Hybrid Assessment Method for Software Engineering Decisions

Rita A. Ribeiro[1], Ana M. Moreira[2], Pim van den Broek[3], Afonso Pimentel[1,2]
[1]UNINOVA, CTS, Departamento Engenharia Electrotécnica, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
email: rar@uninova.pt
[2]CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
[3]Department of Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, the Netherlands

Abstract

During software development, many decisions need to be made to guarantee satisfaction of the stakeholders' requirements and goals. The full satisfaction of all of these requirements and goals may not be possible, requiring decisions over conflicting human interests as well as technological alternatives, with an impact on the quality and cost of the final solution. This work aims at assessing the suitability of multi-criteria decision making (MCDM) methods to support software engineers' decisions. To fulfil this aim, a HAM (Hybrid Assessment Method) is proposed, which gives its user the ability to perceive the influence different decisions may have on the final result. HAM is a simple and efficient method that combines one single pairwise comparison decision matrix (to determine the weights of criteria) with one classical weighted decision matrix (to prioritize the alternatives). To avoid consistency problems regarding the scale and the prioritization method, HAM uses a geometric scale for assessing the criteria and the geometric mean for determining the alternatives ratings.

Keywords: multi-criteria decision making; non-functional software requirements; software engineering; aggregation operators

1. Introduction

In software development, many choices need to be made and decisions to be taken; these depend on obtaining answers for some critical questions, such as: Which non-functional requirements can be satisfied and, from those, which should be implemented first? Which of them are the main drivers for the software architecture design? What is the software architecture design style, or combination of styles, that best satisfies a set of quality attributes? Is there a way to help project managers to decide which requirements should be addressed (as the development spiral evolves and widens, it brings more requirements than those already identified)? The answer to such questions can help software engineers greatly to implement products, or applications, while ensuring better satisfaction of stakeholders' requirements. So far, only a handful of research works have addressed some of these questions, particularly on how the prioritization of a set of alternatives, or ranking, based on a set of criteria, helps software engineers to make informed decisions about software development [6,16,18,22,23,26,32]. Those

rankings have been used to help answering the questions above and others related with planning and managing team work, for example.

The aim of Multi-Criteria Decision Making (MCDM) methods [13,33,36] is to rank a set of alternatives (e.g., non-functional requirements, architectural styles) that best satisfy a given set of criteria (e.g., stakeholders, requirements). Although MCDM methods are widely used as supporting tools for decision making in several domains (see for example [7,15,13,18,28,32,33]), many have limitations. For instance, the well known AHP method [29,30,31] presents two main drawbacks: (i) its partial linear scale and its prioritization method — either the arithmetic average or the eigenvectors prioritization approach [5,11,20,33,34]; (ii) the high number of pairwise contributions, one per criterion matrix plus 1, just for a level-1 problem (e.g., for a problem with 6 criteria and 4 alternatives the user has to fill six (4,4)-matrices and one (6,6)-matrix). Hence, the main motivation for this work is to define a simple MCDM method to support software development decisions, while, at the same time, to provide trade-offs between criteria weights, which have a compensatory nature and avoid known limitations in terms of scale and ranking method.

In this paper we discuss HAM (*Hybrid Assessment Method*), an integrated MCDM method that combines one pairwise comparison matrix with one classical multicriteria decision matrix, hence requiring only 2 matrices for the reasoning process. In short, HAM's two major distinctive characteristics are:

HAM borrows the interesting pairwise comparison capability from the AHP method [14, 29, 30] to automatically determine criteria weights by using trade-offs between attributes (in our case requirements). However, it avoids the problematic Saaty scale and prioritization method by using Ma-Zheng's scale [20] and the geometric means of rows [5, 11], respectively.

HAM uses a classical weighted decision matrix [36], profiting from its simplicity and efficiency, but substitutes the weighted average by the geometric means to ensure uniformity and compatibility in the combined proposed method.

The main objective of HAM is to support software engineers to make better, well-thought decisions, by providing a systematic process to prioritize a set of alternatives based on a set of criteria. A preliminary version of HAM was first devised by the authors as a tool support for Software Product Lines [9], within the AMPLE project [2] and in [27], but its formalization, rationale and generality were not yet discussed nor published.

This paper is organized in eight sections. This first section summarizes the problem addressed. Section 2 motivates this problem by explaining its context and highlighting its importance for software development. Section 3 presents the background foundations for HAM. It starts with a taxonomy for MCDM methods, chooses four representatives, compares them according to a set of criteria, and finishes with a brief overview of the current state of the art of MCDM in Software Engineering. Section 4 details HAM, describing its procedural steps and their respective formalization. Section 5 illustrates HAM with an example, discusses its calculation process and compares the results of using the HAM's geometric means prioritization operator with the results using arithmetic means. Section 6 introduces HAM's supporting tool and Section 7 discusses the method's advantages and drawbacks. Section 8 concludes the paper presenting some final remarks, and highlights the generalization of HAM's usage for any decision-making domain as our goal for the near future.

2. Importance of Software Engineering decisions

Handling Non-Functional Requirements (NFRs) [33] (also known as quality attributes) is a complex task, commonly not (or at most superficially) considering the impact of different stakeholders' preferences on each other or on the choices required during development. For example, deciding on the list of requirements and qualities the system should satisfy may be difficult, never mind exploring one or more corresponding software architecture[1] designs, what requires a long and difficult period of refinement and fine-tuning. This hard task is usually driven by the system qualities, and the architects' expertise and experience. The final architecture may, therefore, never become something truly concrete on the developers' team minds, or even in the final system. Selecting a good architecture is a fundamental step of software development, but often it is "lost somewhere" among the various systems' modules resulting from designing the system.

We believe we can do better. We can provide a systematic process, making available the most appropriate information, techniques and tools to support the developers' choices. For example, if we are able to perform a rigorous trade-off analysis on the requirements, in particular on the system qualities — which are those that will have a stronger impact on the structure and therefore the system's software architecture — we will be in a position to make better informed decisions, even before we need to make any architectural choices [1,8]. Hence, based on the qualities identified (or a subset of interest) and eventual additional qualities, HAM should encompass to:

(1) identify the stakeholders' preferences and objectives (these objectives refer to system qualities) on what the system should do, providing a ranking of the qualities, taking into consideration the stakeholders' importance;

(2) import these qualities to the architectural phase and choose an architecture that tries to satisfy the ranking obtained initially;

(3) obtain a ranking of products that can be commercialized independently of the core, aiming at deciding what should be developed first, for example;

(4) allow software engineers to perform what-if analysis regarding changes in priorities for implementation of qualities, consideration of new candidate alternatives (e.g. other architectures), new architectural styles, new functional requirements, and so forth.

The first point tells us about the stakeholders' goals, and helps solving potential differences in opinion they may have in terms of the qualities the future system (or product, or family of products), should satisfy. A typical example is that security and response time are usually conflicting quality attributes, that is, if we need a strongly secure system, it may well be that the response time will increase. The qualities that conflict each other are fundamental information in a trade-off analysis (point 4). In such cases, we should look at the initial stakeholders' desires, and check which of the conflicting qualities is ranked higher. This ranking can, from thereafter, be used as a guide by the developers to try their best to satisfy the stakeholders' wishes. This, together with the other potential conflicting qualities, should guide the information to be passed over to the architect.

The second point is about having available trustworthy information from where architects can stem a first vision of the software architecture. System qualities are the criteria with higher impact on the architecture. Therefore, it is necessary to identify and prioritize those we need to address. The main

[1] Software architecture represents the structure of a system, which comprises software components, their externally visible properties, and the relationships among them [36]. The architecture gives structure to the system being developed by assigning responsibilities to components and how they should communicate.

difficulty the architects face is to choose the architectural alternative, or combination of alternatives, that best satisfies the set of chosen qualities. The choice of the wrong architectural style, or combination of styles, can bring serious problems throughout the software lifecycle, having a strong impact on the success or failure of the system. As explained in [1], different requirements may pull the architecture in various directions; each quality leads to a number of architectural choices, each one satisfying the system requirements and serving the stakeholder needs with varying levels of stakeholder satisfaction.

The third point addresses questions related with the identification and selection of new sellable components or products by transforming a software system into components and consequently unveiling new business opportunities. Or, in the case of a software product line, decide which product from a range of several products, should first hit the market. The ranking for potential candidate components or products to be developed may support decision makers in identifying new business opportunities with less risks and costs.

The fourth point gives developers the flexibility to play with different choices and understand their impact on the ranking and, consequently, on the decisions they have to make. This is very important, because, it also enables storing decisions and their respective rationale, which might be useful in future similar decision processes.

An example of a real application using the preliminary HAM is described in [16].

3. An overview of multi-criteria decision making

Multi-Criteria Decision Making (MCDM) methods use formal techniques to support decision makers to choose between a discrete set of alternatives [7,13,33,36]. This process is performed by analysing a finite set of alternatives with respect to a set of criteria. A criterion represents a rule on which a judgment has to be made. The alternatives, which represent the options available to the decision maker, have to be prioritized and classified using the judgments or contributions of each criterion.

Although multi-criteria problems can be very different and diverse, there are some common features to all of them [36], such as:

Alternatives: finite set of candidate solutions of a problem.

Criteria: each problem has a set of independent attributes that have to be satisfied by the alternatives.

Units: each criterion may be measured in different units, for example, kilos or meters; however, they all have to use a normalized classification being either a numeric scale or a qualitative scale (e.g., not important, important, very-important).

Weights: these represent the relative importance of each criterion. They can be directly assigned by the decision maker or automatically generated by a trade-off method.

Decision Matrix: the problem can be expressed in a matrix. Usually, the columns represent the criteria and the lines the alternatives. An element x_{ij} of the matrix indicates the degree to which alternative A_i satisfies criterion C_j

3.1. MCDM approaches

Usually, any MCDM method involves an initial process of eliciting the decision makers' preferences or judgments regarding contributions of alternatives to criteria as well as the respective weights associated with the criteria. Afterwards, MCDM methods use mathematical algorithms to

determine the prioritization of alternatives [36]. The logical principle of these methods is to rely on judgements of experts to derive the ranking of alternatives [33]. MCDM methods are used in a broad variety of problems, from financial applications, sports, e-business, software engineering, to selecting the best site for landing a spacecraft [1,7,15,18, 23, 26, 32].

There are many MCDM methods described in the literature and Figure 1 represents a general taxonomy, adapted from [7, 36], for the most well-known multi-criteria methods. This taxonomy depicts four categories of methods, each of which with at least two well-known methods in each category [27].

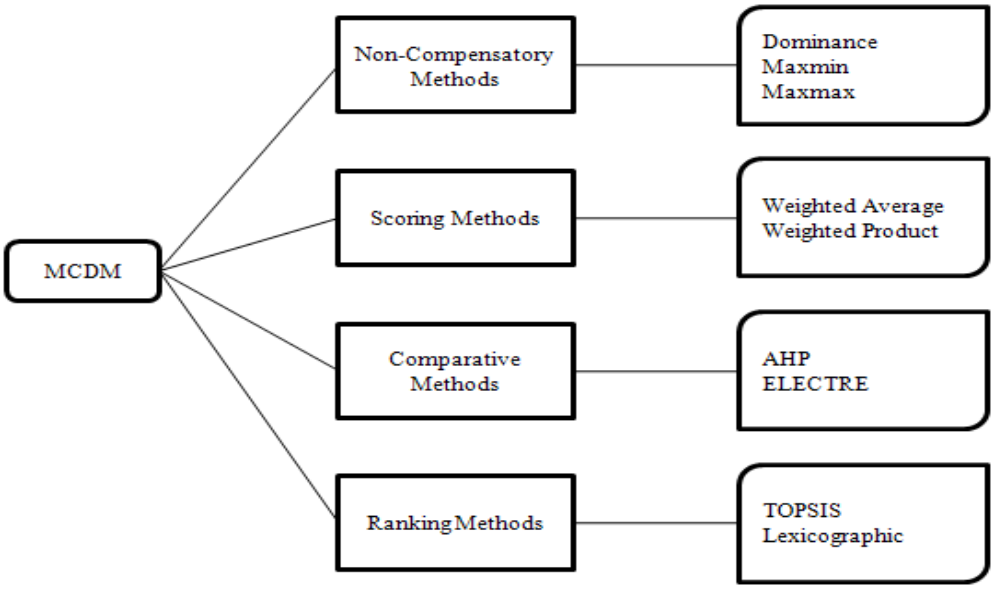

Figure 1. MCDM taxonomy (adapted from [8])

To perform a comprehensible comparative study of MCDM method's advantages and disadvantages to support software engineering decisions, but avoiding a lengthy comparison process (out of scope in this work), we picked-up a representative method from each category and compared them using five criteria. The rational for choosing the four methods was based on their widespread usage and record publications. For example, AHP has more than 1000 references!, and, as mentioned in [33], AHP, WA (Weighted Average) and TOPSIS are the most well known MCDM methods.

The following list describes the main characteristics of the four candidate methods for the comparison:

Maxmin [13,36] from the non-compensatory category. This method selects the alternative whose lowest score on the criteria is maximal It is a simple method using the logic "a chain is only as strong as its weakest link" [11]; for example, an astronaut's life or death in orbit may depend upon his/her worst vital organ. This method main advantages are the use of a pessimist attitude (less risky) and the inclusion of very simple calculations. Its three main disadvantages are: not to allow judgements among criteria; not to provide compensatory behaviour between criteria; to choose an alternative solely based on satisfaction of one single criterion.

Weighted Average (WA) [13,33,36] from the Scoring Methods category. The WA logic is that the final score of alternatives is obtained by the weighted sum of each criterion satisfaction value. WA is the most well-known and widely used MCDM method because it is simple, easy to use and understand, and allows some compensatory trade-offs among criteria with their weights. Furthermore, any spreadsheet can act as tool support since it only requires one decision matrix. However, it does not offer pairwise

comparisons or logical consistency checking. Hence, no dependencies or relationships between criteria can be expressed.

TOPSIS (Technique for Order Preference by Similar to Ideal Solution) [33, 36] from the Ranking Methods category. This method defines the order of the alternatives by their distance to an ideal solution. The basic principle of this method is to find a solution with the shortest distance from the positive-ideal solution and the biggest distance from the negative-ideal solution. It uses the Euclidean distance (or any other) to calculate the shortest distance. This method [36] is easy to understand and it ensures that the trade-off among attributes is compensatory. However, it is requires more calculations and its major drawback is requiring the a-priori definition of two parameters (the positive and negative ideal solution); thus, it is a context-dependent method.

Analytic Hierarchy Process (AHP) [11,29,30,31,33] from the Comparative Methods category. It is one of the most widely known methods offering pairwise comparison trade-offs. AHP relies on expert judgements to derive priority scales (ranking of alternatives). The comparisons are made using a scale of judgments that represent how much more one element is important than another with respect to a given criterion. AHP also provides a measure for detecting logical inconsistencies for these judgements. Nonetheless, its major disadvantages are the inconsistencies introduced by the linear scale and the need for a large number of pairwise comparisons (e.g., considering 4 criteria and 5 alternatives the user has to provide 50 judgements!), which makes it difficult to analyse results for each single matrix. In summary, AHP is low in efficiency and in usability. Besides lacking efficiency, AHP has consistency problems imposed by the used linear scale [1 .. 9], as pointed out by Triantaphyllou and Dong [11, 33].

3.2. Comparison of MCDM methods

The methods briefly described in the previous subsection were compared to assess their main strengths and weaknesses regarding their suitability for software engineering decision support. We have used four criteria for this comparison, whose choice was based on relevant features for software engineering decisions:

Efficiency: this criterion assesses if the method is time-consuming in terms of processing and implementation.

Usability/understandability: this is a quality attribute that assesses how easy user interfaces are to use. The word “usability” in this context also refers to how easy it is to understand the method (by any novice MCDM practitioner such as software engineers (our target users)).

Pairwise comparisons: it is proven by psychological studies that it is easier to compare elements two-by-two (pairwise) to determine their relative importance (weights), instead of assigning importance directly to elements [14]. This feature is particularly interesting to perform trade-offs between criteria in software engineering where we need, for example, to achieve a decision between non-functional requirements or even reaching consensus between stakeholders.

Compensatory: this criterion checks if the method allows some compensatory behaviour between criteria, that is, if all criteria contribute to the final result.

Table 1. Comparing MCDM methods

	Efficiency	Pair-wise Comparisons	Compensatory	Usability	*Final Score*
Maxmin	2	0	0	2	1.00
WA	2	0	2	2	1.50
TOPSIS	1	0	1	1	0.75
AHP	0	2	2	1	1.25

Based on the criteria list above, we compared the four chosen MCDM methods, using a simple discrete scale with three levels (0 – Bad; 1 – Reasonable; 2 – Good) and assuming that any method would require the satisfaction of all criteria, meaning that there should not be a criterion with "bad" classification. Table 1 shows the scores provided by the authors to each criterion and the final aggregated score using a simple average.

Considering the characteristics of each candidate method, we classified their contributions for each criterion. For example, considering that Maxmin is a MCDM method with a very simple logic, its efficiency and usability are good; therefore the assigned value was 2. Another example is pairwise comparisons: since AHP is the only method in the set of candidates with this feature, we classified it with 2 and the others with 0.

Table 1 indicates that WA is the best candidate with a total score of 1.5. However, since none of the methods satisfy all criteria ($C_{ij}>0$) and the winner only lacks the pairwise comparison criterion, we decided to integrate WA and AHP (the best 2 candidates) and the result is the proposed Hybrid Assessment Method (HAM). Details about the proposed method are discussed in Section 4.

3.3. Multi-Criteria methods in Software Engineering

MCDM methods are not new to the Software Engineering world (e.g. [1,4,15,19,35). There are several situations where these methods have been used to solve different types of problems, in particular to identify and prioritize conflicting Non-Functional Requirements (NFRs) [8,35], or even just prioritizing requirements in Requirements Engineering [4,21]. Until recently, efforts have been focused on solving conflicts between aspects for stakeholders, using more formal ways to deal with the issues at hand, and incipient usage of MCDM methods was also proposed to create rankings of concerns, both aspectual and non-aspectual [4, 6,16,27].

As mentioned before, a typical example of a conflict, many times mentioned in the literature, involves the NFRs security and response time, where, on one hand, the system needs to be secure and, on the other hand, its response time needs to be minimized [6,24]. Security and response time contribute negatively to each other since more security measures imply higher response time [8]. Thus, it is not possible to simultaneously maximize security and minimize response time, since they are both in conflict.

The motivation for developing HAM is to provide support for the following software engineering decisions:

- Provide a prioritization (ranking) for non-functional requirements (NFR) to help software engineers to decide which qualities are more important to be satisfied and, in case of conflict, which should be prioritized over others;
- Recommend the best architecture by ranking architectural alternatives considering the stakeholders opinions;
- Allow "what-if" analysis to perform system behaviour analysis by simulating changes to it. For example:
    i. What happens if a new stakeholder or NFR is added to the system? Does the NFR rank change? What is the impact of adding new NFRs to the system?
    ii. What is the impact on the architecture if a new NFR is added? Will the architecture suffer? Will this "disturb" the current balance of the architecture, or change the implementation order?

iii. What happens if we change the contribution value of a given architecture to an NFR, for example? Will this affect the final ranking of the architecture styles?

However, this work focuses on HAM's model and algorithm and not on its functionalities to support the software engineering decision process. An in-depth discussion about its functionalities regarding answering the above questions and other software engineering decisions is left for more specialized papers on the topic.

4. Hybrid Assessment Method (HAM)

HAM is a two-phase process, composed of five steps in total, as shown in Figure 2 [27]. The first step is to elicit the criteria and the alternatives. The second step performs trade-offs between criteria using pairwise comparisons. The third step is to calculate the criteria priority vector, normalize the respective weights and calculate the consistency ratio. If the consistency ratio is under 10% (or <8% if the matrix has less than 5 criteria) [31,33], the logical consistency of the pairwise matrix is considered to be sufficient. If it is 10% or higher, the values should be revised until the consistency ratio becomes less than 10%. These three steps belong to Phase 1 of our process and, roughly, correspond to an automated determination of weights for the decision matrix.

Phase 2 starts in step four, which elicits the contributions of each alternative with respect to each criterion, using a classical WA's decision matrix. The fifth and final step is an aggregation process to determine the prioritization of alternatives (ranking) using a geometric aggregation operator [11]. This step concludes our process by providing the ratings for each alternative.

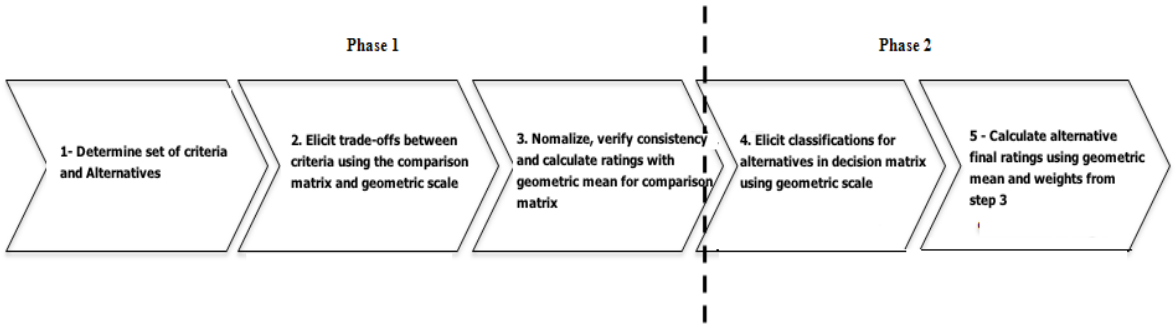

Figure 2. HAM step by step

4.1. HAM Scale

HAM uses the Ma-Zheng [20] scale because, according to [11,33], it is more appropriate for the kind of problems we are handling. For instance, Triantaphyllou et al. [33] performed an exhaustive study considering the choice of scale as a typical multi-criteria problem, where the criteria are: **inversions** and **indiscriminations**. Inversion happens when we know the real ranking of alternatives and the method being tested ranks them in a different order. For example, the real ranking of three alternatives is equal to (1> 3> 2) and the tested method result is (1> 2> 3). Indiscrimination happens when the tested method yields a tie between two or more alternatives, for example, if the calculated result is (1> 3 =2). Furthermore, Triantaphyllou tested 78 different scales, classified as Class 1, Class2, …, Class 78, which depict different performances in terms of the two above mentioned criteria and also on the number of criteria used.

For HAM we observed that the best class of scales is Class 1, since it covers the majority of sizes of sets of criteria and considers equal weights for the criteria. For software engineering problems, inversion and indiscrimination are equally important. Note that Class 1 includes the selected MA-Zheng scale, shown in Table 2.

To better clarify our selection we also performed a comparative behavioural study between three well-known scales (depicted in Figure 3): the Saaty scale, an exponential scale (with c=2 to make it comparable with the other scales), and the Ma-Zheng scale. The Y-axis depicts the normalized scale values and the X-axis presents the interface scale used in HAM, to ensure user-friendliness in the dialogue with the user. Table 2 shows: the interface scale (used to facilitate the visualization/interaction process); the Ma-Zheng scale (used in HAM selected internal calculations); the Linguistic interpretations (for all scales discussed); and the AHP scale (Saaty). As can be observed in Figure 3, Saaty´s scale has a quasi-linear behaviour, what may cause consistency problems. Moreover, it is not discriminative enough in the positive ranges (from extremely important to equal importance) with respect to their reciprocals, since they should depict a more similar inverse behaviour.

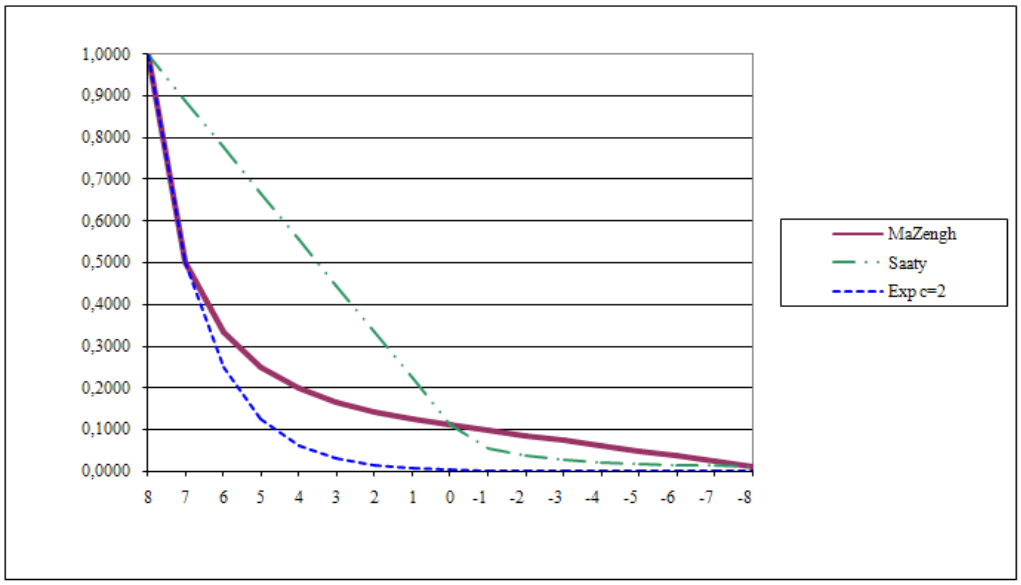

Figure 3. Scales comparison

Comparing the Ma-Zheng scale with the exponential one we observe that in the interval [1 .. 1/9] the Ma-Zheng scale (Table 2) presents quasi-evenly distributed values and in the interval [9 ..1] an almost exponential behaviour. According to [11] the Ma-Zheng scale is a good choice and since it depicts a smoother behaviour for smaller importance/preferences than the exponential scale with c=2 for comparative purposes with the other scales (Figure 3), hence we selected the former.

In addition, it should be highlighted that we used a more intuitive numerical scale in the interface (Column 1 of table 2) so that the decision maker does not have to know details about the less user-friendly Ma-Zheng scale. Another interesting aspect in the interface scale, is how we depict the inverse rating values, $c_{ji} = 1/c_{ij}$ (both used by HAM and AHP), since we represent them with negative values to facilitate the user dialogue with the software system. For example, if an alternative criterion is rated with "high importance" (value 4 in column 1 of Table 2), when we want to express its opposite, i.e., "low importance", we use its negation (i.e., -4). Another example is that for equal importance we believe that using 0 to express "no" preference, instead of 1, allows a more intuitive classification for users. The

interface developed also includes the linguistic interpretation to better guide novice users. Notice, however, that we use the Ma-Zheng scale for HAM calculations.

Table 2. HAM Interface rating values, HAM Scale, interpretation, AHP comparative scale

HAM Interface numerical rating	HAM Scale (calculations done with Ma-Zheng)	Linguistic Interpretation	Comparative AHP scale (Saaty)
8	9/1	Extremely High importance	9
6	9/3	Very High importance	7
4	9/5	High importance	5
2	9/7	Medium High importance	3
0	9/9	Equal importance	1
-2	7/9	Medium Low importance	1/3
-4	5/9	Low importance	1/5
-6	3/9	Very Low importance	1/7
-8	1/9	Extremely Low importance	1/9
Intermediate values	Intermediate values	Intermediate values	Intermediate values

4.2. HAM Computations

This section specifies a stepwise process of the computations performed by HAM, as described in Figure 2. The first step of Phase 1 is to identify the criteria and the alternatives. Let *nc* be the number of criteria and *na* the number of alternatives. The second step elicits the decision maker judgements for the relative importance of each pairwise comparisons; these importance are the elements of the criteria matrix *c* of dimension *nc*. For all i and j with $1 \le i < j \le nc$ the i[th] and j[th] criterion are compared, leading to the value c_{ij} using Table 2, second column (HAM scale). The remaining entries of the matrix are determined by $c_{ji} = 1/c_{ij}$, i.e. the inverse function. In the third step, HAM calculates the priority vector for the criteria, which represents the weights for Phase 2, using the geometric mean. The geometric mean has been selected for the prioritization method for pairwise matrices, following the conclusions of Dong [11] about its superior suitability over Saaty´s eigenvalue method. Thus, the geometric mean is given by:

$$w_i = \sqrt[nc]{\prod c_{ij}} \tag{1}$$

Normalizing this vector, we obtain the vector NW of normalized weights:

$$nw_j = \frac{w_j}{\sum w_j} \tag{2}$$

The next step is to compute the consistency index *CI* of the *c* matrix criteria [16], which is defined as:

$$CI = \frac{\lambda - nc}{nc - 1} \tag{3}$$

where λ, instead of being the largest eigenvalue of *c* [30,33], is an average of the normalized weighted vector (i.e., normalized priority vector *nw* in equation (2)) divided by the normalized priority vector *nw*, as described in [3, 31]. That is, this calculation is performed by multiplying the pairwise matrix by the *nw* vector and then divides each obtained vector value by its respective *nw* vector value. Then, this consistency index must be compared with a Random Index (*RI*), which is the average of the consistency indices of random reciprocal matrices. For the Ma-Zheng scale we used 1000 reciprocal matrices to calculate the *RI*s in Table 3.

Table 3 . Random Index (RI) values for the Ma-Zheng scale and for Saaty´s scale

Number of Criteria	**2**	**3**	**4**	**5**	**6**
HAM RI for HAM (Ma-Zheng scale)	0	0.3416	0.5508	0.7008	0.7962
AHP RI (Saaty' scale) [28]		0.58	0.9	1.12	1.24

Next, the consistency ratio (*CR*) is computed by:

$$CR = \frac{CI}{RI} \tag{4}$$

Finally, to assess the logical consistency of the reciprocal matrices, *CR* should be less than 10% for reciprocal matrices with size n>4; less than 8% for (4,4) matrices and less than 5% for (3,3) matrices [31]. This concludes Phase 1 of the HAM algorithm.

In Phase 2, the contributions of alternatives to each criterion (step 4 of the HAM process, or step 1 of Phase 2) are again elicited from the decision makers. For each 1<=i<=*na* and 1<=j<=*nc* the matrix element a_{ij} of the decision matrix *a* is determined with Table 2.

The aggregation method to determine the prioritization of alternatives in the fifth step of the HAM process, is again the calculation of the geometric means but using the obtained normalized priority vector of phase 1 for its weights, such as:

$$r_i = \prod_{1}^{n} a_{ij}^{nw_j} \tag{5}$$

Then the obtained *r* vector is normalised, giving the rated alternatives:

$$Rate\_Alt_i = \frac{r_i}{\sum r_i} \tag{6}$$

The decreasing ordered alternatives show the final ranking, where the best decision alternative is the one with the highest *Rate_Alt* value.

5. Illustrative Example

This section presents an illustrative example for selecting an architectural style in software engineering development. For the example we consider four criteria and three alternatives. The criteria are: usability, performance, security and modularity. The candidate architectural styles (alternatives) are: MVC, Backboard and Client-Server. Since this is an example, the elicited values for the matrices (Phase 1 and Phase 2), are just illustrative and do not represent opinions of the software engineers. In the implemented tool the user dialog is done with the simplified interface scale, presented in Table 2, but

behind the scenes the calculations are performed using the Ma-Zheng scale. From the user perspective (HAM interface scale) the illustrative example is summarized in Table 4.

Table 4. Illustrative example with Ham interface scale

Phase I

	Usability	**Performance**	**Security**	**Modularity**
Usability	0	-5	-6	-2
Performance	5	0	1	6
Security	6	-1	0	5
Modularity	2	-6	-5	0

Phase 2

	Usability	**Performance**	**Security**	**Modularity**
MVC	4	4	4	4
Blackboard	-6	-4	4	6
Client-Server	4	3	6	1

Note that in Phase 1 the user only needs to provide the values for the cells above the diagonal line. The diagonal is filled with zeros (by default) and HAM automatically adds the values below the diagonal. In Phase 2 the user is required to classify each alternative regarding each criterion using the HAM interface scale.

The following subsection presents the results obtained with the computational model proposed for HAM (Section 4), which, as mentioned before, uses the Ma-Zheng scale and the geometric mean as the aggregation operator for both phases of HAM. The second subsection compares the results with an approach using the weighted average in both phases. The third sub-section compares the results of using HAM selected scale with the same problem using the AHP scale. In these sub-sections we will show the illustrative example already with the appropriate calculations scales (either the selected HAM scale or the AHP scale), according to Table 2.

For calculating the consistency index CR, we used the HAM RI from table 3 for nc= 4, i.e., RI= 0.5508; for the comparison with AHP we used RI=0.9 (Table 3).

5.1. HAM approach

Phase 1: Pairwise comparison to determine criteria weights (importance).

After eliciting the pairwise contributions (just for the values above the diagonal) HAM fills the pairwise matrix with their reciprocals. Then we are able to calculate the weights (eq. 1), normalized weights (eq. 2), and the consistency (equations (3) and (4)), as shown in Table 5. This phase uses a pairwise matrix to automatically obtain the weights for each criterion (quality attribute).

The left matrix represents the pairwise matrix using the Ma-Zheng scale, selected for HAM. On the right side we have the calculations for W (eq. 1) and also for NW (eq. 2). The last column, on the right side, depicts the calculations for λ (eq. 3), which amounts to the multiplication of the pairwise matrix

with the NW vector and then the division of each obtained vector value by the respective NW vector value. Details of these calculations for λ can be found in [16].

Table 5. Phase 1: Example for HAM scale and geometric prioritization

	Usability	Performance	Security	Modularity		Geometric W	Geometric NW		Geometric λ
Usability	1	4/9	3/9	7/9		0.583	0.130		4.044
Performance	9/4	1	9/8	9/3		1.660	0.372		4.044
Security	9/3	8/9	1	9/4		1.565	0.350		4.009
Modularity	9/7	3/9	4/9	1		0.661	0.148		4.024
					Total	**4.468**	**1.000**	**CI =**	**0.010**
								CR =	**1.84%**

As can be observed, the logical consistency is ensured since CR< 8%, which is the recommended value for small matrices [31].

Phase 2: Decision process to determine alternatives ratings

In this phase, again, after eliciting the contributions of each alternative for each criterion (shown in Table 4), we calculate the ratings (eq. 5) and the normalized ratings (eq. 6) for each alternative, using the HAM selected scale for calculations (Table 3). This phase uses a classical decision matrix, where lines depict alternatives and columns depict criteria, and the prioritization operator is also the geometric mean (see Table 6).

Table 6. Phase 2: Example for HAM

Weights	0.1337	0.3467	0.3474	0.1723		Geometric	
	Usability	**Performance**	**Security**	**Modularity**		**Ratings**	**Rate-Alt (%)**
MVC	9/5	9/5	9/5	9/5		1.800	38.781
Blackboard	3/9	5/9	9/5	9/3		0.965	20.787
Client-Server	9/5	9/6	9/3	9/8		1.877	40.432
					Total=	4.687	

The best candidate is the software architecture Client-Server with a rating of 40.432%, followed by MVC with 38.781%. The worst choice is clearly Blackboard with only 20.787%. In such a situation, and depending on the problem being developed, we could use a simple Client-Server architecture style, or use a combination of Client-Server and MVC, where MVC could be used on the server side, for example.

5.2. Comparison of geometric means vs arithmetic means as prioritization operators

In this section we also use the Ma-Zheng scale, but instead of the geometric means prioritization operator we use the simple weighted average for the calculations, both for Phase 1 [3,31] and Phase 2 [13,33,36]. The results obtained are depicted in Table 7.

Table 7. Example with HAM scale and weighted average prioritization in both phases

Phase 1: Pairwise comparison to determine criteria weights

Arithmetic				
	Weights	NW		**Consistency**
Usability	0.639	0.127		4.146
Performance	1.844	0.366		4.115
Security	1.785	0.355		3.958
Modularity	0.766	0.152		3.911
Total	5.033	1.000	CI =	0.011
			CR =	1.96%

Phase 2: Decision process to determine alternatives ratings with weighted average

Arithmetic		
	Ratings	Rate-Alt (%)
MVC	1.800	35.719%
Blackboard	1.226	24.337%
Client-Server	2.013	39.944%

As can be observed, the ranking of alternative architectures (1st- Client-Server, 2nd- MVC, 3rd- Blackboard) is the same as the one obtained with the geometric approach, as should be expected for such small illustrative example and consideing we are using the same geometric scale. However, visible differences can be spotted in the respective rating values:

- geometric: 40.432, 38.781, 20.787
- arithmetic 39.944, 35.719, 24.337
-

Phase 1: the importance for criteria (i.e., requirements) in Table 7 shows that the most important requirement is "performance" followed by "security" (respectively 0.366 and 0.355 in column NW). Now, comparing these values with Table 5, column NW, geometric approach, we detect a bigger difference between the two criteria (0.372 and 0.35), clearly showing that the geometric approach is more discriminative. However, when the comparison values are low (e.g., usability compared with modularity has a medium low importance = 7/9) then the arithmetic approach discriminates more 0.127 versus 0.152 in Table 4, column NW), which is against what we want, i.e., to be more discriminative for high values and less discriminative for lower values. In addition, notice that the values for the consistency index CI vary, and that the logical consistency of the geometric approach is better, since it presents a lower value (1,84% versus 1.96%).

Phase 2: similarly, the same logic is observed in the ratings of the two best alternatives (Client-Server and MVC). However, in this phase, the geometric approach clearly distinguishes when an alternative is

rather bad (Blackboard obtains 20.787% and 24.337%, respectively in the geometric versus arithmetic approach). Another interesting point is that in the geometric approach there is not so much distinction between the MVC and the Client-Server architectures, which allows (as it should be) the software engineer to consider their rating, instead of just discarding the MVC architectural style (as it was suggested by the big difference of results given by the arithmetic approach).

In summary, we can say that the geometric operator offers more balanced results for higher values and more discriminative results for lower values. The logic is that if an alternative is clearly bad it should be immediately discarded (case of the blackboard architectural style), but, if it has some potential, it should be further analyzed by the software developers (case of Client-Server versus MVC). In the latter case, the decision maker can use the flexibility offered by HAM and perform a "what-if" analysis to assess which of the two candidates is better, Client-Server or MVC. Of course, this would not rule out the hypotheses of making a combination between two or more architectural styles, but if a trade-off was necessary, the developer has a more clear distinction between the importance's of each architectural style. This obliges him/her to think twice about the path to take, making ad-hoc decisions harder to justify. So, once more, what-if analysis gives him the flexibility to try different combinations of quality attributes, adding some, removing others, or changing their importance-values, until a consensus is reached among stakeholders. On the other hand, if the developers do not want to perform further analysis, they can always decide in favour of the "best" classified by HAM, which was the Client-Server alternative.

5.3. Comparison with the AHP scale

This section compares the same illustrative example but using the AHP scale (see Table 2) and, as prioritization method, the weighted average, which is a simpler equivalent version of the eigenvector approach [3, 30].

Table 8. Phase 1: Example with HAM scale and weighted average prioritization

Phase I- equivalent AHP scale values (Table 2)

	Usability	*Performance*	*Security*	*Modularity*
Usability	1.00	0.17	0.14	0.33
Performance	6.00	1.00	2.00	7.00
Security	7.00	0.50	1.00	6.00
Modularity	3.00	0.14	0.17	1.00
Total	17.00	1.81	3.31	14.33

Phase I –AHP results for quality attributes weights

	NW		**Consistency**
	0.054		**4.034**
	0.500		**4.378**
	0.352		**4.388**
	0.094		**4.122**
		CI =	**0.077**
		CR =	**8.543%**

The results obtained for Phase I (determination of weights for the quality attributes) using the AHP scale are illustrated in Table 8. These results demonstrate clearly the superiority of the HAM scale as the consistency index (CR) is now above the logical threshold of 8% (8.543%) (recommended for small matrices by Saaty [31]!) Therefore, using AHP scale can distort results that are cognitively consistent and logical because of its lack of discrimination power, which may cause an almost random behaviour. (The scale appropriateness was discussed in Section 4.1.)

Regarding Phase 2 (results in Table 9) another interesting point to observe is that this same lack of discriminative power when using the AHP scale reflects on the final ranking since now the "best" alternative is the MVC followed closely by Client Server. Clearly the AHP scale is also good for discriminating "bad" alternatives (where its behaviour is closer to the Ma-Zheng scale, used by HAM) but for good alternatives it can have unexpected behaviours as it only follows a linear averaging logic.

Table 9. Phase 2: Example with HAM scale and weighted average prioritization

Weights	***0.054***	***0.500***	***0.352***	***0.094***		**Arithmetic**	
	Usability	**Performance**	**Security**	**Modularity**		**Ratings**	**(%)**
MVC	**5.00**	**5.00**	**5.00**	**5.00**		**5.000**	**40.469%**
Blackboard	**0.14**	**0.20**	**5.00**	**6.00**		**2.432**	**19.683%**
Client-Server	**5.00**	**4.00**	**7.00**	**2.00**		**4.923**	**39.848%**
						12.355	**1.000**

Note that the lack of consistency in the AHP scale is clearly visible in this small illustrative example and for our purposes, to support software engineering decisions, it is rather inferior to the scale and prioritization method selected for the HAM method and tool.

6. Tool support

This section describes the preliminary prototype tool that was developed for the HAM method. The tool's main window is displayed in Figure 4. The example shown therein considers four criteria (stakeholders of the software system: company, user, administration, architect) and four alternative requirements to be assessed: safety, portability, extensibility, usability and interoperability. This example was used to show the potential of HAM to also support decisions about priorities for implementing requirements, taking into consideration preferences and importance of the involved stakeholders.

Considering the top-half of the window, we can observe, on the left-hand side, a tree-like structure with the goal of the assessment and the selected criteria and alternatives. In the middle there is the pairwise comparison matrix. This is where the tradeoffs between the criteria will be done. We just need to fill the

right-upper side of the pairwise matrix (using the interface scale of Table 2). The lower-left cells will be filled automatically. The tool processes all calculations using Ma-Zheng's scale, as discussed in previous sections. On the right-hand side is a vector that will display the calculated weights of the criteria (normalized). If the criteria weights are inconsistent, a message pops-up warning the user. Under the pairwise matrix we can see a combo box with the Assign Value. This component is used to assign values to the pairwise matrix cells using Table 2 interface scale. In the bottom-half of the window, we can see a short scale help, so that the user has quick access to the scale. To fill both decision matrices we can use the mentioned combo box or use the keyboard. Finally, on the right-hand side the rating vectors are shown, where the ratings of the criteria (top) and the alternatives (bottom) are displayed. The ratings are displayed in absolute percentage.

When finished filling the pairwise matrix, we press the "Calculate Weights" button. You can view the criteria normalized weights (i.e., priority vector) in the weights vector. As explained in Section 2, if the consistency ratio is greater than 8% (for matrices of 4*4 or lower) and for greater ones > 10%, the user is prompted with an error message and adjustments must be done in the pairwise comparison matrix to ensure the comparisons are logical. When performing any what-if analysis the user can also edit the pairwise matrix by doing "Edit → Matrix to fill → Pairwise Matrix" and press the "Calculate Weights" button. To go back to editing the decision matrix, just do "Edit → Matrix to fill → Decision Matrix".

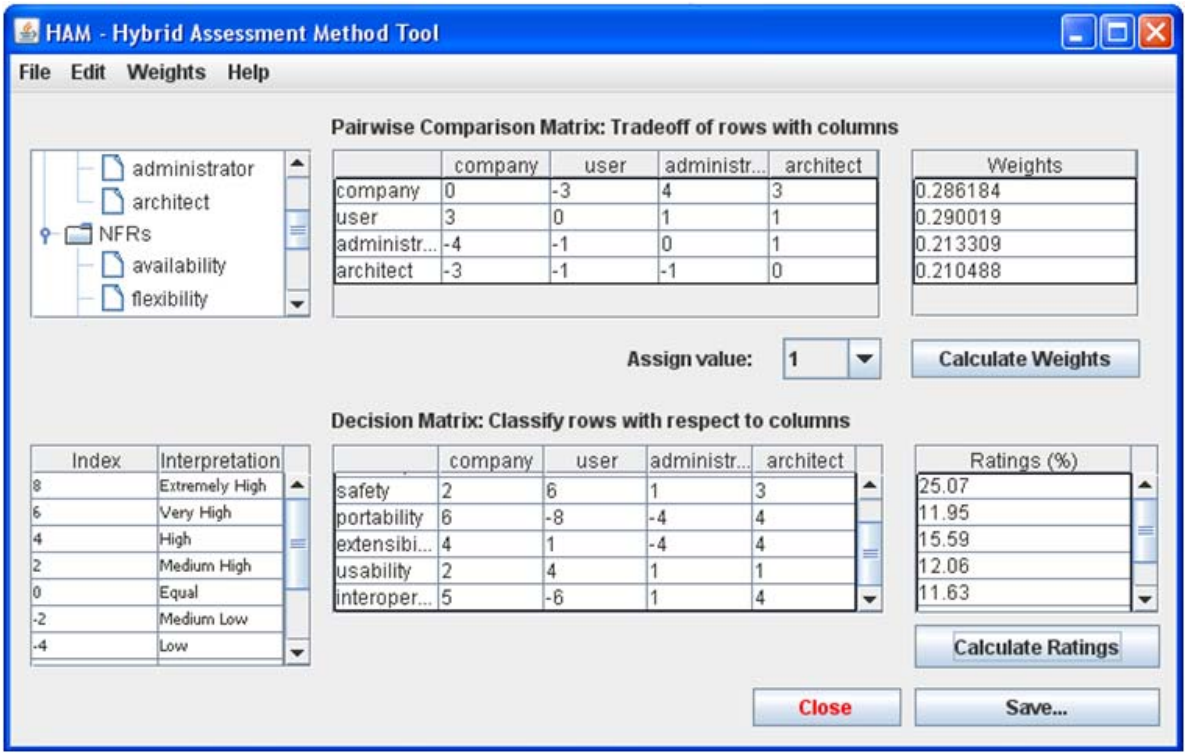

Figure 4. HAM Tool main window

Once phase 1 is finished, pressing the "Calculate Ratings" button will provide the importance/weights for criteria. The criteria ratings are displayed as a weight vector, as shown on top right of Figure 4.

These partial results can also be saved by pressing the "Save…" button, or going to "File → Save…", which will open a save dialog box. Just select the location where you want to save the project and the name of the file. This will create a new Microsoft Excel file with the name you provided, or a new sheet in the file if it exists.

The tool also allows the addition of new criteria, for example a new Stakeholder, or new alternatives, for example a new NFR, to the existing analysis. For instance, to add a new criterion just do "Edit → Add new criterion". A new window opens and you can enter the name of the Stakeholder you want to add. This allows what-if analysis.

The process for phase 2 is somewhat similar, with the difference that there are no pairwise comparisons and the values elicited are direct judgements of users regarding each criterion for each alternative. This matrix must be completely filled by a decision maker or domain specialist, e.g. a software engineer. After filling this decision matrix the user must press the "Calculate Ratings" button and the final rating per alternative is displayed in the ratings vector (bottom right).

The tool was developed in Java 1.6 [31] using Eclipse IDE 3.4.1 [32] to implement the HAM algorithm and NetBeans IDE 6.5 [33] to develop the Graphical User Interface.

7. HAM Discussion

Note that the normalizations of *w* and *r* are not strictly necessary. They are performed since normalised vectors are easier to understand, particularly in the common percentage scale. Note also that the columns of the pairwise decision matrix are not normalized, as they are in the AHP method. This is because the values of the decision matrix are dimensionless quantities. As a consequence, HAM can use the weighted geometric method without suffering from the anomalies of the AHP method, where, for instance, when an alternative is added, the ordering of the other alternatives may change. Moreover, the consideration of the same scale and the geometric mean, in both phases of HAM, provides consistency, flexibility and robustness for the proposed approach, as explained in section 4.

By using the geometric mean and avoiding the problematic Saaty scale we managed to achieve a smoother behaviour for the decision maker judgments and the user-friendlier interface scale used in the HAM tool offers a more transparent usage. Also, by using a two-phase approach, we are able to automatically calculate the weights for the criteria, to perform trade-offs in both phases and to avoid the extra time needed for the complete Saaty's approach. We simplified the user interaction by displaying a user-friendly interface scale to facilitate novice users to elicit judgements about relative importance of criteria.

In summary, HAM´s method's main characteristics are:

a. From comparative methods, HAM uses just one pairwise comparison matrix to perform the trade-off analysis between criteria and obtain their relative importance (or weights), while ensuring logical consistency.
b. From scoring methods, HAM uses the decision matrix logic of WA with weights obtained from the AHP pairwise comparisons to determine the final ratings and ranking for the alternatives.
c. Since the AHP scale and prioritization method suffer from some flaws, such as its linear scale and prioritization operator [10, 17], we included Ma-Zheng's scale [20] and the geometric means as ranking/prioritization method, as suggested in [17].
d. To ensure consistency and uniformity in HAM, we also used the Ma-Zheng scale and the geometric means in the second phase for determining the final ranking.

As any other MCDM method, HAM has advantages and drawbacks, particularly for supporting software engineering decisions. HAM's advantages are:

- It is a consistent, robust and flexible method.
- It allows trade-offs between criteria importance (e.g., NFRs) resulting in the automatic determination of their relative importance (weights).
- It provides sound ratings of alternatives (e.g., architectural choices) by using both a scale and an aggregation operator, scientifically proven as superior to the base constituting methods of HAM.
- It provides a user-friendly interface, enriched by using a simple interface scale as the dialog media with the users.

- It enables the addition of new criteria and/or alternatives at any time, obviously at the expense of having to perform new calculations.
- By combining a scoring method with a comparative method (Figure 1) we avoid the time-consuming problems of comparative methods and profit from having trade-offs for criteria weights importance.
- It allows consistent what-if analysis (e.g., what happens if we add a new requirement to the system, or if we change the values of importance or contribution? Will this change the suggested architecture?).

And the drawbacks of HAM are:

- As any other MCDM method, judgments made by experts (assigned values) always include judgments and HAM only ensures logical consistency for pairwise judgments (comparative method phase).
- As any other MCDM method, it is a decision support model and not a decision making model, not substituting decision makers, and only supporting them to make educated decisions.
- It does not support group decision making (consensus about contribution values for each criteria must be done outside the method).
- Of course, when we are dealing with specific problems, such as software engineering decisions, HAM requires technical knowledge about the context domain for providing judgements for relative importance of criteria, as well as to provide evaluations for the alternatives.

8. Concluding remarks

Our goal was to define a multi-criteria method to assist software engineers in their decisions through the early stages of the software development life cycle. We studied several classical multi-criteria methods, and concluded that none of them fully satisfied our needs, in particular in terms of efficiency, pairwise comparisons, compensatory behaviour and usability. During this study, we revisited the flaws regarding scale and aggregation method in the mathematical background of the two most promising methods to use (from the classes of scoring and comparative methods). To address this deficiencies, the *Hybrid Assessment Method* (HAM) was proposed.

HAM is a hybrid method as it combines a comparison method with a scoring method: it uses only one pairwise comparison decision matrix and combines it with one classical weighted decision matrix. It uses a geometric scale and the geometric mean for determining the priority ratings. Moreover, it can be applied to different types of problems, even though here our motivation was to support software engineering decisions.

Until recently, the importance played by system qualities in software development was dubious. That is, everyone knew and knows how important they are; but treating their influence on each other as well as their impact on the developed software was still a cumbersome and informal process. Most often, each quality was added to the code, without a clear view of the influence such new scattered code (as usually a system quality has an impact on several modules of the system) really had on the final software. And the more qualities are handled in this fashion, the more likely it is to increase severely the maintenance difficulties and evolution complexity. Therefore, being able to study each quality independently first and then their impact on each other and on other modules of the system is a great step

forward on modularization and, consequently, on software maintenance and evolution. HAM offers support for this type of study.

Indeed, HAM supports the identification the stakeholders' goals, these corresponding to system qualities, and is able to rank these qualities taking into consideration the stakeholders preferences. Software developers, in particular software architects, can then consider these qualities as the main driver to design the system's software architecture. But HAM also addresses questions related with identification or selection of new sellable components or products supporting decision makers in identifying new business opportunities with less risks and costs. Finally, HAM gives developers the flexibility to play with different choices and understand their impact on the ranking and, consequently, on the final decisions they have to make.

Moreover, HAM offers the above functionalities always ensuring consistency, efficiency, pairwise comparisons (criteria trade-offs). Moreover, it guarantees logical consistency, provides automatic calculation of criteria weights and its reasoning process is reasonably straightforward. The prototype-supporting tool we developed is a proof-of-concept and validation or our claims. We believe HAM is a consistent, robust and flexible method, particularly when compared with other existing ones. As future work we will continue improving the tool to generalize its usage for any decision-making domain.

To conclude, we will now address HAM's suitability and appropriateness for software engineering decision support. HAM provides a systematic rigorous method to support software engineers' decisions when difficult alternative choices come into play. Therefore, HAM aims identifying the stakeholders' preferences and goals (the latter refer to system qualities) on what the system should do. To accomplish this, HAM allows us to rank system qualities taking into consideration the stakeholders' importance. If some of these qualities contribute negatively to each other [8], as the example given in Section 2 for response time and security, then we can identify that earlier in the software development process. Moreover, since system qualities are the main software architecture drivers, having a ranking of the qualities according to the stakeholders' wishes, and knowing which of those qualities are in conflict, is very valuable information to design the first draft of the software architecture. But HAM offers other useful utilities for software engineers and enterprise managers. For example, suppose that a company wants to launch in the market a range of new products. HAM can help the company manager to decide which product may be wanted in the market, and therefore deploy an economic version first and only then decide on deploying more sophisticated versions. Another useful functionality of the method (and its supporting tool) is to arm decision makers with the flexibility to play with different choices and understand their impact on the ranking and, consequently, on the final decisions they have to make.

9. Acknowledgements

We would like to express our gratitude to João Araújo, Catarina Gomes, António Costa and João Santos who provided contributions for the preliminary version of HAM. This work has been partially supported by the European Commission grant IST-33710 - Aspect-Oriented, Model-Driven Product Line Engineering (AMPLE).

10. References

[1] N. Ahmad and P. A. Laplante, *Software Project Management Tools: Making a Practical Decision Using AHP*, in Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop SEW-30 (SEW'06). 2006.

[2] AMPLE, *http://www.ample-project.net.* 2009.

[3] D. R. Anderson, D.J.Sweeny, and T.A. Williams, *Quantitative Methods for Business*. 2003: South-Western College publishing Co.

[4] Anna Perini, Filippo Ricca and Angelo Susi, *Tool-Supported Requirements Prioritization. Comparing the AHP and CBRank Methods*, Information and Software Technology 51, 2009, pp. 1021-1032

[5] J. Barzilai, W. Cook, and B. Golany, *Consistent Weights for Judgements Matrices of the Relative Importance of Alternatives.* Operations Research Letters, 1987. **vol. 6** (nr. 3).

[6] I. S. Brito, F. Vieira, A. Moreira, and R. A. Ribeiro, *Handling conflicts in aspectual requirements compositions.* Lecture Notes in Computer Science (LNCS), Vol 4620, Transactions on Aspect-Oriented Software Development III, pp. 144-166, 2007.M. Lindvall, R. T. Tvedt, and P. Costa, *An Empirically-Based Process for Software Architecture Evaluation. Empirical Software Engineering.* Springer, 2003. **vol. 8**: pp. 83-108.

[7] S. Chen and L. Hwang, *Fuzzy Multiple Attribute Decision Making - Methods And Applications.* Springer-Verlag, 1992.

[8] L. Chung, B. A., Nixon, E. Yu, J. Mylopoulos, *Non-Functional Requirements in Software Engineering.* Kluwer Academic Publishing, 1999.

[9] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns.* Addison-Wesley, 2002.

[10] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies.* Addison Wesley, 2002.

[11] Y. Dong, Y. Xu, H. Li, and M. Dai. *A comparative study of the numerical scales and the prioritization methods in AHP*. European Journal of Operational Research, 2008. **vol 186**: pp. 229-242.

[12] Eclipse. *http://www.eclipse.org/*.

[13] J. Figueira, S. Greco, and M. Ehrgott, *Multiple Criteria Decision Analysis: State of the Art Surveys.* International Series in Operations Research & Management Science, Springer 2005. **vol 78**.

[14] S. I. Gass, *Decision making, models and algorithms: a first course.* Wiley, 1985.

[15] B. Golden, E. Wasil, and P. Harker, *The analytic hierarchy process: applications and studies.* Springer, Berlin, 1989.

[16] C. Gomes, R. A. Ribeiro, and A. Pimentel, *SWDSS – an experience in transforming a single system into a software product line.* Proceedings of the 4th Iberian Conference on Information Systems and Technologies, Povoa de Varzim PORTUGAL, June 2009: pp. 161-166.

[17] *Java Sun - http://java.sun.com/javase/6/docs/api/.*

[18] Y. Lee, K. Kozar, *Investigating the effect of website quality on ebusiness success: an analytic hierarchy process (AHP) approach*, Decision Support Systems vol. 42 (3) (2006) 1383–1401.

[19] M. Lindvall, R. T. Tvedt, and P. Costa, *An Empirically-Based Process for Software Architecture Evaluation. Empirical Software Engineering.* Springer, 2003. **vol. 8**: pp. 83-108.

[20] D. Ma and X. Zheng, *9/9-9/1 Scale Method of AHP.* 2nd International Symposium on the AHP, Pittsburgh, PA, USA, 1991: pp. 197-202.

[21] N.A.M. Maiden, P. Pavan, A. Gizikis, H. Kim, O. Clause, X. Zhu, Integrating decision-making techniques into requirements engineering, in: Proceedings of Eighth International Workshop on Requirements Engineering: Foundation for Software Quality, Colocated with RE'02 Conference, Essen, Germany, 2002.

[22] J. McCaffrey, *Multi-Attribute Global Inference of Quality (MAGIQ).* Software Test and Performance Magazine, 2005. **vol 2** (nr 7): pp. 28-32

[23] N. R. Mead, *Requirements Prioritization Case Study Using AHP*. https://buildsecurityin.uscert.gov/daisy/bsi/articles/bestpractices/requirements/534.html?branch=1&language=1.

[24] A. Moreira, A. Rashid, and J. Araújo, *Multidimensional Separation of Concerns.* Requirements Engineering, 2005.

[25] *NetBeans - www.netbeans.org*.

[26] T. C. Pais, R. A. Ribeiro, and L.F. Simões, *Uncertainty in dynamically changing input data*, in Computational Intelligence in Complex Decision System, DaRuan, Editor. 2009, World Scientific.
[27] A. Pimentel, *Multi-criteria analysis for architectural choices in software product lines*, in *Master´s dissertation in Computer Science*. 2009, Universidade Nova Lisboa.
[28] T. L. Saaty and L.G. Vargas, *Prediction, Projection and Forecasting in Applications of the Analytical Hierarchy Process in economics, finance, politics, games and sports.* Boston: Kluwer Academic Publishers, 1991.
[29] T.L. Saaty, *Fundamentals of the Analytic Hierarchy Process*. RWS Publications, 4922 Ellsworth Avenue, Pittsburgh, PA 15413, 2000.
[30] T. L. Saaty, *The Analytical Hierarchy Process*. 1980: Network: McGraw-Hill.
[31] T. L. Saaty, *Decision Making for Leaders.* University of Pittsburgh, 1986.
[32] B. Srdjevic, *Linking analytic hierarchy process and social choice methods to support group decision-making in water management*, Decision Support Systems vol.42 (4) (2007) 2261–2273.
[33] E. Triantaphyllou, *Multiple Criteria Decision Making Methods: A Comparative Study*. Kluwer Academic Publishers, 2000.
[34] Y-M. Wang, T.. M. S. Elhag, *An approach to avoiding rank reversal in AHP*. Decision Support Systems vol.42 (2006) 1474–1480.
[35] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, *Requirements engineering paper classification and evaluation criteria: a proposal and a discussion.* Requirements Engineering, 2006. **vol 11 (1)**: pp. 102-107.
[36] K. Yoon, C. Hwang, *Multiple Attribute Decision Making - An Introduction*. 1995: A SAGE University Paper.