A Scenario-based Robust Model for the Next Release Problem

Matheus Paixão
State University of Ceará
1700 Avenida Paranjana
Fortaleza, Brazil
mhepaixao@gmail.com

Jerffeson Souza
State University of Ceará
1700 Avenida Paranjana
Fortaleza, Brazil
jerffeson.souza@uece.br

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 considered in this problem should be taken into account. In this context, this paper proposes a novel formulation to the next release problem based on scenarios and considering the robust optimization framework, which enables the production of robust solutions. In order to measure the "price of robustness", several experiments were designed and executed over artificial and real-world instances. All experimental results are consistent to show that the penalization with regard to solution quality due to robustness is relatively small, which qualifies the proposed model to be applied even in large-scale real-world software projects.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements Specifications—*Methodologies*

General Terms

Algorithms, Measurement, Reliability

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 release. Despite bringing many benefits, this development model embodies more complexity to the project management, including the problem of selecting a set of requirements to be added to the software as part of the next release, which has become known as the Next Release Problem (NRP). The NRP was first modeled as an optimization problem by Bagnall et al. [1]. In this model, each client has an importance to the organization. The goal is to select a subset of clients which will have their requirements added to the next release, so that the sum of their importance is maximized. The total cost required to satisfy the clients is subject to a predefined bound, which must be respected. A variation of this model was proposed by Akker et al. [17], where each requirement has an importance value, which estimates its revenue. The objective here is to select a subset of requirements such that the total revenue is maximal and the available capacity is respected.

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. For instance, the requirement importance could be indicated by the clients and the cost determined by the development team. In both cases, these values are obtained based on estimates, which can be significantly hard to make due to the dynamic environment in which software development occurs. In the work by Harker et al. [8], requirements are classified in six types. Among those types, five are classified as *changing* and only one as *stable*, which stresses the evolving nature of requirements. In this context, requirement's importance and cost are among those features that can change after the initial requirements engineering phase. Indeed, the high level of uncertainty related to the variables of the next release problem generates a fairly complicated context, as pointed out by Zhang, Finkelstein and Harman in [22]:

> "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."

More details about the impact caused by requirements' changes during the release development can be seen in the sensitivity analyses presented in [9]. Therefore, it seems reasonable that the uncertainties related to requirement's importance and cost should be considered when solving the next release problem through an optimization technique.

The robust optimization is an operational research framework that identifies and quantifies uncertainties in optimization problems [4]. It started to gain more visibility after [16] and [2]. This optimization design technique admits that some problem aspects are uncertain. From this assumption, it builds up models which seek robust solutions, i.e., even with noisy input data, it produces good quality solutions while still fulfilling all constraints. Robust optimization has been successfully applied in several engineering disciplines including, but not limited to, production [12], aeronautical [5], electronic [14], mechanical [13], chemical [18] and metallurgical engineering [6].

Accordingly, the robust optimization framework can aid the proposition of original optimization models to the NRP, which could deal with the uncertainties present in this problem, allowing for the production of robust solutions. However, as can be assumed, this desired robustness will necessarily be accompanied by some loss in solution quality, which needs to be contemplated. This measure of loss has been called in the robust optimization literature as the "price of robustness" [3].

Therefore, motivated by this context, this paper aims at answering the following research questions:

- RQ_1: How to model the Next Release Problem 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 proposed Next Release Problem model? In other words, how much would be lost with regard to solution quality in order to gain robustness?

Most of the works in the SBSE literature do not directly consider the uncertainties related to the input variables, which is also the case for the next release problem. However, as one of the exceptions, the work in [21] proposed a model considering the possibility of change in the requirement's importance. In this work, each requirement received an importance value called "today value" and it was assumed that this value, in some moment after the release development, would change to a certain "future value". The approach, therefore, seeks to balance the company's today and future needs by considering these two requirement values, along with the release total cost, via a multiobjective formulation.

In fact, the original and main contribution of this paper lies in the proposal of a novel formulation to the next release problem, considering the robust optimization framework, which was formally defined by Beyer et. al [4]. This new NRP model will allow the production of robust solutions by taking into account the uncertainties related to its input variables.

The remaining of this paper is organized as follows: Section 2 presents the proposed robust NRP formulation. Section 3 exhibits and examines the experiments designed to evaluate the proposed formulation. Finally, Section 4 concludes the paper and points out some future research directions.

2. A ROBUST NEXT RELEASE PROBLEM FORMULATION

Given a set of requirements $R = \{r_1, r_2, \ldots, r_N\}$, the requirement r_i importance value and 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. The decision variable is 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 in [8], the occurrence of certain events can change the requirements' importance values during the release development. Therefore, this type of uncertainty seems adequate to be quantified in a discrete and probabilistic way, using the robust optimization concept of scenarios [20]. A scenario can be defined as a set of values which represent different contexts due to the occurrence of certain events. Thus, one can formally define a set of scenarios $S = \{s_1, s_2, \ldots, s_M\}$, where each scenario is represented by $s_i \subset S | s_i = \{v_1^s, v_2^s, \ldots, v_N^s\}$, with v_i^s expressing the importance of requirement r_i in scenario s. The range a requirement's importance value can vary is discrete and depends on the set of scenarios S. In the assignment of these possible values, one should consider the probability of those events actually taking place. For each scenario s, it is defined an occurrence probability p_s, with $\sum_{i=1}^{M} p_s = 1$. Thus, the requirement's importance v_i in the proposed robust model is defined as:

$$v_i = \sum_{s=1}^{M} v_i^s p_s \tag{3}$$

The above requirement's importance can be looked at as a generalization of the importance in the basic NRP formulation. In fact, by considering a single scenario t, with a probability $p_t = 1$, 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. In this case, 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. Thus, in this paper, the uncertainty related to cost will be quantified in a deterministic and continuous fashion, as follows. Besides the requirement's cost c_i, it is defined a value $\hat{c}_i$ indicating the maximum expected cost variation. This variation is then used to generate lower and upper bounds to the cost c_i, so that $c_i - \hat{c}_i \leq c_i \leq c_i + \hat{c}_i$. For more information about uncertainties quantification strategies, see [4].

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. However, in real software development projects, different development teams have divergent estimating skills, usually related to the team's experience. To consider this assumption in order to generate a more realist model, it is defined a control parameter Γ [3], which indicates the expected level of failure in the cost estimations. This way, in the situation where the team estimates are historically 30% incorrect, in a project with 50 requirements, the control parameter would be $\Gamma = 15$, indicating that there is an expectation that 15 requirements will have real costs different from those that were originally predicted.

Using this new control parameter, the release total cost in the robust model proposed 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 proposed 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 proposed in Equation 5 will be the same as described in Equation 2. In addition, all cost variations will be taken into account when $\Gamma = N$, which carries the proposed 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 proposed robust Next Release Problem formulation is formally described as:

$$\text{maximize} \sum_{i=1}^{N} \sum_{s=1}^{M} v_i^s p_s x_i$$

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

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

R is the set of requirements

N is the number of requirements

M is the number of scenarios

p_s is the scenario s occurrence probability

v_i^s is the value of requirement r_i in scenario s

c_i is the cost 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

Hence, the above formulation is a robust optimization model for the next release problem which considers the uncertainties related to the input variables of this relevant problem which, therefore, answers the research question RQ_1.

3. EXPERIMENTAL EVALUATION

In order to answer the research question RQ_2, that is, what is the "price of robustness" for the next release problem as modeled in this paper, a set of three experiments were performed, as described next:

- **Experiment A:** The first experiment aims at evaluating the "price of robustness" when the proposed model is applied to artificial instances of varying sizes.
- **Experiment B:** In this second experiment, different approaches for setting the cost variations $\hat{c}_i$ were considered. This experiment was designed to discover any relationship between the cost variations and the "price of robustness".
- **Experiment C:** Finally, the model is applied to several realistic instances based on real-world software projects, in order to evaluate whether its behaviour is similar when using synthetic data.

The experiments are presented in sections 3.1, 3.2 and 3.3, with their respective settings, results and analyses.

In order to permit the full replication of all experiments, all artificial and real-world instances are make available at the paper supporting webpage - `http://www.larces.uece.br/~jeff/rnrp/` -, which also contains all results that have to be omitted from this paper due to space constraints.

3.1 Experiment A

As mentioned earlier, an increased level of solution robustness should naturally generate some loss in the solution quality. In this first experiment, the proposed robust model has been applied to a set of artificial instances in order to measure the "price of robustness" for the proposed formulation. The goal is to estimate how much is lost with regard to fitness value as there is an increase in robustness.

3.1.1 Settings

The artificial instance set is composed of 7 randomly generated instances. Each instance has 3 scenarios and each requirement's importance value v_i^s can assume an integer between 1 and 10. The cost c_i also varies from 1 to 10. The cost variation $\hat{c}_i$ is set to 10% of the respective cost. To ensure that it is impossible to include all requirements in the next release, the release budget is set to 70% of the sum of all requirements' costs. The instances were generated with different numbers of requirements, ranging from 50 to 200. In this paper, the artificial instance name is in the format I_S_R, where R is the number of requirements. The instance I_S_100, for example, has 100 requirements.

In the experiments, the metaheuristics Simulated Annealing and Genetic Algorithm were considered, as described next:

- **Simulated Annealing**: algorithm for solving ordinary optimization problems, based on the thermodynamics' annealing process [11].

- **Genetic Algorithm**: widely known evolutionary algorithm, already applied in many optimization problems and inspired by the Darwin's natural selection theory [10].

Each algorithm configuration was empirically obtained, after several preliminary experiments over different instances. The final configurations are described next:

Simulated Annealing. Initial and final temperatures were set to 100 and 10^{-3}, respectively. Cooling rate equals to 0.9995. At each iteration, N (number of requirements) neighbour solutions are evaluated. A neighbour solution is defined as a solution that can be produced from the original one with one requirement addition or removal.

Genetic Algorithm. Population with N individuals. The initial population is randomly generated and composed by feasible individuals. Crossover probability is set to 0.9, using one point crossover. Mutation is performed for each requirement with a $1/(10.N)$ probability, consisting of a single requirement inclusion/exclusion. Both crossover and mutation might generate invalid individuals. Therefore, a repairing method was designed, randomly removing requirements from the individual until the solution becomes feasible. The 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 10000 generations.

Since the metaheuristics are non deterministic approaches, each algorithm was executed 10 times for each instance, to obtain fitness value averages and standard deviations.

In order to measure the "price of robustness", it is considered a 'reduction factor' [3], which indicates the percentage of loss in fitness value due to robustness. Thus, assuming α_k as the fitness value average for $\Gamma = k.N$, the 'reduction factor' δ_k is calculated as follows:

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

Therefore, in order to evaluate how the control parameter Γ impacts the solution quality, each instance was solved using different robustness levels, varying Γ from 0 to N.

3.1.2 Results and Analyses

Table 1 presents the fitness values computed by the Simulated Annealing, while Table 2 presents the results produced by the Genetic Algorithm, considering different levels of robustness. As expected, as there is a gain in robustness, the fitness value decreases. For both metaheuristics, a similar behavior was observed in that regard.

In average, the GA performed better than SA. Furthermore, the GA's standard deviations were significantly smaller than those obtained from SA.

Interestingly, the decrease in fitness value is smoother then linear. For instance, with $\Gamma = 0.5N$, the loss in fitness value is only insignificantly higher when compared with the previous robustness level $\Gamma = 0.25N$, representing around 1% in average. These results indicate that a significant gain in robustness can be obtained with very little loss in solution quality. To highlight that behavior, Figure 1 presents the fitness values computed for the instance I_S_70, both by the Simulated Annealing and the Genetic Algorithm. It is noteworthy the significantly small fitness loss after Γ is set to half the number of requirements. This feature is present in both algorithms, but seems less visible in SA due to its higher fitness variation.

Table 3 presents the reduction factors computed by the Simulated Annealing and Table 4 the results found by the Genetic Algorithm. In general, the fitness value loss is considerably small. Considering all instances, it is possible to obtain a 50% robustness level ($\Gamma = 0.5N$) by losing, in average, only 2.24% and 3.88% in fitness value, for the SA and GA respectively. In the worst case ($\Gamma = N$), where the development team is expected to miss all cost estimates, the SA loses only 2.73% and the GA gives up 4.37% in solution quality, in average.

Figure 2 highlights, for instance I_S_120, the reduction factors computed by the two algorithms. For most Γ values, the GA's reduction factor is higher than SA's. In addition, as can be seen, a maximum of almost 4% reduction in solution quality is reached, even for maximum robustness levels.

The interesting results described above partially answer RQ_2 and clearly show that the proposed robust next release problem formulation can help protecting against the uncer-

Table 1: Simulated Annealing results, regarding the fitness values when increasing the solution robustness

Instance	Γ				
	0	$0.25N$	$0.5N$	$0.75N$	N
I_S_50	231.08 ± 2.84	227.69 ± 3.04	225.37 ± 4.47	223.34 ± 2.62	222.82 ± 3.70
I_S_70	297.72 ± 1.90	293.06 ± 4.30	287.74 ± 4.46	288.58 ± 3.84	288.76 ± 4.67
I_S_100	371.18 ± 4.41	365.05 ± 4.63	360.54 ± 3.82	358.01 ± 3.46	358.44 ± 3.14
I_S_120	474.34 ± 3.94	466.75 ± 5.93	461.73 ± 4.06	460.86 ± 3.28	461.43 ± 4.96
I_S_150	594.52 ± 6.60	590.07 ± 9.85	580.74 ± 7.60	579.12 ± 5.17	578.07 ± 5.80
I_S_170	666.63 ± 6.57	663.54 ± 8.67	659.98 ± 6.05	652.80 ± 6.91	652.57 ± 6.09
I_S_200	731.95 ± 9.43	725.51 ± 5.68	724.45 ± 9.13	717.19 ± 6.54	720.67 ± 6.63

Table 2: Genetic Algorithm results, regarding the fitness values when increasing the solution robustness

Instance	Γ				
	0	$0.25N$	$0.5N$	$0.75N$	N
I_S_50	256.93 ± 0.00	250.77 ± 0.42	247.11 ± 0.20	246.27 ± 0.00	246.27 ± 0.00
I_S_70	342.39 ± 0.22	333.23 ± 0.52	327.73 ± 0.68	326.34 ± 0.40	326.37 ± 0.34
I_S_100	450.20 ± 0.46	439.98 ± 0.50	434.01 ± 0.71	431.79 ± 0.20	431.63 ± 0.46
I_S_120	581.46 ± 0.40	568.39 ± 0.46	561.38 ± 0.45	558.29 ± 0.51	558.47 ± 0.49
I_S_150	740.09 ± 0.41	720.90 ± 0.58	709.85 ± 0.62	706.44 ± 0.26	705.97 ± 0.51
I_S_170	850.20 ± 0.32	828.74 ± 0.65	816.51 ± 0.39	812.51 ± 0.65	812.50 ± 0.53
I_S_200	948.62 ± 0.76	923.69 ± 0.65	910.62 ± 0.42	905.28 ± 0.68	904.48 ± 0.47

Figure 1: Fitness values comparison between SA and GA for the instance I_S_70

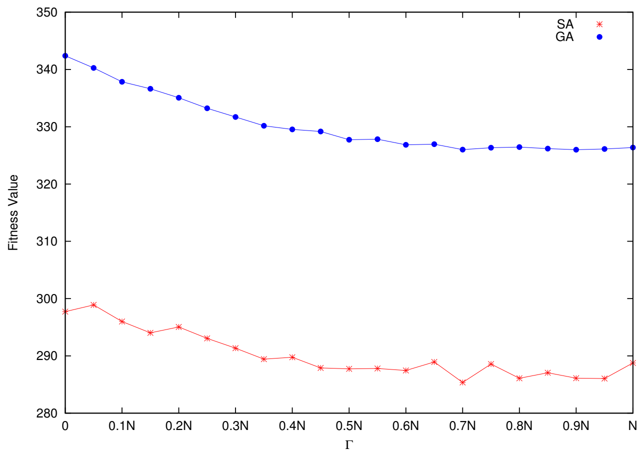


Table 3: Simulated Annealing results, regarding the reduction factor when increasing the solution robustness

Instance	Γ				
	$0.25N$	$0.5N$	$0.75N$	$0.9N$	N
I_S_50	1.46	2.47	3.35	3.47	3.57
I_S_70	1.56	3.35	3.07	3.90	3.01
I_S_100	1.65	2.87	3.55	3.47	3.43
I_S_120	1.60	2.66	2.84	2.70	2.72
I_S_150	0.75	2.32	2.59	3.02	2.77
I_S_170	0.46	1.00	2.07	1.27	2.11
I_S_200	0.88	1.02	2.02	1.53	1.54

tainties related to the input variables with little penalization with respect to solution quality.

3.2 Experiment B

As stated earlier, the cost variations $\hat{c}_i$ were set to represent 10% of the respective cost c_i. In practice, these variations may be estimated in different ways. Hence, in this experiment, the "price of robustness" is computed with different approaches for estimating the $\hat{c}_i$ values.

3.2.1 Settings

Basically the same artificial instances, described in the previous section, were used in this second experiment. The single difference relies on the strategy employed to produce the expected cost variations. This way, besides varying the percentage from 10% to 50% of the original cost, a random approach was also introduced, where each requirement's cost variation $\hat{c}_i$ is independently generated as a random number between 0 and 50% of c_i.

The Simulated Annealing and Genetic Algorithm were configured exactly as described in the previous section.

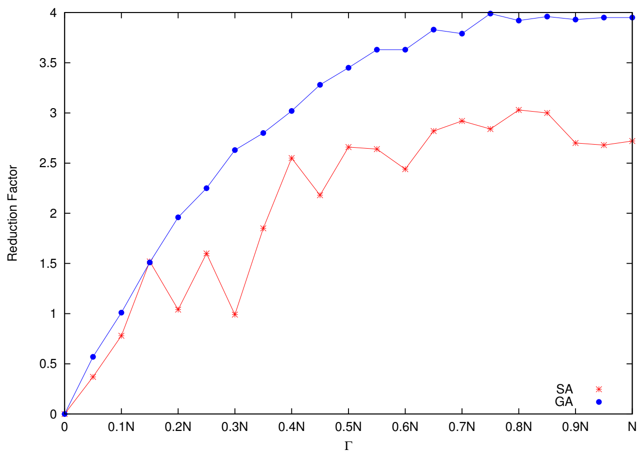
3.2.2 Results and Analyses

Figure 3 presents the particular reduction factors computed by the Simulated Annealing while Figure 4 presents the Genetic Algorithm's results, both for the instance I_S_120. As can be noticed and as expected, the higher the cost variation, the higher the reduction factor. However, this growth

Table 4: Genetic Algorithm results, regarding the reduction factor when increasing the solution robustness

Instance	Γ				
	$0.25N$	$0.5N$	$0.75N$	$0.9N$	N
I_S_50	2.40	3.82	4.15	4.20	4.15
I_S_70	2.68	4.28	4.69	4.79	4.68
I_S_100	2.27	3.60	4.09	4.17	4.13
I_S_120	2.25	3.45	3.99	3.93	3.95
I_S_150	2.59	4.09	4.55	4.60	4.61
I_S_170	2.52	3.96	4.43	4.44	4.43
I_S_200	2.63	4.01	4.57	4.68	4.65

Figure 2: Reduction factor comparison between SA an GA for the instance I_S_120



is not linear and even with significantly high variations, it is still possible to reach significant robustness levels with little loss in fitness value. As an illustrative example, considering the expected cost variation to be half of the requirement's original cost ($\hat{c}_i = 50\%$ of c_i), a 100% robustness level can be achievable by sacrificing only around 17% in fitness value. Still according to the results, a variation increase tends to close the gap between the SA and GA reduction factors, with SA losing more fitness value than GA in some cases.

In Figure 5 are presented the reduction factors computed by the GA with different variation values for the instance I_S_170. Comparing with instance I_S_120, the results are clearly similar. For others instances, the results are also very much alike, but due to space constraints, they have been omitted from this paper.

Thereby, the "price of robustness" of the proposed model remains interestingly low even when considering different strategies for generating the cost variations.

3.3 Experiment C

This experiment aims at evaluating the "price of robustness" of the proposed robust model in real-world instances, which are described next.

3.3.1 Settings

The real-world instances used in this experiment were adapted from the work by Xuan et al. [19]. In his paper, Xuan extracted the instances to the next release problem

Figure 3: Simulated Annealing price of robustness with different cost variation approaches for the instance I_S_120

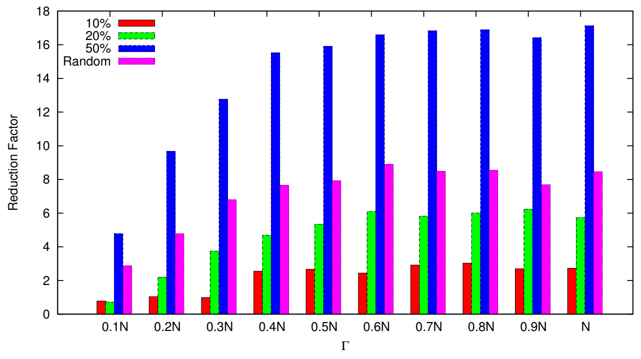
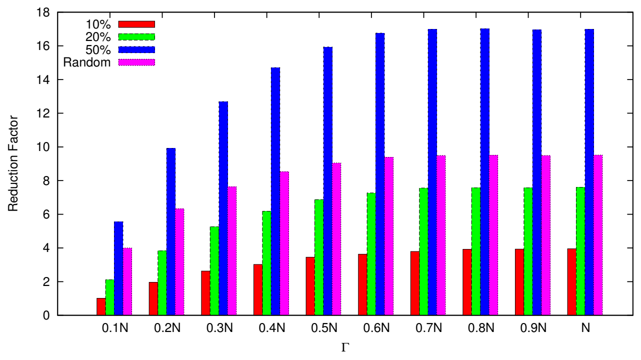


Figure 4: Genetic Algorithm price of robustness with different cost variation approaches for the instance I_S_120



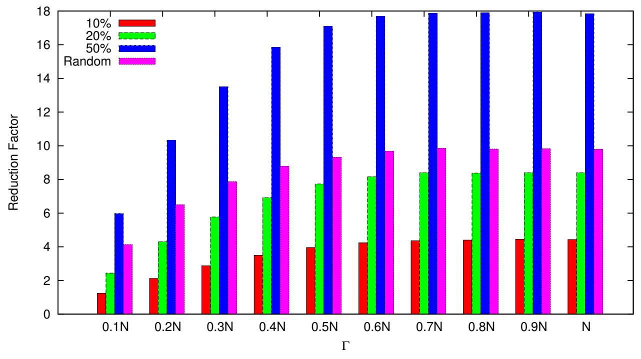
from bug repositories of three big open source projects, including Eclipse (a java integrated development environment) [7] and Mozilla (a set of web applications) [15].

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. The requirement's importance value is then calculated as the number of users that commented on that particular bug report. In addition, the bug severity is mapped to the requirement's cost. Both the requirement's importance and cost are normalized to fall into the 1 to 10 interval. In all cases, each instance was considered to have only one scenario t (with a probability $p_t = 1$).

Three instances, composed by the most important requirements, were extracted from each bug repository. The instances contain 50, 120 and 200 requirements, respectively. The real-world instance names are in the format I_Re_P_R, where P represents the project (E for Eclipse and M for Mozilla) and R is the number of requirements. The instance I_Re_E_120, for example, was generated from the eclipse bug repository and has 120 requirements.

Since the Genetic Algorithm has systematically produced better solutions in previous experiments both regarding fitness values and standard deviation, only this metaheuristic

Figure 5: Genetic Algorithm price of robustness with different cost variation approaches for the instance I_S_170



was considered in this experiment. Its configuration is the same as described previously.

3.3.2 Results and Analyses

Table 5 presents the results for fitness value found by the Genetic Algorithm for each real-world instance, with the requirement's cost variation set to 10% of the original cost. As can be seen, the model behaves nearly the same for artificial and real-world instances, which helps validating all results obtained in Experiment A. Once again, for $\Gamma = 0.5N$, the fitness value decrease becomes minimum when compared with previous robustness levels, even for the instances with 200 requirements.

Table 5: Genetic Algorithm fitness values results for the realistic instance set and standard deviation set to 10% of the cost

Instance	Γ				
	0	$0.25N$	$0.5N$	$0.75N$	N
I_Re_E_50	163.36 ± 0.82	159.09 ± 0.00	155.45 ± 0.00	155.18 ± 0.82	155.45 ± 0.00
I_Re_E_120	339.82 ± 0.68	332.09 ± 1.08	323.45 ± 0.68	322.27 ± 0.45	321.91 ± 0.27
I_Re_E_200	513.55 ± 1.25	500.18 ± 1.21	487.27 ± 1.00	484.73 ± 0.98	484.36 ± 1.67
I_Re_M_50	195.79 ± 0.98	192.14 ± 0.00	187.43 ± 0.65	186.71 ± 0.57	186.64 ± 0.46
I_Re_M_120	361.71 ± 1.40	354.57 ± 1.73	347.21 ± 2.01	344.00 ± 1.16	344.00 ± 1.51
I_Re_M_200	497.21 ± 0.50	486.07 ± 1.02	475.64 ± 0.50	472.14 ± 0.55	472.36 ± 0.85

Regarding reductions factor, Table 6 presents these results for each real-world instance. As mentioned earlier, even in the large real-world instances, the fitness value reduction after $\Gamma = 0.5N$ is very small. As an example, the results over real-world instances show that it is possible to reach a 100% robustness level losing only 5.12% of the fitness value, in average.

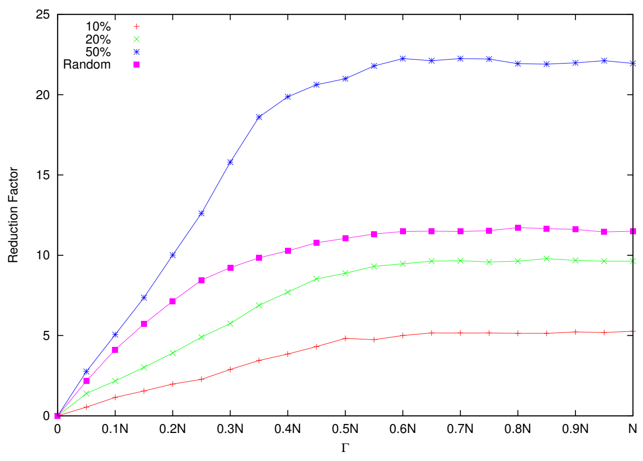
Figure 6 presents the reduction factor computed by the Genetic Algorithm for the instance I_Re_E_120 with different cost variation approaches. In Figure 7 are presented the same results for instance I_Re_E_200.

The reduction factors of both real-world instances are very

Table 6: Reduction factors computed by the Genetic Algorithm for the realistic instance set and standard deviation set to 10% of the cost

Instance	Γ				
	0.25	$0.5N$	$0.75N$	$0.9N$	N
I_Re_E_50	2.62	4.84	5.01	5.23	4.84
I_Re_E_120	2.27	4.82	5.16	5.22	5.27
I_Re_E_200	2.60	5.12	5.61	5.52	5.68
I_Re_M_50	1.86	4.27	4.63	4.63	4.67
I_Re_M_120	2.27	4.82	5.16	5.22	5.27
I_Re_M_200	2.24	4.34	5.04	5.09	5.00

Figure 6: Genetic Algorithm price of robustness with different cost variation approaches for the real-world instance I_Re_E_120



similar, akin to the behavior found in the artificial instances. For the Mozilla real-world instance I_Re_M_120, the results are also very consistent, as can be seen in Figure 8.

In conclusion, all results reported in this experiment are consistent to show that the behavior of the proposed robust formulation, regarding its "price of robustness", are similar to those found over artificial instances. That is, the penalization due to robustness is very small, which qualifies the proposed approach to be considered in real software projects.

Finally, the results obtained in the three experiments help answering RQ_2, pointing out to the ability of the model to produce robust solution with significantly small loss in quality.

3.4 Threats to Validity

The aspects that could affect the validity of the experimental results described in this paper are:

1. Relatively small number, size and diversity of instances: Even though the paper considers 7 artificial and 6 real-world instances, a higher number of instances would, clearly, produce more reliable results.

2. Parameterization of algorithms: In the experiments, the values for the algorithms' parameters were empirically calculated through a simplified experimental process. A more comprehensive parameterization process would be beneficial to the validity of the reported results.

Figure 7: Genetic Algorithm price of robustness with different cost variation approaches for the real-world instance I_Re_E_200

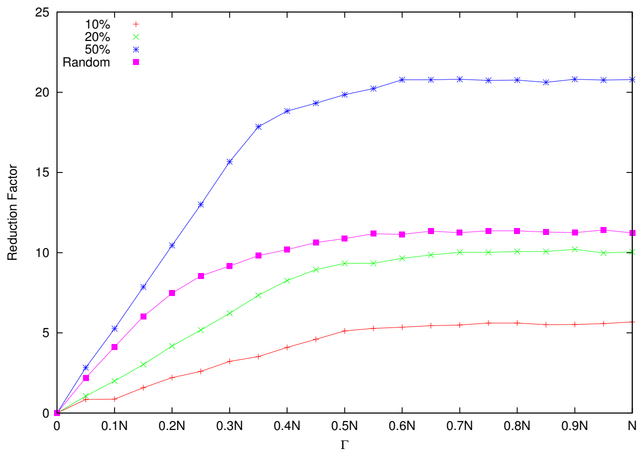
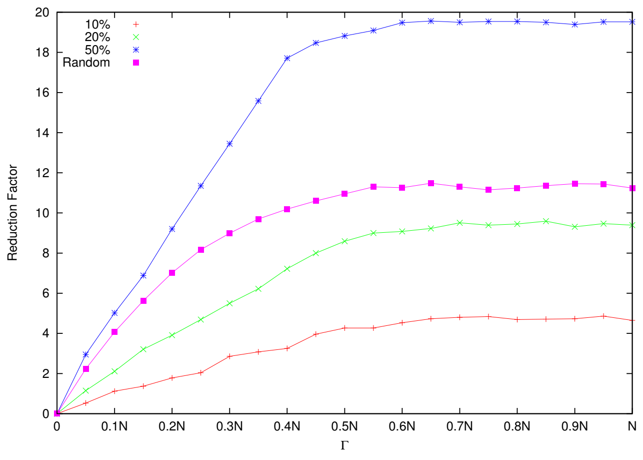


Figure 8: Genetic Algorithm price of robustness with different cost variation approaches for the real-world instance I_Re_M_120



4. 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.

This paper proposed an original robust formulation, based on scenarios, for the next release problem, which takes into account the uncertainties present in the input variables. In the proposed 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 proposed robust model was applied to both artificial and real-world instances extracted from bug repositories of two large open source software projects (Eclipse and Mozilla). In order to evaluate the "price of robustness" for the proposed model, three sets of replicable experiments were designed, executed and analysed.

The first experiment was applied over artificial instances, using a cost variation set to 10% of the original requirement's cost. As a result, it was demonstrated that the gain in robustness was obtained with a loss in fitness value consistently and significantly small for all instances. Furthermore, results showed a nearly constant decrease in fitness value when the robustness control parameter was calibrated to at least half the number of requirements. In the second experiment, the "price of robustness" was computed for the artificial instances with different cost variation approaches. As expected, 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. The last experiment applied the robust model to a set of real-world instances. The results were nearly the same as for the artificial 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.

Since this is the first work to employ the robust optimization framework, in the search based software engineering field, a natural future research direction points out to the application of this framework to other requirements engineering problems, as well as to other software engineering problems subject to uncertainties. In addition, specifically related to the next release problem, other experiments could be proposed to evaluate the "price of robustness" under different numbers of scenarios and different release budgets. 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.

5. REFERENCES

[1] A. Bagnall, V. Rayward-Smith, and I. Whittley. The next release problem. *Information and Software Technology*, 43(14):883–890, 2001.

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

[3] D. Bertsimas and M. Sim. The price of robustness. *Operations research*, 52(1):35–53, 2004.

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

[5] X. Du, Y. Wang, and W. Chen. Methods for robust multidisciplinary design. *AIAA*, 1785:1–10, 2000.

[6] G. Dulikravich and I. Egorov-Yegorov. 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*, pages 25–28, 2005.

[7] Eclipse. http://www.eclipse.org/, January, 2013.

[8] S. Harker, K. Eason, and J. Dobson. 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*, pages 266–272. IEEE, 1993.

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

[10] H. Holland John. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. *USA: University of Michigan*, 1975.

[11] S. Kirkpatrick, M. Vecchi, et al. Optimization by simmulated annealing. *science*, 220(4598):671–680, 1983.

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

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

[14] S. Malcolm and S. Zenios. Robust optimization for power systems capacity expansion under uncertainty. *Journal of the operational research society*, pages 1040–1049, 1994.

[15] Mozilla. http://www.mozilla.org/, January, 2013.

[16] J. Mulvey, R. Vanderbei, and S. Zenios. Robust optimization of large-scale systems. *Operations research*, 43(2):264–281, 1995.

[17] J. Van den Akker, S. Brinkkemper, G. Diepen, and J. Versendaal. 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)*, pages 119–124, 2005.

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

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

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

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

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