A Machine Learning Approach to Software Requirements Prioritization

SATISH TUMULUR¹, KAMBHAMPATI VIJAY KUMAR², Y.V.R.NAGA PAVAN³

¹PG Scholar Dept of CSE, Anurag Engineering College, Telangana, India, E-mail: satishhtumuluri@gmail.com.
²Assistant Professor, Dept of IT, Anurag Engineering College, Telangana, India.
³HOD, Dept of IT, Anurag Engineering College, Telangana, India.

Abstract: Deciding which, among a set of requirements, are to be considered first and in which order is a strategic process in software development. This task is commonly referred to as requirements prioritization. This paper describes a requirements prioritization method called Case-Based Ranking (CBRank), which combines project’s stakeholders preferences with requirements ordering approximations computed through machine learning techniques, bringing promising advantages. First, the human effort to input preference information can be reduced, while preserving the accuracy of the final ranking estimates. Second, domain knowledge encoded as partial order relations defined over the requirement attributes can be exploited, thus supporting an adaptive elicitation process. The techniques CBRank rests on and the associated prioritization process are detailed. Empirical evaluations of properties of CBRank are performed on simulated data and compared with a state-of-the-art prioritization method, providing evidence of the method ability to support the management of the tradeoff between elicitation effort and ranking accuracy and to exploit domain knowledge. A case study on a real software project complements these experimental measurements. Finally, a positioning of CBRank with respect to state-of-the-art requirements prioritization methods is proposed, together with a discussion of benefits and limits of the method.

Keywords: Requirements Management, Requirements Prioritization, Machine Learning.

I. INTRODUCTION

The problem of prioritizing software requirements amounts to ranking a set of desired functionalities and features of the intended software along one or more concerns such as business aspects (e.g., market competition or regulations, customer satisfaction) or technical aspects (e.g., development costs or risks). Requirements prioritization plays a crucial role in software development, and in particular it allows for planning software releases, combining strategies for budget management and scheduling, as well as market strategies. It is, in fact, considered a complex multi-criteria decision-making process. State-of-the-art approaches tend to share a common model for this process, which consists of the following steps.

- The definition of a target criterion for ordering.
- The specification of requirement attributes to encode the chosen criterion.
- The acquisition of specific values for those attributes, for all requirements under consideration.
- The composition of rankings induced by requirement attributes associated to the target criterion.

More precisely, the first step is concerned with the selection of the most appropriate prioritization criterion according to specific strategic goals, such as reducing the development costs or minimizing the overhead of bug fixing. The identification of requirement attributes in the second step is performed in a way to define uni-variety ranking functions on the requirements set. For example, with reference to the goal of reducing development costs and the choice of “development cost” as a target ranking criterion, requirements attributes such as the estimated number of “lines of code” or of “components” are suitable. The third step, namely, the acquisition of attribute values over the set of requirements, usually represents the most expensive task in the prioritization process since it rests on the availability of expert knowledge or on the elicitation of evaluations from stakeholders. Since a target criterion might be encoded by manifold attributes and each attribute induces a ranking of the requirements set, the fourth step is concerned with the composition of the different attribute based rankings into a global ordering corresponding to the target criterion. This composition is usually defined in terms of a weighted aggregation schema.

The widespread use of such a process model for requirements prioritization is also confirmed by a survey review that considered more than 200 papers in the field of requirements prioritization for benefit and cost criteria. The assumption underlying the analyzed approaches is that the ranking criteria, the requirement attributes, and the way to compose them in case of multi-criteria ranking can be defined independently of the nature of the current set of requirements under evaluation. In other words, they adopt an ex-ante perspective about the requirements prioritization problem which prevents exploiting available knowledge on
the project’s application domain. In contrast, an ex-ante perspective will enable the exploitation of this knowledge through a prioritization process that is built on the actual set of requirements under evaluation and will lead to a different realization of steps 2 to 4. Namely, project stakeholders are asked to perform a pair wise comparison of the current requirements, allowing them to decide which requirement is to be given a higher rank between two alternatives without the need to identify a specific requirement attribute to encode the evaluation criterion adopted by the stakeholder. So, for instance, the users of an e-voting system may be asked to decide which of the requirements “Graphical layout of the voting form” and “Getting audio feedback during the voting procedure” is more important (this evaluation will be repeated for different requirement pairs).

The difference between ex-ante and ex-post approaches can be summarized as follows. While in the ex-ante perspective the target criterion is chosen in advance, in ex-post approaches project stakeholders are required to evaluate pairs of requirements along an underlying target criterion. Consequently, requirements ranking is not computed from the values of requirement attributes, but it is derived from the priority relations that are elicited directly from stakeholders, who may take into account implicit information that might not have been preliminarily encoded as requirement attributes. The composition of rankings in case of multi-stakeholder prioritization is provided as instances (examples) of pair wise priority relations and not as the result of the application of an analytical composition schema. An interesting advantage of eliciting input regarding relative values rather than absolute values for attributes is that the noise on the input is recognized to be lower. The ex-post perspective derives from the problem solving paradigm known as case-based reasoning [2], according to which a solution to a new problem is inferred from examples of solutions to similar problems. The Analytical Hierarchy Process (AHP) can be considered the reference method among those which are based on the case-based paradigm.

In this method, the ranking criteria are defined upon an assessment of the relative priority between couples of requirements, expressed by project stakeholders. This assessment encompasses all possible pairs of requirements. The effort required by the human evaluator when pair preferences are elicited grows rapidly with the number of requirements since the number of pairs grows quadratically. This makes AHP difficult to use with large sets of requirements, a problem that is typically dealt with by defining ad hoc heuristics for deciding when the pair preference elicitation process can be stopped without compromising the accuracy of the resulting ranking.1 This may be a main reason for ex-post approaches being less commonly used than ex-ante approaches in requirements prioritization practices. Analogously to AHP, in our work we follow an ex-post perspective to the requirements prioritization problem and propose a method called Case-Based Ranking (CBRank from now on). Differently from AHP, CBRank adopts a highly flexible preference elicitation process which rests on the following basic properties. First, it allows for combining sets of preferences elicited from human decision makers with sets of preferences, which are automatically computed through machine learning techniques. These techniques exploit knowledge about (partial) orders of the requirements that may be encoded in the description of the requirements itself (i.e., in terms of the actual requirement attributes), thus enabling what we call domain adaptively. This accounts for the straightforward applicability of CBRank to different application domains and for the fact that the accuracy of machine-estimated ranking increases with the level of significance of the encoded domain knowledge. Second, CBRank is organized according to an iterative schema which allows for deciding when to stop the elicitation process on the basis of a measure of the tradeoff between the elicitation effort and the accuracy of the resulting ranking. With a reasonable effort, the method can be applied up to 100 requirements.

The objective of this paper is to offer a detailed and comprehensive presentation of the CBRank method, providing:

- A formal definition of the prioritization problem it solves,
- An intuitive description of the machine learning technique it is based on and a characterization of the prioritization process supported by CBRank,
- A comprehensive overview of the empirical measurements which have been performed to assess key properties of the method, and
- A positioning of CBRank with respect to state-of-the art requirements prioritization methods.

The description of the empirical evaluations conducted on CBRank points out the different empirical study techniques that have been applied. In summary, by simulating the prioritization process we collected experimental data at support of the analysis of the effort versus accuracy tradeoff and of the domain adaptive property of the CBRank method. Preliminary results have been presented in [5], [6], and [7] and data from new simulations are discussed in this paper. We investigated the effectiveness of the domain adaptively property of CBRank when used in a real case study. This empirical study complements the previous experiments that are based on simulated prioritization processes. We executed a controlled experiment with 23 subjects aiming at comparing two tool-supported versions of CBRank and AHP with respect to ease-of-use of the methods, time-consumption for performing the prioritization task, and accuracy of the resulting ranking. In this paper, we briefly recall the main result of this experiment and refer the interested reader to for a full description.

The paper is structured as follows: Section 2 defines the Related Work and Prioritization Method, Section 3. The experimental studies carried out to evaluate the effort versus accuracy tradeoff and the domain adaptively property of CBRank are then recalled along with their design, execution,
and the analysis of the results in Sections 4 respectively. Finally, Section 5 concludes and outlines future work.

II. RELATED WORK
In this section we discuss important related work in requirements prioritization and legal requirements.

A. Requirements Prioritization
Herrmann et al. systematically reviewed requirements prioritization techniques based on benefit and cost predictions, noting that prioritization during requirements refinement was under-researched. In addition, dependencies between requirements are often neglected when producing a prioritization. Herein we address the need to prioritize during refinement and manage dependencies between legal requirements. Karlsson classifies two primary approaches to requirements prioritization: the pair-wise comparison technique and the numeral assignment technique. The Analytic Hierarchy Process (AHP) serves as a well-known example of a pair-wise technique show that pair-wise comparison techniques require substantial effort upfront because every pair of requirements must be compared with one another. Much of this effort must be repeated to re-prioritize requirements as they change or evolve. In contrast to pair-wise approaches, existing numeral assignment prioritization techniques take more time and produce less reliable prioritizations than pair-wise comparison techniques. Herein, we develop a numeral assignment based on legal text structure, which can be produced quickly while providing a useful legal requirements prioritization.

B. Legal Requirements
The needs to resolve ambiguities, to follow cross references, and to master extensive domain knowledge make legal compliance particularly challenging for requirements engineers. Researchers have focused on building logical models of regulation directly from actual legal texts. Barth et al. employ a first-order temporal logic approach to model laws and regulations as well as measure compliance. May et al. model legal texts using an access control approach take a goal-oriented modeling approach to Italian data protection laws use production rules, an artificial intelligence technique, to model regulations for requirements. None of this legal requirements research addresses prioritization. Although primarily intended as a methodology to build requirements that achieve complete legal compliance, FBRAM also produces a comprehensive prioritization hierarchy for an entire legal text. The prioritization hierarchy is used to identify which requirement takes precedence in cases of legal exceptions. As a result, it is more useful for proving that a set of requirements has the same priority of outcomes as a set of legal regulations than for prioritizing requirements for the purposes of software construction or maintenance.

Another important consideration for legal requirements is that legal text interpretations can change over time. Laws and regulations may be amended by administrative agencies or interpreted in judicial proceedings for years after their initial passage. Legal norms in the U.S. suggest that HIPAA regulations could be revised as often as every year. Engineers using pair-wise requirements prioritization techniques must reapply those techniques whenever a legal or regulatory change occurs.

III. PRIORITY METHOD
This section will explain the methods of prioritization analyzed in this paper.

A. Case Based Ranking
The Case-Based Ranking (CBRank) method exploits machine learning techniques to guide user preferences elicitation in the prioritization process. The framework rests on an iterative process that can handle single and multiple decision makers (stakeholders) and different criteria (both business goals and technical parameters). Fig.1A sketches the basic steps of the prioritization process, where manual elicitation interleaves with machine supported steps. The main input to the process is the collection of requirements that have to be ranked. The final output of the process is an approximation of the target ranking. The pair sampling activity is an automatic procedure which selects a pair (or a sample of pairs) of requirements on the basis of a predefined selection policy which may take into account information on the currently available rankings. The user performs the evaluation of the requirements pairs, by iterating the following steps till all the pairs in the sample have been evaluated: select a pair from the sample; evaluate the relative importance of the requirements in the pair. That is, given a pair of requirements A and B, the user is asked to specify what is the “most important” requirements among A and B with respect to the given criterion. Differently from AHP, there is no range of values, and the preference is strict. The output of this step is a set of ordered pairs.

The ranking learning activity takes in input the stakeholder preferences acquired in the previous step, and computes an approximation of the ranking function. The learning procedure is based on the boosting approach described and may eventually exploit also available knowledge on the requirements rankings induced by other prioritization criteria (e.g. the cost for the realization of the requirements, the estimated utility) defined on the initial set of requirements in order to best approximate the final ranking. If the ranking produced by the ranking learning activity can be considered a good approximation (e.g. the error measures exploited in the method are minimized), it will be given in output, and otherwise it may become the input to a further iteration of the process. The CBRank method is supported by a web-based tool named SCORE (Supporting Case-based Oriented Rank Elicitation) which allows for a distributed use of the framework, to support the pair-wise priority elicitation by distributed stakeholders. Fig. 2 shows a snapshot of the SCORE graphical user interface. The system supports the whole evaluation process. In particular, SCORE presents the user an agenda of comparisons. The user can analyze each one of the pairs specifying the preferred requirements in the pairs, by indicating which one of the requirements is “the most
important” (see the system user interface showed in Fig.2). Finally, once all the evaluations have been performed, the system computes the rank and, in the case of a further iteration, it presents to the user the set of new pairs of requirements to be evaluated.

B. Cumulative Voting

Technique that produces ratio-scale results is the Cumulative Voting technique (also known as the Hundred-Dollar Test), further on denoted as CV. This technique has been used for a long time within other fields, such as political elections and elections for company boards. In the software engineering domain, CV has not been reported in as many cases as AHP, even though the number of reported cases has grown in the last years, for example in requirements prioritization and in prioritization of process improvements. CV is considered as a simple and straightforward technique where the stakeholders are given 100 imaginary units (money, points, etc.) to distribute among the objects to prioritize. In requirements prioritization, the number of units assigned to a requirement represents requirement’s relative priority (e.g. importance, cost, risk) in relation to the other requirements. Since the requirements are assigned numbers in this way, it is possible for a stakeholder to give a requirements zero in priority. This is not possible in AHP since all requirements take part in pair-wise comparisons, meaning that a requirement always gets some amount of importance. Fig.1B sketches the basic steps of the prioritization process for this method, where manual elicitation interleaves with machine supported steps.

A problem with CV arises when there are too many requirements to prioritize. For example, if you have 25 requirements, there are on average four points to distribute for each requirement faced this problem when there were 17 groups of requirements to prioritize. In the study, they used a fictitious amount of $100,000 to have more freedom in the prioritizations. The subjects in the study were positive about the technique, indicating the possibility to use amounts other than 100 units (e.g. 1,000; 10,000; or 100,000). Another possible problem with CV (especially when there are many requirements) is that the person performing the prioritization miscalculates and the sums do not add up to the prescribed amount of points. This can be prevented by using a tool that informs about how many points that are left to distribute.

IV. EFFORTS VERSUS ACCURACY TRADEOFF

A. Experiment Definition

This first experiment aims at investigating the tradeoff between pair wise elicitation effort and ranking accuracy, following the approach summarized, column. The variables evaluated in the experiment are the accuracy of the final rank obtained with a specific method, measured as disagreement with respect to the target rank, and the elicitation effort, measured as the number of elicited pair wise comparisons. Trends of these two variables while the numbers of requirements increases are also investigated. This analysis is
A Machine Learning Approach to Software Requirements Prioritization

performed with both the CBRank and AHP methods. Since AHP is known to be effective only for small sets of requirements, we are interested in assessing whether the advantage of using CBRank increases more than linearly as the set of requirements becomes larger. In addition, the comparison is extended to include a state-of-the-art implementation of AHP which adopts some heuristics to reduce the elicitation effort.

B. Experiment Execution

The investigation is performed on artificially generated sets of requirements, which are obtained by applying the following procedure. A given number of requirements are defined with unique identifiers. Each requirement is described by a set of attributes, which take integer values in the range between 0 and the cardinality of the requirement set. Attribute values are assigned in such a way that each attribute induces a total rank over the set of requirements. A total rank over the set of requirements is generated as reference target rank, namely, K, which may partially overlap with one of the ranks encoded by the attributes. This procedure allows for parametric generation of data set with an increasing number of requirements. Following the steps of the CBRank prioritization process, as depicted, instances of this process are simulated by applying the following procedures. The first step of the prioritization process, which concerns pair sampling, at the beginning aims to provide coverage of all requirements. The initial set of pairs is defined in such a way that it has cardinality n/2 and that each requirement \( r_i \in Req \) is a member of at least one pair in the initial set.

The second step, namely, Preference elicitation, is the only step that requires human input. We simulate preference elicitation from decision makers as follows: First, a subset of pairs \( (r_i, r_j) \) is selected from the Cartesian product \( Req \times Req \) with the restriction that \( i \neq j \) and \( (r_i, r_j) \neq (r_j, r_i) \). The relative preference function \( \Phi(r_i, r_j) \) value is retrieved by directly sampling the given target ranking function K. For simplicity we restrict the simulation to a monotonic behavior, that is, we assume that the simulated decision maker acts without giving inconsistent answers during the process. The third step invokes the Rank Boost algorithm to produce an estimate of the proper prioritization, \( H(r) \), fully defined over the set Req resulting in an approximated priority rank. For the comparative evaluations we also run an AHP based simulated prioritization process. In this case, the candidate pairs of the set of requirements Req are generated by exploiting a spanning tree (composed by n - 1 pairs), as described. The preference elicitation step is simulated following the same approach used in the case of CBRank and the computation of the rankings is performed through a procedure that implements the AHP algorithm described.

The experimental comparison between CBRank and AHP has been conducted on sets of requirements with cardinality n equal to 25, 35, 50, 75, and 100, respectively. For each set of requirements, we run two prioritization processes, one based on CBRank and the other on AHP. In both cases, we computed the disagreement corresponding to a given number of elicited pair preferences. A second set of measurements focuses on the application of the local stopping rule, which was proposed as a means of determining when the ranking estimate computed by AHP has reached an acceptable error threshold and new pair wise comparisons are no longer needed. Adopting the notation introduced in the previous sections, the local stopping rule can be expressed as follows.

| TABLE I: Disagreement for 25, 35, 50, 75, 100 Requirements, with 100 Elicited Pairs, Computed as Average on 10 Runs in AHP and CBRank |
|---|---|---|---|---|---|
| n  | 25  | 35  | 50  | 75  | 100 |
| \( TDA_{AHP} \) | 31  | 71  | 208 | 693 | 1584 |
| \( TDA_{CBRank} \) | 22  | 54  | 122 | 416 | 990 |
| Difference | 9   | 17  | 86  | 277 | 594 |

Given the function \( K_{AHP} \) that represents the correct ranking and the function \( H_{AHP}(r) \), that is the ranking computed by the AHP algorithm at a certain stage \( \sigma \) of the pair wise elicitation step, the local stopping rule can be represented by the following expression:

\[
| H_{AHP}(r_i) - H_{AHP}(r_j) | < \alpha \quad \forall r_i \in Req
\]

Where \( \alpha \) is a positive real number. Essentially, the rule requires that in two subsequent stages \( \sigma -1 \) and \( \sigma \) the acquisition of new knowledge in terms of pair wise comparisons is not going to significantly modify the value of the ranking of \( r_i \).

C. Results

Table 1 reports on the disagreement measures collected from 10 executions the prioritization process over the same dataset, with the AHP and the CBRank methods, respectively, while varying the number of requirements, namely, \( n = 25, 35, 50, 75, 100 \), and keeping the number of elicited pairs fixed to 100. The second and third rows correspond to the average over 10 runs of the disagreement computed with AHP and CBRank, respectively, and the fourth row reports the difference between the average disagreement computed with AHP and CBRank. We computed the p-value via Mann-Whitney-Wilcoxon test as an indicator of the statistical significance of the difference of the means obtained by the two methods for all the measures the p-value is < 0.05, so it is possible to confirm that the difference between the two methods is statistically significant.
The series of plots, Figs. 3, 4, and 5, aims at comparing the two methods along the prioritization problem dimensions (i.e., number of requirements, elicited pairs, accuracy), considered two by two. Fig. 3 reports on the x-axis the number of elicited requirement pairs as a measure of the elicitation effort. The y-axis corresponds to the ranking accuracy measured as TDA, computed. The plotted values have been obtained as the average of the disagreement values measured on 10 runs on the same data set of 100 requirements. Fig. 4 shows the average disagreement (y-axis) for 25, 35, 50, 75, 100 requirements (x-axis) for AHP and the CBRank for a fixed elicitation effort, namely, for 100 elicited pairs. Fig. 5 plots the average number of pairs to be elicited (y-axis) to limit the disagreement of the resulting ranks to 15 percent of the total number of pairs of requirements for 25, 35, 50, 75, 100 requirements (x-axis), computed with the AHP and CBRank methods.

Fig. 6 reports some results of the application of the local stopping rule with CBRank and AHP. In particular the diagram represents the results over 20 AHP runs, on the same set of 25 requirements and for a = 0.01 (cross label). Every point in the diagram represents the final value of one of these 20 runs. The results obtained running CBRank with the stopping rule, on the same set of requirements, with a = 0.7, are depicted in the same plot (triangle label). At these two slightly different values for the parameter a, the AHP and CBRank prioritization processes exploit similar percentages of the elicited pairs. Fig. 7 shows the variance on the results taken when applying the stopping rules for AHP and CBRank.
A Machine Learning Approach to Software Requirements Prioritization

D. Discussion

The first set of measurements allows a deep analysis of the tradeoff between the elicitation effort and the ranking accuracy with the two methods, and provides empirical evidence that CBRank uniformly outperforms AHP. The results show that CBRank is more effective than AHP, especially for lower numbers of elicited pairs, since the difference between the disagreements measured with the two methods decreases with the increase of the elicited pairs, as shown in the plot depicted in Fig.3. Looking at the behavior of the two methods for low elicitation effort in greater detail (the most interesting case for practical purposes), we can see that the improvement of CBRank with respect to AHP increases when considering larger sets of requirements. In fact, considering in particular, the cases of 50, 75, 100 requirements when 100 pairs are elicited (Fig.4 and Table 1) this trend in the difference between CBRank and AHP is evident. For instance, the difference between the disagreements measured with CBRank and with AHP in the case of 100 requirements is around 594 pairs and is clearly larger than the difference for 25 requirements, which are nine pairs.

This observation is particularly remarkable since, in practice, only a small portion of pairs can be manually elicited. For example, 100 pair wise analysis represents 2 percent of the total pairs in the case of 100 requirements, while 10 percent pair wise analysis on 100 requirements requires performing 495 comparisons.

The difference between the two methods is also evident when considering the objective of having a fixed percentage of disagreement, as shown in Fig.5. Also, in this case the difference between the number of pairs to be elicited in AHP and CBRank for a disagreement of 15 percent increases, while the number of requirements grows. During the experiments, the effectiveness of the spanning tree initialization strategy used in AHP has also been tested with CBRank. The results show that the adoption of this strategy does not produce a faster convergence of the Rank Boost algorithm. A better performance of CBRank with respect to AHP is also found when analyzing the results obtained using the local stopping rule, namely, better accuracy and lower effort, as well as lower variance on the respective measurements. The first result could be expected. In fact, applying the same policy to reduce the elicitation effort to both CBRank and AHP for a class of problems for which CBRank outperforms AHP will not change the relative trend of the two methods.

Concerning the variance of elicitation effort measurements, the following observations are worth being added. The behavior of the local stopping rule as a technique to reduce the elicitation effort is strongly context sensitive. The final outcome is affected by high variance of the numbers of pairs that have to be elicited. For example, in our experiment with 25 requirements depicted in Fig.6, the percentage of elicitation pairs for AHP spans from 10 to 30 percent for 20 runs on the same problem (as also shown in Fig.7 on the left). For CBRank, the effort variance is half with respect to AHP: In fact, for CBRank the effort is in the range between 10 and 20 percent of pairs and the median is around 13 percent, while AHP has a median around 21 percent. It is important to note that for both methods the results show a high variance with respect to the disagreement obtained when the process is stopped by the rule. This is plotted in Fig.7 on the right. The behavior of CBRank seems to reveal a greater benefit from the local stopping rule. CBRank not only halves the variance of the elicitation effort but it lowers the upper bound of the number of pairs that have to be elicited as shown in Fig.8.

V. CONCLUSION AND FUTURE WORK

In this paper, we provided a detailed account of the CBRank method for requirements prioritization. The CBRank method follows the case-based paradigm for problem solving, according to which a solution to a new problem can be derived from (partial) examples of previous solutions to similar problems. In the context of requirements prioritization, these examples are elicited from project stakeholders as pair wise preferences on samples of the set of
requirements to be prioritized, and used to compute an approximated ranking for the whole set. The machine learning technique exploited by the method has been presented, both with the help of an intuitive example and by describing the Rank Boost algorithm, which is implemented in the method. The prioritization process based on CBRank has been presented. A discussion of the method performance, which is defined in terms of tradeoff between preference elicitation effort and ranking accuracy and of its domain adaptively, has been given, with the support of a set of different experimental measurements and of a case study. The experimental measurements were taken by applying CBRank to different prioritization problems, varying the number of requirements, the number of elicited pairs, and the accuracy of the computed ranking. Indicators for the statistical significance of the measurements have been provided.

Finally, the CBRank method has been positioned with respect to state-of-the art approaches, with particular reference to the AHP method, which can also be considered an instance of the case-based problem solving paradigm. Differently from AHP, the CBRank method enables a prioritization process, even over 100 requirements, thanks to the exploitation of machine learning techniques that induce requirements ranking approximations from the acquired data. Some assumptions have been considered in the described work, such as the monotonicity of the elicitation process and random selection as pair sampling policy in the CBRank prioritization process. Future work should address the non-monotonic case and more sophisticated pair sampling policies, possibly contributing to improving the effectiveness of the method in more complex real settings. Further characteristics of the CBRank method that are worth investigating are its ability to support coordination among different stakeholders through negotiation [8]. Moreover, potential advantages of integrating CBRank with other techniques such as planning game or AHP deserve further analysis. Well-known open issues in the requirements prioritization problem such as handling requirements dependencies and “anytime” prioritization, which is updating requirements ranking when new requirements are added (or removed), are also worth being further investigated.

VI. REFERENCES