A Recoverable Robust Approach for the Next Release Problem

Matheus Henrique Esteves Paixao and Jerffeson Teixeira de Souza

Optimization in Software Engineering Group, State University of Ceará,
1700 Avenida Paranjana, 60.714-903, Fortaleza, Brazil
{matheus.paixao,jerffeson.souza}@uece.br

Abstract. Selecting a set of requirements to be included in the next software release, which has become to be known as the Next Release Problem, is an important issue in the iterative and incremental software development model. Since software development is performed under a dynamic environment, some requirements aspects, like importance and effort cost values, are highly subject to uncertainties, which should be taken into account when solving this problem through a search technique. Current robust approaches for dealing with these uncertainties are very conservative, since they perform the selection of the requirements considering all possible uncertainties realizations. Thereby, this paper presents an evolution of this robust model, exploiting the recoverable robust optimization framework, which is capable of producing recoverable solutions for the Next Release Problem. Several experiments were performed over synthetic and real-world instances, with all results showing that the recovery strategy handles well the conservatism and adds more quality to the robust solutions.

Keywords: Recoverable robustness, next release problem, search based software engineering.

1 Introduction

In an iterative and incremental software development model, the selection of a set of requirements to be added to the next software release is known as the Next Release Problem (NRP). In such problem, each requirement is represented by an importance, which refers to the value aggregated to the client when the requirement is included in the next release, and a cost value, related to the effort required to implement the requirement. In that context, the next release problem consists in selecting a subset of requirements in such a way that the sum of their importance is maximized, with the cost needed to implement those requirements respecting the release's available budget [1].

Therefore, in order to employ a search technique to solve the NRP, it is necessary to obtain the importance and cost values of each requirement. The importance values could be indicated by the client and the costs determined by

the development team. In both cases, these values are acquired through estimations, which can be hard to make due to the dynamic environment in which software development takes place.

In fact, changes in the requirements characteristics can be fairly dangerous for the whole system development. As can be perceived in the sensitivity analysis by Harman et. al [2], small changes in the requirements features may have a high impact on the requirements selection. Thereby, it is paramount to consider the uncertainties related to the requirement's importance and cost when solving the NRP through optimization techniques [3].

The robust optimization is an operational research framework that identifies and quantifies uncertainties in generic optimization problems [4]. It started to gain more visibility after the first works in [5] and [6]. The robust optimization framework consists of two steps. The first step seeks to identify and quantify all uncertainties related to the problem. Using these information about the uncertainties, the second step consists of building a model which seeks robust solutions, that is, solutions which are feasible for every realization of the uncertainties.

The robust optimization framework has been previously applied to the NRP [7]. In that paper, it was considered that the requirements importance could assume different values due to the occurrence of certain events. The requirement's importance was then calculated by taking into account all these possible values and the respective occurrence probabilities. In order to examine the cost uncertainties, it was defined a *robustness control parameter*, which indicated the expected level of failure in the team cost estimations. This parameter actually stipulates how many requirements will have their real cost higher than the original estimate. Since there is no way to know in advance which requirements will have different real cost values, this approach guaranteed that, even if the team missed the cost of the most expensive requirements, the produced solution would still be feasible. Experiments showed that the penalization with regard to solution quality due to robustness is relatively small. This approach, however, can still be considered conservative, because it assumes that, invariably, some requirements are wrongfully estimated, and their real costs will be as high as possible. Since the requirements selection is made considering the worst possible cost uncertainty values, any different requirements' cost realization will cause a waste of resources.

In the work by Liebchen et. al [8], seeking to handle the conservatism of the "classical" robust optimization, named *strict robustness*, it was introduced a new concept of robustness, the *recoverable robustness*. A recoverable robust solution is the one which, for a limited set of uncertainties realizations, can be made feasible or *recovered*, by a limited effort [9]. Accordingly, the recoverable robustness framework can improve the model in [7], producing a more "realist" requirements selection for the next release problem.

In this context, this papers aims at answering the following research questions:

- RQ_1: How to model the Next Release Problem considering the recoverable robustness framework?

– RQ_2: How much is gained in solution quality with the new recoverable robust model when compared with the strict robust model?

Consequently, the original and main contribution of this paper relates to the tackling of the uncertainties in the Next Release Problem using the recoverable robustness framework.

The remaining of this paper is organized as follows: in Section 2 the recoverable robustness framework is explained in more details, while in Section 3, the recoverable robust NRP model is evolved. Section 4 exhibits and examines the experiments designed to evaluate the proposed formulation. Finally, Section 5 concludes the paper and points out some future research directions.

2 The Recoverable Robustness Framework

The recoverable robustness framework is composed of three main steps, as discussed in [9] and presented next:

1. Identify the uncertainties related to the problem (step Ω)
2. Develop a model which produces recoverable solutions (step M)
3. Define a set of possible recovery algorithms (step A)

These three steps are intrinsically connected. The model M is developed by considering the uncertainties identified in Ω. The recovery algorithms A recover solutions generated by M. Therefore, the triple (Ω, M, A) does not only states that recovery is possible, but explicitly demonstrates how this recovery can be performed.

For steps Ω and M, different modeling techniques can be employed from the strict robustness literature, however, step A is a little trickier and can be formalized as follows:

Definition 1. *Let F_ω be the set of feasible solutions under uncertainties realization ω. A recovery algorithm A is the one which for every solution x under realization ω, it generates a feasible solution, i.e., $A(x, \omega) \in F_\omega$*

Therefore, the output of a recoverable robust problem is genuinely a pair (x, A), called *precaution*, which is composed of the solution x and a recovery algorithm A. In the case where some uncertainty realization makes the solution x unfeasible, it can be recovered by applying the algorithm A.

A generic recoverable robust problem can then be stated as follows:

$$\text{optimize: } f(x)$$
$$\text{subject to: } \forall \omega \subseteq \Omega : A(x, \omega) \in F_\omega$$

In a practical context, one way to limit the effort needed to recover a solution is to consider the number of changes to that solution during the recovery process. Employing this bound in the recovery phase, the concept of *k-recoverable robustness* was proposed by Büsing et. al [10][11]. This new approach is an attempt to control the recovery phase. The recovery control parameter k acts as a

limit to the recovery algorithm, i.e., $A(x, \omega, k)$ must recover the solution making at most k changes to the original solution. A k-recoverable robust problem can be stated as:

$$\text{optimize: } f(x)$$
$$\text{subject to: } \forall \omega \subseteq \Omega : A(x, \omega, k) \in F_\omega$$

In the next section, this framework will be employed to the Next Release Problem.

3 Evolving a Recoverable Robust Next Release Problem Formulation

Given a set of requirements $R = \{r_1, r_2, \ldots, r_N\}$, the requirement r_i importance and cost values are represented by v_i and c_i, respectively. A classic formulation of the Next Release Problem 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 solution X can be represented as a vector $\{x_1, x_2, \ldots, x_N\}$ where $x_i = 1$ indicates that the requirement r_i is included in the next release, $x_i = 0$ otherwise.

As discussed earlier, the occurrence of certain events can change some requirements' characteristics, including the importance value. Thereby, this type of uncertainty seems adequate to be treated in a discrete and probabilistic way, using the concept of scenarios, as in [7].

A scenario can be defined as a set of importance values due to the occurrence of certain event. Thus, it is defined a set of scenarios $S = \{s_1, s_2, \ldots, s_M\}$, where each scenario is represented by $s \subset S | s = \{v_1^s, v_2^s, \ldots, v_N^s\}$, with v_i^s indicating the importance of requirement r_i under scenario s. The range a requirement importance value can vary is discrete and depends on the set of scenarios. In the assignment of these importance values, the probability of each event taking place can be considered. Thus, for each scenario s it is defined an occurrence probability p_s, with $\sum_{s=1}^{M} p_s = 1$. The requirement importance value v_i is then defined as:

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

The uncertainty related to the requirement's cost is different. Usually, the cost variation is different from one requirement to another and this difference may not be discrete. Thus, it is unlikely that one would be able to raise a set of

scenarios based on possible events. Thereby, the requirement's cost uncertainty will be quantified in a continuous and deterministic way. Let c_i be the estimated requirement cost. It is defined a value $\hat{c}_i$, which indicates the maximum expected cost variation. These values are used to generate lower and upper bounds to the real requirement cost $\bar{c}_i$, so that $c_i - \hat{c}_i \leq \bar{c}_i \leq c_i + \hat{c}_i$.

An alternative robust formulation to the release total cost is presented next:

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

In the above case, besides the requirements' costs, it is also considered the cost variation of all requirements selected to the next release. This approach guarantees that, even in the worst possible case, i.e., when all requirements will cost their upper bounds $(c_i + \hat{c}_i)$, the budget constraint will be satisfied. Clearly, this model is very conservative since it assumes that the development team will miss all cost estimates by the maximum amount.

In order to minimize this conservatism, the release cost considered in the NRP robust formulation proposed in [7] can be seen below:

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

In this model, a control parameter Γ is defined, indicating the maximum number of requirements that may have real costs different from the original estimates. For instance, in a situation where the development team estimates are historically 20% incorrect, in a project with 50 requirements, the control parameter would be $\Gamma = 10$. In order to calculate the release total cost, the variation of all requirements will no longer be considered, but only the variation of the Γ requirements in which the cost variation sum is maximum, represented by the subset $W \subseteq R$. This model guarantees that, even if the team misses the costs of the requirements with highest variations, the generated solution would still be feasible.

Both robust approaches assume that some requirements cost estimates are wrong, and the real cost of these requirements will be the highest possible. From these assumptions, these models find solutions to all possible uncertainties realizations, which characterize them as strict robust approaches. As stated earlier, these kind of robust solutions are still conservative and may waste a considerably amount of resources.

This paper evolves the formulation in [7] by proposing a k-recoverable robust model to the NRP. This improved model would be able to find non-conservative solutions when the requirements costs are correctly estimated, i.e, solutions as close as possible to the classic NRP model. At the same time, if some cost uncertainty realization makes the solution unfeasible due to budget constraints, the solution would be recovered by removing at most k requirements. In order to model the release total cost so that the produced solution has the aforementioned characteristics, the following functions are defined:

$$basicCost(X) = \sum_{i=1}^{N} c_i x_i \tag{6}$$

$$uncertaintyLoss(X, \Gamma) = max_{W \subseteq R, |W| \leq \Gamma} \sum_{i \in W} \hat{c}_i x_i \tag{7}$$

$$recoveryGain(X, k) = min_{Y \subseteq R, |Y| = k} \sum_{i \in Y} c_i x_i \tag{8}$$

The function $basicCost()$ returns the classic NRP release total cost, i.e, the cost sum of all requirements selected to the next release. The function $uncertaintyLoss(\Gamma)$ represents the *robustness level* [7] and calculates the sum of the Γ maximum requirements cost variations. Finally, $recoveryGain(k)$ is controlled by the *recovery parameter* k and can be considered as the *recovery level*. It represents the sum of the k minimum requirements cost estimates. For a certain solution, this recovery level indicates the minimum cost that can be removed from the release in case of recovery.

Thus, in the recoverable robust NRP model proposed in this paper, the release total cost is computed as:

$$max(basicCost(X), basicCost(X) + uncertaintyLoss(X, \Gamma) - recoveryGain(X, k)) \tag{9}$$

The robustness parameter Γ indicates how many requirements could have been wrongfully estimated while the recovery parameter k denotes the number of requirements that can be removed during the recovery operation. Depending on the robustness and recovery paramaters configuration, the value of $recoveryGain(k)$ may be bigger than the $uncertaintyLoss(\Gamma)$ value, which would cause a robust release cost smaller than the classic release cost. Such solution would be necessary to recover even if all requirements costs were correctly estimated, which does not make sense in a practical next release planning environment. In order to avoid this situation, the above release cost formulation guarantees that a solution will not have its total cost lower than the classic NRP model.

Interestingly, this recoverable robust formulation for the release cost can be seen as a generalization of the strict robust and classic models. When considering the recovery parameter $k = 0$, meaning that recovery is not allowed, we get the same robust formulation present in [7]. Using $k = 0$ and $\Gamma = N$, there is no recovery and all cost variations will be considered, characterizing the conservative case shown in Equation (4). Finally, considering $k = \Gamma = 0$, the model falls back into the classic NRP model in Equation (2).

Accordingly to the importance and cost uncertainties quantification (step ω) presented above, it is presented next the proposed formal model that seeks k-recoverable solutions to the Next Release Problem (step M), partially answering the research question RQ_1.

The proposed formulation generates a feasible solution to the NRP, guaranteeing that, even if some cost uncertainty realization makes this solution unfeasible due to budget constraints, by removing at most k requirements, the solution

will be recovered to be once again feasible. To perform this recovery, the proposed algorithm (Algorithm 1) can recover a solution by losing the minimum importance amount.

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

$$\text{subject to: } max(basicCost(),$$
$$basicCost() + uncertaintyLoss(\Gamma) - recoveryGain(k)) \leq b$$

$$\text{where, } basicCost() = \sum_{i=1}^{N} c_i x_i$$

$$uncertaintyLoss(\Gamma) = max_{W \subseteq R, |W| \leq \Gamma} \sum_{i \in W} \hat{c}_i x_i$$

$$recoveryGain(k) = min_{Y \subseteq R, |Y| = k} \sum_{i \in Y} c_i x_i$$

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

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 cost variation of r_i

Γ is the robustness control parameter

k is the recovery control parameter

b is the release budget

Algorithm 1 Minimum Value Loss Recovery Algorithm

```
while release is unfeasible do
  remove the less valuable requirement
  if new release cost ≤ b then
    return recovered solution
  end if
end while
```

This recovery algorithm is straightforward and consists in removing less valuable requirements until the solution becomes feasible once again. As the model guarantees that any k requirements can be removed, the model can recover the release by losing the minimum importance value as possible.

This proposed recovery algorithm represents the last step (step A) in the application of the recoverable robustness framework to the NRP, fully answering the research question RQ_1.

4 Experimental Evaluation

In order to ensure robustness to a solution, some loss regarding quality is inevitable. This measure of loss has become to be known in the literature as the "price of robustness" [12]. As mentioned earlier, strict robust models are conservative approaches that usually waste a significant amount of resources. Under specific conditions, these models may present a relatively high "price of robustness". The research question RQ_2 is related to how much is gained in solution quality when we employ the proposed recoverable robustness model instead of the strict model presented in [7].

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

4.1 Settings

Experiments were performed over 7 synthetic and 14 real-world instances. In the synthetic instance set, each instance has 3 scenarios in which the requirements importance values v_i^s can assume integer values between 1 and 10. The effort cost also varies from 1 to 10. The instances were generated with different numbers of requirements, from 50 to 200. In this paper, the synthetic instance name is in the format I_S_R, where R is the number of requirements. The instance I_S_120, for example, is a synthetic one and has 120 requirements. The real-world instances were adapted from the work by Xuan et. al [13]. These instances were extracted from bug repositories of two big open source projects, Eclipse (a java integrated development environment) [14] and Mozilla (a set of web applications) [15]. Each bug is considered a requirement. Its importance is calculated by the number of users that commented on that bug report. The bug severity is mapped to the requirement's cost. Both importance and cost values were normalized to the 1 to 10 interval. Seven instances were extracted from each bug repository. The instances are formed by the most important requirements and contain from 50 to 200 requirements. The real-world instance name is in the format I_Re_P_R, where P represents the project (E for Eclipse and M for Mozilla) and R indicates the number of requirements. The instance I_Re_M_200, for example, was extracted from the Mozilla bug repository and has 200 requirements.

For all instances, synthetic and real, the cost variation $\hat{c}_i$ of each requirement was configured as a random number between 0 and 50% of the respective requirement cost c_i. To make the selection problem more complex, we ensure that it is not possible to select all requirements to the next release by setting the release available budget to 70% of the sum of all requirements' costs.

Different configurations of the robustness parameter Γ and the recovery parameter k were evaluated. For each instance, the parameter Γ was set to

$\Gamma = \{0, 0.2N, 0.4N, \ldots, N\}$, where N is the number of requirements, while the recovery parameter k was set to $k = \{0, 0.1N, 0.2N, \ldots, N\}$.

In [7], both Genetic Algorithm and Simulated Annealing were employed to solve the NRP strict robust model. Since the Genetic Algorithm [16] achieved better results overall, in this paper we apply only this metaheuristic. In our GA, the population is composed by N (number of requirements) individuals. The initial population is randomly generated. All individuals that have a release cost bigger than the available budget are discarded and new individuals are randomly generated. This process is repeated until the initial population is composed only by feasible individuals. Crossover probability is set to 0.9, using one point crossover. Mutation is performed for each requirement with a $1/(10N)$ probability. It consists of a single requirement inclusion/exclusion. Both crossover and mutation operators might generate invalid individuals. Therefore, a repairing method was designed, randomly removing requirements from the individual until the solution becomes feasible. The implementation employs the elitism strategy, 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. These parameters were all set based on the results of experiments specifically designed to this purpose.

All results, including fitness value averages and standard deviations, were obtained from 10 executions of the algorithm for each instance.

As mentioned early, ensuring robustness to a solution causes some loss regarding quality. In order to measure this "price of robustness", a 'reduction metric' is introduced, which indicates the percentage of loss in fitness value in a certain configuration of the parameters k and Γ, when compared with the classic NRP model ($k = \Gamma = 0$). Thus, assuming α_j^i as the fitness value average for $k = i.N$ and $\Gamma = j.N$, the 'reduction metric' δ_j^i is calculated as follows:

$$\delta_j^i = 100 \times (1 - \frac{\alpha_j^i}{\alpha_0^0}) \tag{10}$$

4.2 Results and Analysis

Table 1 presents the fitness values computed by the Genetic Algorithm for some of the synthetic instances. The $\Gamma = 0$ rows represent the classic NRP while the $k = 0$ column presents the results for the strict robust model in [7].

Considering the results for the strict robust model in [7], as the robustness parameter Γ increases, there is a loss in solution quality. By allowing recovery, the proposed recoverable model improves the solutions. As the recovery parameter k increases, there is a gain in solution quality when compared with the strict model fitness values. For a recovery level of 20%, for example, the generated solutions are in average 2.9% better in terms of quality. Also, in average, a 10% recovery level represents a 1.5% improvement in fitness value, when comparing the proposed recoverable model with the strict model in [7].

As the recovery level grows, the fitness values converge to the classic NRP model. The recovery stability point is around 40% for most instances. Thereby,

Table 1. Fitness values results for some of the synthetic instances, regarding different values for both robustness and recovery parameters

Instance	Γ	k					
		0	$0.2N$	$0.3N$	$0.5N$	$0.7N$	N
I_S_50	0	256.83 ± 0.30	256.83 ± 0.30	256.83 ± 0.30	256.83 ± 0.30	256.83 ± 0.30	256.83 ± 0.30
	$0.2N$	243.69 ± 0.00	251.78 ± 0.43	256.93 ± 0.00	256.81 ± 0.36	256.93 ± 0.00	256.93 ± 0.00
	$0.4N$	237.31 ± 0.55	246.08 ± 0.00	253.61 ± 0.41	256.83 ± 0.30	256.93 ± 0.00	256.93 ± 0.00
	$0.6N$	233.93 ± 0.41	242.33 ± 0.49	250.34 ± 0.53	256.69 ± 0.48	256.93 ± 0.00	256.93 ± 0.00
	$0.8N$	233.03 ± 0.25	241.13 ± 0.30	249.05 ± 0.00	256.93 ± 0.00	256.93 ± 0.00	256.93 ± 0.00
	N	233.08 ± 0.26	241.25 ± 0.07	249.05 ± 0.00	256.93 ± 0.00	256.93 ± 0.00	256.93 ± 0.00
I_S_120	0	581.21 ± 0.38	581.21 ± 0.38	581.21 ± 0.38	581.21 ± 0.38	581.21 ± 0.38	581.21 ± 0.38
	$0.2N$	544.49 ± 0.70	563.05 ± 0.75	578.32 ± 0.49	581.37 ± 0.31	581.34 ± 0.46	581.53 ± 0.25
	$0.4N$	531.82 ± 0.62	549.96 ± 0.28	564.25 ± 0.66	581.37 ± 0.34	581.24 ± 0.58	581.45 ± 0.19
	$0.6N$	526.27 ± 0.70	544.28 ± 0.53	559.02 ± 0.41	581.45 ± 0.44	581.08 ± 0.76	581.51 ± 0.32
	$0.8N$	525.98 ± 0.49	543.27 ± 0.56	557.17 ± 0.61	581.54 ± 0.29	581.32 ± 0.48	581.15 ± 0.60
	N	525.58 ± 0.58	543.51 ± 0.43	557.70 ± 0.31	581.43 ± 0.53	581.55 ± 0.43	581.46 ± 0.40
I_S_200	0	946.62 ± 1.24	946.62 ± 1.24	946.62 ± 1.24	946.62 ± 1.24	946.62 ± 1.24	946.62 ± 1.24
	$0.2N$	886.95 ± 0.69	923.70 ± 1.12	947.08 ± 1.05	947.37 ± 0.76	946.96 ± 1.04	947.92 ± 0.58
	$0.4N$	861.32 ± 1.51	898.55 ± 1.13	928.37 ± 1.15	947.55 ± 1.12	946.56 ± 0.96	946.95 ± 0.80
	$0.6N$	851.83 ± 0.97	887.56 ± 1.44	917.67 ± 0.68	947.54 ± 0.86	947.36 ± 1.33	947.61 ± 1.05
	$0.8N$	850.16 ± 1.17	885.93 ± 0.90	915.04 ± 0.93	947.72 ± 1.38	947.08 ± 1.17	947.84 ± 0.77
	N	850.64 ± 1.01	885.93 ± 1.29	914.99 ± 1.70	946.94 ± 0.79	946.99 ± 1.18	946.77 ± 1.27

a lower robustness parameter (a small quantity of wrongfully estimated requirements) reaches the classic NRP model with a lower recovery parameter.

It is also noteworthy the considerably low standard deviation presented by the Genetic Algorithm. For most experimental results shown in the table, the standard deviation is less than 1, reaching no more than 1.7.

Figure 1 presents the fitness results for some of the instances that were not shown in Table 1. The solutions clearly converge to the classic NRP as k increases, as stated above. Since $k = 0$ represents the fitness value for the strict model in [7], for all robustness parameters values, every recovery level increase adds quality to the solution.

Fig. 1. Fitness value results for synthetic instances I_S_70 and I_S_170

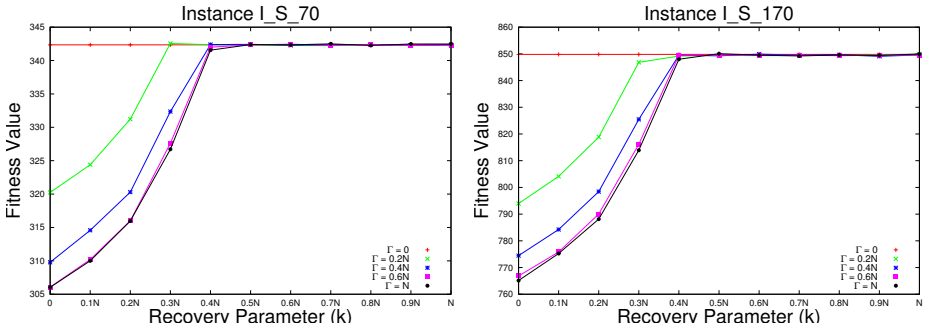

Table 2 presents the reduction factors for some of the synthetic instances. As it has been mentioned, this value indicates the percentage of fitness value loss of some robust solution when compared to the classic NRP ($\Gamma = k = 0$).

Table 2. Reduction factor results for some of the synthetic instances, regarding different values for both robustness and recovery parameters

Instance	Γ	k					
		0	$0.2N$	$0.3N$	$0.5N$	$0.7N$	N
I_S_50	$0.2N$	5.12	1.97	0.04	0.01	0.04	0.04
	$0.4N$	7.60	4.19	1.25	0.00	0.04	0.04
	$0.6N$	8.92	5.65	2.52	0.06	0.04	0.04
	$0.8N$	9.26	6.11	3.03	0.04	0.04	0.04
	N	9.25	6.07	3.03	0.04	0.04	0.04
I_S_120	$0.2N$	6.32	3.12	0.50	0.03	0.02	0.06
	$0.4N$	8.50	5.38	2.92	0.03	0.01	0.04
	$0.6N$	9.45	6.35	3.82	0.04	0.02	0.05
	$0.8N$	9.50	6.53	4.14	0.06	0.02	0.01
	N	9.57	6.49	4.04	0.04	0.06	0.04
I_S_200	$0.2N$	6.30	2.42	0.05	0.08	0.04	0.14
	$0.4N$	9.01	5.08	1.93	0.10	0.01	0.03
	$0.6N$	10.01	6.24	3.06	0.10	0.08	0.10
	$0.8N$	10.19	6.41	3.34	0.12	0.05	0.13
	N	10.14	6.41	3.34	0.03	0.04	0.02

The conservatism of the strict model in [7] produces a reduction factor higher than 9% for most of the robustness levels. By allowing recovery, the proposed model becomes less conservative and there is an improvement in the reduction factor measure. In average, the reduction factor is 40% lower for each 10% recovery level increase. As the recovery stability point is reached, the reduction factor is almost none, i.e, the solution is virtually equal to the one generated by the classic NRP model. These results are consistent to state the improvement in solution quality for all recovery levels, even the small ones.

Figure 2 presents the reduction factors for some of the synthetic instances. The results are very similar to those presented in Table 2, as the reduction factor has a 40% decrease for each 10% recovery level increase, in average.

Fig. 2. Reduction factor results for synthetic instances I_S_100 and I_S_150

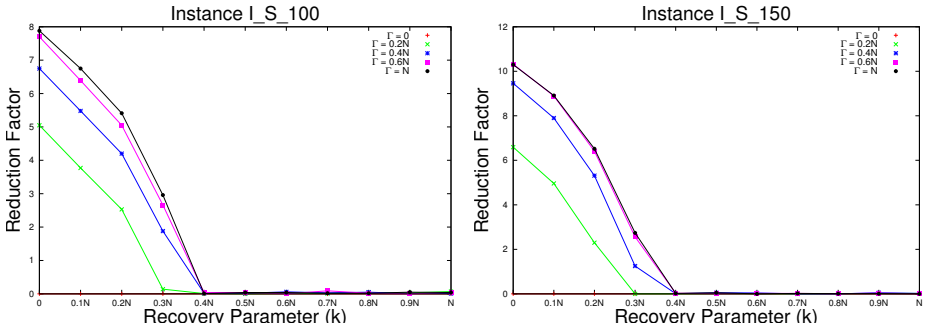

Table 3 presents the fitness values for some real-world instances. Due to space constraints, only one instance from each bug repository is presented.

As can be seen, the proposed recoverable model performs nearly the same for both synthetic and real-world instances, which helps to validate the first results.

Table 3. Fitness values results for real-world instances I_Re_E_100 and I_Re_M_150

Instance	Γ	k					
		0	$0.2N$	$0.3N$	$0.5N$	$0.7N$	N
I_Re_E_100	0	296.73 ± 0.60	296.73 ± 0.60	296.73 ± 0.60	296.73 ± 0.60	296.73 ± 0.60	296.73 ± 0.60
	$0.2N$	270.55 ± 1.01	286.64 ± 0.71	297.00 ± 0.42	296.73 ± 0.45	296.82 ± 0.61	296.82 ± 0.45
	$0.4N$	259.73 ± 1.52	274.45 ± 0.76	294.09 ± 0.61	296.91 ± 0.60	296.36 ± 0.70	296.82 ± 0.73
	$0.6N$	254.45 ± 1.31	268.82 ± 1.08	288.36 ± 0.36	296.82 ± 0.73	296.82 ± 0.45	296.73 ± 0.83
	$0.8N$	254.55 ± 1.35	267.00 ± 1.41	286.82 ± 0.45	296.64 ± 0.58	296.82 ± 0.61	296.55 ± 0.89
	N	254.55 ± 1.22	268.36 ± 0.79	286.82 ± 0.45	296.73 ± 0.45	296.45 ± 0.76	296.55 ± 0.68
I_Re_M_150	0	423.64 ± 0.59	423.64 ± 0.59	423.64 ± 0.59	423.64 ± 0.59	423.64 ± 0.59	423.64 ± 0.59
	$0.2N$	394.93 ± 0.50	419.00 ± 0.97	423.93 ± 0.48	423.93 ± 0.36	423.93 ± 0.58	424.00 ± 0.35
	$0.4N$	379.64 ± 0.73	402.79 ± 0.87	423.71 ± 0.62	423.93 ± 0.36	423.79 ± 0.56	423.71 ± 0.62
	$0.6N$	375.57 ± 0.43	397.29 ± 0.70	423.43 ± 0.53	424.07 ± 0.46	423.86 ± 0.35	424.14 ± 0.43
	$0.8N$	375.43 ± 0.80	396.50 ± 0.59	422.57 ± 0.65	424.00 ± 0.65	423.79 ± 0.33	424.14 ± 0.29
	N	374.93 ± 0.67	396.57 ± 0.83	422.71 ± 0.77	424.00 ± 0.57	424.00 ± 0.65	423.86 ± 0.47

When recovery is not allowed, the fitness value tends to get worse as the robustness level increases. As recovery is enabled, this conservatism is handled and the solutions converge to the classic NRP. Once again, even for small recovery levels, there is already a gain in solution quality when compared with the strict model in [7]. The standard deviation remains considerably small, reaching at most 1.41.

Figure 3 presents fitness value results for some real-world instances, where one instance of each bug repository is presented. The behavior of these real instance results are very similar to the synthetic ones, as could have been seen in Table 3. There is a gain in solution quality for all recovery levels under all robustness parameters values.

Fig. 3. Fitness value results for real-world instances I_Re_E_50 and I_Re_M_200

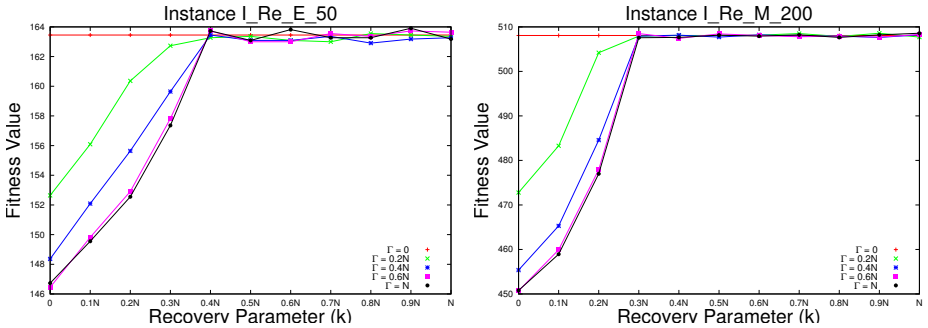

Table 4 presents the reduction factor results for some real-world instances. The conservatism of the model in [7], regarding the real instances, produced a reduction factor even higher than the synthetic instances, reaching more than 14% in some cases. The proposed recoverable robust model behavior was nearly identical of the synthetic instances, despite the high reduction factor when $k = 0$, a recovery level of 20% decreased the reduction almost to the half.

In conclusion, all results are consistent to show that the proposed recoverable robust model have improved previous robust models to the NRP, in order to handle the conservatism by allowing a release recovery. Results are very similar

Table 4. Reduction factor results for real-world instances I_Re_E_100 and I_Re_M_150

Instance	Γ	k					
		0	$0.2N$	$0.3N$	$0.5N$	$0.7N$	N
I_Re_E_100	$0.2N$	8.82	3.40	0.09	0.00	0.03	0.03
	$0.4N$	12.47	7.51	0.89	0.06	0.12	0.03
	$0.6N$	14.25	9.41	2.82	0.03	0.03	0.00
	$0.8N$	14.22	10.02	3.34	0.03	0.03	0.06
	N	14.22	9.56	3.34	0.00	0.09	0.06
I_Re_M_150	$0.2N$	6.78	1.10	0.07	0.07	0.07	0.08
	$0.4N$	10.39	4.92	0.02	0.07	0.03	0.02
	$0.6N$	11.35	6.22	0.05	0.10	0.05	0.12
	$0.8N$	11.38	6.41	0.25	0.08	0.03	0.12
	N	11.50	6.39	0.22	0.08	0.08	0.05

for both synthetic and real-world instances, pointing out the model's reliability and applicability.

Finally, all presented results have helped in answering the research question RQ_2, stating the improvement in solution quality when using the new recoverable robust model for the Next Release Problem.

4.3 Recovery Analysis

In this section, it will be analysed the fitness value behavior when some requirements are wrongfully estimated and, consequently, recovery becomes necessary to fulfill the budget constraint.

The fitness value *before recovery* is the same as presented in the early sections. The fitness value *after recovery* is computed assuming that the costs of the Γ requirements with highest variations were the ones wrongfully estimated, forcing the recovery algorithm to remove a considerably number of requirements in order to make the release feasible once again.

Since the proposed model performs nearly identical for both synthetic and real-world instances, as shown in previous sections, only a synthetic instance will be considered in the analysis.

Table 5 presents the fitness value results before and after recovery for the instance I_S_120, with the results after recovery being highlighted.

Table 5. Before and after recovery fitness values comparison for the instance I_S_120

Instance	Γ	k					
		0	$0.2N$	$0.3N$	$0.5N$	$0.7N$	N
I_S_120	0	581.21 ± 0.38	581.21 ± 0.38	581.21 ± 0.38	581.21 ± 0.38	581.21 ± 0.38	581.21 ± 0.38
		581.21 ± 0.38	581.21 ± 0.38	581.21 ± 0.38	581.21 ± 0.38	581.21 ± 0.38	581.21 ± 0.38
	$0.2N$	544.49 ± 0.70	563.05 ± 0.75	578.32 ± 0.49	581.37 ± 0.31	581.34 ± 0.46	581.53 ± 0.25
		544.49 ± 0.70	533.24 ± 3.58	521.99 ± 3.95	515.28 ± 7.82	517.84 ± 4.76	510.28 ± 7.45
	$0.4N$	531.82 ± 0.62	549.96 ± 0.28	564.25 ± 0.66	581.37 ± 0.34	581.24 ± 0.58	581.45 ± 0.19
		531.82 ± 0.62	522.84 ± 2.98	510.81 ± 5.07	486.17 ± 5.10	484.22 ± 6.75	482.69 ± 6.01
	$0.6N$	526.27 ± 0.70	544.28 ± 0.53	559.02 ± 0.41	581.45 ± 0.44	581.08 ± 0.76	581.51 ± 0.32
		526.27 ± 0.70	518.04 ± 2.67	509.58 ± 4.55	480.49 ± 7.82	478.16 ± 5.39	478.11 ± 6.70
	$0.8N$	525.98 ± 0.49	543.27 ± 0.56	557.17 ± 0.61	581.54 ± 0.29	581.32 ± 0.48	581.15 ± 0.60
		525.98 ± 0.49	518.07 ± 1.39	509.17 ± 3.28	482.48 ± 7.41	478.70 ± 3.40	485.00 ± 5.51
	N	525.58 ± 0.58	543.51 ± 0.43	557.70 ± 0.31	581.43 ± 0.53	581.55 ± 0.43	581.46 ± 0.40
		525.58 ± 0.58	518.89 ± 2.43	509.38 ± 4.35	478.75 ± 4.28	481.24 ± 5.74	480.83 ± 8.82

Since the fitness values before recovery are the same as in this previous sections, as the recovery parameter k increases, the fitness values converge to the

classic NRP. However, the fitness values after recovery behaves the opposite, since as the recovery levels grow, the solutions lose in quality if the recovery is performed, as can be seen in the table. That behavior highlights an interesting characteristic of the recoverable robust framework, that is, the recovery possibility can be considered a bet, that is, if the requirements' costs are correctly estimated, the solutions will behave very similarly to the ones from the classic NRP and significantly better than the results produced by the original robust framework. On the other hand, if the requirements' costs are wrongfully estimated, the solution after recovery will be worse than the conservative robust model. The decision maker must be aware of this trade-off to choose the robustness and recovery parameters which fit better in a particular release planning situation.

It is also worthwhile to highlight the considerable increase in the standard deviation for the fitness values after recovery. While for the before recovery values the highest standard deviation is 0.76, when considering the after recovery values, the highest standard deviation was 8.82.

Due to space constraints, it is not possible to show more after recovery results and analyis in this paper, but all results and instances are available at the paper supporting webpage, as mentioned previously.

5 Conclusion and Future Works

The Next Release Problem is an important activity in the iterative and incremental software development model. Since the requirements' characteristics, such as importance and cost, may change during the release development, robust approaches have been proposed to address this uncertainty in the NRP. However, the current robust methods to the NRP are very conservative because they select a subset of requirements in order to fulfill all possible uncertainties realizations.

This paper proposed an improvement to the state-of-art robust models to the NRP by considering the recoverable robust optimization framework. This modeling technique can handle the conservatism of the classic robust methods by adding a recovery possibility to the solution. If some cost uncertainty realization make the solution unfeasible due to budget constraints, the release can be recovered removing a controlled quantity of requirements.

The improved recoverable robust method was applied to both synthetic and real-world instances, varying the number of requirements in each instance. The real-world instances were extracted from bug repositories of two big open source projects, Eclipse and Mozilla. Experiments were performed in order to measure how much is the gain in solution quality when using the improved recoverable model instead of the conservative robust models.

For all instances and for all robustness level configurations, a small recovery level allows the production of solutions with more quality than the conservative strict robust models. Furthermore, as the recovery possibility increases, there is more gain in solution quality. However, if recovery is necessary, depending on the uncertainty realization, the solution quality after recovery can be worse than the conservative model. Therefore, the recovery possibility is a risky decision to

make and it is fundamental to perform a deep analysis in order to choose the best robustness and recovery levels for each situation.

As a future research direction, the recoverable robust optimization framework can be used to tackle other problems that have to cope with uncertainty in the SBSE field. Specifically related to the next release problem, the interdependencies between requirements could be considered. Furthermore, other experiments could be performed, varying the release available budget and using other strategies to cope with the cost uncertainty. Finally, other metaheuristics approaches, such ant colony optimization or particle swarm optimization could be tried.

References

1. Bagnall, A., Rayward-Smith, V., Whittley, I.: The next release problem. Information and Software Technology **43** (2001) 883–890
2. Harman, M., Krinke, J., Ren, J., Yoo, S.: Search based data sensitivity analysis applied to requirement engineering. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, ACM (2009) 1681–1688
3. Zhang, Y., Finkelstein, A., Harman, M.: Search based requirements optimisation: Existing work and challenges. Requirements Engineering: Foundation for Software Quality (2008) 88–94
4. Beyer, H., Sendhoff, B.: Robust optimization–a comprehensive survey. Computer methods in applied mechanics and engineering **196** (2007) 3190–3218
5. Bai, D., Carpenter, T., Mulvey, J.: Making a case for robust optimization models. Management science **43** (1997) 895–907
6. Mulvey, J., Vanderbei, R., Zenios, S.: Robust optimization of large-scale systems. Operations research **43** (1995) 264–281
7. Paixao, M., Souza, J.: A scenario-based robust model for the next release problem. To appear in ACM Genetic and Evolutionary Computation COnference (GECCO 2013) (2013)
8. Liebchen, C., Lübbecke, M., Möhring, R.H., Stiller, S.: Recoverable robustness. (2007)
9. Liebchen, C., Lübbecke, M., Möhring, R., Stiller, S.: The concept of recoverable robustness, linear programming recovery, and railway applications. Robust and online large-scale optimization (2009) 1–27
10. Büsing, C., Koster, A.M., Kutschka, M.: Recoverable robust knapsacks: the discrete scenario case. Optimization Letters **5** (2011) 379–392
11. Büsing, C., Koster, A., Kutschka, M.: Recoverable robust knapsacks: γ-scenarios. Network Optimization (2011) 583–588
12. Bertsimas, D., Sim, M.: The price of robustness. Operations research **52** (2004) 35–53
13. Xuan, J., Jiang, H., Ren, Z., Luo, Z.: Solving the large scale next release problem with a backbone-based multilevel algorithm. Software Engineering, IEEE Transactions on **38** (2012) 1195 –1212
14. Eclipse. http://www.eclipse.org/ (January, 2013)
15. Mozilla. http://www.mozilla.org/ (January, 2013)
16. Holland John, H.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. USA: University of Michigan (1975)