CAHIER DU LAMSADE

Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision
(Université Paris-Dauphine)
Unité de Recherche Associée au CNRS ESA 7024

MULTICRITERIA METHODOLOGY CONTRIBUTION TO THE SOFTWARE QUALITY EVALUATION

CAHIER N° 155
mai 1998

Marie-José BLIN [1]
Alexis TSOUKIÀS [1]

reçu: janvier 1998.

[1] *LAMSADE, Université Paris-Dauphine, Place du Maréchal De Lattre de Tassigny, 75775 Paris Cedex 16, France (e-mail: {blin,tsoukias}@lamsade.dauphine.fr).*

CONTENTS

Expériences d'utilisation de la méthodologie multicritère pour évaluer la qualité des logiciels

Résumé

Les modèles d'évaluation de la qualité des logiciels possèdent des caractéristiques propre: une structure hiérarchique induisant l'utilisation de méthodes d'agrégation pour calculer des évaluations à différents niveaux d'abstraction, l'aspect prédictif des évaluations et le mélange de mesures de différentes natures et de préférences nécessitant des procédures d'agrégation spécifiques. Par ailleurs, ils sont formalisés par des normes internationales utilisées couramment par les responsables d'évaluations.

Ces modèles présentent des difficultés d'application et ne sont pas entièrement satisfaisants. L'utilisation de la méthodologie multicritère offre une alternative. Les résultats de différentes expériences menées sur des cas réels sont exposés et une discussion critique est proposée.

Mots clés: qualité logiciel, méthodologie multicritère, évaluation de la qualité, modèles de qualité, processus d'évaluation, méthodes d'agrégation, mesures et préférences

Multicriteria Methodology Contribution to the Software Quality Evaluation

Abstract

The quality software evaluation models present some specific characteristics: a hierarchical structure and the use of aggregation methods to calculate the evaluations at different abstraction levels, the presume aspect of the evaluations and the mixture of measures of different types and of preferences requiring suitable aggregation procedures. In addition, they are formalized by international standards largely used by practitioners as guidelines for the evaluations.

In reality, such models are difficult to apply for concrete evaluations and are not completely satisfactory. The use of the multicriteria methodology allows to solve some of the problems these models present. Such a methodology was experimented on several real cases already processed with the traditional method. The results are presented in this paper and a critical discussion is proposed

Key words: software quality, multicriteria methodology, quality evaluation, quality model, evaluation process, aggregation methods, measures and preferences

Introduction

Software quality is one of the most important enterprises' challenges of 2000's. A great number of standards about software quality have been edited by ISO [ISO 8402, 1994], [ISO 9000, 1987], [ISO 9000-3, 1991], [ISO 9001, 1994], [ISO 9126, 1991], [ISO 12119, 1994] for several years. In ISO standard about definitions (ISO 8402), the quality of software is defined as the set of properties and characteristics which define the capability of the software to satisfy expressed or implicit needs. Such characteristics deal with the product's functions and the product's non functional qualities and constraints (like costs and availability).

In fact, two important concepts are hidden under the term "quality": insurance of quality and evaluation of quality. The quality insurance is the set of pre-established and systematically executed activities for insuring that an entity will fulfill the quality requirements. The quality insurance is a set of precautionary measures.

The evaluation of quality is a set of attributes and measurement methods for describing and measuring the quality of a software. Unlike insurance quality, the evaluation of the quality is realized afterwards.

During the development of a specific software, both quality insurance and quality evaluation can be used: quality insurance as precautionary measures and quality evaluation as control.

In this paper, we are only interested by the quality evaluation of "commercial off the shelf" software (COTS software) [Kontio, 1996]. This process is defined by ISO 9126 and IEEE 1061 [IEEE, 1992] standards which propose to define the quality as a set of attributes, organized in a tree in which each attribute has a weight and the weighted sum is used to aggregate the measures. We studied three existing industrial cases of software evaluation calculated in accordance with the standards and, faced with the difficulties to really exploit the results, we experimented the use of multicriteria methodology. This work allows us to understand some of the problems generated by the application of the standards, to propose new principles for evaluating software quality and to suggest future research for adapting multicriteria methodology to software quality evaluation.

Section 1 of the paper discuss two ways to evaluate a software, the-pre and the post-evaluation, the concerned ISO standards and the software industry challenges about quality. Section 2 presents the basic principles of quality evaluation developed in the standards and the difficulties to apply the standards to concrete evaluations of COTS software. Section 3 describes three quality evaluation existing cases already processed in accordance with ISO 9126 standard and explains how we applied the multicriteria methodology. Section 4 comments the experiment. Section 5 presents some significant work in the software quality evaluation area and section 6 concludes.

1 Evaluation of software quality

1.1 Two types of evaluation

A software can be evaluated in two ways. It can be pre-evaluated (the software itself does not exist or it is not available, at least in an executable version), or it can be post-evaluated (running version of the software is available besides working experience)

A pre-evaluation is calculated from a model of quality which: 1^o) defines a set of measures like number of different screens, number of data in a screen, number of global variables used in the code, number of code modules or existence of specific functions, 2^o) establishes relationships between measures and some global characteristics of the software like maintainability or usability. A post-evaluation is made by measuring some interesting

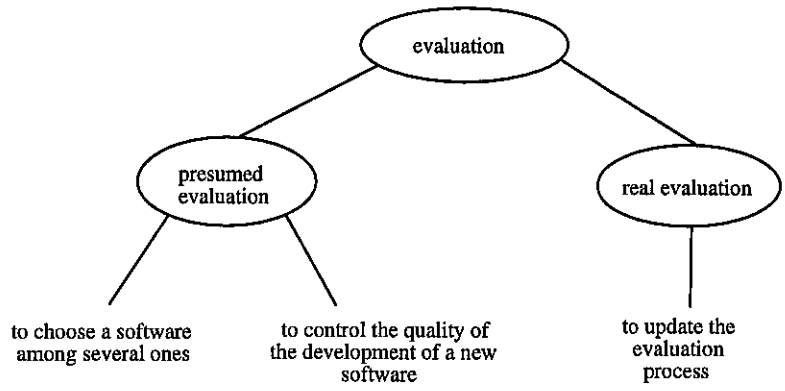

Figure 1: the two types of software quality evaluation

aspects of the software seen like a black box, for instance the time spent to realize a specific treatment or the reply time in a particular context.

A pre-evaluation may be used: 1^o) to choose a ready-made software adapted to the needs of a specific user among several software supposed to fulfill the same kind of needs, 2^o) during the successive development phases of a specific software to control that the specifications, the design, the programming and the tests fulfill the quality requirements of the phases and as a result, that the final software will be conform to the needs.

A post-evaluation allows to compare the presumed results with the real ones. It also allows to understand how the evaluation process can be modified and how to introduce any correcting factors in the pre-evaluation.

1.2 Standards about quality

Works about quality began during the last world war in the military domain in United States of America. A few years later, in Japan, the national quality policy was initiated. During the 60's, then the 70's, in United States of America and in France, standards have been created to control the quality of software development, first in the space research programs and after in the nuclear power ones. Now, every industrial and economic sector is concerned. Several ISO and IEEE standards have been created from the existing work in the quality area. The most important ones concerning the software domain are: ISO 9001, ISO 9000 - 3, ISO 12119, ISO 9126 and IEEE 1061.

The last two standards are specifically relevant to the software quality evaluation. ISO 9126 standard presents a model to describe and to evaluate software quality. IEEE 1061 standard details a methodology to establish software quality requirements, to identify and implement software quality metrics, to analyze the software metrics results and to validate the software quality metrics.

1.3 The software industry challenges

Now, firms are engaged in a cut-throat world competition and bringing a certification process into play allows to guarantee the skill of the firm or the conformity of a product, of a service or of an organization to a predefined reference. Two kinds of certification exist: the certification of enterprises and the certification of products.

The certification of enterprises provides references for the customer. It guarantees that an enterprise is concerned by quality all along the production process.

The certification of a product or of a service attests that the characteristics of the product or of the service is in accordance with technical specifications.

No general certification of specific software exists, only certification of software relevant to a particular domain, for instance the critical software used in aeronautical domain.

Concerning COTS software in France, the "NF logiciel" [AFNOR, 1996] label certifies that: *the functions of the software match their description in the software documentation provided to the customer before purchasing, the software was tested by a registered laboratory, the software quality level is in accordance with the ISO 12119 standard requirements, the quality policy and practices of the supplier are verified, an after-sale service is provided and the software characteristics are durable.*

2 Using the standards to evaluate COTS software

In the last section, we presented two ways to evaluate a software and the relevant standards. Both ways of evaluation described may be used to choose an appropriate COTS software or to control the development quality of a specific software. In this paper, we are only interested by the pre-evaluation of COTS software. This section describes the major principles developed in the ISO 9126 and IEEE 1061 standards, and the difficulties to use these standards for concrete cases.

2.1 Basic principles of quality evaluation developed in the standards

ISO 9126 and IEEE 1061 standards, based on research work in software quality [Boehm, 1978] [Mc Call, 1977] propose: 1°) a common definition of the concept of quality which is "the set of properties and characteristics of a software which defines its capability to satisfy some expressed or implicit needs", 2°) a common quality model (figure 2.a) which defines the quality as an expression of several elements named quality factors.

Quality factors are defined from the user's point of view. As the measures of the quality are made on the components of the software, each quality factor is associated with several quality criteria which express the quality from the software point of view. One or several metrics are finally associated with each quality criterion. For example, in figure 2.b, the quality of a word processing package is defined as: 1°) the capability of the software to execute some specific functions, 2°) the effort needed to learn its using and 3°) its user-friendliness. Three quality factors are thus defined: functionality, learning, user-friendliness. The factor "functionality" is expressed by the criteria: editing, text formatting, data recording, printing. The criterion "editing" is measured by the metrics: granularity of the searching (character, word, line, page), integration of objects (schemes, pictures, texts from another source, hypertext pointers), undoing (number of undoing levels), cutting (with or without the possibility to transfer the information to another software), pasting (with or without the possibility to paste information from another software, to change or to keep the format of the information, the size and the type of characters), copying (with an only one operation or with two operations).

IEEE 1061 and ISO 9126 standards propose six quality factors: efficiency, functionality, maintainability, portability, reliability and usability, a list of quality "criteria" for each of these factors (figure 3) and a list of metrics associated with each "criterion" (figure 4).

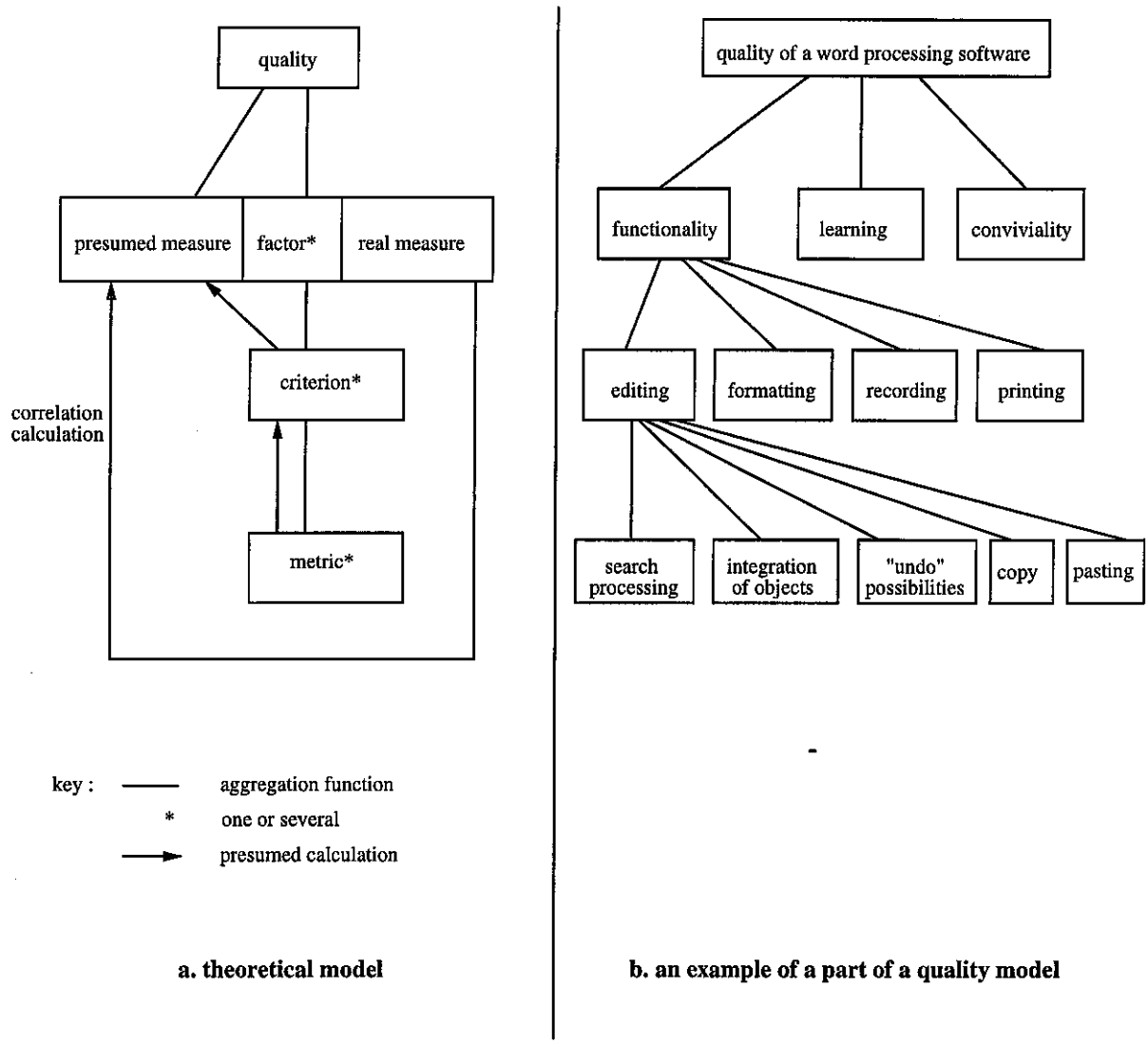

Figure 2: a quality model in accordance with the ISO 9126 and IEEE 1061 standards

Factors	Criteria
functionality	completeness correctness security compatibility interoperability
maintainability	consistency correctness modularity traceability expandability
efficiency	time economy hardware resource economy software resource economy communication economy

Figure 3: example of quality factors and associated criteria proposed by standards

Criteria	Metrics
completeness	- ratio of number of completed documents to total number of documents - ratio of number of completed software components to total number of software components - ratio of number of implemented functions to total number of required functions - ratio of number of implemented user interfaces to total number of required user interfaces
traceability	- ratio of number of software components of this phase that can be traced to the previous phase, to total number of software components of this phase - ratio of number of documents of this phase that can be traced to the previous phase, to total number of documents of this phase - ratio of number of functions that can be traced to the requirements, to total number of functions - ratio of number of user interfaces that can be traced to the requirements, to total number of user interfaces

Figure 4: example of metrics proposed by quality standards

A complex hierarchy is so defined, producing a tree where the root is the global quality evaluation and the leaves are the metrics available on specific features of the software. The intermediate nodes represent "criteria" and sub-criteria depending on the detail and depth of the quality model.

In the primary work from Mc Call [Mac Call, 1977], the measure of each metric was

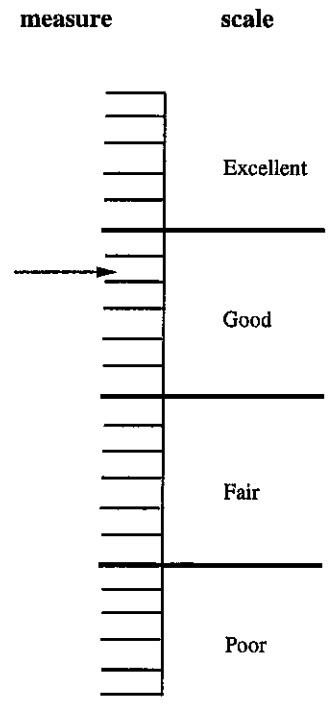

Figure 5: example of scale used in the standards

calculated from answers to a set of questions: the number of positive answers to the set of questions were divided by the total number of questions and the result, which was between 0 and 1, was the measure of the metric.

But, in practice, evaluators need measures of different types: qualitative, quantitative or Boolean. So, scales are used to transform them into an unified score (figure 5).

An aggregation method calculates the measure of each quality criterion using the relative importance of the metrics associated with the criterion and the measure of each quality factor using the relative importance of the criteria of the factor. Finally, the aggregation method calculates the measure of the global quality using the relative importance of the factors. The aggregation method widely used is the weighted sum.

In fact, this process allows to predict the measure of quality factors of either a software that an enterprise wants to buy or of a software development in progress. When the software is bought or finished and is used, factors can be directly measured and correlation between predictive and direct measures can be calculated to control the evaluation model and to adapt it for future uses.

2.2 The difficulties to use the standards for concrete evaluations of COTS software

In this section, we make treasure of the difficulties encountered in the practical use of the standards. These results can be completed by the reading of [Fenton, Schneidewind, 1996] who discuss weak and strong points of standards.

A COTS software is generally largely used in the organizations which purchase them and it is impossible to define and to simulate all the applications the software will go through. Therefore, the evaluation has to consider mainly the software features rather

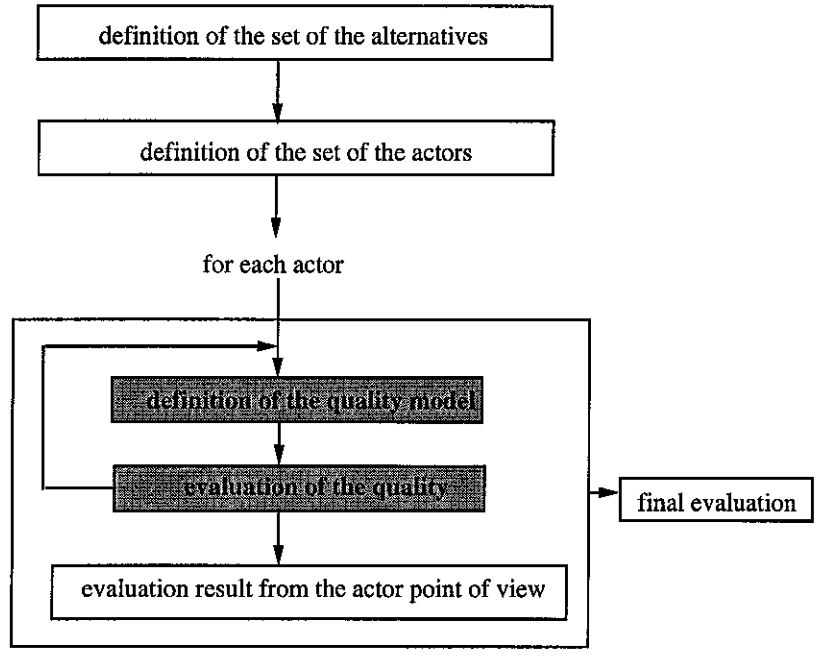

Figure 6: the iterative quality evaluation process

than their behaviour in a real context.

The evaluation of a COTS software is often a long process evolving in time and usually several actors are implied, for example, the final users, the purchase manager, the maintainers of the software, the manager responsible for the integration of the software in the organization or in the technical environment. Each of the actors has his own point of view and his own quality model. Several models have, often, common parts. Generally, each actor builds an a priori quality model with a great number of factors, sub-factors and criteria from his knowledge of domain and from his experience. But it is very difficult for him to determine the decisive elements of the model and to associate"weights" to factors, sub-factors and criteria. Moreover, the different elements forming a quality model are not always independent. For example, a same criterion may be associated with several factors. As a result (using the weighted sum aggregation procedure), this criterion has a greatest weight in the quality model than another one.

Usually, the evaluators proceed by trial and error in order to determine the right choices. So, in fact, the using of a quality model comes within an iterative decision process throughout each actor determine the relevant quality model in refining and adapting the original one (figure 6). But, the standards do not provide means to validate the model and to simplify and customize it.

In addition, a software is not a monolithic block but it is composed, bought and delivered in several parts depending on the needs of the users and it is complemented by services like assistance to its use or maintenance.

As we have seen, the standards propose to transform the measures on the leaves of the hierarchy in homogeneous numerical evaluations (in order to use the weighted sum as aggregation procedure). However, such transformations are often arbitrary and the result is therefore meaningless. Some research have pointed out such a drawback (for a discussion see [Kitchenham, Pfleeger 1996]) on which we will come back in section 4.

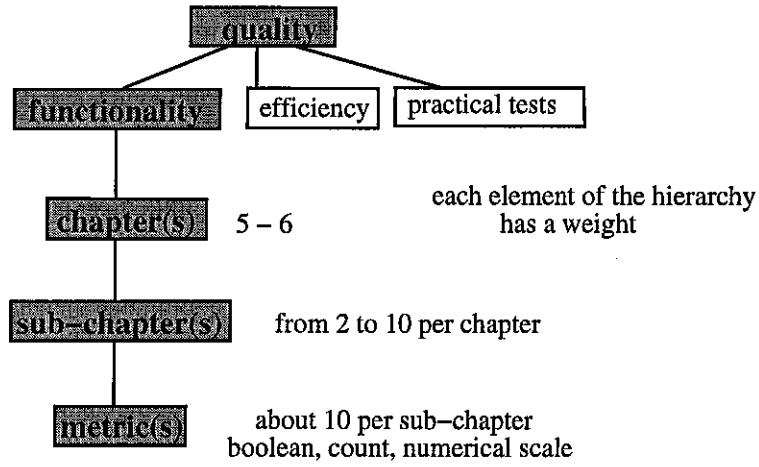

Figure 7: the quality model of the studied cases

3 Applying the multicriteria methodology for concrete quality evaluations of COTS software

In the last section, we explained the main principles of ISO 9126 and IEEE 1061 standards and we stated the difficulties to use these standards to evaluate the quality of COTS software. We present now an experiment of the use of different aggregation procedures (instead of the usual weighted sum) in three real software quality evaluations. The three cases were already gone through an ISO 9126 based evaluation which is first described. Then, the use of ordinal and geometric aggregation is explained. Finally, the results of the experiment are discussed.

3.1 The evaluation problem

The cases studied belong to a French Laboratory which provides comparative studies of COTS software and of computer materials for publishing.

Each of these cases involves six or seven software and two or three actors. They are processed in accordance with ISO 9126 standard using a five level hierarchical quality model (figure 7). The different actors use the same hierarchy but give different weights to the elements. We worked only on a sub-tree of the quality model which contained from 200 to 300 leaves.

The measures of the metrics may be: counts, Booleans or ranges of numerical scales. To normalize them, measures are transformed in marks by the formula below:

mark = (measure / the highest measure of the metric) / the sum of the weight of the metrics of the sub-chapter) * the weight of the metric

For example, Figure 8 represents the measures and their normalization of the metrics of the sub-chapter "Calendars" for two products A and B. Three metrics are defined: last year of the permanent calendar, maximum number of official holidays, definition of a specific calendar of a task. The first metric is a count, the second one is a range of the numerical scale $\{0, 5, 10\}$, the last one is a Boolean.

metrics	type	weight	product A		product B	
			measure	mark	measure	mark
last year of the permanent calendar	count	1	2049	$\frac{2049}{2129} \times \frac{1}{8}$	2129	$\frac{2129}{2129} \times \frac{1}{8}$
maximum number of official holidays	range	2	5	$\frac{5}{10} \times \frac{2}{8}$	10	$\frac{10}{10} \times \frac{2}{8}$
definition of a specific calendar of a task	Boolean	5	0	$\frac{0}{1} \times \frac{5}{8}$	1	$\frac{1}{1} \times \frac{5}{8}$

Figure 8: example of measure normalization in the cases studied

3.2 Ordinal aggregation

This section briefly presents the use of an ordinal aggregation procedure belonging to the family of the ELECTRE methods [Roy, 1991]. A detailed description of the method can be found in Appendix.

Such a procedure was applied from the leaves to the root of the quality model. At each level of the tree, an ordering of the alternatives (the software to evaluate) is calculated for each node of the level. These orderings are used at the next level to calculate new orderings and so on (aggregation of preferences) (figure 10).

The concordance formula was only used. The responsible of the evaluation in the Laboratory was not able to indicate any veto condition on the criteria. Moreover, as most of the criteria were ordinal, it was very difficult to state any veto threshold. Finally the responsible considered that the existence of a veto could act as an "apriori" elimination in which case the set of products to evaluate should be considered as badly chosen.

In order to be able to repeat the calculation at each level of the quality model, the outranking relation obtained at each node of the hierarchy was transformed into a weak order using the "Score method". This method consists, for each alternative, in subtracting the number of times this alternative outranks the others and the number of times it is outranked ("final score" of each alternative) (figure 9). The weak order is established on the basis of the final score of each alternative.

	A1	A2	A3	A4	# of alternatives outranked by x
A1	1	0	0	0	1
A2	1	1	1	1	4
A3	1	0	1	1	3
A4	1	0	0	1	2
# of alternatives outranking x	4	1	2	3	
final score	-3	3	1	-1	
final order used in the next level of hierarchy	4	1	2	3	(1 being the best)

Figure 9: using the Score method to order alternative at each node of the quality model

The ordinal aggregation may conceal situations of incomparability which have to be analysed before calculating the final order of the alternatives at each level of the hierarchy. When incomparable alternatives were detected, a sensitivity analysis was applied. Every alternative better or worst than the incomparable alternatives was kept away. The

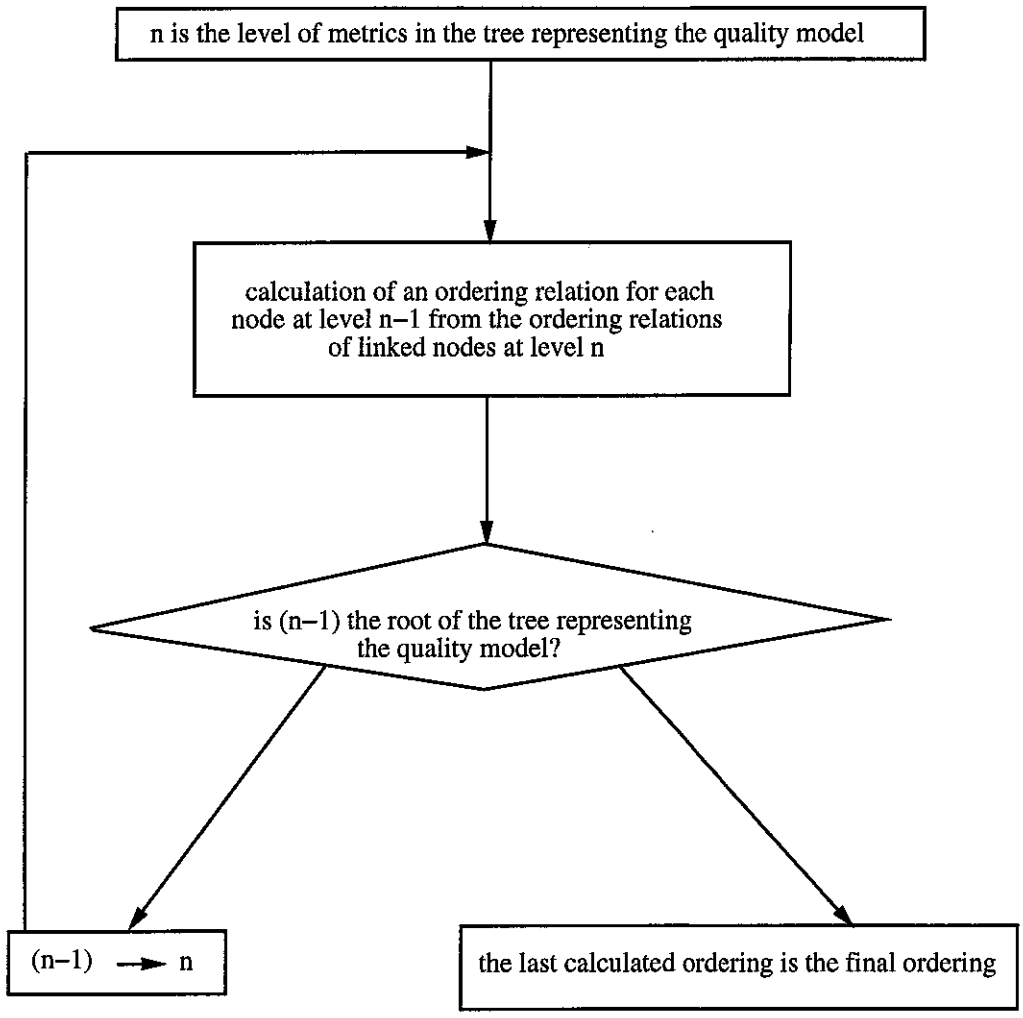

Figure 10: applying ELECTRE II for comparing software

incomparable alternatives and every alternatives ordered between them were retained and the calculation were remade with a new concordance threshold until all incomparabilities disappeared. The idea of the sensitivity analysis is to verify at what confidence level all the alternatives can be compared. In fact, the decision maker wanted to verify if the incomparability was due to the imposition of high confidence or to intrinsic characteristics of the alternatives.

3.3 Geometric means

Another experiment was conducted using the geometric and dual geometric means as aggregation procedures. So, we substituted the weighted sum of the scores used in the existing evaluation procedure for first, the weighted product of the scores and then, for the weighted product of the complement of the scores. More specifically, the two formulas below were used:

$$u(x) = \prod_j (u_j(x))^{w_j} \quad \text{geometric mean}$$

$$u(x) = 1 - \prod_j (1 - u_j(x))^{w_j} \quad \text{dual geometric mean}$$

where:
- $u(x)$: score of alternative x on the parent node;
- $u_j(x)$: score of alternative x on the son node j;
- w_j: relative importance of the son node j;

Obviously the formulas make sense when all evaluations are expressed in the interval $[0, 1]$. However, in our experiment we avoided the extreme values 0 and 1 since the presence of just one of them in the son nodes will keep the global score to 0 or 1 independently from the rest of the evaluations (in other words we attenuated the non compensation effect of the formula).

3.4 Comments on the experience

The use of the ordinal aggregation methods presented two positive features and a negative one.

1. It enables to handle homogeneously non homogeneous information in a meaningful way, since it does not impose any restriction on the information expressed on the criteria (sub-criteria, etc.).

2. It enables to put in evidence situations of incomparability which otherwise could be concealed during the aggregation. Therefore, the ordinal aggregation can be a way to validate the quality model.

3. The information contained in each criterion (sub-criterion, etc.) is often richer than the simple order of the alternatives. It is sometimes a ratio or interval information on the comparison of the alternatives, other times an external measurement or a qualitative judgment, but in all such cases it contains knowledge about a metric and its properties. A purely ordinal aggregation in every level eliminates these information since it focuses on the order of the alternatives. This may lead to a poor conclusion from the point of view of the decision maker. Particularly, in our case, although he was aware that a large part of his criteria was purely ordinal, the decision maker would like to have measures of the distances between the alternatives.

Geometric mean brings out specific "bad" performances of the alternatives since the global score deteriorates exponentially with respect to the importance of the criterion on which the "bad" score is expressed (conversely the dual geometric mean will bring out alternatives with "good" evaluations). Under such a perspective, these both means introduce a non linear compensation effect among the criteria and therefore can be used as measures of "attractiveness" in the interval $[0, 1]$ in the presence of ordinal information also. They may also be replaced with other kinds of ordered statistics and triangular means meaningful as "attractiveness" measures.

4 Discussion

Section 3 describes how we applied the multicriteria methodology to three real cases of quality evaluation and comments the results. This section discusses different issues relevant to quality evaluation: what a measure is, the determination of a measurement scale, the difference of a measure and an evaluation, the drawbacks of aggregation, the properties of the elements of a software quality model and the consequences of the choice of an aggregation method.

4.1 General issues

The problem of quality evaluation (of a service, a product or whatsoever) is often addressed in a confusing way. The basic confusion arises between the "measurement" of quality and the choice (of an alternative) based on quality attributes. These are two completely different activities and have to be treated as such.

The construction of a "measure" requires:

1. the definition of the semantics of the measure (what we measure?);
2. the definition of the structure of the metric (what scale is used?);
3. the definition of one or more standards (how the measure is performed?).

On the other hand, evaluating a set of alternatives under a decision perspective requires to answer questions of the type:
- who evaluates?
- why is the evaluation necessary?
- for what purpose is the evaluation?
- how the evaluation has to be done?
- who is responsible for the consequences?
- what resources are available for the evaluation?
- is there any uncertainty?

A measure is a unary function $m : A \mapsto M$ mapping the set of objects A to the set of measures M. The set M is equipped with a structure $\mathcal{E}$ which is the scale on which the measure is established. Such scales can be ordinal, ratio, interval or absolute [Roberts, 1979]. Each type is univoquely defined by its admissible transformations. Measuring the elements of A can be done only if M is defined. So, an external reference system and standards are necessary (represented by M).

A preference (for decision reasons) is usually represented by a binary relation R, $R \subseteq A \times A$, so that the set A is mapped to itself. We obviously need to know under which conditions $r(x, y) \quad x, y \in A$ is true, but there is no need of external reference system. When R is a complete binary relation $(\forall x, y \in A \quad r(x, y) \vee r(y, x))$, then it admits a

numerical representation which depends on what other properties R fulfills. For instance, if R satisfies the Ferrers property and semi-transitivity (for such concepts, see [Roubens, Vincke, 1985]), then it is known that $\exists v : A \mapsto \mathcal{R} : \ r(x,y) \Leftrightarrow v(x) \geq v(y)+k$ (k being a real constant). A typical confusion is to consider the function v as a "measurement" applied on the set A. Actually there exist an infinity of functions v representing the relation R and anyone could be chosen. Since there is no standard (or metric) any of such functions v is just a numerical representation of R, but not a measure (although we may use a special "preference measurement" concept). For instance, if x is indifferent to y, y is indifferent to z, but x is preferred to z, then two numerical representations of such preferences are $u(x) = 10, u(y) = 12, u(z) = 14, k = 3$ and $v(x) = 50, v(y) = 55, u(z) = 60, k = 6$.

Finally if for a given set A a measurement function exists, it is always possible to infer a preference relation from the measurement. However, such a preference relation is not unique (the fact that two objects have a different length, which is a measure, does not imply a precise preference among them). Suppose that $\exists l : A \mapsto \mathcal{R}$ (a measurement mapping the set A to the reals, let us say a length), then the following expressions are all admissible:

- $r(x,y) \Leftrightarrow l(x) \geq l(y)$
- $r(x,y) \Leftrightarrow l(x) \leq l(y)$
- $r(x,y) \Leftrightarrow l(x) \leq l(y) + k$
- $r(x,y) \Leftrightarrow l(x) \geq 2l(y)$, etc.

These are all admissible preference relations, but with an obvious different semantic. The choice of the "correct" one depends on the answers on the evaluation questions. An evaluation is therefore always a part of a decision aid process and represents its subjective dimension.

The other way around is not always possible to obtain a measurement scale from a preference relation. First of all the preference relation needs to be a complete binary relation (otherwise there is no guarantee that the numerical representation exists), but this is not always the case. Secondly, if the numerical representation exists, it is not necessarily unique. In such a case it is difficult to choose the "correct" measure since we need to know all the possible sets on which such a measurement could apply. Finally it is necessary to build an external metric and this may not be always possible.

The differences between measurement and evaluation (seen as preference modelling) reflect also the possibilities we have to obtain aggregated measures or preferences from sets of attributes or criteria.

4.2 Drawbacks of aggregation

Aggregating measures or preferences is a very common activity. Observations and/or evaluations provide measures or preferences on several distinct attributes or criteria. But we need a comprehensive measure or preference relation which may represent all the different dimensions we want to consider. It is surprising how often the choice of the aggregation operator is done without any critical consideration about its properties. Let us take two examples.

Example 4.1 *Suppose one has two three dimension objects a, b, for which their dimensions are known $(l(a), l(b), h(a), h(b), d(a), d(b))$. In order to have an aggregate measure of each object dimension, one may naturally compute their volume, that is $v(a) = l(a)h(a)d(a)$ and $v(b) = l(b)h(b)d(b)$. If the three dimension are prices, one may use, however, an average, that is $p(a) = l(a) + h(a) + d(a)/3$ and $p(b) = l(b) + h(b) + d(b)/3$.*

From a mathematical point of view, both operators are admissible (when $l(x), h(x), d(x)$ are ratio scales as in our example). However, the semantics of the two measures are quite

different. It will make no sense to compute a geometric mean in order to have an idea of the price of a, b as it will make no sense to compute an arithmetic mean in order to have an idea of the dimension of a, b. The choice between the geometric and the arithmetic means depends on the semantics of the single measures and of the aggregated ones.

Example 4.2 *Suppose one has two objects a, b and two criteria (in the Multicriteria Decision Aid Methodology, a criterion is a preference relation with a numerical representation) g_1 and g_2 such that, $\forall x, y \;\; p_j(x,y) \Leftrightarrow g_j(x) \geq g_j(y)$. Moreover $g_1 : A \mapsto [0,1]$ and $g_1(a) = 0$ and $g_1(b) = 1$ and $g_2 : A \mapsto [0,2]$ and $g_2(a) = 2$ and $g_2(b) = 1$. Under the hypothesis that the both criteria are of equal importance, many people will compute the average and infer the global preference relation. In our case, one has $i(a,b)$ ($i(x,y)$ representing indifference) since $g(a) = g_1(a) + g_2(a)/2 = 1$ and $g(b) = g_1(b) + g_2(b)/2 = 1$. However if an average is used, it is implicitly assumed that g_1 and g_2 admit ratio transformations. Therefore it is possible to replace g_2 by $g_2' : A \mapsto [0,1]$ so that $g_2'(a) = 1$ and $g_2'(b) = 1/2$ (known as scale normalization). Under the usual hypothesis of equal importance of the two criteria, we obtain now $p(b,a)$ since $g(a) = 1/2$ and $g(b) = 3/4$. Where is the problem?*

The problem is that the average aggregation was chosen without verifying if the conditions under which, are admissible hold. First of all, if the values of a and b are obtained from ordinal evaluations (of the type good, medium, bad, etc.), then the numerical representation does not admit a ratio transformation (in other words we cannot use its cardinal information). Secondly, even if the ratio transformation was admissible, the concept of criteria importance is misleading. In a "weighted arithmetic mean" (as the average is) the "weights" are constants representing the ratio between the evaluation scales. In the example, if we reduce g_2 to g_2', we have to give to g_2' twice the importance of g_1 in order to keep true the concept of "equal importance". In other words, it is not possible to speak about importance of the criteria (in the weighted arithmetic mean case) without considering the cardinality of their co-domains.

Example 4.3 *Let us take an example from the case discussed in section 3.1. Consider two software a and b, a set of n "evaluation dimensions" where $n-1$ dimensions are of Boolean type (a software fulfills or not a certain property) and one dimension indicates the time horizon of the internal clock (in the example of section 3.1, the years 2049 and 2129). If all the evaluations are reduced in the $[0,1]$ scale, then the information of the last dimension is completely destroyed since it is an absolute scale which does not admit any transformation. Further on, a weighted sum on such evaluations is meaningless since the ordinal information of the $n-1$ Boolean dimensions is transformed into a ratio evaluation in an arbitrary way.*

From the above examples, we can induce a simple rule. *In order to choose appropriately an aggregation operator, it is necessary to take into consideration the semantics of the operator and of each single preference or measure and the properties (axiomatics) of the aggregation operator.* In other words, if the aggregation operator is chosen randomly, neither the correctness of the result, nor its meaningfullness can be guaranteed.

4.3 Software evaluation

As already discussed in the previous sections, software evaluation uses a complex hierarchical quality model. Moreover, the evaluation may concern parts of the software itself (or the whole), different dimensions and can be done for different purposes [Morisio, Tsoukiàs, 1997], [Stamelos, Tsoukiàs, 1998].

Provided that a first idea of the evaluation to be performed is outlined, it is necessary to establish an evaluation model. Following the IUSWARE specifications [Morisio, Tsoukiàs, 1997], we may consider that a set of alternatives A (a set of software, or parts to be evaluated), a set of dimensions or "points of view" V under which the evaluation has to be done and a problem statement Π describing the purpose of the evaluation are available. We call such a triple a "problem formulation" at time t $\Gamma_t = \langle A, V, \Pi \rangle$.

An evaluation model consists in a n-uple $\mathcal{M} = \langle A^*, D, M, \mathcal{E}, G, R$, where A^* is the set of alternatives to evaluate after an eventual preliminary screening, D is the set of attributes considered instantiating V, M and $\mathcal{E}$ are any measures and relative scales to be used, G is a coherent set of criteria to represent any preferences and R is a set of aggregation operators to apply on M and/or G. As we already said, the construction of such a model may be a long process including different activities, feedbacks and revisions. In the present, we will concentrate our attention on two specific activities.

Build the sets M and G. This is usually inferred from D which provides an explicit representation of the different points of view (set V) introduced in the problem formulation. Usually the set D is obtained using international standards (as the ones included in ISO 9126 and IEEE 1061) and/or the client's specific knowledge about the kind of the software to evaluate.

Two processes are performed in a parallel way. The first, top-down, in which general dimensions (factors in the IEEE terminology) are disaggregated to specific sub-dimensions and so on until sub^n dimensions are reached on which the client is able to express or gather for or build up some information. The second, bottom-up, from the client's specific knowledge who identifies subsets of evaluation dimensions as sub-dimensions of a dimension on a higher level. The result of the two processes is the definition of a hierarchy of the type presented in the previous sections.

However, in such an activity it is necessary to pay attention not only to the semantic relevance of the son nodes of a parent node, but also in verifying their independence. The basic and necessary independence condition to meet is the "separability" of the "son-nodes" (the nodes to be aggregated in a parent node). Intuitively the notion of separability means that if two objects are perfectly equivalent on all son-nodes except one, then the difference on such single son-node should be reflected to the parent node. In other words, every single son-node should be able to discriminate two objects alone. If such a condition is not verified, then the set of son-nodes has to be reconsidered. Further independence conditions can be imposed, but they deal with specific aggregation operators and will not be discussed here (see, however, [Roberts, 1979], [Von Winterfeldt, Edwards, 1986], [Vincke, 1992] and [Roy, 1996]).

Summarizing, given a more or less intuitively created evaluation hierarchy, both a semantical and independence verification has to be performed before it can be included in the evaluation model $\mathcal{M}$.

Define the set R. Associating a subset of nodes to a parent node implicitly assumes that an aggregation operator is also associated to the parent node such that the information contained in the son-nodes is propagated to the parent node. As already observed at the beginning of the section, the choice of an aggregation operator is not neutral and has to be done appropriately. A first distinction to be done is between preference and measurement aggregation.

1. Preference aggregation. If the parent node is expected to represent preferences on the set A^*, then the son-nodes have to carry such an information. If it is not the case, then a preference relation has to be associated to each son-node. If any

measures are available and if they are at least expressed on an ordinal scale, then a preference relation can be inferred applying such a measurement on the set A^* (for instance, if the measure m_j is available, we can define $p_j(x,y) \Leftrightarrow m_j(x) > m_j(y) + k$ or $p_j(x,y) \Leftrightarrow m_j(x) > 2m_j(y)$, etc.).

If the measures available on a son-node are just nominal, then an ad-hoc preference model has to be established on the set A^*. Once each son-node is equipped with a preference model, a preference aggregation method can be used, acting either on the numerical representations of the preference relations (if they exist), or directly on the binary relations. Some basic rules can be remembered.

- If at least one of the numerical representations is obtained from an ordinal scale, then only ordinal aggregation operators can be used.
- If a linear multiattribute value function is going to be used, then linear preferential independence on the set G has to hold, a compensation principle is accepted and weights are trade-offs.
- If an ordinal aggregation has to be used and a complete global preference relation is required, then either it will be dictatorial, or it will not respect the independence from irrelevant alternatives [Arrow, 1951].

For a comprehensive discussion, see [Keeney, Raiffa, 1976] and [Vincke, 1992].

2. Measurement aggregation. If a new measurement scale is defined on the parent node, then the basic operation is to establish the semantics of the new scale and its structure. Once a metric has been chosen, the most appropriate aggregation operator can be identified. Again some basic rules can be remembered.

   - A result of an aggregation cannot carry more information than the one contained in the aggregated nodes (for instance, it is not possible to construct a ratio scale aggregating ordinal scales).
   - If the aggregation operator requires scale transformations, these have to be compatible with the admissible transformations of any single aggregated measurement (for instance, an interval transformation is not admissible on a ratio scale).
   - If weighted statistics are used, weights should respect scale ratios (provided that such a ratio makes sense).

   In order to give an example, let us consider a scale whose values are 10, 20 and 30. If it is a ratio scale, then a linear transformation of the type $\alpha x + \beta$, with $\alpha = 1$ and $\beta = 5$, will give the new scale with values 15, 25 and 35 where the ratio information is lost. Such a problem is particularly relevant when a normalization of the scales is required due to different reasons. Unfortunately this is possible only if the scales are of the same type.

Once an evaluation model is constructed, a validation process should be performed in order to verify if the model effectively turns out the expected results. As far as the aggregation operators are concerned, from our experience, two types of results are questionable and have to be discussed with the client.

1. Unexpected measures for some objects on some nodes are obtained. Clearly, the aggregated measure, as defined in the model, does not apply well to the set A^* and the measures on these nodes have to be redefined.

2. An incomparability emerges for a couple of objects in a node. If it is unexpected, we have to look for the reasons: 1°) perhaps, very different objects are considered and the set A^* has to be discussed and eventually redefined, 2°) the quality model may represent strongly conflicting evaluations and a new compromise has to be established, 3°) the information available is not sufficient and further investigation is necessary.

5 Related work

A great number of papers present experiences of application of ISO 9126 and IEEE 1061 standards to evaluate COTS software or to evaluate the productions of the different software development phases. Such papers focus on different aspects of quality evaluation: definition of the model to use, control of the quality obtained and comparison with the required quality, introduction of the concept of point of view and of software component, representation of dependencies between elements of the model and simplification of the model. In the following, we present and discuss some of these papers related with our research.

Two additional aspects of quality evaluation - the aggregation methods used to calculate the quality of factors, sub-factors and entire software and the evolution of the quality model - are not yet really studied by researchers in software quality but researchers in other communities propose solutions which may be efficiently reused. Some of these propositions are also presented and discussed in the following.

A final aspect of software quality evaluation is not yet studied: the correlation between predictive qualities and real ones measured on the operating software. ISO 9126 and IEEE 1061 standards claim that measures have to be taken on the operating software in order to calculate the correlation between the predictive measures and the real ones so that to be able to correct the quality model or to add corrective parameters for future evaluations. On this point and up to our knowledge, no literature exists. Under the same perspective, two questions are asked in [Kitchenham, Pfleeger, 1996] but left without answer. The first question concerns the fact that some companies take a product-based approach while others focus on process. Presently, there is no proof that the improvement of the quality of the process enhances the quality of the product. The second question is: the quality of a product is evaluated in measuring internal quality indicators, but how can we be sure that internal quality assures external one and how is it possible to determine which aspects of internal quality really affect the final quality of the product?

Definition of the quality model

Most of the software quality evaluations use models inspired from ISO 9126 and IEEE 1061 standards. But the question is: how to determine relevant elements of the model and their relative weight? In the system described in [Meskens, 1994], an ISO 9126 like quality model (including factors, sub-factors, criteria, weights and aggregation methods) is dynamically built and proposed to the evaluator who may adapt it. The process uses a knowledge base testing the domain of the application and the target software environment. Each criterion is associated with metrics which concern different elements of the code and elements of check lists like the uniqueness of the meaning of each variable, the indentation of the code, the use of a programming method. The measure of each element of the check lists is calculated from elementary metrics using rules of the knowledge base. Aggregation methods are defined by other rules of the base which specify how to calculate the quality of each criterion from its associated metrics, of each sub-factor from its associated criteria, of each factor from its associated sub-factors and of total quality from the factors. However,

no discussion concerning the relevance of the aggregation methods for the different possible evaluations is presented.

In [Erikkson, McFadden, 1993], the Quality Function Deployment method (a Japanese method introduced in 1972) is used. The main purpose of this method is to allow all employees in the organization to participate in the design of new products. The higher level of the model defines customer quality requirements. Each of these requirements is linked to software characteristics which are themselves linked to software sub-characteristics. At the two lower levels, design and implementation of product features relevant metrics are proposed. The model is represented at each level by matrices.

In [MacDonell, 1993], the Goal/Question/Metric paradigm is used to determine a set of metrics associated with the most used specification representations: entity-relationships, data flow diagrams, transaction model, user interfaces and functional decomposition hierarchy.

In [Dromey, 1996], the author proposes, for each phase of software development (requirement, design and implementation), relevant ISO 9126 high-level quality factors, the components of the phase and for each of them, its quality attributes and their relationships with quality factors.

In fact, it is very difficult to define a quality model because, even if the enterprise has already made other evaluations of the same type, the situations are always different. So, the quality model has to be changed and it is important to provide the evaluators with means to validate their model.

Control of the quality obtained and comparison with the required quality

In this area, concerned researchers try to answer to the questions: how to build a software with specific qualities? How to control that a software has some specific qualities? The papers which try to answer to these questions provide means to determine the acceptable values of the different elements of the model and to include them into the model. In [Meskens, 1994], rules are included in the knowledge base to compare the calculated quality of each criteria, of each sub-factor and of each factor with allowed values, to provide diagnoses using a bad, medium and good scale and to propose actions to the user for improving the quality.

In [Spinelli and al., 1995], the authors present a model and a related tool which, using the ISO 9126 standard quality model, allow to compare quality requirements to measured qualities. The standard quality model is completed by a quality matrix which provides a required evaluation of each quality factor (functionality, portability, efficiency,...) for each component of the software. As the measurement process may be long and costly, the authors focus on the pruning of the model. So, they try to reuse measurement apparatus from the literature, consider only the characteristics for which non-low value is required and stop measurements as soon as further measurements would not change the final result. They underline the need to tune the model in trying to correlate the a priori estimates (measurements on the product) with a posteriori facts (data on the program use, faults, maintenance costs,...).

In [Tervonen, 1996], a tool to help and to guide design choices and design inspections is described. It is based on ISO 9126 quality model where the standard quality factors are decomposed in sub-factors adapted to object-oriented design. Specific metrics relevant to class and sub-class design are associated to each sub-factor. Factors and sub-factors are associated to a ISO 9126 like scale. Nothing is said about aggregation of measures and calculation of the scores of factors and sub-factors. During inspections, inspectors record measures about each metric in the tool. Replies from developers are also recorded in the tool. During design, developers can use the model to evaluate the quality of design

alternatives.

[Khoshgoftaar and al., 1996] present a discriminant analysis based method for predicting quality of software modules from their design. The method uses metrics of the call graph and of the control graph of modules representing 1,3 million lines of code of a very large communication system.

[Drake, 1996] presents an initiative of the National Security Agency of USA to reduce maintenance costs while improving software development. Measures on 25 millions lines of C, C++, FORTRAN and ADA code have been collected and correlated to the results of dynamic testing of executable programs in a real-world environment. Metrics include Halstead's software measures (purity ratio, volume and effort), Mac Cabe's cyclomatic complexity, functional density, number of executable paths per function, number of segments per path, comments and span of references for variables. Results concern productivity, maintainability, testability and performance. For each metric, an ideal value and a range of satisfactory values are statistically calculated. An example of measures of two programs coding the same function is described: an untestable and low-performance program and a good one.

[Ogasawara and al., 1996] present a tool that automatically measures the number of procedures per module and the number of conditions, of loops, of arguments and of comments per procedure. These different measures are recorded all along the design, the programming and the test phases and their progression are calculated. The tool also allows to record the number of faults discovered during the test phase. The paper reports experiences of using the results of measures during programming and testing reviews to take necessary actions for keeping the measures under a certain level and to examine the progression of detected faults.

In [Dromey, 1995], a model based on the ISO 9126 standard makes clear and direct links between high level quality attributes (those of the standard), the different components of a software (variables, modules, statements, expressions,...), explicit product characteristics (complete, consistency, used, generic,...) and metrics. A rule-based tool detects defects on product characteristics and quality attributes per thousand-lines-of-code. It provides measures of the global quality or of a specific quality attribute. In addition, the proposed model provides direct guidance for building quality into software. It was applied on C source codes, but it can also be adapted for defining and controlling the quality of the requirements.

Introduction of the concept of point of view and of software components

In ISO 9126 and IEEE 1061 standards, the model is static and does not integrate the concepts of point of view and of software component. Some authors introduce them. Generally, they propose to build several models, each of them being adapted to an actor of the evaluation and to add a level "component" to the models. But, even though both the notions of point of view and of software component have to be considered in some evaluations, the authors generally introduce only one of them. In [Verner and al., 1996], the current state-of-practice for software quality in Information Systems Departments in Hong Kong is analyzed from replies to 175 questionnaires sent to a wide variety of occupation groups and enterprises. The authors underline the importance to consider several points of view in defining software quality, for example, the developer, the buyer, the user, the maintainer, the project manager, the accountant and different specialists like lawyers, and the need to adapt the quality model to the evaluator. In fact, the authors report that: 1°) most of the quality factors proposed in the ISO standards were important for the asked people, but a few of them are ignored, while new factors are added, 2°) if some of the factors used by the different groups to evaluate the software quality are the

same like reliability, maintainability and functional correctness, other ones are specific and the relative importance of the factors is always different.

In [Feuk and al., 1995], the metrics are disconnect from the model to allow the building of different quality models using the same metrics and the same results of metrics. The structure of the model is equivalent to the structure proposed in the ISO and IEEE 1061 standards, except that the model does not have only one root, but as many roots as different points of view.

In [Paulussen, 1995], the author underlines that a software interests several groups of people, for example the users and the maintainers and that each of these groups does not consider the same characteristics of the software as essential. Only one quality model is defined but it is the result of the merged opinions of the people concerned.

ISO 9126 and IEEE 1061 standards do not integrate the notion of component. Software are considered as monolithic blocks. Some authors [Dromey, 1995], [Spinelli and al., 1995] remark that required qualities and metrics are often different for the different components of a software and introduce the level "component" in the quality model. Each component is linked with quality factors and with metrics.

Representation of the dependencies between the elements of the model

Implicitly, ISO 9126 and IEEE 1061 standards consider that factors and sub-factors are independent. Especially, the calculation of quality values based on the weighted sum requires such an independence. In fact, very often, the elements of the quality model are not all independent. Some authors introduced tools to represent dependencies. In [Paulussen, 1995], the Quint quality model contains two levels: at the top, characteristics which are equivalent to factors in the ISO 9126 standard, and at the second level, indicators, each of them being associated with one characteristic. To represent the dependencies between characteristics, the model adds a correlation table providing the impacts of each design decision on the quality measures of each characteristic (that provides an answer to the question: does the design decision increase or decrease the quality?). An algorithm clusters the characteristics that are affected by the same design decisions. The correlation table is constructed a priori, not with the result of measurement of quality of each characteristic.

In [Erikkson, McFadden, 1993], matrices represent the dependencies between elements at each level of the model.

Practically, it is often impossible to introduce the dependencies in the model because they are not known at the beginning of the evaluation process. So, it is interesting that the evaluation method provides means to discover them. The explicit identification of the incomparabilities may allow this.

Simplification of the model

A quality model is generally complex, but most of the researchers do not try to simplify it. However, the knowledge of the decisive elements which cause a fall in the quality or which place a COTS software ahead of others in a league may efficiently help the decision makers.

A method to calculate a minimal model is presented in [Anderson, Chen, 1997]. The authors are interested by calculating a statistical evaluation of the software user satisfaction. They present a model to compare several COTS software and a method to reduce the initial model. Software are categorized and six attributes measure their quality. Several hundred users of software products were asked to evaluate each attribute of the product(s) they own, according to their satisfaction using scaled replies ranging between 1 to 10. Each software is evaluated using the weighted sum of the average measure of each attribute.

The weight of the attributes is statistically calculated from the users' answers. A method based on principal component analysis allows to discover components of attributes. It also allows to observe that three components are enough to evaluate software (useful components are not necessarily the same for different categories of software). This work allows the software editors to improve the products or to adapt them to the expectations of the users.

The consideration of the iterative aspect of the decision process and the providing of several aggregation methods to the evaluators may allow them, by successive tests, to bring out significant elements of the model and so to simplify it.

Aggregation methods

Most of the work which focus on the quality model do not reference the aggregation methods used to calculate the values of the different elements of the model. In fact, aggregation methods are not really discussed in the papers concerned by software quality. In [Meskens, 1994], the author precises that the model includes the aggregation method and the relative importance of factors and of criteria, but unfortunately, every example described in the paper uses the weighted sum.

Researchers belonging to the multicriteria community have published interesting papers about aggregation methods but these studies do not consider any quality model. For example, in [Le Blanc, Jelassi, 1994], the results of using the LWA (Linear Weighted Attribute) and the MAUT (Multi-Attribute Utility Theory) aggregation methods for the evaluation of a small set of software are compared. Both methods give the same results but the statistical analysis of the MAUT method results gives more precise and more detailed information about the differences of the software to the evaluator. However, in this paper, no quality model is used for the evaluation.

In the case of software evaluation, if the providing of several aggregation methods is suitable, it is also important to warn the evaluators about the validity of each of them to prevent bad use.

Evolution of the model

In the experiences presented, one or sometimes several quality models are defined at the beginning of the decision process and they do not evolve during the process. In fact, a decision process is long, hesitating and iterative. So, the models cannot remain static. For example, in an other application area, [Ballou, Tayi, 1996] propose a method and a tool to schedule project maintenance and to assign staff to these projects which consider the non linear nature of the decision process and that intuitive and no quantifiable factors influence the decision. The purpose of the method is to provide information that guides the decision maker in obtaining a satisfying solution. Thus, the method allows the decision makers, by an iterative process, to modify the problem environment and to generate different solutions, one of which is ultimately selected. The method is based on an integer programming formulation, defines the maintenance projects by their priority and cost, and integrates constraints on projects and on costs (for example, two projects cannot be simultaneously processed or the total cost of the projects has to be less than a defined amount).

A software evaluation system has to allow, like in the example just described, to keep the measures of the metrics for reusing them with a changed quality model and to compare the different results obtained.

6 Conclusion

We presented in this paper an experiment of applying multicriteria methodology to the evaluation of COTS software quality. This experiment used real industrial cases already treated with a traditional method inspired from ISO 9126 and IEEE 1061 standards. We analyzed the limits of this traditional method and explained the difficulties of its practical use when the hierarchy of the quality model is deep, the measures are of different types, the number of elements of the model is great and the elements are not all independent. The use of three multicriteria methods was experimented: ordinal aggregation procedures of the ELECTRE type, geometric mean and dual geometric mean. The advantages of the use of ELECTRE type methods are to allow the handling of non homogeneous information in a meaningful way and to detect incomparabilities, helping the validation of the model. One drawback is that the result is only an order of the alternatives without measures of the distances between the alternatives. Geometric and dual geometric means bring out specific "bad" or "good" performances of the alternatives and can be used as measures of "attractiveness" in the [0,1] interval in presence of ordinal information.

In fact, this experiment could be extended in two directions. First, since the validity of the aggregation method depends on the types and of the semantics of the measures, it is wrong to integrate only one aggregation method in the quality model. So, it should be interesting to build a system which considers several aggregation methods, associating each node of the quality model with one of them, and which helps the user to validate his choices. Another more thorough investigation should be to experiment the use of aggregation procedures for sorting purposes which would calculate ordinal scales for each node of the quality model and thus, would provide more insight about the alternatives values and their distance.

More long-term interesting research could concern three points. First, the validation of the model and the iterative aspect of the decision-process. For that, the detection of incomparabilities has to be exploited to understand the decisive elements of the model and so to simplify it. Moreover the formal verification of coherence among the criteria could detect redundancies and repetitions which could again help the decision maker to simplify his model identifying only the relevant criteria. Second, the problem of incomplete or partial models: some measures may be impossible or irrelevant for some alternatives. In such cases, the evaluators are often led to give a meaningless value for some metrics of some alternatives. Third, the correlation between predictive evaluations and real ones. Quality models are built a priori and there is no proof that the metrics correspond to the right measures of the quality of the factors. The points to consider here are how to measure factors a posteriori, how to determine metrics which are responsible of differences between predictive and real measures and how to correct the model.

References

Evan E. Anderson, Yu-Min Chen, (1997), *Microcomputer software evaluation: An economic model*, Decision Support Systems 19, pages 75-92, 1997

Arrow K., (1951), *Social Choice and Individual Values*, J. Wiley, New York.

Association Française de NORmalisation, (1996), *Règlement NF Logiciel, Edition 1.0, 8 mai 1996*

Donald P. Ballou and Giri Kumar Tayi, (1996), *A Decision Aid for the Selection and*

Scheduling of Software Maintenance Projects, IEEE Transactions on Systems, Man and Cybernetics, Volume 26 number 2, march 1996, pages 203 - 212

Barry W. Boehm, (1978), *Characteristics of Software Quality*, North Holland Publishing Company

Thomas Drake, (1996), *Measuring Software Quality: A Case Study*, IEEE Computer November 1996, pages 78 - 87

R. Geoff Dromey, (1995), *A Model for Software Product Quality*, IEEE Transactions on Software Engineering, Vol 21, No. 2, February 1995, pages 146 - 162

R. Geoff Dromey, (1996), *Cornering the Chimera*, IEEE Software, January 1996, pages 33 - 43

I. Erikkson, F. McFadden, (1993), *Quality function deployment: a tool to improve software quality*, Information and Software Technology, Volume 35, number 9, September 1993, pages 491 - 498

Norman Fenton and Norman F. Schneidewind, (1996), *Do Standards Improve Quality?*, IEEE Software, January 1996, pages 22 - 24

N. Feuk, R. Whitty and Y. Ilruka, (1995), *Applying the Goal/Question Metric paradigm in the experience factory*, International Thomson Computer Press, London, pages 23-44

The Institute of Electrical and Electronics Engineers, (1992), *Standard for a Software Quality Metrics Methodology*, December 1992

International Organization for Standardization, (1994), *ISO 8402: Management of quality and insurance of quality (vocabulary)*

International Organization for Standardization, (1987), *ISO 9000: Quality management and quality assurance standards - Guidelines for selection and use*

International Organization for Standardization, (1991), *ISO 9000-3: Guidelines on the application of ISO 9001 to the development, supply and maintenance of software*

International Organization for Standardization, (1994), *ISO 9001: Quality systems - Model for quality assurance in design/development, production, installation and servicing*

International Organization for Standardization, (1991) *ISO 9126: Information Technology - Software product evaluation - Quality characteristics and guidelines for their use*

International Organization for Standardization, (1994) *ISO 12119: Information Technology - Software, Package, Quality Requirements and Testing*

Keeney R., Raiffa H., (1976), *Decision with multiple objectives: preferences and value trade-offs*, J. Wiley, New York.

Taghi M. Khoshgoftaar, Edward B. Allen, Kalai S. Kalaichelvan, Nishith Goel, (1996),

Early Quality Prediction: A Case Study in Telecommunications, IEEE Software, January 1996, pages 65 - 71

Barbara Kitchenham, Shari Lawrence Pfleeger, (1996), *Software Quality: The Elusive Target*, IEEE Software, January 1996, pages 12 - 21

J. Kontio, (1996), *A Case Study in Applying a Systematic Method for COTS Selection*, Proceedings of the 18th ICSE, 1996, pages 201 - 209

Louis le Blanc, Tawfik Jelassi, (1994), *An empirical assessment of choice models for software selection: a comparison of the LWA and MAUT techniques*, Revue des systèmes de décision vol. 3 - no.2, pages 115 - 126

J. A. Mac Call, (1977), *Factors in Software Quality, Journal of General Electric, no. 77, C15-OL, June 1977*

S. G. MacDonell, (1993), *Deriving relevant functional measures for automated development projects*, Information and Software Technology Volume 35, number 9, September 1993, pages 499 - 512

Nadine Meskens, (1994), *A knowledge-based system for measuring the quality of existing software*, Revue des systèmes de décision vol. 3, no. 3, pages 201 - 220

M. Morisio, A.Tsoukiàs, (1997), *IusWare: a methodology for the evaluation and selection of software products*, IEE.-Softw. Eng. Vol. 144, pages 162 - 174

Hideto Ogasawara, Atsushi Tamada, Michiko Kojo, (1996), *Experiences of Software Quality Management Using Metrics through the Life-Cycle*, Proceedings of ICSE - 18, March 25 - 29 1996, Berlin, Germany, pages 179 - 188

Robert M.C. Paulussen, (1995), *The Quint Approach to the Specification of Software Quality*, 1st World Congress for Software Quality, San Francisco, June 20-22, 1995

Roberts F.S., (1979), *Measurement Theory with Applications to Decision Making, Utility and the Social Sciences*, Addison Wesley, New York.

Roubens M., Vincke Ph., (1985), *Preference Modelling*, LNEMS 250, Springer Verlag

Roy B., (1991), *The outranking approach and the foundations of ELECTRE methods*, Theory and Decision, vol. 31, pages 49-73

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

Andrea Spinelli, Daniela Pina, Paolo Salvaneschi, Ernani Crivelli, Roberto Meda, (1995), *Quality Measurement of Software Products: An Experience About a Large Automation System*, Lecture Notes in Computer Science No. 926, Proceedings of the Second Symposium on Software Quality Techniques and Acquisition Criteria, Florence, Italy, Mai 1995, pages 192 - 206

Stamelos J., Tsoukiàs A., (1998), "Software evaluation problem situations", submitted

Lorenzo Strigini, (1996), *Limiting the Dangers of Intuitive Decision Making*, IEEE Software, January 1996, pages 101 - 103

Ilkka Tervonen, (1996), *Support for Quality-Based Design and Inspection*, IEEE Software, January 1996, pages 44 - 54

Dr. June Verner, DR. Trevor Moores, Mr. A.R. Barrett, (1996), *Software quality: perceptions and practices in Hong Kong*, Working Paper Series, City University of Hong Kong, Faculty of Business Department of Informations systems, WP96/02, April 1996

Vincke Ph., (1992), *Exploitation of a crisp relation in a ranking problem*, Theory and Decision, vol. 32, pages 221-240.

Winterfeldt Von D., Edwards W., (1986), *Decision Analysis and Behavioral Research*, Cambridge University Press, Cambridge MA.

Appendix: ELECTRE II method applied to the experiment

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

More precisely for any pair of alternatives (x, y), we have

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

where:
$S(x,y)$: the alternative x is at least as good as y (x outranks y);
$C(x,y)$: concordance condition (in our specific case):

$$C(x,y) \Leftrightarrow \frac{\sum_{j \in J^{\geq}_{xy}} w_j}{\sum_j w_j} \geq c \text{ and } \frac{\sum_{j \in J^{>}_{xy}} w_j}{\sum_{j \in J^{<}_{xy}} w_j} \geq 1$$

with:
$J^{\geq}_{xy}$: criteria for which $S_j(x,y)$ holds;
$J^{>}_{xy}$: criteria for which $\neg S_j(y,x)$ holds;
$J^{<}_{xy}$: criteria for which $\neg S_j(x,y)$ holds;

$$D(x,y) \Leftrightarrow \exists\, g_j : v_j(x,y)$$

where:
$v_j(x,y) \Leftrightarrow g_j(y) > g_j(x) + v_j$, or
$v_j(x,y) \Leftrightarrow \frac{g_j(y) - g_j(x)}{t_j - b_j} \geq v_j$
with:
$v_j(x,y)$: veto condition on criterion g_j;
v_j: veto threshold on criterion g_j;

t_j: max value of criterion g_j;
b_j: min value of criterion g_j;

The conditions under which $S_j(s,y)$ holds depend on the preference structure of each criterion. If g_j is:
- a weak order: $S_j(x,y) \Leftrightarrow g_j(x) \geq g_j(y)$,
- a semi order: $S_j(x,y) \Leftrightarrow g_j(x) \geq g_j(y) - k_j$,
- and so on with interval orders, pseudo orders, etc.

Once the global outranking relation is obtained, we can deduce:
- a strict preference relation $p(x,y)$ between x and y:
$p(x,y) \Leftrightarrow s(x,y) \wedge \neg s(y,x)$;
- an indifference relation $i(x,y)$ between x and y:
$i(x,y) \Leftrightarrow s(x,y) \wedge s(y,x)$;
- an incomparability relation $r(x,y)$ between x and y:
$r(x,y) \Leftrightarrow \neg s(x,y) \wedge \neg s(y,x)$;

The definition of the relation $s(x,y)$ is such that only the property of reflexivity is guaranteed. Therefore, neither completeness nor transitivity holds, and thus $s(x,y)$ is not an order on the set A. In order to obtain an operational prescription, the relation $s(x,y)$ is transformed in a partial or a complete order through an "exploiting procedure" which can be of different nature (see Vincke, 1992).