IusWare: a methodology for the evaluation and selection of software products

M. Morisio
A. Tsoukiás

Indexing terms: Software product evaluation, Methodology, Multicriteria decision aid

Abstract: IusWare (IUSTitia SoftWARis) is a methodology designed to evaluate software products in a formal and rigorous way. The methodology is based on the multicriteria decision aid approach and encompasses activities such as comparison, assessment and selection of software artefacts. The methodology defines an evaluation process which consists of two main phases, designing an evaluation model and applying it. The design phase is made up of the following activities: first, identifying the actors relevant to the evaluation, their role, the purpose of the evaluation, the resources available and the object(s) of the evaluation; secondly, identifying the type of evaluation required: either a formal description of products or the ranking of products from the most preferred to the least preferred or a partitioning into two sets of the best and the remaining products; thirdly, defining a nonredundant hierarchy of evaluation attributes, often corresponding with the quality characteristics of quality models; fourthly, associating a measure, a criterion scale and a function to transform the measure scale into the criterion scale to each basic attribute; and finally, choosing an aggregation technique so as to aggregate values on criteria to form a recommendation for the selection. In the application phase attributes of products are measured, measures are transformed into values on criteria and aggregated to form a recommendation.

1 Introduction

The question as to whether product A or product B should be adopted or whether module X developed by a subcontractor should be accepted or whether COTS (component off the shelf) Y or Z should be reused is a question which has to be faced more and more frequently as the use of software becomes more diffuse and it has become increasingly necessary to develop a software quality evaluation technology.

According to Fenton's [1] measurement framework, a product can be evaluated by considering a number of product attributes, which can be either external (functionality, usability, maintainability, etc.) or internal (size, algorithmic complexity, etc.) and they can be either directly or indirectly associated with a measure.

Documents such as [2, 3] define an evaluation process which involves
(a) The identification of the relevant quality attributes of the product
(b) The measurement of the attributes using suitable measures
(c) The aggregation of measures

Other relevant work will be presented and analysed using this evaluation process which will also be discussed. However, the strictly related topic of software production process evaluation will not be dealt with in this paper.

1.1 Software product evaluation

1.1.1 Identification of quality attributes: In fixed quality model approaches [4, 5], quality attributes have been definitely identified and customised for a particular domain and type of evaluation and in such cases the evaluator has only to fill in the scores. This approach is practical but, of course, not general. In a similar way the documents [3, 6] define a set of external quality attributes for CASE tools. In [7], this approach is used to evaluate any software system with a simple model composed of nine factors.

In constructive quality model approaches [2, 8, 9] the evaluator has to customise a general quality model which means that there is maximum flexibility, but there is also need for skilled people and time. To our knowledge only the goal question metric approach [10, 11] has been proposed as an aid in this field.

The need to test the nonredundancy of the quality attributes chosen for the constructed quality model has not been dealt with in any detail in the literature, although this is a very important condition. For instance in cases where some quality attributes are redundant and a weighted sum method is used as an aggregation procedure, the use of this procedure increases the performance of a particular product in an unacceptable way.

1.1.2 Measurement of attributes: The measurement of attributes, especially external ones, is difficult,
either because the collection of data is time consuming and impracticable, or because no correlation has been established between the proposed measures and the attribute.

The idea that some form of judgement has to be made in any evaluation can be found in [10, 12, 13]. Measures are numbers or symbols objectively assigned to an attribute [1], preferences are subjectively assigned to an attribute in terms of binary relations. For instance the length of an object is a measure (an objective assignment, assigning standard length measurement) while the preference between a long object and a short one is a subjective judgement depending on the purpose and the context of the evaluation. It is important not to confuse the concepts of measures and preferences. Preferences can be defined by starting from measures (they are associated to measured attributes) or independently (they are associated to nonmeasured attributes). An attribute which is assigned a preference relation is a criterion. An essential judgement is made using criteria based on measured attributes. A nonrepeatable judgement is made using criteria based on nonmeasured attributes [10].

The distinction between measures and preferences has not been defined with sufficient clarity [4, 5, 14–18] and thus there is the risk of confusing preferences, usually presented numerically, with measures, which means that the element of subjectivity is not immediately evident.

An essential judgement is to be preferred as it allows less space to conscious or unconscious bias in evaluations. A nonrepeatable judgement should be used only when the measure does not yet exist or it is impractical or too expensive to use. The choice should always be discussed and justified.

1.1.3 Aggregation techniques: Measures or preferences collected on several attributes and several products constitute a large amount of data. To make a selection decision, these data have to be summarised into the smallest number of aggregated indicators by using an aggregation technique.

The weighted average sum (WAS) aggregation technique is used in [4, 5, 14, 15, 17]. It is also used in most everyday evaluations. However, it should be noted that WAS requires attributes which have at least ratio scales, while the majority of attributes in evaluations can be characterised only by ordinal scales.

Current practice is often to transform an ordinal scale (for instance very high, high, fair, low, very low associated to attribute usability) into a ratio (the scale becomes 5, 4, 3, 2, 1) WAS interprets four as twice two, while it is meaningless to say that high is twice low. This interpretation is essential to WAS when weights are applied, since weights are in fact trade-offs between attributes, or ratios between the scales. If weight two is assigned to usability, a low in usability becomes a high. This is meaningless with ordinal scales.

Even when attributes have ratio scales, WAS can have a dangerous effect, called compensation, that hides situations of incomparability with indifference. Given a model with quality and cost, product A scoring five on quality and one on cost, product B scoring one on quality and five on cost, the aggregated score according to WAS is three for both products, that is to say they are indifferent.

The arbitrary assignment of weights to attributes has another dangerous effect. If quality is assigned weight two, cost weight one, two units of cost compensate one unit of quality. This is not necessarily incorrect, but clients are not usually aware of this relationship, or that its immediate consequence is that a preference of two units in cost can completely compensate for an inverse preference of one unit of quality. Since the assignment of weights is arbitrary, a weight can always be found to make the worst product on attribute quality have the best aggregated score.

The analytic hierarchy process (AHP) (described in [19]) is used in [16, 20]. The hierarchical nature of the data used by the AHP may correspond in some way to hierarchical quality models. However, this is not always the case and the assumption that there should be complete comparability and the imposition of ratio scales at all levels of the hierarchy is very demanding. The AHP technique does not offer an answer in situations where there is incomparability and there is evidence of preferential reversal drawbacks in particular cases [21–23]. A study on a selection of off-the-shelf products [24] reported that the use of WAS and AHP gave different final results and that WAS gave less insight into the evaluation process.

The heuristic approach proposed by [25] is similar to the concordance part in the outranking methods (see [26] and Section 3) but is not sustained methodologically.

The subjective probability and the expected utility theories are used in [27]. This technique has a very strong axiomatisation [28] derived from a normative approach in decision making. Such an approach, however, has been strongly criticised in descriptive and prescriptive situations [29] as being inflexible and unrealistic.

Our fundamental criticism is not that such techniques should not be used, but that the choice of which one is used is arbitrary. The choice of one technique (instead of another) is not always justified and the validation of the technique (different techniques are not suitable for all kinds of problem) is not always discussed. In other words, the aggregation technique is a key variable of the evaluation model, and should be chosen to be consistent with the other components of the model.

1.1.4 Evaluation process: Evaluations involve different people, resources and activities. The need to define a precise evaluation process model in which the roles of people (decision maker, evaluator, user, client and producer) and their responsibilities, the activities, the temporal constraints among activities are described is not recognised in [4, 5, 16, 20].

However, an evaluation process, mainly from a managerial, organisational and legal point of view has been defined in [30, 2, 31] but it considers only the assessment and certification of a product.

There is a need for an evaluation process model which takes into account both the technical issues and a variety of possible evaluations such as the description of products, the ranking of products in a linear scale from the most preferred to the least preferred, the choice of one or more products as acceptable and the others as unacceptable.

1.2 Requirements for an evaluation methodology

In the authors' opinion, the central issue in software product evaluation is to adopt a general approach, so as to provide an evaluator with the conceptual tools
which will enable him to build reliable, robust and useful models of his problem. We propose the following requirements for an evaluation methodology.

**Decision process:** Many actors are involved in an evaluation, each one having different objectives and providing partial information [32]. Many activities are performed and a lot of information is produced. Therefore the evaluation process is not trivial, and should be clearly defined.

**Hierarchy of conflicting criteria:** An evaluation involves many criteria (also referred to as evaluation attributes, or quality attributes), usually organised in a hierarchy. In most evaluations, some criteria are in conflict with others (for example, quality and cost).

**Judgement:** An evaluation tries to find which product best fits a need. Since a need depends on people or organisations expressing it, an evaluation also depends on people or organisations. In other words, an evaluation is a subjective, qualitative process that involves judgement. In particular, this requires the use of aggregation techniques capable of handling judgement.

**Measures, preferences, uncertainty:** In an evaluation different types of information have to be considered: measures (an objective assessment), preferences (a subjective assessment) and uncertainty. These three information sources should not be confused (otherwise meaningless results may appear) and require specific aggregation techniques. For instance uncertainty can be aggregated using a ‘max’ operator (which is meaningless for the measures and preferences), measures can be aggregated by different kind of means (geometric and arithmetic ones, meaningless for both uncertainty and preferences), preferences can be aggregated using voting procedures (meaningless for measures and uncertainty).

**Aggregation technique:** The aggregation procedure is not neutral vis-à-vis the information provided (measures, preferences and uncertainty) and the desired result and is therefore part of the evaluation model. Thus it has to be chosen with the same care as the evaluation attributes and should be consistent with the whole evaluation model.

**Flexibility and consistency:** Each evaluation problem is different, with different criteria, different points of view, different purposes. An evaluation methodology should be flexible enough to address all these points. Conversely, an evaluation methodology should verify the internal consistency of the evaluation. In particular the aggregation technique should not be taken for granted, but chosen to be consistent with the other components of the evaluation.

Most of the above requirements are satisfied by the multicriteria decision aid (MCDA) approach [34–36]. The MCDA approach consists in a conceptual framework, unifying different aggregation techniques (such as multi-attribute value theory, outranking methods, AHP, etc.) under common concepts, while providing some general guidelines for choosing a specific technique which satisfies the following requirements: coherence of all components of an evaluation model to guarantee meaningful results; and suitability of the evaluation model to the decision maker’s needs to guarantee useful results. This paper presents IusWare, a methodology that adapts the MCDA approach to evaluations in the software product domain.

### 2 Method

The IusWare method (IUStitia SoftWARis, to be read as UseWare) will be described in the same terms as the entities which make up the software process, i.e. resources, products and activities.

The software product evaluation process (SPEP), is the set of resources, products and activities consumed, produced and performed between the perception of a software product evaluation problem and the adoption of a formally motivated final solution for the problem. The SPEP is equivalent to the concept of the decision-aid process used in the MCDA approach and involves resources, products and activities which are presented below.

#### 2.1 Resources

**Actor** refers to the people or organisation involved in the SPEP. An actor can play the role of either buyer, vendor, user, producer of a software product, or its evaluator (the party performing the evaluation). The client is the actor whose point of view is used to build the evaluation model and who will use it. The client and the evaluator are the core actors in the SPEP.

The commitment, knowledge and time which each actor can dedicate to the evaluation are also to be regarded as resources.

#### 2.2 Activities and products

In this Section the activities performed during a SPEP and their respective final products are introduced. Figs. 1–3 contain dataflow diagrams corresponding to the SPEP.

Three top level activities (see Fig. 1) can be identified: the problem formulation, the design of the evaluation model and the application of the evaluation model. The similarity between these activities and the classical decision-making model of Simon [36] should be noted. The top level activities will now be described.

![Fig.1 Top level activities and products of the SPEP](image)

**2.2.1 Problem formulation:** As soon as it is apparent that a formal decision aid is needed, a number of questions have to be raised and answered to guide the definition of the evaluation model.

The evaluator, on the basis of interviews with the client, defines a problem formulation $\Gamma' = (A, V, \Pi)$ where: $A$ is the set of software products on which the SPEP is focused (what will be evaluated?). Great care should be taken in defining $A$, since the client may consider subsets of a given $A$, either to test his ideas or as a result of undetermined discrimination. Analogously,
unexpressed wishes of the client should be checked for. $V$ is the set of points of view (what is the bias of the evaluation?). The points of view depend on the actors (who wants the evaluation?) involved in the SPEP. Typical points of view are those of the user, producer, producer-technician, producer-manager, maintainer and vendor. Normally the point of view of one or more actors is adopted.

$\Pi$ is a problem statement defining what the final result is expected to be (what is the purpose of the evaluation?). Possible values of $\Pi$ are

- **Choice** – partition the set products into a set of best product(s) and rest product(s)
- **Sorting** – partition the set of products following previously defined profiles of good, bad, etc.
- **Classification** – rank the products from the most preferred to the least preferred
- **Description** – provide a formal description of the products, without any ranking
- **Conceptualisation** – identify ideal or quasi-ideal products not available at the moment and possibly conceive new alternatives
- different combinations of the above statements

The first three problem statements are called operational.

### 2.2.2 Design of the evaluation model

In this activity (see Fig. 2) a detailed evaluation model is defined. The orders of precedence between activities are those shown in Fig. 2. The three main streams of action are the choice of $A'$, the definition of $D$, $E$, $M$, $G$ and the choice of $\mathcal{R}$.

![Decomposition of design evaluation model activity](image)

Choose $A'$: $A' \subseteq A$ is a subset of the set $A$ defined in the problem formulation. The elements of $A'$ are assumed to be independent, in the sense that the evaluation of one product should not influence the evaluation of the others. $A$ is purged to $A'$ to satisfy the independence constraint, to reduce the evaluation effort and to eventually to meet any mandatory requirement.

**Define $D$:** $D$ is a set of evaluation attributes. Typically they correspond to the quality attributes of quality models and are derived from the points of view in $V$. Attributes can be decomposed hierarchically. The first level of decomposition introduces subattributes, the second level sub-subattributes, etc. An attribute which is decomposed is called a composed attribute, an attribute which is not decomposed is called a basic attribute. The decomposition of an attribute into subattributes means that the attribute depends on the subattributes and that the dependency is qualitative. For instance, the attribute quality of product may depend on the subattributes functionality, operating system, modularity and diffusion.

**Define $M$:** $M$ is the set of measures associated to basic attributes. For each basic attribute, the evaluator chooses or defines a measure capable of characterising it. The evaluator can also decide, but should justify any decision, not to measure an attribute. A measured attribute is an attribute associated to a measure. An essential judgement is built upon such attributes. A non-measured attribute is an attribute which is not associated to a measure. Non-repeatable judgements are built upon such attributes. For instance, functionality could be measured in terms of the number of low level functionalities available, operating system by classifying the operating system, while modularity could be judged and diffusion could be measured by the number of licenses sold. Functionality, operating system and diffusion are measured attributes while modularity is not measured.

**Define $E$:** $E$ is the set of scales associated to $D$. The scales can be of any type, including nominal. $e(a)$ denotes the values obtained by each $a_i \in A'$ on each measure associated to a basic attribute $d_i$. For measured attributes the scale usually corresponds to the scale of the measure. For non-measured attributes the scale is declared by the client. For instance functionality and diffusion have scales $e_f, e_d$ of sets of positive integers, operating system has the scale $e_o = [\text{Unix}, \text{Dos}, \text{Vms}]$, modularity has the scale $e_m = [\text{high}, \text{medium}, \text{low}]$.

**Define $G$:** The goal of this activity is to define the rules to be used to transform measures into preferences, according to the client’s needs [37].

To formalise the concept of preference and the rules to compute it from measures, we introduce the key concepts of preference structure and criterion.

A preference structure is a set of preference relations characterised by their properties. In the simplest and most common case such a set collapses to a single binary relation $s$, built in different ways upon the basic relations $p$: strict preference, $i$: indifference and $r$: incomparability. For instance a common definition of $s$ is: $\forall x, y \in A' \ s(x, y) = p(x, y) \lor i(x, y)$. $s$ can have different properties such as completeness, ($a$)symmetry, transitivity, etc. depending on how it is defined. For instance if incomparability is empty, strict preference is transitive and indifference is not, then $s$ is a semi-order [38].

A criterion $g_p = (d_i, s)$ is an attribute equipped with a preference structure. If $s$ is a complete binary relation, then a numerical representation exists which is equivalent to it. In that case we introduce for each criterion a function $g_p : E_i \rightarrow R$ (where $E_i$ is the set of values of the scale $e_p$ which maps the attribute scale to the reals, satisfying the constraints imposed by the properties of the preference structure and the type of scale. $G$ is the set of all $g_p$, $g_p(a)$ denotes the value of product $a_i$ on criterion $g_p$.
For instance if $s_j$ is a semi-order, its admissible numerical representation is:

\[ \forall x, y \; p(x, y) \iff g(x) - g(y) \geq k \]
\[ \forall x, y \; \delta(x, y) \iff |g(x) - g(y)| < k \]

representing the threshold (constant) at which discrimination is difficult.

Assuming that \( \exists j \) \( s_j \) is a complete binary relation, the goal of this activity is to define \( G \). This activity and the activity \textit{Transform nominal scales} mark the first important transition from measurement to formalised essential judgement, that is to say the first important step in modelling the client's need for the evaluation.

Operationally the evaluator may face the following cases:

(i) Accept the order induced by the scale of the attribute as the preference structure; in this case \( \forall x \; g_j(x) = e_j(x) \)

(ii) Modify the numerical representation induced by the scale to take into account the specific nature of the client's preferences; any functional transformation (inverting, using logarithms, introducing thresholds, etc.) is possible and actually a scale transformation is carried out; in this case \( \forall x \; g(x) = g_j(e_j(x)) \)

(iii) Build \( s_j \) directly by performing pairwise comparisons of the elements in \( A' \), then deduce \( g_j \); this approach is typically used either when the client is not able to declare \( g_j \) explicitly or when \( s_j \) is not complete (in this case \( g_j \) does not exist). Measured attributes with nominal scale (for instance attribute \textit{operating system}) are a particular case. Then, either a direct pairwise comparison of the products may help in defining the associated criterion, or a preference structure is built among the elements of the nominal scale (for instance in the case of \textit{operating system} the user may declare that \( \text{Dos} \geq \text{Unix} = \text{Vms} \)). In this case the preference structure of the scale applies directly on \( A' \) (if \( e_j(a_1) = \text{Dos} \), \( e_j(a_2) = \text{Unix} \), then \( a_1 \) is preferred to \( a_2 \)).

In the two former cases the preference structure is defined implicitly through \( g_j \) (and is verified later, see activity \textit{Verify S}), while in the latter case the preference structure is defined and computed explicitly.

In our example the criteria could be defined as functionality: if the client prefers products with more functionality

\[ \forall x, y \in A' \; s_j(x, y) \iff g(x) \geq g(y) \text{ with } g_j = e_j \]

if the client prefers products with less functionality

\[ \forall x, y \in A' \; s_j(x, y) \iff g(x) \leq g(y) \text{ with } g_j = -e_j \]

if the client prefers products with more functionality, besides a threshold

\[ \forall x, y \in A' \; s_j(x, y) \iff g(x) \geq g(y) + 10 \text{ with } g_j = e_j \]

\textit{operating system: } \forall x, y \in A' \; s_j(x, y) \iff g(x) \geq g(y) \text{ with } g_j(\text{Vms}) = 0, g_j(\text{Unix}) = 0, g_j(\text{Dos}) = 1

\textit{modularity: } \forall x, y \in A' \; s_j(x, y) \iff g_m(x) \geq g_m(y) \text{ with } g_m(\text{high}) = 3, g_m(\text{medium}) = 2, g_m(\text{low}) = 1

\textit{diffusion: } \forall x, y \in A' \; s_j(x, y) \iff g_d(x) \geq g_d(y) \text{ with } g_d = \log(e_d) \text{ because the client wants to compare orders of magnitude of licences sold.}

Choose \( R \): \( R \) is an aggregation technique, described by an algorithm, capable of transforming the set of all \( s_j \) into a prescription for the client. A \textit{prescription} is an order on \( A' \). \( R \) can be used only with operational problem statements.

Usually an aggregation of the preferences of the
criteria at a certain level of decomposition results in a
global binary relation which may not be an order. An
\textit{exploiting procedure} transforms the global relation into
an order.

Four families of techniques are considered: first, multi-attribute value (utility) theory, a marginal value (utility) function is associated to each criterion and a
global value (utility) function is computed in an additive or multiplicative form [39]. WAS is a special case of the theory. Utility functions are especially suitable in
the stochastic case.

Secondly, interactive techniques, the final prescription is obtained through the interactive exploration of the set of nondominated solutions using the client’s preferences (e.g. multi-objective programing, goal programming) [40]. Thirdly, analytic hierarchy processes, the decisional goal is decomposed into a hierarchy of goals and ratio comparisons are performed on a fixed ratio scale. Then overall priorities are computed using an eigenvalue technique on the comparison matrix [19]. Finally, outranking techniques, a global preference relation is computed via direct aggregation of the preference structure and then exploited by the evaluator to compute the prescription. There are many aggregation and exploiting procedures and they enable the evaluator to tune the technique to the problem situation [35]. Unlike the other techniques, outranking techniques distinguish between classification and choice and solve the sorting problem statement.

The choice of \( R \) is constrained by (see [35] for a
deeper analysis)

(i) The type of problem statement \( \Pi \)

(ii) \( A' \) being a continuous or discrete set. Actually \( A' \)
in software evaluation is always a discrete set. However,
several aggregation techniques fit better when \( A' \)
is a continuous set (it is the case for several interactive
procedures)

(iii) The type of scales of criteria; in the example used
above the WAS technique cannot be used because two
criteria have ordinal scales, while WAS requires at least
ratio scales

(iv) The type of dependency among the criteria (see
also [41]). A basic condition to be satisfied is isollability.
A criterion \( g_k \) is isollable in \( G \) iff

\[ \forall x, y \in A' \; \forall j \in G \setminus \{g_k\} \; g_j(x) = g_j(y) \text{ and } g_k(x) \neq g_k(y) \]

where \( S \) is the global preference relation. \( S = P \lor I \)
with \( P \) global preference and \( I \) global indifference. \( G \)
satisfies the isollability condition if \( \forall k \in G \; g_k \) is isollable.
When \( G \) is nonredundant (see the later activity: verify
nonredundancy), then isollability is also satisfied.

A stronger condition is that of additive preferential
independence. Let \( J \) be a subset of \( G, J ' \) its complem-
ent. \( J \) is preferentially independent in \( G \)

\[ \forall x, y, w \; g(x) = g(y) \lor \forall j \in J \; g_j(z) = g_j(w) \]

then

\[ P(x, y) \iff P(z, w) \text{ and } I(x, y) \iff I(z, w) \]

where \( P \) is global preference and \( I \) is global indifference.

If \( \forall j \subseteq G \; J \) is preferentially independent in \( G \), then \( G \)
fulfils the additive preferential independence condition
[35]. The WAS technique cannot be used if \( G \) does not
fulfil the additive preferential independence condition.
(v) The client accepts or rejects compensation among criteria. Multi-attribute value techniques are the best choice if trade-offs among the criteria are to be used.

\[ \Gamma = \{ A, V, \Pi, A^*, D, E, M, G, \mathcal{R} \} \] is the main product of the Design evaluation model activity and is called the evaluation model.

### 2.2.3 Application of evaluation model:
In this phase (see Fig. 3) the evaluation model is applied and a prescription is obtained.

![Decomposition of apply evaluation model activity](image)

*Fig. 3 Decomposition of apply evaluation model activity*

**Measure/judge attributes:** for each basic measured attribute and for each software product in \( A^* \), a measurement is performed to obtain \( e(a) \). The client judges \( e(a) \) for each basic nonmeasured attribute and for each software product in \( A^* \).

**Apply scale transformations and criteria:** nominal scales are transformed and \( g(a) \) are computed.

**Derive:** for each \( j \in s \), is deduced by using the criteria.

**Verify:** the evaluator verifies, on relevant attributes, whether the preference structure deduced in this way actually models the client’s preferences and needs. In other words, since in real cases often preferences are deduced from existing measures, it is important to verify that the criteria construction respects the user’s preferences. A typical example is to verify the necessity to introduce a discrimination threshold. Often the discrimination problem appears only after the preference structure has been built up.

**Verify nonredundancy:** the nonredundancy of basic and composed criteria is verified by using two tests:

(i) \( \forall g_i \in G \) such that \( g(a) = g(b) \) then

\[ \forall c \quad P(c, b) \rightarrow P(c, a) \quad \text{and} \quad P(b, c) \rightarrow P(a, c) \]

\[ \forall c \quad R(c, b) \rightarrow R(c, a) \quad \text{and} \quad R(b, c) \rightarrow R(a, c) \]

(ii) \( \forall g_i \in G \setminus \{ g_1 \} \) such that \( g(d) = g(b) \) and \( g_i(d) \geq g_i(b) \)

and

\[ \forall g_i \in G \setminus \{ g_1 \} \] such that \( g(a) = g(c) \) and \( g_i(a) \geq g_i(c) \)

then

\[ P(b, a) \rightarrow P(d, c) \]

and

\[ R(b, a) \rightarrow R(d, c) \]

if a reduced set of criteria satisfies the two tests, then \( G \) is redundant.

Intuitively the two tests verify what happens if a criterion is eliminated from the set \( G \). The first test says that two products \( a, b \) which are equivalent in all criteria should behave globally in the same way when compared with any third product \( c \) (first coherence test). The second test says that if the product \( d \) dominates the product \( b \) and the product \( a \) dominates the product \( c \), then when \( b \) is strictly better than \( a \) then \( d \) should be strictly better than \( c \) and when \( a \) and \( b \) are indifferent then \( c \) could not be strictly better than \( d \) (second coherence test).

If after eliminating a criterion the two tests are no longer satisfied, then the set \( G \) is the minimal set of criteria that guarantees coherency. Conversely, if after eliminating a criterion the tests are still verified, then the set \( G \) is no more minimal and hence can be considered redundant.

**Apply aggregation technique:** \( \mathcal{R} \) is applied iteratively to \( s \) and a prescription is obtained. A prescription is an order on \( A^* \). In the case of nonoperational problem statements it corresponds to \( s \). In the case of operational problem statements, it is the result of the application of \( \mathcal{R} \) to \( s \). In particular

(a) If \( \Pi \) is a choice then it corresponds to the *kernel* of the global preference relation, i.e. the elements of \( A^* \) which jointly outperform all the others

(b) If \( \Pi \) is a sorting then it corresponds to a strict total order of equivalence classes where each class is a cluster of elements of \( A^* \); each cluster is built around an external profile (an imaginary ideal *good* product or *bad* product, etc.)

(c) If \( \Pi \) is a classification then it corresponds to a strict total order of equivalence classes where each class is built from products judged to be indifferent

\( \Gamma \) must not change during the application of the model. If this happens (i.e. \( A \) and \( A^* \) receive or lose elements, \( G \) changes) a new iteration of the design phase should be performed.

### 3 A case study

An account of an evaluation carried out with IusWare is now presented.

#### 3.1 The problem

A CASE tool to support the production of software in a CIM environment has to be selected from the many offered on the market. Basically two functionalities are offered by the products: first, to build a model of a production facility to simulate its behaviour and evaluate its performance; and secondly, to build a model of the software to be used in the production facility so that it can be generated.

CIM software typically has responsibility for production planning, production control and monitoring. The software is event intensive, has (soft) real-time constraints, is embedded and runs in a heterogeneous hardware and software environment.

The simulation model of a facility has to have the model of the software embedded in it, so that the two functionalities can be integrated. First, a model of the whole facility is made, including models of the manufacturing devices, and is used for simulation. Then the model is transformed, by substituting the models of the manufacturing devices with the actual software interfaces which link them, and the software embedded in the facility is generated.
3.2 Application of lusWare

The results of each activity of the SPEP are presented in the same order as in Section 2.

3.2.1 Problem formulation: The actors of the evaluation are the buyer, the user and the evaluator. The point of view of the evaluation is the one of the buyer and the one of the user, who have jointly decided on a common approach to the evaluation problem.

Three problem formulations, or \( A, V, \Pi \), were built up:

(i) \( \Gamma_1 \): a description of the products. \( A \) is the set of CASE products available on the market (more than 20 products). \( V \) coincides, for the user, with the software quality attributes listed in the ISO 9126 standard [2]: functionality, portability, efficiency, reliability, usability and maintainability. The buyer's point of view also includes a concern for the life of the purchase. \( \Pi \) is a description, i.e. a description of each product compared with each software quality attribute. \( \Gamma_1 \) is a cautious problem formulation with no aggregation techniques, therefore there is no operational prescription (which product to buy). It is indeed useful to help the client to clarify his ideas so that a more thorough investigation can be carried out if he thinks it is necessary.

(ii) \( \Gamma_2 \): the choice of product. \( A \) and \( V \) are the same as in \( \Gamma_1 \). \( \Pi \) is a choice, i.e. the identification of the set of products which jointly outperform all the others. \( \Gamma_2 \) is a more pressing problem formulation which encourages the client to adopt an immediate operational attitude. If the set of outperforming products includes more than one product it is necessary to carry out a more thorough analysis restricted to the products in that set.

(iii) \( \Gamma_3 \): rank the products. \( A \) and \( V \) are the same as in \( \Gamma_1 \) and \( \Gamma_2 \). \( \Pi \) is a classification, i.e. a complete ranking of \( A \) from the most preferred product to the least preferred one. The related evaluation model \( \Gamma_3 \) is presented later. \( \Gamma_3 \) is an intermediate problem formulation, which can help the client to understand the consequences of his preferences on the whole set of possible purchases and therefore enable him either to decide to carry out a further analysis or to make a purchase.

The difference between choice and classification is that with classification, all the products are ordered, possibly after building many subsets, from best to worst; while with choice, only two subsets are built, best and not best. It should be noted that, given the same initial parameters, the best set computed using classification can be different from the best set computed using choice.

3.2.2 Design of the evaluation model: The client's choice was \( \Gamma_3 \), because \( \Gamma_1 \) was not operational, and \( \Gamma_2 \) did not provide a complete ranking of the products. Activities and products which correspond to the design of the evaluation model for \( \Gamma_3 \) are now presented.

Choose \( A^* \): A preliminary screening of the set \( A \), based on features considered as absolutely necessary, reduced the set to seven products hereby denoted as \( A^* = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\} \).

Define \( D \): The set of points of view \( V \) has been transformed into four evaluation attributes: functionality (\( d_1 \)), usability (\( d_2 \)), portability (\( d_3 \)) and maturity (\( d_4 \)). These attributes are then decomposed into two hierarchical levels.

The decomposition hierarchy and the definition of attributes are as follows:

\( d_1 \) (functionality) is decomposed into the subattributes

\( d_{11} \) (editing): the possibility of editing the model
\( d_{12} \) (executing): the possibility of executing the model
\( d_{13} \) (data analysis): the possibility of analysing the data produced by executing the model
\( d_{14} \) (debugging): the possibility of debugging the model from inside the product
\( d_{15} \) (simulation): the possibility of simulating the model, decomposed into the sub-attributes

\( d_{151} \) (data collection): the collection of statistical data
\( d_{152} \) (statistical libraries): the availability of statistical libraries
\( d_{153} \) (data structure libraries): the availability of data structure libraries
\( d_{154} \) (graphical analysis): the possibility of analysing the data collected graphically
\( d_{155} \) (software generation): the possibility of generating software from the model, decomposed into the sub-attributes

\( d_{161} \) (field interfaces): the availability of interfaces to field devices
\( d_{162} \) (database interfaces): the availability of interfaces with commercial databases
\( d_{163} \) (graphical interfaces): the availability of graphical interfaces
\( d_{164} \) (data structure libraries): the availability of data structure libraries

\( d_2 \) (usability) is decomposed into the subattributes

\( d_{21} \) (usability of simulation language): the usability attributes of the language used to build the simulation model, decomposed into

\( d_{211} \) (modularity): language modularity
\( d_{212} \) (expressiveness): language expressiveness
\( d_{22} \) (usability of generation language): the usability attributes of the language used to build the software generation model, decomposed into

\( d_{221} \) (modularity): language modularity
\( d_{222} \) (expressiveness): language expressiveness
\( d_{23} \) (ease of learning): the facility to learn how to use the product
\( d_{24} \) (documentation): the quality of written and online documentation about the product
\( d_{25} \) (hot-line assistance): the existence of emergency assistance for the product
\( d_{26} \) (ergonomics): the ergonomics of the user interface

In building up the attributes \( d_{15}, d_{16}, d_{21}, d_{22} \) it was necessary to distinguish between features concerned with the simulation model and those concerned with the software generation model. Actually there were products which performed differently with regard to the same feature (for instance availability of libraries) when it concerned the two different application domains. Therefore \( d_{153} \) and \( d_{164}, d_{211}, d_{221}, d_{212}, d_{222} \) are not merged.

\( d_3 \) (portability) is decomposed into two subattributes

\( d_{31} \) (portability of the simulation or software generation model): decomposed into two sub-subattributes
Table 1: Scales for basic attributes, or $e_i$

<table>
<thead>
<tr>
<th>Attribute and criterion name</th>
<th>Scale for criterion</th>
<th>Attribute scale type</th>
<th>Criterion scale type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_{11}$ editing</td>
<td>yes &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{12}$ executing</td>
<td>yes &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{13}$ data analysis</td>
<td>good (g) &gt; normal (n) &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{14}$ debugging</td>
<td>integrated (l) &gt; half integrated (hl) &gt; external (e)</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{15}$ data collection</td>
<td>automatic (a) &gt; manual (m) &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{16}$ statistical libraries</td>
<td>yes &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{17}$ data structure libraries</td>
<td>large (l) &gt; medium (m) &gt; few (f) &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{18}$ graphical analysis</td>
<td>yes &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{19}$ field interfaces</td>
<td>many (n) &gt; medium (m) &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{20}$ db interfaces</td>
<td>many (n) &gt; medium (m) &gt; few (f) &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{21}$ graphical interfaces</td>
<td>many (n) &gt; few (f) &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{22}$ data structure libraries</td>
<td>many (n) &gt; medium (m) &gt; few (f) &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{23}$ modularity</td>
<td>high (h) &gt; medium (m) &gt; low (l)</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{24}$ expressiveness</td>
<td>high (h) &gt; medium (m) &gt; low (l)</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{25}$ modularity</td>
<td>high (h) &gt; medium (m) &gt; low (l)</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{26}$ expressiveness</td>
<td>high (h) &gt; medium (m) &gt; low (l)</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{27}$ ease of learning</td>
<td>good (g) &gt; medium (m) &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{28}$ documentation</td>
<td>good (g) &gt; medium (m) &gt; low (l)</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{29}$ hot-line assistance</td>
<td>good (g) &gt; medium (m) &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{30}$ ergonomics</td>
<td>good (g) &gt; medium (m) &gt; low (l)</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{31}$ platform</td>
<td>workstation and pc (wpc) &gt; workstation (w) = workstation and other (wo) &gt; pc &gt; other (o)</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{32}$ os</td>
<td>unix and dos and vms (udv) &gt; unix and dos (ud) &gt; unix and vms (uv) &gt; unix (u) = unix and other (uo) &gt; dos (d) &gt; vms (v) &gt; other (o)</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{33}$ platform</td>
<td>workstation and pc (wpc) &gt; workstation (w) = workstation and other (wo) &gt; pc &gt; other (o)</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{34}$ os</td>
<td>unix and dos and vms (udv) &gt; unix and dos (ud) &gt; unix and vms (uv) &gt; unix (u) = unix and other (uo) &gt; dos (d) &gt; vms (v) &gt; other (o)</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{35}$ ui</td>
<td>x and windows (xw) &gt; x = windows (w) &gt; other (o) &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{36}$ network protocol</td>
<td>tcp-ip (t) &gt; other (o) &gt; no</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
<tr>
<td>$d_{37}$ diffusion</td>
<td>reals</td>
<td>absolute</td>
<td>ratio</td>
</tr>
<tr>
<td>$d_{38}$ geographical diffusion</td>
<td>USA and Europe (UE) &gt; USA (U) &gt; Europe (E) &gt; Italy (I)</td>
<td>nominal</td>
<td>ordinal</td>
</tr>
</tbody>
</table>

$d_{31}$ (platform): the hardware platform necessary to operate the model

$d_{32}$ (operating system): the operating system necessary to operate the model

$d_{33}$ (portability of the software generated by the software generation model): decomposed into four sub-subattributes

$d_{34}$ (platform): hardware platform

$d_{35}$ (operating system): operating system

$d_{36}$ (user interfaces): supported user interfaces

$d_{37}$ (network protocol): supported network protocols for distributed software

As in attribute $d_{2}$, it is necessary to distinguish between simulation and software generation (there are products in which the platform and the operating system are not the same for the two cases, therefore $d_{31}$ and $d_{32}$, $d_{33}$ and $d_{34}$ are not merged).

$d_{4}$ (maturity) is decomposed into two subattributes

$d_{41}$ (diffusion): the number of installed licences worldwide

$d_{42}$ (geographical diffusion): the geographical areas of major diffusion of the product

Define $M$, $E$: All basic attributes, except $d_{33}$, $d_{42}$, $d_{31}$, $d_{22}$, $d_{23}$, $d_{24}$, $d_{25}$ and $d_{26}$ are measured attributes. The scales for the associated measures are of a nominal type, except for $d_{41}$ which has an absolute scale. The values for such measures can be seen in Table 1, they are the values in column scale, as long as the symbol $>$ is ignored. The value no indicates that a product lacks the feature measured. Nonmeasured attributes all have nominal scales, these values can also be seen in Table 1.

Define $G$: Table 1 shows the values of the scale for each basic criterion as well as their ordering (with |) to indicate an abbreviation for the value, and and to indicate a combination of values). For instance when attribute $d_{41}$ was considered the scale is yes greater than no, showing that the client prefers products capable of editing to the ones which are not capable of supporting this function.

The value no is always considered to be the worst value. This is an important assumption and will be discussed in Section 3.3.

The relation $s_j$ is defined as $s_j = p_j \cup i_j$ for every basic and composed attribute, (with $p = \text{strict preference}$ and $i = \text{indifference}$).

The mappings $g_j$ are of the form $[3, 2, 1]$ when the scale is of the form $[a > b > c]$. In the case of $d_{41}$ a
transformation \( g_{41} = \log(e_{41}) \) is defined because the client wanted to compare the orders of magnitude of the licenses sold.

The criteria for each attribute \( d_i \) except \( d_{41} \), are of the form \( \forall x, y \in A' \ s_3(x, y) \Leftrightarrow g(x) \geq g(y) \). The criterion for \( d_{41} \) is \( \forall x, y \in A' \ s_4(x, y) \Leftrightarrow g_4(x) \geq g_4(y) \land g_4(x) \neq 10 \).

Choose \( R \): The constraints to be considered in choosing \( R \) are

(i) The problem statement is a classification
(ii) \( A' \) is discrete
(iii) The scales of criteria are of an ordinal and absolute type
(iv) The client is not able to indicate trade-offs between the criteria, therefore compensation is not admissible
(v) The criteria are nonredundant (see later Verify nonredundancy) and therefore satisfy the isolability condition.

An outranking aggregation technique has been adopted for this step and the one presented in the ELECTRE techniques [26] has been chosen because it satisfies the constraints listed above. The AHP technique has been discarded because the scales are not of a ratio type. Multi-attribute utility theory (and therefore WAS) has not been used because the scales of most criteria are ordinal and because it was not possible to establish trade-offs between criteria.

The basic concept of the chosen \( R \) is the outranking relation or \( S \), which has to be computed between each pair of products of \( A' \) and should be read as is at least as good as. The outranking relation holds if the concordance and nondiscordance tests are satisfied.

The concordance test is the majority strength to be reached to be able to establish with a certain degree of confidence the outranking relation. Such a majority is generally computed using the relative importance of each criterion.

The nondiscordance test is the minority strength below which it is possible to establish the outranking relation. Such a minority is generally computed using the relative importance of each criterion.

In general, \( S \) is not complete and transitive and to overcome this problem an exploiting procedure is introduced to reduce the global outranking relation to a final complete ranking (this binary relation is denoted by \( \geq \); and \( > \) represents strict preference and \( = \) represents indifference).

More formally, and assuming that the criteria are equally important, as in the case study

\[ S(x, y) \Leftrightarrow C_6(x, y) \land \neg D_6(x, y) \]

or: \( x \) is at least as good as \( y \) if the ordered pair \((x, y)\) satisfies the concordance test and satisfies the nondiscordance test. Where

\[ C_6(x, y) \Leftrightarrow \lvert G_6 \rvert \geq c \land \lvert G_6' \rvert \geq \lvert G_6' \rvert \]

or: \((x, y)\) satisfies the concordance test iff the number of criteria for which \( x \) is at least as good as \( y \) is not inferior to a certain number of criteria \( c \) (majority

<table>
<thead>
<tr>
<th>criterion</th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( a_3 )</th>
<th>( a_4 )</th>
<th>( a_5 )</th>
<th>( a_6 )</th>
<th>( a_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>editing</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>executing</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>data analysis</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>n</td>
<td>n</td>
<td>g</td>
<td>g</td>
</tr>
<tr>
<td>debugging</td>
<td>hi</td>
<td>hi</td>
<td>hi</td>
<td>i</td>
<td>hi</td>
<td>i</td>
<td>i</td>
</tr>
<tr>
<td>data collection</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>a</td>
<td>m</td>
</tr>
<tr>
<td>statistical libraries</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>data structure libraries</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>l</td>
<td>d</td>
<td>l</td>
</tr>
<tr>
<td>graphical analysis</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>field interfaces</td>
<td>d</td>
<td>n</td>
<td>no</td>
<td>d</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>db interfaces</td>
<td>f</td>
<td>d</td>
<td>no</td>
<td>f</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>graphical interfaces</td>
<td>no</td>
<td>n</td>
<td>no</td>
<td>f</td>
<td>f</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>data structure libraries</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>n</td>
<td>d</td>
<td>n</td>
</tr>
<tr>
<td>modularity</td>
<td>no</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>no</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>expressiveness</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>l</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>modularity</td>
<td>no</td>
<td>d</td>
<td>d</td>
<td>no</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>expressiveness</td>
<td>h</td>
<td>h</td>
<td>h</td>
<td>d</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>ease of learning</td>
<td>d</td>
<td>g</td>
<td>g</td>
<td>g</td>
<td>no</td>
<td>no</td>
<td>g</td>
</tr>
<tr>
<td>documentation</td>
<td>d</td>
<td>g</td>
<td>g</td>
<td>g</td>
<td>l</td>
<td>g</td>
<td>g</td>
</tr>
<tr>
<td>hot-line assistance</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>ergonomics</td>
<td>d</td>
<td>d</td>
<td>g</td>
<td>g</td>
<td>g</td>
<td>l</td>
<td>g</td>
</tr>
<tr>
<td>platform</td>
<td>wo</td>
<td>pc</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>pc</td>
<td>wpc</td>
</tr>
<tr>
<td>os</td>
<td>uv</td>
<td>u</td>
<td>uo</td>
<td>u</td>
<td>u</td>
<td>udv</td>
<td>ud</td>
</tr>
<tr>
<td>platform</td>
<td>w</td>
<td>pc</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>o</td>
<td>wpc</td>
</tr>
<tr>
<td>os</td>
<td>uv</td>
<td>u</td>
<td>u</td>
<td>uv</td>
<td>u</td>
<td>o</td>
<td>d</td>
</tr>
<tr>
<td>ui</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>x</td>
<td>x</td>
<td>no</td>
<td>o</td>
</tr>
<tr>
<td>network protocol</td>
<td>t</td>
<td>o</td>
<td>o</td>
<td>no</td>
<td>t</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>diffusion</td>
<td>50</td>
<td>500</td>
<td>50</td>
<td>500</td>
<td>5</td>
<td>500</td>
<td>50</td>
</tr>
<tr>
<td>geographical diffusion</td>
<td>1</td>
<td>UE</td>
<td>U</td>
<td>UE</td>
<td>I</td>
<td>UE</td>
<td>E</td>
</tr>
</tbody>
</table>
Table 3: Preference structure, or s

<table>
<thead>
<tr>
<th>d11</th>
<th>editing</th>
<th>d1 = d2 = d3 = d4 = d5 = d7 &gt; d6</th>
</tr>
</thead>
<tbody>
<tr>
<td>d12</td>
<td>executing</td>
<td>a1 = a5 = a6 = a7 = a8 = a9</td>
</tr>
<tr>
<td>d13</td>
<td>data analysis</td>
<td>a9 &gt; a1 &gt; a3 &gt; a8 &gt; a7 &gt; a4 &gt; a2 = a5</td>
</tr>
<tr>
<td>d14</td>
<td>debugging</td>
<td>a1 = a2 = a3 = a4 = a5 = a6</td>
</tr>
<tr>
<td>d15</td>
<td>data collection</td>
<td>a1 &gt; a2 &gt; a3 &gt; a5 = a6 = a7</td>
</tr>
<tr>
<td>d16</td>
<td>statistical libraries</td>
<td>a1 = a2 = a3 = a4 = a6</td>
</tr>
<tr>
<td>d17</td>
<td>data structure libraries</td>
<td>a1 = a2 &gt; a3 &gt; a4 = a6 = a7</td>
</tr>
<tr>
<td>d18</td>
<td>graphical analysis</td>
<td>a1 = a2 = a3 = a4 &gt; a7</td>
</tr>
<tr>
<td>d19</td>
<td>field interfaces</td>
<td>a1 = a2 = a3 &gt; a7</td>
</tr>
<tr>
<td>d20</td>
<td>db interfaces</td>
<td>a1 = a2 = a3 = a4 = a6</td>
</tr>
<tr>
<td>d21</td>
<td>graphical interfaces</td>
<td>a1 = a2 = a3 = a7</td>
</tr>
<tr>
<td>d22</td>
<td>data structure libraries</td>
<td>a1 = a2 = a3 = a7</td>
</tr>
<tr>
<td>d23</td>
<td>expressiveness</td>
<td>a1 = a2 = a3 &gt; a7</td>
</tr>
<tr>
<td>d24</td>
<td>expressiveness</td>
<td>a1 = a2 = a3 &gt; a7</td>
</tr>
<tr>
<td>d25</td>
<td>expressiveness</td>
<td>a1 = a2 = a3 &gt; a7</td>
</tr>
<tr>
<td>d26</td>
<td>expressiveness</td>
<td>a1 = a2 = a3 &gt; a7</td>
</tr>
<tr>
<td>d27</td>
<td>expressiveness</td>
<td>a1 = a2 = a3 &gt; a7</td>
</tr>
<tr>
<td>d28</td>
<td>expressiveness</td>
<td>a1 = a2 = a3 &gt; a7</td>
</tr>
<tr>
<td>d29</td>
<td>expressiveness</td>
<td>a1 = a2 = a3 &gt; a7</td>
</tr>
<tr>
<td>d30</td>
<td>expressiveness</td>
<td>a1 = a2 = a3 &gt; a7</td>
</tr>
<tr>
<td>d31</td>
<td>expressiveness</td>
<td>a1 = a2 = a3 &gt; a7</td>
</tr>
<tr>
<td>d32</td>
<td>expressiveness</td>
<td>a1 = a2 = a3 &gt; a7</td>
</tr>
<tr>
<td>d33</td>
<td>expressiveness</td>
<td>a1 = a2 = a3 &gt; a7</td>
</tr>
<tr>
<td>d34</td>
<td>network protocol</td>
<td>a1 = a2 = a3 = a7</td>
</tr>
<tr>
<td>d35</td>
<td>diffusion</td>
<td>a1 = a2 = a3 = a7</td>
</tr>
<tr>
<td>d36</td>
<td>geographical diffusion</td>
<td>a1 = a2 = a3 = a7</td>
</tr>
</tbody>
</table>

in other words the score of x in S' is the number of products outranked by x minus the number of products outranking x

(4) Rank the elements of A'/C on the basis of their score. More formally

\[ x > y \iff f(x, S') > f(y, S') \]

\[ x = y \iff f(x, S') = f(y, S') \]

3.2.3 Application of the evaluation model:

Measure/judge attributes: Table 2 contains the values for basic attributes, or e(a).

Apply scale transformations and criteria: Basically Table 2 also shows g(x), that coincide with e(a) except for g11 whose values are 1.69, 2.69, 1.69, 2.69, 0.69, 2.69, 1.69.

Derive s: The preference structure, deduced by g(a), is shown in Table 3.

Verify s: The client accepted s as a suitable representation of her preferences.

Verify nonredundancy: The nonredundancy of the criteria has been verified using the tests introduced in Section 2.2.3.

Apply aggregation technique: The prescription is computed by applying R to s repeatedly. Since the client was not able to define the relative importance of composed criteria, each composed criterion is assumed to have the same importance. No vetos have been expressed by the client. First R is applied to the preference structures associated to the sub-attributes. The unanimity rule means \( c = |G_j| \), where \( j \) identifies a decomposition level. The result is shown in Table 4. R

Table 4

<table>
<thead>
<tr>
<th>Aggregation of</th>
<th>Ranking</th>
<th>Aggregation parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>d15, d32 to d16</td>
<td>a5 = a6 = a7 = a8 = a9 = a10 = a11 = a12 = c = 3, d = 2</td>
<td></td>
</tr>
<tr>
<td>d16, d32 to d14</td>
<td>a5 = a6 = a7 = a8 = a9 = a10 = a11 = a12 = c = 3, d = 2</td>
<td></td>
</tr>
<tr>
<td>d13, d32, d31</td>
<td>a1 = a2 = a3 = a4 = a5 = a6 = a7 = a8 = a9 = a10 = a11 = a12 = a13 = c = 3, d = 2</td>
<td></td>
</tr>
<tr>
<td>d32, d31, d30</td>
<td>a1 = a2 = a3 = a4 = a5 = a6 = a7 = a8 = a9 = a10 = a11 = a12 = a13 = c = 3, d = 2</td>
<td></td>
</tr>
<tr>
<td>d31, d32, d30</td>
<td>a1 = a2 = a3 = a4 = a5 = a6 = a7 = a8 = a9 = a10 = a11 = a12 = a13 = c = 3, d = 2</td>
<td></td>
</tr>
</tbody>
</table>

is then applied to the preferences associated to the sub-attributes with the result shown in Table 5.

Table 5

<table>
<thead>
<tr>
<th>Aggregation of</th>
<th>Ranking</th>
<th>Aggregation parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1, d11 to d16</td>
<td>a1 &gt; a2 &gt; a3 = a4 = a5 &gt; a6 = a7 = a8 = a9 = a10 = a11 = a12 = a13 = c = 5, d = 2</td>
<td></td>
</tr>
<tr>
<td>d11, d12 to d30</td>
<td>a1 &gt; a2 &gt; a3 = a4 = a5 &gt; a6 = a7 = a8 = a9 = a10 = a11 = a12 = a13 = c = 5, d = 2</td>
<td></td>
</tr>
<tr>
<td>d31, d32, d30</td>
<td>a1 = a2 = a3 = a4 = a5 = a6 = a7 = a8 = a9 = a10 = a11 = a12 = a13 = c = 3, d = 2</td>
<td></td>
</tr>
<tr>
<td>d32, d31, d30</td>
<td>a1 = a2 = a3 = a4 = a5 = a6 = a7 = a8 = a9 = a10 = a11 = a12 = a13 = c = 3, d = 2</td>
<td></td>
</tr>
<tr>
<td>d31, d32, d30</td>
<td>a1 = a2 = a3 = a4 = a5 = a6 = a7 = a8 = a9 = a10 = a11 = a12 = a13 = c = 3, d = 2</td>
<td></td>
</tr>
</tbody>
</table>

Finally R is applied to the attributes giving the final ranking, or prescription, as shown in Table 6.

Table 6

<table>
<thead>
<tr>
<th>Aggregation of</th>
<th>Ranking</th>
<th>Aggregation parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1, d11 to d12</td>
<td>a1 &gt; a2 &gt; a3 = a4 = a5 &gt; a6 = a7 = a8 = a9 = a10 = a11 = a12 = a13 = c = 5, d = 2</td>
<td></td>
</tr>
<tr>
<td>d11, d12 to d30</td>
<td>a1 &gt; a2 &gt; a3 = a4 = a5 &gt; a6 = a7 = a8 = a9 = a10 = a11 = a12 = a13 = c = 5, d = 2</td>
<td></td>
</tr>
<tr>
<td>d31, d32, d30</td>
<td>a1 = a2 = a3 = a4 = a5 = a6 = a7 = a8 = a9 = a10 = a11 = a12 = a13 = c = 3, d = 2</td>
<td></td>
</tr>
<tr>
<td>d32, d31, d30</td>
<td>a1 = a2 = a3 = a4 = a5 = a6 = a7 = a8 = a9 = a10 = a11 = a12 = a13 = c = 3, d = 2</td>
<td></td>
</tr>
</tbody>
</table>

3.3 Discussion

The discussion is centred on the most critical points of the SPEP: identification of attributes, decomposition of attributes, selection of metrics, trade-offs between judgement, metrics and decomposition.

3.3.1 Identification of attributes: The attributes $d_1$ to $d_4$ are related exclusively to external attributes. The reasons for this choice are many; first it is difficult to obtain information from the producers about intermediate products (source code, design documents, results of verification and validation activities), then there are constraints on the budget and the time allocated to the evaluation and finally the lack of safety critical issues.

The attributes were chosen from the software quality characteristics (SQc) defined by ISO 9126 [2] and the client considered only functionality, usability and portability as relevant, while maintainability, efficiency and reliability were not considered to be relevant. Maturity, which is not present in ISO 9126, was added to take into account the buyer’s point of view.

The maintainability of the CASE tool was not relevant for the user while efficiency and reliability were, but they were discarded for two reasons, there was neither sufficient time nor resources to evaluate them, and these two attributes are much less important than other SQCs.

Maintainability, efficiency and reliability are relevant attributes for software operating in a production facility.

If a CASE tool produces this software, its efficiency and reliability depend both on the CASE tool (especially on the software generator which is contained in it) and on the model of the software defined by the software engineer. The client decided that it was not relevant to evaluate the software generator which might be present in the CASE tools.

Maintainability depends both on the model and on the formalism used to define it, but not on the CASE tool, thus it was not evaluated.

The attribute maturity, defined as the risk of a product disappearing from the market, is not a technical attribute but has been added at the request of the buyer involved in the selection.

3.3.2 Decomposition of attributes: The decomposition of attributes was a long and hard process, probably the hardest and most intellectually intensive part of the evaluation process.

One reason for this is that decomposition is another way of drawing up the user’s requirements document which either does not exist (as in this case study) or is incomplete, ambiguous, contradictory, redundant and unstable. The decomposition process shares all the difficulties of specifying the user’s requirements.

Another reason is the fact that tool evaluation and software process evaluation cannot be easily separated. The CASE tools considered in the evaluation are built upon a number of assumptions regarding the software production process. First the software life cycle (either prototyping, where the production facility is simulated, then the actual control software is rewritten, or incremental refinement, where the production facility is simulated, then the actual control software is automatically generated); secondly, the high level formalism used to define the model of the production facility (either production rules, or Petri nets, or finite state automata) and finally, the low level formalism (C, C++, Smalltalk).

Since the client had not yet made a choice regarding these issues, the selection was a selection of CASE tools and software processes, and this made the evaluation harder. On the other hand, given the constraint that CASE tools be bought on the market and not developed in house, it would be illogical to choose a software process which cannot be supported by available tools.

A further reason is the distance between attributes and basic attributes: given a standard set of attributes (such as ISO 9126), virtually an infinite number of sets of basic attributes can be defined. On the one hand there is no solution to this problem, since an evaluation of quality over external attributes depends, by the definition of external attribute, not only on the product but on the product and its environment (needs, skill, experience and software process of the user, hardware and software platform, etc.). So every evaluation is different and has to be customised to the user’s needs. On the other hand, effort should be made to define and standardise the frameworks of those attributes adapted to specific problem domains (in line with [3, 6, 42]) so as to provide a starting point for the customisation process.

3.3.3 Quality measures selection: The number of nonmeasured attributes used in the evaluation is not negligible.

The use of nonmeasured attributes can be classified under two headings: attributes in which no established measures exist (e.g. $d_{121}$ expressiveness, $d_{13}$ ease of learning, $d_{16}$ ergonomics); and attributes in which measures exist, but where further analysis of the domain and of the client’s needs is required (e.g. $d_{153}$ and $d_{194}$ data structure libraries, $d_{163}$ field interfaces, $d_{182}$ db interfaces, $d_{163}$ graphical interfaces, should be improved by identifying which data structure or interface is really needed by the user instead of just counting how many are offered).

The second class was accepted for attributes for which the client decided not to use evaluation resources.

3.3.4 Decomposition, measurement and judgement: There is dependency between the use of judgement or measurement and decomposition of attributes.

In some cases further decomposition of a nonmeasured attribute facilitates the use of measurements at the lower level. For instance $d_{11}$ has the scale [large, medium, no] and no associated measure; if it were decomposed into, say, $d_{111}$ availability of abstract data types, with scale [yes, no], and $d_{112}$ support to encapsulation, with scale [yes, no], then measurement could be used. However, whatever approach is adopted, judgement, at least partially, determines the decomposition.

Another attribute in which judgement strongly influences the decomposition, even if it is associated only to measures, is $d_4$: de facto it penalises in the evaluation those products which have just been launched on the market, because very few licenses will have been sold.

$d_{24}$ is a basic attribute, while a full document [42] decomposes it. This demonstrates that judgement is
also involved in the decision about where to stop the decomposition.

In other cases a value of a measure could be refined. For instance the value yes for \( d_{152} \) could be refined with the new values random number generation and statistical data analysis. Alternatively, the attribute could be refined into two subattributes \( d_{1521} \) random number generation and \( d_{1522} \) statistical data analysis both with a scale [yes, no].

The relationship between judgement, measurement and decomposition can be summarised in the following statements: first, the depth of decomposition of attributes can vary. It depends on needs, time, budget and information available. Secondly, the level of judgement remains the same, distributed between judgement regarding nonmeasured attributes and decomposition decisions. Finally, as more detail is added through increasing levels of decomposition, the element of judgement intrinsic in an evaluation is more apparent and formalised. In other words it is better to substitute judgement in nonmeasured attributes with further decomposition and measures.

3.3.5 Aggregation technique: First, it is necessary to emphasise that the choice of the specific aggregation procedure used in the case study does not imply a definite methodological choice. The number of aggregation procedures that are available in MCDA is extremely large. Any technique can be used to aggregate preferences provided that the information about the problem is used correctly and that the client’s requirements are satisfied in accordance with the decision process being used. What is claimed is that the arbitrary use of aggregation techniques should be avoided since they may produce meaningless results.

Some specific comments can be made about the case study presented here. In many criteria (e.g. \( d_{13}, d_{151} \)) the value no on the attribute scale represents the absence of a feature. Whenever this case occurred, the value no was considered to be the worst one on the criterion scale. In fact the absence of a feature should be represented with the incomparability of no with all the other values, but this violates the complete comparability constraint often imposed by the wish to have a numerical representation. The same situation occurs when preferences on basic criteria are aggregated, since the aggregation produces, as a general rule, a partial order. Aggregation techniques that enable the direct aggregation of partial orders should be used, but at the moment they are at an experimental state [43, 44].

4 Conclusion

A method to evaluate and select software products has been presented. The method satisfies the requirements, stated in Section 1.2, to assure robust and reliable evaluations, and has the following advantages:

The decision process is clearly defined, in terms of activities performed, and information flow. Each item of information corresponds to a choice, and the client and the evaluator are aware of which choices are made in which activity. The client can always understand the detailed motivation of the conclusions of each activity of the process and detect hidden decisions, if any. Essential issues that deeply influence an evaluation (which actors are involved in the evaluation, their points of view, the purpose of the evaluation) are addressed at the very beginning of the process.

An evaluation model is composed of a hierarchy of evaluation attributes. Products are independently evaluated on each basic attribute. The evaluator is asked to define, in a way appropriate to the aggregation technique, the relative importance of attributes. If possible, conflicts are solved by the aggregation technique, otherwise they are signalled to the client and discussed until they are solved.

IusWare assumes that judgement is an essential part of an evaluation. Instead of trying to hide it, IusWare identifies the points where judgement is involved: the decomposition of attributes, the use of nonmeasured attributes, the transformation of nominal scales of measures, the definition of criteria. The client is asked to openly declare, discuss and justify these points. Moreover, IusWare provides tools, such as preferences and aggregation techniques, to handle judgement properly.

IusWare distinguishes between objective assessments of basic attributes (measures) and subjective ratings (preferences). Preferences are needed in an evaluation since they formalise how well a product fits a need, while measures are objective and independent of any need. The method suggests that a start should be made with measures, which should then be transformed into preferences. If measures are not available, it is possible to start directly with preferences. Preferences are aggregated with specific techniques. Preferences involve pairwise comparisons of products: in many cases the client finds it easier to compare products with regard to an attribute rather than to assign absolute ratings.

To provide flexibility, an evaluation model can be created with many degrees of freedom: problem statement (classification, ranking, choice, to be chosen with regard to the purpose of the evaluation); hierarchy of evaluation attributes (to be chosen with regard to the actors in the evaluation and their points of view, starting from the ISO 9126 framework); aggregation technique (to be chosen with regard to the problem statement, hierarchy and scales of attributes).

The whole process is client driven. Attributes, scales, aggregation techniques are not imposed by the method but decided by the client who receives suggestions regarding more technical matters from the evaluator. If the client is not sure that a decision adopted as a prescription will ensure the required result, the process can be backtracked and any critical options can be changed.

Building the evaluation model is a creative activity, that uses the degrees of freedom given above. IusWare provides this freedom, but also provides verification and validation activities to check for the internal and external consistency of the evaluation model and its components. Internally, the evaluation model is verified for the independence of evaluated objects, nonredundancy and preferential independence of criteria, constraints within the aggregation technique and the other components of the evaluation model. Externally, the evaluation model is validated to check its suitability for the client’s needs.

The aggregation technique is a key variable of the evaluation model and is chosen so that it is coherent with the other components. In other words, no single aggregation technique, such as WAS, is considered as suitable for any evaluation model. A verification activity is defined to check the type of dependency among
criteria (such as isolability and preferential independence) and therefore to choose a suitable aggregation technique.

5 Acknowledgments

The authors would like to thank Norman Fenton for his kind help and suggestions, and an anonymous referee for making several useful remarks.

6 References

3 ‘1209 A recommended practice for the evaluation and selection of CASE tools’, IEEE, New York, 1992

34 ‘Rethinking the process of operational research and systems analysis’ (Pergamon Press, Oxford, 1964), pp. 169–186
35 Roy, B.: ‘Methodologie multicieter d’aide à la décision’ (Econolica, Paris, 1985)