ESSE: An Expert System for Software Evaluation

I.Vlahavas[+], I. Stamelos[+], I. Refanidis[+] and A. 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 different - often contradictory - criteria must be considered in order to reach a decision. This paper presents ESSE, a prototype expert system for software evaluation that embodies various aspects of the Multiple-Criteria Decision Aid (MCDA) methodology. Its main features are the flexibility in problem modeling and the built-in knowledge about software problem solving and software attribute assessment. Evaluation problems are modeled around top-level software attributes, such as quality and cost. Expert assistants guide the evaluator in feeding values to the decision model. ESSE covers all important dimensions of software evaluation through the integration of different technologies.

Keywords: expert system, multiple criteria decision aid, automated software evaluation

This work has been partially supported by Greek Ministry of Development, General Secretariat of Research and Technology, within the program "PENED-95".

1. Introduction

The demand for qualitative and reliable software, conforming to international standards and easy to integrate into existing system structures is continuously growing. Besides, the cost of software production and software maintenance is rising dramatically, as a consequence of the increasing complexity and the need for better designed and user friendly software products. Consequently, the evaluation of such software aspects is of great importance. We will use the term *software evaluation* throughout this paper to denote evaluation of various aspects of software.

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 product 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. Thus, 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 recent 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, 10, 14, 18, 24, 25, 30, 36]. The need for systematic software evaluation throughout the software life cycle has been recognized and well defined procedures have been proposed [37], while there are on-going standardization activities in this field [14]. In [37], the authors expect that, as the evaluation activity grows and matures, new evaluation techniques and tools will appear and that hopefully they will be integrated in the evaluation process they have proposed.

A useful technique for performing evaluations of any kind is the Multiple-Criteria Decision Aid (MCDA) methodology [29, 35]. This methodology is applied to those evaluation problems where the final decision depends on many criteria. In order to perform an evaluation we must select a number of attributes, which can be directly or indirectly associated to measures. This selection is crucial and reflects the point of view under which the evaluation is made. The problem is that these attributes cannot be exactly defined, their relative importance is unclear and their measurement in many cases is not feasible. Another point in the application of MCDA to an evaluation problem is the selection of the appropriate MCDA method. In the literature, a large number of MCDA methods can be found. The task of selecting the most suitable method for an evaluation problem is still based on human experience.

Although supported by dedicated tools, the use of MCDA for software evaluation is rather limited [20], since most decision-makers avoid sophisticated decision processes in the software industry. Even when MCDA methods are used, it is quite probable that their application is incorrect. In 1997, one of the authors was involved, as director of a systems integration department, in a decision process related to the evolution of an information system. A senior consulting firm was asked for assistance and one

of the first steps taken was the preparation of the evaluation framework. The framework that was proposed by the consultants suffered from some of the most common problems in the application of decision support theory: redundant evaluation criteria, insufficient understanding of the measurement methods and the decision process itself, etc.

This paper describes ESSE, a prototype Expert System for Software Evaluation that embodies various aspects of the MCDA methodology and has the following features:

- automation of the software evaluation process
- suggestion of a software evaluation model, according to the type of the problem
- consistency check of the evaluation model and detection of possible critical points
- support of the selection of the appropriate MCDA method, depending on the available information
- assistance provided by expert modules, called throughout this paper *Expert Assistants*, which help the evaluator in assigning values to the attributes of the software evaluation model
- management of past evaluation results, in order to be reused in new evaluation problems

We do not state that tools like ESSE come up with the right responses always, since in general there do not exist objectively right responses. Decisions may be right for a specific client, in a specific context, during a specific decision process. Automated software evaluation improves the evaluation process in terms of better understanding, improved confidence and increased meaningfulness of the result. This belief comes up both from the empirical evidence using ESSE and more generic evaluation tools. Additionally, these tools provide a variety of methods and procedures that the evaluator may use for evaluation purposes. In general, automatic software evaluation is expected to help in avoiding problems similar to those observed in the case of the evaluation framework prepared by the consulting firm mentioned above.

The rest of the paper is organized as follows: Section 2 presents related work. Section 3 presents the principles of software evaluation using MCDA methodology. Section 4 describes the structure of ESSE. Section 5 deals with implementation issues. Section 6 gives an example of an evaluation problem and how it has been handled with ESSE. Finally, section 7 concludes the paper and poses future directions.

2. Related Work

A significant amount of literature and research has been dedicated to decision making support for various problems related to software [2, 3, 4, 6, 19, 23, 25, 27, 37]. In general, the goal is to reach a decision by assigning an absolute value to some software entity (e.g. a software product) or by selecting *one out of N* alternatives. Normally, the decision making task is performed through some mathematical mechanism (utility function in [3], MCDA methods in [25]). Knowledge based techniques have also been employed [4, 23] to resolve specific software engineering problems.

In one of the fundamental books on software engineering [3], various mechanisms for decision

support (figures of merit, utility functions) are reviewed, proposed and compared (e.g. Weighted Sum vs. Delivered System Capability Figure of Merit) and examples of their application are given. Specific software engineering problems are described either within examples (software package selection) or explicitly (optimal hardware-software configuration, optimal computer networks, optimal computing algorithms).

There are various aspects under which an approach related to decision support for software evaluation may be examined and assessed. In our opinion, the most important are the following:

- *Evaluation Scope*: how many different evaluation problems may be solved with the specific approach. Is it possible to resolve new problems with the same basic approach?
- *Software Characteristics*: the software attributes used to reach the decision. Provided that the problem remains the same, is it possible/easy to accommodate new and/or different software attributes to resolve the problem?
- *Decision Support Method* (mathematical or knowledge based): This is more a technical issue, since each approach has its own advantages and disadvantages and may be more suitable to certain types of problems, while less appropriate to others.

The above three main aspects constitute a framework according to which various works related to software evaluation support might be viewed and compared in a systematic way. On this basis we will attempt an analysis of related published material, in order to illustrate where the concepts underlying ESSE are placed in the picture of decision support for software evaluation. Given the high number of related publications, only representative works will be referenced. It must be pointed out that direct comparison is not feasible because of the different target set by each of these works. We will discuss first the main points of the referenced works in order to construct the final picture.

Recently, various papers have suggested techniques and tools that are very helpful to a software manager or engineer but have a limited scope in respect with the list of the possible software evaluation problems, i.e. they deal with a specific evaluation problem (testing tools in [27], component off-the-shelf selection in [19]), with a specific project phase (requirements negotiation in [4]) or with a specific product attribute ([23] deals with the evaluation of a single criterion, *quality*). The work of Kontio [19] is particularly interesting because special attention is given to the definition of custom evaluation criteria.

In [4], Boehm and Hoh state that "*...given the overall scarcity of software expertise, it is worth trying to capture it and make it more broadly available via automated aids...*". They have implemented a system aiming to assist the user in dealing with conflicting quality attributes during the user requirements definition phase, on the basis of a knowledge base. On the other hand, the method proposed by Meskens in [23] concerns software written in traditional languages such as COBOL or Pascal. The target of this work is program quality analysis, in order to highlight the characteristics of a computer program that is difficult to maintain. The ultimate goal is to suggest reengineering actions

through the improvement of various quality factors. The tool used is a set of checklists together with expert rules.

Morisio and Tsoukias propose IUSWARE [25], a framework based on MCDA for the evaluation and selection of software products and can be considered as the methodological reference of ESSE. IUSWARE aims at the evaluation of software products in a formal and rigorous way and provides the basis for the design and application of an evaluation model. IUSWARE assumes that judgment is present in any evaluation and, consequently, measurement and judgment should be used jointly. The main software aspect considered is quality.

In [2]

that X=2 and Y=1 (in the sense that X>Y because 2>1), but we are not allowed to sum or multiply such numbers because they are just ordinal measures. ESSE helps in detecting the second type of errors through its methodological knowledge base and provides advice about the correct use of the available tools (it will tell us for instance that we cannot use a weighted sum in presence of ordinal evaluations). The detection of the first type of errors requires the use of more complicated validation procedures exceeding ESSE capabilities.

The advantages of ESSE are the flexibility in problem modeling and the built-in knowledge about software problem solving, all in an integrated environment with a common user interface. The evaluator is allowed to define his/her own 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 criteria. It is clear that, through these mechanisms, additional criteria, such as *'organizational impact'* or *'benefit'* [6], may be easily introduced in the evaluation model and that a well balanced evaluation result, with the optimum utilization of the evaluation resources, may be obtained. A quick result, based mainly on experience and intuition, may be reached by using only high level criteria, while a more sophisticated approach may be adopted through an expanded attribute structure, supported by measurements related to certain selected attributes. 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. Already proposed knowledge based systems [23] for the quantification of software attributes are comparable to expert assistants in ESSE and, in fact, may be used as such.

Features	**MESKENS 1995**	**BASILI 1995**	**BOLOIX 1995**	**KONTIO 1996**	**BOEHM 1996**	**MORISIO 1997**	**ESSE 1998**
Problem Scope (>1 evaluation problems)		•					•
Evaluation Scope (> 1 Criteria and /or attributes considered)	•	•	•	•	•	•	•
Custom Attribute Definition Support				•			•
Knowledge Base	•	•			•		•
Decision Support Mechanism			•	•		•	•

Table 1: *Summary view of representative works concerning software evaluation aspects.*

Table 1 provides a summary view of the referenced works, depicting the degree to which the most important aspects of software evaluation are covered. Although, as mentioned above, direct comparison is not possible, it is evident that ESSE supports all important dimensions of software

evaluation, achieving the integration of different technologies, namely automated decision support and knowledge based systems.

Finally, another interesting initiated research activity [5] has also a wide scope and focuses on the "*...need for knowledge-based support systems that assist senior management in requirer, acquirer and developer organizations to determine the overall feasibility of a proposed software system architecture, together with the system's requirements, at the earliest possible time and continuing thereafter"*. It is expected that projects like this, striving for 'knowledge-based support systems', will address acquisition problems in a way similar to the approach presented in this paper. These systems will inevitably incorporate decision making support (selection among various architectural solutions, decisions related to project evolution), integrated with more or less stand-alone modules, based on human expertise.

3. Software Evaluation with Multiple-Criteria Decision Aid (MCDA)

An evaluation problem solved by MCDA can be modeled as a 7-ple {*A,T,D,M,E,G,R*} where [35]:

- *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 [25]:

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 evaluation of a single alternative is a problem that arises rarely. In this case, we have to define a prototype entity, which fulfills specific requirements. The evaluation of a single alternative is then translated to the comparison between the actual one and the defined prototype. 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. Finally, evaluation can take place not only for a software product as a whole, but also for parts of it or for certain processes of particular interest (such as production, maintenance etc.)

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*: This is the most important step of the evaluation process. In this step we must define the attributes that will be taken into account during the evaluation and their hierarchy.

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, which is part of the one used in the example of section 6, is shown in figure 1.

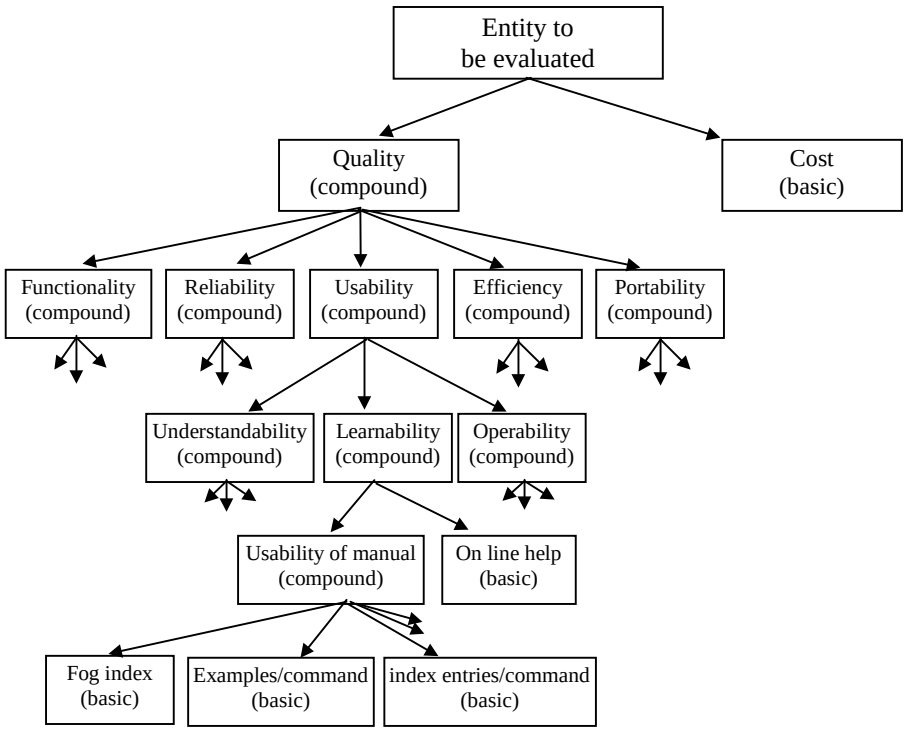

Figure 1: *Example of an attribute hierarchy*

The definition of *D* reflects the point of view under which we make the evaluation. There are two different approaches in an evaluation problem. In the first approach, named “fixed models” [26, 27], a fixed structure is used, where *D* has been definitely identified and customized for a particular domain and type of evaluation. In such cases we have just to fill in the measures. This approach is easy to use but lacks flexibility. In the second approach, named “constructive models”, a general model must be customized [14, 18]. In this approach, *D* is a tree of predefined attributes, depending on the kind of the

problem. *D* may be expanded, modified or reduced. In this case there is more flexibility, but user experience is also required.

A significant constraint is the mutual independence of the attributes. If there is redundancy among them and an MCDA method is used for aggregation, the result will be wrong. Consequently, in constructive models it is important to test for redundancy.

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 "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 has 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 for *d* and let $m_d(a_2)$ be the value of a_2 for *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 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* [17], the *outranking methods* [35] and the *interactive methods* [34]. The selection of an aggregation method depends on the following parameters [35]:

- 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. An illustrative example is presented in section 6.

4. The Expert System Structure

ESSE consists of two main parts, the *Intelligent Front End (IFE)* and the *Kernel*. The structure of the system is shown in figure 2 while the various parts are described in the following.

4.1 Intelligent Front End

The profile of a typical user of the system is expected to be a medium-range manager or a software engineer, with an average knowledge of software engineering practice, but not necessarily familiar with such aspects as systematic cost estimation and quality evaluation. Moreover, the user may be trained neither in software evaluation nor in multicriteria methodology. While the former could be part of his/her background, the latter may be quite unknown.

The role of the IFE is to guide the evaluator in the correct application of the MCDA methodology and to provide expertise on software attribute evaluation. The 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. Software attribute evaluation is supported through Expert Assistants.

IFE consists of three modules, namely *User Interface*, *IFE Tools* and *Expert Assistants*, described in the following:

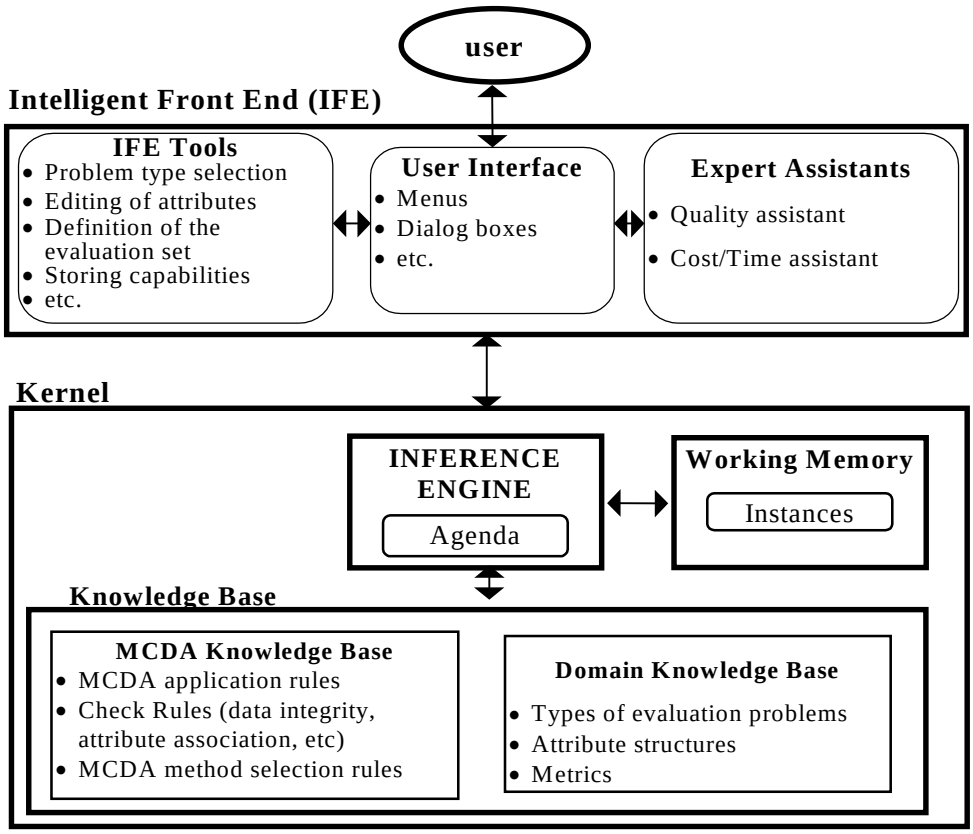

Figure 2: *The structure of ESSE*

4.1.1 User Interface

This module is responsible for the communication between the user and the other modules of ESSE. Communication is accomplished through menus, dialog boxes and other window facilities (see figures 3 through 5, in section 6). The friendly interface of ESSE makes possible the use of the system by relatively inexperienced users (see discussion at the end of section 6).

4.1.2 IFE Tools

IFE Tools are invoked by the User Interface to perform tasks related with the construction of the evaluation model. The most important tasks performed by these tools are the selection of a problem type, the definition and modification of attributes, the definition of the evaluation set and various storing tasks, briefly described in the following:

a) Problem type selection

Each time the evaluator initiates a new evaluation problem, a problem type must be selected. As it will be described in paragraph 4.2.1, ESSE's knowledge base maintains a number of different evaluation problem types. According to the type of the problem that will be selected, ESSE proposes to the evaluator a subsequent tree of evaluation attributes, based on its domain knowledge base.

b) Editing of attributes

According to the selected problem type a hierarchy of default attributes is proposed to the evaluator. The user can modify their definitions, create new and delete existing ones.

c) Definition of the evaluation set

This tool enables the definition and the modification of the evaluation set A, i.e. the software entities to be evaluated.

d) Storing capabilities

This tool provides the ability to store problems in the historical database. Stored problems can be recalled later for further processing. Moreover, it is allowed to store separately only the attributes and their structure, creating additional problem types.

4.1.3. Expert Assistants

These modules 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. Whenever the need for estimation of the above software characteristics appears, the expert assistants may be invoked. A brief description of these two modules follows.

a) Cost/Time Expert Assistant

Humans have been involved in *cost* and *schedule* (*time*) assessments since the first days of computer science. Methods for software development cost estimation fall in three main categories, namely *expert judgment*, *estimation by analogy* and *algorithmic cost estimation* [3]. Expert judgment relies purely on the experience of one or more *experts*. Estimation by analogy compares the software project under consideration with a few (two or three) similar historical projects (i.e. projects with known characteristics, effort and schedule). Algorithmic cost estimation involves the application of a *cost model*, i.e. one or more mathematical formulae which, typically, have been derived through statistical data analysis.

Modern software cost estimation consists of various activities, i.e. the set-up of an appropriate estimation environment (selection and acquisition of methods and tools and collection of historical data), the definition of a cost estimation strategy for a specific software project, the application of one or more cost estimation methods, the combination of estimates obtained through the application of different methods and the assessment of the estimate reliability. Various sub-modules cover all the activities mentioned above, incorporating experience from many different estimation situations.

In particular, the task of generation of estimates is supported mainly through the use of algorithmic models. A lot of expertise is needed in order to avoid the pitfalls of these sophisticated tools and use correctly their results. ESSE provides two popular tools, COCOMO [3] and Function Point Analysis [1, 15], along with rules for their correct application. However, the application of other estimation tools is allowed and an existing knowledge base assists the use of a number of commercial

products. These tools differ in terms of estimate reliability, usability, suitability to a specific application domain, etc. According to the software life cycle phase during which the evaluation takes place (before, during or after development), the evaluator is advised on the feasibility, precision etc. of the measurements that will be needed. For example, the estimation of software size is much more difficult during the requirement definition phase than after product delivery and is supported by rules for the correct sizing of a software product in terms of source code length.

An interesting point is that ESSE's concepts may be applied in the context of expert assistants as well. As an example, consider the problem of algorithmic model quality assessment, part of the estimation environment sub-module. Models are characterized through a number of criteria such as *constructiveness*, *stability* ([3], chap. 28) and they may be evaluated and compared through a multiple criteria analysis. The problem of cost model selection may be described using ESSE, each cost model criterion may be assessed through guidelines based on expertise and, finally, any available MCDA method in ESSE may be employed to provide aggregate model assessment and selection. In addition, ESSE provides the results of a completed quality assessment between a number of available commercial tools for cost estimation and time.

The knowledge base of the cost expert assistant described above, has been derived from the authors' experience in software cost modeling and the extensive use of commercial tools in a telecommunications environment [7, 9]. However, the knowledge base is not biased towards any specific type of systems, since various system typologies can be found in the telecommunications field (network management systems, management information systems, billing systems, real-time systems, etc.). Additionally, the rules related to the estimation strategy may be applied to an arbitrary cost estimation situation.

b) Quality Expert Assistant

This module supports the quantification of quality basic attributes. The quality expert assistant has the form of on-line help, proposing measurement methods. The following example provides instructions for assigning values to the attribute 'stability' which is sub-attribute of 'maturity' [14]. As mentioned above, this knowledge comes mainly from published results in software engineering studies.

> To estimate the stability of a product, the following metrics
> may be taken into account:
> - number of corrected faults
> - estimated number of total faults
> To estimate the number of total faults you have to fit the curve
> of the distribution of corrected faults versus time and integrate it.
> Stability can be expressed as the ratio of the above metrics.

Similar instructions are provided for the majority of the basic quality attributes proposed by ESSE.

4.2 Kernel

The kernel of ESSE consists of three main parts: the *Inference Engine,* the *Working Memory* and the *Knowledge Base*.

The *Inference Engine* is used each time a set of rules is invoked by the rule agenda. Rules used by ESSE follow the forward chaining reasoning technique [11]. At each cycle of the inference process, a rule, with satisfied conditions, is triggered and its 'then' part is executed. If no rule conditions are satisfied, the inference process stops and either another set of rules are invoked, or the control is returned to the Front End.

The *Working Memory* keeps information about the current status of the evaluation problem. It is represented with instances of frames defined in the knowledge base. For example, working memory includes instances with the current values of the attributes for each choice of the evaluation set.

The *Knowledge Base* is the most significant part of the system. It consists of two parts, namely the *MCDA Knowledge Base* and the *Domain Knowledge Base*. In the following sub-sections these modules are presented in detail.

4.2.1 Domain Knowledge Base

This part of knowledge base contains structural knowledge in the form of *frames*. It contains knowledge about various types of evaluation problems, attribute definitions and metrics. More specifically:

a) Types of evaluation problems

There are various types of software evaluation problems. Each time the evaluator encounters an evaluation problem of a type that has been met in the past, the system proposes a default attribute hierarchy, recalled from the Domain Knowledge Base, which can be reused with any modifications. When applicable, this facility significantly reduces the evaluation effort. Moreover, using the storing capabilities of the front end, the evaluator can define his own problem types.

The types of problems supported by ESSE are the following:

i) Keep or Change: This situation arises when a particular software product is already in place and, due to certain business needs, it is examined whether it is still valid or should be replaced by a new product.

ii) Make or Buy: This situation arises when a particular software product is required due to the business needs of a company and a decision must be made: should the product be acquired from a choice available in the market or should the product be developed under the control of the company?

iii) Commercial Products Evaluation: In this situation, a decision has been made to buy a ready-to-use commercial product and there are more than one such products. Each of these products should be evaluated and the most suitable will be acquired.

iv) Tenders Evaluation: Such a situation occurs mainly after an "external buy" decision is made. In this situation, the evaluation will be based on the specifications of the different tenders.

v) Software certification: This is typically an activity carried on by a testing laboratory, either specialized in specific software or general purpose (commercial products). The evaluation could be done either according to the national or international standards, or according to the demander of the certification.

vi) Software process evaluation: Software products are manufactured through software development processes. An evaluation problem arises when a software process must be chosen for the development of a specific product or as a standard software corporate process.

vii) Software architecture selection: In this situation we must choose one software platform among others, i.e. Windows NT or UNIX, Oracle or Ingress, etc. Alternatively, a specific architectural style may be chosen for an information system.

viii) System Design Selection: In this situation we must choose one out of *N* possible design options (for example, when defining the detailed software architecture of the product to be developed).

Although covering a large range of possible situations, the list of problem types presented does not constitute an exhaustive list. Moreover, it is possible to find combined problem situations that span over more than one of these items. ESSE does not provide currently an automated combination of different problem types, an issue that is in our future plans. However, since ESSE does not oblige the user to follow specific problem formulations, it is still possible to take advantage from the experience with typical problem types.

Past evaluation results are handled in a relatively simple way. Upon request, the system presents a list with the types of evaluation problems that have been met in the past. The user selects the problem type that matches the evaluation situation in hand. At this point, he may consult the list of problem instances of the selected type that have been solved in the past.

Feedback is given through structured comments of the decision-maker. For instance, an evaluation is rated as “successful”, if the result has been judged as satisfactory (e.g. a commercial product has been chosen and has been used for some time with acceptable performance). Comments may include tips for improvements in the evaluation process that has been followed. In case of evaluation “failure”, a list of points where wrong steps have been taken (wrong measurement method, disproportionate weight assignment) may be provided, along with suggestions for possible amendments.

At this point, the user may select a successful solution and follow the steps of the previous decision maker, or try a less successful case, for which it is clear how to proceed, taking profit from the errors made in the first place. In this case, he inherits all reusable information and knowledge in his

problem. Alternatively, he is allowed to start from scratch and define a new problem type, provided that some basic evaluation elements are different from those of the already existing problem types (e.g. a new attribute structure is needed).

b) Attribute structure

Whenever the evaluator decides that the problem to be solved matches one of these typical problems, a standard predefined structure of attributes is proposed by the system. Typically, the solution proposed consists of a constructive model, where the tree of attributes *D* contains three or four top-level attributes. However, it is always possible to modify the proposed model to match the specific problem requirements. The default top-level attributes are: '*Quality*', '*Cost*', '*Time*' and '*Standards*' and are discussed separately in the following paragraphs. Cost of production, time to market and quality have been identified as most important in the software industry [15]. Compliance with standards has been added to accommodate the increasing demand for software complying with specific standards. These four attributes are used frequently in software evaluations and there is a significant amount of human experience to support them. Moreover, frequently a *veto* preference is expressed on them (meaning that the failure of an alternative to satisfy the thresholds imposed on one of these attributes will result in the rejection of this alternative).

i) '*Quality*' is defined [14] as the set of attributes of a product or part of it that determine its ability to satisfy stated and implied needs. Each of these attributes can be decomposed into sub-attributes. Each basic attribute is assessed according to measurements and expert judgment, supported by historical data. According to [14], quality is decomposed in the following way (in the parentheses are the sub-attributes of each attribute of quality):

'Functionality' ('suitability', 'accuracy', 'interoperability', 'compliance', 'security')
'Reliability' ('maturity', 'fault tolerance', 'recoverability', 'availability')
'Usability' ('selectability', 'learnability', 'operability')
'Efficiency' ('time behavior', 'resource utilization')
'Maintainability' ('analyzability', 'changeability', 'stability', 'testability')
'Portability' ('adaptability', 'installability', 'conformance', 'replaceability')

The attributes referenced above within parentheses can be further decomposed. For example, 'operability' may be decomposed in 'availability of setup installation', 'default value availability ratio', 'message clearness', 'mean time between human errors', 'ultimate operation time', 'status or progress report availability ratio' and 'human error operation cancelability ratio'.

ii) '*Cost*' and '*Time*', in the most general case, are composed of acquisition, customization, training, operation and maintenance cost and time. Evaluation resources used are cost estimation methods and tools (expert judgment, estimation by analogy, algorithmic cost models), statistics and historical data on past software projects.

iii) '*Standards*' denotes the compliance with international, national or corporate standards and is assessed according to inspections and validation tests on the specifications or the software product/process itself.

As an example, the frame that defines the attribute 'availability of setup installation', which is sub-attribute of 'operability' is as follows:

frame 'availability of setup installation' is an attribute;
default type is 'nominal' and
default list is {'yes', 'no'} and
default unit is unknown and
default parent is 'operability' and
default weight is 0.5 .

For each attribute the system maintains the type, the valid values, the unit, the weight and the parent attribute. ESSE supports three types of attributes with the characterizations 'arithmetic', 'nominal' and 'no type'. Usually basic attributes are either 'arithmetic' or 'nominal', whereas compound attributes are of 'no type'.

In MCDA, an attribute is introduced at the appropriate hierarchy level (i.e. either as a basic or as a compound one), depending on its importance. As an example, consider *'Standards'*, which is proposed by ESSE in various problems. Along with this default option, an alternative solution is the introduction of *'compliance'* (sub-attribute of *'Quality'*) instead of *'Standards'*, if this problem characteristic is considered by the evaluator of less importance than the other top level attributes.

As another example, consider again the quality hierarchy described in [14]. Some of these attributes are related to '*Cost*' rather than pure '*Quality*'. These are 'Usability', 'Maintainability' and partially, 'Portability', which are defined in terms of "...effort to understand, learn, operate, analyze, modify, adapt, install, etc". Exceptions may be 'Stability' (defined as the risk of unexpected effects from modifications), 'Adaptability' (defined as the effort required to adapt a software product to different specified environments) and 'Replaceability' (defined as the opportunity and the effort required to put a software product in place of other specified software).

The system proposes either the standard quality structure of [14] or refined solutions, in which these dependencies and redundancies have been resolved (e.g. redundant sub-attributes have been removed). In all cases, due to the check rules in the MCDA knowledge base that will be discussed later, possible dependencies and redundancies will be detected, resulting in warning messages and advising on the eligible MCDA methods.

c) Metrics

As already discussed in section 3, the fourth step of the application of MCDA is the definition of the measurement methods to be used in the decision process. In case that the decision maker wishes to adopt a custom criterion or the problem is not of a known type, the system will prompt for attribute

structure and measurement profile for each attribute, in the form of the 4-tuple *<value type, range, weight, metric>*, e.g. *<ratio,0-100, 0.5, currency>*.

There has been much research into the usefulness of various software metrics and there is empirical evidence that some of them are related to various software product and process attributes. Examples of software metrics are the *McCabe* complexity metric [22], the *Halstead* measure [12], the source code length [8], the *Function Points* [1, 15], etc. Additionally, CASE tools have been available for many years and widely used, providing, among other, automatic support for software measurement. Consequently, a software evaluator may wish to utilize modern software measurement results in some points of his/her software evaluation model, with the necessary precautions in metrics use [10]. ESSE supports the correct use of software metrics, by offering embedded knowledge about the proposed measurement methods in the related literature and the up to date metric and attribute inter-relationship findings, through the appropriate expert assistants and consistency check rules.

ESSE's knowledge base provides a metrics library consisting of typical business metrics (such as *currency*, *time*) and metrics specific to software [10]. Moreover, Prolog *facts* represent the relationships between metrics, derived from experience and practice in software engineering. For example, a straightforward relationship exists between *man-months* (effort) and *currency* (development financial cost). Another example are the various hypotheses that have been made for the possible correlation between the McCabe complexity metric (*MCM*) and number of defects (*ND*) and between MCM and source code length (SCL) [33]. ESSE implements this knowledge in terms of facts, such as:

```
associated(
        metric(MCM),
        metric(ND)
        because(explanation),
        ).
```

and the explanation is «*… a study by Troster on a large SQL product has revealed a relatively strong statistical correlation between the two metrics*». Although in some cases, associations between attributes are straightforward, in subtle situations the model consistency check is quite useful. The knowledge embedded in the system has been extracted from published results of related studies [10, 16] and is continuously updated, as more results become available. User defined *ad hoc* metrics, metrics inter-relationship definition and explicit inter-relationship between attributes (such as those described in [4]) are also allowed by the system. Attributes and their characterization are captured and inserted in the library for future reuse and consultation.

4.2.2 MCDA Knowledge Base

This part of knowledge base contains behavioral knowledge in the form of production and deductive rules. We can distinguish these rules in the following groups: MCDA application rules, Check rules and MCDA method selection rules. More specifically:

a) MCDA application rules

These rules are mainly deductive rules and perform the various steps of the supported MCDA methods. Currently, three MCDA methods have been implemented, namely *Weighted Average Sum,* or briefly WAS [35], *ELECTRE II* [28] and *ELECTRE IV* [13]. WAS is a multiple-attribute utility method, which requires ratio scales at the level of basic attributes and the definition of weights in terms of trade-offs for all attributes. ELECTRE II is an outranking method that can deal with both arithmetic and nominal attributes with weights while ELECTRE IV is an outranking method that ignores weights completely.

b) Check rules

These are productive rules and are used to verify the validity and integrity of the evaluation model. We can distinguish two types of check rules. The first type deals with the *data integrity* of the evaluation model, to ensure that the selected MCDA method is applicable. This is accomplished by verifying that values and weights (if needed) have been assigned to all basic attributes, otherwise warning messages are displayed.

The second type of check rules deals with *attribute associations*. For the correct application of the MCDA methodology, redundancies are not allowed between the selected attributes. These rules check, using the domain knowledge base, the existence of redundancies between the attributes of the evaluation model and display appropriate warning messages.

c) MCDA method selection rules

This part of the MCDA knowledge base contains production rules which help the user to select the MCDA method that is eligible for the application of the model, taking into account the attribute hierarchy and the defined measurement profiles. For instance, methods that do not use weights are expected to have a lot of top level attributes, while methods that do use weights may have few top level attributes (e.g. only two). The corresponding rule is:

```
IF count(top_level (Attribute),X) and X is less than 4
THEN suggest_usage_of_weights.
```

Another rule proposes an outranking method if there are basic attributes that are not arithmetic:

```
IF basic_attribute(Attribute) AND
Attribute's type is different from 'arithmetic'
THEN  suggest_usage_of_outranking_method.
```

5. Implementation Issues

Currently a prototype of the system has been developed, using as development platform both LPA WinProlog[1] and Flex Expert System Shell [21], running on MS-Windows Personal Computer. Both

[1] LPA Prolog and Flex are trademarks of LPA Ltd, London England

LPA WinProlog and Flex provide a large number of high-level programming primitives. LPA WinProlog in particular, provides many graphical primitives, which have helped the development of the Intelligent Front End.

The Domain Knowledge Base (attributes, evaluated entities, etc.) is represented with frames, instances (supported by FLEX) and facts (supported by Prolog) and is stored in text files. In this way, it can be accessed either with ESSE or with a text editor. The MCDA Knowledge Base is represented with production rules, supported by Flex and deductive rules, supported by Prolog.

Quality and Cost expert assistants have been implemented, as described in section *4*. Moreover, other expert assistants have been integrated with ESSE. For example, the Function Points Assistant (FPA) supports the calculation of the functionality of a software product, using Function Points Analysis. The measurement of Function Points may be involved in functional size and cost-time estimation.

6. Example

As an example, we will use the evaluation of three commercial expert system shells. According to section *3*, the first step is to define the evaluation set *A*, which is the following:

- FLEX Expert System Shell developed by Logic Programming Associates
- NExpert developed by Neuron Data
- CLIPS developed by NASA.

The second step is the definition of the type of the evaluation. Currently ESSE supports only sorting. This type is the most general, ranking the elements of *A* from the best to the worst.

The third step is to define the tree of attributes *D*. In order to do this, the user must select the type of the problem. This example was classified as a 'Commercial Products Evaluation' problem. For this type of problem, ESSE proposes three top-level attributes, '*Quality*', '*Cost*' and '*Time*'. '*Time*' was considered irrelevant by the evaluator, because the commercial products under consideration were already developed and no customization activities were foreseen. '*Cost*' was not analyzed further but was the official price of each product.

As described in 4.2.1b, '*Quality*' is decomposed in six sub-attributes, namely 'functionality', 'reliability', 'usability', 'efficiency', 'maintainability' and 'portability'. 'Maintainability' was removed, because it was irrelevant for the systems under evaluation. The rest of the sub-attributes were modified in order to fit better to the characteristics of the expert system shells. For example, new sub-attributes were added to 'functionality', such as 'forward chaining', 'backward chaining', 'multiple inheritance' (see figure *3*).

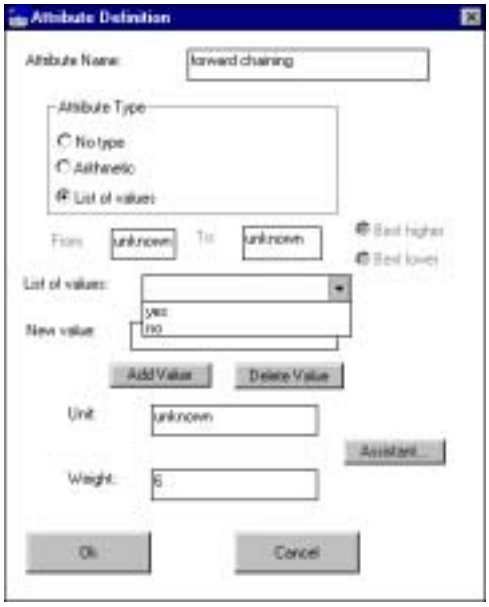

Figure 3: *The definition of attribute 'forward chaining'*

Some of the basic attributes were defined as 'arithmetic' while some other as 'nominal'. To each arithmetic attribute a scale was assigned, whereas to each nominal attribute valid values were defined (steps *4* and *5*, according to section *3*). Moreover, weights were assigned to all of them. Notice that in general, each weight must be a non-negative number, without any upper limit. What are taken into account are the ratios between the weights of attributes with common parent. For example, 'functionality' was given a weight equal to *8* while 'usability' was given a weight equal to *4*. This means that 'functionality' was twice important as 'usability', according to our point of view.

Thresholds were assigned to all arithmetic basic attributes and to all intermediate compound attributes (step 6). The threshold of an attribute indicates how much one software entity must surpass in this attribute another entity, so that the former be considered better than the later, with respect to this attribute. These thresholds are used by outranking aggregation methods, such as ELECTRE.

After constructing the evaluation model, the system suggested the use of ELECTRE II (step *7* of section *3*). For this suggestion the two rules referenced in 4.2.2c fired. The first rule detected that the top-level attributes are only two and suggested the use of a method that employs weights. The second rule detected that the model had nominal basic attributes and suggested the use of an outranking method. The only implemented method fulfilling these requirements is the ELECTRE II, which is both an outranking method and supports weights.

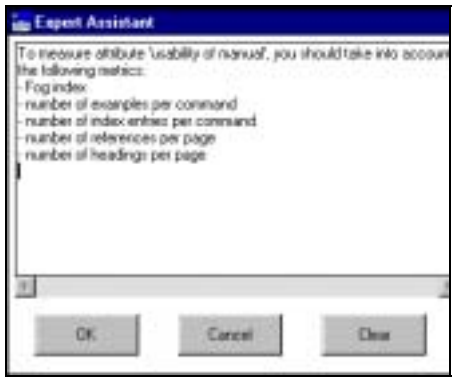

Figure 4: *Expert assistant for the attribute 'usability of manual'*

Finally, values were assigned to the basic attributes of each product. This task was assisted by the help facility provided by the quality expert assistant, guiding the measurement activities. In figure *4*, instructions for the measurement of 'usability of manual', sub-attribute of 'learnability', are provided. Appendix A presents all the attributes used in this example, along with types, ranges, weights and values assigned to the three products for each of the basic attributes.

The final step is to carry out this method and obtain the results. ELECTRE II ranked the three products in the following way (figure 5):

CLIPS > FLEX > NExpert

The low price of CLIPS played an important role in the result. In order to evaluate the products taking into account only their quality, we set the weight of '*Cost*' equal to *0* and carried out one more evaluation using ELECTRE II, with the following results:

FLEX > CLIPS > NExpert

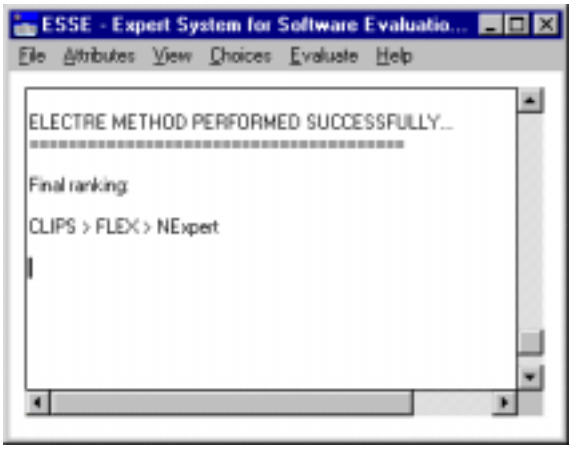

Figure 5: *Results obtained with ELECTRE II*

It is worth mentioning that the above results are perfectly consistent with the authors' impression of these products. After completing the above example, the attributes used in this problem were saved into a file, without details for the evaluation set, in order to be reused in future evaluations with problems for the same type.

Having in mind the example above, it should be noticed that the results reflect the point of view under which the evaluation is performed. Carrying out the evaluation under a different point of view, i.e. with a different attribute hierarchy and/or weights, the final results might have been different. Though, no system can prevent the user from assigning arbitrary weights to the attributes or defining a meaningless attribute hierarchy and, ultimately, obtain any result from the evaluation process. What ESSE does here is the proposal of a reasonable attribute hierarchy together with weights. However, it must be noted that assigning weights to attributes is not a key issue by itself. The real key issue is the choice and the implementation of the aggregation procedure, since we know that there exists no universal procedure for preference aggregation. ESSE provides methodological knowledge on how to choose the most appropriate aggregation procedure and further it provides advice on how to define correctly its parameters. It is up to the user's responsibility to make good use of ESSE's features for his/her own benefit.

Finally, ESSE does not implement a precise procedure for sensitivity and robustness analysis, but facilitates such analysis providing the possibility to test different combinations of the parameters. However, the system is quite insensitive to small variations of the attribute values and weights. This is a main characteristic of the MCDA methodology and especially of the outranking methods and is due to the thresholds that are used by them. For example, in order that ELECTRE concludes that product X is better than product Y, X must substantially surpasses product Y in an adequate number of attributes. With small variations in the values of the attributes and/or the weights it is impossible to conclude that product Y is better than product X (although it is possible to conclude that the two products are of equal preference).

The above example demonstrates how the authors have used ESSE for an evaluation of their interest. However, other users have employed the system as well. Recently, ESSE has been used by a large transport organization, which was faced with a rather typical evaluation problem: the evaluation of certain proposals for the evolution of its corporate information system [31]. Moreover, ESSE has been used for the evaluation of educational software products [32]. In this case, the knowledge base of the system has been enhanced with a number of additional attributes, concerning the educational effectiveness of the various products. In all cases, ESSE has helped the evaluators in the construction and correct application of the evaluation framework for each problem.

7. Conclusions and Future Work

In this paper we described ESSE, a prototype expert system for software evaluation that embodies the Multiple-Criteria Decision Aid methodology. It is obvious that the software evaluation process is a complicated task. Issues, such as selection of the appropriate attributes, creation of their hierarchy, assignment of relative weights to them and application of a sound decision aid methodology arise frequently for non-experienced users but also for skilled ones.

ESSE, through its Domain Knowledge Base, helps the evaluator to select the appropriate criteria, according to the type of the evaluation problem and construct the evaluation model. Evaluation models for various types of software evaluation problems solved in the past are stored in the knowledge base. The evaluator can reuse these models in new problems, having the ability to modify them by adding, removing and modifying attributes. In addition, the MCDA Knowledge Base, supports the selection of an appropriate MCDA method, according to the problem formulation.

The Expert Assistants support the quantification of the basic attributes used in the evaluation. The Cost Expert Assistant helps the evaluator to select and apply the appropriate cost model in order to estimate the cost of a software development project, while the Quality Expert Assistant, in the form of on-line help, guides the evaluator to assign values to basic attributes related to quality. Finally ESSE allows the feedback of evaluation results, in order to capture the evaluator’s satisfaction and therefore facilitate future reuse of the evaluations.

Our future plans foresee the integration of more MCDA methods in ESSE, such as other outranking and multiple attribute utility methods, some interactive methods, etc. We will explore the applicability of the supported MCDA methods to the various types of software evaluation problems, obtaining rules of experience. Moreover, it is planned to maintain the knowledge bases, by inserting new findings in software engineering practice and by applying ESSE to numerous software evaluation problems of different types. We are also working on a user directed algorithm for automated combination of past evaluation problems.

Finally, an interesting idea is the development of a WEB based version of the system. The purpose of this version is twofold: first, many people could take advantage from the use of the system and second, the enhancement of the knowledge base of the system could be accelerated drastically.

As a conclusion, we believe that ESSE is a useful tool for decision-makers dealing with software evaluations and will contribute in clarifying and standardizing in the future this vague area.

Acknowledgements

We would like to thank the anonymous referees of the KBS journal for their helpful comments on the first submitted version of the paper.

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] Boehm B.W., *Software Engineering Economics* (Prentice-Hall, 1981).

[4] Boehm B., In Hoh, Aids for Identifying Conflicts Among Quality Requirements, *IEEE Software*, (1996).

[5] Boehm B. and Scacchi W., SAMSA: Simulation and Modeling for Software Acquisition: Air Force Opportunities, Extended Report (1997).

[6] Boloix G. and Robillard N.P., A Software Evaluation Framework, *IEEE Computer*, vol. 28, no.12, (1995) 17-26.

[7] Capacci C. and Stamelos I., Constructing Software Cost Models, *Proc. 2nd Conf. on Achieving Quality In Software* (1993).

[8] Conte S.D., Shen V.Y. and Dunsmore H.E., *Software Engineering Metrics and Models* (Benjamin Cummins Publishing Inc., 1986).

[9] Costamagna M., De Bonis R., Squarotti R. and Stamelos I., An Integrated Environment for Productivity Assessment of Telecommunications Software, *Proc. European Conference on Software Cost Modelling '95*, pp. 21.1-21.16 (1995).

[10] Fenton N., *Software metrics - A Rigorous Approach* (Chapman & Hall, London, 1991).

[11] Giarratano J. and Riley G., *Expert Systems: Principles and Programming* (PWS Publishing Company, Boston, 1994).

[12] Halstead M.H., *Elements of Software Science*, (Elsevier, N-Holland, 1975).

[13] Hugonnard J. and Roy B., Ranking of suburban line extension projects for the Paris metro system by a multicriteria method, *Transportation Research* 16A (1982), 301-312.

[14] ISO/IEC 9126-1, Information Technology - Software quality chracteristics and sub-characteristics (1996).

[15] Jones Capers, *Applied Software Measurement*, (McGraw-Hill Inc., New York, 1991).

[16] Kan S. H., *Metrics and Models in Software Quality Engineering*, (Addison Wesley Longman, Inc., 1994)

[17] Keeney R.L. and Raiffa H., *Decision with multiple objectives*, (John Wiley, New York, 1976).

[18] Kitchenham B., Towards a constructive quality model. Part 1: Software quality modeling, measurement and prediction, *Software Engineering Journal* (July 1987).

[19] Kontio, A Case Study in Applying a Systematic Method for COTS Selection, *Proceedings of the IEEE Int'l Conference on Software Engineering* (1996).

[20] 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), pp. 115-126.

[21] LPA flex, *LPA flex Technical Reference*, (Logic Programming Associates ltd, 1997).

[22] McCabe T.J., A complexity measure, *IEEE Trans Soft Eng* 2(4) (1976), 308-320.

[23] Meskens N., A knowledge-based system for measuring the quality of existing software, *Revue des systemes de decision*, vol.3, no.3 (1994), 201-220.

[24] Miyoshi T. and Azuma M., An empirical study of evaluating software development environment quality, *IEEE Transactions of Software Engineering*, SE-19 (1993).

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

[26] Mosley V., How to assess tools efficiently and quantatively, *IEEE-Software* (May 1992).

[27] Poston R.M. and Sexton M.P., Evaluating and selecting testing tools, *IEEE Software*, (May 1992).

[28] Roy B. and Bertier P., La methode ELECTRE II - Une application au media planning, *in OR72*, M. Ross (ed.), North Holland, Amsterdam (1973), 291-302.

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

[30] Schneidewind N.F., New software quality metrics methodology. Standard fills measurement needs, *IEEE Computer*, vol. 26, no. 4 (1993), 105-106.

[31] I.Stamelos, I. Vlahavas, I. Refanidis and A. Tsoukias, Knowledge Based Evaluation of Software Systems: a Case Study, Aristotle University, Dept. of Informatics, technical report.

[32] I. Stamelos, I. Refanidis, P. Katsaros, A. Tsoukias, I. Vlahavas and A. Pombortsis, Automating the Evaluation of Educational Software, to be presented at the 5th International Conference of the Decision Sciences Institute, Athens, 4-7 July 1999.

[33] Troster J., *Assessing Design-Quality Metrics on Legacy Software* (Software Engineering Process Group, IBM Canada Ltd. Laboratory, North York, Ontario, 1992).

[34] Vanderpooten D. and Vincke P., Description and analysis of some representative interactive multicriteria procedures, *Mathematical and computer modelling*, 12 (1989), 1221-1238.

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

[36] Vollman T.E., Software quality assessment and standards, *IEEE Computer*, vol.26, no.6 (1993), 118-120.

[37] Welzel D., Hausen H.L, Boegh J., Metric-Based Software Evaluation Method, *Proceedings BCS 1st European International Conference on Software Testing, Analysis and Review* (London, 1993).

APPENDIX A

Attributes and values used in the Expert System Shell evaluation example

(In the following A means 'arithmetic', N means 'nominal' and a hyphen (-) means 'no type')

Attributes	Type	Range	Weight	Flex	NExpert	Clips
quality	-	-	2			
functionality	-	-	8			
suitability	-	-	1			
functional specification change ratio	-	-	1			
forward chaining	N	yes/no	6	yes	yes	yes
backward chaining	N	yes/no	6	yes	yes	no
frames/instances	N	yes/no	4	yes	yes	yes
objects	N	yes/no	2	no	no	yes
multiple inheritance	N	yes/no	2	yes	yes	no
actions/procedures	N	yes/no	2	yes	no	yes
demons	N	yes/no	3	yes	yes	yes
custom dialogs	N	yes/no	4	yes	no	no
displaying agenda	N	yes/no	3	yes	yes	yes
number of conflict resolution	A	[1,10]	4	6	7	7
fuzziness	N	yes/no	2	no	no	yes
supporting other languages	N	yes/no	3	yes	no	no
connectivity to other languages	N	yes/no	2	yes	yes	yes
GUI	N	yes/no	3	yes	yes	yes
connectivity to databases	N	yes/no	3	yes	yes	yes
reliability	-	-	1			
maturity	-	-	1			
age in years	A	[0,20]	1	6	10	8
usability	-	-	4			
understandability	-	-	1			
availability of demonstration software	N	yes/no	1	no	no	yes
learnability	-	-	3			
usability of manual	-	-	1			
Fog index	A	[0,10]	1	7	7	7
number of examples per command	A	[0,5]	1	2	1	2
number of index entries per command	A	[0,5]	1	1	1	1
number of references per page	A	[0,5]	1	0	0	0
number of headings per page	A	[0,5]	1	3	2	2
on line help	N	yes/no	1	yes	yes	yes
operability	-	-	1			
availability of setup installation procedure	N	yes/no	1	yes	yes	yes
message clearness	N	good average bad	1	average	average	good
efficiency	-	-	2			
resource utilization	-	-	1			
main memory utilization	A	[0,32]	1	8	8	8
disk utilization	A	[1,20]	1	5	18	5
portability	-	-	2			
MS-Windows	N	yes/no	3	yes	yes	yes
MacOS	N	yes/no	2	yes	yes	yes
Unix	N	yes/no	2	no	yes	yes
cost	A	[0,3000]	1	2300	2600	150