Hindawi
Complexity
Volume 2019, Article ID 2306213, 20 pages
https://doi.org/10.1155/2019/2306213

Research Article

Selection and Prioritization of Software Requirements Applying Verbal Decision Analysis

Paulo A. M. Barbosa,[1,2] **Plácido R. Pinheiro**,[1] **Francisca R. V. Silveira**,[1,2] **and Marum Simão Filho**[3]

[1]*Program in Applied Informatics, University of Fortaleza, Fortaleza 60811-905, Brazil*
[2]*IT Department, Federal Institute of Ceará, Tianguá 62320-000, Brazil*
[3]*7 de September College, Fortaleza 60135-420, Brazil*

Correspondence should be addressed to Paulo A. M. Barbosa; albertobmap@hotmail.com

Received 7 March 2019; Revised 17 June 2019; Accepted 15 July 2019; Published 7 August 2019

Academic Editor: Roberto Natella

During the software development process, the decision maker (DM) must master many variables inherent in this process. Software releases represent the order in which a set of requirements is implemented and delivered to the customer. Structuring and enumerating a set of releases with prioritized requirements represents a challenging task because the requirements contain their characteristics, such as technical precedence, the cost required for implementation, the importance that one or more customers add to the requirement, among other factors. To facilitate this work of selection and prioritization of releases, the decision maker may adopt some support tools. One field of study already known to solve this type of problem is the Search-Based Software Engineering (SBSE) that uses metaheuristics as a means to find reasonable solutions taking into account a set of well-defined objectives and constraints. In this paper, we seek to increase the possibilities of solving the Next Release Problem using the methods available in Verbal Decision Analysis (VDA). We generate a problem and submit it so that the VDA and SBSE methods try to resolve it. To validate this research, we compared the results obtained through VDA and compared with the SBSE results. We present and discuss the results in the respective sections.

1. Introduction

Software Engineering encompasses a set of methods and activities that support the development, maintenance, and improvement framework for Software. As part of the software development process, we can find Software Release Planning (RP) that supports all issues related to the assignment and selection of resources to a sequence of consecutive product releases [1]. Poorly done RP can lead to problems for the development process itself. Moreover, this is because the Releases, components of the Software delivered in modules, are composed of several requirements that represent a part of the development in the Software.

The planning that precedes the stage of software development is essential and brings a positive cost-benefit to the developer company. Also, the software delivery challenge is about reducing the time to deliver the system without compromising quality [2].

The Release model, which comes from incremental software development, allows customers to receive parts of the software in advance. Because we can divide this software, we can have a unitary view of each requirement, which corresponds to one or several cases of software use. In this context, this allows the assignment of individual values for each developed requirement. A customer can have personal preferences for each requirement and assign preference values to those requirements [1].

Note that agile methods also use incremental software deliveries. These deliveries, known as sprints, have a short interval to be developed. The addition of a set of features included in the increment comes from negotiation between the clients and the development team. These selected features

come from a set of high-level requirements that cover all customer needs [3].

In some types of software, especially those that are large and complex, the development and maintenance companies of this type of system have difficulty in knowing which set of requirements will deliver in the next version. [4].

After eliciting and validating requirements, the decision maker (DM) has the task of determining actions on these requirements, such as assigned risk, technical precedence, individual cost, degree of complexity versus available team for implementation, the time required for development, the interests of the software developer company, the interests of a group of customers by specific requirement, and the degree of stability (or volatility) of the requirement representing the change rate of a requirement over time. Requirements already elicited can be modified by the client. Therefore, it is necessary to implement these less stable (more volatile) requirements in later releases. Therefore, giving preference for implementing requirements that tend to have few modifications is interesting.

The requirements have many characteristics depending on the type of project. Typically, development companies and their clients make contracts for software deliveries. In this sense, DM is responsible for ordering these requirements in a queue of deliverables. For the set of requirements delivered in a batch already implemented, we call Software Release. Several methods support DM to facilitate the work of ordering requirements while optimizing Releases according to a set of constraints aggregated to the business. These restrictions may be the limit of resources that can be allocated to the project, the interests of customers by specific Releases, and the preferential implementation of more stable requirements, leaving the unstable, and thus subject to change, to the later stages of development, requirements of the Releases, among other things that we can consider as the restriction.

As mentioned, the tools that support DM are of fundamental importance to the success of the project. In literature, we can find many methods that seek to facilitate this work [5–7]. These automated methods have several methodologies for finding solutions, or orders for implementations, taking into account goals and constraints. So, it is trivial in many respects (company reputation, business value, and business investors); the reasonable solutions to this problem are found.

This paper aims to demonstrate the ability of methods that make up the field of Verbal Decision Analysis as an alternative to perform the task of selection and ordering of requirements. We know that this task is usually arduous for DM, and wrong choices during the process of selecting and requesting requirements can lead to serious failures during project execution.

We have seen in previous research [8–10] some challenges, such as trying to reduce the number of questions asked to the decision maker, since this factor made the prioritization process time-consuming. Another factor that we seek to improve is the solutions found. Thus, to achieve these results, we seek in this work to select and prioritize software requirements using two VDA methods, the ORCLASS classifier, and the ZAPROS III-*i* prioritizer. The ORCLASS method seeks to select requirements by acting as a filter and allowing passing to the next step only requirements that somehow fit the objectives and constraints of the project. Next, we use the ZAPROS III-*i* method as a requirements sorter ordering and inserting into releases in order. We seek to (i) decrease the time required to complete this process, (ii) increase the efficiency in the use of available resources for the project, (iii) increase the level of satisfaction of decision makers, and (iv) achieve desirable solutions for decision-making.

We organized the structure of this work into eleven sections. In Section 2, we will know the Software Requirements Prioritization problem and its challenges. Section 3 presents the concept of Search-Based Software Engineering (SBSE), a field of software engineering aimed at solving multiobjective optimization problems. Let us know the two methods of this field used in this work, NSGA-II and MOCell. In Section 4, we describe the Verbal Decision Analysis (VDA), whose field of action is the focus of this work. This section presents the ORCLASS method, which performs classification operations, and the ZAPROS III-*i* method, which is a ranking method. In Section 5, we offer the methodology of this work that tries to use two methods available in the VDA field, the ORCLASS method and the ZAPROS III-*i* method, to select and order software requirements. Next, we compared the results generated with those obtained by two methods of SBSE, NSGA-II, and MOCell. Section 6 shows that, to reach our goal, we will have to create an adaptation of the VDA method to support the handling of the software requirement selection and ordering problem. This section defines the mathematical formulation adopted by this work.

Following the flow of this research, in Section 7 we have the configuration adopted for the methods of the SBSE and its application and in Section 8 the configuration chosen for the methods of the VDA with a description of the series of procedures necessary for a later comparison of results. Section 9 presents all the results of this research with comparisons and comments. Finally, we offer the conclusion of this work in Section 10 and limitations and future work in Section 11.

2. Software Requirements Prioritization

The paper published by [4] addresses the problem of the next release, which is part of the requirements planning process. The author describes that clients may have different levels of relationship with the company and may indicate prerequisites for requirements. Therefore, the implementation of these requirements must be in the same or previous release.

One of the most significant risks faced by organizations developing commercial software is associated with not meeting the needs and expectations of users. For these authors, this risk can lead to damage to reputation, loss of orders, and reduction of company profits [11]. To improve this, we must have proper execution of the elicitation and allocation phase of requirements. It is one of the activities that occur early in software development.

The number of variables that make up releases is often higher than those already perceived by the software manager. In this way, elaboration of adequate planning is essential,

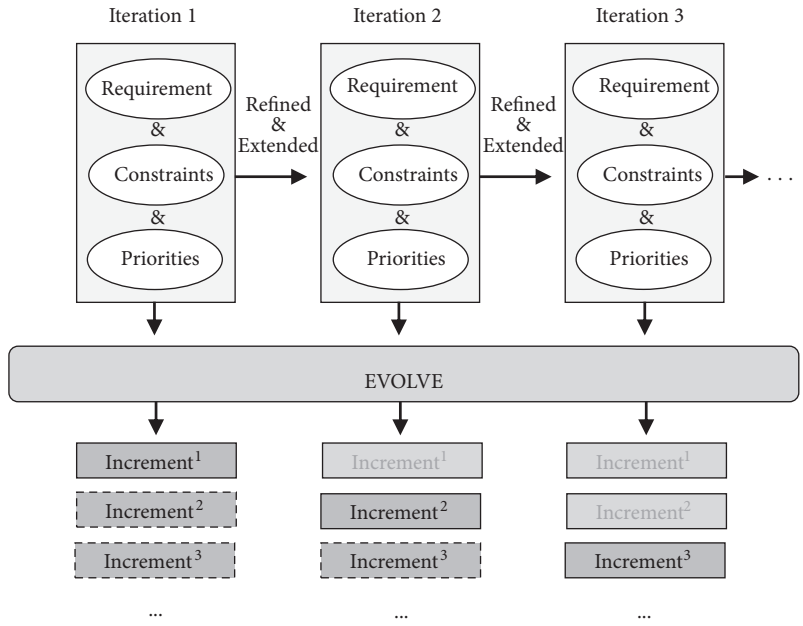

Figure 1: EVOLVE methodology [15].

because poorly designed planning can result in dissatisfied customers, planning of releases that do not comply with restrictions, and releases that do not add value to the business.

The selection of requirements for the framework in the next release should take into account the prerequisites of each requirement, customer satisfaction, customer priority, and the time and effort of implementing each requirement [4].

However, prioritization is relevant because users have expectations regarding software production. The delivery of requirements, according to users' perceptions, besides contributing to their satisfaction, may be essential for the continuity of the project [12].

Also, we highlight other aspects, such as the volatility that can influence the task of prioritizing the requirements. It takes sophisticated, organized work to select and prioritize volatile requirements. We understand that these volatile requirements are generally considered a problem to be solved. Requirements characteristics, such as repeated changes throughout the project, may impact during the software development process [13]. For example, a study conducted in [14] shows that volatile requirements represent a significant proportion of the problems handled by software development companies.

In the same research [14], the author points out that he did a study in the real environment of a software development company to verify the causes of volatile requirements and the effect of this on the projects of that company. In order of increasing importance, the authors identified that the reasons for changes in requirements are (a) changing system requirements, (b) discontinuation of requirements implementation, and (c) changing requirements specifications.

The authors also pointed out that these changes are due to external factors, such as (a) changes in market demands, (b) more critical understanding of the developers regarding the implementation of requirements, and (c) other client considerations, such as reduction of scope and changes in the policy of the company [13]. Therefore, in this work, we consider the stability of requirements as a selection criterion (function 2 of Section 6, below).

Greer [15] published a paper that uses genetic algorithms in Software Release Planning. The EVOLVE method, as defined, has two functions whose objectives are to maximize total benefit and minimize penalties. Therefore, the method assists in making decisions in environments subject to change. Each stakeholder has importance to the organization and assigns values of importance and urgency to each requirement. This approach considers the dependencies and technical precedence that may exist between requirements. In this scenario, the number of requirements is variable. To evaluate the proposal of the EVOLVE application, the author applied it in a case study with an abstract design. In Figure 1, we have the methodology of this method.

Prioritization of more stable requirements ensures that the requirements subject to change are implemented later. Of course, we understand that this leads to a reduction in the total cost of the project, as well as ensuring that deadlines are met since the first requirements delivered are more stable than the others.

Therefore, the task of allocation and prioritization of requirements is trivial for the success of the software development project. In the following sections, we present methods already known and tested in the area of Software Engineering that aim to find suitable solutions and present the proposal for this work.

3. Search-Based Software Engineering (SBSE)

In software engineering, we can find problems that are related to conflicting constraints, ambiguous and imprecise information within a broad set of choices or decisions.

Solving these problems is a complex task considering that there is no optimal solution [16]. If we take into account a set of requirements for composing multiple releases, the number of ordering possibilities of requirements for this problem can be pervasive. Also, the compositions of the solutions must comply with constraints such as customer satisfaction, time, and cost.

The solution to these problems has been addressed by a new field of research called *Search-Based Software Engineering* (SBSE), which is a research area that applies search-based optimization algorithms to automate the construction of solutions to software engineering problems [17].

In SBSE, optimization issues are part of the software engineering field. Research-based techniques, such as genetic algorithms, can solve this type of problem. The problems to be addressed are related to complex human tasks. SBSE provides automated search techniques that encapsulate human assumptions and intuition, thus reducing human effort. SBSE originated in 1990 when there was the application of the research-based technique to solve problems of software testing and project management. Harman and Jones first used the term SBSE in 2001 in an article that was considered a presentation of the SBSE, and since then, there are many articles published by this research community [16].

In the literature, we find several strategies that propose to solve the problems of SBSE. Among several, in this work, we choose those strategies that have a long time of existence and, therefore, more used and tested in the scientific community. Also, the formatting of the input data and the format in which they have after processing may be the same as those used by the VDA methods, seen later, which facilitates the comparison work between the results presented by the SBSE and the VDA methodology.

3.1. NSGA-II. The Nondominated Sorting Genetic Algorithm (NSGA) [18] has the unique characteristic of ordering the solutions through the layer in which they lie in front of Pareto. In this way, it maintains nondominated solutions from one generation to another, just as some dominated solutions can also remain.

The operation of the NSGA-II algorithm starts with a population P_0 and to this population is applied the technique *Fast nondominated Sort*, that looks for solutions near the front of Pareto. Each of the solutions receives a rank according to its level of nondominance. This rank defines the index of the frontier (front) to which the solution belongs. Through the standard operators of the genetic algorithms, a new population Q_0 is constituted. Generally, in generation t, we have Pt and Qt, the latter being generated through selection, recombination, and mutation operators.

The next step of this metaheuristic is to generate a population $Rt = Pt \cup Qt$. Then, the algorithm *Fast nondominated Sort* receives the population Rt, and the boundaries are generated. The solutions contained in $F1$ are the nondominated ones and form the set of candidate solutions ahead of Pareto. After determining the values of the fronts, the *crowding* values are calculated for the last frost, as follows. Let *n_sol* be the solution, where *n_pop* is the number of NSGA-II individuals belonging to a given *n_rank* front. If the solution

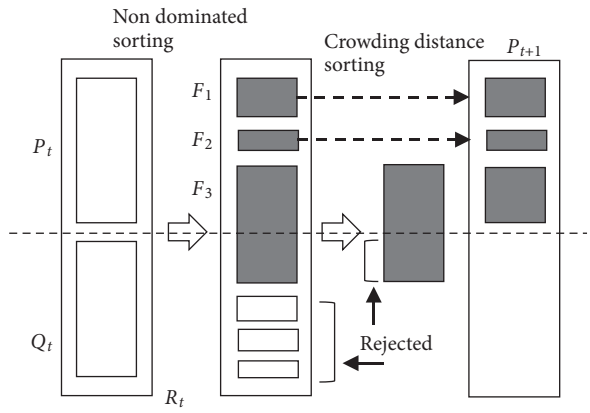

Figure 2: NSGA-II procedure [18].

n_sol + 1 belongs to the front *n_rank + 1*, then the computation of *crowding distance* is not necessary, since all the solutions until *n_pop* will be accepted for the next generation. But if the solution *n_sol + 1* belongs to the same front *n_rank*, then the computation of the *crowding distance* in each solution of the front *n_rank* is carried out and the solutions that will pass to the next generation will be the solutions to the front *n_rank*, being in this front will be chosen solutions that have the highest values of *crowding distance*.

About elitism, the solutions that belong to smaller fronts are initially preferred, and then the *crowding distance* is calculated in the front where this is necessary, and solutions with higher *crowding distance* values are preferred. Figure 2 demonstrates the methodology adopted in NSGA-II.

3.2. MOCell. The MOCell metaheuristic [19] is a multiobjective algorithm based on the GA (Genetic Algorithm) model. In the multiobjective case, this algorithm has a Pareto front. Furthermore, to implements insertions in the Pareto front in order to obtain a set of diversified solutions, a crowding distance estimator is used, as seen in the previous item. This method is also used to eliminate solutions from the Pareto front when this front is very dense.

This metaheuristic starts by forming an empty Pareto front. A two-dimensional Cartesian plane receives individuals. Until it reaches a stop condition, a reproductive cycle is applied. Thus, for each, the algorithm consists of selecting parents in the neighborhood, recombining them to obtain a descendant, mutating it, evaluating the resulting individual and inserting it in an auxiliary population (if this solution not is dominated) and in the front of Pareto. Closing, after each generation created, an old population is replaced by an auxiliary population, and a feedback process is initiated to replace a fixed number of randomly selected individuals of the population.

4. Verbal Decision Analysis (VDA)

The Verbal Decision Analysis (VDA) comprises a set of several methods for classifying and ordering alternatives, which consider multiple criteria in solving problems [20]. Therefore,

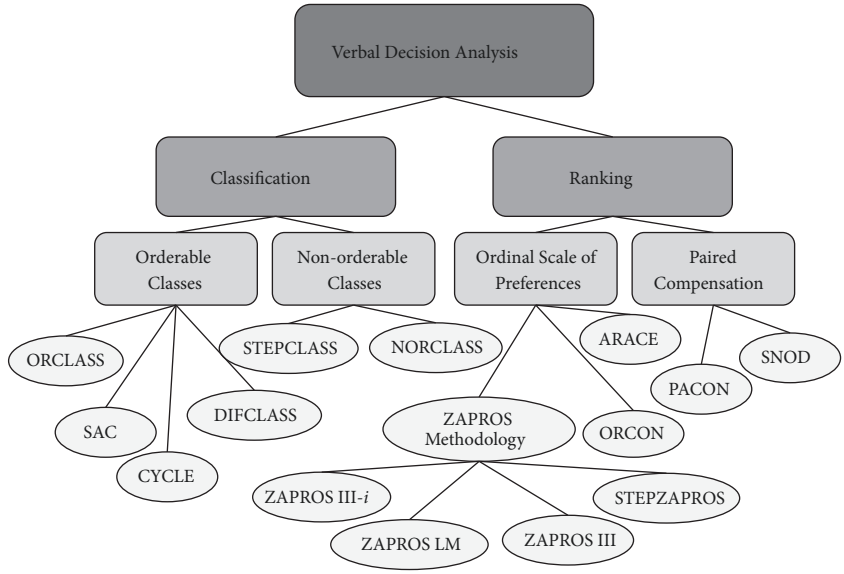

Figure 3: VDA methods for classification and ordering [19].

VDA proposes a systematic analysis and support of decisions based on verbal factors and qualitative analysis of attributes, rather than the commonly used quantitative methods. We do not need to perform any numerical conversion in this field. Figure 3 shows the VDA Methods for classification and sorting.

The methods that are part of the VDA framework have several features and benefits [21], among which we highlight the following: (i) the objective is to report problems; VDA methods offer a language that is natural for the decision maker; (ii) the methods include steps to process inconsistent entries in the preferences indicated by the decision maker, such as consistency inspection and unconditionality criteria; (iii) methods use transparent procedures from the decision maker viewpoint; (iv) they allow us to review the preferences that were given by providing explanations of the results generated; and (v) the methods use the verbal information strategy to produce preferences, which allows them to develop psychologically acceptable measures considering the view of the decision maker.

We also emphasize that Verbal Decision Analysis has a high insertion in problems that have a substantial number of alternatives and a reasonably small set of criteria and their values.

The use of these strategies, especially the ZAPROS III-*i*, to a given problem presents a significant amount of solution possibilities. Besides, it happens due to the various combinations of criteria values to have situations to be evaluated, where at the end of the process, they deal with a decision rule. This excessive amount of combinations leads to the stages of preference elicitation and comparison of alternatives to a sophistication that would be difficult to perform manually [22].

Another method we can find among those that exist in VDA is the ORCLASS (Ordinal Classification) method that classifies multicriteria alternatives, which are the options to determine the problem according to the criteria provided. The method aims to classify a set of alternatives to a small number of decision classes or groups. The decision maker can rearrange these groups according to their preferences [23].

As described in detail in Section 5, this paper proposes to classify and order software requirements using two methods covered by VDA: ORCLASS and ZAPROS III-*i*.

4.1. ORCLASS Method: Overview and Structure. In VDA, we can find many different methods to solve multicriteria classification problems, some described in Figure 3. The ORCLASS methodology aims to classify the alternatives in a given group: DM needs these alternatives to be categorized in a small number of decision classes, usually two. According to [24], it presents a flowchart with steps to apply the ORCLASS method in Figure 4. The application of this method takes place in three distinct stages: Problem Formulation, Classification Rule Structure, and Analysis of the Information Obtained [25].

The methodology adopted in ORCLASS follows the same problem formulation proposed in [24, 25].

(1) $M = 1, 2, \ldots, N$, illustrating a set of N criteria.

(2) n_p describes the number of possible values on the scale of p-th criterion, (q ∈ K); for the problems provided, as in this case, generally $n_p \leq 4$.

(3) $X_p = \{x_{ip}\}$ describes a set of values to the p-th criterion, which is on this criterion scale; $|X_p| = n_p$ $(p \in K)$.

They are ordering the best range of values for the worst, and this type of order is independent of the values of other generated scales.

(4) $X = Y_1 \times Y_2 \times \ldots \times Y_n$ represents a set of vectors x_i (every possible alternative: real alternatives + supposed alternatives) such that $x_i = (x_{i1}; x_{i2}; \ldots; x_{iQ})$, and $x_i \in X$, $x_{ip} \in Y_p$ and $Q = |X|$, such that $|X| = \prod_{p=1}^{Q} n_p$.

(5) $B = \{b_i\} \in X$, $i = 1, 2, \ldots, t$ in which the totality of t vectors illustrates the definition of the real alternatives.

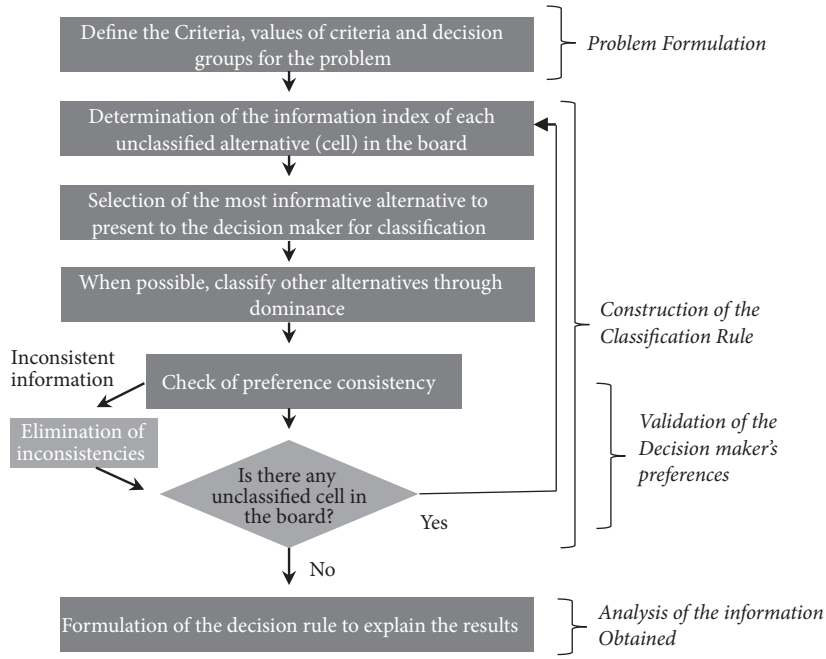

Figure 4: ORCLASS methodology [25].

(6) $C = \{C_1, C_2, \ldots, C_L\ is\}$ representing the set of ordered decision classes.

As presented in [25], the creation of the classification rule will be executed based on the preferences of the decision maker. To show the use of the method extensively, we consider the same ideas presented in [24], which presents a classification task of a set of alternatives.

Each element is composed of a combination of values for each criterion of the problem, which means a potential alternative to the problem. We take into account a problem with the criteria X, Y, and Z, the criteria values: X = {X1, X2, X3}, Y = {Y1, Y2, Y3}, Z = {Z1, Z2, Z3} and two decision classes (1 and 2), so that, according to the decision maker, the first class is preferable to the second one. For the given problem, we can say that an alternative composed by the best characteristics ([X1, Y1, Z1]) will always belong to Class 1, and an alternative that has the worst peculiarities ([X3, Y3, Z3]) will always be in Class 2.

As defined in [24, 25], it is shown to the decision maker to choose, an alternative that has criteria values [X1, Y1, Z3]. Assuming that the decision maker fits the alternative into Class 1, then we can deduce that the better alternative, like [X1, Y1, Z2], certainly belongs to the same Class, since the latter is indeed more preferable than the first. This solution would define two cells in the leaderboard. However, if the decision maker decides that the alternative [X1, Y1, Z3] belongs to Class 2, then these harmful alternatives will be inserted into group 2. Thus, alternatives [X1, Y2, Z3], [X1, Y3, Z3], [X3, Y1, Z3], [X2, Y2, Z3], [X2, Y3, Z3], and [X3, Y1, Z3] are in Decision Class 2, since they are less desirable than the previous one.

In this work, we consider two possible classification groups of requirements: "implementable" and "not implementable."

We justify the use of the ORCLASS classifier in this work because it can anticipate a selection of requirements for implementation. As seen, this is an excellent classifier widely known in the VDA field and has an interface developed to facilitate use by the decision maker. In previous works [8–10] all 20 requirements of a software problem were immediately prioritized for implementation; that is, there was no selection to predefine requirements that somehow or should not go to the prioritization queue. Thus, if less relevant requirements do not go to the prioritization list, fewer questions will be presented at the stage of choice (seen later) to the decision maker. Also, it makes this process more comfortable and faster. In previous works, when we did not use this classification tool, we verified a high index of requirements that are no longer implemented due to lack of resources, since this is one of the constraints of this project. In this way, the insertion of a classifier already marks some requirements as the type "will be implemented later," and this is an everyday activity in development environments. We note that this is different from promising requirements, but will not be implemented because there are not enough resources, and this serves as additional information for decision makers.

Additionally, to facilitate the decision-making process, a tool hosted on a website (http://www2.unifor.br/OrclassWeb) was created by [25]. The OrclassWeb tool serves as a means of data entry and facilitates the follow-up of the methodology of this work because it already has the ORCLASS method incorporated into it.

4.2. ZAPROS III-i Method: Overview and Structure. The critical point in the decision-making process for ordering software requirements is that the project manager or the decision maker (DM) achieves a satisfactory result, taking into account a set of characteristics that make up those requirements. These alternatives often have similar characteristics. The use of methods that support DM can help mitigate the likely impacts of bad decisions [26]. VDA and its methods follow this trail to show alternatives to problems in natural and friendly language. In VDA, we can count on many methods in this sense, among which there is ZAPROS III-*i* [26], which makes the process of eliciting preferences less inconsistent with previous methods [26]. This methodology uses the preference elicitation of values that represent the distances between the evaluations of two criteria, called Quality Variations (QV). It uses the Formal Quality Index (FIQ), which orders the alternatives to minimize the number of comparison of pairs of alternatives, in order to obtain a result for the problem [27]. Some of these alternatives may be unmatched, and this leads to unsatisfactory results in decision-making models. Thus, the ZAPROS III-*i* method had similar origins to ZAPROS III, but it has differences in the methodology of comparison of alternatives aiming to increase the performance of the decision method [27]. Thus, the use of the ZAPROS III-*i* methodology as a way to solve problems of ordering software requirements can be promising, since this method also takes into account the opinion of the DM.

ZAPROS III-*i* is a method that composes the VDA and aims to group alternatives in a context where there is a small set of criteria and their respective values and many alternatives available. The method gets preferences around values that represent the distances between criteria. A preference scale can be constructed, allowing the comparison of alternatives [28].

As shown in [28], the ZAPROS III-*i* method occurs in three steps: Problem Formulation, Elicitation of Preferences, and Comparison of Alternatives. In the first step, we obtain the criteria and their values relevant to the decision-making process. In the second step, we generated a preference scale based on DM preference. The process occurs in two stages: (i) obtaining preferences for QV of the criterion and (ii) preference elicitation between two criteria. Finally, the method compares alternatives based on the preferences of the decision maker. For details on the procedure, see [28].

To use this method, the methodology that we adopted in this research is the same one used in the formal declaration of [21, 28]:

(1) $M = 1, 2, \ldots, N$, illustrating a set of N criteria.

(2) n_p describes the number of possible values on the scale of p-th criterion, (q ∈ K).

(3) $X_p = \{x_{ip}\}$ describes a set of values to the p-th criterion, which is on this criterion scale; $|X_p| = n\ p\ (p \in K)$, where the values entered in the scale are sorted from the most relevant to the least relevant. This ranking is independent of the criteria values that exist in the other scales.

(4) $X = Y_1 \times Y_2 \times \ldots \times Y_n$ represents a set of vectors x_i such that $x_i = (x_{i1}; x_{i2}; \ldots; x_{iQ})$, and $x_i \in X$, $x_{ip} \in Y_p$ and $Q = |X|$, such that $|X| = \prod_{p=1}^{Q} n_p$.

(5) $B = \{b_i\} \in X$, $i = 1, 2, \ldots, t$ in which the totality of t vectors illustrates the definition of the real alternatives.

The flowchart with the procedure used in the ZAPROS III-*i* method to order a set of alternatives is shown in [28] and in Figure 5. In the first step, Formulation of Problems, through the decision-making process, the essential criteria and their values are obtained. In the next step, Elicitation of Preferences, the preference scale is constructed based on the DM preference. As mentioned, this step occurs in two steps: (i) obtaining preferences for QV of the criterion and (ii) preference elicitation between two criteria. Finally, in the last step, we have the Comparison of Alternatives, where the alternatives are compared based on the preferences of the decision makers.

In the Elicitation of Preference phase, the DM's answers allow classifying all the quality variations (QV) of the scales of two criteria. We call this the Joint Scale of Quality Variation (JSQV) for a pair of criteria. All criteria are subject to the same procedure. In the end, the preference scale for quality variations (JSQV) for all criteria is constructed [28].

In order to demonstrate the flow shown in Figure 5, we briefly present that the task of extracting DM preferences consists of comparing all the Quality Variations (QV) obtained on two criteria scale through questionnaires made to it. After obtaining all QV's, we have a joint result and the Joint Quality Variation Scale (JSQV), e.g., z1 ≺ x1 ≺ y1 ≺ x2 ≺ y2 ≺ z2 ≺ x3 ≺ y3 ≺ z3. As each criterion has its own set of alternatives (A1 = {X1, Y2, Z2} and A2 = {X2, Y1, Z2}), we can compare these alternatives concerning JSQV and obtain a Formal Index of Quality (FIQ) each of these alternatives, e.g., A1 = 3 and A2 = 9. In this example, we have that A1 is more desirable than A2. Reference [28] presents a detailed explanation.

As shown in Figure 4, Verbal Decision Analysis has solving methods for different ranking problems using natural language as input to the solution search. In previous research [8–10], we noticed that the alternative ranking generated by the ZAPROS III-*i* methodology was satisfactory for decision makers. The ease of application of this method and its adaptation to the problems proposed in this work lead us to adopt it in our research.

Also, the ARANAÚ [29] tool was developed to support the decision-making process. The tool was initially implemented to support the ZAPROS III methodology. In this research, we make use of an updated version that already contemplates the method ZAPROS III-*i*. The ARANAÚ tool serves as a form of data entry and facilitates the monitoring of the methodology of this work since it already has the ZAPROS III-*i* method embedded in it.

5. Materials and Methods

In previous works, we have obtained satisfactory conclusions when comparing the results obtained by the ZAPROS III-*i* methodology about the results obtained by the SBSE methodologies for the same problem. As already described,

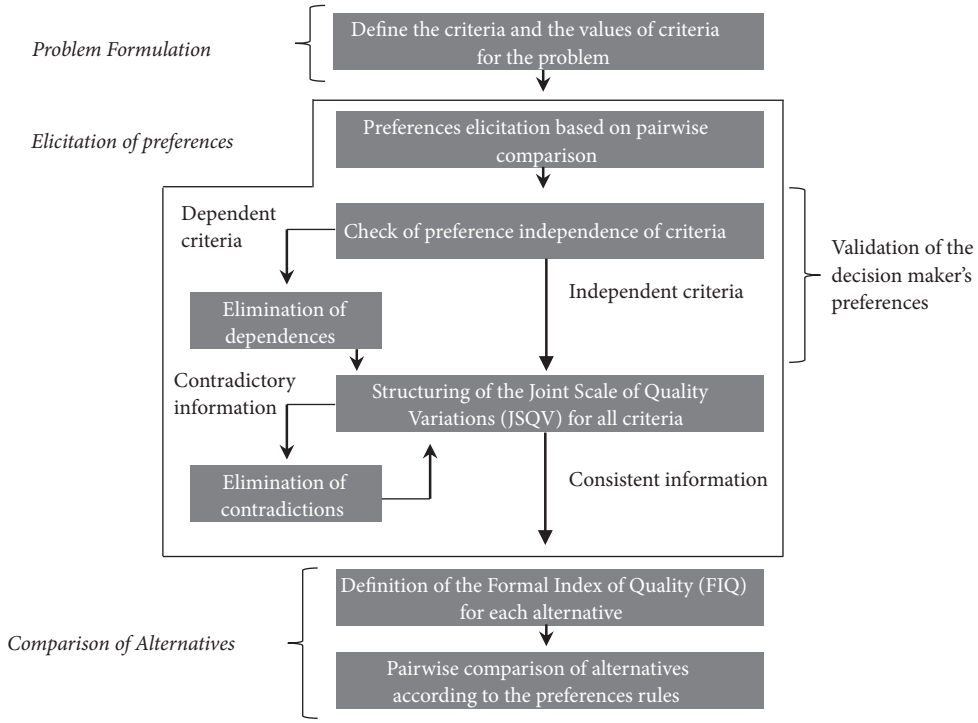

Figure 5: Procedure to apply ZAPROS III-*i* method [28].

the results generated by the VDA method do not exceed those generated by the SBSE methods, but when approaching them, it makes the VDA field an exciting and safe alternative to performing the requirements prioritization task.

However, selecting and prioritizing many requirements in a software project can become an exhausting task. It is common for software development companies often to have a budget that is less than the sum of the resources needed to implement all project requirements. Therefore, the implementation of all project requirements is not guaranteed. In this way, it is necessary to optimize available resources and ensure that at least the most essential requirements are implemented and delivered to the most critical customer group for the software developer company. If we can classify these requirements in advance between an "implementable" group and a "nonimplementable" group, we can reduce the number of requirements in the prioritization step. This classification follows predetermined criteria by DM and will act in the process as a filter where only the requirements with higher chances of implementation will be executed.

Figure 6 shows the workflow used in this work. Initially, we used an instance generator to generate simulations. These simulations have empirical data that demonstrate a picture similar to that faced by DMs in real software projects. In Section 6, we will present in detail this stage of the work. After that, in the first flow, the generated requirements and configurations are submitted to the classification and ordering process using the SBSE methods. In this case, we use the NSGA-II and MOCell metaheuristics. These metaheuristics received the same data and generated distinct and independent results from each other. Also, this is for purposes of comparison of results, as we can buy the results of VDA with more than one metaheuristic. In Section 7, we describe the methodology of this work. Parallel to this workflow, the data submitted to the search for solutions to the SBSE were submitted to VDA methods. First, we use the ORCLASS algorithm as a classifier, and then we use the ZAPROS III-*i* algorithm to sort the requirements into software releases. Section 8 describes the methodology used by VDA methods. After obtaining all the results, the DM had the opportunity to evaluate the generated results. We draw comparative charts and discuss the solutions found in Section 9.

6. Problem Generation

As previously shown, we use data that simulate situations faced by software development companies. For this, we try to use mathematical modeling that allows the generation of data in this context. The generation of these scenarios occurs through software developed for this. After the entered numerical parameters, it randomly generates data representing four software projects. For purposes of comparisons with previous papers [8, 9], we maintain the same mathematical formulation as shown below:

$$\max \quad f_{VALUE}(y) = \sum_{i=1}^{N} S_i \bullet y_i, \tag{1}$$

$$\min \quad f_{VOLATILITY}\left(x^{Pos}\right) = \sum_{i=1}^{N} \left(B_i \bullet x^{Pos}{}_i\right) \bullet y_i, \tag{2}$$

$$\text{Subject to:} \quad x^{Pos}{}_i < x^{Pos}{}_j \tag{3}$$

$$\sum_{i=1}^{N} cost_i \bullet y_i \leq R \tag{4}$$

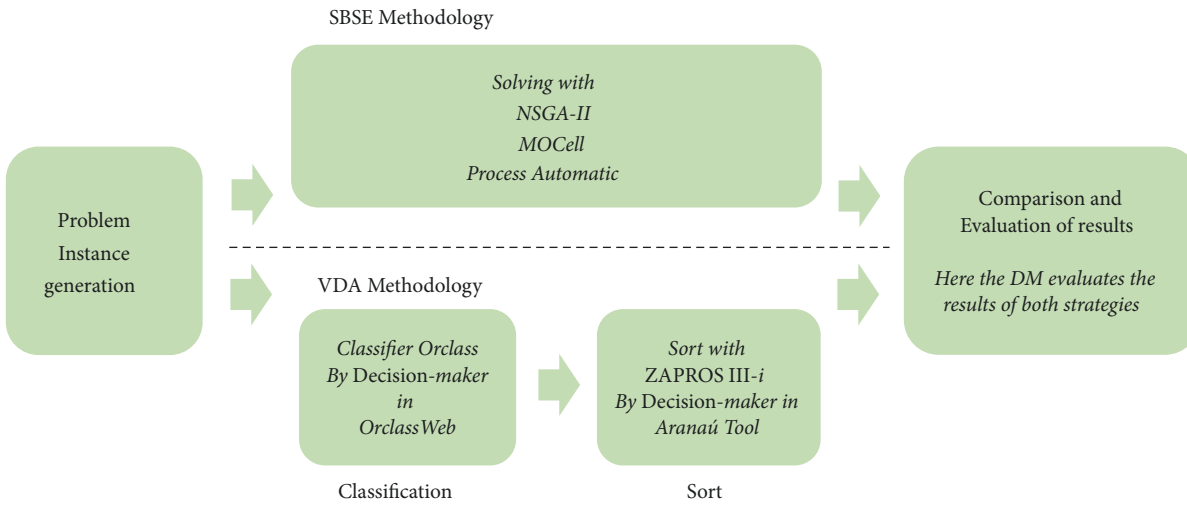

Figure 6: Workflow.

The formulations of the objectives are represented in functions (1) and (2), as follows.

In (1), max $f_{VALUE}(y)$ is objective of maximizing customer satisfaction and aggregate business value (S_i), where the score $S_i = \sum_{m=1}^{W} w_m \bullet Value\ (m, i)$ expresses the business value to the requirement r_i. In this way, and considering the importance $Value\ (m, i)$, with which the client w_m associates a requirement r_i, the function gains value when choosing more requirements. Thus, the essential requirements (m) and those that respect the technical precedence (yi) (according to function (3)) are chosen for implementation according to the most valuable customers for the development company.

Equation (2) describes the level of stability B of the requirements of project i by developing the requirements considered to be more stable. In this way, the function calculates the product between the position for which it was allocated and the stability of the requirement. Also, $x^{Pos}{}_i$ is the position of the requirement (r_i), which can have a value between $\{0, 1, 2, .., N\}$, in the order of implementation defined by prioritization, for $i = 1, 2, ...N$, where one 1 shows that the requirement has a higher priority than others. Therefore, a low value for this function indicates the prioritization of more stable requirements. Thus, this function increases the stability level of the project requirements, initially performing the requirements with more excellent aggregate stability.

The mathematical wording for the constraints is as follows.

Equation (3) represents the constraint of precedence between the requirements, wherein the case presented above, r_i technically precedes r_j. If a requirement r_i precedes a requirement r_j, then r_i must be implemented before r_j ($x^{Pos}{}_i < x^{Pos}{}_j$).

Equation (4) represents the cost constraint of implementing the requirements to the available budget R, where the variable y_i indicates whether the requirement r_i will be implemented ($y_i = 1$) or not ($y_i = 0$), to $i = 1, 2, ...N$. Commonly, the total cost to develop all project requirements may be higher than the total resources available and allocated to the project itself. Due to this budget constraint, some requirements may fall outside the group of requirements implemented. Therefore, it is crucial to develop the most critical requirements while still considering the most valuable customer for the development company.

In a software project, the number of requirements is relative and depends on the size of each project. In research using only the methods that make up the SBSE, we can find reasonable solutions, even considering projects with 2,000 requirements [30]. In the VDA field, we know that the number of criteria is still somewhat restricted because we consider, for example, the human effort to respond to questionnaires that contain many criteria and their respective criteria values [8, 9]. In order to maintain the isonomy between SBSE and VDA methods, in this work, we consider that the characteristics of the requirements (SBSE) are criteria in VDA. Thus, we consider that the four files generated in the simulations contain 20 requirements. We hope to be able to work in the future with VDA methods that can support more than 20 requirements. Our research is moving in this direction.

As stated earlier, we keep the resource value available to the project less than the sum of the cost required to execute all 20 requirements. This constraint forces the algorithms to seek the best solutions considering the objectives already exposed. In this case, we keep the number of resources available between 70% and 80% of the total amount required to execute the entire project, which in this case corresponds to the sum of the cost of the 20 requirements. We considered five the number of customers interested in the project. Each client carries weight to the software developer company, and each client has their preferences for each of the 20 requirements assigning scores to them. The goal here is to implement the most critical requirements for the most critical customers.

Technical precedence between requirements is something natural in any software project. In this way, we consider this aspect of this work. We set a level of technical precedence between requirements of 10% to 20%.

Table 1: Settings adopted in this work.

File	Number of project requirements	Number of project clients	Available budget	Percentage of technical precedence between requirements
File 1	20	5	70%	10%
File 2	20	5	80%	10%
File 3	20	5	70%	20%
File 4	20	5	80%	20%

Table 2: Representation of the data generated for file 1.

Value	Description
20	Number of requirements
05	Number of customers interested in the project requirements
05	Release Number
1: 49, 2: 50, 3: 47, 4: 36 and 5: 12	Individual cost of Releases
1: 08, 2: 03, 3: 05, 4: 05 and 5: 09	Score that measures the importance of the stakeholder (customer) to the company *(value 01: low importance and value 10: high importance.)*
1: 13, 2: 13, 3: 17, 4: 17, 5: 11, 6: 12, 7: 18, 8: 15, 9: 11, 10: 16, 11: 13, 12: 18, 13: 12, 14: 10, 15: 15, 16: 12, 17: 11, 18: 18, 19: 14 and 20: 12	Cost of each requirement *(value 10: low cost and value 20: high cost.)*
1: 7, 2: 7, 3: 3, 4: 3, 5: 1, 6: 10, 7: 4, 8: 3, 9: 2, 10: 10, 11: 1, 12: 9, 13: 3, 14: 10, 15: 2, 16: 6, 17: 5, 18: 5, 19: 1 and 20: 6.	Stability value for each requirement *(value 1: high stability and value 10: low stability.)*
8 ⟶ 9 and 6 ⟶ 17	Technical precedence between requirements, which for file 1 is 10%
1° Customer 4 8 2 8 1 10 8 8 4 6 9 6 8 6 3 8 5 8 10 4 2° Customer 3 1 10 3 2 10 3 8 8 7 10 2 10 1 8 7 10 3 4 1 3° Customer 4 8 2 4 7 3 2 4 10 6 5 7 9 10 3 8 2 10 5 10 4° Customer 6 5 2 5 6 9 7 3 6 5 3 4 1 9 9 5 2 4 10 2 5° Customer 2 9 4 8 4 8 9 6 3 8 4 10 8 8 1 8 3 5 3 2	This is the score that measures the value that the customer gives a requirement. *The position of the indicated value represents the identification of the requirement. For example, 4 is the value that customer 1 gives for requirement, and so on.* *(value 01: low importance and value 10: high importance.)*

The configurations described so far can be summarized, as shown in Table 1.

To illustrate, Table 2 demonstrates the content generated for file 1, which represents one of the four software projects used in this research. An instance generator generated the values presented in Table 2. These values are empirical and defined within a scale of maximum values, and minimum values appropriate to the context of this research. These values are empirical and simulate situations that software development firms face, such as constraints and contradictions.

Table 2 presents the values and characteristics of the 20 requirements of a project. To illustrate the understanding of this table, we will take as an example of requirement 1. The table shows that this requirement has a cost value of 13, considering a scale of 10 (minimum cost for implementation) to 20 (high cost for implementation), so, we can deduce that this requirement has "low cost for implementation." For the stability characteristic of requirement 1, the table shows the value 07, on a scale of 01 to 10 points, where 01 represents a more stable requirement, and 10 is a less stable requirement. Less stable requirements may change throughout the project.

In this project, we estimate the level of technical precedence between requirements by 10%, i.e., 2 of the 20 requirements will take precedence over other requirements. In the case of requirement 1, the table shows that it has no technical precedence; only requirements 06 and 08 have this characteristic. This same table shows the level of interest of 05 clients by the project requirements. Each customer individually selects their preferences for each requirement employing a score. In case of requirement 1, clients 1 and 3 reported 4 as preference level, considering a scale of 1 (least preference) to 10 points (most preferred). Customer 2 and 5 reported 3 and 2 respectively, and customer 4 reported 6 as their preferred level. In this way, we noticed that four clients reported scores below 5 points. Besides, this demonstrates that requirement 1 is not very important to most of these customers. With this information, the algorithm can leave this requirement to be implemented in later releases. However, this still depends on

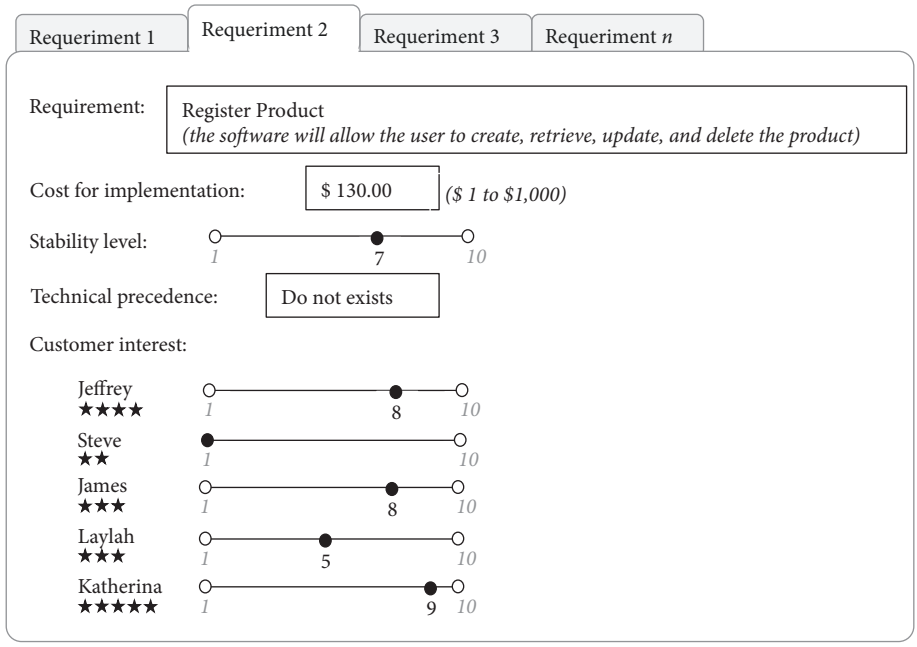

Figure 7: Illustration of requirement 2.

other variables so that the algorithm can decide on which release this requirement should be implemented. Table 5 indicates that release 2 contains requirement 1.

Figure 7 presents a second illustration for us. It refers to requirement 2, in this case, called the "Product Registration." This requirement has a cost of $ 130.00, in amounts proportional to Table 2. The stability level of this requirement is 7 scores, and it does not have technical precedence with another requirement. As shown in Table 2, each project client has a level of importance to the development company, where the opinion of a director or manager may be more important than that of a supervisor, for example. In this case, Figure 7 illustrates with stars the level of importance of these clients. Each customer still has their own opinions about each requirement. We show in scale which level of importance each customer gives to the requirement "Product registration." The number of releases and the cost required to implement these releases is not shown for requirement 2 because they are global project information.

Following the mathematical formulation presented at the beginning of this section and considering the four generated files presented in Table 1, we performed our experiment following the methodology shown in Figure 6 in order to find suitable solutions using VDA methods, wherein this work we use a hybrid proposal that uses a classifier method and a ranking method. In parallel, we performed these experiments with the same data using metaheuristics (SBSE) to generate a baseline and allow us to compare with the results obtained by the VDA methods. We do not expect the results of the VDA methods to be better than the SBSE methods, because SBSE has a tradition in solving this type of problem, but when we use VDA methods to solve this type of problem and, the results approximate the results generated by SBSE methods, we have a profit for the VDA field because we now know that this field can solve requirements selection and sorting problems.

7. Use of the SBSE Methodology

As mentioned, in this work, we use two metaheuristics of the SBSE field: NSGA-II and MOCell. Both were used in previous work [8–10] as a means to compare their results with the results obtained by VDA methods for the problem of requirements prioritization in an empirical software development project. The results show that SBSE methods are better than VDA methods. However, we expect this, but we are interested in getting a baseline, or Pareto front, as a benchmark comparison to the results of VDA methods. According to Table 3, in this work, we use the following configurations for the NSGA-II and MOCell metaheuristics.

As a support tool for the application of metaheuristics on the problem, we use the jMetal framework [31]. This framework has a wealth of metaheuristics implemented in the Java language.

Thus, after obtaining the data produced by the instance generator, we configure jMetal with the characteristics proposed by the mathematical formulation (objectives and constraints). To avoid out-of-context solutions, we perform ten executions for each of the four files in Table 1. Each metaheuristic can generate dozens of possible solutions in one run. Each generated solution represents an ordered sequence of requirements. We used Cartesian points to present the solutions. In this way, we have a set of Cartesian points for each metaheuristic. As we are dealing with a multiobjective

Table 3: Metaheuristic parameters.

Metaheuristics	Parameters
NSGA-II	(i) Initial population: 500 individuals; (ii) Maximum number of evaluations: 1,000,000; (iii) Crossing: 0.9; (iv) Mutation: 1.0; (v) For selection, we use the binary tournament method.
MOCell	(i) Initial population: 256 individuals; (ii) External size file: 256; (iii) Number of evaluations: 102,400; (iv) Feedback: 20; (v) Crossing: 0.9 (rate); (vi) Mutation: 1.0 (rate); (vii) For selection, we use the binary tournament method.

problem, we obtained a Pareto front in which, on the one hand, we have a set of requirements sequences that satisfy more the preferences of the clients and in the other those solutions that have requirements that are less volatile (more stable). Also, to assist in the visualization of these data, we presented graphs with these solutions in Section 9. These kinds of problems are part of the metaheuristics work plan. The procedures adopted for the use of metaheuristics were more direct than the procedures necessary to adapt the VDA to solve this type of problem.

8. Use of the VDA Methodology

The task of classifying software requirements using structured methods in VDA is relatively new. The methodology used in this work was attested in previous studies [8–10] and proved to be useful in solving quantitative problems common to SBSE by qualitative methods present in VDA. It was, therefore, necessary to create a specific methodology for this adaptation. In order to favor the progress of this version of our research, we used the OrclassWeb, and Aranaú tools support for the use of ORCLASS and ZAPROS III-*i* algorithms, respectively.

8.1. Adaptive Setup. The data of this work are empirical and generated through an instance generator that creates files (Table 1) that simulate a software development project scenario. These files contain a set of software requirements and their technical characteristics (cost, stability, stakeholder value, and stakeholder requirement value). Since they are quantitative files, they are represented in numerical form. For example, to represent the cost associated with a given requirement, a value is set on a scale of 10 to 20, where 10 represents the lowest cost for implementation and 20 is the highest cost required to implement that requirement. Moreover, this follows for the other generated parameters. To adapt quantitative data to qualitative methods, we need to create an adaptive table to help insert these types of numerical data into the qualitative methods platforms ORCLASS and ZAPROS III-*i*.

Following in the adaptation of the problem, in VDA, the characteristics of a requirement are called "criteria". Thus, the requirement *n* has "criteria" cost, stability, among others. Each "criterion" of this requirement has the corresponding values assigned to the requirement, in VDA known as "Criteria Value". Table 4 presents the table used to perform this conversion of numerical values into qualitative ones. Besides, to know the adaptation adopted, we include in this table the quantitative values of the requirements equivalent to the qualitative values considered in this work.

Thus, if the requirement *n* has twelve assigned to its implementation cost, it will be allocated in item 2.2 of Table 4. In the end, this requirement will no longer have numerical values but qualitative values. With the numerical data converted into qualitative data, we can proceed to the application of the ORCLASS classifier algorithm.

8.2. Applying ORCLASS. The ORCLASS method ranked the 20 requirements generated in two classes, "implementable" and "unimplementable." For this, this platform received the data of these requirements. As in previous work [10], we invited 12 project managers with at least one year of experience in the field of requirements engineering. These DMs were informed about the purpose of this work and agreed to respond to the qualitative questionnaires presented by the OrclassWeb tool.

Taking into consideration their experience, project objectives, constraints, and requirements characteristics, the project manager answered questions related to preference elicitation. At the end of this questionnaire, OrclassWeb generated a classification report of these requirements. Figure 8 shows the OrclassWeb.

As the purpose here is to use only the 'implementable' requirements or at least have a chance to be implemented in the project (based on constraints as the available resource), we submit to the next step (Aranaú tool) only that group of requirements.

8.3. Applying ZAPROS III-i. After defining the requirements classified as "implementable", we can make use of the Aranaú tool as an instrument for the application of the ZAPROS III-*i* algorithm. In this way, we insert in the tool only this set of requirements for the four generated files (Table 1).

Table 4: Adaptive setup table.

Criteria	Equivalence	
	Criteria Values	Quantitative values
1 Stability	1.1 The requirement will hardly be changed	1, 2
	1.2 The requirement may change, but there is little possibility of this happening	3, 4, 5
	1.3 The requirement can change and that there is a high possibility of this happening	6, 7, 8
	1.4 The requirement will be changed	9, 10
2 Cost	2.1 The requirement has a low implementation cost	10, 11
	2.2 The requirement has a substantially low implementation cost	12, 13, 14
	2.3 The requirement has a substantially high implementation cost	15, 16, 17
	2.4 The implementation cost of the requirement is high	18, 19, 20
3 Stakeholder	3.1 The stakeholder is valuable to the company	9, 10
	3.2 The stakeholder is of high importance for the company	6, 7, 8
	3.3 The stakeholder has partially low importance for the company	3, 4, 5
	3.4 The stakeholder is of low importance to the company	1, 2
4 Value of the requirement for the customer	4.1 The requirement has excellent value for the customer	9, 10
	4.2 The requirement has a substantially high value for the customer	6, 7, 8
	4.3 The requirement has a substantially low value for the customer	3, 4, 5
	4.4 The requirement has low value for the customer	1, 2

Table 5: Result for file 1 for the VDA methods.

Rank	1	2	3	4	5	6	7	8	9	10	11	12	13
Requirement	14	06	02	16	20	08	01	13	10	04	12	03	05
Release	1	1	1	1	2	2	2	3	3	3	4	4	5

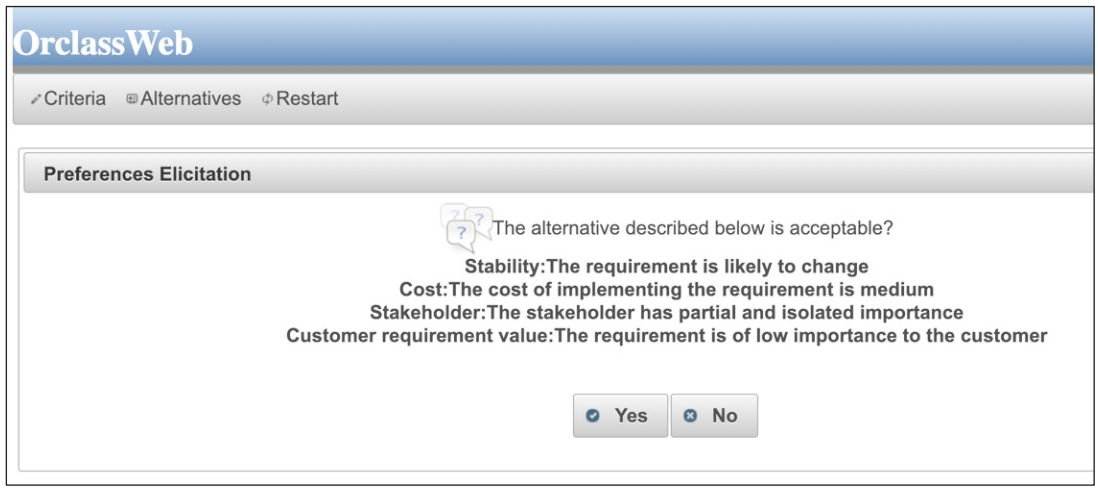

Figure 8: OrclassWeb view.

Once again, the 12 DMs invited in the previous stage answered the questionnaires issued by the Aranaú tool (Figure 9) regarding issues related to the project and were able to insert in this tool their personal preferences. These guests were aware of the project's goals and constraints.

In the field of Verbal Decision Analysis, the methods work with natural language and present to the DM alternatives to the problems succinctly and objectively to choose. The Aranaú tool (ZAPROS III-*i*) presents several alternatives in order to solve a sorting problem. The possibility of using the subjectivity of an experienced DM can make a difference in the process of choosing requirements and the order of implementation of those requirements in Software releases. Also, this algorithm allows treating inconsistencies generated between alternatives when they appear, showing them to the DM and asking for the best solution to solve it. As this method is objective with the presentation of simple questions, each DM solved all four problems proposed in this work in an average time of fewer than 10 minutes, which made the process fast and not tiring.

At the end of the application of the preferences forms for the four projects of this work, the DMs were able to know the solution generated by the VDA method. The DMs could then evaluate the solution generated by assigning it a score.

The next section presents the results generated by both methodologies, SBSE and VDA, as well as the evaluation of DMs.

9. Results and Discussion

The results expressed by the SBSE methods represent dozens of solutions to the same problem. In a single result set, we can have different order sorting alternatives. In VDA, the solution generated for a file represents only one alternative that took into account what was applied in the ORCLASS and ZAPROS III-*i* by the DM.

Table 5 shows the values corresponding to a single result generated by the VDA methods for file 1. We can observe that the first requirement to be implemented is 14 and 05, the last one. The table also shows the release number in which the requirement will be implemented. The cost of the release set implementation value must be less than or equal to the total value assigned to the project. Release 1 will be the first to be implemented and therefore has the highest priority requirements.

Interestingly in this project (file 1) Release 1 has the most requirements. Due to the technical constraints of the project, such as the low rate of the importance of the requirement by the client, budget or other issues inherent in the software development process, the other requirements not presented in Table 5 will not be implemented by the project. One of the constraints of this work is that technical precedence between requirements must be respected. We take this into account during the requirements prioritization process. Therefore, in the raking presented in Table 5, the requirements were ordered, considering still the technical precedence between them.

All the care was taken in the stages of this project, in order to keep it coherent and isonomic as to the application of the methodologies. Also, this guarantees us fair results so that we can make an equal evaluation of them in the end.

As a constraint measure, the available budget ranged from 70% to 80% of the total project value of each file. Thus, the methods had to work to optimize the available resource by implementing as many requirements as possible. In Table 6, we present the efficiency index of these methods about the

Table 6: Level of efficiency in resource use for VDA and SBSE methods.

File	Total cost required	Available budget	VDA methods		SBSE methods	
			Budget used	%	Budget used	%
File 1	278	194	194	100%	184	94,8%
File 2	288	230	220	95,6%	219	95,2%
File 3	306	214	204	95,3%	214	100%
File 4	275	220	205	93,1%	210	95,4%

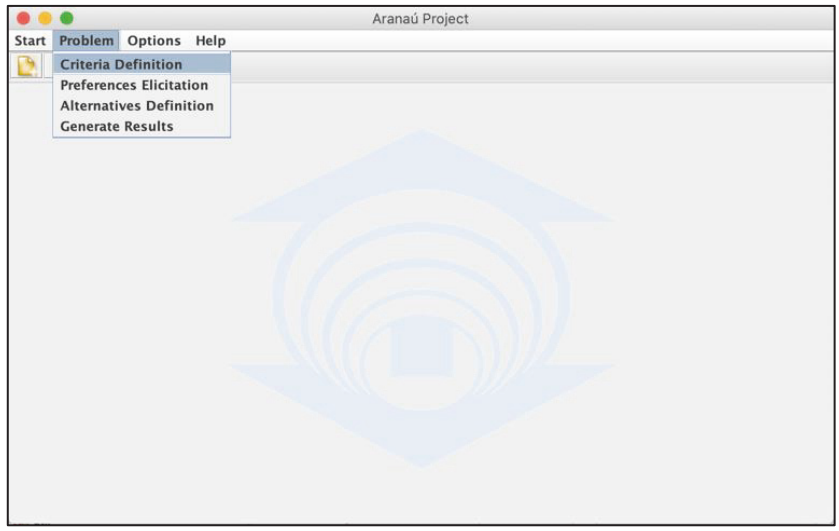

Figure 9: Aranaú tool.

use of resources. We note that for File 1, the methodology of this work using VDA has achieved 100% efficiency in the use of resources. This same methodology reached for the file 4, 93.1% the lowest among the tests. As in SBSE, we have two different methodologies; we adopted for file 1 and 2 the results of NSGA-II and for files 3 and 4 the results of the MOCell method. These methods achieved 100% efficiency for file 3 and 94.8% for file 1. When we had resources that were not fully used, it means that the remaining resource was not enough to fully implement the next requirement. In general, we have had relatively balanced results here showing that the methods, although different, can achieve comparable results.

We organized the results generated in both methodologies (SBSE and VDA) in spreadsheets. After that, these results were consolidated and presented in graphs to give of the comparison of results. In a Pareto front, we present the results of the NSGA-II and MOCell metaheuristics, and additionally, we have a single black dot representing the solution obtained by the VDA methods.

Figures 10, 11, 12, and 13 represent the presentation and comparison of the results generated by the SBSE and VDA algorithms for files 1, 2, 3, and 4, respectively.

In these charts, as mentioned earlier, each point generated by metaheuristics represents an ordered sequence of requirements. Therefore, this unique solution represents an ordered sequence of requirements organized in software releases that meet the DM requirements during VDA application phases. The black dot represents a unique solution generated by VDA methods (ZAPROS III-*i* and ORCLASS). The blue dots

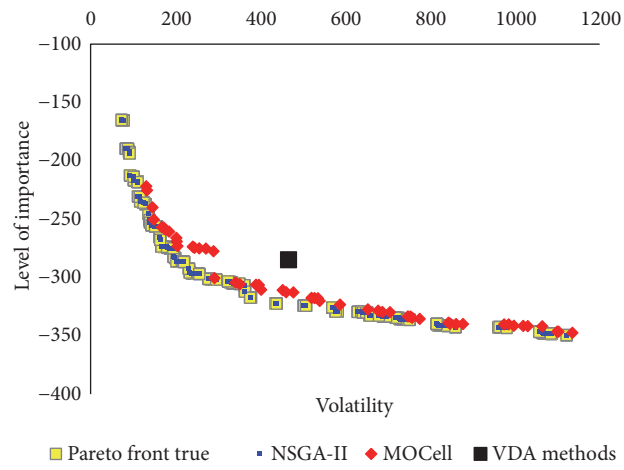

Figure 10: Results for file 1.

represent the results of the NSGA-II, the red dots represent the results of the MOCell metaheuristic, and the yellow dots represent the results of the real Pareto front that is the conjunction of all the nondominated points of all the solutions.

Figures 10–13 represent four file problems. These problems contain 20 requirements with their technical characteristics. On the other hand, there are five interested customers, each with their personal preferences to the requirements, assigning them preference scores. Finally, these customers

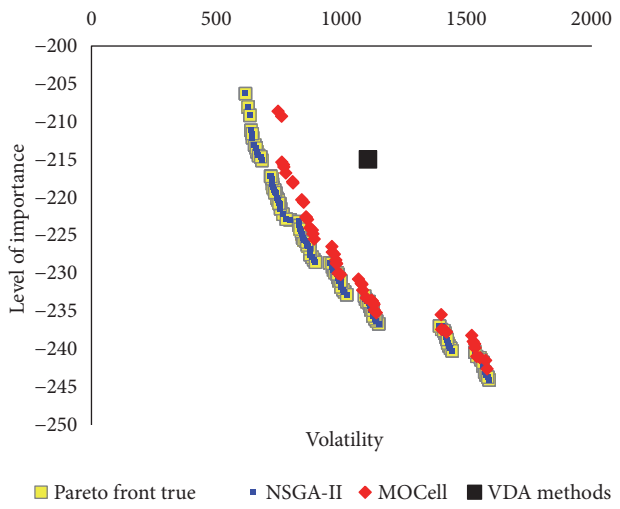

Figure 11: Results for file 2.

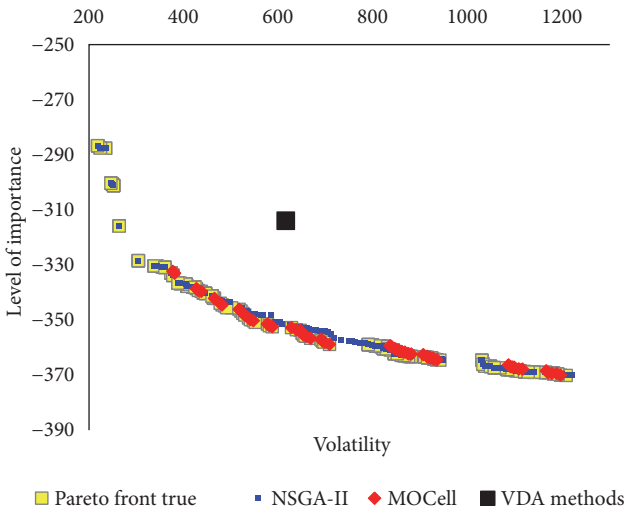

Figure 12: Results for file 3.

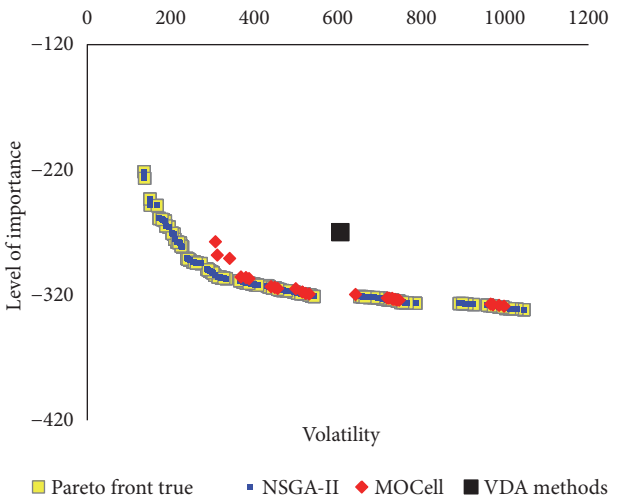

Figure 13: Results for file 4.

have a level of importance to the company. As already explained, this information is empirical. In addition to the generated internal data being different between these files, the total resource available for project execution and the level of technical precedence between requirements is also different among these files. Figures 10 and 12 show the results for files containing 70% of the available resources. However, Figure 10 expresses the results of a file with 10% technical precedence between requirements and Figure 12, 20%. Figures 12 and 13 represent problems with available resources of 80%. Figure 11 has 10% technical precedence between requirements and Figure 13, 20%.

In all cases, we can observe that the results generated by the VDA methods were very close to the real Pareto generated by the SBSE methods. These are undoubtedly good results when we consider that the results of metaheuristics are the baseline of our project, by the tradition of these metaheuristics in solving this type of problem. In Figures 10, 12, and 13 we have a result very close to the Pareto front true, and in Figure 11 we have a modest distance from the result of the VDA methods of this work, but within the expected in this research.

In a way, we can understand that, regardless of the level of difficulty faced by VDA methods, such as the level of technical precedence being higher or lower and/or the number of resources available to be higher or lower, the results were very similar to each other; they remained close to the front of Pareto. We would consider problematic for our methodology the results that were far behind the front of Pareto generated by metaheuristics or much ahead of it, surpassing it. In all the generated, tests we did not have these situations.

In order to have more singularity in the presentation of our results, we calculate the distance between the points generated by the VDA method about the actual Pareto front that is the front generated by all the nondominated solutions in the graph. We can check the methodology and the results of this calculation of the distances between points, with the application of Generational Distance (GD) [32], below.

9.1. Generational Distance (GD). The Generational Distance (GD) metric calculates between the front points of the current Pareto PF_{known}, in the case of VDA we have only one point, with the actual Pareto front (PF_{true}) [33]. For each solution contained in PF_{known}, calculates whether the shortest distance with the current Pareto front true PF_{true}, as follows:

$$d_i = \min_j \left(\left\| f_{true}^j - f_{known}^i \right\| \right)_2 \quad (5)$$

To set the distance, d_i is multiplied by e_i^{GER}. Through the application of $(d_i - f_j)/(\overline{f_j} - \underline{f}, j,)$ we can obtain normalization. We can define the metric as being:

$$NDWD_{GER} = \sum_{i=1}^{N} e_i^{GER} \left(\frac{d_i - f_{/j}}{\overline{f_j} - \underline{f_j}}. \right) \quad (6)$$

Table 7: The result of applying the metric GD.

File	Result in GD
File 1	0.1543
File 2	0.3585
File 3	0.3287
File 4	0.4283

A low value for this metric implies a higher approximation of results ahead of real Pareto meaning a good result. Therefore, the value 0 here means that the point generated is on the front of the real Pareto and therefore is part of it. In Table 7, we have the results of the points generated by VDA in the four files about the metaheuristic results.

The results in Table 7 show that problems with a more considerable budget constraint performed better. This conclusion comes from the fact that these results are closer to the real Pareto front than the other solutions. In some software development projects, a budget is initially required to implement all the requirements, but by the internal policy of the customer, in a given period it may have available a lower than estimated financial value and therefore requests that the crucial requirements are implemented first, leaving the implementation of the less essential ones to later.

File 4 had the worst solution among the four problems. However, we emphasize that this result in this research is of great value. Additionally, we know that metaheuristics have a long way to solve this type of problem. We emphasize the use of the GD metric in this research because it gives us a perception of the results obtained mathematically, calculating the relative approximation of our point in front of real Pareto.

After the presentation and analysis of the results to the DMs on the solutions generated by the VDA methods, we asked them to answer a small questionnaire by assigning a score to the generated solution, where 0 meant an unfortunate result and 100 a good result. Figure 14 shows that the DMs consulted were satisfied with the solutions generated. The results of the questionnaires reached 85 points of average score. Good scores to these results show us the potential of Verbal Decision Analysis methods in solving problems related to selection and prioritization of software requirements.

Overall, the DMs considered that 15% of the results achieved were not within what they expected. Half of this group claimed dissatisfaction about the total budget that is not used at the end, as shown in Table 6. We justify to them that the remaining amount is not enough to implement any of the requirements that were queued up. If it were possible, the methodology had put it in Release 5, the latter. The other half of the group of dissatisfied DMs did not agree with the ordering of requirements shown by our methodology. They believe in a different order of prioritization. Perhaps we will be able to consider in the future an extra burden for the requirement, and thus the DM may show a particular interest for it, although the requirement is not as good as the others towards efficiency in meeting objectives and constraints.

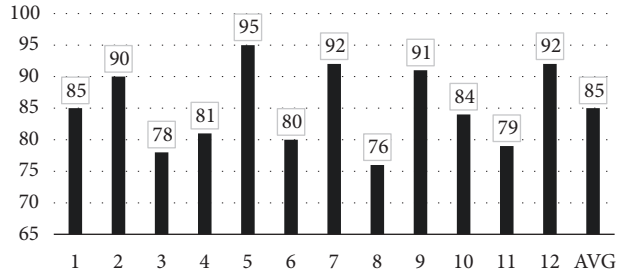

Figure 14: Evaluation of decision makers.

Figure 14 shows the scores of each DM as well as the average between the scores of all DMs.

Finally, to get an idea of the average time to solve these problems, we construct Table 8 that contains three forms of resolution. The first one was done manually only with the experience of the DM, the second one made with the VDA methods, which considers the DM's opinion and the third with the purely mathematical SBSE NSGA-II method. The time obtained by MOCell metaheuristics was quite similar to the time obtained by the NSGA-II metaheuristic.

We can see that file 2 has less time to reach solutions in the three analyzed methods. Also, this is because it is simpler mathematically and statistically to solve the problem in this file than those of the other problems. Unlike file 2, file 4 takes longer to arrive at a result.

It is understandable that the manual method requires more time and that the metaheuristics (SBSE), because they use only computational power, take less time. The results of this work are between these two methodologies (manual and SBSE), using human experience and, at the same time, computational power, but with a low response time for the decision maker.

As shown in Table 1, the four files have distinct configurations with variations in their constraints. These variations range from budget availability (70% to 80%) and the level of technical precedence between requirements (10% to 20%). These combined characteristics make the problem more challenging to solve because we increase the complexity when we increase the constraints. Additionally, each file has data that simulates a software development project scenario, and therefore, a different level of complexity generated each solution.

The results of Table 8 show that, even with an intentional distortion between problematic scenarios (budget and level of technical precedence), the results remained stable; that is, we did not have one file with results distorted concerning the others. Therefore, this demonstrates the stability of the results obtained by VDA methods, even under different scenarios and constraints.

10. Conclusions

This work shows that it is possible for Verbal Decision Analysis methods to make selection and ordering of requirements. In this way, we seek to generate problems that resemble those faced by DMs in real projects as well as meet the

Table 8: Comparison of time spent between three methods of resolution.

File	Manually	VDA methods	SBSE (NSGA-II)
File 1	75 min	3 min	0,5 min
File 2	65 min	2,5 min	0,5 min
File 3	73 min	2,4 min	0,6 min
File 4	78 min	3 min	0,7 min

technical capacity currently available for VDA methods. If we did not have goals, constraints, and a budget available for all requirements to be implemented, we would not need to make any requirements selection. However, in real projects, these strands are considered, and requirements selections are commonly used to more satisfactorily serve a customer at a cost that he can pay at that moment.

Barbosa [8–10] had already achieved excellent results with this methodology to solve the problem of requirements ranking. A set of 20 requirements were ranked using ZAPROS III-*i* methods. On the other hand, this leaves the process more time-consuming because there are more requirements for ZAPROS III-*i* to treat, and therefore, there are more inconsistencies to be addressed. Thus, the method shows the DM a more significant number of questions. The differential of this work concerning the others is that before using the ZAPROS III-*i* method to rank these requirements, we inserted the ORCLASS method that also composes the VDA field and is an effective classifier. Thus, the requirements, which are far from meeting the objectives and constraints, will not be classified for the next step, according to the methodology shown in Figure 6. Also, this led to a significant process time reduction. The number of requirements handled by ZAPROS III-*i* decreased and the number of questions as well.

Additionally, another gain in this work is the fact that we can hear the opinion of the DM during the classification and ranking process since their experience in applying their personal preferences adds value to the results generated. The black points shown in Figures 10, 11, 12, and 13 demonstrate the results that express the desire of DM, since it was their choices in the classification and ranking process that directly generated this result. For small problems, this is of great value. As shown in the mathematical formulation (Section 6), doing the job of selecting and ranking the requirements in releases manually is a combinatorial problem, which is extremely tiring and challenging for the human doing. It is precisely this point that Table 8 shows us where the human took an hour to do this task.

In Table 8, we realize that VDA results outweigh the results of a manually reached solution. As stated, we do not expect to outperform SBSE, but results equivalent to this kind of resolution are satisfactory to us.

The Search-Based Software Engineer has a vast field of research dedicated to using metaheuristics to solve problems of verification, allocation, selection, prioritization, and ordering of requirements. We know that automated processes help reduce task execution time [34, 35]. However, this work shows that we can have other equally effective alternatives to solve this type of problem. That way we have a new window in VDA that can handle this and other problems.

The approximation levels shown by the GD metric and the level of satisfaction of the DM at the end of this project demonstrate to us that our results reached satisfactory levels. Future research can further improve these indices.

Finally, the results obtained in this work, in both the computational area and the level of satisfaction of the DMs in the solutions obtained by the VDA, demonstrate the feasibility of using the methodology proposed here for this purpose. The DMs indicate that this research is valuable because the results presented were satisfactory to them since the requirements that the decision makers wanted to be implemented first were in the first releases, and the less essential requirements were in the latest releases. Therefore, the results achieved by this work provide elements that demonstrate its technical viability and attest its use by software developers.

The main contribution of this work is the elaboration of a methodology that, using Verbal Decision Analysis, can support the decision maker when selecting and prioritizing software requirements in releases. We know this can be done manually or by using tools available in SBSE, however, we seek to achieve satisfactory results using different methods. In this work, we were able to evolve our research to achieve better results by selecting more experienced DMs, using more current versions of metaheuristics, inserting the ORCLASS method in the work methodology to make an initial selection of requirements by classifying those "implementable" and internal enhancements to VDA algorithms.

11. Limitations and Future Work

Verbal Decision Analysis is an area that studies analysis and decision support based on verbal factors, different from existing quantitative methods. In other perspectives, this field of research favors software engineering in the search for solutions to various problems, but in engineering requirements, this is still challenging. Our work seeks to improve this scenario by searching in the VDA methodology, methods that solve this type of problem.

In this work there is a combination of two VDA methods to select and prioritize software requirements. In addition to this innovation, we also sought to extract from VDA its uniqueness, which is the possibility of listening, in a natural language, to the decision maker in this process.

However, as this is a different study methodology, we find some limitations that make research in this field even more

challenging. The number of requirements used is one such challenge. We know of this technical limitation for this field since we are seeking to solve a problem of one area through methods from another area.

We have seen that other VDA methods have recently been introduced [36] and can help us refine this research. Among these methods, we can highlight two, the NORCLASS classifier [37], which is a knowledge-based classification method that aims to classify alternatives (objects) into decision groups, which can not be ordered according to their typicality for a group. The second is the ARACES method [38], which was developed based on ZAPROS, proposing a flexible approach to deal with inconsistencies in the preferences of the decision maker. The method introduces the idea of using an array to represent the dominance relationships between criteria values, and if the decision maker provides an inconsistent input, the process receives the new array. These methods can provide the use of a substantial set of requirements.

Another limiting factor is that the number of requirements used in VDA is directly linked to the number of questions asked to the decision maker to solve a problem. In this way, the more requirements worked, the higher the number of questions. We observed this in the works, but in this work we were able to improve this and reduce this number of questions by 50% because of the insertion of the ORCLASS method that selected and removed requirements that did not fit the project premises, moving to the next step a smaller number and selected requirements. This insertion has reduced the number of questions, improved the results achieved, and less time to find a satisfactory solution.

For future work, our goal is to increase the number of requirements supported and decrease the number of questions asked by the VDA methods to the decision maker, and this is an essential factor. Therefore, we also hope to develop a framework using the methodology of this work to simplify the use of this technique.

These are natural limitations, but we are seeking better solutions as this research advances, and new VDA approaches are published.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

Plácido Rogério Pinheiro is grateful to the National Council for Scientific and Technological Development (CNPq) for the contribution received to develop this project. Paulo Alberto Melo Barbosa is grateful to the Federal Institute of Cerar and the prorector of research and postgraduation (PRPI) by the consideration received that supported the development of this project.

References

[1] G. Ruhe and M. O. Saliu, "The art and science of software release planning," *IEEE Software*, vol. 22, no. 6, pp. 47–53, 2005.

[2] I. Sommerville, *Software Engineering*, Pearson Addison-Wesley, São Paulo, Brazil, 7th edition, 2004.

[3] J. D. Sagrado and I. M. D. Aguila, "Ant Colony Optimization for Requirement selection in Incremental Software development," in *Proceedings of the 1st International Symposium on Search Based Software Engineering (SSBSE '09)*, pp. 67–76, Windsor: IEEE Computer Society, 2009.

[4] A. Bagnall, V. Rayward-Smith, and I. Whittley, "The next release problem," *Information and Software Technology*, vol. 43, no. 14, pp. 883–890, 2001.

[5] M. H. E. Paixao and J. T. Souza, "A robust optimization approach to the next release problem in the presence of uncertainties," *The Journal of Systems and Software*, vol. 103, pp. 281–295, 2015.

[6] T. d. Ferreira, A. A. Araújo, A. D. Basílio Neto, and J. T. de Souza, "Incorporating user preferences in ant colony optimization for the next release problem," *Applied Soft Computing*, vol. 49, pp. 1283–1296, 2016.

[7] M. M. Brasil, T. G. da Silva, F. G. de Freitas, J. T. de Souza, and M. I. Cortés, "A multiobjective optimization approach to the software release planning with undefined number of releases and interdependent requirements," in *Enterprise Information Systems*, vol. 102 of *Lecture Notes in Business Information Processing*, pp. 300–314, Springer Berlin Heidelberg, 2012.

[8] P. A. M. Barbosa, P. R. Pinheiro, F. R. de Vasconcelos Silveira, and M. Simão Filho, "Applying verbal analysis of decision to prioritize software requirement considering the stability of the requirement," in *Proceedings of the 6th Computer Science Online Conference (CSOC '17)*, vol. 575 of *Advances in Intelligent Systems and Computing*, pp. 416–426, Springer International Publishing, April 2017.

[9] P. A. Barbosa, P. R. Pinheiro, F. R. Silveira, and M. S. Filho, "Selection and prioritization of software requirements using the Verbal Decision Analysis paradigm," in *Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering (SEKE '17)*, pp. 213–217, 2017.

[10] P. A. M. Barbosa, P. R. Pinheiro, and F. R. de Vasconcelos Silveira, "Towards the verbal decision analysis paradigm for implementable prioritization of software requirements," *Algorithms*, vol. 11, no. 11, p. 176, 2018.

[11] J. Karlsson and K. Ryan, "Supporting the selection of software requirements," in *Proceedings of the 8th International Workshop on Software Specification and Design (IWSSD 96)*, pp. 146–149, 1996.

[12] A. G. Cordeir and A. L. P. Freitas, "Priorização de requisitos e avaliação da qualidade de software segundo a percepção dos usuários," *Ci. Inf., Brasília, DF*, vol. 40, no. 2, pp. 160–179, 2011.

[13] N. Nurmuliani, D. Zowghi, and S. Powell, "Analysis of requirements volatility during software development life cycle," in *Proceedings of the Australian Software Engineering Conference (ASWEC '04)*, pp. 28–37, April 2004.

[14] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Communications of the ACM*, vol. 31, no. 11, pp. 1268–1287.

[15] D. Greer and G. Ruhe, "Software release planning: an evolutionary and iterative approach," *Information and Software Technology*, vol. 46, no. 4, pp. 243–253, 2004.

[16] S. Vergilio, T. E. Colanzi, A. T. R. Pozo, and W. K. G. Assuncao, "Search-basedd software engineering: a review from the brazilian symposium on software engineering," in *Proceedings of the XXV Simpósio Brasileiro de Engenharia de Software (SBES '11)*, pp. 49–54, 2011.

[17] M. Harman and B. F. Jones, "Search-based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, 2001.

[18] A. J. Nebro, J. J. Durillo, M. Machín, C. A. Coello Coello, and B. Dorronsoro, "A study of the combination of variation operators in the NSGA-II algorithm," in *Proceedings of the Conference of the Spanish Association for Artificial Intelligence*, pp. 269–278, September 2013.

[19] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, "MOCell: a cellular genetic algorithm for multiobjective optimization," *International Journal of Intelligent Systems*, vol. 24, no. 7, pp. 726–746, 2009.

[20] I. Tamanini and P. R. Pinheiro, "Reducing incomparability in multicriteria decision analysis: an extension of the ZAPROS method," *Pesquisa Operacional*, vol. 31, no. 2, pp. 251–270, 2011.

[21] I. Tamanini, P. R. Pinheiro, T. C. S. Machado et al., "Hybrid approaches of verbal decision analysis in the selection of project management approaches," *Procedia Computer Science*, vol. 55, pp. 1183–1192, 2015.

[22] O. Larichev and H. M. Moshkovich, *Verbal Decision Analysis for Unstructured Problems*, Kluwer Academic Publishers, Boston, MS, USA, 1997.

[23] O. I. Larichev and H. M. Moshkovich, *Verbal Decision Analysis for Unstructured Problems*, Kluwer Academic Publishers, 1997.

[24] I. Ashikhmin and E. Furems, "UniComBOS—intelligent decision support system for multi-criteria comparison and choice," *Journal of Multi-Criteria Decision Analysis*, vol. 13, no. 2-3, pp. 147–157, 2005.

[25] R. P. Plácido, C. S. M. Thais, and T. Isabelle, "Handing the classification methodology ORCLASS by tool orclassweb," *Lecture Notes in Computer Science*, vol. 8588, 2015.

[26] J. Figueira, S. Greco, and M. Ehrgott, *Multiple Criteria Decision Analysis: State of the Art Surveys*, Springer Verlag, London, UK, 2005.

[27] M. Doumpos and C. Zopounidis, *Multicriteria Decision Aid Classification Methods*, Springer Verlag, London, UK, 2002.

[28] I. Tamanini and P. R. Pinheiro, "Challenging the incomparability problem: an approach methodology based on ZAPROS," in *Modelling, Computation and Optimization in Information Systems and Management Sciences*, vol. 14 of *Communications in Computer and Information Science*, pp. 338–347, Springer Berlin Heidelberg, 2008.

[29] I. Tamanini, P. R. Pinheiro, and A. L. Carvalho, "Aranaú software: a new tool of the verbal decision analysis," Technical Report, University of Fortaleza, 2007.

[30] P. A. M. Barbosa, *An Optimal-Based to thePrioritizationn of Software Requirements, Considering the Stability of the Requirement [Master Thesis]*, Academic Master in Computer Science - Ceará State University, Brazil, 2013.

[31] J. J. Durilo and A. J. Nebro, "JMetal: a java framework for developing multi-objective optimization metaheuristics," Tech-Report ITI-2006-10, University of Málaga, Spain, 2006.

[32] D. A. V. Veldhuizen and G. B. Lamont, "Evolutionary Computation and Convergence to a Pareto Front," 2019, https://pdfs.semanticscholar.org/f329/eb18a4549daa83fae28043d19b83fe8356fa.pdf.

[33] C. A. C. Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, Kluwer Academics Publishers, Boston, MS, USA, 2007.

[34] M. Simão Filho, P. R. Pinheiro, and A. B. Albuquerque, "Task assignment to distributed teams aided by a hybrid methodology of verbal decision analysis," *IET Software*, vol. 11, no. 5, pp. 245–255, 2017.

[35] M. S. Filho, P. R. Pinheiro, A. B. Albuquerque, and J. J. Rodrigues, "Task allocation in distributed software development: a systematic literature review," *Complexity*, vol. 2018, Article ID 6071718, 13 pages, 2018.

[36] P. R. Pinheiro, I. Tamanini, M. C. Dantas Pinheiro, and V. H. de Albuquerque, "Evaluation of the Alzheimer's disease clinical stages under the optics of hybrid approaches in Verbal Decision Analysis," *Telematics and Informatics*, vol. 35, no. 4, pp. 776–789, 2018.

[37] E. M. Furems, "Domain structuring for knowledge-based multiattribute classification (a verbal decision analysis approach)," *TOP*, vol. 19, no. 2, pp. 402–420, 2011.

[38] D. P. Oleynikov, L. N. Butenko, H. M. Moshkovich, and A. Mechitov, "ARACE – a new method for verbal decision analysis," *Int. J. Inform. Technol. Decision Making*, vol. 14, no. 1, pp. 115–140, 2014.