision depends on many, often contradictory, criteria.

This paper identifies some common flaws in decision support for software evaluation. Our approach consists in exploiting packaged evaluation expertise in the form of an intelligent system, in order to avoid common mistakes and to speed up the evaluation process. We have developed ESSE, an Expert System for Software Evaluation, which supports various methods of the MCDA methodology and we present its use in a real example, the evaluation of five different proposals for the information system modernisation in a large transport company.

The rest of the paper is organised as follows: Section 2 introduces decision support for software evaluation and discusses some common errors during the implementation of a decision process for software evaluation. Section 3 highlights the manifestation of these errors in a practical example, while section 4 presents briefly ESSE. Section 5 shows the generation of a correct evaluation model using this tool. Finally, section 6 concludes the paper and poses future directions

2. Decision Making for Software System Evaluations

Given the importance of software, it is necessary to evaluate software products and related processes in a systematic way. Various criteria must be analysed and assessed to reach the final decision. As already mentioned, software evaluation problems fall into the class of decision making problems that are handled through the MCDA methodology. In order to render this paper self-contained, the basic steps of a systematic MCDA decision process are described in section 2.1. Finally, section 2.2 discusses the current situation in software industry regarding decision making and identifies a list of typical wrong steps, taken frequently in nowadays decision making for software evaluation.

2.1. Steps of a Typical Decision Process

An evaluation problem solved by MCDA can be modeled as a 7-ple $\{A,T,D,M,E,G,R\}$ where [20]:

- A is the set of alternatives under evaluation in the model

- T is the type of the evaluation

- *D* is the tree of the evaluation attributes

- *M* is the set of associated measures

- *E* is the set of scales associated to the attributes

- *G* is the set of criteria constructed in order to represent the user's preferences

- *R* is the preference aggregation procedure

In order to solve an evaluation problem, a specific procedure must be followed [12]:

Step 1: *Definition of the evaluation set A:* The first step is to define exactly the set of possible choices. Usually there is a set *A* of alternatives to be evaluated and the best must be selected. The definition of *A* could be thought as first-level evaluation, because if some alternatives do not fulfill certain requirements, they may be rejected from this set.

Step 2: *Definition of the type T of the evaluation*: In this step we must define the type of the desired result. Some possible choices are the following:

- choice: partition the set of possible choices into a sub-set of best choices and a sub-set of not best ones.

- classification: partition the set of possible choices into a number of sub-sets, each one having a characterization such as good, bad, etc.

- sorting: rank the set of possible choices from the best choice to the worst one.

- description: provide a formal description of each choice, without any ranking.

Step 3: *Definition of the tree of evaluation attributes D*: In this step the attributes that will be taken into account during the evaluation and their hierarchy must be defined. Attributes that can be analyzed in sub-attributes are called compound attributes. Sub-attributes can also consist of sub-sub-attributes and so on. The attributes that can not be divided further are called *basic attributes*. An example of such an attribute hierarchy is shown in figure 1.

It should be noted that there exist mandatory independence conditions, such as the separability condition, and contingent independence conditions, depending on the aggregation procedure adopted (see [18]).
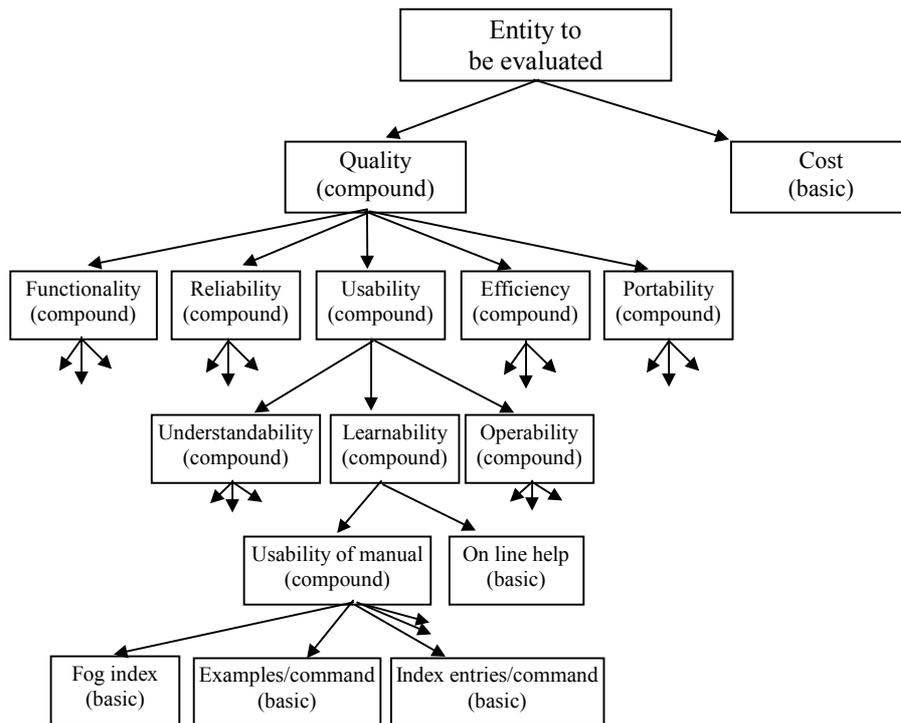
Figure 1: *Example of an attribute hierarchy*

Step 4: *Definition of the set of measurement methods M*: For every basic attribute d we must define a method M_d that will be used to assign values to it. There are two kinds of values, the *arithmetic values* (ratio, interval or absolute) and the *nominal values*. The first type of values are numbers, while the second type are verbal characterizations, such as "red", "yellow", "good", "bad", "big", "small", etc.

A problem with the definition of M_d is that d may not be measurable, because of its measurement being non-practical or impossible. In such cases an arbitrary value may be given, based upon expert judgment, introducing a subjectivity factor. Alternatively, d may be decomposed into a set of sub-attributes d_1, d_2, ... d_n, which are measurable. In this case the expression of arbitrary judgment is avoided, but subjectivity is involved in the decomposition.

Step 5: *Definition of the set of measurement scales E*: A scale e_d must be associated to every basic attribute d. For arithmetic attributes, the scale usually corresponds to the scale of the metric used, while for nominal attributes, e_d must be declared by the evaluator. Scales must be at least ordinal, implying that, within e_d, it must be clear which of any two values is the most preferred (in some cases there are different values with the same preference). For example, for d = 'operating system', e_d could

be [UNIX, Windows NT, Windows-95, DOS, VMS] and a possible preference could be [UNIX = Windows NT > Windows-95 = VMS > DOS].

Step 6: *Definition of the set of Preference Structure Rules G*: For each attribute and for the measures attached to it, a rule or a set of rules have to be defined, with the ability to transform measures to preference structures. A preference structure compares two distinct alternatives (e.g. two software products), on the basis of a specific attribute. Basic preferences can be combined, using some aggregation method, to produce a global preference structure.

For example, let a_1 and a_2 be two alternatives and let d be a basic attribute. Let also $m_d(a_1)$ be the value of a_1 concerning d and let $m_d(a_2)$ be the value of a_2 concerning d. Suppose that d is measurable and of positive integer type. In such a case, a preference structure rule could be the following:

- *product a_1 is better than a_2 on the basis of d, if $m_d(a_1)$ is greater than $m_d(a_2)$ plus K, where K is a positive integer*

- *products a_1 and a_2 are equal on the basis of d, if the absolute difference between $m_d(a_1)$ and $m_d(a_2)$ is equal or less than K, where K is a positive integer*

Step 7: *Selection of the appropriate aggregation method R*: An aggregation method is an algorithm, capable of transforming the set of preference relations into a *prescription* for the evaluator. A prescription is usually an order on *A*.

The MCDA methodology consists of a set of different aggregation methods, which fall into three classes. These are the *multiple attribute utility methods* [8], the *outranking methods* [20] and the *interactive methods* [19] (although originated by different methodological concerns we may consider AHP as a multi-attribute utility method since priorities computed by this procedure define a value function). The selection of an aggregation method depends on the following parameters [20]:

- The type of the problem

- The type of the set of possible choices (continuous or discrete)

- The type of measurement scales

- The kind of importance parameters (weights) associated to the attributes

- The type of dependency among the attributes (i.e. *isolability*, *preferential independence*)

- The kind of uncertainty present (if any)

Notice that the execution of the steps mentioned above is not straightforward. For example, it is

allowed to define first *D* and then, or in parallel, define *A,* or even select *R* in the middle of the process.

2.2. Current Practice and Common Problems in Decision Making for Software Evaluation

Although the need for a sound methodology is widely recognised, the evaluators generally avoid the use of MCDA methods. In [5] it was observed that only two MCDA methods were used by the decision makers, while [10] reported that even unsophisticated MCDA tools were avoided. Nevertheless, a lot of research has been dedicated to the problem of selecting the best method for software evaluation. During controlled experiments, it was found that, generally, different methods gave similar results when applied to the same problem by the researchers themselves. During a study by [11] it was empirically assessed that the above statement was true for evaluations made by independent decision makers as well. Yet, the degree of the application difficulty differs between the MCDA techniques and, in certain cases, a particular MCDA method is recommended. As an example, in [18] the use of the *Multiple Attribute Utility Technique* (*MAUT*) is suggested, while in [23] the AHP method has been adopted.

In many cases no systematic decision making method is used at all. Evaluations are based purely on expert subjective judgement. While this approach may produce reliable results, it is more difficult to apply (because experts are not easily found, formal procedures are hard to define) and has a number of pitfalls, like:

− inability to understand completely and reproduce the evaluation results,
− poor explanation of the decision process and the associated reasoning,
− important problem details for the evaluation may be missed,
− high probability that different experts will produce different results without the ability to decide which one is correct,
− difficulty in exploiting past evaluations,
− risk to produce meaningless results.

The number of top-level attributes used in the evaluation model plays a significant role in software evaluation problems. It may seem reasonable to presume that the higher the number of attributes used, the better the evaluation will be. However, our experience in applying MCDA has demonstrated that at most seven or eight top-level attributes must be employed. The reason is that this number of attributes is sufficient to express the most important dimensions of the problem. The evaluator must focus on a relatively limited set of attributes and must avoid wasting time in low importance analysis. If necessary, broader attributes may be defined, incorporating excessive lower importance attributes. The latter may be accommodated in lower levels of the hierarchy tree of figure 1. Another weak choice is to include in the model attributes for which the alternative solutions in the choice set *A* can not be differentiated. The result is the waste of evaluation resources without adding

any value to the decision process. Finally, custom evaluation attribute sets (e.g. [14]) are rarely used in practice, meaning that packaged evaluation experience is not adequately exploited.

Another point concerning attributes is redundancy. Hasty definition and poor understanding of the evaluation attributes may lead to this common error. As an example, consider the quality attribute hierarchy described in [7], part of which is shown in figure 1. Suppose that the evaluator decides to utilise this schema in a software evaluation problem, in which, probably along with other attributes, he is considering *Cost* and *Quality*. Some of the quality sub-attributes are related to the attribute *Cost*. These are *Usability*, *Maintainability*, and partially, *Portability*, which are defined in [7] in terms of "...effort to understand, learn, operate, analyse, modify, adapt, install, etc.". If *Cost* is defined to comprise maintenance, installation, training, etc. costs, then the combined cost and quality attribute structure will force the evaluator to consider twice the above mentioned cost components. Although less critical, redundancy may also manifest in the form of metrics that are strongly related, e.g. "elapsed time from user query to initiation of system response" and "elapsed time from user query until completion of system response".

Another common source of errors is the use of arbitrary judgements instead of structural measurements. In general, arithmetic scales should be preferred to ordinal scales whenever possible, because of the subjectivity involved with the latter. On the other hand, there is already a large volume of literature for the use of structural metrics, such as the complexity and size metrics referred above, to compute or predict various software attributes. Although there is a lot of debate about their practical usefulness and measurements may require dedicated tools, these metrics should be assessed and used in software evaluations. However, in practice, most evaluators prefer the use of ordinal values, relying purely on subjective assessments and avoiding the overhead of software measurements. This may not necessarily lead to wrong results, provided that appropriate ordinal aggregation procedures are used. However, as mentioned above, software engineers are rarely aware of such techniques. The outcome is that the evaluation results are produced in subtle ways and the decision makers and their management are rarely convinced about their correctness.

The most critical point in evaluation problems is the selection of the appropriate aggregation method, according to the specific problem. In practice, it seems that the most widely known method in the software community is the Weighted Sum, and consequently, whenever a systematic approach is taken, Weighted Sum or Weighted Average Sum is the most frequent choice. This is probably caused by the fact that Weighted Sum has been also used in various software engineering disciplines, such as Function Point Analysis [1].

As mentioned in step 7 of *2.1*, the selection of a suitable aggregation method depends on a number of parameters, such as the type of attributes and the type of the measurement methods. In practice, frequently an inappropriate aggregation method is chosen and its application is subject to

errors. For example, WAS is applied with attributes for which, although initially an ordinal scale is assigned, an arithmetic scale is forced to produce values suitable for WAS application (*good* is made "equal" to *1*, *average* to *2*, …). As another example, an outranking method could be selected while trade-offs among the criteria are requested. In fact our criticism focuses on the unjustified choice of a procedure without verifying its applicability. For instance AHP requires that all comparisons be done in ratio scales. This is not always the case. If we apply such a procedure on ordinal information we obtain meaningless results. A more mathematical treatment of the above issues and a detailed analysis on aggregation problems and drawbacks can be found in [3] (see also [4, 13]).

Finally, another issue in the software evaluation process is the management of the decision process. Typically, the effort and resources needed for an evaluation are underestimated. A software evaluation is a difficult task, requesting significant human effort. Systematic evaluation necessitates thorough execution of the seven process steps referred in *2.1*, probably with computer support for the application of an MCDA method or the measurement of certain software attributes. Software evaluations are normally performed with limited staff, under a tight schedule and without clear understanding of the decision process and its implications. In general, managers seem to be mainly interested in the generation of a final result, i.e. they are willing to reach the end of the decision process, but they seem somewhat careless about the details, the data accuracy and the overall validity of the approach taken.

3. A Real World Example: a Weak Evaluation Model

Recently a large transport organisation was faced with a rather typical evaluation problem: the evaluation of certain proposals for the evolution of its information system. System specifications were prepared by a team of engineers, who defined also the evaluation model to be used (denoted by *M* in the following). A number of distinct offers were submitted by various vendors. A second team (the *evaluators*) was also formed within the organisation. They applied the imposed model, but the results they obtained were in contradiction with the common sense.

At that point the evaluators contacted the authors, presenting their problem. The authors realised that a number of errors, akin to the issues discussed in section *2.2*, were manifested in this model. The authors suggested to the evaluators the use of *ESSE* (Expert System for Software Evaluation, [21]), a system aiming to the automation of the software evaluation process. A new, quite satisfactory, evaluation model was generated with the assistance of ESSE. Section 3.1 gives a brief description of the information system, while section 3.2 presents the initial evaluation model and discusses the model problems. ESSE is described briefly in section 4 and the new model is presented in section 5.

3.1. Description of the Evaluation Problem

The new information system was intended to substitute the legacy information system of the acquirer organisation. The old system was of the mainframe type, but the new one should support the development of client server applications, while maintaining the existing functionality.

The main hardware components of the system were the central server machine, certain network components and a predefined number of client terminals and peripherals (various models of printers, scanners, tape streamers). In all cases there were various requests concerning the accompanying software, e.g. there were specific requests about the server's operating system, the communications software, etc. Additionally, some software components were also required: an RDBMS system, a document management system (including the necessary hardware), a specific package for electronics design and a number of programming language tools.

An additional requirement was the porting of the existing application software, that used to run on the legacy information system, to the new system. No development of new functionality was foreseen. Finally, certain typical requirements were also included, regarding training, maintenance, company profile (company experience, company size, company core business, ability to manage similar projects), etc.

Five offers (i.e. technical proposals, denoted by P_i, where $i = 1,2,3,4,5$) were submitted as a response to the request for tenders, consisting of a technical and a financial part. Only the technical part of the proposals was subject to the analysis examined in this paper. Of course, the final choice was decided taking into account both aspects.

3.2. The Initial Evaluation Model

The initial evaluation model consisted of 39 basic attributes, with various weights having been assigned to them. The proposed aggregation method was the *Weighted Sum* (*WS*). A full description of this model is shown in Appendix A.

Although, at a first glance, the attributes seem to be organised in a hierarchy (e.g. *NETWORK* seems to be composed of *Network design, Active elements, Requirements coverage*), they are actually structured in a flat manner, due to the requirement for WS. This means that each top level attribute (denoted by bold letters in table 1) has an inherent weight, which equals the sum of the weights of its sub-attributes. This is depicted in Table 1.

According to the model, an arithmetic value must be assigned to each attribute. Two types of arithmetic scales are permitted, ranging from 40 to 70 and 50 to 70 correspondingly, as shown in Appendix A. The weights were assigned in *M* with the intention to reflect the opinion of the people that assembled the model about the relative importance of the attributes. The problems related to *M* are the following:

Attributes	Description	Weight
1	**HARDWARE**	**64**
1.1	NETWORK	18
1.2	SERVER	24
1.3	WIRING - NETWORK INSTALLATION	6
1.4	CLIENT TERMINALS	7
1.5	PRINTERS	3
1.6	SENSOR DEVICES	1
1.7	DOC. MGMT SYSTEM (HW)	1
1.8	PERIPHERALS	4
2	**SOFTWARE**	**27**
2.1	OPERATING SYSTEM	4
2.2	PROGRAMMING TOOLS	18
2.3	COMMUNICATIONS SW	3
2.4	ELECTRONICS CAD	1
2.5	DOC. MGMT SYSTEM (SW)	1
3	**LEGACY SW PORTING**	**1**
4	**NETWORK MGMT SW**	**1**
5	**TRAINING**	**3**
6	**MAINTENANCE**	**1**
7	**COMPANY PROFILE**	**3**
8	**MISCELLANEOUS**	**3**

Table 1: Top level attribute inherent weights.

1. The attribute structure is not an optimal one. Although the number of attributes used is relatively high, there is no real hierarchical structure. In addition, there is no balancing in the attribute analysis: certain attributes are decomposed in lower ones, while others are not. A direct side effect of this characteristic of *M* is that the attributes that are composed of lower ones are given a relatively high weight (the sum of the lower attribute weights) compared to the basic attributes. This generates the risk of assigning low importance to an attribute, simply because it cannot be easily decomposed. Another weak point is the inclusion in *M* of attributes, for which, although a significant importance had been assigned, it was inherently hard to propose more than one technical solutions, such as *Programming Languages 1* and *2*, with weight *3*.

2. There is some redundancy between certain attributes. For example, it is difficult to think that the selection of the network active elements and the network wiring are completely independent.

3. Although the weights are supposed to reflect the personal opinion of the requirement engineers, they are evidently counter-intuitive and misleading. For example, consider the weight of *3*, assigned to *Company Profile* and compare to the summary weights of *Hardware* (*64*) and *Software* (*27*). *Company Profile* is an important factor, affecting the entire project. In this setting, *Company Profile* counts approximately as *1/21* of *Hardware* and *1/30* of *Hardware* plus *Software*, meaning that it is practically irrelevant. *Legacy Software Porting* is assigned a weight of *1*, implying that it

is *1/64* important in respect with *Hardware* and *1/91* with *Hardware* plus *Software*. However, failing to meet the requirements of this part of the project would cancel all benefits of the system modernisation. In general, attributes *3* to *8* in table 1, are severely penalised in respect with the first two.

4. The specified measurement method generates side effects, which distort the evaluation result. Consider for example the effect of the lower limit of the two arithmetic scales. Suppose that, P_i and P_k offer poor technical solutions for two attributes C_1 and C_2 respectively and that the evaluator decides to assign the lowest possible rating, while still keeping alive the two proposals. Suppose also that C_1 and C_2 have been given an equal weight, but scale 40-70 is allowed for C_1 and 50-70 is allowed for C_2. Therefore, P_i is given a *40* for C_1, while P_k is given a *50* for C_2. This means that, although the evaluator attempts to assign a virtual *zero* value to both proposals, the net result is that P_k gains ten evaluation points more than P_i. If the weight of the two attributes is *6*, this is translated to a difference of sixty points for P_k, a difference that could even result to the selection of this proposal, in case that the final scores of the proposals are too close to each other.

5. The selected measurement and aggregation methods are inappropriate. No specific metrics have been proposed by the authors of *M*, implying that subjective assessments are to be used. However, arithmetic values are expected for the attribute quantification and, as already discussed above, this leads to the problem of translating a nominal scale to an arithmetic one, in order to apply the imposed aggregation method, i.e. *Weighted Sum*.

Other similar cases, supporting the evidence that current practice in decision support for software is very error-prone have been observed by the authors.

4. The Expert System for Software Evaluation (ESSE)

ESSE (Expert System for Software Evaluation, [21]) is an intelligent system that supports various types of evaluations where software is involved. This is accomplished through the:

- automation of the evaluation process,
- suggestion of an evaluation model, according to the type of the problem,
- support of the selection of the appropriate MCDA method, depending on the available information,
- assistance provided by expert modules (the *Expert Assistants*), which help the evaluator in assigning values to the basic attributes of the software evaluation model,
- management of past evaluation results, in order to be exploited in new evaluation problems,
- consistency check of the evaluation model and detection of possible critical points.
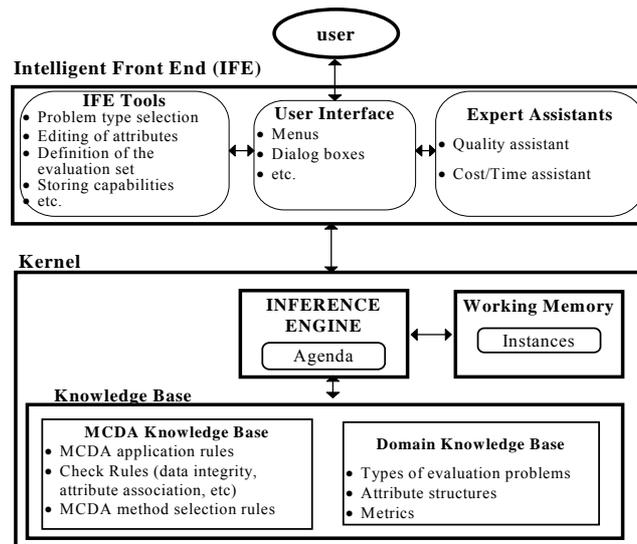
Figure 2: *The architecture of ESSE*

ESSE, is designed to support generic software evaluation problem solving. ESSE proposes default evaluation models based on certain fundamental software attributes (quality, cost, solution delivery time, compliance to standards), according to the evaluation problem. In addition, mechanisms are provided for flexible decision model construction, supporting the definition of new attributes and their structure. Furthermore, embedded knowledge provides support for attribute quantification.

The advantages of ESSE lie in the built-in knowledge about software evaluation problem solving and in the flexibility in problem modeling, all in an integrated environment with a common user interface. The evaluator is allowed to define his own problem attributes, along with their measurement definition, providing maximum flexibility in problem formulation and subsequent decision making, permitting better representation and resolution of both high and low level contradicting attributes.

Through these mechanisms, additional attributes may be easily introduced in the evaluation model and a well balanced evaluation result, with the optimum utilisation of the evaluation resources, may be obtained. Moreover, expert knowledge is provided, allowing the detection of critical points in the model, and, ultimately, the suggestion of the appropriate MCDA method and the quantification of software attributes. The architecture of ESSE is depicted in figure 2 and is described briefly in the following paragraphs.

ESSE consists of two main components, the *Intelligent Front End* (IFE) and the *Kernel*. The role of *IFE* is to guide the evaluator in the correct application of the MCDA methodology and to provide expertise in software attribute evaluation. IFE supports the collection of the necessary information and the validation of the model created. Validation is achieved both through tutorials (on line help) and through expert rules that identify model critical points.

IFE consists of three modules, namely *User Interface, IFE Tools* and *Expert Assistants*. The *User Interface* is responsible for the communication between the user and the other modules of ESSE. The *IFE Tools* are invoked by the User Interface to perform tasks related with the construction and validation of the evaluation model, such as selection of a problem type, definition and modification of attributes, definition of the evaluation set, verification of the evaluation model and various storing tasks. Finally, the *Expert Assistants* support the assignment of values to basic attributes. Human knowledge, related to software evaluation attributes, has been accumulated in the past. ESSE includes expert assistants for the estimation of software development cost and time and software quality.

The *Kernel* of ESSE consists of three parts: the *Inference Engine,* the *Working Memory* and the *Knowledge Base*. The Knowledge Base contains both domain knowledge and MCDA knowledge. Domain knowledge consists of structural knowledge in the form of *frames*, concerning the various types of evaluation problems, attribute definitions and corresponding metrics. MCDA knowledge consists of behavioral knowledge in the form of production and deductive rules. The rules can also be distinguished in MCDA application rules, evaluation model check rules and MCDA method selection rules.

A detailed description of ESSE is given in [21].

5. Using ESSE to improve the Evaluation Model

In this section we present the application of two evaluation models in the problem described in the third section. The first model is the initial model discussed in 3.2, while the second model is a new one obtained with the assistance of ESSE. The two models and the results they produced are presented in sections 5.1 and 5.2 correspondingly.

5.1. Application of the initial evaluation model

Initially (i.e. before consultation with the authors) the evaluators examined the technical contents of the five proposals. From this informal analysis it was quite evident that the second proposal (*P2*) was the best from the technical point of view. Subsequently, the evaluators assigned values to all basic attributes for the various alternatives and obtained the results of Table 2 (the entire model *M* is presented in Appendix A - the assigned values are not shown for simplicity). The results shown in Table 2 were against the initial informal judgement: proposal *P3* was ranked first, while *P2* was ranked second. Besides, the differences between the scores obtained for each proposal were quite small, giving the idea that the five proposals were very similar from the technical point of view. This may be true for the system software and client terminals, but not for other attributes. The results were not surprising given the criticism presented above against *M* and the expected distortion of the evaluators' opinion.

Proposal	Score
P3	6009
P2	6002
P1	5903
P4	5890
P5	5794

Evaluation Results:

P3 > P2 > P1 > P4 > P5

Table 2: Results of the application of the initial evaluation model *M*, using Weighted Sum

5.2. Generation and application of an improved evaluation model using ESSE

The problem we are dealing with is characterised by ESSE as "Information Systems Evaluation" and is a sub-class of the more generic class of the "Commercial Product Evaluation" problems. The knowledge base of ESSE already contained past evaluation problems of the same type. The evaluators merely asked ESSE to propose an attribute hierarchy, together with the corresponding weights and scales, for this type of problems. The model proposed by ESSE consisted of a hierarchical attribute structure, with 8 top level attributes, decomposed in a number of sub-attributes. Additionally, certain model items concerning application software were expanded with predefined quality attribute structures, which were refined versions of the quality scheme of [7].

The evaluators modified the proposed model, according to the special characteristics of the problem imposed by the initial model *M*. Essentially, they removed some attributes they considered irrelevant and they added some others, which they considered important.

For example, ESSE suggested that top-level attributes such as 'Maintenance', 'Training' and lower level attributes such as 'Database Management System', 'Printers' etc. should be further analysed. Moreover, ESSE proposed different weights for the attributes and suggested measurement scales for the basic ones. The evaluators accepted the proposed analysis and they made few modifications to the weights, according to their own preferences. Finally, in some cases the evaluators did not accept the attributes proposed by ESSE and they removed them.

To illustrate this more clearly, consider attribute 'Maintenance'. In the initial model *M* this attribute was a top-level attribute without further analysis. Its measurement scale was the range [40, 70]. ESSE proposed to analyse 'Maintenance', as it is shown in Table 3.

Attribute	Description	Weight	Scales
6	**MAINTENANCE**	**6**	
6.1	24 hours service	2	{yes, no}
6.2	spare part availability	2	12-36 months
6.3	average response time	3	6-48h
6.4	backup system	3	{yes, no}

Table 3: The suggestion of ESSE for attribute 'Maintenance'

The evaluators accepted the above analysis, except for the sub-attribute 'backup system', which they considered irrelevant and they removed it. Moreover, they accepted the proposed scales but they gave a weight of 4 to the sub-attribute '24 hours service', because they considered it more important.

The complete improved model (denoted by *M'*) is shown in Appendix B, while a top level view may be found in Table 4.

Index	Attributes	Weights
1	**NETWORK**	6
2	**SERVER**	6
3	**CLIENT TERMINALS AND PERIPHERALS**	4
4	**SYSTEM SOFTWARE**	6
5	**APPLICATION SOFTWARE**	6
6	**MAINTENANCE**	6
7	**TRAINING**	4
8	**COMPANY PROFILE & PROJECT MANAGEMENT**	5

Table 4: Top level view of the improved evaluation model M'

The aggregation method proposed by ESSE is of the ELECTRE II type [15], because of the presence of ordinal scales and of ordinal importance parameters. A brief presentation of the method is provided in Appendix C.

The evaluators assigned values to all basic attributes of *M'* for the various alternatives (not shown here for simplicity). According to the method, for each pair of alternatives (x,y) and for each attribute j, a relation S_j may hold, meaning that the alternative x is at least as good as the alternative y, with respect to the attribute j. Table 5 shows for each pair (x,y) of alternatives, which of the relations $S_j(x,y)$, $j=1..8$, hold for the various top level attributes. These relations have been extracted by setting the concordance threshold to the value $c=0.8$ and without having veto conditions.

$S_j(x,y)$ $\quad$ y $\quad$ x	P1	P2	P3	P4	P5
P1			5	1,2,3,4,5,6,7,8	3,7,8
P2	1,2,3,4,5,6,7,8		3,4,5,6,7,8	1,2,3,4,5,6,7,8	1,2,3,4,5,6,7,8
P3	1,2,3,4,5,6,7,8	1,2,3,8		1,2,3,4,5,6,7,8	1,2,3,4,7,8
P4	1,3,4,5,6,7,8		5		1,3,7,8
P5	1,2,3,4,5,6,7,8	6	4,5,6	1,2,3,4,5,6,7,8	

Table 5: The $S_j(x,y)$ relations for the top-level attributes of model M'

At each entry of Table 5, the numbers indicate the top level attributes for which a relation $S_j(x,y)$ holds. For example, the entry in row P5 and column P3 with values 4, 5 and 6 indicates that the relations $S_4(P5,P3)$, $S_5(P5,P3)$ and $S_6(P5,P3)$ hold, i.e. P5 is at least as good as P3 with respect to the attributes 4 ('System Software'), 5 ('Application Software') and 6 ('Maintenance').

Having computed the $S_j(x,y)$ relations for the top level attributes and taking into account their weights, we can compute general S relations with respect to all the top level attributes simultaneously. These are the following:

$S(P1, P4)$

$S(P2, P1)$, $S(P2, P3)$, $S(P2, P4)$, $S(P2, P5)$

$S(P3, P1)$, $S(P3, P4)$, $S(P3, P5)$

$S(P4, P1)$

$S(P5, P1)$, $S(P5, P4)$

From these relations and by using an exploitation procedure we can extract an ordering on the set of the alternatives. In our case, we can assign to each alternative a score, according to the formula: $\sigma(x) = |\{ y: S(x,y)\}| - |\{ y: S(y,x)\}|$, $x \in A$. Taking into account the above relations, the following scores are computed:

$$\sigma(P1) = -3, \sigma(P2) = 4, \sigma(P3) = 2, \sigma(P4) = -3, \sigma(P5) = -1$$

leading to the final ordering:

$$P2 > P3 > P5 > P1 = P4$$

This ordering was in accordance with the evaluators' intuition, ranking proposal $P2$ first and proposal $P3$ second. We also observe that there are differences in the lower positions of the ordering in respect with the results obtained by the initial model M, while there is also indifference between $P1$ and $P4$.

6. Conclusions - Future Research

In this paper we have discussed common errors made during the evaluation of software systems. These errors are caused because of insufficient understanding of certain fundamental principles of decision support methodologies and wrong practical application of evaluation techniques in the software industry. The manifestation of these problems has been exemplified through a real world example: a tender evaluation concerning the acquisition of a new information system for a large organisation. The pitfalls of a specific evaluation model have been pinpointed and it has been shown how an improved model has been generated with the use of an intelligent system, which exploits successfully packaged knowledge about software problem solving.

We plan to examine more evaluation situations in order both to enhance the knowledge bases of ESSE and to obtain more feedback on the benefits of the adoption of a knowledge based solution combined with a sound decision support methodology.

Moreover, we plan to study and implement evaluation knowledge maintenance within ESSE. Currently, the system proposes an evaluation model obtained by a specific past evaluation problem or defined by an expert. Our intention is to derive an evaluation model from various past evaluation problems of the same type, in order to exploit the expansion of the knowledge base of the system.

Finally, more MCDA methods will be implemented within ESSE, along with the necessary knowledge for their correct application.

7. References

[1] Albrecht A.J. and Gaffney J.E., Software function, source lines of code, and development effort prediction: a software science validation, IEEE Trans. 6 (1983) 639-648.
[2] Basili V.R., Applying the GQM paradigm in the experience factory, in N. Fenton, R. Whitty and Y. Iizuka ed., Software Quality Assurance and Measurement, (Thomson Computer Press, London, 1995) 23-37.
[3] M-J. Blin and A. Tsoukiàs, Multi-criteria Methodology Contribution to the Software Quality Evaluation, Cahier du LAMSADE N. 155, LAMSADE, Université Paris Dauphine (1998).
[4] M-J. Blin and A. Tsoukiàs, Evaluation of COTS using multi-criteria methodology, proc. of the 6th European Conference on Software Quality (1999), 429 - 438.
[5] Evans G.W, An Overview of Techniques for Solving Multi-objective Mathematical Problems, Management Science 11 (30) (1984) 1268-1282.
[6] Fenton N. and Pfleeger S-L, Software metrics - A Rigorous and Practical Approach, 2nd edition, International Thomson Computing Press (1996).
[7] ISO/IEC 9126-1, Information Technology - Software quality characteristics and sub-characteristics (1996).
[8] Keeney R.L. and Raiffa H., Decision with multiple objectives, John Wiley, New York (1976).

[9]   Kitchenham B., Towards a constructive quality model. Part 1: Software quality modelling, measurement and prediction, Software Engineering Journal 2 no. 4 (1987) 105-113.

[10]   Kotteman J.E. and Davis F.D., Decisional Conflict and User Acceptance of Multicriteria Decision-Making Aids, Decision Sciences 22 no 4 (1991) 918-926.

[11]   Le Blank L. and Jelassi T., An empirical assessment of choice models for software selection: a comparison of the LWA and MAUT techniques, Revue des systemes de decision, vol.3 no. 2 (1994) 115-126.

[12]   Morisio M. and Tsoukiàs A., IusWare, A methodology for the evaluation and selection of software products, IEEE Proceedings on Software Engineering 144 (1997) 162-174.

[13]   Paschetta E. and Tsoukiàs A., A real world MCDA application: evaluating software, Document du LAMSADE N. 113, LAMSADE, Université Paris Dauphine, 1999.

[14]   Poston R.M. and Sexton M.P. Evaluating and Selecting Testing Tools, IEEE Software 9, 3 (1992) 33-42.

[15]   Roy B., The outranking approach and the foundation of ELECTRE methods, Theory and Decision, 31 (1991), 49-73

[16]   Roy B., Multicriteria Methodology for Decision Aiding, Kluwer Academic, Dordrecht (1996).

[17]   Saaty T., The Analytic Hierarchy Process, Mc Graw Hill, New York (1980).

[18]   Shoval P. and Lugasi Y., 1987 Models for Computer System Evaluation and Selection, Information and Management 12 no 3, (1987) 117-129.

[19]   Vanderpooten D. and Vincke P., Description and analysis of some representative interactive multicriteria procedures, Mathematical and computer modelling, no.12, (1989).

[20]   Vincke P., Multicriteria decision aid, John Wiley, New York (1992).

[21]   Vlahavas I., Stamelos I., Refanidis I., and Tsoukias A., ESSE: An Expert System for Software Evaluation, Knowledge Based Systems, Elsevier, 4 (12) (1999) 183-197.

[22]   Welzel D., Hausen H.L and Boegh J., Metric-Based Software Evaluation Method, Software Testing, Verification & Reliability, vol. 3, no.3/4 (1993) 181-194.

[23]   Zahedi F., A method for quantitative evaluation of expert systems, European Journal of Operational Research, vol. 48, (1990), 136 - 147.

Appendix A

Initial Evaluation Model *M*: List of attributes and associated weights and scales. The aggregation method is Weighted Sum.

Attrib.	Description	W	Scale
1	**HARDWARE**		
1.1	NETWORK		
1.1.1	Design	6	50 - 70
1.1.2	Active elements	6	50 - 70
1.1.3	Requirements coverage	6	50 - 70
1.2	SERVER		
1.2.1	Central Unit	6	50 - 70
1.2.2	Disk Storage	6	50 - 70
1.2.3	Other magnetic storage	6	50 - 70
1.2.4	Requirements coverage	6	50 - 70
1.3	WIRING - NETWORK INSTALLATION		
1.3.1	Design	3	50 - 70
1.3.2	Requirements coverage	3	50 - 70
1.4	CLIENT TERMINALS		
1.4.1	Low capability model	3	40 - 70
1.4.2	Average model	3	40 - 70
1.4.3	Enhanced model	1[*]	40 - 70
1.5	PRINTERS		
1.5.1	Low capability model	1	40 - 70
1.5.2	Average model	1	40 - 70
1.5.3	Enhanced model	1	40 - 70
1.6	SENSOR DEVICES	1	40 - 70
1.7	DOC. MGMT SYSTEM (HW)	1	40 - 70

Attributes	Description	Weight	Scale
1.8	PERIPHERALS		
1.8.1	Scanners	1	40 - 70
1.8.2	Auxilliary Memory Devices	1	40 - 70
1.8.3	UPS	1	40 - 70
1.8.4	Tape Streamers	1	40 - 70
2	**SOFTWARE**		
2.1	OPERATING SYSTEM	4	40 - 70
2.2	PROGRAMMING TOOLS		
2.2.1	Progr. Language 1	3	50 - 70
2.2.2	Progr. Language 2	3	50 - 70
2.2.3	Data Base		
2.2.3.1	Data Base Mgmt System	3	50 - 70
2.2.3.2	Development Tools	3	50 - 70
2.2.3.3	Auxilliary Tools	3	50 - 70
2.2.3.4	Installed Base/Experience	3	50 - 70
2.3	COMMUNICATIONS SW	3	40 - 70
2.4	ELECTRONICS CAD	1	40 - 70
2.5	DOC. MGMT SYSTEM (SW)	1	40 - 70
3	**LEGACY SW PORTING**	1	40 - 70
4	**NETWORK MGMT SW**	1	40 - 70
5	**TRAINING**	3	50 - 70
6	**MAINTENANCE**	1	40 - 70
7	**COMPANY PROFILE**	3	50 - 70
8	**MISCELLANEOUS**		
8.1	System Delivery Schedule	1	40 - 70
8.2	Project Plan	1	40 - 70
8.3	Solution Homogeneity	1	40 - 70

---

[*] Weight 1 is due to the fact that only few terminals of this type were about to be purchased.

Appendix B

Improved Evaluation Model, *M'*, generated with the support of ESSE: List of attributes and associated weights and scales. The aggregation method is Electre II.

Attrib	Description	Weight	Scales
1	**NETWORK**	**6**	
1.1	Network design	2	
1.1.1	MAC Protocol	1	{Ethernet, Token Bus, TokenRing}
1.1.2	Cable type	1	{Fiber optic, Broadband Coaxial Cable, Base band coaxial cable, Twistaid pair}
1.1.3	Wiring requirements coverage	1	{high, average, low}
1.2	Active elements	2	
1.2.1	Repeaters	1	{yes, no}
1.2.2	Bridges	1	{yes, no}
1.2.3	Gateways	1	{yes, no}
1.3	Protocols	2	
1.3.1	TCP/IP	1	{yes, no}
1.3.2	IPX	1	{yes, no}
1.3.3	NETBEUI	1	{yes, no}
1.4	Network Management Software	3	{Novell, MS-Windows, VINES} -
2	**SERVER**	**6**	-
2.1	CPU	2	-
2.1.1	Number of CPUs	1	1-4
2.1.2	CPU type	1	{Pentium,Alpha, UltraSparc, ...}
2.1.3	CPU frequency	1	200-500 MHz
2.2	Memory	2	32-256 MB
2.3	Total Disk Capacity	1	10-50 GB
2.4	Requirements coverage	2	{high, average, low}
3	**CLIENT TERMINALS AND PERIPHERALS**	**4**	
3.1	Low capability PC model	3	
3.1.1	CPU clock	2	80-133 MHz
3.12	Memory	3	8-32 MB
3.1.3	Total Disk Capacity	2	1-3 GB
3.1.4	CD ROM speed	1	8x-24x
3.1.5	Monitor	1	-
3.1.5.1	Diameter	2	14-15"
3.15.2	Radiation	1	{low, average, high}
3.1.5.3	Overall quality	2	{high, average, small}
3.1.6	VGA card	1	-
3.1.6.1	card memory	1	512-2048 KB
3.1.6.2	graphics acceleration	1	{high, average, small}
3.2	Average PC mocel	4	
3.2.1	CPU clock	2	80-133 MHz
3.22	Memory	3	8-32 MB
3.2.3	Total Disk Capacity	2	1-3 GB
3.2.4	CD ROM speed	1	8x-24x
3.2.5	Monitor	1	-
3.2.5.1	Diameter	2	14-15"
3.25.2	Radiation	1	{low, average, high}
3.2.5.3	Overall quality	2	{high, average, small}
3.2.6	VGA card	1	-
3.2.6.1	card memory	1	512-2048 KB
3.2.6.2	graphics acceleration	1	{high, average, small}
3.3	Enhanced PC model	5	
3.3.1	CPU clock	2	133-266 MHz
3.3.2	Memory	3	16-32 MB
3.3.3	Total Disk Capacity	2	1-3 GB
3.3.4	CD ROM speed	1	8x-24x
3.3.5	Monitor	1	-
3.3.5.1	Diameter	2	14-15"
3.35.2	Radiation	1	{low, average, high}
3.3.5.3	Overall quality	2	{high, average, small}
3.3.6	VGA card	1	-
3.3.6.1	card memory	1	512-2048 KB
3.3.6.2	graphics acceleration	1	{high, average, small}
3.4	Low capability printer (dot matrix)	2	-
3.4.1	type	2	{line, character}
3.4.2	pins	1	9-24
3.4.3	speed	1	1-10 p/s
3.5	Average printer model	3	-
3.5.1	type	3	{laser,inkjet}
3.5.2	resolution	2	300-600 dpi

Attrib	Description	Weight	Scales
3.5.3	speed	2	4-12p/s
3.5.4	color	1	{yes, no}
3.6	Enhanced printer model	4	-
3.6.1	type	3	{laser,inkjet}
3.6.2	resolution	2	600-1200 dpi
3.6.3	speed	2	8-16p/s
3.6.4	color	1	{yes, no}
3.7	Scanner	2	
3.7.1	type	1	{desktop, hand}
3.7.2	resolution	1	300-1200 dpi
3.7.3	colors	1	{24b, 16b, 8b, B/W}
3.8	Jukebox CD-ROM device	2	-
3.8.1	Number of drives	2	6-64
3.8.2	Speed	1	8x-24x
3.9	Tape streamers	1	-
3.9.1	Capacity	2	500-4000 KB
3.9.2	Write speed	1	10-50 MB/min
3.9.3	Read spead	1	10-50 MB/min
3.10	UPS	2	
3.10.1	power	1	1000-5000 VA
4	**SYSTEM SOFTWARE**	**6**	
4.1	OPERATING SYSTEM	6	{NT, Novell, Unix,...}
4.2	C compiler	3	
4.2.1	interface	1	{graphics, text}
4.2.2	Visual programming	1	{yes, no}
4.2.3	interface with other languages	1	{yes, no}
4.2.4	intergrated environment	2	{yes, no}
4.3	COBOL compiler	2	
4.3.1	interface	1	{graphics, text}
4.3.2	interface with other languages	1	{yes, no}
4.3.3	intergrated enveronment	2	{yes, no}
5	**APPLICATION SOFTWARE**	**6**	
5.1	DBMS	3	
5.1.1	platform	2	{Oracle, Ingress, Access, SYBASE}
5.1.2	functionality	3	{high, average, low}
5.1.3	reliability	3	{high, average, low}
5.1.4	usability	3	{high, average, low}
5.1.5	efficiency	2	{high, average, low}
5.1.6	portability	1	{high, average, low}
5.2	DBMS Development Tools	2	
5.2.1	Schema editor	1	{yes, no}
5.2.2	ER diagram editor	2	{yes, no}
5.2.3	Case tools	1	{yes, no}
5.3	Auxiliary Tools	2	
5.3.1	wordProcessor	2	{MS-Word, WordPerfect...}
5.3.2	spreadsheet	2	{MS-Excel, Lotus123}
5.3.3	image processing	1	{Photoshop, CorelPaint,
5.3.4	intergration between tools	3	{high, average, low}
5.4	maturity/Experience	1	{high, average, low}
5.6	Doc. Mgmt system (Sw)	2	
5.6.1	security function	1	{yes, no}
5.6.2	electronic mail	1	{yes, no}
5.6.3	OCR success rate	1	85-100
5.7	Doc. Mgmt system (Hw)	2	
5.7.1	rewritable optical disks	1	{yes, no}
5.7.2	storage expandability rate	1	0-100
5.7.3	hardware data compression	1	{yes, no}
5.8	Legacy Software Porting	1	
5.8.1	experience	1	{high, average, low}
5.8.2	project plan	1	{high, average, low}
6	**MAINTENANCE**	**6**	
6.1	24h service	4	{yes, no}
6.2	spare part availability	2	12-36 months
6.3	average response time	3	6-48h
7	**TRAINING**	**4**	
7.1	manmonths	3	10-100
7.2	quality	2	{high, average. low}
7.3	experience	1	{high, average, low}
7.4	certification	1	{yes, no}
8	**COMPANY PROFILE & PROJECT MANAGEMENT**	**5**	
8.1	Company Profile	2	{high, average, low}
8.2	Delivery Time	3	3-9 months

Appendix C: The ELECTRE II type method applied to the experiment

ELECTRE II [15] provides a complete or a partial ordering of equivalence classes from the best ones to the worst ones. It considers ties and incomparable classes. Equivalent classes are composed of alternatives characterised by criteria. ELECTRE II calculates an ordering relation on all possible pairs built on the alternative set and constructs a preference relation on this set.

More precisely consider a set of alternatives A and a set of criteria G. We denote by $g_j(x)$ the value of alternative $x \in A$ under the criterion $j \in G$. Further on consider a binary relation S (to be read: "*at least as good as*"). Then for any pair of alternatives (x, y), we have:

$$S(x,y) \Leftrightarrow C(x,y) \wedge \neg D(x,y)$$

where:

$S(x,y)$: the alternative x is at least as good as y (x outranks y);

$C(x,y)$: concordance condition (in our specific case):

$$C(x, y) \Leftrightarrow \frac{\sum_{j \in J_{xy}^{\geq}} w_j}{\sum_{j \in J} w_j} \geq c \ \text{ and } \ \frac{\sum_{j \in J_{xy}^{\geq}} w_j}{\sum_{j \in J_{xy}^{<}} w_j} \geq 1 \qquad \left\{ \begin{array}{l} w_j: \text{ importance parameter of criterion j} \\ J_{xy}^{\geq} : \text{ criteria for which } S_j(x,y) \text{ holds} \\ J_{xy}^{>} : \text{ criteria for which } \neg S_j(y,x) \text{ holds} \\ J_{xy}^{<} : \text{ criteria for which } \neg S_j(x,y) \text{ holds} \end{array} \right.$$

$$D(x,y) \Leftrightarrow \exists\, g_j : v_j(x,y) \qquad \left\{ \begin{array}{l} v_j(x,y) \Leftrightarrow g_j(y) > g_j(x) + v_j, \text{ or, } v_j(x,y) \Leftrightarrow \dfrac{g_j(y) - g_j(x)}{t_j - b_j} \geq v_j \\[2mm] \text{with: } v_j(x,y): \text{ veto condition on criterion } g_j \,;\ v_j: \text{ veto threshold on } \\ \text{criterion } g_j \,;\ t_j: \text{ max value of criterion } g_j \,;\ b_j: \text{ min value of criterion } g_j; \end{array} \right.$$

The conditions under which $S_j(x,y)$ holds depend on the preference structure of each criterion. If g_j is:

- a weak order: $S_j(x,y) \Leftrightarrow g_j(x) \geq g_j(y)$,

- a semi order: $S_j(x,y) \Leftrightarrow g_j(x) \geq g_j(y) - k_j$,

- and so on with interval orders, pseudo orders, etc.

Once the global outranking relation is obtained, we can deduce:

- a strict preference relation $p(x,y)$, between x and y:

$p(x,y) \Leftrightarrow s(x,y) \land \neg s(y,x)$;

- an indifference relation $i(x,y)$, between x and y:

$i(x,y) \Leftrightarrow s(x,y) \land s(y,x)$;

- an incomparability relation $r(x,y)$ between x and y:

$r(x,y) \Leftrightarrow \neg s(x,y) \land \neg s(y,x)$;

The definition of the relation $S(x,y)$ is such that only the property of reflexivity is guaranteed. Therefore, neither completeness nor transitivity holds, and thus $S(x,y)$ is not an order on set A. In order to obtain an operational prescription, the relation $s(x,y)$ is transformed in a partial or a complete order through an "exploiting procedure" which can be of different nature ([20]). In this specific case a "score procedure" has been adopted. More precisely consider a binary relation $\geq$ being a weak order. Then we have:

$$\geq(x,y) \Leftrightarrow \sigma(x) \geq \sigma(y)$$

where $\sigma(x) = |\{ y: S(x,y)\}| - |\{ y: S(y,x)\}|$.

$\sigma(x)$ being the "score" of x computed as the difference between the number of alternatives to which x is "at least as good as" and the number of alternatives which are "at least as good as" x.