A Robust Optimization Approach to the Next Release Problem in the Presence of Uncertainties

Matheus Paixao[a], Jerffeson Souza[a]

[a]*State University of Ceará, 1700 Avenida Paranjana, Fortaleza, Brazil*

Abstract

The next release problem is a significant task in the iterative and incremental software development model, involving the selection of a set of requirements to be included in the next software release. Given the dynamic environment in which modern software development occurs, the uncertainties related to the input variables of this problem should be taken into account. In this context, this paper presents a formulation to the next release problem considering the robust optimization framework, which enables the production of robust solutions. In order to measure the "price of robustness", which is the loss in solution quality due to robustness, a large empirical evaluation was executed over synthetical and real-world instances. Several next release planning situations were considered, including different number of requirements, estimating skills and interdependencies between requirements. All empirical results are consistent to show that the penalization with regard to solution quality is relatively small. In addition, the proposed model's behavior is statistically the same for all considered instances, which qualifies it to be applied even in large-scale real-world software projects.

Keywords: Next Release Problem, Robust Optimization, Search Based Software Engineering

1. Introduction

In an iterative and incremental software development process, a stable and executable version of the product delivered to the clients is called re-

Email addresses: matheus.paixao@uece.br (Matheus Paixao), jerffeson.souza@uece.br (Jerffeson Souza)

lease. Despite bringing many benefits, this development model embodies more complexity to the project management. The choice of which requirements will be added to the next release has to be made in a way to keep the clients interested and involved in the project. The effort cost to develop the requirements selected to the next release must respect a predefined budget. In addition, the requirements usually present interdependencies between each other. Such constraints must also be respected. In the Search Based Software Engineering field, this problem has become known as the Next Release Problem (NRP).

The NRP was first modeled as an optimization problem by Bagnall et al. (2001). In this model, each client has an importance to the organization and requests a subset of requirements to be implemented in the next release. The goal is to select a subset of clients to be satisfied so that the sum of their importance is maximized. A client is considered satisfied when all requirements that he/she has requested are included in the release. A variation of this model was proposed by Van den Akker et al. (2005). In this model, instead of the client, the requirements have importance values, which estimates their revenue for the company. The objective here is to select a subset of requirements such that the total revenue is maximal.

In order to employ an optimization technique to solve the Next Release Problem, it is necessary to obtain the values of importance and cost for each requirement. As reported in the empirical study by Cao and Ramesh (2008), companies using an iterative and incremental software development model usually employ an iterative Requirements Engineering process, rather than a formal one. Among most of the seven different iterative RE techniques reported, the clients indication of the requirements' business values is a commonplace. These iterative RE practices also usually use some kind of expertise-based technique for requirements effort cost estimation. The experts, usually the development team, provide estimates based on personal experiences along with knowledge obtained in past projects (Boehm et al., 2000).

Both requirements' importance and cost estimates can be significantly hard to make due to the dynamic environment in which software development occurs. Harker et al. (1993) classifies software requirements in six types. Among those types, five are considered as *changing* requirements and only one as *stable* requirements, which stresses the evolving nature of software requirements. In this context, requirements' importance and cost are among those features that can change during the release development. Indeed, the

high level of uncertainty related to the variables of the NRP generates a fairly complicated context, as pointed out by Zhang et al. (2008):

> "Software engineering problems are typically 'messy' problems in which the available information is often incomplete, sometimes vague and almost always subject to a high degree of change (including unforeseen change). Requirements change frequently, and small changes in the initial stages often lead to large changes to the solutions, affecting the solution complexity and making the results of these initial stages potentially fragile."

Regarding the requirements' cost estimates, the sensitivity analysis performed by Harman et al. (2009) shows the impact that an inaccurate cost estimation can have in the final NRP solution. It is stated that in most of the cases, the more expensive the requirement is, the greater impact it will have on the result when wrongly estimated. Also, similarly, the higher the level of error, the bigger is the impact.

As can be seen, some of the approaches used to estimate the requirements' importance and cost in incremental software development can introduce some degree of uncertainty to the Next Release Problem variables. Such uncertainties, if not treated carefully, can have a big impact in the subset of requirements selected to the next release. Therefore, it seems reasonable that the uncertainties related to requirements' importance and cost should be considered when solving the NRP through an optimization technique.

Admitting that some problem aspects are uncertain, the robust optimization is an operational research framework that identifies and quantifies uncertainties in optimization problems (Beyer and Sendhoff, 2007). Taking the requirements' cost uncertainty as an example, both the *robustness level* and the respective *uncertainty variation* have to be defined. Among the set of requirements' cost estimates, the *robustness level* is related to how many cost estimates will be considered as wrong estimates. The cost *uncertainty variation* represents how much each requirement cost could vary from the original estimate. Based on these assumptions, the robust optimization framework builds up models which seek robust solutions, i.e., even with noisy input data, it produces good quality solutions while still fulfilling all constraints. More details about the robust optimization framework are given in Section 3.

Accordingly, the robust optimization framework can be used to model the NRP considering the uncertainties present in this problem. This robust

model could handle the requirements' importance and effort cost uncertainties, allowing for the production of robust NRP solutions.

In order to ensure robustness to a certain optimization problem, some loss in solution quality is inevitable, which is highly connected to both the *robustness level* and *uncertainty variation*, and needs to be contemplated. This measure of quality loss of the robust solution when compared to the non-robust solution has been called in the robust optimization literature as the "price of robustness" (Bertsimas and Sim, 2004).

Actually, the robust NRP model presented in this paper has already been proposed by the authors of this paper in another work (Paixao and Souza, 2013). This paper is intended to be an extension of the previous one, improving both the robust model explanation and evaluation. Therefore, motivated by this context, this paper aims at answering the following research questions:

- RQ_1: How the Next Release Problem can be modeled as an optimization problem considering the uncertainties related to its input variables in order to allow for the production of robust solutions?
- RQ_2: What is the "price of robustness" for the presented Next Release Problem model? In other words, how much would be lost with regard to solution quality in order to gain robustness?

The RQ_2 was broken in the following sub-questions in order to facilitate the "price of robustness" analysis:

- $RQ_{2.1}$: What is the "price of robustness" for the presented NRP model when using different robustness levels?
- $RQ_{2.2}$: What is the "price of robustness" for the presented NRP model when using different variations in the requirements' costs?
- $RQ_{2.3}$: What is the "price of robustness" for the presented NRP model when there are interdependencies between requirements?
- $RQ_{2.4}$: What is the "price of robustness" for the presented NRP model when it is applied to real-world NRP instances?

The remaining of this paper is organized as follows: Section 2 presents some related work to this paper and Section 3 gives more details about the

robust optimization framework. Section 4 presents the robust NRP formulation. Section 5 exhibits and examines the empirical study designed to evaluate the presented formulation. Finally, Section 6 concludes the paper and points out some future research directions.

2. Related Work

As stated earlier, the NRP was first modeled as a search problem by Bagnall et al. (2001) and a variation of this first model was proposed by Van den Akker et al. (2005). After that, many works have addressed this problem with different focuses, like proposals to apply the model to real-world instances (Baker et al., 2006), to solve large NRP instances (Jiang et al., 2010)(Xuan et al., 2012), to consider interdependencies between requirements (Del Sagrado et al., 2011) and to apply different metaheuristic approaches like Ant Colony Optimization (Del Sagrado et al., 2010). All approaches mentioned above use a single objective formulation and can be thought as a decision making tool. The software engineer would insert the requirements data and the tool would return a set of requirements to be implemented in the next release. Search based algorithms usually perform at the edge of the constraints, aiming to find the best solutions. A wrong estimation of the requirements' costs can make the final solution infeasible due to budget constraints.

In order to provide a decision support tool, allowing the software engineer to make decisions based in different preferences and priorities, a multiobjective version of the NRP was proposed by Zhang et al. (2007). Here, the release total cost is treated as an objective instead of a constraint like in the single objective version. A multiobjective evolutionary algorithm can be employed to generate a pareto front of solutions, i.e., releases that are equally good between themselves. The software engineer can choose a solution from this set of solutions and can also make some kind of 'what if analysis' like: "how much would be lost in total importance if the release total cost was 20% higher than estimated"?. However, this approach does not consider any importance uncertainty and these cost uncertainties analysis can only be done based on the total release cost.

The changing nature of the requirements' importance values during the software lifecycle has been addressed by Zhang et al. (2010). In this work, each requirement receives an importance value called "today value", which represents the company needs at the release planning phase. It is assumed

that this value, in some moment after the release development, would change to a certain "future value", representing a possible future importance value for the requirement. The approach, therefore, seeks to balance the company's today and future needs by considering these two requirement importance values, along with the release total cost, via a multiobjective formulation. Considering a long time usage of a software system, it is unlikely that the requirements would have just one possible future importance value, tough. These values can change constantly during the software life and consider all these values via a multiobjective approach is not scalable.

In regard to another requirements related problems modeled as optimization problems and tainted with uncertainties, the optimal release time can be mentioned. The NRP and the aforementioned problem have different meanings for the 'release' term. The first one considers release as the next version of the software to be delivered to the client and tries to select the requirements to be implemented in this next version. The last one considers 'release time' as the time the software system, after developed, spends being tested before being delivered to the client, in order to guarantee a certain degree of reliability. Some papers have tackled this problem using Software Reliability Models (SRM) to predict the system's reliability and minimize the time and cost spended in testing (Okumoto and Goel, 1980)(Yamada et al., 1984)(Huang and Lyu, 2005). Such SRMs are based on some testing estimates, which can contain uncertainties. Therefore, there exists a risk that the reliability requirement cannot be guaranteed due to the parameter uncertainties in the SRM. The paper by Peng et al. (2013) proposes a treatment of the SRMs input parameters in order to generate a release time that still guarantees reliability even when such parameters are uncertain.

Regarding uncertainty handling in another software engineering problems, the work by Antoniol et al. (2004) proposes a search based approach to project management in the presence of uncertainties. The proposal consists in a tandem genetic algorithm used to find both the best sequence for work packages and the best allocation of staff to project teams. The uncertainties treatment is actually a sensitivity analysis of what would happen if some problem aspects were wrong estimated, for example. It does not propose any approach to deal with these situations, though.

Dynamic Adaptive Systems (DAS) are highly subject to environmental uncertainties and changing conditions. Such systems must be able to adapt their behavior in face of a series of possible unexpected situations. The paper by Ramirez et al. (2012) proposes a search based approach to automatically

define the RELAXation of the adaptation rules of a DAS. Adaptation rules RELAXing consists in defining fuzzy rules that specify the extent to which a goal can become temporarily unsatisfied and yet deliver acceptable behavior. The approach has generated rules that are able to reduce the number of adaptations and goal models of equal or greater quality than those manually created by a requirements engineer.

It is noteworthy that, from the authors knowledge and from a literature review, none of the previous related work have handled the Next Release Problem uncertainties through the robust optimization framework. Such framework is presented with more details in the following section.

3. The Robust Optimization Framework

The robust optimization is an operational research framework that handles uncertainties in generic optimization problems and generates robust solutions for these problems (Beyer and Sendhoff, 2007). A solution is considered robust if, even with noisy input data, it still fulfills all problem's constraints.

Despite previous works dating from the 80s (Taguchi, 1986), robust optimization has gained more visibility after the works by Mulvey et al. (1995) and Bai et al. (1997). It has been successfully applied to several engineering disciplines including, but not limited to, production (Leung et al., 2007), aeronautical (Du et al., 2000), electronic (Malcolm and Zenios, 1994), mechanical (Li and Azarm, 2008), chemical (Wang and Rong, 2009) and metallurgical engineering (Dulikravich and Egorov-Yegorov, 2005).

It basically consists in three steps: i) identify and quantify the problem's uncertainties; ii) build up a robust model which generates robust solutions; iii) solve the robust model using some optimization technique;.

Regarding the first step, uncertainties in a generic optimization problem fit into one of the following types (Beyer and Sendhoff, 2007):

(A) **Changing environmental and operating conditions**: the problem's model is usually developed from a pre-determined operating environment configuration. However, a real environment changes its features constantly.

(B) **Decision variables imprecision**: when the decision variables are subject to some kind of perturbation during the optimization process.

(C) **Results uncertainties**: related to the use of simulations. When employing the solution computed by the optimization algorithm in a real operating environment, the real results may be different from the simulated ones.

(D) **Constraints feasibility uncertainties**: constraints defined from estimates and simulations can be uncertain. Moreover, when considering the uncertainties (A) and/or (B), the solution must continue to fulfill all problem's constraints.

The uncertainties types above can also be called *fralty points* (Roy, 2010), which represent the source of the uncertainty. After identifying the uncertainties from each fralty point, one must quantify them. The uncertainties types presented above (A-D), can be quantified in a discrete or continuous way (Aissi et al., 2009), following at least one of the strategies next (Beyer and Sendhoff, 2007):

(1) **Deterministic**: defines especific domains in which the uncertainties can vary. Usually represented as a finite set.

(2) **Probabilistic**: defines probability measures by which a certain event may occur.

(3) **Possibilistic**: uses fuzzy logic to deal with subjective uncertainties, indicating the occurrence possibility of certain event.

As one can see, an uncertainty can be quantified using different strategies. The selection of a specific set of strategies to quantify each of the problem's uncertainties characterizes what is called in the literature as a *version* of the robust model (Roy, 2008).

A robust optimization model is similar to a regular optimization model, being composed by a set of objective functions and a set of constraints. Differently, a robust model has to be designed considering the problem's uncertainties, in such a way that it became able to generate robust solutions. From that point, any optimization algorithm can be used to solve the robust model, from mathematical programming to metaheuristics and evolutionary algorithms.

The Robust Next Release Problem model used in this paper is presented in the next section.

4. A Robust Next Release Problem Formulation

The following robust NRP model was first proposed by Paixao and Souza (2013). This section explains this model with more details and gives a practical example of its usage.

Consider N the number of requirements. Given a set of requirements $R = \{r_1, r_2, \ldots, r_N\}$, the requirement r_i importance value and effort cost are represented by v_i and c_i, respectively. A basic Next Release Problem formulation is presented next:

$$\text{maximize} \sum_{i=1}^{N} v_i x_i \tag{1}$$

$$\text{subject to} \sum_{i=1}^{N} c_i x_i \leq b \tag{2}$$

where b is the release budget. It maximizes the release overall importance while respects the budget constraint. The decision variable is represented as a vector $X = \{x_1, x_2, \ldots, x_N\}$, where $x_i = 1$ indicates that requirement r_i is included in the next release and $x_i = 0$ otherwise.

As pointed out by Harker et al. (1993), the occurrence of certain events can change the requirements' importance values during the release development. The requirements' importance uncertainty can be therefore, classified as a **Changing environmental and operating conditions** (type A) uncertainty. Imagine a Commercial off-the-shelf (COTS) software development company, for example. A potentially innovative functionality would be possibly considered a highly valuable requirement and would be developed as soon as possible. But, in the case where a competitor company releases its software with the same or similar functionality before the first company does, the importance of this requirement for the first company will decrease. In this situation, the first company would have wasted resources in developing this requirement with such priority. As one can see, the range a requirement importance can vary is discrete and depends on the set of events that actually have some influence in the requirement importance.

Thus, the requirements' importance value uncertainty seems adequate to be quantified using the robust optimization concept of scenarios (Yu, 1996). A scenario can be defined as a set of values which represent different contexts due to the occurrence of certain events, like in the given example.

Thus, based on the set of events that have some influence in the requirements' importance values, the requirements engineer can formally define a set of scenarios $S = \{s_1, s_2, \ldots, s_M\}$, where each scenario is represented by $s_j \subset S | s_j = \{v_1^j, v_2^j, \ldots, v_N^j\}$, with v_i^j expressing the importance of requirement r_i in scenario j. In order to consider all possible scenarios of the requirements' importance values, one must define the occurrence probability of each scenario. Thus, for each scenario j, it is defined an occurrence probability p_j, with $\sum_{j=1}^{M} p_j = 1$. The requirement's importance v_i in the robust model is then defined as:

$$v_i = \sum_{j=1}^{M} v_i^j p_j \tag{3}$$

The robust requirement's importance formulation above considers all possible scenarios, weighting each scenario importance value by its occurrence probability. In a situation where the occurrence probabilities are unlikely to foresee, one can consider a equal probability for all scenarios. The above requeriment importance can be looked at as a generalization of the importance in the basic NRP formulation. In fact, by considering a single scenario j, with a probability $p_j = 1$, the requirement's importance will be the same as the one in Equation 1.

The uncertainty related to the cost of the requirements is intrinsically distinct. When considering the effort cost uncertainties, the model has to ensure that the solution will still respect the budget constraint. Such uncertainty can be identified as a **Constraint feasibility** (type D) uncertainty. Regarding the quantification, it seems unreasonable to expect one to raise a set of scenarios based on certain events, since those costs usually vary independently and this change may not be discrete. Since the effort cost uncertainties are continuous random variables, they are likely to be quantified using variance intervals (Yang et al., 2008).

Thus, the uncertainty related to the requirements' effort costs will be quantified as follows. Let c_i be the estimated cost of requirement r_i. It is defined a value $\hat{c}_i$, which indicates the maximum expected cost variation. The requirement cost at the end of the release development is represented by $\bar{c}_i$ and is a function of its own estimate and expected variation, so that $c_i - \hat{c}_i \leq \bar{c}_i \leq c_i + \hat{c}_i$. In other words, the real effort cost of a requirement in the release planning phase is unknown, but it will necessarily be within the

range defined by its own estimate and expected variation.

Therefore, a possible robust formulation for the release total cost is as follows:

$$\sum_{i=1}^{N} c_i x_i + \sum_{i=1}^{N} \hat{c}_i x_i \tag{4}$$

In the above case, besides the sum of all requirements' costs selected to the next release, the total release cost will also consider the sum of all respective cost variations $\hat{c}_i$. This approach will guarantee that, even in the worst case, when all selected requirements will cost their upper bounds (given by $c_i + \hat{c}_i$), the release budget will be satisfied. This, clearly, represents a very conservative approach, since it assumes that all cost estimates will be missed by the maximum amount.

In real software development projects, however, different development teams have divergent estimating skills, usually related to the team's experience. These skills can be measured through a historical analysis among previous projects. For a certain development team, one can determine, in average, how many requirements cost estimates were actually precise. To consider this assumption, in order to generate a more realistic model, it is defined a control parameter Γ (Bertsimas and Sim, 2004), which indicates the expected level of failure in the cost estimations. Thus, in a situation where the team's estimates are historically 30% incorrect, in a project with 50 requirements, for example, the control parameter would be $\Gamma = 15$. It indicates that there is an expectation that 15 requirements will have real effort costs different from those that were originally predicted.

Using this new control parameter, the release total cost in the robust NRP model presented in this paper is computed as:

$$\sum_{i=1}^{N} c_i x_i + max_{W \subseteq R, |W| \leq \Gamma} \sum_{i \in W} \hat{c}_i x_i \tag{5}$$

The release total cost is composed by the sum of the cost estimates c_i and a second factor, which was added to guarantee a certain robustness level controlled by Γ, as explained next. Considering that there is an expectation that Γ requirements will have costs that were wrongfully predicted, the formulation will seek a subset $W \subseteq R$ with cardinality $|W| \leq \Gamma$, where the sum

of cost variations $\hat{c}_i$ is maximum. In other words, since there is no way to know in advance which requirements may have erroneous cost estimates, the model guarantees that, even if the development team misses the costs of the requirements with highest variations, the solution will still be valid.

Once again, it is straightforward to reach the basic NRP model or the conservative approach. Using $\Gamma = 0$, it is assumed that the team won't miss a single cost estimate. In this case, the total release cost in Equation 5 will be the same as described in Equation 2, the classic NRP model. In addition, all cost variations will be taken into account when $\Gamma = N$, which carries the formulation back to the conservative approach materialized by Equation 4. Finally, it is noteworthy that it is also possible to return to the basic formulation by setting all cost variations $\hat{c}_i$ to 0.

Therefore, the robust Next Release Problem formulation used in this paper is formally described as:

$$\text{maximize} \sum_{i=1}^{N} \sum_{j=1}^{M} v_i^j p_j x_i$$

$$\text{subject to} \sum_{i=1}^{N} c_i x_i + max_{W \subseteq R, |W| \leq \Gamma} \sum_{i \in W} \hat{c}_i x_i \leq b$$

where, $x_i \in \{0, 1\}$

R is the set of requirements

N is the number of requirements

M is the number of scenarios

v_i^j is the importance of requirement r_i in scenario j

p_j is the scenario j occurrence probability

c_i is the cost estimate of requirement r_i

$\hat{c}_i$ is the expected cost variation of r_i

Γ is the robustness control parameter

b is the release budget

The behavior of the robust model presented above is outlined in the fol-

lowing example.

Consider a NRP instance with six requirements $R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$. The customer(s) indicated two possible scenarios, s_1 and s_2, for the requirements importance values. They are represented by $s_1 = \{4, 1, 9, 3, 8, 5\}$ and $s_2 = \{6, 5, 8, 2, 3, 5\}$. The occurrence probability of each scenario is $p_1 = 0.7$ and $p_2 = 0.3$. The importance value of each requirement is then calculated by Equation 3, which generates the set of requirements importance values $V = \{4.6, 2.2, 8.7, 2.7, 6.5, 5\}$. Because the first scenario is more likely to occur than the second, the resulting values of V are closer to the s_1 values than to the s_2 values. By using the requirements' importance values V, the model can maximize the release total importance considering all possible requirements importance values.

Consider that the development team defined the effort cost estimate of each requirements as $C = \{4, 5, 5, 4, 6, 3\}$. The expected cost variation was configured to be half the requirement cost and it is represented by $\hat{C} = \{2, 2.5, 2.5, 2, 3, 1.5\}$. The release budget was set to $b = 22$. Considering the robustness level $\Gamma = 0$, the model falls back to the classic NRP and the optimum solution is the subset $\{r_1, r_3, r_4, r_5, r_6\}$, which consists in a total importance of 27.5 (the sum of the importance values of requirements r_1, r_3, r_4, r_5 and r_6). As the robustness level is $\Gamma = 0$, none of the cost variations are considered and the total effort cost is 22 (the sum of the estimate cost of requirements r_1, r_3, r_4, r_5 and r_6). As stated earlier, the optimum solutions are usually found at the edge of the constraints. When considering the cost uncertainties, such solutions may be actually infeasible.

When the robustness level is setted to $\Gamma = N$, for example, it is reached the worst case, when all requirements estimates are considered wrong. In such situation, all cost variations would be considered and all requirements would cost their upper bounds $(c_i + \hat{c}_i)$. In this new environment, the optimum solution would be the subset $\{r_3, r_5, r_6\}$, which presents a release importance of 20.2 and a effort cost of 21. Because of the cost uncertainties, the requirements r_1 and r_4 had to be removed from the previous optimum solution. Despite being robust, this solution is very conservative because it is unlikely that the development team misses the estimates of all requirements' costs.

The presented NRP robust model handles this conservatism by allowing a possible variation of the robustness control parameter Γ. For example, if the team's effort cost estimates are historically 40% of the number of requirements wrong, the robustness level would be $\Gamma = 2$, indicating that

two requirements are expected to have effort costs different from those originally estimated. In this situation, the optimum solution would be the subset $\{r_1, r_3, r_4, r_6\}$, which presents a release value of 21 and a total cost of 20.5. Since it is impossible to know in advance which of the two requirements are wrongfully estimated, the model considers the two bigger cost variations, i.e., $\hat{c_1}$ and $\hat{c_3}$. This way, during the release development, if any two of the four requirements selected to the release cost their upper bounds, the release would still respect the budget constraint. Therefore, an intermediate robustness level allowed the addition of one more requirement, increasing the release total importance when compared to the worst case situation ($\Gamma = N$).

The presented model, along with the robustness parameter Γ, can handle the effort cost uncertainties by allowing the requirements engineer to consider such uncertainties in different levels. The classic NRP model can be viewed as an optimistic approach, which generates the best possible next release planning. However, such solution is subject to infeasibility due to the uncertainties. The worst case scenario can also be achieved by the robust model, but it implies in a considerable loss in solution quality. An intermediate robustness level can, at the same time, guarantee a robust solution and improve the solution quality when compared to the conservative approach.

Hence, the next release formulation proposed by Paixao and Souza (2013) and presented in more details in this paper, is a robust optimization model which considers the uncertainties related to the input variables of this relevant problem which, therefore, answers the research question RQ_1.

5. Empirical Study

The research question RQ_2 consists in evaluating the "price of robustness" of the NRP robust model under different next release planning situations. The "price of robustness" represents how much is lost in solution quality when a robust solution to a certain instance is compared to the non-robust solution of the same instance. The quality of a solution is represented by its fitness value. This loss in solution quality is measured through a 'reduction factor' (Bertsimas and Sim, 2004), which indicates the percentage of loss in fitness value due to robustness.

Consider α_k as the average fitness value found for some instance when the robustness level is set to $\Gamma = k \times N$, where N is the number of requirements and k represents the percentage of requirements which are wrongly estimated. The 'reduction factor' is calculated by comparing α_k to the fitness value found

by the non-robust NRP model, i.e., when $\Gamma = 0$. Therefore, the 'reduction factor' δ_k can be calculated as follows:

$$\delta_k = 100 \times (1 - \frac{\alpha_k}{\alpha_0}) \tag{6}$$

Consider the same example given in the previous section. The fitness value of the classic NRP model ($\Gamma = 0$) is $\alpha_0 = 27.5$. Considering a 40% robustness level, $\Gamma = 0.4N$, the fitness value is $\alpha_{0.4} = 21$. Using Equation 6, a reduction factor $\delta_{0.4} = 23.63$ is found. This value indicates that, in order to ensure 40% of robustness to the solution, one will lose 23.63% in solution quality. Similarly, for $\Gamma = N$, a reduction factor $\delta_1 = 26.64$ is achieved, representing a loss of 26.64% in fitness value.

Subsection 5.1 depicts the settings of the empirical evaluation. It shows the instances set configuration and presents the search techniques used in the evaluation. Subsection 5.2 presents the results of the empirical evaluation, aiming at answer the research question RQ_2, where each of its subquestions are treated separetly.

5.1. Empirical Evaluation Settings

The instances and search techniques settings are presented apart from each other.

5.1.1. Instances Configuration

The instances set is composed by both synthetical and real-world NRP instances. The synthetical instances were randomly generated following an uniform distribution and designed to present distinct next release planning situations, including different number of requirements, different cost variations strategies and different interdependencies between requirements.

The synthetical instances were generated with a different number of requirements, represented by $\{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$, respectively. Each instance has $2, 3$ or 4 scenarios. The number of scenarios in the instance was randomly chosen. The occurrence probability of each scenario was also defined at random. The requirements' importance values v_i^j can assume an integer between 1 and 10. The effort cost estimate c_i also varies from 1 to 10. Both the requirements' cost and importance value in each scenario were randomly defined.

Different cost variations strategies were considered. The cost variation $\hat{c_i}$ was set to one of the following values $\{10\%, 20\%, 30\%, 40\%, 50\%\}$. A cost variation of 10% means that each cost variation $\hat{c_i}$ is considered to be 10% of the respective requirement estimated cost c_i. Besides varying the percentage from 10% to 50% of the original cost, a random approach was also introduced, where each requirement cost variation $\hat{c_i}$ is independently generated as a random number between 0 and 50% of c_i.

Requirements usually present different relationships between each other, which are called in the literature as interdependencies (Carlshamre et al., 2001). These interactions can be divided into two groups: *functional* interdependencies and *value* interdependencies (Del Sagrado et al., 2011). The first ones are related to strong functional interactions, usually translated into problem's constraints, like:

- *Precedence*: $r_i \Rightarrow r_j$. A requirement r_i cannot be selected to the next release if a requirement r_j has not been implemented yet.
- *Coupling*: $r_i \odot r_j$. Requirements r_i and r_j must be implemented in the same release
- *Exclusion*: $r_i \oplus r_j$. Requirements r_i and r_j cannot be implemented in the same release

Functional interdependencies present a considerable impact in the search space and must be taken into account when selecting requirements to the next release. However, *value* interdependencies can be ignored because they only cause variations in the characteristics of other requirements, without affecting the search space (Del Sagrado et al., 2011).

The interdependencies density between requirements can assume one of the following values $\{0, 10\%, 20\%\}$. The interdependency density indicates the percentage of requirements which will have interdependencies. Two requirements are randomly chosen and the type of interdependency they will have is also chosen at random. This process is repeated until the interdependency density is reached.

All possible next release planning aspects described above (number of requirements, cost variation strategy and interdependency density) were combined, resulting in a set with 180 synthetical instances. The synthetical instances names are in the format I_S_R_C_I, where R represents the number of requirements, C indicates the cost variation strategy and I denotes

the interdependency density. The instance I_S_250_20_10, for example, is a synthetical one, has 250 requirements, $\hat{c}_i$ is set to be 20% of c_i and presents an interdependency density of 10%.

The real-world instances were adapted from Xuan et al. (2012). In that paper, the instances to the NRP were extracted from bug repositories of two big open source projects, Eclipse (a java integrated development environment) (Eclipse, 2014) and Mozilla (a set of web applications) (Mozilla, 2014).

A bug repository is a forum where users (end users, developers, testers, etc) can report bugs related to the project. Each bug is then considered as a requirement. In this forum, one bug report may be commented by many users. Each requirement importance value is then calculated as the number of users that commented on that particular bug report, which represents the only possible scenario. In addition, the bug severity is mapped to the requirement cost estimate. Both the requirement's importance and cost are normalized to fall into the 1 to 10 interval.

Several instances were extracted from each bug repository. They are composed only by the most important requirements. The number of requirements of the real-world instances are $\{20, 50, 100, 150, 200\}$, respectively. Regarding the cost variation, the same strategies used in the synthetical instances were also applied to the real ones. Since bugs are usually independent from each other, there are no interdependencies in the real instances.

Therefore, the combination of the possible aspects of the real instances, such as bug repositories, number of requirements and cost variation strategies, resulted in 60 real-world NRP instances. The real instances names are in the format I_Re_P_R_C, where P represents the project (E for Eclipse and M for Mozilla), R is the number of requirements and C is the cost variation strategy. The instance I_Re_E_150_50, for example, was generated from the eclipse bug repository, has 150 requirements and the cost variation $\hat{c}_i$ is set to be 50% of the estimated cost c_i.

Considering both synthetical and real-world NRP instances, the empirical evaluation was performed over 240 instances.

5.1.2. Search Techniques Employed

In the empirical evaluation, the search techniques Genetic Algorithm, Simulated Annealing and Random Search were considered, as described next:

- **Genetic Algorithm**: widely known evolutionary algorithm, already

applied to many optimization problems and inspired by the Darwin's natural selection theory (Holland, 1975).

- **Simulated Annealing**: algorithm for solving ordinary optimization problems, based on the thermodynamics' annealing process (Kirkpatrick et al., 1983).

- **Random Search**: algorithm that searches for solutions completely at random. It is used as a sanity check, as suggested by Harman (2007).

Both Genetic Algorithm (GA) and Simulated Annealing (SA) were considered because they are two of the most used search techniques in SBSE literature (Harman, 2007).

The GA and SA parameters were empirically obtained through a experimentation process inspired by Souza et al. (2011). First, 20 of the 180 synthetical instances were randomly selected to participate in the configuration process. For both GA and SA, different configurations were considered, varying the values of each technique parameters. For the GA, a total of 27 configurations were produced, varying the values of crossover probability $(60\%, 80\%, 95\%)$, mutation probability $(1/N\%, 0.1\%, 1\%)$ and elitism rate $(0, 10\%, 20\%)$, where N is the number of requirements. For the SA, initial temperature $(20, 50, 200)$ and cooling rate $(1/N\%, 0.1\%, 1\%)$ were examined. It resulted in 9 different SA configurations. For all selected instances, the robustness parameter was set to $\Gamma = 0$. Each algorithm configuration was executed 30 times for each instance, in order to obtain fitness value averages and standard deviations. The release budget is set to 70% of the sum of all requirements' costs. The selected configuration for each search technique was the one which generated the best results among most of the 20 selected instances.

The adaptation of the search techniques to the NRP, along with the results of the parameters configuration process, are presented next.

Genetic Algorithm. Population with N individuals. The initial population is randomly generated and composed only by feasible individuals. If an initial random individual is infeasible, due to budget or interdependencies constraints, it got repaired. The repairing method consists in randomly remove requirements until the solution becomes feasible. Crossover probability is set to 95%, using one point crossover. Mutation is performed for each requirement in the solution with a $1/N\%$ probability, consisting of a single requirement inclusion/exclusion. If necessary, the repairing mechanism is also

performed after the crossover and mutation operators. The GA implementation employs elitism, with 20% of the best individuals in the population being automatically included in the next generation. The algorithm returns the best individual after 1000 generations.

Simulated Annealing. The initial solution is randomly generated using the same procedure as in the GA initial population. Initial temperature and cooling rate were set to 50 and $1/N\%$, respectively. At each iteration, N neighbour solutions are evaluated. A neighbour solution is defined as a solution that can be produced from the original one with just one requirement addition or removal. All search techniques were configured to perform the same number of fitness evaluations. Thus, the final temperature is dynamically calculated in order to permit only $1000 \times N$ fitness evaluations.

Random Search. The random search algorithm consists in generate $1000 \times N$ random solutions. The algorithm returns the best solution overall. The random solution generation is the same as in the GA initial population.

Each search technique was executed for all 240 NRP instances, with different robustness levels for each instance. The robustness parameter value Γ was set to $\{0, 0.05N, 0.1N, 0.15N, \ldots, N\}$. For each instance and each robustness level, each algorithm was executed for 30 times, obtaining fitness value average and standard deviation. The release budget is again setted to 70% of the total possible cost. Considering all search techniques, all instances and all robustness levels, a total of 15120 executions were performed.

In order to permit the full replication of the whole empirical evaluation, all synthetical and real-world instances, along with the source code used, are available at the paper supporting webpage `http://goes.uece.br/mhepaixao/robustNRP/`. It also contains all results that have to be omitted from this paper due to space constraints.

5.2. Results and Analysis

The empirical study results are presented in this section. Each subquestion of the research question RQ_2 is treated separetly as follows.

5.2.1. $RQ_{2.1}$: What is the "price of robustness" for the presented NRP model when using different robustness levels?

This question consists in evaluating how much is lost in solution quality as the robustness parameter Γ varies. In order to perform a fair analysis, one must consider instances with the same cost variation and same interdependency density. The different cost variations strategies and interdependency

densities are addressed in questions $RQ_{2.2}$ and $RQ_{2.3}$, respectively. Therefore, this analysis is based only upon instances with 10% cost variation and no interdependencies. Since the real-world instances are addressed in question $RQ_{2.4}$, only synthetical instances are used in this analysis.

Table 1 presents the results computed by the Genetic Algorithm (GA), Simulated Annealing (SA) and Random Search (RS) for the synthetical instances with 10% cost variation and no interdependencies. Some of the robustness levels are omitted due to space constraints.

As can be seen from the table, as the robustness parameter Γ increases, the fitness value tends to decrease. This behavior is the same for all instances and all search techniques employed. Regarding solution quality, the GA achieved the best results for all instances and all robustness levels. The GA also presented the lowest standard deviation. The RS algorithm could not find good solutions for the presented NRP robust model, being overcame in solution quality by both GA and SA for all instances and robustness levels.

Figure 1 presents the fitness values results of the three search techniques for some of the synthetical instances. The results for the other instances are also similar to those presented in the figure. As stated above, the fitness value clearly decreases as the robustness level increases. In addition, it is also visible the best results found by the GA.

The reduction factor results for the synthetical instances with 10% cost variation and no interdependencies are presented in Table 2. As can be seen from this table, the fitness penalization due to robustness is considerably small. Considering all instances, the GA can achieve 10% of robustness by losing 1.2% in solution quality, in average. For higher robustness levels, like 50% and 100%, the GA average loss is 4.22% and 4.79%, respectively. In other words, using the GA, one can guarantee 100% of robustness by losing a little amount of 4.79% in solution quality. For the SA, the reduction factor for $\Gamma = 0.5N$ and $\Gamma = N$ are 6.94% and 7.21%, respectively. The RS presented low reduction factor results, but since it is actually unable to find good solutions, these results do not express the proper behavior of the presented robust model.

The common assumption is that a different robustness level leads to a different reduction factor. Actually, for low levels of robustness, the reduction factor difference seems to be significant. However, after Γ being set to half the number of requirements, the reduction factor appears to stabilize, as can also be viewed in Figure 1. In order to evaluate this behavior, several samples are composed containing the reduction factors of all instances

Table 1: Fitness values results for synthetical instances with 10% cost variation and no interdependencies

Instance	**ST**	Γ						
		0	$0.1N$	$0.3N$	$0.5N$	$0.7N$	$0.9N$	N
I_S_50_10_0	GA	247.18 ± 2.37	244.34 ± 2.42	239.14 ± 1.47	235.41 ± 2.31	233.38 ± 2.52	234.28 ± 2.31	233.9 ± 2.21
	SA	221.37 ± 3.77	219.21 ± 3.45	213.8 ± 3.54	211.91 ± 3.52	210.81 ± 3.31	209.94 ± 3.07	210.9 ± 2.83
	RS	216.59 ± 4.46	213.95 ± 3.36	211.53 ± 3.83	208.71 ± 3.15	207.58 ± 4.12	207.25 ± 3.46	208.3 ± 3.89
I_S_100_10_0	GA	442.02 ± 1.77	436.79 ± 1.24	427.41 ± 2.21	422.09 ± 2.07	421.1 ± 1.91	419.59 ± 2.06	420.19 ± 1.72
	SA	374.03 ± 7.14	370.97 ± 5.05	362.23 ± 5.35	359.38 ± 5.3	358.05 ± 6.05	356.76 ± 4.36	359.77 ± 5.48
	RS	365.71 ± 6.66	363.56 ± 6.56	357.52 ± 4.93	355.83 ± 4.55	355.21 ± 4.52	355.57 ± 4.37	355.82 ± 6.0
I_S_150_10_0	GA	726.53 ± 2.6	718.86 ± 2.54	709.97 ± 2.09	703.17 ± 2.46	698.73 ± 3.39	699.14 ± 2.4	699.03 ± 2.76
	SA	571.66 ± 14.79	565.89 ± 18.9	555.91 ± 12.7	545.41 ± 11.7	545.45 ± 8.08	542.24 ± 8.63	546.08 ± 7.68
	RS	548.74 ± 10.04	545.02 ± 9.33	545.89 ± 10.64	541.67 ± 6.55	541.31 ± 7.76	538.73 ± 6.71	545.0 ± 9.63
I_S_200_10_0	GA	954.89 ± 1.41	942.07 ± 2.64	924.52 ± 1.92	911.59 ± 2.71	905.24 ± 2.72	905.23 ± 1.84	904.86 ± 2.43
	SA	786.23 ± 18.64	764.01 ± 21.8	739.73 ± 18.62	725.13 ± 12.71	723.47 ± 12.81	725.2 ± 11.12	723.95 ± 13.14
	RS	720.08 ± 11.59	722.52 ± 10.68	719.31 ± 10.98	717.45 ± 9.83	715.73 ± 8.32	716.09 ± 10.54	714.66 ± 7.48
I_S_250_10_0	GA	1143.95 ± 2.83	1129.95 ± 3.21	1108.57 ± 2.99	1094.12 ± 3.34	1088.55 ± 2.97	1087.74 ± 2.51	1087.15 ± 3.12
	SA	941.78 ± 24.0	915.15 ± 26.8	881.67 ± 21.36	867.41 ± 22.0	862.25 ± 18.84	857.91 ± 17.76	869.01 ± 16.08
	RS	849.4 ± 10.87	848.74 ± 10.67	846.99 ± 9.67	848.46 ± 9.07	849.3 ± 9.47	847.99 ± 10.86	848.58 ± 11.38
I_S_300_10_0	GA	1469.1 ± 3.82	1453.43 ± 3.16	1427.47 ± 4.91	1412.69 ± 2.78	1406.98 ± 3.76	1405.29 ± 2.91	1405.84 ± 3.71
	SA	1162.39 ± 24.26	1129.45 ± 35.45	1090.82 ± 39.95	1069.84 ± 25.68	1055.64 ± 19.3	1060.28 ± 28.04	1063.2 ± 20.69
	RS	1037.7 ± 15.38	1035.03 ± 11.91	1032.42 ± 12.49	1038.18 ± 14.54	1032.91 ± 11.72	1034.29 ± 12.77	1036.44 ± 10.15
I_S_350_10_0	GA	1674.29 ± 3.26	1654.59 ± 3.0	1624.21 ± 3.28	1605.46 ± 4.25	1596.94 ± 4.54	1596.16 ± 3.71	1596.66 ± 3.25
	SA	1340.63 ± 24.98	1316.59 ± 26.96	1249.19 ± 22.7	1229.91 ± 26.63	1223.46 ± 22.15	1226.32 ± 30.14	1229.05 ± 29.26
	RS	1190.15 ± 17.95	1189.73 ± 15.16	1194.7 ± 16.25	1190.77 ± 14.23	1197.38 ± 17.01	1190.05 ± 15.23	1195.32 ± 14.39
I_S_400_10_0	GA	1889.65 ± 2.97	1866.14 ± 3.77	1831.72 ± 4.71	1812.2 ± 4.72	1804.9 ± 4.25	1804.19 ± 5.02	1803.19 ± 5.11
	SA	1487.35 ± 34.28	1459.83 ± 34.11	1384.28 ± 24.31	1370.63 ± 39.38	1364.46 ± 29.07	1359.43 ± 33.8	1356.56 ± 34.57
	RS	1313.27 ± 19.02	1311.23 ± 15.39	1304.73 ± 15.68	1309.03 ± 17.98	1302.14 ± 12.27	1308.73 ± 16.42	1303.68 ± 16.13
I_S_450_10_0	GA	2141.9 ± 5.03	2113.57 ± 4.3	2071.92 ± 5.36	2045.75 ± 5.22	2031.64 ± 5.39	2031.18 ± 5.62	2030.51 ± 5.81
	SA	1758.86 ± 25.95	1705.22 ± 32.22	1645.25 ± 32.19	1608.39 ± 32.09	1610.41 ± 27.25	1603.45 ± 29.08	1605.75 ± 22.72
	RS	1310.17 ± 16.28	1311.23 ± 15.39	1304.73 ± 15.68	1309.03 ± 17.98	1302.14 ± 12.27	1308.73 ± 16.42	1303.68 ± 16.13
I_S_500_10_0	GA	2385.36 ± 3.1	2356.31 ± 4.45	2310.83 ± 4.65	2283.37 ± 4.28	2271.31 ± 5.99	2270.22 ± 4.47	2268.79 ± 4.87
	SA	1920.01 ± 34.36	1871.46 ± 30.91	1794.33 ± 38.5	1760.09 ± 35.33	1749.81 ± 30.49	1748.15 ± 30.9	1744.9 ± 39.74
	RS	1643.72 ± 11.84	1646.83 ± 21.31	1647.24 ± 16.07	1647.95 ± 21.04	1647.01 ± 15.94	1647.96 ± 20.51	1647.55 ± 21.66

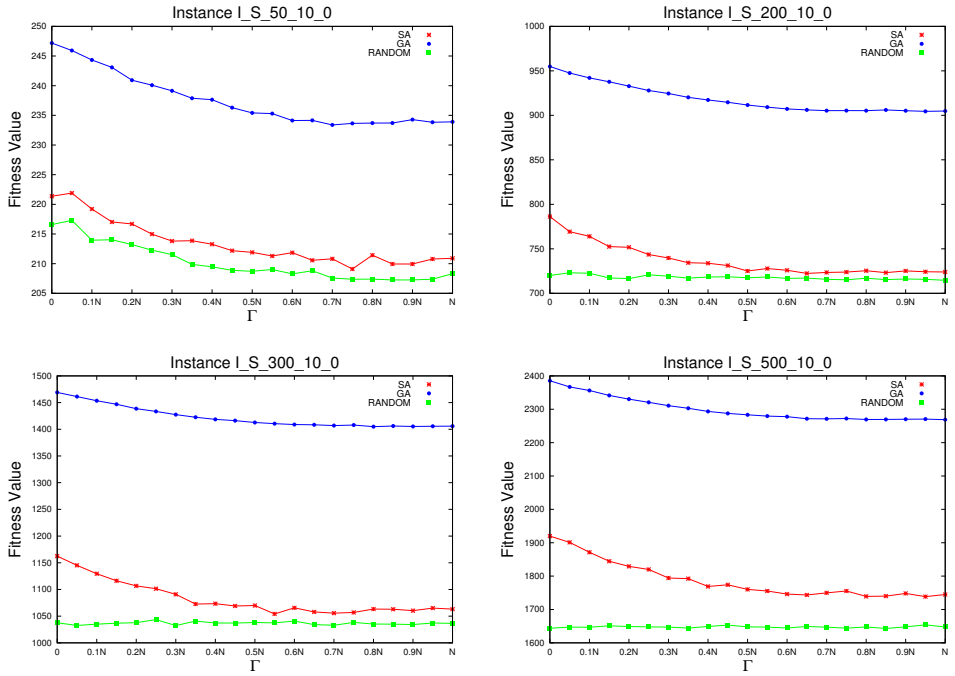

Figure 1: GA, SA and RS fitness value results for some synthetical instances with 10% cost variation and no interdependencies

for a certain robustness level and search technique. Considering the GA, for example, the sample containing the reduction factors for $\Gamma = 0.05N$ is $\{0.51, 0.74, 0.58, 0.77, 0.65, 0.53, 0.57, 0.64, 0.69, 0.77\}$.

As suggested by Arcuri and Briand (2011), the Wilcoxon test of the statistical computing tool *R* (R-Project, 2014) was employed to compare different samples and state if they are statistically different. Such test takes two samples as input and results in a *p-value*. This *p-value* can be understood as the probability of the two samples being actually the same. In this paper, two samples are considered statistically different when the Wilcoxon test results in a *p-value* ≤ 0.05, which represents a significance level of 95%. For this particular analysis, let $p_{a,b}^{ST}$ be the *p-value* for the comparison between the sample with robustness level a and the sample with robustness level b, for the search technique ST.

Considering the GA, for low levels of robustness, like $\Gamma = 0.05N$ and $\Gamma = 0.1N$, the *p-value* is $p_{0.05,0.1}^{GA} = 1.79 \times 10^{-3}$. This value means that the probability of the samples being really different is greater than 99%. In

Table 2: Reduction factor results for synthetical instances with 10% cost variation and no interdependencies

Instance	**ST**	Γ						
		$0.05N$	$0.1N$	$0.3N$	$0.5N$	$0.7N$	$0.9N$	N
I_S_50_10_0	GA	0.51	1.15	3.25	4.76	5.58	5.22	5.37
	SA	0.23	0.98	3.42	4.27	4.77	5.16	4.73
	RS	0.33	1.22	2.34	3.64	4.16	4.31	3.83
I_S_100_10_0	GA	0.74	1.18	3.31	4.51	4.73	5.07	4.94
	SA	0.82	0.82	3.15	3.92	4.27	4.62	3.81
	RS	0.3	0.59	2.24	2.7	2.87	2.77	2.7
I_S_150_10_0	GA	0.58	1.06	2.28	3.22	3.83	3.77	3.79
	SA	0.54	1.01	2.76	4.59	4.58	5.15	4.47
	RS	0.71	0.68	0.52	1.29	1.35	1.82	0.68
I_S_200_10_0	GA	0.77	1.34	3.18	4.53	5.2	5.2	5.24
	SA	2.16	2.83	5.91	7.77	7.98	7.76	7.92
	RS	0.42	0.34	0.11	0.37	0.6	0.55	0.75
I_S_250_10_0	GA	0.65	1.22	3.09	4.36	4.84	4.91	4.97
	SA	1.4	2.83	6.38	7.9	8.44	8.91	7.73
	RS	0.26	0.08	0.28	0.11	0.01	0.17	0.1
I_S_300_10_0	GA	0.53	1.07	2.83	3.84	4.23	4.34	4.31
	SA	1.48	2.83	6.16	7.96	9.18	8.78	8.53
	RS	0.49	0.26	0.51	0.05	0.46	0.33	0.12
I_S_350_10_0	GA	0.57	1.18	2.99	4.11	4.62	4.67	4.64
	SA	1.02	1.79	6.82	8.26	8.74	8.53	8.32
	RS	0.33	0.04	0.38	0.05	0.61	0.01	0.43
I_S_400_10_0	GA	0.64	1.24	3.07	4.1	4.48	4.52	4.58
	SA	1.28	1.85	6.93	7.85	8.26	8.6	8.79
	RS	0.24	0.08	0.42	0.09	0.61	0.11	0.5
I_S_450_10_0	GA	0.69	1.32	3.27	4.49	5.15	5.17	5.2
	SA	0.86	3.05	6.46	8.55	8.44	8.84	8.71
	RS	0.13	0.16	0.19	0.19	0.28	0.21	0.23
I_S_500_10_0	GA	0.77	1.22	3.12	4.28	4.78	4.83	4.89
	SA	0.99	2.53	6.55	8.33	8.86	8.95	9.12
	RS	0.21	0.19	0.21	0.26	0.2	0.26	0.23

other words, it is almost certain that a increase in the robustness level from $\Gamma = 0.05N$ to $\Gamma = 0.1N$ leads to a significant change in solution quality. For other low values of Γ, this behavior continues: $p^{GA}_{0.1,0.15} = 1.78 \times 10^{-3}$, $p^{GA}_{0.15,0.2} = 3.24 \times 10^{-3}$, $p^{GA}_{0.2,0.25} = 4.0 \times 10^{-2}$. But, as the robustness level increases, the differences between samples became statistically less obvious. For $\Gamma = 0.5N$ and $\Gamma = 0.55N$, for example, the test result is $p^{GA}_{0.5,0.55} = 0.40$, which indicates a 40% probability that the samples are actually the same. For bigger values of Γ, the *p-values* are even bigger: $p^{GA}_{0.9,0.95} = 0.96$ and $p^{GA}_{0.95,1} = 1$. Such results indicate that, for higher levels of robustness, small changes in Γ do not influence the "price of robustness".

For the SA the results are similar. For small values of Γ, the *p-values* are $p^{SA}_{0.05,0.1} = 0.028$, $p^{SA}_{0.1,0.15} = 0.053$ and $p^{SA}_{0.15,0.2} = 0.075$. For bigger robustness

levels, the *p-values* are $p^{SA}_{0.85,0.9} = 0.59$, $p^{SA}_{0.9,0.95} = 0.79$ and $p^{SA}_{0.95,1} = 0.63$. As one can see, the same behavior was found by the SA. For small robustness levels, a minor increase in Γ leads to a statistically significant change in fitness value. For greater values, a increase in Γ does not influence the reduction factor.

An interesting fact is that the reduction factor results between instances seem to be similar. The Wilcoxon test is also used to evaluate this statement. In this case, a sample consists in the reduction factor results of a single instance for all robustness levels, including the ones that are not presented in Table 2. For this analysis, let $p^{ST}_{a,b}$ be the *p-value* for the comparison between the sample with number of requirements a and the sample with number of requirements b, for the search technique ST.

Considering the GA results for the instances with 200 and 450 requirements, for example, the *p-value* computed is $p^{GA}_{200,450} = 0.56$, which higlhy indicates no differences between results. In the other hand, for the instances with 50 and 150 requirements, the *p-value* is $p^{GA}_{50,150} = 0.01$, which statistically states a difference between the reduction factor of the two instances. Thus, in order to compare the "price of robustness" between instances with different number of requirements, the test was executed for all possible pairs of such instances.

As a result, 36 out of 45 tests presented a *p-value* smaller than 0.05. Therefore, there is strong empirical and statistical evidence that the "price of the robustness" of the presented model is independent of the number of requirements in the instance.

The question $RQ_{2.1}$ is related to the "price of robustness" of the presented robust NRP model and its variance as the robustness level changes. The analysis presented above shows that, as the robustness level increases, the fitness value tends to decrease. However, this loss in solution quality is only statistical significant for small values of Γ. From $\Gamma = 0.5N$, there is no statistical evidence to comprove that a higher level of robustness leads to a greater loss in solution quality. In other words, ensure a robustness level of 50% is not much different than ensure a robustness level of 55% or even more. In addition, regarding the "price of robustness", the presented model performs nearly the same for instances with different number of requirements. Encouraged by this stability, it is possible to claim that a high level of robustness can be achieved by losing a considerably small amount of fitness. Considering the GA, a 100% robustness can be ensured with a average fitness loss of just 4.79%.

Since the GA has found the best solutions and has presented the lowest standard deviation for all instances and all robustness levels, it will be the only search technique considered in the further analysis.

5.2.2. $RQ_{2.2}$: What is the "price of robustness" for the presented NRP model when using different variations in the requirements' costs?

As stated earlier, different development teams have divergent estimating skills. This question $RQ_{2.2}$ is related to the "price of robustness" when different cost variation strategies are employed. Table 3 presents the reduction factor results computed by the GA for different cost variation strategies and different robustness levels. The different cost variations (CV) considered were: 10%, 20%, 30%, 40%, 50% and random (RD), as presented in the empirical study settings. Interdependencies between requirements are not considered. Some instances results are omitted due to space contraints.

As can be noticed and as expected, considering a single instance, the higher the cost variation, the higher is the reduction factor. This growth is not linear, tough. For the instance with 300 requirements, for example, with a 10% cost variation, a robustness level of 30% leads to a reduction factor of 2.83%. Changing the cost variation to 30% and 50%, the lost in fitness grows to 8.39% and 13.77%, respectively. A cost variation of 50% can be considered a big one, and even with this high variation, it is still possible to reach significant robustness levels with considerably little loss in solution quality. Considering all instances and setting the expected cost variation to be half of the requirement's original cost ($\hat{c}_i = 50\%$), a 100% robustness level can be achieved by sacrificing only around 19% in fitness value.

Figure 2 presents the GA reduction factor results for some of the synthetical instances that are not presented in Table 3. As can be noticed, the reduction factor visibly increases as the cost variation also increases. It is also noteworthy the results similarities between all different instances.

Considering any of the instances in Figure 2, the reduction factor results of different cost variations strategies seem to be almost completely different from each other. In order to evaluate this behavior, the samples are now formed by the reduction factors of a certain cost variation strategy for a certain instance. Let $p^{NR}_{a,b}$ be the *p-value* for the comparison between the sample with cost variation a% and the sample with cost variation b%, for the instance with number of requirements NR.

For the instance with 250 requirements, for example, the results are: $p^{250}_{10,20} = 3 \times 10^{-3}$, $p^{250}_{20,30} = 3.3 \times 10^{-2}$, $p^{250}_{30,40} = 0.012$, $p^{250}_{40,50} = 0.017$ and

Table 3: Genetic Algorithm reduction factor results for synthetical instances with different cost variation and no interdependencies

Instance	**CV**	Γ						
		$0.05N$	$0.1N$	$0.3N$	$0.5N$	$0.7N$	$0.9N$	N
I_S_100_CV_0	10%	0.74	1.18	3.31	4.51	4.73	5.07	4.94
	20%	1.31	2.53	6.31	8.37	9.03	9.16	9.11
	30%	2.04	4.01	9.31	12.14	13.0	12.84	13.01
	40%	2.53	5.09	12.13	15.3	16.2	16.02	15.94
	50%	3.3	6.35	15.19	18.45	19.19	18.93	18.98
	RD	2.8	5.09	9.02	10.17	10.44	10.39	10.41
I_S_200_CV_0	10%	0.77	1.34	3.18	4.53	5.2	5.2	5.24
	20%	1.34	2.47	6.37	8.85	9.85	9.85	9.84
	30%	1.99	3.84	9.33	12.84	13.87	13.83	13.81
	40%	2.8	5.13	12.31	16.42	17.48	17.39	17.55
	50%	3.41	6.26	15.24	19.78	20.73	20.77	20.75
	RD	2.33	4.09	8.83	10.79	11.28	11.38	11.29
I_S_300_CV_0	10%	0.53	1.07	2.83	3.84	4.23	4.34	4.31
	20%	1.18	2.3	5.69	7.55	8.21	8.3	8.35
	30%	1.85	3.44	8.39	11.02	11.87	11.95	11.84
	40%	2.35	4.59	11.1	14.15	15.06	15.12	14.89
	50%	3.03	5.74	13.77	17.28	18.11	17.98	17.98
	RD	2.38	4.16	7.99	9.51	10.03	9.95	9.88
I_S_400_CV_0	10%	0.64	1.24	3.07	4.1	4.48	4.52	4.58
	20%	1.23	2.33	5.92	7.81	8.59	8.53	8.51
	30%	1.91	3.66	8.93	11.51	12.31	12.19	12.32
	40%	2.57	4.85	11.68	14.79	15.67	15.7	15.71
	50%	3.19	6.16	14.31	18.01	18.6	18.6	18.65
	RD	2.33	4.1	8.03	9.87	10.27	10.15	10.24
I_S_500_CV_0	10%	0.77	1.22	3.12	4.28	4.78	4.83	4.89
	20%	1.28	2.39	6.08	8.18	9.02	9.01	9.03
	30%	1.83	3.7	9.0	11.8	12.83	12.91	12.78
	40%	2.59	4.87	11.98	15.38	16.39	16.31	16.31
	50%	3.22	6.04	14.62	18.56	19.49	19.54	19.41
	RD	2.47	4.28	8.29	9.94	10.46	10.41	10.53

$p^{250}_{50,RD} = 4.9 \times 10^{-3}$. As one can see, the reduction factor results for different cost variations are really different from each other. These results statistically state that a change in the cost variation strategy leads to a different "price of robustness". For all other instances the results are similar.

It was stated in the previous section that, for a cost variation of 10% and no interdependencies, the "price of robustness" is only statistically influenced by the robustness level for small values of Γ. When all possible cost variation strategies are considered, this behavior remains the same. For this analysis, the samples are formed by the reduction factor results of all instances and all cost variations for a certain robustness level. Interdependencies are not considered. For the GA, the *p-values* for small values of Γ are $p^{GA}_{0.05,0.1} = 5.32 \times 10^{-9}$, $p^{GA}_{0.1,0.15} = 3.33 \times 10^{-3}$ and $p^{GA}_{0.15,0.2} = 8.09 \times 10^{-2}$. For bigger robustness levels the *p-values* are $p^{GA}_{0.5,0.55} = 0.62$, $p^{GA}_{0.9,0.95} = 0.97$ and $p^{GA}_{0.95,1} = 0.96$.

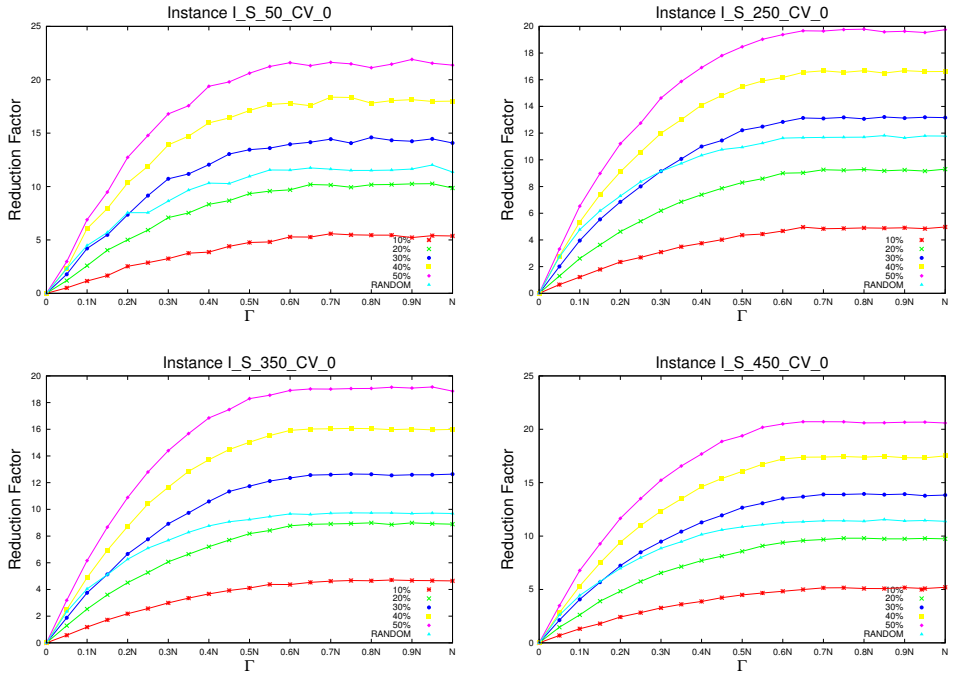

Figure 2: GA reduction factor results for some synthetical instances with different cost variations and no interdependencies

Thereby, the "price of robustness" behavior of becoming almost constant from $\Gamma = 0.5N$ is statistically proved for all cost variation strategies.

The cost variation strategy plays an important role in the "price of robustness". It is statistically proved that different cost variations leads to different reduction factors, in a way that as higher the variation, higher is the reduction. This increase in the "price of robustness" is not linear, though. Even with high cost variation, like 50%, it is possible to achieve a 100% of robustness losing, in average, just 19% in solution quality. It is also statistically proved, for all cost variations considered in this analysis, that a change in the robustness level is only significant at low values of Γ. For bigger robustness levels, the "price of robustness" can be considered the same.

5.2.3. $RQ_{2.3}$: What is the "price of robustness" for the presented NRP model when there are interdependencies between requirements?

In a next release planning situation, requirements can have interdependencies between each other. The interdependency density (ID) represents the

percentage of requirements which have interdependencies in the instance, as presented in the empirical study settings.

In this empirical study, the different interdependency densities considered were: 0, 10% and 20%. Since the different cost variation strategies were addressed in the previous section, this analysis considers only the instances with 10% cost variation. Table 4 presents the GA reduction factor results for the instances with 10% cost variation and different interdependency densities.

Table 4: Genetic Algorithm reduction factor results for synthetical instances with 10% cost variation and different interdependency densities

Instance	**ID**	Γ						
		$0.05N$	$0.1N$	$0.3N$	$0.5N$	$0.7N$	$0.9N$	N
I_S_50_10_ID	0	0.51	1.15	3.25	4.76	5.58	5.22	5.37
	10%	0.41	1.29	3.75	4.93	5.28	6.03	5.12
	20%	3.22	3.53	5.58	4.9	7.79	7.12	5.39
I_S_100_10_ID	0	0.74	1.18	3.31	4.51	4.73	5.07	4.94
	10%	0.83	1.55	3.58	4.62	5.29	5.27	5.22
	20%	0.24	1.26	2.97	4.68	4.83	4.14	4.72
I_S_150_10_ID	0	0.58	1.06	2.28	3.22	3.83	3.77	3.79
	10%	0.4	0.55	2.4	3.35	3.09	3.57	3.24
	20%	1.1	0.98	0.14	1.89	1.11	1.25	0.69
I_S_200_10_ID	0	0.77	1.34	3.18	4.53	5.2	5.2	5.24
	10%	1.19	1.61	3.27	4.13	5.41	4.69	5.01
	20%	0.89	1.28	3.07	4.8	5.16	4.82	4.89
I_S_250_10_ID	0	0.65	1.22	3.09	4.36	4.84	4.91	4.97
	10%	0.65	1.7	3.46	4.4	5.08	5.38	5.41
	20%	1.1	1.37	3.58	4.24	5.1	5.33	5.17
I_S_300_10_ID	0	0.53	1.07	2.83	3.84	4.23	4.34	4.31
	10%	0.5	1.0	2.63	3.3	4.01	3.75	3.83
	20%	0.08	0.86	2.49	3.36	3.96	3.45	3.68
I_S_350_10_ID	0	0.57	1.18	2.99	4.11	4.62	4.67	4.64
	10%	0.25	0.76	2.99	4.02	4.38	4.4	4.72
	20%	0.01	0.88	2.86	4.09	4.11	4.03	4.18
I_S_400_10_ID	0	0.64	1.24	3.07	4.1	4.48	4.52	4.58
	10%	0.57	0.98	2.94	3.73	4.26	4.44	4.35
	20%	0.2	0.15	2.06	3.64	3.58	3.36	3.52
I_S_450_10_ID	0	0.69	1.32	3.27	4.49	5.15	5.17	5.2
	10%	0.77	1.43	3.85	4.88	5.34	5.56	5.48
	20%	0.34	0.27	1.51	2.49	3.19	3.12	3.1
I_S_500_10_ID	0	0.77	1.22	3.12	4.28	4.78	4.83	4.89
	10%	1.28	1.16	3.46	4.94	4.95	5.18	5.04
	20%	0.2	0.1	0.9	2.17	2.35	2.27	2.18

For an interdependency density of 10%, for example, one can achieve a 30% of robustness by losing 3.23% in fitness value, in average. Regarding an interdependency density of 20%, 70% of robustness is reached by losing, in average, 4.11% in solution quality. When the interdependencies between requirements are considered, the "price of robustness" is still considerably small.

As one can see, considering any of the instances presented in Table 4, the reduction factor results seem to be similar, regardless the interdependency density considered. In order to statistically evaluate this statement, the samples are formed by the reduction factor results of a certain instance for a certain value of interdependency density. Let $p_{a,b}^{NR}$ be the *p-value* for the comparison between the sample with interdependency density $a\%$ and the sample with interdependency density $b\%$, for the instance with number of requirements NR.

For the instance with 200 requirements, for example, the *p-values* found are: $p_{0,10}^{200} = 0.696$, $p_{0,20}^{200} = 0.489$ and $p_{10,20}^{200} = 0.959$. For the other instances presented in the table, the results are also quite similar. Such results statistically state that the "price of robustness" is the same for different interdependency densities between requirements. For the same instances, with different cost variation strategies, the results are also similar.

Even when interdependencies are considered, the impact of the robustness level in the "price of robustness" is pretty much the same as stated in the previous sections. Being the samples now formed by the reduction factor results of a certain robustness level for the instances with different interdependency densities and 10% cost variation, the *p-values* for small values of Γ are: $p_{0.05,0.1}^{GA} = 4.77 \times 10^{-5}$, $p_{0.1,0.15}^{GA} = 2.95 \times 10^{-5}$ and $p_{0.15,0.2}^{GA} = 8.10 \times 10^{-3}$. For bigger robustness levels, the *p-values* are: $p_{0.6,0.65}^{GA} = 0.437$, $p_{0.9,0.95}^{GA} = 0.727$ and $p_{0.95,1}^{GA} = 0.767$. A variation in the robustness level only has a significant influence in the "price of robustness" for small values of Γ. From $\Gamma = 0.6N$, the loss in solution quality became almost constant.

From the results presented, it is possible to state that, in general, the interdependency density does not influence the "price of robustness" of the presented model. As in the previous analysis, the penalization due to robustness is small and the reduction factor for different interdependency densities are statistically the same, regardless the number of requirements and cost variation strategy. Moreover, the robust model behavior for different robustness levels is not affected by the differences in the interdependency densities. Variations in the Γ parameter are only statistically significant for small robustness levels.

5.2.4. $RQ_{2.4}$: What is the "price of robustness" for the presented NRP model when it is applied to real-world NRP instances?

In order to answer this question, the same analysis that have been done for the synthetical instances are also performed for the real-world instances.

Table 5 presents the GA results for some of the real-world instances with different cost variation strategies. Results from both Eclipse and Mozilla projects are preseented. The results of the remaining real-world instances are omitted due to space constraints.

Table 5: Genetic Algorithm reduction factor results for real-world instances with different cost variations

Instance	**PD**	Γ						
		$0.05N$	$0.1N$	$0.3N$	$0.5N$	$0.7N$	$0.9N$	N
I_Re_E_20_CV	10%	3.02	3.22	4.45	6.24	6.03	7.33	8.22
	20%	1.45	3.94	8.37	12.24	11.68	11.34	11.89
	30%	2.96	4.82	10.74	15.91	16.53	16.53	15.91
	40%	4.33	7.49	14.85	18.56	21.92	21.24	21.03
	50%	3.96	7.5	17.98	22.29	24.38	24.38	22.98
	RD	4.3	5.93	11.39	13.51	13.51	13.78	13.71
I_Re_M_50_CV	10%	0.22	0.86	3.03	4.29	5.02	4.87	4.9
	20%	0.84	2.28	6.1	8.67	9.89	10.06	10.26
	30%	1.58	3.87	9.57	13.09	14.03	14.24	14.16
	40%	2.04	4.73	12.77	16.62	17.24	16.95	16.85
	50%	2.31	5.58	15.29	19.42	20.09	20.21	20.09
	RD	2.0	4.32	8.29	10.53	11.32	11.09	10.68
I_Re_E_100_CV	10%	0.72	1.31	3.54	5.59	6.49	6.44	6.28
	20%	1.91	2.72	7.49	11.7	12.25	12.16	12.12
	30%	2.56	4.21	10.79	16.04	16.86	16.73	16.96
	40%	2.76	5.66	14.3	20.12	20.65	21.04	20.84
	50%	4.06	7.42	18.0	23.82	24.89	24.59	24.88
	RD	3.45	5.68	11.46	13.55	13.95	14.31	14.17
I_Re_M_150_CV	10%	0.47	0.98	2.52	4.12	4.76	4.66	4.78
	20%	1.04	2.19	5.56	8.57	9.3	9.15	9.29
	30%	1.69	3.23	8.26	12.12	13.02	13.1	13.17
	40%	2.38	4.53	11.19	16.11	16.74	16.57	16.51
	50%	2.91	5.45	13.81	19.08	19.48	19.64	19.56
	RD	2.04	4.11	8.94	10.94	11.14	10.99	11.22
I_Re_E_200_CV	10%	0.85	1.55	3.75	5.91	6.84	6.65	6.7
	20%	1.59	2.76	7.42	11.64	12.21	12.35	12.42
	30%	2.66	4.84	11.36	16.42	17.55	17.54	17.34
	40%	2.96	5.62	14.94	20.5	21.58	21.49	21.28
	50%	3.82	7.28	18.59	24.65	25.11	25.14	25.26
	RD	3.14	5.5	11.78	13.45	13.47	13.97	14.09

For a cost variation of 10%, a robustness level of 100% is achieved by losing 6.17% in fitness value, in average. For the real-world NRP instances, as well as for the synthetical instances, the presented NRP robust model allows a high level of robustness with a considerably low penalization regarding solution quality. As the cost variation increases, the reduction factor also increases, which is a behavior also present in the synthetical instances results. For a high value of cost variation, like 50%, one can ensure a 100% of robustness by losing around 22% in solution quality. Such "price of robustness" can still be considered a small one.

When comparing the reduction factors of a same instance with different cost variations, as presented by the synthetical instances, the differences in the results are statistically proved. Let $p_{a,b}^{P\_NR}$ be the *p-value* for the comparison between the sample with cost variation $a\%$ and the sample with cost variation $b\%$, for the instance from project P (E for Eclipse and M for Mozilla) and number of requirements NR. For the instance from the Eclipse project with 100 requirements, for example, the *p-values* are: $p_{10,20}^{E\_100} = 7.1 \times 10^{-3}$, $p_{20,30}^{E\_100} = 6.3 \times 10^{-2}$, $p_{30,40}^{E\_100} = 0.015$, $p_{40,50}^{E\_100} = 0.014$ and $p_{50,RD}^{E\_100} = 7.8 \times 10^{-3}$. For all others real instances, the *p-values* are similar. Therefore, the cost variation strategy has a significant influence in the "price of robustness" of the real-world instances.

Regarding the "price of robustness" for the real-world instances when the robustness level varies, the results are also similar to the synthetical instances. Samples are composed by the reduction factor results of a certain robustness level for all real instances, including the ones that are not presented in the table. For small values of Γ, a change in robustness presents a significant change in the reduction factor. The *p-values* are: $p_{0.05,0.1}^{GA} = 1.21 \times 10^{-8}$, $p_{0.1,0.15}^{GA} = 3.65 \times 10^{-3}$ and $p_{0.15,0.2}^{GA} = 5.1 \times 10^{-2}$. For bigger values of Γ, the change in reduction factor is almos negligible: $p_{0.5,0.55}^{GA} = 0.7$, $p_{0.9,0.95}^{GA} = 0.98$ and $p_{0.95,1}^{GA} = 0.96$.

In conclusion, all results reported in this analysis are consistent to show that the behavior of the presented robust NRP formulation, regarding its "price of robustness", are similar to those found over synthetical instances. That is, the penalization due to robustness is very small and the cost variation strategy employed has a major role in the "price of robustness". In addition, the robustness level presents a statistical influence in the "price of robustness" only for small values of Γ.

Finally, the results presented in the above sections have helped in fully answering the reasearch question RQ_2, pointing out the ability of the model to produce robust solutions with significantly small loss in solution quality, regardless the number of requirements, cost variation strategy and interdependency density.

5.2.5. Threats to Validity

In the paper by Barros and Dias-Neto (2011), several threats to the validity of SBSE empirical studies are presented. Such threats are potential risks involved in the design and execution of empirical studies that may limit the reliability of the study and complicate the generalization of the results to a

larger population of instances. These threats can be classified into different types: conclusion, internal, external and construct threats.

In this subsection, the major threats to validity in SBSE empirical studies are considered and it is presented how this paper tried to handle such threats in order to minimize their effects.

- **Conclusion**

Not accounting for random variation. In order to account for the non-deterministic aspect of the search techniques employed, each algorithm was executed 30 times for each instance and each robustness level. Despite 30 being a well accepted number of runs by the SBSE literature (Arcuri and Briand, 2011), a greater number of runs would have provided more reliable results.

Lack of good descriptive statistics. Most of the conclusions drawed from the empirical study results are based on statistical tests.

Lack of a meaningful comparison baseline. The Random Search is used as a sanity check for the search techniques employed in the empirical study. Since this paper is focused in evaluating the behavior of the presented robust model, there is no actually need to use more sophisticated search techniques.

- **Internal**

Poor parameter settings. The search techniques parametrization procedure is well discussed in this paper. Various different configurations were evaluated in more than 10% of the total number of instances used in the empirical study. The best configuration of each search technique was choosen. This parametrization process is actually based in the procedure presented by Souza et al. (2011).

Lack of discussion on instrumentation of code. The instrumentation of the source code used in the empirical study is not discussed in the paper but the code is available at the paper supporting webpage. All instances are also available, which makes the empirical study fully replicable.

Lack of description of data collection procedures. The design and generation of the synthetical instances are well discussed in the empirical study settings section. Although the instances were generated presenting different next release planning situations, a bigger number of instances would have provided more generalizable results.

Lack of using real problem instances. This paper uses real-world NRP instances from the work by Xuan et al. (2012), which were adapted from bug repositories. Due to the nature of these instances, they do not present different scenarios. Therefore, different kinds of real instances, not only from bug repositories, that could fully explore the model's features, would have also provided more generalizable results.

- **External**

Lack of a definition of target instances. The motivation for the design of the synthetical instances is discussed along the instances generation procedure.

Lack of instances of growing size. Instances of different sizes were used in the empirical study, including considerably big instances, with 400, 450 and 500 requirements.

Lack of instances of growing complexity. Regarding the NRP, the complexity of a instance is directly related to its size. Thus, the empirical study considered instances with different complexities as well.

- **Construct**

Lack of validity of cost measures. This empirical study does not assess nor compare execution cost and performance of search techniques.

Lack of validity of effectiveness measures. Related to the adaptation of a software engineering problem to a fitness function. The fitness function used in this empirical study is a generalization of the fitness function found in the NRP literature, which is highly studied and accepted as a good representation of a real next release planning situation.

6. Conclusion and Future Works

The next release problem is an important task in the iterative and incremental software development model. For this problem, optimization models have been proposed to search for solutions based on estimates of requirements' importance and cost. However, such estimates may turn out to be erroneous, which can invalidate the search process.

The robust optimization is a framework that identifies and quantifies uncertainties in optimization problems. It can be used to handle the NRP input uncertainties and produce robust solutions for this problem. A robust

optimization model for the NRP have already been proposed by Paixao and Souza (2013) and this paper is intended to be a extension of this original work.

In the presented robust model, the uncertainties related to the requirement's importance were modeled in a discrete way using the concept of scenarios. Differently, the uncertainties regarding requirement's cost were treated in a continuous way by considering the expected cost variation for each requirement, along with a control parameter which allows for the adjustment of the desired level of robustness based on the development team estimating history.

The model was applied to both synthetical and real-world NRP instances. The first ones were randomly generated and the second ones were extracted from bug repositories of two large open source software projects (Eclipse and Mozilla). The instances set was designed to present different next release planning situations, including instances with different number of requirements, different cost variations strategies and different interdependencies between requirements. In order to evaluate the "price of robustness" for the presented model, an empirical study was designed, executed and analyzed.

First, the model was applied to synthetical instances, using a cost variation set to 10% of the original requirement's cost and no interdependencies. As a result, it was demonstrated that the gain in robustness was obtained with a considerably small loss in fitness value for all instances. Statistical tests were used to evaluate the impact of the robustness level in the "price of robustness". As a result, the model presented a nearly constant fitness loss when the robustness control parameter was calibrated to at least half the number of requirements. Furthermore, statiscal tests also stated that the "price of robustness" is pratically independent from the number of requirements in the instance.

After that, the "price of robustness" was computed for the synthetical instances with different cost variation approaches. As expected and statistically proved, the higher the cost variation, the higher the fitness value loss. Nonetheless, even with high variations, it was still possible to achieve high robustness levels by losing only a small fitness fraction. Despite influencing in the "price of robustness" itself, the cost variation does not influence the model behavior of presenting an almost constant fitness loss after intermediate robustness levels.

In sequence, the model was applied to instances with different interdependencies between requirements. It was proved that different interdepen-

dencies do not have influence in the "price of robustness". The model could still generate solutions with little penalization due to robustness and could still maintain the same loss in solution quality at intermediate levels of robustness.

At the end, the robust model was applied to the set of real-world instances. The results were nearly the same as for the synthetical instances. Accordingly, all results indicate that the proposed formulation can be employed to produce robust solutions with very little loss with regard to quality, even in large-scale real-world projects.

Based on the results of both synthetical and real instances, a summary of the main findings of the empirical study performed in this paper is presented next:

- The "price of robustness" is considerably small, even in worst case situations.
- The loss in solution quality is only influenced by the level of robustness on its low values. For robustness levels bigger than half the number of requirements, the "price of robustness" is almost constant.
- As higher the cost variation, higher is the "price of robustness".
- Interdependencies do not influence the "price of robustness".
- The "price of robustness" is independent of the number of requirements.

As an extension of the work by Paixao and Souza (2013), this paper presents a separated section for both related works and the robust optimization framework. The first one is useful to contextualize this paper in the NRP literature and the second one is helpful in provide more information about the robust optimization, which makes the robust NRP model easy to understand. Regarding the model's evaluation, the new empirical study presents a considerably increase in the number and diversity of instances. Interdependencies between requirements are now considered. Since the number of algorithms executions was more than three times greater, the results found by this empirical study are more reliable. In addition, almost all model's behaviors drawed from the results are based on statistical tests, which increases the generalization of the conclusions.

Since the robust optimization framework has been only applied to problems related to requirements engineering, a natural future research direction

points out to the application of this framework to other software engineering problems subject to uncertainties. In addition, specifically related to the next release problem, other empirical studies could be proposed to evaluate the "price of robustness" under different next release planning conditions. It is also expected to look into different quantification strategies for the requirements' importance and cost uncertainties. Further studies on different strategies to obtain the robustness control parameter are needed. Finally, it seems also interesting to consider other metaheuristics, such as ant colony optimization or particle swarm optimization, as well as exact techniques, to evaluate whether different behaviors can be found.

References

Aissi, H., Bazgan, C., Vanderpooten, D., 2009. Min–max and min–max regret versions of combinatorial optimization problems: A survey. European Journal of Operational Research 197 (2), 427–438.

Antoniol, G., Di Penta, M., Harman, M., 2004. A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty. In: Software Metrics, 2004. Proceedings. 10th International Symposium on. IEEE, pp. 172–183.

Arcuri, A., Briand, L., 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Software Engineering (ICSE), 2011 33rd International Conference on. IEEE, pp. 1–10.

Bagnall, A. J., Rayward-Smith, V. J., Whittley, I. M., 2001. The next release problem. Information and Software Technology 43 (14), 883–890.

Bai, D., Carpenter, T., Mulvey, J., 1997. Making a case for robust optimization models. Management science 43 (7), 895–907.

Baker, P., Harman, M., Steinhofel, K., Skaliotis, A., 2006. Search based approaches to component selection and prioritization for the next release problem. In: Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on. IEEE, pp. 176–185.

Barros, M., Dias-Neto, A. C., 2011. A survey of empirical investigations on ssbse papers. In: Search Based Software Engineering. Springer, pp. 268–268.

Bertsimas, D., Sim, M., 2004. The price of robustness. Operations research 52 (1), 35–53.

Beyer, H. G., Sendhoff, B., 2007. Robust optimization–a comprehensive survey. Computer methods in applied mechanics and engineering 196 (33), 3190–3218.

Boehm, B., Abts, C., Chulani, S., 2000. Software development cost estimation approaches, a survey. Annals of Software Engineering 10 (1), 177–205.

Cao, L., Ramesh, B., 2008. Agile requirements engineering practices: An empirical study. Software, IEEE 25 (1), 60–67.

Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Natt och Dag, J., 2001. An industrial survey of requirements interdependencies in software product release planning. In: Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on. IEEE, pp. 84–91.

Del Sagrado, J., Aguila, I. M., Orellana, F. J., 2011. Requirements interaction in the next release problem. In: Proceedings of the 13th annual conference companion on Genetic and evolutionary computation. ACM, pp. 241–242.

Del Sagrado, J., del Aguila, I. M., Orellana, F. J., 2010. Ant colony optimization for the next release problem: A comparative study. In: Search Based Software Engineering (SSBSE), 2010 Second International Symposium on. IEEE, pp. 67–76.

Du, X., Wang, Y., Chen, W., 2000. Methods for robust multidisciplinary design. AIAA 1785, 1–10.

Dulikravich, G. S., Egorov-Yegorov, I. N., 2005. Robust optimization of concentrations of alloying elements in steel for maximum temperature, strength, time-to-rupture and minimum cost and weight. ECCOMAS–Computational Methods for Coupled Problems in Science and Engineering, 25–28.

Eclipse, 2014. `http://www.eclipse.org/`, acessado em: Junho de 2014.

Harker, S. D. P., Eason, K. D., Dobson, J. E., 1993. The change and evolution of requirements as a challenge to the practice of software engineering. In: Requirements Engineering, 1993., Proceedings of IEEE International Symposium on. IEEE, pp. 266–272.

Harman, M., 2007. The current state and future of search based software engineering. In: 2007 Future of Software Engineering. IEEE Computer Society, pp. 342–357.

Harman, M., Krinke, J., Ren, J., Yoo, S., 2009. Search based data sensitivity analysis applied to requirement engineering. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. ACM, pp. 1681–1688.

Holland, J., 1975. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. USA: University of Michigan.

Huang, C., Lyu, M. R., 2005. Optimal release time for software systems considering cost, testing-effort, and test efficiency. Reliability, IEEE Transactions on 54 (4), 583–591.

Jiang, H., Zhang, J., Xuan, J., Ren, Z., Hu, Y., 2010. A hybrid aco algorithm for the next release problem. In: Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on. IEEE, pp. 166–171.

Kirkpatrick, S., Vecchi, M. P., Gelatt, C. D., 1983. Optimization by simmulated annealing. science 220 (4598), 671–680.

Leung, S. C. H., Tsang, S. O. S., Ng, W., Wu, Y., 2007. A robust optimization model for multi-site production planning problem in an uncertain environment. European Journal of Operational Research 181 (1), 224–238.

Li, M., Azarm, S., 2008. Multiobjective collaborative robust optimization with interval uncertainty and interdisciplinary uncertainty propagation. Journal of mechanical design 130 (8).

Malcolm, S. A., Zenios, S. A., 1994. Robust optimization for power systems capacity expansion under uncertainty. Journal of the operational research society, 1040–1049.

Mozilla, 2014. `http://www.mozilla.org/`, acessado em: Junho de 2014.

Mulvey, J. M., Vanderbei, R. J., Zenios, S. A., 1995. Robust optimization of large-scale systems. Operations research 43 (2), 264–281.

Okumoto, K., Goel, A. L., 1980. Optimum release time for software systems based on reliability and cost criteria. Journal of Systems and Software 1, 315–318.

Paixao, M., Souza, J. T., 2013. A scenario-based robust model for the next release problem. In: Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference. ACM, pp. 1469–1476.

Peng, R., Li, Y., Zhang, J., Li, X., 2013. A risk-reduction approach for optimal software release time determination with the delay incurred cost. International Journal of Systems Science (ahead-of-print), 1–10.

R-Project, 2014. `http://www.r-project.org/`, acessado em: Junho de 2014.

Ramirez, A. J., Fredericks, E. M., Jensen, A. C., Cheng, B. H. C., 2012. Automatically relaxing a goal model to cope with uncertainty. In: Search Based Software Engineering. Springer, pp. 198–212.

Roy, B., 2008. Robustness in operations research and decision aiding. In: Flexibility and Robustness in Scheduling. Wiley, pp. 35–52.

Roy, B., 2010. To better respond to the robustness concern in decision aiding: four proposals based on a twofold observation. In: Handbook of Multicriteria Analysis. Springer, pp. 3–24.

Souza, J. T., Maia, C., Ferreira, T., Carmo, R., Brasil, M., 2011. An ant colony optimization approach to the software release planning with dependent requirements. Search Based Software Engineering, 142–157.

Taguchi, G., 1986. Introduction to quality engineering: designing quality into products and processes.

Van den Akker, J. M., Brinkkemper, S., Diepen, G., Versendaal, J., 2005. Determination of the next release of a software product: an approach using integer linear programming. In: Proceeding of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2005). pp. 119–124.

Wang, J., Rong, G., 2009. Robust optimization model for crude oil scheduling under uncertainty. Industrial & Engineering Chemistry Research 49 (4), 1737–1748.

Xuan, J., Jiang, H., Ren, Z., Luo, Z., sept.-oct. 2012. Solving the large scale next release problem with a backbone-based multilevel algorithm. Software Engineering, IEEE Transactions on 38 (5), 1195 –1212.

Yamada, S., Narigisa, H., Osaki, S., 1984. Optimum release policies for a software system with a scheduled software delivery time. International Journal of Systems Science 15 (8), 905–914.

Yang, B., Hu, H., Jia, L., 2008. A study of uncertainty in software cost and its impact on optimal software release time. Software Engineering, IEEE Transactions on 34 (6), 813–825.

Yu, G., 1996. On the max-min 0-1 knapsack problem with robust optimization applications. Operations Research 44 (2), 407–415.

Zhang, Y., Alba, E., Durillo, J. J., Eldh, S., Harman, M., 2010. Today/future importance analysis. In: ACM Genetic and Evolutionary Computation COnference (GECCO 2010). pp. 1357–1364.

Zhang, Y., Finkelstein, A., Harman, M., 2008. Search based requirements optimisation: Existing work and challenges. Requirements Engineering: Foundation for Software Quality, 88–94.

Zhang, Y., Harman, M., Mansouri, S. A., 2007. The multi-objective next release problem. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation. ACM, pp. 1129–1137.