University of London
University College London
Department of Computer Science

Exact Analysis for Requirements Selection and Optimisation

Lingbo Li

Submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy in Computer Science of the University of

and
the Diploma of University College London, July 2017

Abstract

Requirements engineering is the prerequisite of software engineering, and plays a critically strategic role in the success of software development. Insufficient management of uncertainty in the requirements engineering process has been recognised as a key reason for software project failure. The essence of uncertainty may arise from partially observable, stochastic environments, or ignorance. To ease the impact of uncertainty in the software development process, it is important to provide techniques that explicitly manage uncertainty in requirements selection and optimisation.

This thesis presents a decision support framework to exactly address the uncertainty in requirements selection and optimisation. Three types of uncertainty are managed. They are requirements uncertainty, algorithmic uncertainty, and uncertainty of resource constraints. Firstly, a probabilistic robust optimisation model is introduced to enable the manageability of requirements uncertainty. Requirements uncertainty is probabilistically simulated by Monte-Carlo Simulation and then formulated as one of the optimisation objectives. Secondly, a probabilistic uncertainty analysis and a quantitative analysis sub-framework *METRO* is designed to cater for requirements selection decision support under uncertainty. An exact Non-dominated Sorting Conflict Graph based Dynamic Programming algorithm lies at the heart of METRO to guarantee the elimination of algorithmic uncertainty and the discovery of guaranteed optimal solutions. Consequently, any information loss due to algorithmic uncertainty can be completely avoided. Moreover, a data analytic approach is integrated in *METRO* to help the decision maker to understand the remaining requirements uncertainty propagation throughout the requirements selection process, and to interpret the analysis results. Finally, a more generic exact multi-objective integrated release and schedule planning approach *iRASPA* is introduced to holistically manage the uncertainty of resource constraints for requirements selection and optimisation. Software release and schedule plans are integrated into a single activity and solved simultaneously. Accordingly, a more advanced globally optimal result can be produced by accommodating and managing the inherent additional uncertainty due to resource constraints as well as that due to requirements. To settle the algorithmic uncertainty problem and guarantee the exactness of results, an ε-constraint Quadratic Programming approach is used in *iRASPA*.

Acknowledgements

I would like to thank all the those people who so generously helped to the work presented in this thesis.

First and foremost, I would like to express my sincere gratitude to my primary supervisor Prof. Mark Harman for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. I would also like to thank my secondary supervisor Dr. Emmanuel Letier for his support and comments and my co-supervisor Dr. Yuanyuan Zhang for her guidance, insightful and helpful discussions, and valuable assistance. I am very grateful to Prof. Inmaculada Medina-Bulo and Prof. Francisco Palomo-Lozano for the excellent collaboration we had and for the assistance they provided.

My sincere thanks and appreciations also go to all my colleagues: 1) at the Centre for Research in Evolution, Search and Testing for their general support and discussions, especially to Prof. William Langdon, Dr. Yue Jia, Dr. Ke Mao, Dr. Fan Wu, Dr. Federica Sarro, and Mr. Zheng Gao; and 2) at Department of Computer Science for their encouragement and support: Mr. Leslie Kanthan and Mr. Michail Basios. I would also like to thank the various anonymous referees for their comments on all my papers submitted for publication; their comments and feedback have been extremely beneficial.

I gratefully acknowledge the China Scholarship Council, University College London Doctoral school conference fund, EPSRC DAASE project fund, and Microsoft Azure for Research grant schema for the financial and research support that they have provided.

Finally, I would like to dedicate my particular appreciation to my supportive family for their constant encouragement, support and patience throughout all difficult times in my Ph.D study.

Dedication

I, Lingbo Li , confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

The work presented in this thesis is original work undertaken between September 2013 and April 2017 at University College London, University of London. Some of the work presented in this thesis has previously been published in and submitted to the following publications:

1. Lingbo Li, Mark Harman, Emmanuel Letier, and Yuanyuan Zhang. Robust next release problem: Handling uncertainty during optimization. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 1247–1254. ACM, 2014
2. Lingbo Li. Exact analysis for next release problem. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 438–443, Sept 2016
3. L. Li, M. Harman, F. Wu, and Y. Zhang. The value of exact analysis in requirements selection. *IEEE Transactions on Software Engineering*, 43(6):580–596, June 2017
4. Lingbo Li, Inmaculada Medina-Bulo, Francisco Palomo-Lozano, Mark Harman, and Yuanyuan Zhang. iRASPA: An exact multi-objective integrated release and schedule planning approach. *Submitted to IEEE Transactions on Software Engineering*, 2017. Under review

Moreover, the following work has also been published or submitted during the programme of study, but does not feature in this thesis itself. This work was conducted by the student during the course of his Ph.D study in collaboration with others.

1. Haitao Dan, Mark Harman, Jens Krinke, Lingbo Li, Alexandru Marginean, and Fan Wu. *Pidgin Crasher: Searching for Minimised Crashing GUI Event Se-*

quences, chapter Search-Based Software Engineering: 6th International Symposium, SSBSE '14, Fortaleza, Brazil, August 26-29, 2014. Proceedings, pages 253–258. Springer International Publishing, Cham, 2014

2. Lingbo Li, Mark Harman, Fan Wu, and Yuanyuan Zhang. *SBSelector: Search Based Component Selection for Budget Hardware*, chapter Search-Based Software Engineering: 7th International Symposium, SSBSE '15, Bergamo, Italy, September 5-7, 2015, Proceedings, pages 289–294. Springer International Publishing, Cham, 2015

3. Michail Basios, Lingbo Li, Fan Wu, Leslie Kanthan, Donald Lawrence, and Earl Barr. Darwinian data structure selection. *Submitted to IEEE Transactions on Software Engineering*, 2017. Under review

4. Michail Basios, Lingbo Li, Fan Wu, Leslie Kanthan, and Earl T. Barr. *Optimising Darwinian Data Structures on Google Guava*, pages 161–167. Springer International Publishing, Cham, 2017

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In software engineering, determining the set of requirements to implement in the release is a critical foundation for the success of a project. Inappropriately including or excluding requirements may result in products that fail to satisfy stakeholders' needs, and might cause loss of revenue. However, uncertainty (characterised by incomplete understanding) is inevitable in the early phase of requirements engineering, and could lead to unsound requirement decisions. To overcome uncertainty, the requirement engineering decision support needs uncertainty management.

This thesis presents a decision support framework for managing and analysing the uncertainty in requirements selection and optimisation. The uncertainties include uncertainty about development resource availability, the impact of dynamic and frequent changes at software runtime [13, 14], and the requirements ambiguity [15]. In this thesis, three types of uncertainty are taken into consideration: Firstly, we introduce a simulation-based requirements selection and optimisation model to manage the requirements uncertainty. Secondly, we design a Dynamic-Programming-based exact requirements selection solver to eliminate algorithmic uncertainty and a requirements uncertainty analysis framework *METRO* to interpret the impact of requirements uncertainty. Lastly, we take into account and address the uncertainty concerning allocation of resources by introducing an integrated software release and schedule planning model with

ε-constraint based Quadratic Programming.

In this chapter, we firstly provide the motivation for conducting this study. We then lay out the objectives and primary contributions of this work. Finally, we provide an overview of the structure of this thesis.

1.1 Motivation of the Research

The term *software requirement* is defined as “the property which must be exhibited in order to solve some problem in the real world" [16]. Determining an appropriate subset of requirements to be delivered in the following releases of a software system is a critical aspect in software engineering, especially under limited resources. Essentially, requirements selection can be regarded as a complex combinatorial decision problem in which several stakeholders and restrictions have to be satisfied in different deployment environments [17].

Therefore, the requirements selection and optimisation problem can be formulated as a constrained optimisation problem. In 1996, Karlsson used the *Analytical Hierarchy Process* for supporting software requirements selection and prioritisation [18]. In 2001, Bagnall et al. [19] further formulated it as Next Release Problem (NRP) and used a search-based approach to explore the search space in pursuit of requirements combinations with maximal expected business value. In 2004, Greer and Ruhe [20] extended this model to the software release planning problem to cater for several releases. In their formulation, stakeholders' objectives are modelled quantitatively, and optimisation techniques (i.e., meta-heuristic algorithms, dynamic programming) are employed to explore and select a subset of requirements that is both feasible and well-suited to stakeholders' requirements. It is a non-trivial problem, known to be *NP-hard* [19, 21, 22]. The search space of this problem increases exponentially with the number of requirements.

Unfortunately, uncertainty is also an inherent characteristic of the software engineering process [23]. The essence of uncertainty is the lack of complete knowledge at the time a decision must be taken [24]. In requirements selection and optimisation, uncertainty may arise from limited knowledge or ignorance in domain knowledge, technical knowledge, and project management [25, 26].

The uncertainty in requirements selection and optimisation may be found from the requirement itself. Requirements are often incomplete, vague and subject to change. Requirement uncertainties include uncertainty about the development resource availability, the impact of dynamic and frequent changes in the whole software development life cycle, and the accuracy of the software project estimation. The requirements of a new system are uncertain if the users have not started to use it [27]. Besides, the uncertainty in requirements selection and optimisation can also be found in the algorithm used to tackle it. Non-deterministic approximate algorithm may introduce unnecessary uncertainty into the analysis of the problem and thereby produce the wrong advice, misleading the decision maker. This algorithmic uncertainty is especially undesirable in those situations where the decision maker has also to struggle with the inherent data uncertainty of the problem itself. Additionally, the uncertainty of project management information, such as the uncertainty of resources used to allocate the requirements, is also one source of uncertainty. Ignoring the resource constraints (i.e., the capacities and expertise of developers) is likely to result in invalid software release plan. Decision makers have to make decisions under such uncertainties. Underestimated or ignored uncertainties may bring risks into software projects, and might even result in project failure [27].

Over the last decade, various requirements selection and optimisation techniques have been developed in a context where the input requirements' attributes are concerned with point-based estimations, which are estimated by human requirements engineers [19, 28, 29, 30]. In those studies, the attributes of requirements and stakeholders are quantified as explicit values, and requirement uncertainty is either underestimated or completely overlooked [31]. For example, given a set of quantified requirements, although those point-based estimation approaches can provide optimal solutions in terms of expected cost and revenue, they fail to offer an assessment of the confidence placed in results. Thus, they may mislead the decision making and amplify the consequences of risks. According to Hubbard:

> “Certainty about real-world quantities is usually beyond their reach. The fact that some amount of error is unavoidable but can still be an improvement on prior knowledge is central to how experiments, surveys, and other scientific measurements are performed." [32]

the impact of uncertainty could be mitigated by performing mathematical measurement methods and scientific uncertainty management methods.

On the basis of previous research work, in order to manage uncertainty in requirements selection and optimisation, two approaches have been conducted:

The first approach is sensitivity analysis, which is an uncertainty-handling method. The purpose of performing sensitivity analysis is to evaluate the robustness of the outputs of a model in the presence of uncertainty, and to achieve uncertainty reduction through identifying sensitive model inputs. The principal of sensitivity analysis is to study how different sources of uncertainties in the input of a system will contribute to the uncertainties in the output of the system [33, 34]. Usually, sensitivity analysis applies changes in the input of the system, and studies what effect this produces on the output. This enables engineers to understand the consequences and sources of uncertainties, on account of building robust models.

In search-based requirements selection and optimisation, Harman et al. [22, 35] have applied sensitivity analysis to NRP to look into the sensitivities of the attribute of requirements. In their studies, they performed a local sensitivity analysis approach “One-At-a-Time" [36], which perturbs input requirements variables upward or downward to try out various ‘what-if’ scenarios. However, their one-at-a-time approach cannot scale to the context of many input parameters and higher order analysis, where the number of perturbed input parameters increases and the requirement interactions are taken into consideration. Harman et al. [35] used meta-heuristic algorithms as a modelling approach in their work. This may invalidate the interpreted results by introducing unnecessary uncertainty from the algorithm to the problem. Moreover, sensitivity analysis can only provide information on the sensitivities of parameters. It does not generate robust solutions, which can tolerate perturbations that might otherwise affect solutions offered by non-robust approaches, thereby making such a more robust solution more desirable.

Instead of making use of sensitivity analysis as a post-analysis for quantifying the sensitivity characteristics of the associated input in the underlying problem, some researchers have suggested the construction of a solution that is feasible for any realisation of the uncertain values, using robust optimisation [37, 38]. Robust optimisation is distinctly different from sensitivity analysis. Robust optimisation formulates the optimisation problem as one in which solutions that have *a priori* ensured robustness are sought against prescribed uncertainty [39]. Robust optimisation explores the solution space and takes uncertainty into account simultaneously.

Paixão et al. introduced robust optimisation to NRP to cater for uncertainty in search-based requirement optimisation [40]. In their proposed optimisation model, requirement uncertainty was represented as deterministic variability in the value of the input parameters. They used worst case analysis, which implies that the uncertainty was formulated as max-min bounds, to treat requirement uncertainty. Accordingly, their generated release plan, solutions were deterministically immune to realisations of the uncertainty, but were overly conservative in order to ensure this immunity. The major drawback of this non-probabilistic robust model is that the solutions it produces are conservatively expensive. This non-probabilistic robust model is well-grounded only when the uncertainty is not stochastic, or the distribution is unavailable [39].

Probabilistic robust optimisation has been proposed to take advantage of the principles of statistics to relax uncertainty [37, 41]. The principal of this probabilistic model is to quantify the uncertainty in the 'expected' value of the input of interest usage probability distribution function. These have been traditionally classified as stochastic programming and stochastic optimisation models.

Compared to non-probabilistic robust optimisation techniques, probabilistic robust optimisation techniques provide a notion of a *budget of uncertainty* [39]. This allows decision makers to choose the trade-off between robustness and performance, as well as the corresponding level of probabilistic protection. More importantly, compared to those solutions produced by non uncertainty-aware methods, probabilistic robust optimisation can produce solutions that sacrifice a little quality but reduce the uncertainty by a considerable degree.

On the other hand, to deal with uncertainty, it is important to know that all uncertainty derives from the problem itself and not from the algorithm used to tackle it. Nevertheless, the aforementioned previous works adopted non-deterministic approaches, such as Genetic Algorithms (GAs) [42], and Evolution Strategies (ES) [43], in their frameworks. Relying solely on approximate algorithms can only guarantee reasonable approximate solutions. In other words, there is information loss in the solution, and additional uncertainty from the algorithm is thereby introduced. While such information loss is acceptable in general, for the specific problem of handling uncertainty we face here, it is important to ensure that any uncertainty present in solutions offered, derives from problem itself not from the algorithm used to find solutions. This motivates the usage of exact algorithms for uncertainty handling. However, previous exact algorithms are problem-dependent and computationally expensive [44, 45]. Their execution time may increase exponentially with respect to the number of input parameters and the dimensions (number of objectives) of the problem.

In addition, apart from requirements uncertainty and algorithmic uncertainty, there is another uncertainty which has not received widespread attention from the requirements engineering community. Resource allocation during the software development process is dealt independently with requirements selection and optimisation. Therefore, the uncertainty of resource constraints is largely overlooked in the requirements selection and optimisation process, which may lead to the mismatch between requirements selection and requirements allocation. Exact selection of requirements to satisfy the software development in different deployment environments not only requires the management of requirements and algorithmic uncertainty but also needs managers and project leaders to address the problem of resource allocation.

Modelling resources typically involves a large number of variables related to budget allocation, resource availability, staffing, developer skills and scheduling. This process is mainly driven by human behaviour [46]. Basically, this problem has been modelled either as a constraint satisfaction problem, in which only the resource constraints are taken into account, or a constrained optimisation problem, in which different optimisation objectives are dealt with.

However, the existing requirements selection and optimisation problems and resource allocation problems that have been solved iteratively to assist decision makers in finding better subsets of requirements along with resource allocations aiming to on-time delivery. In fact, the vast majority of the literature is concerned with managing both problems independently, dealing with the resource allocation stage, only after the requirements selection stage has been completed. This two-stage approach may produce suboptimal results. Li et. al [47] identified this suboptimality and proposed an integrated approach, as they found that resources are unlikely to be allocated in an optimal way when the capacities and expertise of developers are taken into account only after requirements have been fixed in the release plan.

1.2 Objectives of the Research

The principal goal of this research is to provide a framework to manage uncertainty exactly and efficiently in requirements selection and optimisation. The detailed aims and objectives of this thesis are as follows:

1. Investigating the feasibility of applying probabilistic robust (simulation-based) optimisation for managing requirements uncertainty in NRP.

   Our technique uses Monte-Carlo Simulation (MCS) to probabilistically evaluate the requirements, simulating various sources of requirements uncertainty that affect their 'true' value, and calculating their statistical 'expected true' value and associated uncertain consequences over the range of resultant outcomes. The numeric uncertainty information produces is then taken as input or formulated as an extra fitness function to requirements selection and optimisation.

2. Proposing an exact NRP optimisation technique that guarantees to find optimal solutions in a finite amount of time.

   To eliminate algorithmic uncertainty, our proposed approach is based on the Nemhauser-Ullmann algorithm, an exact dynamic programming algorithm, as the core NRP solver, and augmented by a novel Conflict Graph to deal with the requirements interaction. The Conflict Graph converts the NRP to a search tree data structure to distinguish the original problem into many sub-problems (at the leaves) all of which have no requirement constraints and can thus be solved by Nemhauser-Ullmann algorithm directly.

3. Providing a probabilistic uncertainty analysis and quantitative analysis framework to help the decision makers to study requirements uncertainty propagation in the requirements optimisation process, and interpret the produced results.

   When solutions are found by our approach, one additional issue arises: notwithstanding that the NRP approach significantly reduces the complexity of decision making by lessening the solution space, it is still complex and laborious for decision makers who have to choose one from among them. Another objective is thus to help decision makers to make better-informed decisions by explaining results, based on novel quantitative analysis methods we adapt for this purpose.

4. Designing a more generic approach to holistically manage the uncertainty of resource constraints for requirements selection and optimisation.

Ignoring the uncertainty of resource constraints in software release planning problems makes software projects prone to failure. We aim to design a more generic and holistic software release planning approach to produce globally optimal release plans that would be able to accommodate both algorithmic uncertainty and uncertainty of resource constraints.

1.3 Contributions

This thesis introduces a decision support framework for analysing uncertainty in the requirement selection and optimisation process. The main contributions of the thesis are the following:

1. A probabilistic robust (simulation-based) optimisation model (*sNRP*) is introduced.

   This model uses Monte-Carlo Simulation to probabilistically evaluate requirements uncertainty, and formulates the simulated requirements uncertainty as one of the objectives for optimisation. This model enables a decision maker to analyse and optimise requirements and takes uncertainty into account simultaneously. Furthermore, the solutions produced by *sNRP* are very close to those produced by non uncertainty-aware methods in terms of the cost and revenue of solutions. The ranked proportion of requirements being selected in solutions on the Pareto-front produced by *sNRP* is significantly correlated with those produced by non uncertainty-aware models. According to our empirical study, the Kentall's τ_B coefficients (a non-parametric measure of association) are greater than 0.7, and p-values (chance of Type I error) are very close to zero in all cases.

2. An exact NRP optimisation solver *NSGDP* is designed that can guarantee to find the optimal solutions and eliminate algorithmic uncertainty.

   By virtue of this exact approach, decision makers can ensure that the variations between the fragile, but optimal, results and the conservative non-optimal results derive from the inherent uncertainties of the requirements, thus the stochastic nature of the approximate algorithms can be excluded. Our experimental studies reveal that, with the aid of *NSGDP*, the decision maker can avoid information loss (without which he or she will lose up to 99.95% of the optimal solutions and will make up to 36.48% inexact requirement selection decisions). Also, fortunately, as distinct from other previous time-consuming exact algorithms [44, 45], *NSGDP* is comparatively efficient. The execution time of *NSGDP* is better than NSGA-II. On average, *NSGDP* takes $0.37s$ (without accounting for requirements uncertainty), and $35.33s$ (when taking requirements uncertainty into consideration). By contrast, NSGA-II takes more than 10 minutes, whether or not requirements uncertainty is taken into account. We thus have introduced an algorithm that removes uncertainty due to inexact algorithm, yet also improves performance.

3. A decision support framework, *METRO*, is provided that allows decision makers to study requirements uncertainty propagation in the requirements optimisation process probabilistically, and interpret the results produced.

   METRO investigates the difference between the optimal-yet-risky solutions and robust-yet-suboptimal solutions. Two indicators are used: expected risk premium and risk reduction. Our experimental results show that, developing a software project based on optimal-yet-risky release plan rather than robust-yet-suboptimal release plan, may suffer up to 10% probability of overrunning more than 150% budget but gaining less than 0.39 expected risk premium. A series of quantitative techniques are provided for highlighting the characteristics of requirements and solutions. The difference of requirement selection probability between two NRP approaches is analysed and presented in a stacked bar plot. We found that the risk-aware *sNRP* approach is more likely to include requirements with low uncertainties than *pNRP* approach is (Kendall‘s τ_B is no smaller in magnitude than -0.675). *METRO* clusters requirements according to design space proximity rather than objective space proximity.

4. A generic exact multi-objective integrated release and schedule planning approach *iRASPA* based on a Quadratic Programming model is provided to holistically manage the uncertainty of resource constraints for requirements selection and optimisation.

   iRASPA not only provides a release plan that maximises the value of the delivered software and minimises the variance of the workload, but also meets all the resource allocation constraints. We argue that *iRASPA* can effectively help decision makers to avoid suboptimality and algorithmic uncertainty. *iRASPA* is evaluated on seven real-world software projects, instantiated as 245 instances augmented with synthetic data to cater for missing values. The experimental study shows that *iRASPA* can effectively generate the guaranteed exact Pareto front, unlike the current state-of-the-art, which misses 87.84% of the optimal solutions on average and up to 93.38% for some instances. In addition, it takes *iRASPA* 28.23% less time on average to solve all the instances under study.

1.4 Organisation of the PhD Thesis

The structure of rest of the thesis is organised as follows:

Chapter 2 Literature Review: Briefly surveys state-of-the-art of related work in requirements optimisation, uncertainty analysis, robust optimisation, and software project resource allocation.

Chapter 3 Simulation based Robust Next Release Problem Model: Presents a Monte-Carlo Simulation based Multi-Objective Next Release Problem framework. Requirements uncertainty is simulated and the consequence of uncertainty is formulated as an extra objective to optimise. Two experimental studies are carried out to address the usefulness and efficiency of this NRP model.

Chapter 4 The Value of Exact Analysis in Next Release Problem: Introduces a decision support framework *METRO* for the Next Release Problem to manage algorithmic uncertainty and requirements uncertainty. A novel exact technique is developed to guarantee the exactness of solutions to eliminate algorithmic uncertainty. Some quantitative analysis approaches are presented to support decision makers in their understanding of the impact of requirement uncertainty. The aim is to inspire them to prioritise the requirements for further evaluation and inclusion. The results and analysis of three experimental studies are provided.

Chapter 5 Exact Analysis in Integrated Release and Schedule Planning Problem: Provides a holistic software release planning and schedule planning approach *iRASPA*, which utilises ε-constraint based Quadratic Programming as the solver, to solve requirement selection problem while taking uncertainty of resource constraints into account. The experimental study of evaluating *iRASPA* on 245 synthetic software project instances, derived from 7 real-world software projects, is analysed and explained.

Chapter 6 Conclusions and Future Work: Concludes the thesis with a discussion on the threats to validity and suggestions for potential future work directions.

Chapter 2

Literature Review

This chapter presents a literature review to establish a foundation for the research undertaken in this thesis. The PhD thesis intersects with three main research fields: the field of requirements engineering, uncertainty handling, and software project resource allocation. First, this chapter briefly introduces concepts related to requirements engineering. It then provides a detailed description about requirements selection and optimisation problem, as well as the state-of-the-art optimisation techniques. Next, uncertainty management, in general, and in requirements optimisation are described. Finally, approaches to resource allocation in the software development process related activities are explained.

2.1 Overview of Requirements Engineering

2.1.1 Requirement

Before building a software system, one must answer the question: "what is the goal one wants to achieve?". Such objectives are expressed as a set of necessary services, capabilities, constraints, and the quality of the requested system that should be offered to a set of stakeholders. Requirements engineering determines what the functionalities the system should provide, how it exhibits these functionalities, and what are the qualities and constraints that the system must satisfy. In software engineering, the term *requirement* has been used to express these purposes [48]. In 1990, the Institute of Electrical and Electronics Engineers Computer Society (IEEE) formally defines requirements as [49]:

1. A condition or capability needed by a user to solve a problem or achieve an objective.

2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.

3. A documented representation of a condition or capability as in definition 1 or 2.

According to the definitions & specification of requirements and the group of requirement readers [9], the requirements can be categorised into two types: a) user requirements and b) system requirements.

User requirements state the expected services, capabilities, constraints, and the quality of the requested system should be offered in a way that the system users can understand without any professional technical knowledge background. User requirements are the high-level statements of what the system should do. The potential readers of user requirements are client managers, system end-users, client engineers, contractor managers, and system architects.

System requirements are the structured and in-depth specification of user requirements. As distinguished from the user requirements, system requirements technically set out in-depth descriptions of the requested functionalities, services, and operational constraints of the system. System requirements provide the solution to answer user requirements. The system requirements are illustrated by the system model and impose a degree of uniformity on the specification.

The requirements also can be classified into functional requirements, non-functional requirements, and domain requirements [9].

Functional requirements prescribe the functionalities and specific behaviours of a software system. Functional requirements interpret what the software system is supposed to do in particular situations, and form the inputs, procedures, and outputs of the system. A typical functional requirement example is :"The university library system should display the books based on users' queries".

Non-functional requirements specify the constraints on how the performance characteristics should be provided by the software system. Non-functional requirements specify criteria for how well the system does, rather than requests what should the system do. Hence, non-functional requirements are also regarded as quality requirements. In software development: "In the university library system, the books query should be retrieved within 1 second." The typical non-functional requirements include but are not limited to : scalability, capacity, reliability, maintainability, security, and recoverability. Figure 2.1 illustrates the non-functional requirement type.

Domain requirements reflect the fundamentals and characteristics of the application domain of the system, and the environment in which the system operates. The system will not work satisfactorily if the domain requirements fail to be satisfied. Usually, a domain requirement is expressed in special domain terminology by domain experts. It is possible that software engineers misunderstand the domain requirement and implement requirement in the wrong way. For example, an online academic library system requests that the H-index of each author should be computed. To implement this domain requirement, software engineers have to know some statistics knowledge about H-index.

Requirements are not only an essential input for system design, but also an absolutely necessary element for system verification. In 1987, Brooks and Frederick indicated that establishing the requirements is the hardest phase of software development life cycle:

"The hardest single part of building a software system is deciding pre-

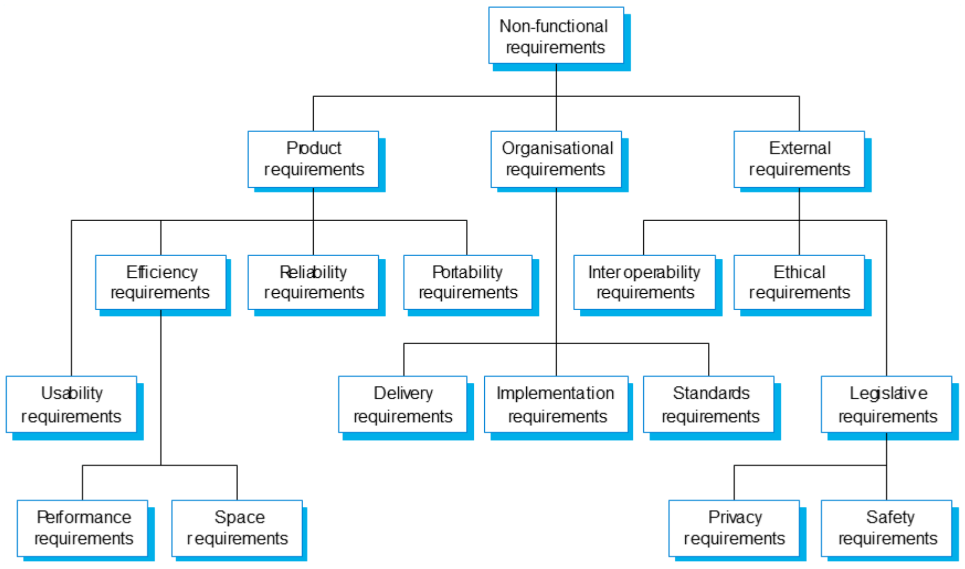


Figure 2.1: The hierarchy of non-functional requirements types [9].

cisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements . . . No other part of the work so cripples the resulting system if done wrong. No other part is as difficult to rectify later" [50]

In practice, the stakeholder of a software system tends to be a set of individuals who have varied and inconsistent requirements of the system. They usually lack the ability to technically and thoroughly present their requirements. Moreover, due to the complexity of software systems and the fact that stakeholders may lack complete knowledge about what they want, requirements are generally altered with time. This situation further multiplies the difficulties of producing the high quality software requirement. As a result, it is necessary to establish a structured procedure to investigate and process the requirements.

2.1.2 Requirements Engineering

Requirements engineering is the process of developing the requirements through systematically analysing, documenting, and reviewing the understanding of the problem [51]. It identifies the purpose and properties of a software system and it is formed at the early phase in the software development life cycle (SDLC) [52]. The activities in this phase have been recognised as the prerequisite and foundation of the entire SDLC process. It was regarded as a crucially important role in the entire SDLC processes by industry and academia [53] and was proposed as the term *Requirements Engineering* (RE) in 1976 [54]. The term *Requirements Engineering* was then formally defined by Dorfman in IEEE Computer Society Tutorial in 1990 [55].

In the latest IEEE Standard, the term *Requirements Engineering* was defined as:

> “Interdisciplinary function that mediates between the domains of the acquitter and supplier to establish and maintain the requirements to be met by the system, software, or service of interest." [56]

Hence, RE is a multi-disciplinary, human-centred process. The IEEE standard defines requirements engineering as: “serial activities that concerned with stakeholder requirements definition process, requirement analysis process, requirement formalising & validating & documenting process, and requirement management [56]". It establishes the bridges among different groups of stakeholders as well as connecting stakeholders with engineering technologies and resources. According to this IEEE standard, a typical taxonomy of the requirements engineering process is elicitation, analysis, specification, validation & verification (acceptance), and management [57].

Requirements elicitation

Requirements elicitation is the process of identifying the stakeholder of the system, eliciting the stakeholder requirements, defining the constraints of the system, documenting and analysing the completed elicited requirements, and maintaining the traceability of stakeholder requirements. It is regarded as the first activity in the requirement engineering process. The requirements elicitation firstly captures the requirements, identifies the stakeholder, and then gathers the requirements by interpreting, analysing, modelling and validating to make requirements engineers feel confident about the next activity “requirements analysis". The outcomes of the successful implementation of requirements elicitation include the characteristics of the anticipated system, the constraints on the system solution, the defined stakeholder requirements, and the validation for stakeholder requirements.

Requirements modelling and analysis

The fundamental activity in Requirement Engineering is requirements modelling and analysis [58]. Requirements modelling and analysis constructs the abstract elicited requirements into an accurate presentation of the products of the RE process by utilising requirements analysis techniques. The purpose of the requirements modelling and analysis is refining the user’s needs and constraints. According to Nuseibeh and Easterbrook [57], there are four general categories of requirements modelling and analysis: Enterprise Modelling, Domain Modelling, Behavioural Modelling, and Data Modelling.

Enterprise Modelling is high-level goal modelling. It is used to capture the need of a system based on the organisational structure and business rules.

Domain Modelling sets up the gathered domain description which incorporates all relevant domains under the developed software system. An explicit domain model offers an abstract description of the proposed system.

Behavioural Modelling is the process which models the dynamic or functional behaviour of the stakeholder and systems. It converts the statement of system needs into the systematical statement of requirements.

Data Modelling structures the semi-structured or unstructured data that would be managed in the RE process with constraints or limitations. Data modelling aids requirement engineers in understanding, manipulating, and managing the large volume of information used in the proposed system by transferring this semi-structured or unstructured information into a structured precise model.

Requirements specification

Requirements specification is the process of documenting the user's needs and constraints (functional and non-functional) clearly and precisely. Usually, requirements specification is followed by the requirements elicitation and analysis process. The requirements specification documents RE activity descriptions and the description of the behaviour of a system to be developed. Namely, it describes the essential technical requirements for system, and the criteria for determining whether those requirements are met.

The requirements specification can be established and refined in natural language or formal language. Natural language expresses the requirements in the natural form. Since the requirements documents are written by semi-technical analysts in cooperation with customers‘ experts and potential users and have to be signed-off by non-technical business executives, the majority of informal requirement specifications are written in natural languages [59]. The natural language used in requirement specification provides a widely accepted form of communication for most people and effectively gains understanding and agreement by both customers and developers.

The latter, formal specification, is a mathematically based language whose purpose is to support engineers to construct systems. The formal specification adopts precisely defined vocabulary, and mathematical syntax to represent the system, and analyse its behaviour [60]. Since it forces a detailed analysis of the requirement by precise and rigorous representation in the early phases of software development, formal specification can reduce requirement error as well as discover and resolve the incompleteness and inconsistencies [61, 62]. Moreover, the well structured and precisely described requirements formal specification offers the basis for agreement between the stakeholders on the expectation of the system to be developed, and formats the rigorous assessment for defining the condition or capability to which the system must conform [56]. Albert the formal specification language has limited practical applicability [57], during the last 30 years it has reached maturity level. In 2009, Hierons et al. [62] pointed out that the formal specification language can bridge the gap between requirements engineering and software testing.

Requirements validation and verification

Requirements validation and verification are independent procedures that take place throughout the whole software development life cycle. They are used for checking and ensuring that the specification is complete, consistent, modifiable and traceable, and that the produced system meets the specifications and fulfils the intended requirements.

Although validation and verification are usually mentioned together in requirement engineering literature, they are not the same. According to the latest IEEE project management institute standard [63], the definitions and activities of *validation* and *verification* are not the same (as shown below).

> “*Validation*. The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with verification."
>
> “*Verification*. The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with validation. "

In the light of the IEEE standard definitions, validation is intended to be processed to establish the evidence to prove that the performed activities (service), and generated output fulfil the defined design specifications and quality criteria. In layman's terms, validation can be expressed as the question “Are you building the right thing?". The answer to this question is the degree of completeness and correctness of requirements. Since requirement validation often involves end users and other product stakeholders directly and closely, it is often an external process. There are a host of techniques to process the validation described in literature. The most popular ones are: review and inspection, prototyping, traceability, and testing [64].

Verification is refereed to query “Are you building it right?". It is a process used to assess whether the designed and built system are fully satisfied with the specifications. Namely, it denotes checking that the specifications are correctly implemented by the system. In most of the cases, a fruitful verification consists in carrying out various inspections, tests, consistency management, and analyses throughout the whole software development life cycle [64, 65].

Requirements management

The requirements management process is a crucial process which goes through all of the software development life cycle. Requirements management is a process to establish a common understanding among the stakeholder, and oversee the software system development through to delivery and operation of the project processing to make sure that the delivery of capability meets intended requirements in a timely and cost effective manner [66, 67, 68]. Robust requirements management can help to lay foundation for the system affordability, and mitigate (or even avoid) the unanticipated consequences of changes through rigorous documentation.

The activities of requirements management involve requirements traceability management, requirements quality assurance, impact analysis, and reuse of requirements. The core task of requirements management is requirement traceability management. It documents changes and the rationale of these changes as well as maintaining bi-directional traceability (both forwards and backwards) [69]. Requirement traceability management enables tracking the progress of a project, assessing the impact of various changes by documenting exhaustive information. One of the widely used requirement traceability management techniques in industry is Requirements Traceability Matrix (RTM) [70]. All requirements in the specification, the system-wide effects of the requirements change (including the decomposition of the requirements, the derivation of the requirements, and the allocation history of the requirements), and the rationale for all entries and requirements changes can be captured by RTM.

2.2 Requirements Selection and Optimisation

Requirements may have different values to stakeholders, and require different effort to implement. In an ideal world, all requirements would be simply selected to be implemented in one release, thereby yielding maximal functionality and value to stakeholders. However, in practice, resource constraints need to be taken into consideration. Given limited budgetary resources, it is impossible to implement all requirements in one release. Requirement engineers have to make decisions to determine the priority of requirements and decide whether a requirement should be implemented in the next release of the system or not, meanwhile the outcome of the software system in the next release can be maximised [16].

In 1990, Yeh and Ng found that implementing the ranked requirements in sequence can bring benefits to the software development project [71]. The benefits include, but are not limited to:

1 Providing decision support for resolving trade-offs in subsequent development.

2 Helping project managers to predict the expected customer satisfaction and market performance of the software system.

Requirements selection and prioritisation activity is concerned with selecting a subset of requirements to implement to meet the demands of stakeholders and maximise the value of delivered software product, at the same time, scale to ensure that there are sufficient resources to undertake the development [17]. Such a requirements optimisation problem is recognised as a complex combinatorial problem.

In the literature, various techniques have been developed to address requirements optimisation problem. Technically, the requirements optimisation techniques can be categorised into priority-based requirements optimisation, search-based requirements optimisation, and exact requirements optimisation [72].

2.2.1 Priority-based Requirements Optimisation

Priority-based requirements optimisation is an intuitive approach in which the requirements are sorted from 'best' to 'worst' based on their characteristics or the interests of the stakeholders. The ranking of requirements implicitly indicates the priority of requirements. The developers can choose the requirements with the highest priority to implement, accordingly, achieving the earliest satisfaction.

In 1996, Karlsson [18] proposed two approaches to prioritise requirements. The first approach is Analytical Hierarchy Process (AHP), which is a pair-wise comparison mathematical prioritisation methodology [73]. It requires decision makers to manually estimate the *relative* importance of each requirements by pair-wise comparison. The second approach is *numeral assignment technique*, which is concerned with Quality Function Deployment (QFD) [74]. The *numeral assignment technique* uses a symbol to represent the requirement's perceived importance. The requirement's priority is indicated by an ordinal scale (ranging from 0 to 5). The drawback of these two approaches is obvious: Neither of them can handle requirements dependency. Besides, the scalability of both approaches is barely satisfactory: For a software project with n candidate requirements, the *numeral assignment technique* requires n requirement's priority assignment, while the AHP approach needs $n \times (n-1)/2$ pair-wise comparisons.

Karlsson and Ryan developed a cost-value approach, which is built upon AHP requirements prioritisation approach, to estimate requirement's relative cost and value in 1997 [75]. Comparing with AHP requirements prioritisation approach, the improved cost-value method can better interpret the potential cost-value contribution of a candidate requirement to stakeholder satisfaction with the resulting system.

To systematically investigate the state-of-the-practice requirements prioritisation methods in requirements engineering, Karlsson et al. [76] evaluated and compared six requirements optimisation techniques, which are AHP, hierarchy AHP, spanning tree matrix, bubble sort, binary search tree and priority groups.

Hierarchy AHP is extended from aforementioned AHP method. It possesses similar characteristics to AHP. It structures requirements in a hierarchy of interrelated requirements, and placed the requirements from top to bottom based on generality. As prioritisation, hierarchy AHP compares all outlined pairs of requirements at the same hierarchy level by AHP for each level. The requirements' priorities are then propagated down the hierarchy. Due to the amount of redundancy, hierarchy AHP is more sensitive than AHP, even though it can reduce the required number of pair-wise comparison.

The principle of minimal spanning tree [77] is constructing a directed graph in which there is at the minimum one path between the requirements rather than pair-wise compared, thus $n - 1$ unique pairs of requirements can be determined. According to the constructed minimal spanning tree, decision makers can assess the requirement's importance by taking the geometric mean of connected requirements which have already been assessed. The reduced number of pair-wise comparisons dramatically improves the speed of prioritisation, but makes this approach more sensitive to judgemental errors.

Bubble sort is a basic sorting method [78], and it is closely related to AHP. It requires $n \times (n - 1)/2$ pair-wise requirement comparisons, in the meantime, the extent is not required.

The idea of applying binary search tree [79, 80] to requirements prioritisation is similar to minimal spanning tree approach. It prioritises n candidate requirements by constructing a binary search tree consisting of n nodes. Consequently, the construction takes, on average, $O(n \log n)$ comparisons, and the requirements are prioritised on an ordinal scale.

The priority groups method proposes to reduce the required prioritisation effort by not to compare the requirements in distinct sets. Firstly, it divides the candidate requirements into separate groups based on a rough prioritisation (high, medium, and low). In the following prioritisation session, one of the aforementioned prioritisation approach is applied to further prioritise the requirements in sub-groups.

The investigation results revealed that AHP is the most promising requirements prioritisation method, although it is labour-intensive and may be problematic to scale-up. Moreover, none of them can offer the ability to support for requirement dependency.

To account for the large scale distributed software project development and reduce the cognition load for stakeholders, Regnell et al. [81] introduced a distributed requirements prioritisation framework. The sequence of prioritisation process is: firstly, all candidate requirements are divided into two abstract levels (high-level and low-level); secondly, the high-level requirements are assigned to all stakeholders for parallel prioritisation; and then, stakeholders communicate to each other and the Product Strategy Team aggregates the individual priorities; lastly, the results on high-level requirements priorities are then used as a guidance when prioritising low-level requirements.

In 2011, Lim et al. [82] proposed StakeSource2.0, a web-based requirements prioritisation tool that uses ‘crowdsourcing’ approach to identify and prioritise stakeholders and their requirements in large scale software projects. Social network analysis and collaborative filtering techniques are used to automatically collect requirements and their ratings from each stakeholder. Each stakeholder is asked to provide private judgements about requirement importance and requirement influence. The requirement priority is then aggregated by its importance and weighted influence. However, these two frameworks are restricted by the prerequisite that stakeholders should be familiar with at least one of the prioritisation methods, and the required effort is most likely to be overwhelming.

2.2.2 Heuristic Search-based Requirements Optimisation

According to Search-based Software Engineering (SBSE), which was coined by Harman and Jones in 2001 [21], complex, multi-objective, and highly constrained software engineering problems can be formulated as search-based optimisation problems that can be tackled with heuristic search algorithms. To convert a software engineering problem into a computational search problem, a fitness function is needed to measure the quality of candidate software engineering problem solutions. Naturally, the requirements optimisation problem is a requirements combination problem, and can be viewed as an application area for SBSE [19, 29, 83].

Bagnall et al. [19] proposed the term *Next Release Problem* (NRP), and attempt to formulate requirements selection and optimisation as a combination-based requirements release planning problem. The NRP model assumes that there is a set of stakeholders and their features in the next release of a software system. The set of stakeholders is denoted by Eq.2.1 and the set of possible requirements is denoted by Eq.2.2.

$$C = \{c_1, \cdots, c_m\} \tag{2.1}$$

$$R = \{r_1, \cdots, r_n\} \tag{2.2}$$

where m is the number of stakeholders, and n is the number of features.

In this thesis, all requirements are independent of each other. During the software development, some resources (e.g., human resources and facility resources) need to be allocated to satisfy each requirement. NRP uses cost to measure the amount of resource needed to fulfil the requirement as given by Eq.2.3.

$$Cost = \{cost_1, \cdots, cost_n\} \tag{2.3}$$

There is a weight vector which reflects the degree of importance of each stakeholder for the company. The relative weight vector related to each stakeholder c $(1 \leq j \leq m)$ is denoted as Eq.2.4:

$$Weight = \{w_1, \cdots, w_m\} \tag{2.4}$$

subject to: $w_j \in [0, 1]$, and $\sum_{j=1}^{m} w_j = 1$.

The authors assumed that the importance of each requirement for each stakeholder is different. Given a stakeholder, the level of satisfaction of this stakeholder is based on the requirements that are satisfied in the evolved suggestion for the next release of the software system. Based on this assumption, each requirement r_i $(1 \leq i \leq n)$ is assigned a *value* (r_i, c_j) by each stakeholder c_j $(1 \leq j \leq m)$. The overall revenue of a given requirement r_i $(1 \leq i \leq n)$ for the company is denoted by Eq.2.5.

$$Revenue_i = \sum_{j=1}^{m} (w_j \cdot value(r_i, c_j)) \tag{2.5}$$

In NRP, the requirements selection and optimisation solution is presented as a decision vector $\vec{x} = \{x_1, \cdots, x_n\} \in \{0, 1\}$ to determine the requirements that are to be selected in the next release. In this vector, x_i is 1 if requirement i is selected and 0 otherwise.

In 2002, Feather and Menzies [84] first proposed a multiple objectives NRP model. They formulated two objectives together (risk and cost) into a weighting-based single objective, and then applied Simulated Annealing [85, 86] to produce Pareto-front via iteratively adjusting the weight. Greer and Ruhe [29, 87] proposed a single objective genetic-algorithm-based framework EVOLVE to extended the NRP model to support incremental software release planning model (agile model). This framework takes the trade-off relationship among different releases into account, and provides the ability to optimally allocate requirements to increment releases by assessing and optimising the quality of candidate release plans while satisfying requirements dependency constraints as well as resource constraints. The authors argued that, offering a small set of solutions could provide decision maker with some additional flexibility. So, instead of offering just one solution, EVOLVE picks the L-best ($L > 1$) solutions, which achieve at least 95 percent of the maximum objective function value, to decision makers. The efficiency of EVOLVE framework was further improved by integrating Integer Linear Programming (ILP), a mathematical optimisation algorithm, with genetic algorithm to reduce the search space [88, 89].

In 2006, Harman et al. [90, 91] and Baker et al. [92] formulated the selection of candidate software components as a series of feature selection problems. Two meta-heuristic algorithms, Simulated Annealing and Greedy Algorithm, were used for solving this selection problem.

To deal with multiple criteria, the aforementioned approaches used a weighting-based approach to combine various objectives into a single objective. The underlying assumption of weighting-based approach is that decision makers have a fair idea of their subjective preferences on criteria. This assumption is unrealistic because the weighting-based approach introduces additional parameters into the problem, as well as decision makers' bias. Determining the proper weights when decision makers do not have enough knowledge about the problem is difficult. More specifically, because of the inappropriate weights, the concave portions of true Pareto-front curve may be missed in the presence of non-convex search spaces [93, 94, 95].

In order to resolve this problem, Zhang et al. [30] conducted a multi-objective formulations of the problem. They suggested to consider each criterion as a separate objective, and optimise all objectives simultaneously, rather than converting some of criteria as constraints or aggregating them together as one single objective. This model was termed the "Multi-Objective Next Release Problem (MONRP)". In this thesis, requirement value and requirement cost were considered as two separate objectives, and the problem was solved by Non-Dominated Sorting Genetic Algorithm II (NSGA-II), a multi-objectives evolutionary algorithm [96].

Finkelstein et al. [28, 97] investigated the trade-offs and conflicts in three notions of fairness among multiple stakeholders using a formulation of MONRP. Each notion of fairness forms an objective for multiple objective optimisation. Zhang et al. [98] formulated the conflicts between two release periods (today and tomorrow) based on MONRP model. Additionally, Zhang and Harman [99] introduced and evaluated the influences of five basic requirement dependencies among the requirements within MONRP optimisation. The five requirement dependencies are *And*, *XOR*, *Precedence*, *Value-related*, *Cost-related*. The results revealed that requirements dependencies could have a very strong impact on the MONRP optimisation process.

Durillo et al. [100] and Zhang et al. [101] studied the performance of some state-of-the-art multi-objective meta-heuristics for solving MONRP. In the former research, NSGA-II and MOCell [102] were studied. They found that NSGA-II can find better solutions in large instances than MOCell, while covering narrower range of solutions. The latter research compared the performance of NSGA-II with Two-Archive algorithm [103] for the analysis of multi-stakeholder trade-offs in MONRP. The results of their research showed that Two-Archive algorithm outperforms NSGA-II on the many objective problem. The performance of NSGA-II deteriorates significantly as the number of objectives increases.

Zhang et al. [104] comprehensive studied the different meta- and hyper-heuristic search algorithms for MONRP. Hill Climbing [105], Simulated Annealing [86] and NSGA-II were investigated together with hyper-heuristic [106] versions of each of these three meta-heuristics, which were denoted as HHC, HSA and HGA. Their work showed that hyper-heuristics more effective than meta-heuristics in terms of both solution quality and execution time. Furthermore, hyper-heuristics were considered scalable with respect to the number of requirements.

Though such heuristic search-based approaches might provide an approximate solution in a reasonable time and scale well, in special cases, decision makers need exact approach to guarantee the exactness of the results. In this thesis, using approximate approach to analyse and manage the uncertainty in NRP might introduce uncertainty from the nature of the algorithms used to optimise. Therefore, our approach is focusing on exact optimisation approach.

2.2.3 Exact Requirements Optimisation

Exact optimisation method is the optimisation method that can guarantee finding all optimal solutions. In principle, the optimality of generated solution can be mathematically demonstrated. Therefore, exact optimisation is also termed as mathematical optimisation. However, exact optimisation approach is impractical usually. The effort of solving an optimisation problem by exact optimisation grows exponentially with the problem size in general. For example, to solve a problem by a brute force approach, the execution time increases exponentially respect to the dimensions of the problem. Even so, there remain demands for seeking exact optimal solutions in requirements selection and optimisation.

The idea of applying exact optimisation approach to requirements selection and optimisation is similar with heuristic search-based requirements optimisation. The only difference is that, instead of using heuristic search-based optimisation algorithm, the search-based requirements selection and optimisation problem is tracked with exact search-based optimisation algorithm.

In 1998, Jung [107] introduced linear programming techniques to prioritise independent requirements. In his approach, the only objective is maximising the sum of selected requirements' value with the implementation budget constraint. Ignoring requirements independences is overly simplistic in practice. Carlshamre [108] designed and implemented a linear programming tool to handle requirements release planning while taking requirements independences into account.

Akker et al. [109, 110, 111] extended the work of Jung by introducing release planning into resource management. They sketched the context for the integrated problem mathematically, and developed an Integer Linear Programming (ILP) algorithm [112] based tool to solve it. They assumed that the optimal solution to the problem was the one offered maximal revenue against budgetary constraints in a given time period.

Moreover, Li [113, 114] described two integer linear programming models to further study the integrated software release planning and resource management problem. Given resource and precedence constraints, their first model advanced a schedule that the duration of development was minimised. On the other hand, the second model converted project duration as a constraint and maximised project revenues.

Harman et al. [22] proposed to use dynamic programming as an exact optimisation approach to attack the Next Release Problem. In order to manage the scalability of the approach and deal with multiple objective simultaneously, they introduced a variant of the Nemhauser-Ullmann's algorithm [115] as a solver for NRP. Nemhauser-Ullmann's algorithm treats each of the objectives as a separate objective function. This enables decision makers to obtain the Pareto front of non-dominated solutions. In addition, notwithstanding dealing with multiple objective, the execution time of this approach is still fast. The primary drawback of this exact optimisation approach is that Nemhauser-Ullmann's algorithm fails to take care of requirements interdependencies.

Leiter et al. [116] aimed to obtain the optimal set of requirements by a brute force approach. Although the exhaustive search will always find the optimal solution, it is inherently expensive and may not scale sufficiently to be more generally applicable. For a NRP model which consists of n requirements, there are 2^n solutions in the objective space.

Veerapen et al. [117] investigated the feasibility of solving single-objective and bi-objective NRP by Integer Linear Programming. In their approach, the requirements dependencies were formulated as constraints mathematically. To address the bi-objective problem, the epsilon-constraint ($\varepsilon-$constraint) method [118] was considered to be integrated into single-objective Integer Linear Programming approach. The authors compared the performance of the techniques against NSGA-II. The findings revealed that, 1) for the single-objective NRP, ILP could exactly solve large instances very quickly; 2) and for the bi-objective NRP, $\varepsilon-$constraint ILP could guarantee obtaining exact Pareto-fronts but inefficiently compared to heuristic search-based algorithm.

2.3 Uncertainty Handling

Uncertainty is ubiquitous and accompanies all events in the real world. It covers all fields of scientific studies, and is inevitable in many aspects of decision making [119]. According to the US National Research Council:

> “uncertainty is a general concept that reflects our lack of sureness about something or someone, ranging from just short of complete sureness to an almost complete lack of conviction about an outcome" [120]

The essence of uncertainty is the lack of complete knowledge at the time a decision must be made [24]. Uncertainty arises from different sources in various forms, and complicates and affects decision making [119]. It is worth mentioning that when the potential outcome of uncertainty as well as the odds of this outcome are known in advance, the uncertainty should be defined as risk [121].

Though it is hardly likely that uncertainty could be eliminated completely, it is worthwhile to identify and handle uncertainty to avoid unfavourable hazards [122]. To provide a confident final decision, there are two straightforward approaches to cope with uncertainty so far proposed in literature [123, 124]. One approach is to analyse uncertainty as a post-analysis method [24, 125]. Another one is robust optimisation, an approach that includes modelling and optimising the systems while taking uncertainty into account [37, 126].

2.3.1 Analysing Uncertainty in Requirements Optimisation

Conceptually, in order to analyse the impact of uncertainty in a decision-making or modelling process and provide an evaluation of the confidence in the model, two interrelated approaches to analyse uncertainty are found in the literature and practice [127, 128, 129]. These are sensitivity analysis, and uncertainty analysis.

Sensitivity analysis is performed in order to identify variations in results obtained from original and perturbed model input values [34, 127, 130]. It is the study of how the individual uncertainty of model input contributes to the overall uncertainty of a model output. This offers the knowledge about which one of the inputs drives the majority of the variance in the output. Meanwhile, uncertainty analysis attempts to explain the possible outcomes, together with their associated probability of occurrence [128]. Uncertainty analysis thus measures the overall uncertainty of the conclusions of the model [11, 128]. Compared to sensitivity analysis, uncertainty analysis concentrates on uncertainty quantification and propagation of uncertainty.

In the context of requirements optimisation, analysing uncertainty can help decision makers to identify the sensitive requirements, evaluate the robustness of the release plan in the presence of uncertainty, and thus better inform the decision making process. To study the uncertainty in the area of requirements optimisation, Harman et al. [35] used a local sensitivity analysis approach “One-At-a-Time" [36] to analyse the requirements sensitivity in NRP and MONRP. The requirements sensitivity was measured by perturbing one variable upward or downward at a time and keep all other variables fixed to baseline values to try out various ‘what-if’ scenarios.

However, their approach cannot fully explore the input space, since the probability of change, the extent of change, the interactions between requirements, as well as the simultaneous variation of requirements are not taken into account. In 2010, Al-Emran et al. [131, 132] performed probabilistic sensitivity analysis to evaluate the impact of requirements uncertainties in operational release planning and product release planning.

The uncertainties of properties were explained by probability distributions. Monte-Carlo Simulation-based analysis was then applied to pro-actively investigate the impact of uncertainty in estimates of requirement implementation effort and developers' productivity together with their Probability Density Function (PDF).

Although approximate approaches were scale well for requirements optimisation problems [100, 101, 104], these approaches do not guarantee that they find globally optimal solutions. This means that the additional uncertainty may be introduced due to the non-deterministic nature of the approximation algorithm. To avoid such information loss, in 2014, Harman et al. [22] applied a naive exact algorithm, a variant of the Nemhauser-Ullmann's algorithm [133], to study precise sensitivity analysis of NRP without considering requirements interaction.

2.3.2 Robust Optimisation in Requirements Optimisation

Though analysing uncertainty can evaluate how sensitive solutions are to possible estimation uncertainties, it cannot offer robust solutions by itself, based on a decision makers' degree of risk aversion. Hans-Georg and Bernhard [37, 134, 135] suggest to investigate uncertainties *during* the process of optimisation rather than using post-analysis. The term of *robust optimisation* was investigated by Soyster [136]. Robust optimisation is regarded as the approach that searching and optimising the solutions that are immune on production tolerances, parameter drifts, and model sensitivities [137].

Based on the optimisation approach, there are two main classes of robust optimisations [37]. One is robust optimisation using mathematical programming, while the other is robust optimisation using heuristic algorithms. The mathematical approach means adopting exact algorithms to solve convex constrained optimisation, such as linear programming and quadratic optimisation [138, 139, 140]. The main shortcoming of this approach is that it relies on formulating the optimised problem into a linear or quadratic model. With increases in the number of variables, it is a non-trivial task to construct an approximate mathematical model. Nevertheless, in the case that the value of uncertainty cannot be formulated into a mathematical expression and can only be obtained by simulation, the mathematical methodology is not applicable.

Heuristic robust optimisation measures uncertainty by 1) numerical techniques [141] or 2) simulation techniques [142]. Subsequently, the value of uncertainty is treated as one of the objectives for optimising. The former approach, which was referred as the deterministic approach to robust optimisation, has the same limitations with mathematical type robust optimisation since it still relies on strong mathematical assumption to measure the uncertainty. Additionally, the deterministic approach furnishes the “worst-case” robust solution which looks “too conservative”. Li [10] introduced a novel metric of uncertainty to simplify the mathematical complexity, and used it to guide multi-objective optimisation problems. In his work, the uncertainty of a parameter’s true value was represented as an interval, while solution uncertainty was represented as tolerance region (uncertainty size).

The latter approach, which was referred as simulation optimisation, does not need to model explicit complex mathematical information, but only the simulated uncertainty value use to compute the probability-based quality and robustness of the search point. The simulation optimisation usually uses the expected value, variance measure, and the probability of achieving the objectives to assess the robustness of a solution [143, 144, 145]. This approach offers a probabilistic guarantee, which allows the decision maker to flexibly choose a balanced trade-off between robustness and performance. Moreover, the simulation optimisation provides alternatives for decision maker, so that they can choose the corresponding level of probabilistic protection.

The applications and studies of robust optimisation can be found in other non-software-engineering research literature [10, 143, 144, 146, 147] but are seldom found in the requirements engineering literature. In requirements engineering, there are only three studies that apply robust optimisation to the requirements optimisation area:

In 2011, Heaven and Letier first proposed a search-based optimisation framework, which integrated with stochastic simulation, for guiding the choice of system design solutions on high level goals in quantitative Goal Models [148]. The simulated possible feature-response-time satisfaction rate was formalised as a fitness function.

Paixão and Souza were the first authors to introduce a robust optimisation framework for the NRP problem in 2013 [40]. They used the interval to model the uncertainties of requirements implementation cost, and defined a small population of scenarios to represent the uncertainty of requirement value. Thus, the uncertainty of a requirement's value is represented as a discrete variable. Each scenario is assigned a probability of occurrence. The desired level of robustness of decision makers is determined by a *control parameter*. Their robust NRP model tries to maximise the overall release solution value for all possible scenarios, while minimising the implementation cost of release solutions for worst case. Thereby, the outcome of their approach is a conservative and robust solution, which can avoid the impact of uncertainty in the worst case scenario.

In 2014, Leiter et al. applied statistical decision theory to illustrate the *expected information value* of model parameters [148] to offer further decision suggestions on requirements selection [116]. To overcome the limitations of approximate meta-heuristic algorithms, exhaustive search was adapted to explore the full solution space. The statistical *expected information value* was computed to explain the 'robustness' of a solution, and then treated as an/the objective for maximising.

2.4 Software Project Resource Allocation

A good upfront software planning is the foundation of a successful software development project [149]. It consists of determining the required activities, resources and goals as well as their proper management for completing the project. The state-of-the-art software release planning, which merely relies on requirements selection and optimisation model, provides overall direction for the software project and helps to define the deliverables (selected requirements), while the resource allocation is only addressed afterwards separately and sequentially [150, 151, 152, 153]. A survey of empirical literature on software project failures concluded that unrealistic deadlines, budgets overrun, poorly defined objectives, and a lack of project scheduling plan are the main reasons of project failures [154].

The software project resource allocation problem, which is also defined as software schedule planning, is the process of assigning available employees to software tasks for achieving the desired objectives, under certain constraints [155]. Modelling resource allocation is hard, it typically involves a large number of variables related to budget allocation, resource availability, staffing, developer skills and scheduling. There are also many constraints among these variables that increase problem complexity. For instance, available employees may not master all skills required for a given task.

Software project resource allocation is mainly driven by human behaviour [46, 156]. Different engineer backgrounds, trainings, and experiences of employees as well as project managers make the software project resource allocation unlikely to be optimal.

This problem has been studied for several decades [157, 158, 159, 160]. It has been modelled either as a constraint satisfaction problem, in which only the resource constraints are taken into account, or a constrained optimisation problem, in which different optimisation objectives are dealt with.

Several researchers [161, 162, 163] used AHP to tackle the software resource allocation problem. The matching degree between the task and the developers are assessed and prioritised by several stakeholders and developers. The main disadvantage of this method is that it is laborious to manage many variables and constraints simultaneously.

In 2001, Chang et al. [164] were the first to formulate the software project resource allocation problem as project scheduling problem and used a meta-heuristic approach to automate the software project management problem. They proposed a Software Project Management Net (SPMNet) approach for project scheduling and resource allocation. In their research, the project span is minimised, but constraints like the productivity and skills of developers are not considered. Duggan et al. [165] extended this problem to a multi-objective optimisation model, and modelled the competencies of developers by using a categorical variable with five levels; the level of competency expects as input productivities per day and numbers of defects. Acuna and Juristo [166] then connected labour psychology and software production by taking the behavioural competencies of the developers into account.

Ngo-The and Ruhe formulated the resource allocation problem as a constrained problem [150]. The overall goal of their model is, first, to maximise the utility value of releases with respect to a set of constraints over requirements and, second, to allocate the resources to the release plan obtained while meeting the resource constraints. In 2013, Ferrucci et al. [151] proposed to use different objectives, such as the project overrun risk. The idea is to manage the overtime risk and achieve project stability while focusing on project schedule minimisation.

Chapter 3

Simulation based Robust Next Release Problem Model

This chapter will introduce a Simulation based robust NRP model to mitigate the impact of requirements uncertainty. Requirements uncertainty is inevitable in software requirement engineering, especially in NRP. In order to deliver a quality software product respect to both robustness and performance, the decision makers have to balance the trade-offs among many aspects. Previous work applied post-analysis method to evaluate requirements uncertainty after optimisation. By contrast, our approach formulates requirements uncertainty as an objective to optimise, based on Monte Carlo Simulation. This offers decision makers an option to balance the trade-off between probabilistic robustness and performance.

3.1 Motivation

This chapter is the first work on Search-Based Software Engineering (SBSE) [21] to introduce simulation optimisation which utilises MCS to simulate the uncertainties of NRP as one of the objectives to guide the search to explore the moderately conservative robust Pareto-optimal front. In this chapter, we adopt a search-based optimisation technique with Monte-Carlo Simulation (MCS) to address requirements uncertainty and risk in the early stages of the software engineering development process. Our approach makes explicit the trade-off between requirements uncertainty/risk and traditional attributes of cost & revenue. It is assumed that the Probability Density Function (PDF) of features (in terms of cost and revenue) has been determined by a prior risk analysis [167].

The chapter builds novel formulations of requirements uncertainty to guide the NRP and presents robust Pareto-optimal solutions to decision makers. There are two notions of requirements uncertainty measurement introduced: size of uncertainty region [10] and the failure probability: the probability that actual cost exceeds a threshold. There are two definitions of robust solution considered in the chapter: 1) the solution's *payoff* (in terms of cost and revenue) has narrow fluctuation range, 2) the actual cost of solution has low probability to exceed the threshold. Each is 'robust' in the sense that it minimises the risk associated with a requirement choice. We compute the uncertainties of variables as probability distributions, and simulate them by MCS. We measure the two kinds of robustness, and explore the Pareto-front by using multi-objective evolution algorithm. Our approach can provide the solutions that balance the trade-off among revenue, cost, and robustness in a software project.

3.2 Problem Formulation

In this section, we describe the definition of the NRP and the metrics that capture requirements uncertainty in our approach.

3.2.1 Robust MONRP formulation

This chapter considers two types of robustness in MONRP. These two definitions of robust solution are "reduction of the uncertainty size", and "reduction of the probability

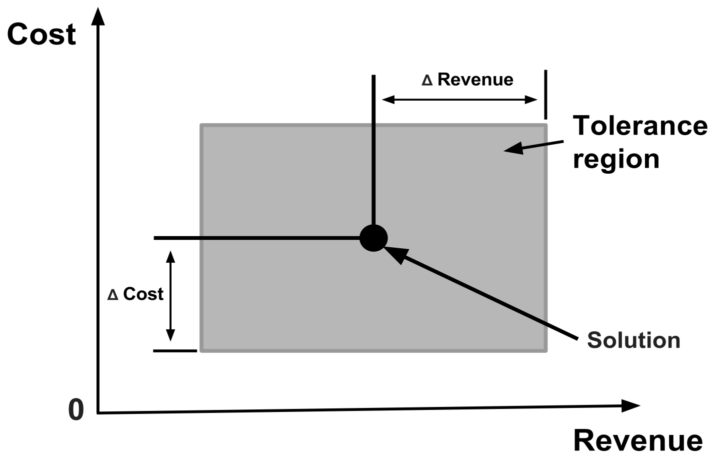


Figure 3.1: The tolerance region of a MONRP solution [10].

that actual cost exceeds a threshold" (named "failure risk reduction").

Uncertainty Size Reduction (MCNRP-US)

Uncertainty size is used to measure the tolerance region of the solutions of multi-objective optimisation problem in d dimensions (d is the number of the objectives). For example, in NRP, $\Delta cost_i$ is an acceptable fluctuation range of the cost of the ith requirement. The tolerance region consists of the confidence levels of each fitness value. The confidence level indicates the most likely fluctuation range of fitness values. Figure 3.1 illustrates a tolerance region for a NRP solution with cost and revenue objective functions. The shaded area is the tolerance region of the given solution. In Li's work [10], the standard deviation of each fitness value is used as confidence level. Hence, the tolerance region is composed of the standard deviation of each fitness value.

The size of tolerance region is presented by its normalised hyper-perimeter (Eq.3.1) and hyper-volume (Eq.3.2). To normalise the metric of each fitness value, we need to define fitness value referent with respect to each objective function.

$$perimeter(\vec{x}) = \sum_{k=1}^{d} \frac{2 \cdot \Delta fitness_k(\vec{x})}{referent\_fitness_k} \tag{3.1}$$

$$volume(\vec{x}) = \prod_{k=1}^{d} \frac{2 \cdot \Delta fitness_k(\vec{x})}{referent\_fitness_k} \tag{3.2}$$

where d is the number of objective functions. Therefore, all our fitness values lie in a normalised unit space. This facilitates comparison of Pareto-front using Euclidean Distance.

Besides, the weighted sum of these two metrics is defined as the *uncertainty size* and shown in Eq.3.3

$$Size(\vec{x}) = \alpha \cdot volume(\vec{x}) + \beta \cdot perimeter(\vec{x}) \tag{3.3}$$

Where $\alpha + \beta = 1$. In this work, we defined $\alpha = 0.5$ and $\beta = 0.5$.

We named this model as *MCNRP-US* (MCS for NRP-Uncertainty Size). The *MCNRP-US* consisting of the objective functions can be presented as follows (Eq.3.4, Eq.3.5, and Eq.3.6):

$$Maximise f_1(\vec{x}) = \sum_{i=1}^{n}(x_i \cdot Expected\_Revenue_i) \tag{3.4}$$

$$Minimise f_2(\vec{x}) = \sum_{i=1}^{n}(x_i \cdot Expected\_Cost_i) \tag{3.5}$$

$$Minimise f_3(\vec{x}) = Size(\vec{x}) \tag{3.6}$$

Failure Risk Reduction (MCNRP-R)

In our approach, the risk of a given solution is measured by the probability that the actual cost exceeds a threshold determined by the decision maker. In order to reduce the risk of budget overrun, our second approach minimises the probability that actual cost exceeds the budget (Eq.3.7).

$$Risk(\vec{x}) = \mathbb{P}(actual\_cost(\vec{x}) > \theta \cdot Expected\_Cost(\vec{x})) \tag{3.7}$$

Where θ is the percentage assigned by the decision maker (in our experimental study, we set $\theta = 150\%$), and $\mathbb{P}$ means Probability.

This model named *MCNRP-R* (MCS for NRP-Risk). The objective functions of *MCNPR-R* are shown as Eq.3.4, Eq.3.5, and Eq.3.8:

$$Minimise f_3(\vec{x}) \quad = \quad Risk(\vec{x}) \tag{3.8}$$

3.3 Optimisation Approach

Our approach contains two procedures: MCS and multi-objective optimisation. MCS enables us to simulate and evaluate a large number of scenarios effectively. The output of MCS process is used by the multi-objective optimisation process. The multi-objective optimisation is used to optimise multiple and possibly conflicting objectives simultaneously. In this chapter, we adopt the NSGA-II algorithm for optimisation.

Monte Carlo Simulation (MCS) [168] is a computerised mathematical technique to explore the range of possible outcomes of the model and the probability that these

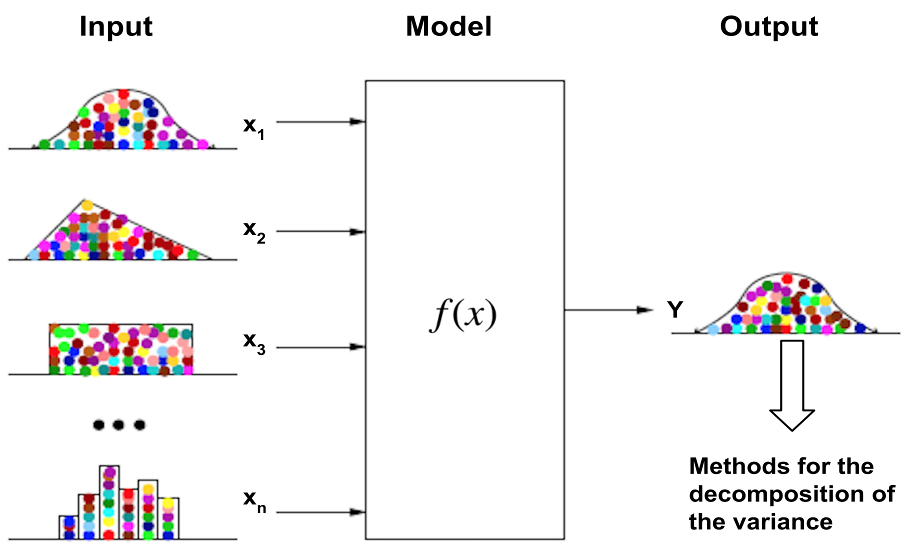


Figure 3.2: Overview of Monte Carlo Simulation approach [11]

outcomes will occur. The principle of MCS is to sample a large number of scenarios generated by substituting the probability distributions of model parameter values. It then calculates the results of model for all scenarios.

MCS generates a "scenarios database": an $s \times n$ matrix, *Simulations*, where s is the number of scenarios and n is the number of requirements. The element $Simulations[i, j]$ denotes the value of requirement j in ith scenario. The value includes the simulated revenue and the simulated cost of a given requirement. The number of scenarios was set to $10,000$, which means $10,000$ runs per fitness function evaluation. An overview of MCS approach showed in Fig3.2:

The well-known Non-dominated Sorting Genetic Algorithm-II (NSGA-II) was introduced by Deb et al.[96]. We use NSGA-II to provide a Pareto-front that captures the trade-off between cost & revenue and risk (assessed using MCS).

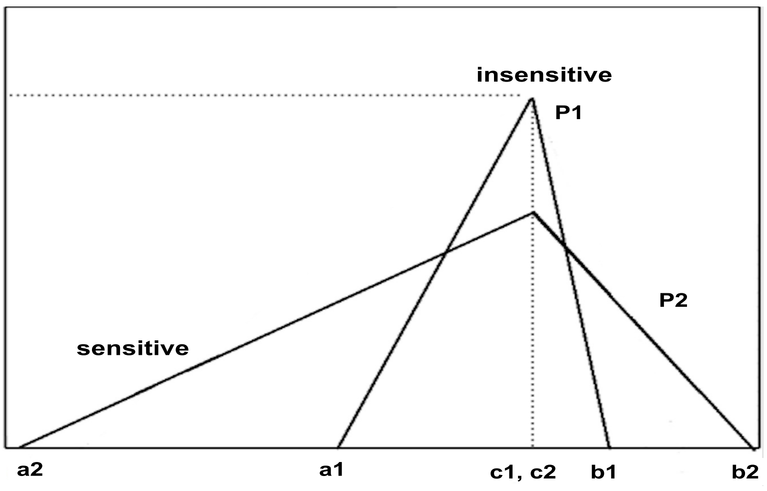


Figure 3.3: Illustration of the triangle probability distribution and the classification of sensitive and insensitive distributions [12]. c_1 and c_2 are the mode value of probability distribution P_1 and probability distribution P_2, respectively. a_1 and b_1 is the lowest value and highest of P_1 respectively while a_2 and b_2 is the lowest value and highest of P_2. P_1 is considered to be more 'stable' (insensitive).

3.4 Experimental Set Up

3.4.1 Data Sets

There are four synthetic data sets used in our experiments. The four data sets are synthetically constructed from one real project data set from Motorola [92]. The Motorola data set concerns a set of 35 requirements for hand held communication devices. Each requirement has the estimated implementation cost and expected revenue level. There is no uncertainty information for the cost and revenue of requirements. Our approaches can accept most kinds of probability distributions, such as uniform distribution, normal distribution, and discrete distribution. In this work, we simulated these uncertainties according to the "triangle probability distribution" illustrated in Figure 3.3.

Our four synthetic data sets represent four general scenarios (*S1* - *S4*), according to the degree of uncertainty about requirements' cost.

S1 Requirements for low cost have low probability to change (insensitive), while requirements for high cost have high probability to change (sensitive).

Table 3.1: Illustrative fragment of *S1* data

	Cost				Revenue		
NAME	Mode	Min	Max	Sensitivity	Mode	Min	Max
REQ 1	100.00	79.42	127.91	insensitive	3.00	0.65	3.32
REQ 2	50.00	15.08	53.51	insensitive	3.00	1.30	3.95
REQ 3	300.00	270.74	1154.15	sensitive	3.00	0.32	4.76
REQ 4	80.00	52.73	105.30	insensitive	3.00	1.31	5.50
REQ 5	70.00	42.00	78.77	insensitive	3.00	1.66	4.62
REQ 6	100.00	87.34	133.04	insensitive	3.00	1.01	4.19
REQ 7	1000.00	620.75	3671.35	sensitive	3.00	0.77	5.68

S2 Requirements for low cost have high probability to change (sensitive), while requirements for high cost have low probability to change (insensitive).

S3 Requirements for low cost have low probability to change (insensitive), while requirements for high cost have low probability to change (insensitive).

S4 Requirements for low cost have high probability to change (sensitive), while requirements for high cost have high probability to change (sensitive).

The ith requirement would be classified as low cost requirement if $cost_i < \frac{\sum_{j=1}^{n} cost_j}{n}$, otherwise high cost requirement, where n is the number of requirements. We define low probability as the possible change range is within 100%, while for high probability it is within 250%. The uncertainty of each cost is stochastically generated based on the above definitions. The uncertainty of revenue is randomly generated to have low probability (insensitive). A partial data of *S1* reported in Table 3.1.

3.4.2 Search Algorithmic Tuning

We base our algorithmic parameter and tuning on those used in previous work on MONRP [30]. We used binary encoding to represent the decision vector. The initial population size was set to 500. The algorithm was run for a maximum of $50,000$ function evaluations. The genetic operators used in our approaches are tournament selection (with tournament size of 5), single-point crossover (with crossover probability

0.8) and bitwise mutation (with the mutation probability $1/n$ where n is the number of requirements). The algorithm was executed 30 times for each data set, to cater for the stochastic nature of the algorithm.

3.4.3 Evaluation

Price of Robustness

In order to measure such loss between the proposed robust Pareto-front and original Pareto-front with regard to cost and revenue objectives, we utilised the "reduction factor" [169] to measure the "Price of Robustness". This factor measures the distance between two fronts [35]. To compute such distance, we defined $(A_1, \cdots, A_p)$ as the $front_a$ which contains p solutions, while $(B_1, \cdots, B_q)$ denotes the q solutions in $front_b$, where p and q are the number of solutions in $front_a$ and $front_b$ respectively.

The distance from solution A to solution B is computed by the normalised objective values and Euclidean Distance. In the case of "Price of Robustness", the distance between solution A and B is defined as:

$$Dis(A, B) = \pm\sqrt{\sum_{i=1}^{d}(A\_fit_i - B\_fit_i)^2} \tag{3.9}$$

Where d is the number of objectives. A_fit_i and B_fit_i are the normalised ith objectives value of A and B, respectively.

The distance from solution A to geometrically closest solution B on $front_b$ is presented as the distance from solution A to $front_b$ (Eq.3.10).

$$Dis(A, front_b) = Dis(A, B) \tag{3.10}$$

Therefore, the distance from $front_a$ to $front_b$ is the mean value of the distance from every solution on $front_a$ to $front_b$.

$$Dis(front_a, front_b) = \frac{\sum_{i=1}^{p} Dis(A_i, front_b)}{p} \tag{3.11}$$

Probabilistic Sensitivity Analysis

To measure the amount of robustness improvement achieved by our robust optimisation approach, we performed a probabilistic sensitivity analysis. Firstly, we used the same sampling technique to simulate the uncertainties of data. After that, we adopted robustness formulations defined in this chapter to calculate the robustness of Pareto-optimal solutions generated by traditional approach.

3.4.4 Research Questions

In order to evaluate the effectiveness and usefulness of the approaches, we carried out two experimental studies to assess the efficiency of the approaches and four scenarios to evaluate its usefulness. In the experiments, we compared the results obtained from our approaches with the ones obtained from MONRP, and formalised one research question. The question is whether the proposed approaches can provide more robust solutions to decision makers with less sacrifice? This question formulated into three more detailed sub-questions (***RQ1***, ***RQ2***, and ***RQ3***).

Additionally, to aid the decision making support before preforming such professional tools, this chapter also investigated the correlations between attributes of a requirement and its inclusion in solutions on the Pareto-front. This is formulated into the fourth question ***RQ4***.

RQ1 Do the proposed two kinds of robust optimisation improve robustness? This question will be answered by analysing and comparing the robustness of solutions which were generated by our approaches and the original MONRP.

RQ2 How much "Price of Robustness" would be paid for the proposed robust optimisation approaches? We will answer this question by calculating the distance between the Pareto-front obtained from our approaches and those obtained from the original MONRP. The distance was used to measure the loss in *pay-off*.

RQ3 How similar are the Pareto-fronts produced by our new approaches and the one produced by traditional MONRP? We computed and ranked the proportion of requirements being selected in solutions on the Pareto-front. Then we used Kendall's τ_B correlation coefficient to statistically investigate the degree of similarity between the rankings of requirements included in solutions on the Pareto-front to answer this question.

RQ4 Which attributes of a requirement are correlated with inclusion in solutions on the Pareto-front?

We performed an intuitive analysis to answer the ***RQ1*** and ***RQ2***, while more statistical analysis to answer the ***RQ3*** and ***RQ4***.

3.5 Experimental Results and Analysis

This section presents two different robust models and the results of applying these two models on four synthetic problem instances. Two experiments were conducted and the illustrations of results are presented in Figures 3.4 and 3.5 (Figures 3.4a, 3.4b, 3.4c, 3.4d, and Figures 3.5a, 3.5b, 3.5c, 3.5d), for *E1* & *E2* respectively. The two experiments, *E1* & *E2*, are described below:

E1 The first experiment aims at evaluating the *MCNRP-US* approach and the "Price of Robustness" of this approach, when the decision maker expects to obtain robust solutions within a defined fluctuation range.

E2 The second experiment evaluates the *MCNRP-R* approach and its "Price of Robustness" for the situation in which the decision maker would like to acquire robust solutions which have a low risk of budget overrun.

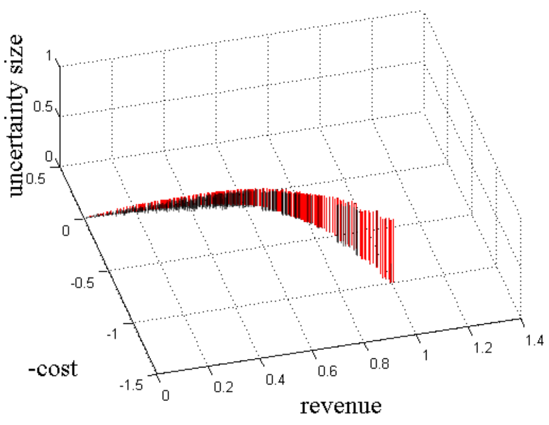
In order to compare our proposed approach to the traditional MONRP approach, the Pareto-fronts of proposed approach are presented by dark black patterns and traditional ones by grey (red when viewed in colour) patterns. This selection quantitatively analysis and answer the *RQ3* and *RQ4* as well.

3.5.1 Experiment One (E1)

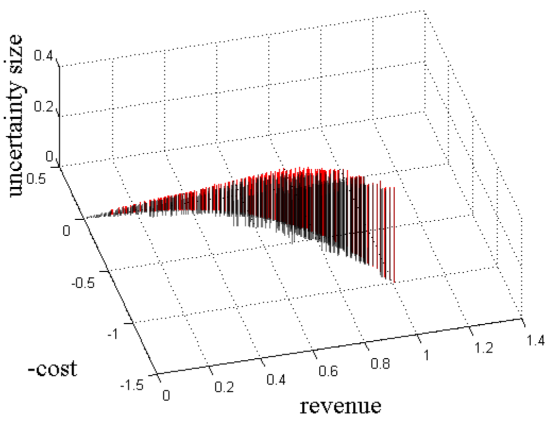
In *E1*, the *uncertainty size* of a solution is taken into account. The results of *E1* are shown in Figures 3.4a, 3.4b, 3.4c, and 3.4d corresponding to scenarios *S1*, *S2*, *S3*, and *S4*, respectively. The figures illustrate the three-dimensional Pareto surface. Each bar represents a solution on the Pareto-front. The location of each bar in the *cost-revenue* plane presents the *cost* and *revenue* of the solution respectively. The height of the bar shows the *uncertainty size* for each solution.

From the results of *E1* for *S1*, *S2*, *S3*, and *S4*, we observe that, as the overall fulfilled *cost* increases, the *uncertainty size* of solution also increases. We also observe that there are minor differences between the Pareto-fronts of *MCNRP-US* and the traditional approach in *S1* and *S4* (Figures 3.4a and 3.4d), while there are larger differences between *S2* and *S3* (Figures 3.4b and 3.4c). High cost requirements naturally have more impact on solution sensitivity than low cost requirements [35]. Requirements with high cost are stable in *S2* and *S3*, and the proposed first approach tends to select the "stable" solution rather than the solutions just have good economic performance but "unstable".

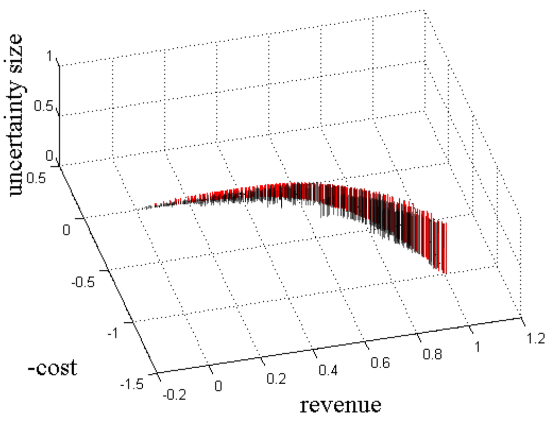
Table 3.2 presents the results of probabilistic sensitivity analysis for *E1*, the "*Price of Robustness*" for "*MCNRP-US*" approach, and how much robustness with regarded to *uncertainty size* improved by applying this approach.



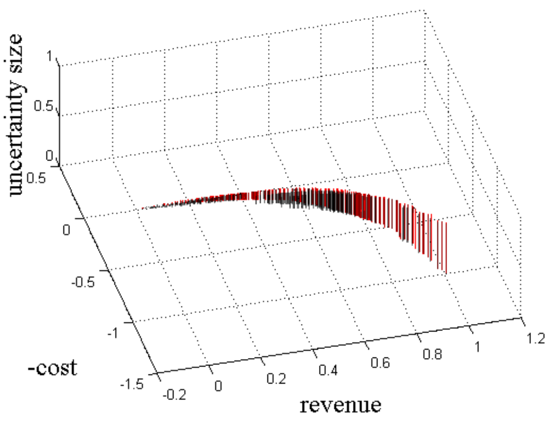
(a) The results of *E1* in *S1*



(b) The results of *E1* in *S2*



(c) The results of *E1* in *S3*

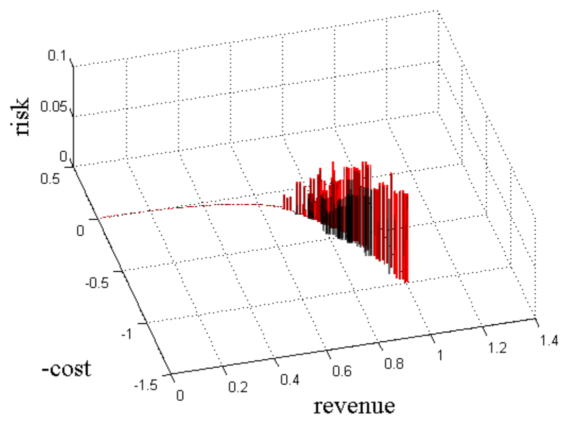


(d) The results of *E1* in *S4*

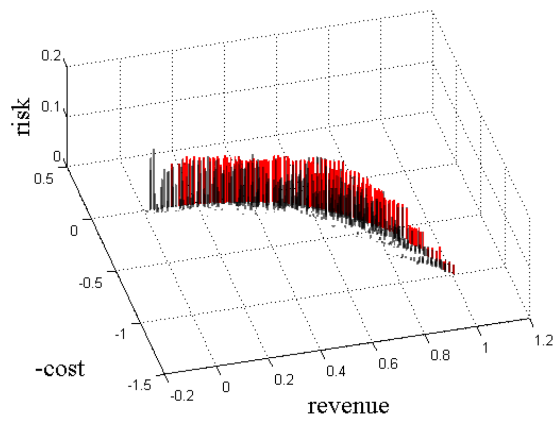
Figure 3.4: The Pareto-front of *MCNRP-US* and Original Approach

Based on the results in this table we answer ***RQ1*** and ***RQ2*** (for *MCNRP-US*) as follows: On average, the *MCNRP-US* generates more robust solutions with respect to *uncertainty size*. The overall improvement is not large: after normalizing *cost* and *revenue*, the magnitude of standard deviation of *cost* and *revenue* is small, so the magnitude of *uncertainty size* is small. Even so, it is interesting that the robustness improvements for *S1* and *S4* (22.78% and 14.65% respectively) are better than the improvements for *S2* and *S3* (2.54% and 7.19% respectively).

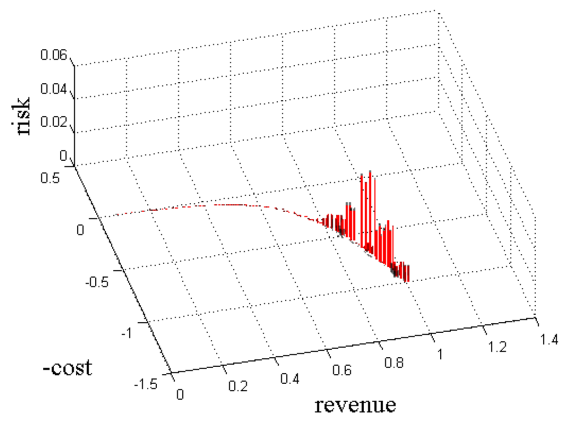
Although the improvement of *MCNRP-US* is not dramatic, it pays a little as the "*Price of Robustness*". Therefore, we conclude that applying our *MCNRP-US* approach, the decision maker pays a small price to obtain a more robust Pareto-front, whose solutions have smaller *uncertainty size*.



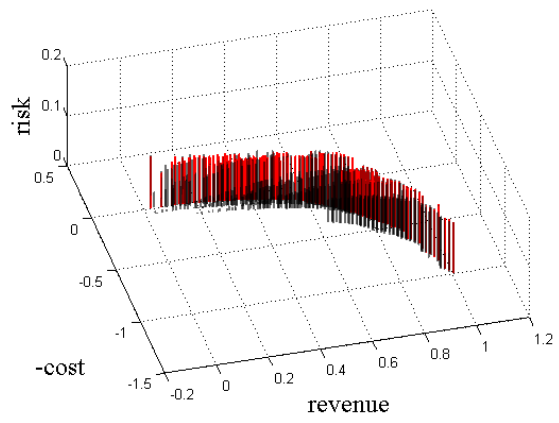
(a) The results of *E2* in *S1*



(b) The results of *E2* in *S2*



(c) The results of *E2* in *S3*



(d) The results of *E2* in *S4*

Figure 3.5: The Pareto-front of *MCNRP-R* and Original Approach

3.5.2 Experiment Two (E2)

The results of *E2* are plotted in Figures 3.5a, 3.5b, 3.5c, and 3.5d. In *E2*, the *risk* was considered as a third objective.

From the results of *E2* in *S1*, *S2*, *S3*, and *S4*, a general trend is observed: the degree of *risk* increases as overall *cost* increases. However, there is an interesting observation in Figure 3.5b. The *risk* is inversely proportional to *cost*. The reason for this phenomenon is that the *risk* is directly proportional to the stability of the probability distribution of *cost*. The more stable the probability distribution is, the lower risk there will be. In *E2*, there are some other interesting observations: According to the results, we observe that the obtained "robust" Pareto-fronts are quite close to those obtained from original

Table 3.2: The Robustness & Comparison of the *MCNRP-US* Approach and the Traditional Approach

	S1	*S2*	*S3*	*S4*
MCNRP-US	0.1531	0.1558	0.1850	0.1290
Original Approach	0.1983	0.1599	0.1993	0.1511
Price of Robustness	0.0110	0.0201	0.0154	0.0102
Robustness Improvement	22.78%	2.54%	7.19%	14.65%

Table 3.3: The Robustness & Comparison of the *MCNRP-R* Approach and the Traditional Approach

	S1	*S2*	*S3*	*S4*
MCNRP-R	0.0396	0.0404	0.0109	0.0591
Original Approach	0.0500	0.0755	0.0132	0.0888
Price of Robustness	0.0036	0.0253	0.0003	0.0285
Robustness Improvement	20.82%	46.49%	17.70%	33.37%

MONRP in *S1* and *S3*, while there are a big gap in *S2* and *S4*. This is because the probability distribution of high *cost* is unstable in *S2* and *S4*.

Table 3.3 shows that the robustness with regards to risk can be noticeably improved by the *MCNRP-R* approach compared to traditional approach. Moreover, the payment (*Price of Robustness*) is low.

As an overall answer ***RQ1*** and ***RQ2*** (for MONRP-R) we find that we can achieve an improvement of at least 18% in robustness with only a little change in 2D cost-revenue Pareto-front (maximum 0.0285 in a unit space). That is, the penalization due to robustness is very small for all scenarios, which qualifies the effectiveness of *MCNRP-R* approach.

3.5.3 Statistical Analysis

To answer the ***RQ3*** and ***RQ4*** statistically, Kendall's τ_B correlation coefficient τ_B is used to quantitatively analyse the correlation between and within the approaches. Table 3.4 shows Kendall's τ_B correlation coefficient and corresponding p-value calculated for the relation between the paired approach (MONRP and MONRP-R, MONRP and

Table 3.4: The Correlation of Rankings of Requirements

		MONRP&R	MONRP&US	US&R
S1	τ_B	0.9361	0.7345	0.7311
	p-value	< 0.000	< 0.000	< 0.000
S2	τ_B	0.8646	0.7872	0.8756
	p-value	< 0.000	< 0.000	< 0.000
S3	τ_B	0.9655	0.7233	0.7311
	p-value	< 0.000	< 0.000	< 0.000
S4	τ_B	0.8646	0.8713	0.8387
	p-value	< 0.000	< 0.000	< 0.000

In this table, R means MONRP-R, and US means MONRP-US.

MONRP-US, and MONRP-R and MONRP-US) with regard to each scenario. If all solutions in Pareto-front agree on a requirement to be selected, the requirement is said to be "closed" [116]. Here, we generalise this notion of "closed" decision to investigate correlations between degrees of "closedness".

The Table 3.4 reveals that there are existing strong correlations between the rankings of requirements produced by each approach on each scenario. All τ_B coefficients are greater than 0.7, and p-values are very close to zero. This confirms that the rankings of requirements produced by each approach are similar to each other.

Hence, the Pareto-fronts on Cost-Revenue dimension generated by each approach are similar to each other. We further observe that the correlation is stronger between MONRP and MONRP-R than MONRP and MONRP-US. This answers ***RQ3***.

In order to answer ***RQ4***, Table 3.5 uses Kendall's τ correlation analysis to statistically describe the correlation between the attributes of requirements and the rankings of requirements. The results reveal that, in general, the requirement's *Revenue-to-Cost* ratio and *Cost* have strong monotonic correlation with its likelihood of inclusion, while its *Revenue* is uncorrelated. The requirement's *Revenue-to-Cost* ratio is the most strongly correlated.

Table 3.5: The Correlation between the Attributes of Requirement and its Ranking

	Cost		Revenue		R/C	
MONRP	τ_B	$p-$Value	τ_B	$p-$Value	τ_B	$p-$Value
S1	-0.7748	< 0.000	0.0723	0.55358	0.9597	< 0.000
S2	-0.7569	< 0.000	0.1413	0.23846	0.9521	< 0.000
S3	-0.7771	< 0.000	0.074	0.54138	0.9521	< 0.000
S4	-0.7704	< 0.000	0.1346	0.26185	0.9554	< 0.000
MONRP-US	τ_B	$p-$Value	τ_B	$p-$Value	τ_B	$p-$Value
S1	-0.5899	< 0.000	0.0824	0.49827	0.721	< 0.000
S2	-0.6034	< 0.000	0.2336	0.049495	0.7714	< 0.000
S3	-0.5832	< 0.000	0.0924	0.44599	0.7008	< 0.000
S4	-0.6807	< 0.000	0.1765	0.14052	0.8521	< 0.000
MONRP-R	τ_B	$p-$Value	τ_B	$p-$Value	τ_B	$p-$Value
S1	-0.7244	< 0.000	0.1092	0.3661	0.8958	< 0.000
S2	-0.6807	< 0.000	0.1966	0.09972	0.8555	< 0.000
S3	-0.758	< 0.000	0.0924	0.44599	0.9294	< 0.000
S4	-0.674	< 0.000	0.1899	0.11213	0.8521	< 0.000

In this table, Cost is the Expected Cost, Revenue is the Expected Revenue, and R/C is the Expected *Revenue-to-Cost* Ratio.

3.6 Summary

In this chapter, we introduced an MCS-based robust optimisation approach for requirement analysis and optimisation.

We introduced two notions of requirement uncertainty measurements for NRP. According to the experiments upon which this chapter reports, the proposed two robust MONRP approaches (*MCNRP-US* and *MCNRP-R*) overcome the limitation of the traditional MONRP approach which underestimates (or even hides) requirements uncertainty. These allow the decision maker to choose different approaches for controlling different types of requirements uncertainty and different levels of probabilistic guarantee for robustness, while retaining the performance of traditional solutions. The *MCNRP-US* offers decision makers a way to control the fluctuation range of *payoff* for solutions. The *MCNRP-R* model helps decision makers to explore solutions with lower *risk* of budget overrun.

We found that MONRP-R decisions are more closely correlated to traditional MONRP decisions regarding requirement choice, than MONRP-US. We also find that, while cost is closely correlated to inclusion of a requirement in the Pareto-front, revenue is not.

However, our approach in this chapter has relied upon meta-heuristic algorithms with the notable drawback that the algorithm itself, being a randomised algorithm, contributes to the overall uncertainty. This is a problem we address in the next chapters by using an exact algorithm.

Chapter 4

The Value of Exact Analysis in Next Release Problem

As described in Chapters 2 and 3, previous uncertainty management for NRP relied solely upon (non-deterministic) randomised meta-heuristic algorithms. In this chapter, we propose a decision support framework *METRO* for the Next Release Problem to manage algorithmic uncertainty and requirements uncertainty. An exact NRP solver (*NSGDP*) is at the heart of the *METRO* to eliminate interference caused by existing approximate NRP solvers. Furthermore, *METRO* takes account of risk in the decisions suggested, rather than simply report upon its possible pernicious effects. This explicitly helps decision makers to understand and make the trade-off between uncertainty/risk and conventional objectives of cost & revenue based on information so far.

4.1 Motivation

To address uncertainty in Next Release Problem, previous work by Heaven et al. [148], Paixão et al. [40], and Li et al. [1] relied solely upon (non-deterministic) randomised meta-heuristic algorithms. These previously proposed meta-heuristic algorithms can only find reasonable approximate solutions, and lead to information loss. While this is acceptable in general, for the specific problem of handling risk we face here, it is important for the decision maker to know that all uncertainty derives from the problem itself and not from the algorithm used to tackle it. Moreover, they reported robust NRP solutions, but yet they did not guide decision makers to select the solutions from thousands of candidates.

In order to aid decision support in the early stage of software engineering over simply reporting the results explored, we develop a decision support framework for Next Release Problem (called *METRO*), which utilises both a simulation-based robust optimisation approach and a point-based optimisation approach. Our approach adopts an exact algorithm combined with a Monte-Carlo Simulation (MCS) to deal with algorithm non-determinism and capture requirements uncertainty. In this manner, our approach eliminates the algorithmic uncertainty, and explicitly helps decision makers to understand and make the trade-off between uncertainty/risk and conventional objectives of cost & revenue based on information so far.

To handle the requirements uncertainty, *METRO* takes into account the quantified cost and revenue of requirements as well as the Probability Density Function (PDF) of uncertainties associated with these requirement attributes (cost and revenue). With the aid of PDF of uncertainties, *METRO* utilises MCS to simulate uncertainties in terms of their impact on specific objectives, and then a set of solutions will be picked by exact optimisation approach. *METRO* quantitatively analyses the outcomes of optimisation, and interprets the findings through a set of visualisations. These visualisations depict the tension between two different objectives regarding the objective space, and illustrate the characteristics of requirements regarding the design space. This information allows decision makers to understand the impact of requirements uncertainty and determine the requirement priority.

The chapter's primary contribution is to introduce exact multi-objective dependence-respecting NRP solver to deal with algorithmic uncertainty and requirements uncertainty. More specifically, the following contributions are made:

1. The first contribution is about eliminating algorithmic uncertainty. We develop an exact NRP optimisation solver *NSGDP* in our framework *METRO*. Our experimental studies reveal that, with the aid of *NSGDP*, the decision maker can avoid information loss (without which he or she will lose up to 99.95% of optimal solutions and will make up to 36.48% inexact requirement selection decisions). Furthermore, the execution time of *NSGDP* is better than NSGA-II: On average, *NSGDP* takes $0.37s$ (without accounting for requirements uncertainty), and $35.33s$ (when taking requirements uncertainty into consideration). By contrast, NSGA-II takes more than 10 minutes, whether or not requirements uncertainty is taken into account.

2. The second contribution is our introduction of an approach to cater for requirements uncertainty. *METRO* investigates the difference between the optimal-yet-risky solutions and robust-yet-suboptimal solutions. Two indicators are used: expected risk premium and risk reduction. Our experimental results show that, developing software project based on optimal-yet-risky release plan rather than robust-yet-suboptimal release plan, may suffer up to 10.09% probability of overrunning more than 150% budget but gaining less than 0.39 expected risk premium.

3. The third contribution is that the proposed framework can better support decision makers in understanding the requirements selection. A series of quantitative techniques is provided for highlighting the characteristics of requirements and solutions. The difference of requirement selection probability between two NRP approaches is analysed and presented in a stacked bar plot. We found that risk-aware *sNRP* approach is more likely to include the requirement with low uncertainty than *pNRP* approach does (Kendall's τ_B up to -0.675). *METRO* clusters requirements according to design space proximity rather than objective space proximity.

4.2 Background

This section describes the techniques used in this chapter. Firstly, a RIM model: Conflict Graph is presented to show how to construct requirement dependencies, and then Nemhauser-Ullmann Algorithm, which is the exact NRP solver used in our framework, is introduced.

4.2.1 Next Release Problem with Conflict Graphs

In practice, there may be different constraints between the requirements in NRP. These constraints describe the relationships between the various requirements [170]. Mutual exclusion is a typical constraint, which denotes at most one of the two mutually exclusive requirements can be selected simultaneously. In graph theory, conflict graphs are usually used to construct such logical relations between objects. More precisely, a conflict graph G contains a set of vertices and edges between two vertices (Eq. 4.1):

$$\begin{aligned} G =& (V, E) \\ V =& \{v_i\} \\ E =& \{(v_i, v_j) | \text{The } v_i \text{ and } v_j \text{ is mutually exclusive}\} \end{aligned} \tag{4.1}$$

where V is the vertex set, in which each vertex represents a distinct object, and E is the edge set, in which each edge means two connected vertices exclude each other (thus cannot be selected at the same time). The isolated vertices denote that those vertices can be selected with every other isolated vertex at the same time.

Conflict graphs have been successfully applied to Knapsack-like Problems with Conflicts [171, 172, 173], which are strongly *NP-hard* in general. Moreover, in 2009, Pferschy and Schauer [174] proved that forming Knapsack-like Problem with Conflict Graph (KCG) in the search tree can carry forward fully polynomial time approximation schemes (FPTAS). Accordingly, it is promising to model NRP in the form of the conflict graph, and then reconstruct it to search tree. We would interpret how to construct NRP with conflict graph to the search tree exemplified by the general knapsack-like problem.

To reconstruct a knapsack-like problem from a conflict graph data structure to a search tree data structure, the first step is processing G in depth-first-search. Then picking a constrained vertex v_i (conflicting with vertex v_j) to distinguish the problem into two sub-problems from top-down:

- Necessarily including v_i in the sub-problem, and excluding v_j
- Always excluding v_i in the sub-problem, and keeping the decision concerning v_j open.

Mathematically, the process of constructing problem tree is presented as follows.

Definition 1 *$G \setminus v$ means subtracting a vertex $v \in V$ from graph G: $G \setminus v = (V', E')$, where $V' = V - \{v\}$ and $E' = \{(v_i, v_j) \mid (v_i, v_j) \in E, v_i \in V', v_j \in V'\}$.*

Definition 2 *For graph $G = (V, E)$ and a vertex $v \in V$, $C(v)$ represents a set of objects including v and those have constraints with v: $C(v) = \{u \in V \mid u = v \text{ or } (u, v) \in E\}$.*

When all leaves of the root problem tree have no edge at all ($|E| = 0$), the problem is solved bottom up. The procedure of solving KCG is described in Algorithm 1. In Algorithm 1, if G has no constraints at all ($|E| = 0$), then solve the problem using dynamic programming, otherwise the problem G is divided into two sub-problems $G \setminus v$ and $G \setminus C(v)$ with respect to a chosen constraint v. The former one assumes v is not selected in all of the solutions and the latter one assumes v selected, thus all the objects that conflict with it cannot be selected in the final solutions and are removed from the problem as well ($C(v)$ contains the objects that have connections with v). After these two sub-problems are solved recursively, the algorithm sets $x_v = 1$ in all of the Pareto solutions for the second sub-problem, since v is assumed to be selected in the second sub-problem. At last the algorithm merges these two sets of non-dominated solutions together and removes those being dominated to form the Pareto solution set for the problem G.

Algorithm 1 Solve KCG $S = Solve(G)$

```
Require: conflict graph $G = (V, E)$
  if $E = \varnothing$ then
    return $KnapsackProblemSolver(G)$
  end if
  Pick $v \in V$ that has an edge in $E$
  $S_0 = Solve(G \setminus v)$
  $S_1 = Solve(G \setminus C(v))$
  for all $\vec{s} \in S_1$ do
    set $v$ selected: $s_v = 1$
  end for
  return $S = Merge(S_0, S_1)$
```

4.2.2 Nemhauser-Ullmann Algorithm

To solve NRP exactly, we build an exact NRP solver *NSGDP* using the Nemhauser-Ullmann algorithm to solve specific instances in a decision tree solution space. The Nemhauser-Ullmann algorithm is a dynamic programming algorithm proposed by Nemhauser and Ullmann in 1969 [115]. It is a non-dominated sorting based multi-objective exact optimisation algorithm to enumerate the Pareto set of knapsack-like problems [175]. However, it has an obvious drawback. It cannot deal with knapsack-like problems with constraints. Harman et al. [22] employed it to materialise an exact NRP solver that focuses on NRP with the independent requirement.

For a given NRP problem with n requirements, the Nemhauser-Ullmann algorithm starts with considering 0 requirements, and then iteratively inserts the next requirement i into every solution in the Pareto-front $P(i-1)$, where $P(i-1)$ denotes the Pareto-front of first $i-1$ requirements. After merging two solutions to set $P'(i) = P(i-1) \cup (P(i-1)+i)$, the Nemhauser-Ullmann algorithm uses non-dominated sorting (the so-called *staircase* function) [96] to compute the Pareto-front of first i requirements $P(i) = \text{Non-dominated-Sorting}(P'(i))$. $P(i-1)+i$ denotes the set of solutions that is obtained by setting the ith requirement to be selected for all solutions from $P(i-1)$. Following these steps, the final result $P(n)$ is computed inductively. Summarising, the Nemhauser-Ullmann algorithm is formalised by Algorithm 2:

Algorithm 2 Nemhauser-Ullmann algorithm for NRP

```
Require: A set of $n$ requirements
  for $i = 1, .., n$ do
    $P'(i) = P(i-1) \cup (P(i-1)+i)$
    $P(i) = \text{Non-dominated-Sorting}(P'(i))$
  end for
  return $P(n)$
```

4.3 Simulation based NRP Decision Analysis Framework METRO

Multi-Objective NRP approaches produce a Pareto-front which may contain a large number of solutions. It is laborious for engineers to understand and identify one solution from thousands of candidate solutions, especially taking uncertainty into account. To aid decision makers to tackle the latent information within optimal solutions, this chapter proposes a simulation NRP decision analysis framework, *METRO*. Instead of merely generating the optimal solutions themselves, *METRO* statistically analyses the optimal solutions, mines information from them, and provides the insight of these solutions. The main processes of *METRO* (Figure 4.1) are:

1. Pre-processing the requirements dependencies.

2. Adopting *sNRP* and *pNRP* to model the requirements optimisation problem separately, and then using exact optimisation solver to produce the optimal solutions.

3. Statistically analysing results of two solutions, and visualising the refined information as well as the implicit requirement pattern for decision processes.

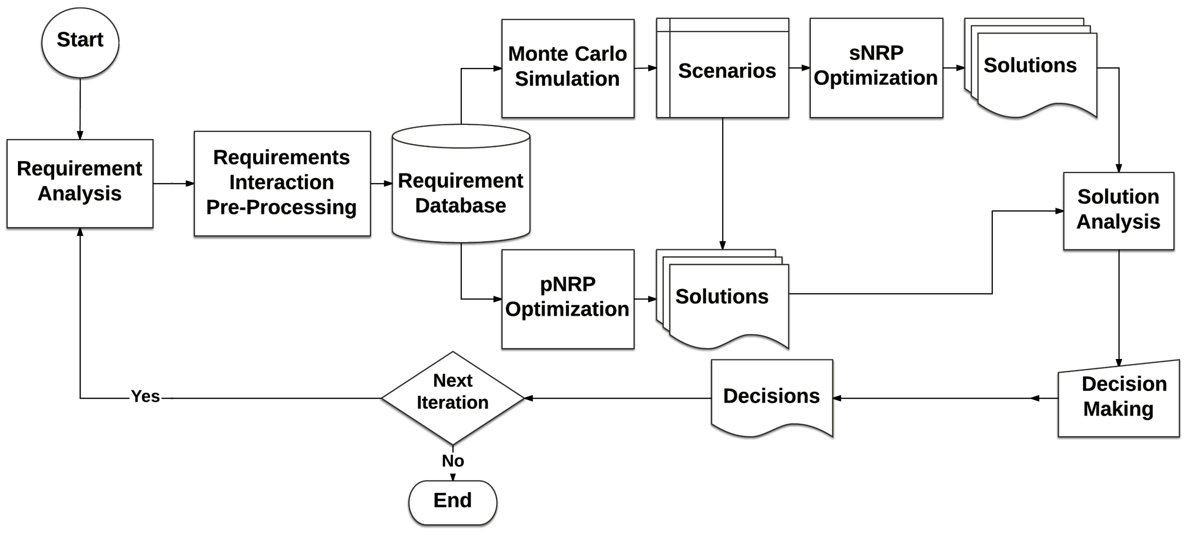


Figure 4.1: NRP Decision Analysis Framework: *METRO*

4. Performing this analysis in the next iteration.

4.3.1 Requirements Interaction Pre-Processing

Requirements may depend on each other [176]. Some requirements may interact with other requirements due to the constraints or limitations that come from techniques, or business related issues. Requirement implementations may be mutually exclusive, or should be fulfilled together on the basis of their interactions. Failure to consider requirement interactions, may yield infeasible decisions.

Requirements Interaction Management (RIM) has been proposed to analyse and manage the dependences among requirements [170, 177]. In NRP, RIM involves at least two types of interactions (*And*, and *XOR*). The *And* dependence between two requirements means the selections of requirements have to be in the same release. On the other hand, the selection of two requirements which have *XOR* dependence are "repelling" each other because these two requirements are mutually exclusive. Table 5.1 presents the mathematical expressions of these interaction.

Although the original RIM defines the dependencies between requirements, RIM can be

Table 4.1: Requirement Interactions. The sets ξ, and φ present the interaction types *And*, and *XOR*, respectively. The set $\xi \cap \varphi = \varnothing$.

And	$\forall (i, j) \in \xi, x_i = x_j$
XOR	$\forall (i, j) \in \varphi, x_i \wedge x_j = 0$

simplified to enable fast execution and better convergence. In our proposed approach, the *And* dependence satisfies $\forall (i, j) \in \xi, x_i = x_j$. By transitivity, if $(i, j) \in \xi$ and $(j, k) \in \xi$, then $x_i = x_j = x_k$. Therefore, a super-requirement $Req_{i,j,k}$ can be used to represent requirement i, j, and k in a single decision variable. This simplification, reduces the computational cost for requirements constraint handling and the search space within which we seek solutions.

4.3.2 Exact NRP optimisation Solver

After requirement data pre-processing, decision makers have to decide which requirements are critical and should be included in the next release of system under budget constraints. For this step, the objectives and formulations should be clearly defined. The conventional criteria for NRP is maximising the expected revenue and minimising the expected release cost. Decision makers can also define other criteria, such as the satisfaction degree of customers, the fairness level among different stakeholders [28], and the utility of release.

Taking uncertainty into account, project risk could be an extra objective to optimise. In a software project, the project risk is related to future events that may have undesired consequences for the project [178]. Project risk could include budget overrun, the number of requirements becoming inflated, departure of a key person, and productivity failing to meet expected estimates [179, 180]. There are existing risk analysis methods that identify and elicit these software project uncertainties quantitatively and use probability distributions to represent the uncertainty [181]. After these uncertainties have been elicited, our framework formulates the fitness function to optimise project risk.

In NRP, such multi-objective decision support problems can be investigated using a multi-objective optimisation algorithm. In order to ensure that the variations in results do not come from the stochastic nature of the algorithm, we design an exact NRP optimisation solver: **N**on-dominated **S**orting Conflict **G**raph based **D**ynamic **P**rogramming algorithm (*NSGDP*). The *NSGDP* uses the Nemhauser-Ullmann algorithm, an exact dynamic programming algorithm, as the core NRP solver, and augmented by Conflict Graph to deal with the requirements interaction. Firstly, the NRP problem with constraints is modelled into Conflict Graph. Then, the root problem is broken down into sub-problems according to Algorithm 1 until there is no constraint in sub-problems. Lastly, Algorithm 2 is used to solve NRP without constraint directly. It is worth mentioning that, our algorithm is applicable to, not only the case we study in this chapter, but also any kind of knapsack-like problems with exclusive conditions.

To further improve the performance of our algorithm, we introduce an array to store the processing order. This is because, when a graph G is divided to two graphs $G \setminus v = (V_0, E_0)$ and $G \setminus C(v) = (V_1, E_1)$, $G \setminus C(v)$ is a sub-graph of $G \setminus v$ ($V_0 \supset V_1$ and $E_0 \supset E_1$). If further divided, the 'offspring'(s) of $G \setminus v$ may be exactly the same as $G \setminus C(v)$, thus does not need to be solved multiple times.

There is no strict rule of which v should be chosen as long as it has at least one constraint on it. In our algorithm, we always choose the vertex v with the biggest degree (has the biggest number of edges connecting it), thus the number of edges in G_1 is minimised to have a minimal depth of subsequent dividing.

Figure 4.2 illustrates the breakdown process of our *NSGDP* algorithm with a simple problem instance (Figure 4.2a). There are 7 requirements and 5 conflicting interactions in this instance. The edge connects two requirements means these two requirements are conflicting with each other. So the expressions of this instance are $V = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$ and $E = \{(r_2, r_3), (r_2, r_4), (r_2, r_5), (r_5, r_6), (r_3, r_7)\}$. For this instance, the problem is divided based on requirement r_2 firstly. The reason is that r_2 conflicts with most requirements (r_3, r_4, and r_5), so its degree is the biggest (degree 3).

Then the problem is broken down into two sub-problem. The r_2 is not selected in the first sub-problem $G_b = G_a \setminus r_2$ (Figure 4.2b), so r_2 and the edges connected to r_2 are removed. In the second sub-problem $G_c = G_a \setminus Ext(r_2)$ (Figure 4.2c), req_2 is selected. Accordingly, the requirements have connection with r_2 are removed. The dashed line in Figure 4.2c denotes that, in order to solve the problem $G_c = G_a \setminus Ext(r_2)$, *NSGDP* firstly solves the right part, and then computes the optimal frontier of whole problem G_c by merging the consideration of left part requirements.

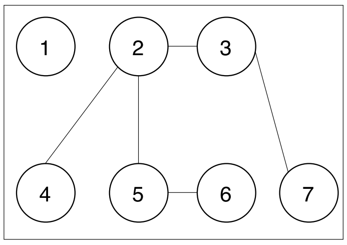
Subsequently, *NSGDP* further divides these two generated sub-problems. Because there is no edge in G_c (Figure 4.2c), no further breakdown would be performed on G_c. Since there are two conflicts in G_b ($E_1 = \{(r_3, r_7), (r_5, r_6)\}$), and each conflicted requirement has same degree (degree 1), *NSGDP* picks r_3 by requirement id order. Thus, sub-problem G_b is divided into sub-problems $G_d = G_b \setminus r_3$ (Figure 4.2d) and $G_e = G_b \setminus Ext(r_3)$ (Figure 4.2e). *NSGDP* continues to breakdown the problem until there is no further conflict that can be subdivided. In this instance, there are 5 leaf node sub-problems generated.

After the breakdown process is terminated, the *NSGDP* solves the problem from the bottom up. Figure 4.2j illustrates the procedure by a dashed line. According to the composition of problems G_i, G_h, and G_g, the algorithm solves the G_i first. Then the results of G_i can be used for solving the other two leaf nodes sub-problem G_h and G_g. Thus, the re-computation can be avoided by storing previous steps' results.

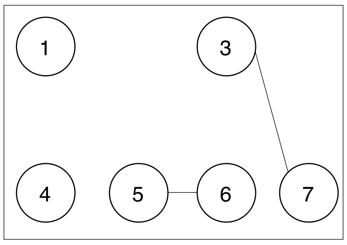
4.3.3 Results Analysis & visualisation

The last step of *METRO* is to analyse the solutions on two Pareto-fronts, one of which is produced by *pNRP*, and the other by *sNRP*. The shape of generated Pareto-frontier exposes the possible trade-off among all conflicting objectives.

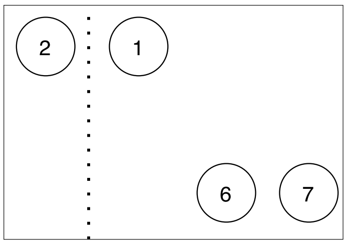
The shape of Pareto-front helps decision makers to understand the possible trade-off among all conflicting objectives, yet it does not provide other intelligible information



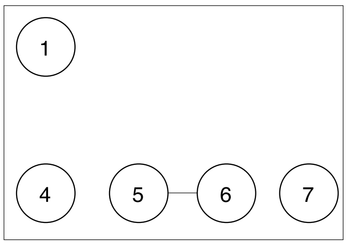
(a) $G_a = G$



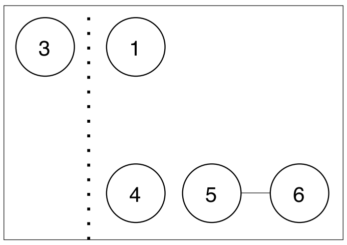
(b) $G_b = G_a \backslash r_2$



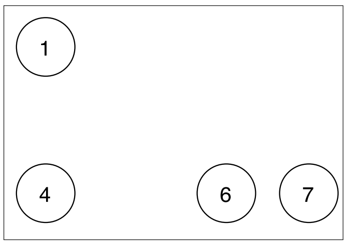
(c) $G_c = G_a \backslash C(r_2) + r_2$



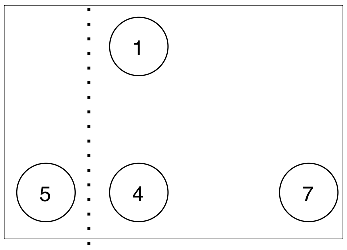
(d) $G_d = G_b \backslash r_3$



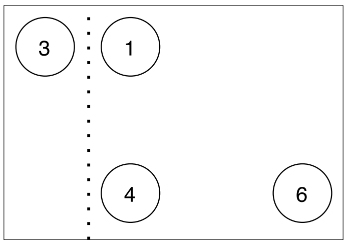
(e) $G_e = G_b \backslash C(r_3) + r_3$



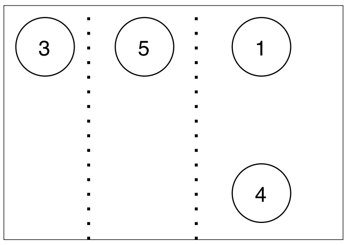
(f) $G_f = G_d \backslash r_5$



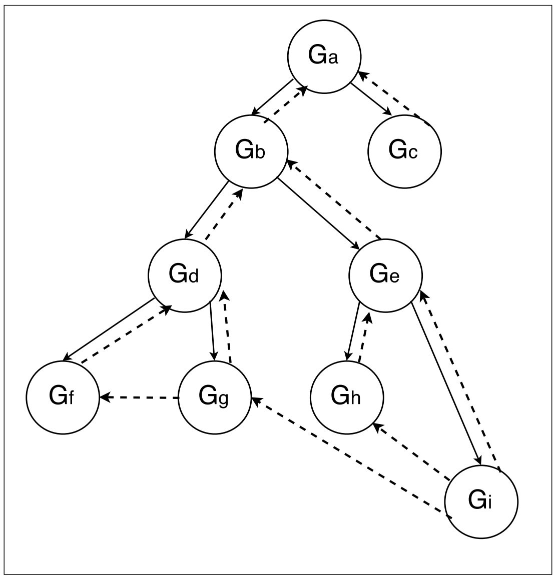
(g) $G_g = G_d \backslash C(r_5) + r_5$



(h) $G_h = G_e \backslash r_5$



(i) $G_i = G_e \backslash C(r_5) + r_5$



(j) The breakdown procedure (solid line) generated by *NSGDP* algorithm, and the solving procedure (dashed line) for problem G_a

Figure 4.2: The illustration of the subdivision process of the *NSGDP* for an instance with 7 requirements and 5 conflicting interactions. Figure 4.2a to Figure 4.2i are each generated sub-problem in the subdivision phase. Figure 4.2j illustrates the generated sub-problems and the solution path of *NSGDP* algorithm.

to interpret the variations among the solutions as well as the characteristics of requirements. In particular, the number of solutions on Pareto-front maybe large, thereby requiring further analysis support to help the decision maker understand the implication for requirement release decisions. By contrast, *METRO* performs a series of posterior analysis procedures to help decision makers to concentrate on the impacts of requirements uncertainty, most interesting solutions, and most urgent and worthwhile requirements.

In order to assess the impact of requirements uncertainty, we introduced the concept of the *expected risk premium*, which is a variant of the *risk premium* [182]. This measures the difference between robust-yet-suboptimal solutions and optimal-yet-risky solutions. The robust-yet-suboptimal solution is simply that which has the lowest uncertainty variance in the distribution of possible values. Suppose we use the point based method to find a particular optimal-yet-risky solution (a set of requirements), $\vec{a}$, with given cost, $cost(\vec{a})$ and value, $value(\vec{a})$. We can find the robust-yet-suboptimal solution, $\vec{r}$ with cost $cost(\vec{r})$ closest to $cost(\vec{a})$ that does not exceed $cost(\vec{a})$. This is the greatest lower bound, on robust-yet-suboptimal solutions, bounded by the cost of $\vec{a}$. Because the robust-yet-suboptimal solution takes account of uncertainty, it has a range of possible values, of which the expected value, $value(\vec{r})$, is simply the most probable. The *expected risk premium* is simply the difference between $(value(\vec{a}) - cost(\vec{a}))$ and $(value(\vec{r}) - cost(\vec{r}))$, where the *value* and *cost* are all normalised. It is an 'expected' assessment of the return that will be lost by maximally reducing uncertainty. It is thus a way of understanding the penalty that is paid for reducing uncertainty in terms of reduced expected return.

To compute the *expected risk premium*, a ***solution compare pair*** which contains an optimal-yet-risky solution and a robust-yet-suboptimal solution should be determined first. For each robust-yet-suboptimal solution $\vec{r}$, each optimal-yet-risky solution from *pNRP*, which has the closest cost to $\vec{r}$ and the cost is not lower than the cost of $\vec{r}$, is chosen as the paired solution $\vec{a}$. Thus, the ***solution compare pair*** is expressed as $Pair(\vec{a}, \vec{r})$. The procedure to generate ***solution compare pair*** is illustrated in Algorithm 3.

Algorithm 3 Generate solution compare pairs

```
Require: $sNRP$ solution set $S_1$, and $pNRP$ $S_2$.
  set $Pairs = \varnothing$
  for all $\vec{s_1} \in S_1$ do
    $\vec{s} = S_2[0]$
    for all $\vec{s_2} \in S_2 \wedge Cost(\vec{s_2}) \geq Cost(\vec{s_1})$ do
      if $Cost(\vec{s_2}) < Cost(\vec{s})$ then
        $\vec{s} = \vec{s_2}$
      end if
    end for
    $Pairs = Pairs \cup Pair(\vec{s_1}, \vec{s})$
  end for
  return $Pairs$
```

4.4 Applying our approach to the RALIC dataset

In this section, we illustrate the insights that can be obtained by applying the proposed framework on a large real-world example: the RALIC dataset.

4.4.1 Experimental set up

The detail of dataset, and the targeted objectives of the experimental study are presented as follows.

Dataset

The RALIC project is an access control system developed at University College London, UK. This project was established in 2009 and deployed in 2011. The requirement data was collected by using the StakeNet stakeholder analysis method and StakeRare requirement elicitation method [183]. The implementation cost of each requirement was derived from the RALIC posterior implementation report. The cost is represented as the total man-hours spent on the requirement during the whole project development life cycle. The detail information of RALIC data is publicly published at `http://soolinglim.wordpress.com/datasets/`.

Because there is no uncertainty information about the attributes of requirements in RALIC dataset, we synthetically simulated these uncertainties following guidelines from the literature [116] which advocate a triangle probability distribution. In the early requirements engineering phase, due to the lack of definition or understanding of the requirements to be done, the level of software cost estimation accuracy ranged from 25% to 400% [167, 184, 185]. According to Jørgensen and Moløkken-Østvold's review [185], the Standish Group CHAOS Report [186] indicates that 52.7% of software projects will overrun the 89% of their original budget estimation. Therefore, in our study, we define the range of uncertainty for requirement cost as $[25\%, 400\%]$. There have also been studies on the accuracy of the software profit estimation or the satisfaction of stakeholders. Michael Bloch et al. study large-scale IT projects and report that the average benefit shortfall of IT projects is 56%, but no range is reported. As Fogelstrom et al. [187] pointed out in 2009, business risk-related uncertainty has received little attention, which means that we have little guidance as to the likely bounds we should place on uncertainty. Therefore, we have allowed for potential boundary scenarios in choosing our uncertainty bounds. That is, the range of uncertainty for satisfaction of stakeholder is defined as $[10\%, 300\%]$. We believe that the true uncertainty value for any realistic project is likely to lie within this extreme range.

There are two versions of RALIC datasets: 'PointP' and 'RankP'. In this chapter, we empirically studied our framework on the 'PointP' dataset, which consists of 143 requirements, 86 *And* dependencies, and 23 *XOR* dependencies. To generalise the study, three NRP instances are derived. There are two boundary scenarios, in which the uncertainty of a requirement is estimated, either highly optimistically or pessimistically, and one 'in-between' scenario. In highly optimistic scenarios, the requirements uncertainty is totally underestimated (mode value equals to the lowest value). By contrast, the requirements uncertainty is overestimated in highly pessimist scenarios (mode value equals to the highest value). After the pre-processing described in Section 4.3.1, there are 57 refined requirements, and 4 *XOR* dependencies.

NRP Objective Formulation

In our experiment, three attributes of software release planning were considered as the optimisation objectives: cost, satisfaction level, and the probability of budget overrun. The objective cost and satisfaction level were viewed as the utility of software release attainment and expressed as normalisation functions. We assumed that these two objectives were aggregatable. Thus, the expressions of the objective cost and satisfaction can be defined as follows (Eq. 4.2 and 4.3):

$$U(\vec{x}, cost) = \frac{\sum_{i=1}^{n}(x_i \cdot Cost_i)}{\sum_{i=1}^{n} Cost_i} \tag{4.2}$$

$$U(\vec{x}, satisfaction) = \frac{\sum_{i=1}^{n}(x_i \cdot Satisfaction_i)}{\sum_{i=1}^{n} Satisfaction_i} \tag{4.3}$$

The quality of the solution is measured as the utility score of the solution (Eq. 4.4):

$$Quality(\vec{x}) = U(\vec{x}, satisfaction) - U(\vec{x}, cost) \tag{4.4}$$

The expression of the probability of budget overrun remains the same (Eq. 3.7). The extent θ is set as 150%. To reduce the simulations errors introduced by Monte-Carlo Simulation, in our experimental study, the number of simulations is set as $10,000$.

4.4.2 Research Questions

To evaluate the *METRO* framework, we carried out an experimental study to assess the ability of this approach to manage the algorithmic uncertainty and capture the impact of requirements uncertainties. In the experiment, we demonstrate why the requirements optimisation community should take care with algorithmic uncertainty, and how to employ *METRO* as a tool to assist decision makers to comprehend the results, thus raising three main research questions:

RQ1: What is the effectiveness of our exact NRP solver *NSGDP* for eliminating algorithmic uncertainty?

We investigate how much difference can be observed between the solutions found by NSGA-II and *NSGDP*. This research question is a foundation for applying *NSGDP*. We compare the solutions found by NSGA-II with the benchmarks which are found by *NSGDP*. The differences between NSGA-II solutions and benchmarks reveal additional (unnecessary & unhelpful) uncertainty introduced by NSGA-II.

RQ1.1: How close are the solutions found by NSGA-II to the ones found by *NSGDP* in objective space?

RQ1.2: Comparing the solutions provided by *NSGDP* and NSGA-II, how much difference can be observed in design space?

The remaining research questions are more concerned with scrutinising the impact of uncertainty that came from requirement itself.

RQ2: What is the impact of the requirements uncertainty?

This question can be expressed in a quantified manner as to how much expected risk premium can be obtained when a decision moves from an optimal-yet-risky solution to a robust-yet-suboptimal one under the same budget.

RQ3: Is there any pattern between the requirements characteristics and requirements inclusions? If so, what kind of pattern can be observed?

The third research question investigates the possible insight of the requirement characteristics, which may help decision makers to concentrate on the most interesting property of requirements. This question is composed of two more detailed sub-questions (***RQ3.1***, and ***RQ3.2***):

RQ3.1: Which requirements are the most sensitive, so require closest attention from the decision makers?

RQ3.2: Which requirements have the same inclusion behaviours, and can thus be clustered together?

4.4.3 Experiment Results

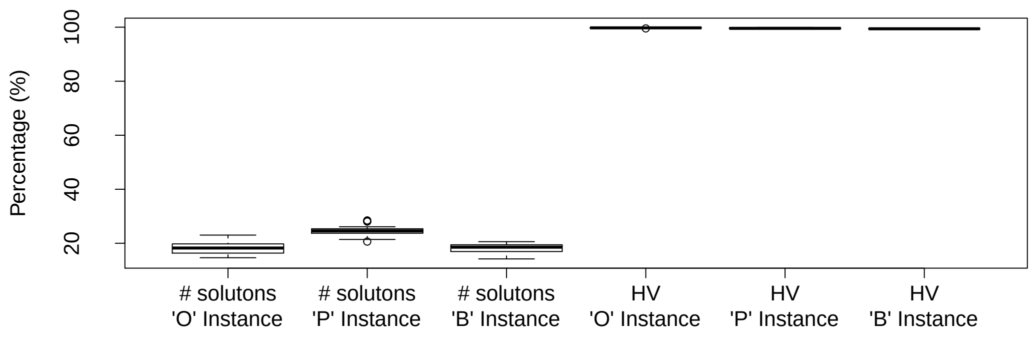
In this sub-section, we present the results of experimental study, and provide a decision analysis guidance for decision makers by interpreting the research questions sequentially and separately.

RQ1: What is the effectiveness of our exact NRP solver *NSGDP* for eliminating algorithmic uncertainty?

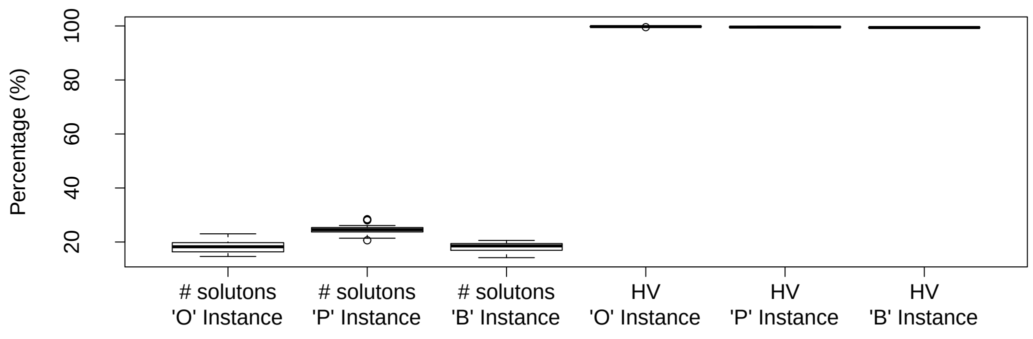
RQ1.1 How close are the solutions found by NSGA-II to the ones found by *NSGDP* in objective space?

We answer this question by comparing the quality of solutions found by NSGA-II and *NSGDP*. Three quality indicators are used; the percentage of optimal solutions found, the relative hypervolume of the solution set, and the execution time. Figure 4.3a and 4.3b present two quality indicators of the solutions generated by NSGA-II in objective space. The execution times of NSGA-II and *NSGDP* are reported in Figure 4.3c and 4.3d. We study the effectiveness of NSGA-II and *NSGDP* on three synthetic NRP instances. We execute NSGA-II on each instance over 30 runs. In order to intuitively observe the differences, in Figure 4.3a and 4.3b, we report only the proportion of optimal solutions, and relative hypervolume of Pareto-front found by NSGA-II.

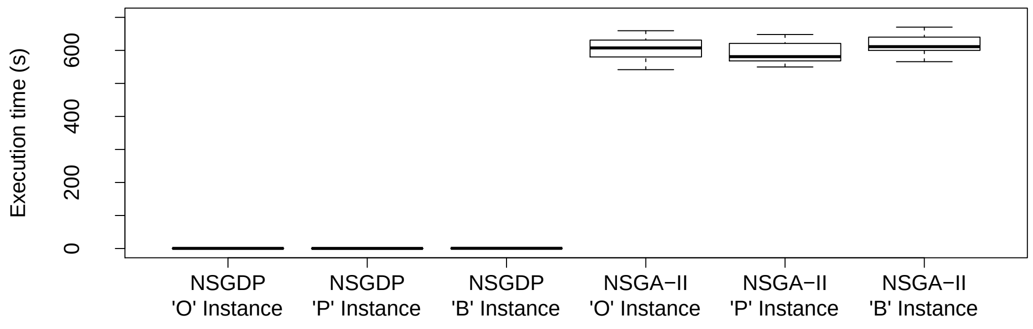
***RQ1.1* can be answered with Figure 4.3**. In all cases, there are thousands of solutions on the true Pareto-fronts. In all three RALIC instances, the relative hypervolume of solutions found by NAGA-II ranges from 98.68% to 99.96% – fairly close to



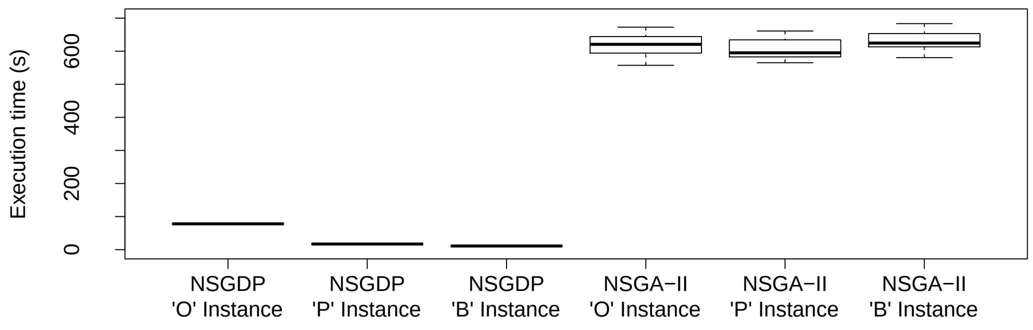
(a) # solution and HV of the results found by *pNRP*



(b) # solution and HV of the results found by *sNRP*



(c) The time for executing each approach (*pNRP*)



(d) The time for executing each approach (*sNRP*)

Figure 4.3: **Answers *RQ1.1***. These figures illustrate the differences between the solutions found by NSGA-II and *NSGDP*. Figures 4.3a and 4.3b present the differences, based on two quality indicators (number of optimal solutions found and relative hypervolume). Figures 4.3c and 4.3d present execution time differences. '# solution' denotes the percentage of optimal solutions, and 'HV' stands for relative hypervolume. The names of instance 'O', 'P', and 'B' stands for the highly Optimistic RALIC instance, and highly Pessimistic RALIC instance, and 'in-Between' RALIC instance, respectively.

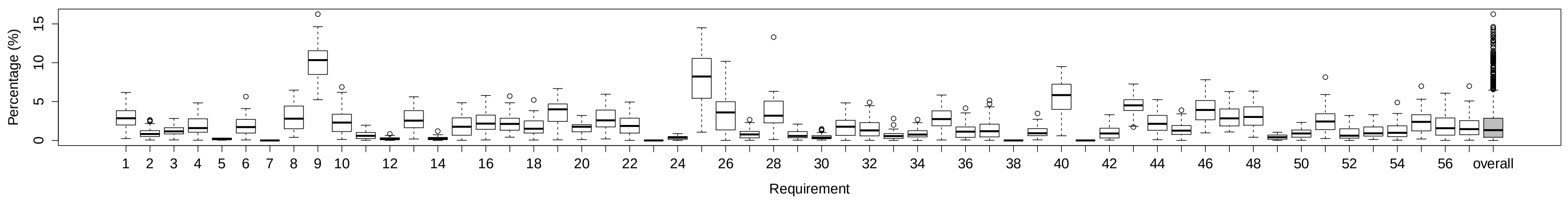
the optimal solutions. It denotes that, in our study, NSGA-II is able to find the solutions with a good convergence near the true Pareto-optimal front. This is because we allowed NSGA-II to use sufficient computation resources with 1000 population and 1000 generations. However, with respect to the number of optimal solutions found, NSGA-II may fail to find at least 73.03% of the optimal solutions. The percentage of missed optimal solutions can be yet up to 99.95% when considering uncertainty as an extra optimisation objective. Therefore, despite the fact that the convergence of NSGA-II is close to true Pareto-front for NRP, the randomness of NSGA-II makes it difficult to find complete optimal solutions. It reveals that additional uncertainty is introduced to solutions by the algorithm itself. Additionally, according to Figure 4.3c, when decision makers do not consider requirements uncertainty, they can get response from *NSGDP* immediately (0.37s on average), and wait for up to 616.93s to get results from NSGA-II. If decision makers take requirements uncertainty into consideration, *NSGDP* is (35.33s on average) still faster than NSGA-II (675.26s on average) in general (Figure 4.3d).

RQ1.2 Comparing the decisions provided by *NSGDP* and NSGA-II, how much difference can be observed in design space?

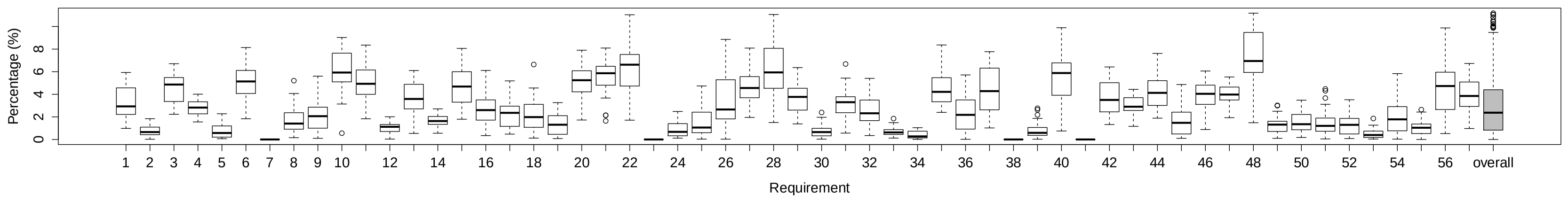
According to the answer of ***RQ1.1***, we can see that, even through NSGA-II converges to the true Pareto-front in objective space, it can find only a small proportion of optimal solutions. However, it is possible that such a small difference in objective space is caused by prominent difference in design space. In order to investigate the hypothesis, we intend to inspect the *requirements selection probability*, which we define as the chance of requirement being included in the entire generated solution set. Therefore, we compare the overall requirements selection probability provided by NSGA-II and *NSGDP*, and analyse how much chance that the requirements decision is wrong when applying NSGA-II instead of exact approach. The probability of getting wrong requirements decision is measured by the difference of the requirements selection between NSGA-II solutions and benchmarks regarding to each requirement. Figure 4.4 and 4.5 picture the chance that NSGA-II gives wrong requirements decision with respect to each requirement and the overall probability. The figures depict the essential impact raised from using an approximate algorithm.

In RALIC experimental study, due to the effects of randomness from an approximate algorithm, in different runs, the requirements selections are volatile. According to Figure 4.4 and 4.5, requirements uncertainty would aggravate the impact of inexactness of NSGA-II in general. In all instances, the probability of receiving a wrong decision in *sNRP* is almost double than that of *pNRP*. Therefore, in the present algorithmic uncertainty, requirements uncertainty places decision makers at more serious risk of getting wrong requirements decision. We can see that, in an 'in-between' scenario, the upper bound of the chance of getting wrong requirements decision could rise from 16.25% to 36.48%, and the median overall chance rises from 2.01% to 10.94%). Even if it were possible that in a particular run or particular scenario NSGA-II can offer a minor wrong decision, the non-determinism makes it produce a different answer in a different run. The erratic result may result in providing completely disorganised decisions. It emphasises that decision makers definitely should raise concerns about the impact of stochastic algorithm on requirements selection. It is noteworthy that such impact implies some patterns. We found that the chance of getting a wrong decision is negatively correlated with the implementation cost of requirement (Spearman ρ up to 0.72 and $p \ll 0.0001$). That is, the larger the requirement implementation cost, the less the chance that making wrong decision in requirements selection. There are only three exceptions (Requirement 7, 23, and 38). NSGA-II has nearly perfect match agreement on these three requirements over three NRP instances. The reason for this exception may be due to the inherit exclusive-or dependencies between these requirements (discussed in Section 4.4.3). (**This answers *RQ1.2***).

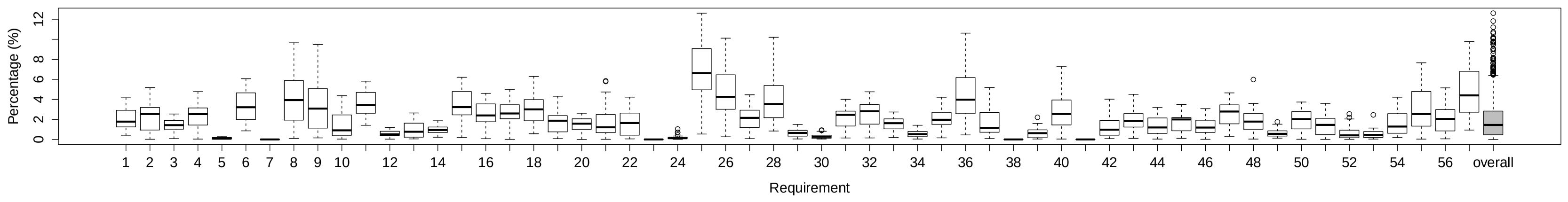
In summary, to account for uncertainty in NRP, it is important for decision makers to understand the source of uncertainty in solution. Although NSGA-II can generate approximate solutions with good convergence with respect to objective space, the results of NSGA-II are still incomplete and suboptimal. However, the solution quality information is not meaningful for decision makers when they have to make decisions for each requirement. Even for the solutions which are very close to true Pareto-front,



(a) Without considering uncertainty (*pNRP* - highly optimistic scenario)

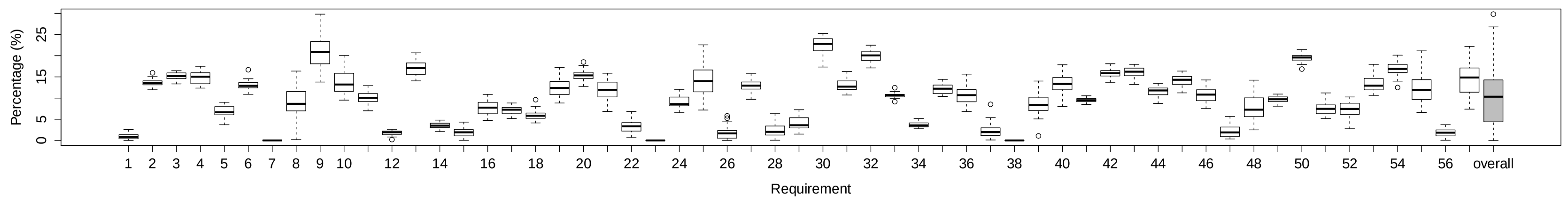


(b) Without considering uncertainty (*pNRP* - highly pessimistic scenario)

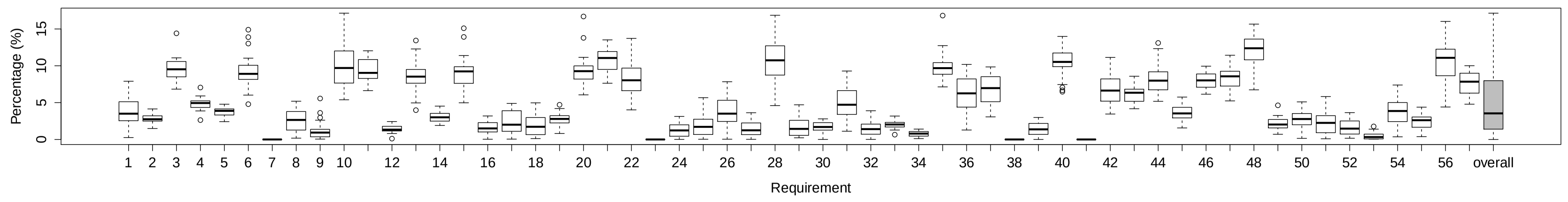


(c) Without considering uncertainty (*pNRP* - 'in-between' scenario)

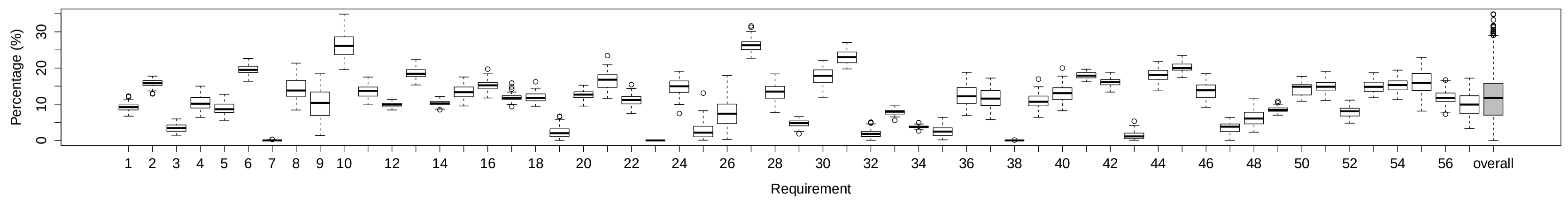
Figure 4.4: **Answers *RQ1.2***. These box plots show the chance that NSGA-II provide wrong requirements selection decision for each requirement in RALIC instance without considering uncertainty. The grey box plot depicts the overall chance of getting wrong requirements decision.



(a) Taking account of uncertainty (*sNRP* - *highly optimistic scenario*)



(b) Taking account of uncertainty (*sNRP* - *highly pessimistic scenario*)



(c) Taking account of uncertainty (*sNRP* - *'in-between' scenario*)

Figure 4.5: **Answers *RQ1.2***. These box plots show the chance that NSGA-II provide wrong requirements selection decision for each requirement in RALIC instance with considering uncertainty. The grey box plot depicts the overall chance of getting wrong requirements decision.

the decisions of selecting requirements are surprisingly distinguishable. In other words, relying on the requirements selections generated by NSGA-II would result in misleading the requirement decision. Consequently, the wrong decision would further cause the failure of a software project. Last but not least, *NSGDP* not only can guarantee the exactness of result but also outperform NSGA-II by offering a faster response to decision makers. This enables decision makers to receive feedback from our framework instantaneously. All of the above results emphasise the value of *NSGDP*, thus, answering why the requirement optimisation community should consider exact approach, and promote the motivation of our research.

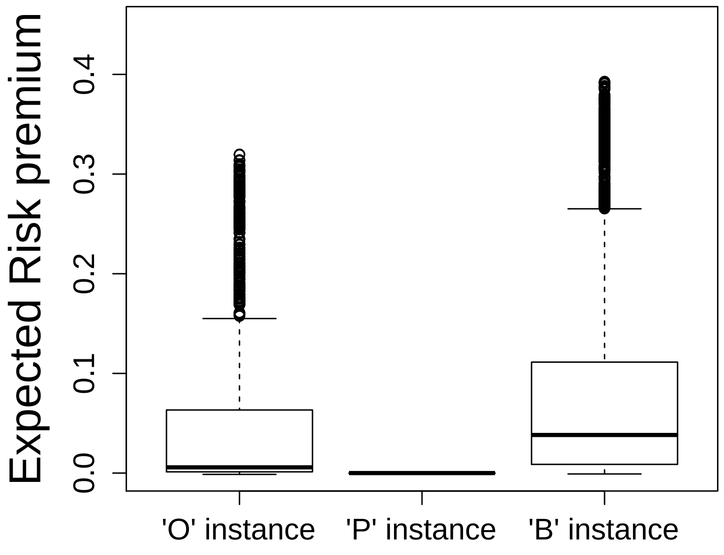
RQ2: What is the impact of the requirements uncertainty?

After ruling out the algorithmic uncertainty, we would like to evaluate the impact of the requirements uncertainty. The results of the analysis are depicted in Figures 4.6.

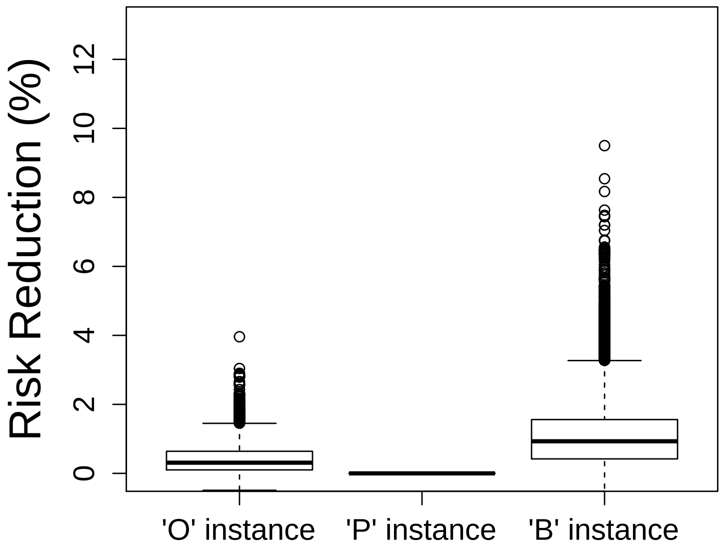
Figure 4.6 statistically explains the impact of requirements uncertainties on RALIC NRP instance (with 150% budget overrun). There are $9,868$, $1,149$, and $31,417$ optimal solutions found when considering requirements uncertainty (*sNPR*) in a highly optimistic scenario, a highly pessimistic scenario, and an 'in-between' scenario, respectively. And there are 221, 0, and $1,045$ outliers in Figure 4.6b, and 572, 0, and 967 outliers in Figure 4.6a. The percentage of outliers is lower than 5.79%. It could be observed that, overlooking requirements uncertainty can contribute to suffer up to 10.09% risk that overrun more than 150% budget, and get at most 0.39 utility in return.

The impact of requirements uncertainty is negligible in a highly pessimistic scenario. This is because the worst case requirements uncertainty has been taken into account in requirements estimation. Taking account of uncertainty during requirements selection does not matter much for decision makers.

On the other hand, the impact of requirements uncertainty in a highly optimistic scenario is slightly less than in an 'in-between' scenario. This circumstance probably is



(a) Expected risk premium



(b) Reduction of risk

Figure 4.6: The box-plots show how much expected risk premium and reduction of risk can achieve by taking account of requirements uncertainty. The names of instance 'O', 'P', and 'B' represents the highly optimistic RALIC instance, and highly pessimistic RALIC instance, and 'in-between' RALIC instance, respectively. **These figures answer *RQ2*.**

the consequence of involving extremely large uncertainty in requirements estimation. In general, the principal of *sNRP*, which provides robust-yet-suboptimal solutions, is replacing uncertain requirement with appropriate 'less uncertain' requirement(s). Meanwhile, all requirements in a highly optimistic scenario are extremely uncertain. There is a few relatively 'less uncertain' requirements can be chosen. Accordingly, our framework cannot reduce too much impact of requirements uncertainty in a highly optimistic scenario. In spite of sacrificing a little extra utility to reduce the risk by a small degree, this still offers decision makers more options than point-based estimation approach. For a decision maker, who is risk-averse, this risk reduction is more valuable than the gained utility for him or her. So the decision maker will not choose optimal-yet-risky solutions, and would accept the guaranteed robust-yet-suboptimal solutions. Otherwise, optimal-yet-risky solutions would be more attractive for risk-loving decision makers. (**This answers *RQ2***).

In summary, requirements uncertainty would result in uncertainty for the overall software release plan. In order to minimise this risk, some loss of perceived utility must be accepted. The 'loss' involved is only a 'perceived' loss, in any case, because it is a calculation of loss based on the point-based estimate, which is unrealistic.

RQ3: Is there any pattern between the requirements characteristics and requirements inclusions?

The previous answer to research question ***RQ2*** offers a 'macroscopic' suggestion to a decision maker, helping them to understand the trade-off among different objectives. However, the result cannot provide more details about the nature of requirements, which may inspire decision makers to prioritise the requirements for further evaluation and inclusion. To aid this problem, ***RQ3*** promotes a detailed 'microscopic' investigation of requirements analysis for RALIC dataset.

RQ3.1 Which requirements are the most sensitive, so require closest attention from decision makers?

As distinct from conventional sensitivity analysis, here we take an algorithmic view of the problem. Figure 4.7 describes the difference in the paired candidate solutions in terms of the requirements selection probability.

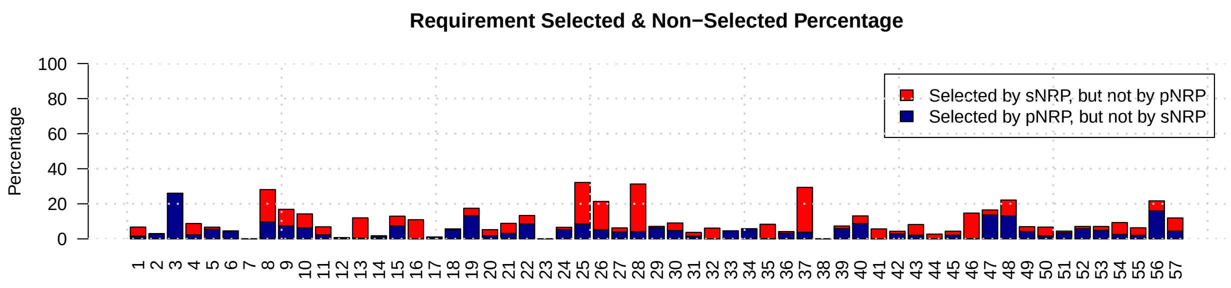
The difference in the paired candidate solutions is defined as follows. For a particular requirement *req*, set A denotes the paired solutions that contain *req* in *sNRP* solutions, and set B denotes the paired solutions that contain *req* in *pNRP* solutions. The intersection of these sets indicates the number of pairs that contain *req* in both parts (set $A \cap B$). The height of bar in Figure 4.7 is the symmetric difference of A and B ($A \triangle B$). More precisely, $A \setminus B$ is denoted by the height of the red (light grey in black and white) bar, while the size of $B \setminus A$ is denoted by the height of the blue (dark grey in black and white) bar. If the height of bar is 0, it means the selection of this requirement in all paired candidate solutions is identical. In this situation, the result reveals that, although it is unrealistic in general, in this specific instance the point-based estimate can be relied upon (even in the presence of the extreme range of risks we model ($10\% - 300\%$), and two boundary NRP instances). From Figure 4.7 we observe that 3 of the 57 requirements have this common property in all instances. For these 3 requirements, our analysis has thus revealed that we could simply revert to considering the point-based estimate as sufficiently robust, even in the presence of extreme risk. However, for the remaining 54 requirements, our analysis demonstrates the importance of modelling risk. Another interesting finding is, in a highly pessimistic scenario, there is no difference between *sNRP* and *pNRP* methods, and the difference is minor in highly optimistic scenario. The reason has been discussed in Section 4.4.3.

We take Kendall's τ_B correlation coefficient to statistically analyse the correlation between the difference of requirements selection probability ($\mathbb{P}(selected\_sNRP) - \mathbb{P}(selected\_pNRP)$) and its own risk in the highly optimistic and 'in-between' scenarios experiments ($\mathbb{P}$ measures probability). The analysis result shows that there is a negative correlation between these two attributes ($p \ll 0.001$ and τ_B up to -0.675). Namely, the requirement with lower uncertainty has more chance to be selected by a risk-aware approach. Therefore, decision makers can observe the sensitivity of each requirement from the perspective of algorithm. In RALIC experimental study, Requirements 3 is the most sensitive requirement with respect to 150% budget overrun in both highly optimistic and 'in-between' NRP scenarios. By contrast, Requirements 25, 26, 28, 37, and 46 are more insensitive. This recommends a risk-averse decision maker to be deeply concerned with Requirement 3, and assign high priority to Requirements 25, 26, 28, 37, and 46 (**answer *RQ3.1***).

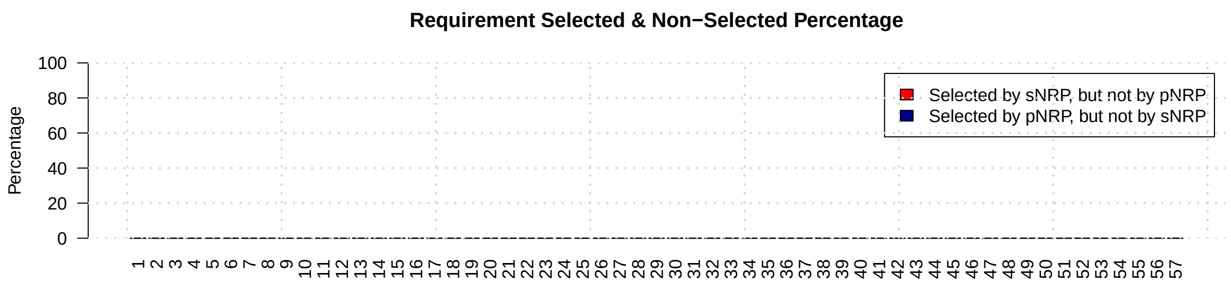
RQ3.2 Which requirements have the same inclusion behaviours, and can thus be clustered together?

With increasing numbers of requirements, it will be tedious and time consuming to analyse each requirement manually. Identifying inclusion behaviours, the tendency of including a requirement in the solutions on the Pareto-front as the budget increases, and analysing the differences between them may allow us to cluster requirements to reduce cognitive overload. *METRO* uses a heat-map (Figures 4.8 and 4.9) to visualise the inclusion of requirements in the solutions on the Pareto-front with respect to the results generated by *pNRP* and *sNRP*. Moreover, in order to measure and highlight the similarities and differences, we cluster related requirements by computing the *complete Euclidean* distance among requirements' inclusion percentage and present the results of the corresponding *Hierarchical Clustering*. This approach is exemplified by the results of 'in-between' NRP instance. From Figures 4.8 and 4.9, we can observe that there are 4 major clusters identified in the result of both *pNRP* and *sNRP*, which can help the decision maker to inspect at a much smaller number of groups of related requirements. Additionally, instead of prioritising all requirements, decision makers can first prioritise the requirements groups before prioritising the requirements within each cluster.

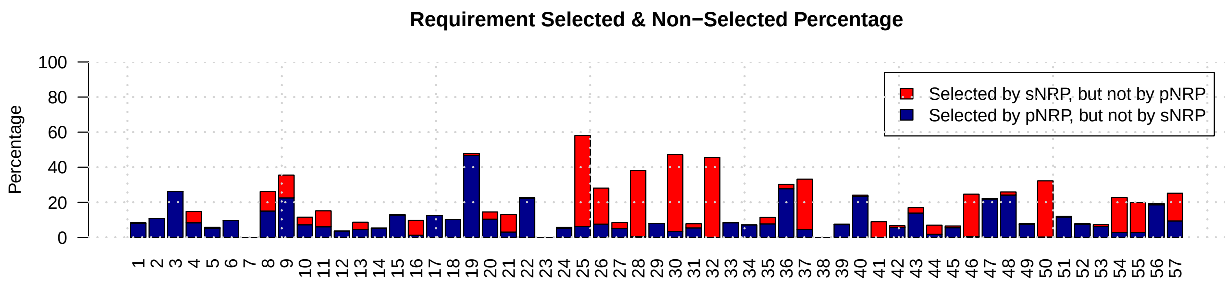
The answer of *RQ3.2* is that, in 'in-between' RALIC instance, Requirement 12 is treated similarly with Requirement $13, 14, 19, 36$ and prioritised as the most critical requirements group by *pNRP* approach, while Requirement 12 is grouped with $13, 14, 21, 28, 46$, and 55 which are all formalised as the most critical requirements by *sNRP* approach. Therefore, in order to gain higher profit performance while minimis-



(a) highly optimistic scenario



(b) highly pessimistic scenario



(c) 'in-between' scenario

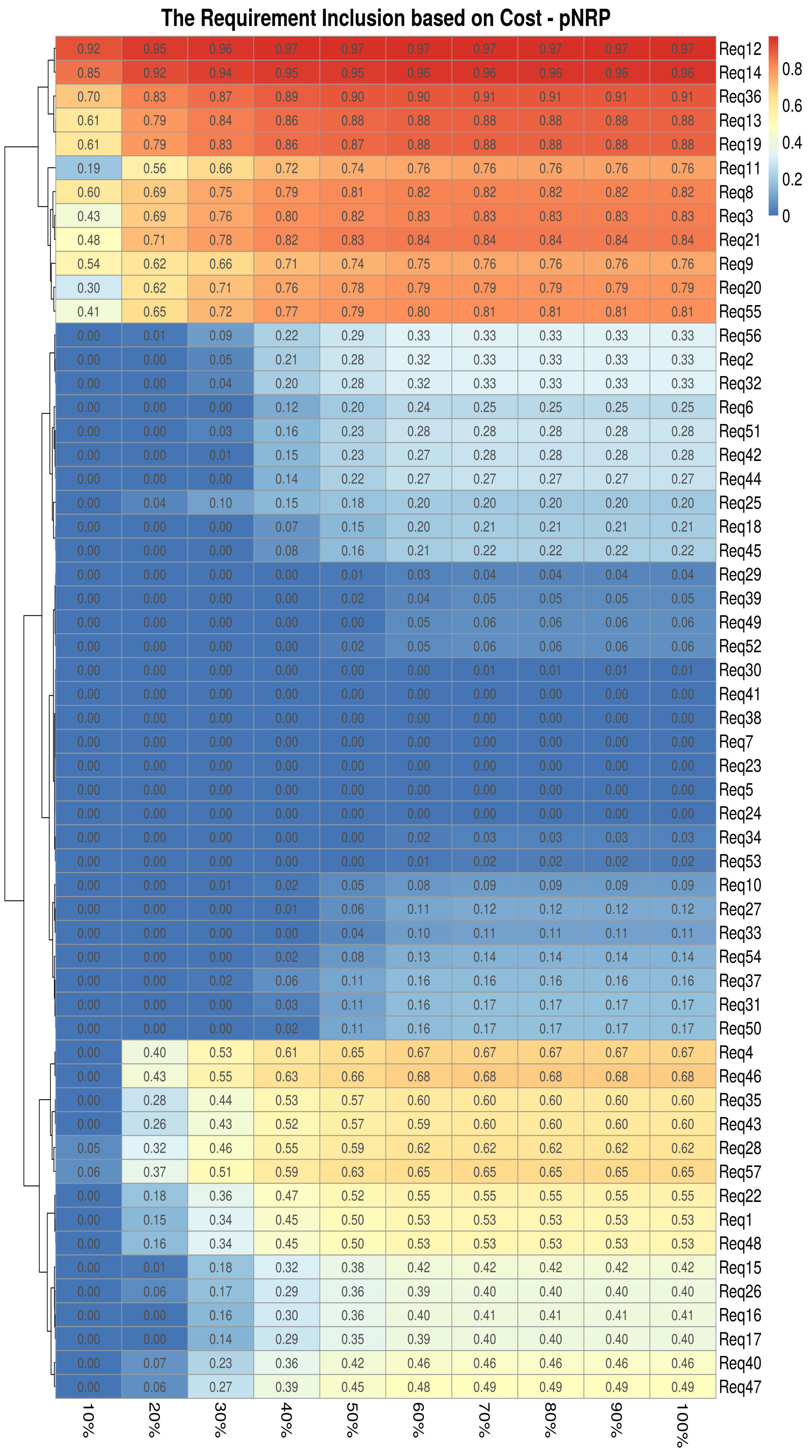
Figure 4.7: **Answers *RQ3.1***. The difference of requirement inclusion between robust-yet-suboptimal solutions and the corresponding optimal-yet-risky solutions in terms of requirements selection probability.

ing budget overrun risk, Requirements $12, 13, 14, 21, 28, 46$, and 55 should be prioritised as the highest priority requirements group. Strikingly, some requirements could never be selected in any solution with any budget by the point-based approach. By looking at these requirements, we find that some of them (Requirements 3, 7, 23, and 41) participate in *XOR* dependencies. These 4 requirements are strongly dominated by corresponding mutually exclusive requirements in terms of optimisation goals. However, there is a nuance in the results offered by simulation-based approach. The risk-aware sNRP approach *does* select Requirement 47 in some circumstances. The possible reason is that, by taking uncertainty into account, Requirement 47 is more robust than Requirement 41. Therefore, Requirement 47 can attract considerable attention when there is abundant budget (i.e., 50% of total budget) to neutralise its unrewarding traditional optimisation goals (i.e., revenue and cost).

To sum up, requirement characteristics play an important role in their inclusion in the solutions on Pareto-front. *METRO* can provide support to help decision makers identify relations between the different solutions by looking at the details of each individual solution. With respect to independent requirements, intrinsic uncertainty negatively correlates with inclusion when minimising solution risk. For mutually exclusive requirements, the inclusion of one requirement relies on the dominance of these requirements' fitness value. Therefore, the dominated requirements are seldom selected, compared with their conflicted twin.

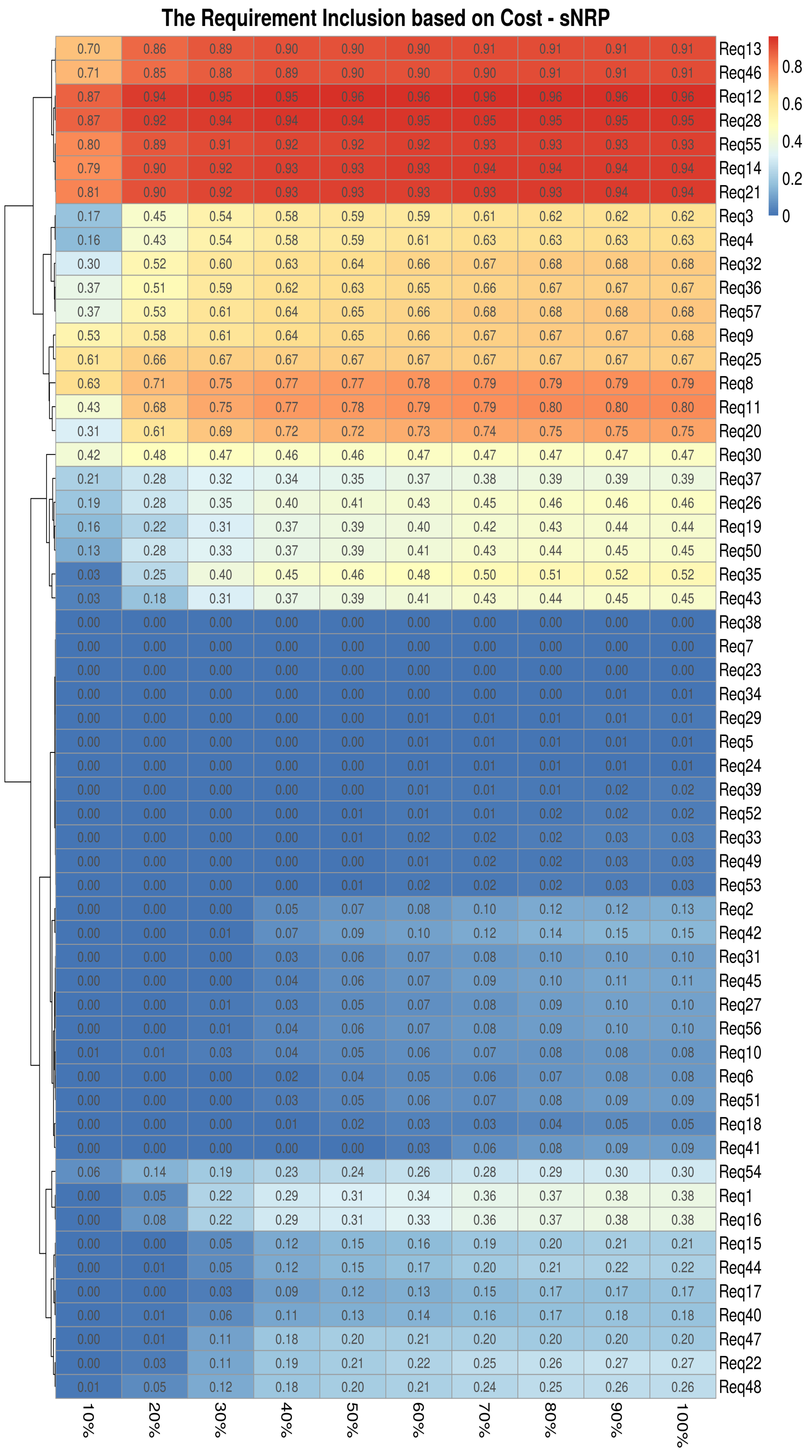
4.5 Summary

In this chapter, we introduced a decision analysis framework *METRO*. *METRO* utilises an exact optimisation approach, incorporating a simulation optimisation technique, to address both algorithmic uncertainty and requirements uncertainty. A novel requirements interaction pre-processing approach is presented to enable fast execution and better convergence during the optimisation. A systematically analysis is offered at the



(a) *pNRP*

Figure 4.8: ***Answers RQ3.2***. The clustered inclusion trends of requirements where $\theta = 150\%$ for *pNRP*



(a) *sNRP*

Figure 4.9: **Answers *RQ3.2*.** The clustered inclusion trends of requirements where $\theta = 150\%$ for *sNRP*

end of optimisation to aid the decision support in the presence of uncertainty.

We applied *METRO* on three NRP instances, derived from a real world NRP instance, RALIC. There are two boundary scenarios, in which the uncertainty of a requirement is estimated, either highly optimistically or pessimistically, and one 'in-between' scenario. We carried out three empirical studies to evaluate the effectiveness and efficiency of *METRO*.

In the first study, NSGA-II, a widely-used approximate technique, is used to compare with our exact NRP solver, *NSGDP*, to investigate the value of exact approach. The results of this empirical study illustrated that *NSGDP* outperforms NSGA-II in terms of execution time, meanwhile, and guarantee the exactness of result.

We looked over the impact of uncertainty that came from requirement itself in the second study. We proposed two metrics to measure the impact, that is, the risk reduction and the 'expected risk premium' which are designed to measure the differences between robust-yet-suboptimal solutions and optimal-yet-risky solutions. The results illustrate that, in order to minimise the probability of project budget overrun, some loss of perceived utility is inevitable.

In the last study, we statistically analysed the relationships between the requirements characteristics and the requirements selection decisions. The results indicate that requirement characteristics play a vitally important role in their inclusion in the solutions on Pareto-front. With respect to independent requirements, the requirement with lower uncertainty has more chance to be selected by a risk-aware approach, and vice versa. For mutually exclusive requirements, the requirements, which are dominated by their conflicted twin in terms of their fitness value, are seldom selected. We believed that this statistical analysis information can offer a detailed 'microscopic' investigation of requirements analysis to a decision maker, helping them to further evaluate and understand the nature of requirements.

To sum up, with the support of *METRO*, requirement engineers can ensure the results' correctness, and systematically analyse the impacts of requirements' intrinsic uncertainty. The information interpreted by *METRO* allows requirement engineers to judge and weigh the trade-off between relatively robust-yet-suboptimal solutions and optimal-yet-risky solutions, and capture elaborate requirement priorities to further reduce the cognitive load.

However, the requirements selection problem addressed in this chapter does not take into account the resource allocation. The primary limitation of this approach is that solving separately the requirements selection problem and the resource allocation problem may produce suboptimal software project planning results. Ignoring the constraints and interactions between those two problems may generate a software project plan that is prone to failures. In the next chapter, we propose a holistic and systematic approach to manage this limitation.

Chapter 5

Exact Analysis in Integrated Release and Schedule Planning Problem

Planning the releases of software is essential for requirements engineers to determine which requirements to implement in the next release. Meanwhile, allocating all the necessary resources to implement the requirements is a well-known complex process in software project management. Although extensive research has been conducted in those areas, they are generally handled in isolation and solved using heuristic search-based techniques. This raises another concern about the uncertainty of resource constraints for requirements selection and optimisation. Ignoring the uncertainty of resource constraints may lead to information loss and suboptimality in the plans produced.

In this chapter, we introduce an exact multi-objective integrated release and schedule planning approach, *iRASPA*, to address both algorithmic uncertainty and uncertainty of resource constraints. *iRASPA* not only provides a release plan that maximises the value of the delivered software and minimises the variance of the workload, but also meets all the resource allocation constraints. We argue that the proposed approach can effectively help decision makers to avoid suboptimality and algorithmic uncertainty.

5.1 Motivation

Incremental software development is both an effective way to deliver the business value of software earlier and an enabler of swifter market feedback [150]. Every release planned through an incremental software development process includes a collection of requirements that are bound to be implemented and delivered by a fixed release deadline.

In order to manage software development incrementally, there are two major problems faced by the requirements engineering and software project management research communities. The first problem is determining which subset of a large set of candidate requirements should be assigned to which release of the software product [20]. The second is how to allocate the corresponding resources while satisfying the resource constraints.

Solving both problems requires taking several variables into consideration simultaneously. A defective solution might trigger that some pivotal requirements are not provided at the right time or coerce software developers to overwork. This might further lead to overwhelming project delays and additional spending, quality degradation and even health risks for employees. Overall, dissatisfied stakeholders and diminished market competitiveness would easily arise [188] and could jeopardise the whole project.

There exist release planning and schedule planning approaches that can straightforwardly prioritise the requirements and assign the resources to implement the requirements, separately and sequentially [150, 151, 152, 153]. However, those approaches suffer from two key limitations:

1. Uncertainty of resource constraints is ignored. Release planning gets mismatched with schedule planning [47]. Release planning usually works with requirements and time [104], while schedule planning simultaneously takes into account requirements, resources and time [189, 190]. When dealing with both problems separately the solutions obtained generally reflect local optima [47] that may yield cost overruns, diminished revenues, underused resources and release date delays.

2. Algorithmic uncertainty arises [22]. Research on release planning and schedule planning have been primarily concerned with Search-Based Software Engineering (SBSE), which has been successful in addressing a broad range of software engineering problems by using algorithmic search techniques [95, 191]. Algorithms used in SBSE are usually heuristic and stochastic. Thus, exactness and repeatability are sacrificed for the sake of speed and flexibility, which is not just convenient, but fine in most situations. However, it is still crucial for mission critical projects to base decisions on exact results. Li et al. [3] have shown that, when solving the Next Release Problem, which is a special case of release planning, the exact approach can avoid information loss and wrong requirement selection due to nature of the approximation algorithm.

Both issues have the potential to cause the failure of a software project [192]. In order to address them, we propose *iRASPA*, an exact multi-objective *i*ntegrated *R*elease *A*nd *S*chedule *P*lanning *A*pproach to support better complex decision making in software release and schedule planning. Our approach joins the ε-constraint method [193] with Quadratic Programming (QP) to produce exact Pareto fronts and guarantee that, subject to the planning constraints, the results are optimal and cannot be improved.

Instead of scheduling the resources after selecting the requirements, *iRASPA* follows a holistic process, integrating software release planning and schedule planning and solving both problems at the same time, thus avoiding the loss of global optimality incurred by other approaches, where those problems are solved sequentially, i.e. in two stages [47].

The primary contributions of the chapter follows:

1. Resolving the uncertainty of resource constraints in software release planning process. A holistic planning process, integrating software release planning and software schedule planning in a single activity and producing globally optimal results. We formalise a multi-objective integrated software release and schedule planning problem combining both processes. The resultant mathematical model

is able to accommodate the value vs. workload balance trade-off, while meeting both resource and time constraints. Our experiments reveal that a state-of-the-art implementation of the traditional two-stage approach can miss up to 93.38% of the optimal solutions, unlike *iRASPA*. Besides, it takes *iRASPA* 28.23% less time on average to solve all the instances under consideration.

2. Settling the algorithmic uncertainty problem and guaranteeing the exactness of results. We introduce an ε-constraint Quadratic Programming (QP) approach and present an empirical study on seven real-world software projects. Our experiments show the applicability of *iRASPA*, which can be used to solve the integrated planning problem and can generate the guaranteed optimal Pareto front. Moreover, no scalability issues have been found for variants of the same size that the real-world projects. The average execution time on the projects under study is 6.37 minutes with mainstream cloud-computing infrastructure, at a cost of £ 0.8 per hour, and all the variants could be solved.

3. An empirical study in which the proposed approach is tested. Our results illustrate the effect of different impact factors: dependency density, developer expertise and value-to-cost ratio. The study clearly reveals that these three factors impact *iRASPA* outcomes. Our study reveals that dependency density is negatively correlated with the number of solutions found, while developer expertise and value-to-cost ratio are positively correlated with the number of solutions found. The execution time of *iRASPA* on the instances studied is highly positively correlated with all of the aforementioned impact factors. Statistical tests find Kendall's τ correlation coefficients greater than 0.60 and p-values much lower than 0.001.

5.2 Problem Statement

This section briefly depicts the statements corresponding to the software release planning and software schedule planning problems, regarded as independent problems.

Then, the integrated model is introduced. All these problems are well-known NP-hard problems. See, for example [194].

5.2.1 Software Release Planning

Deciding which requirements should be included in the next release of a product is critical for the success of a software project and a basic problem in requirements engineering [30, 75, 107]. The Next Release Problem was introduced by Bagnall et al. [19], who formulate the requirements selection and prioritisation problems as SBSE problems [21]. In order to make those well-established problems fit for an incremental software development life cycle, a related problem, software release planning, has been advocated by several authors [152, 195]. Software release planning is, roughly speaking, a next release problem generalised to several releases.

In order to define the software release planning problem, let us consider a set $R = \{r_1, \ldots, r_n\}$ of n candidate requirements in a software project. Implementing each requirement implies a certain cost and is expected to bring a benefit to the stakeholders. In the context of cost/value-based requirements engineering, human effort is one of the metrics used to measure the cost, $C = \{c_1, \ldots, c_n\}$, needed to fulfil each requirement. Besides, a utility function can be employed to attach a value, $V = \{v_1, \ldots, v_n\}$, to each requirement, e.g., measuring the satisfaction, importance, expected revenue or any combination thereof, representing the business value delivered by the product to the market. Therefore, each requirement r_k can be assigned a cost c_k and a value v_k for each $k \in [1, n]$.

It is assumed that software should be delivered in a sequence of s releases. The time span of each release is usually fixed and equal. Commonly, decision makers just look at the next immediate release and a next release problem arises ($s = 1$). However, it is also important to see the potential effect of long-term planning ($s > 1$).

A solution to this problem can be represented as a decision vector $x = [x_1, \ldots, x_n] \in$

Table 5.1: Requirements interactions. The corresponding relations are represented by $\xi, \varphi, \chi \subseteq R^2$, which are pairwise disjoint sets.

Interaction	Predicate
Combination	$r_i \; \xi \; r_j \equiv x_i = x_j$
Exclusion	$r_i \; \varphi \; r_j \equiv x_i = 0 \vee x_j = 0$
Precedence	$r_i \; \chi \; r_j \equiv x_j = 0 \vee 0 < x_i \leq x_j$

$[0, s]^n$ encoding the requirements planned in the next s releases. In this vector, decision variable x_k is 0 if requirement r_k is not to be implemented in a horizon of s releases. Otherwise, $x_k \in [1, s]$ indicates in which release r_k will be delivered.

Now, we can formalise the single-objective software release planning problem as maximising

$$\text{Value}(x) = \sum_{1 \leq k \leq n} v_k \psi(x_k) \quad \text{subject to} \quad \text{Cost}(x) = \sum_{1 \leq k \leq n} c_k \psi(x_k) \leq b$$

where b is the project budget, which cannot be exceeded, and $\psi(x) = 0$ if $x = 0$, $\psi(x) = 1$ otherwise. Henceforth, vectors as $c = [c_1, \ldots, c_n]$ and $v = [v_1, \ldots, v_n]$, and the dot product "$\cdot$", will be used as a convenient means to simplify notation.

However, software release planning may have to fulfil further constraints: requirements interactions, resource constraints, etc. The precise type and formulation of the constraints to be considered may vary depending on the particular project at hand, something to be assessed by requirements engineers. In this chapter, we will focus on requirements interactions, in particular, on structural dependencies among requirements. Requirements interactions are extensively detailed in a survey by Robinson et al. [177].

Table 5.1 contains precise definitions for the three types of requirements dependencies considered in the current work for the software release planning problem: combination (and), exclusion (XOR) and precedence. Those requirements interactions have been used for the next release problem by Zhang et al. [99, 170].

Formally, the mathematical optimisation model corresponding to the above software release planning problem can be expressed as follows:

$$
\begin{aligned}
&\max \quad \text{Value}(x) = v \cdot \psi(x) \\
&\text{subject to} \\
&\qquad \text{Cost}(x) = c \cdot \psi(x) \leq b \\
&\qquad x \in [0, s]^n \\
&\qquad x_i = x_j \qquad\qquad\qquad \text{for all } r_i \; \xi \; r_j \\
&\qquad x_i = 0 \vee x_j = 0 \qquad\quad\; \text{for all } r_i \; \varphi \; r_j \\
&\qquad x_j = 0 \vee 0 < x_i \leq x_j \quad \text{for all } r_i \; \chi \; r_j \\
&\text{where} \\
&\qquad \psi(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases}
\end{aligned}
\tag{5.1}
$$

The exclusion and precedence constraints are not linear but they can be transformed into linear constraints by using standard tricks. Thus, instances of this problem can be solved using Integer Linear Programming (ILP).

5.2.2 Software Schedule Planning

Software schedule planning addresses the resource allocation problem and has been listed as one of the crucial processes in software project management [196]. The goal is allocating employees and other resources to requirements or tasks, so that the desired objectives can be achieved under certain constraints [151, 197]. It has been widely studied and formulated as an optimisation problem, e.g., in [150, 190, 198].

Typically, this problem deals with situations in which employees must be allocated to different requirements to be fulfilled. It schedules employees based on their availability and skills, while meeting additional constraints like number of developers per requirement constraints or maximum workload constraints. Meanwhile, some measures such as the project overrun risk or the fairness among the employees are used as objectives for optimisation.

Let $E = \{e_1, \ldots, e_m\}$ be a team of m employees who are intended to develop the requirements. Each employee is qualified by a set of skills, which she or he masters. Analogously, each requirement involves a set of skills for its implementation. Let $S(e_i)$ and $S(r_j)$ be the sets of skills corresponding to employee e_i and requirement r_j, respectively. Employee e_i can be allocated to requirement r_j only if she masters all the skills required, i.e. $S(r_j) \subseteq S(e_i)$.

A solution to this problem can be represented by a binary decision matrix $Y = [y_{ij}]_{m \times n}$ encoding the allocation of employees to requirements in such a way that $y_{ij} = 1$ when e_i is allocated to r_j and $y_{ij} = 0$ otherwise.

Henceforth, let Y_{i*} be the row vector $[y_{i1}, \ldots, y_{in}]$ and Y_{*j} the column vector $[y_{1j}, \ldots, y_{mj}]^\mathsf{T}$. Y_{i*} encodes which requirements are assigned to employee e_i, while Y_{*j} encodes which employees are allocated to requirement r_j.

We consider here two additional sets of constraints to satisfy: the uniqueness constraints and the workload constraints. Although one employee can be allocated to different tasks, each task can only be assigned to a unique employee. Thus, if all the requirements must be fulfilled, $\sum Y_{*j} = 1$ for every requirement r_j. Moreover, from the perspective of human resource management, employees ought not to be assigned overwhelming workloads. The workload of employee e_i can be calculated as $W(e_i) = c \cdot Y_{i*}$. A maximum workload T_E is fixed for all the employees. Therefore, $W(e_i) \leq T_E$ for any employee e_i.

In order to strive for fairness regarding the workloads of the employees and try to avoid peaks and troughs of workload as much as possible, we are interested in minimising the variance of workloads. Formally, the mathematical optimisation model corresponding to the above software scheduling planning problem can be expressed as follows:

$$
\begin{aligned}
&\min \quad \mathrm{Var}(Y) = \frac{1}{m} \sum_{1 \le i \le m} (W(e_i) - \mu)^2 \\
&\text{where} \\
&\qquad \mu = \frac{1}{m} \sum_{1 \le i \le m} W(e_i) \\
&\text{subject to} \\
&\qquad y_{ij} = 1 \to S(r_j) \subseteq S(e_i) \quad \text{for all } e_i \in E, r_j \in R \\
&\qquad y_{ij} \in \{0, 1\} \quad \text{for all } e_i \in E, r_j \in R \\
&\qquad \sum Y_{*j} = 1 \quad \text{for all } r_j \in R \\
&\qquad \max_{e_i \in E} W(e_i) \le T_E
\end{aligned}
\tag{5.2}
$$

This is not an ILP any more, as the objective function is quadratic. However, instances of this problem can be solved using Quadratic Programming (QP).

5.2.3 Integrated Release and Schedule Planning

Today, it is still common practice to handle software release plans and software schedule plans in isolation, addressing the latter after the former in a well-defined sequence of two stages. As a result, the software schedule plans produced in the second stage may not be able to properly allocate resources to the release plan produced in the first stage.

One reason for this limitation is that, by doing so, software release planning is oblivious of the constraints existing among resources, while software schedule planning is addressed too late to fix the problem. Goals in both processes can be conflicting and optimisation is a complex problem on its own. Managing processes in stages gives more latitude to the first stage and makes the work in the second stage harder. This favours less than optimal plans and, if care is not taken, even inconsistent plans. In the best case, several iterations may be needed to obtain reasonable plans, which is far from effective and can be prone to error.

Aiming to overcome this limitation, Li et al. [47] suggested that both processes should be coordinated or, even better, integrated in one software planning process. This integrated planning was addressed as a single-objective optimisation problem. However, this approach has its own limitation: that requirement engineers have to make the decision to combine different objectives into a single objective. The usual approach, assigning weights to each objective function and combining them into a weighted sum, has been nowadays recognised to have major drawbacks and this approach seems to have been abandoned in recent research.

Next, we formally introduce *iRASPA*, which effectively addresses those limitations and is able to properly integrate the models presented in Subsections 5.2.1 and 5.2.2 into an integrated software release and schedule planning model. Moreover, *iRASPA* is not just a model, but it is accompanied by an exact multi-objective optimisation approach, which is presented in Section 5.3.

Let S be the set of releases and $s = |S|$ the number of releases to plan or planning horizon. A solution to the integrated software planning problem can be represented by an integer decision matrix $Z = [z_{ij}]_{m \times n}$ such that $z_{ij} \in [0, s]$. Z encodes the releases in which employees are allocated to requirements: $z_{ij} = 0$ if e_i is not allocated to r_j and $z_{ij} = k > 0$ if e_i is allocated to r_j in release k.

The following map reports whether employees have been allocated to requirements, whichever the release:

$$f(Z) = [y_{ij}]_{m \times n} \quad \text{where} \quad y_{ij} = \begin{cases} 0 & \text{if } z_{ij} = 0 \\ 1 & \text{otherwise} \end{cases} \tag{5.3}$$

Besides, the following map reports whether employees have been allocated to requirements for a certain release k:

$$g(Z,k) = [t_{ij}]_{m\times n} \quad \text{where} \quad t_{ij} = \begin{cases} 1 & \text{if } z_{ij} = k \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

Henceforth, let Z_{i*} be the row vector $[z_{i1}, \ldots, z_{in}]$ and Z_{*j} the column vector $[z_{1j}, \ldots, z_{mj}]^{\mathsf{T}}$. Z_{i*} encodes which requirements are assigned to employee e_i and their planned release, while Z_{*j} encodes which employees are allocated to requirement r_j and in which release. The above maps can be applied to Z_{i*} and Z_{*j} too.

Regarding the interaction constraints, uniqueness constraints and workload constraints, we relax the uniqueness constraint to allow for optional requirements: it is not always mandatory that all the requirements are fulfilled. This provides more flexibility. Consequently, although one employee can be allocated to different tasks, each task will only be assigned to at most one employee. Therefore, $\sum f(Z_{*j}) = \sum Y_{*j} \leq 1$ for every requirement r_j.

As for the workload constraints, the workload $W(e_i)$ of employee e_i, the accumulated workload $W(E)$ of all the employees, which is the workload of the whole project, and the workload $W(s_k)$ of release s_k can be expressed as follows:

$$W(e_i) = c \cdot f(Z_{i*}) \tag{5.5}$$

$$W(E) = \sum_{1\leq i\leq m} W(e_i) \tag{5.6}$$

$$W(s_k) = \sum_{1\leq i\leq m} c \cdot g(Z_{i*}, k) \tag{5.7}$$

Our goal is maximising the overall importance or business value of all the planned releases of the product, Value(Z), while minimising the variance of workloads, Var(Z). Formally, *iRASPA*, which is the mathematical optimisation model corresponding to the above integrated software release and planning problem, can be expressed as follows:

$$
\begin{aligned}
&\max \quad \text{Value}(Z) = \sum_{1\le i\le m} v \cdot f(Z_{i*}) \\
&\min \quad \text{Var}(Z) = \frac{1}{m} \sum_{1\le i\le m} (W(e_i) - \mu)^2 \\
&\text{where} \\
&\qquad \mu = \frac{W(E)}{m} \\
&\text{subject to} \\
&\qquad W(E) \le T_P \\
&\qquad W(e_i) \le T_E && \text{for all } e_i \in E \\
&\qquad W(s_i) \le T_S && \text{for all } s_i \in S \\
&\qquad z_{ij} > 0 \to S(r_j) \subseteq S(e_i) && \text{for all } e_i \in E, r_j \in R \\
&\qquad z_{ij} \in [0, s] && \text{for all } e_i \in E, r_j \in R \\
&\qquad \sum f(Z_{*j}) \le 1 && \text{for all } r_j \in R \\
&\qquad x_i = x_j && \text{for all } r_i \ \xi \ r_j \\
&\qquad x_i = 0 \lor x_j = 0 && \text{for all } r_i \ \varphi \ r_j \\
&\qquad x_j = 0 \lor 0 < x_i \le x_j && \text{for all } r_i \ \chi \ r_j
\end{aligned}
\tag{5.8}
$$

In the *iRASPA* model, parameters T_P, T_E and T_S represent the maximum workload for the whole project, per employee and per release, respectively. Besides, $x_k = \sum Z_{*k}$ indicates the release in which requirement r_k is included.

Again, this is not an ILP, as the second objective function is quadratic. However, instances of *iRASPA* can be solved using Quadratic Programming (QP).

5.3 The Solution Approach

Since software release planning and schedule planning are both characterised by the presence of multiple complex and conflicting objectives, simply optimising one of the objectives or combining them with different weights to enable a single-objective optimisation approach is not realistic and results in suboptimal plans. Furthermore, Li et al. [3] have shown that relying exclusively upon approximation meta-heuristic algorithms can bring algorithmic uncertainty into the outcomes, which may cause information loss and lead to wrong decisions. A different approach is then needed.

We propose using exact multi-objective optimisation to find the true Pareto front of the *iRASPA* model, i.e. to compute the Pareto optimal set of solutions for any instance of the model. In our proposal, the ε-constraint technique is combined with an exact single-objective solver to produce the Pareto front. There are two essential advantages when this approach is applied. First, different objectives can be taken into account simultaneously. This enables the decision makers to understand the trade-offs between conflicting objectives and make better decisions. Second, being exact, algorithmic uncertainty can be eliminated from the results.

The ε-constraint technique is not new [199]. The underlying idea is optimising one of the objectives while transforming the rest in constraints bounded by ε. Then, ε is varied to progressively restrict the search space and the Pareto front is produced as the transformed problems are solved.

In the *iRASPA* model there are two objectives: minimising the variance of the workload, $\text{Var}(Z)$, and maximising the value of the solution, $\text{Value}(Z)$. The former can be kept as an optimisation objective while the search space is reduced by the latter, as this objective is replaced with the bounding constraint $\text{Value}(Z) \geq \varepsilon$. An exact Pareto front is produced by solving the resulting single-objective instance while the boundary ε is increased. This process is illustrated by Algorithm 4, where v is the maximum of $\text{Value}(Z)$ and p_ε is the optimal solution produced by the single-objective QP solver

Algorithm 4 *iRASPA* algorithm with ε-constraint and QP

```
Input: iRASPA model, $M$
Input: Maximum value, $v$
Output: Pareto front, $P$
  $P \leftarrow \varnothing$
  for $\varepsilon \leftarrow 0$ to $v$ do
    $p_\varepsilon \leftarrow$ QP-SOLVER$(M, \varepsilon)$
    $P \leftarrow P \cup \{p_\varepsilon\}$
  end for
  return $P$
```

under the transformed constraint Value$(Z) \geq \varepsilon$.

5.4 Empirical Study

An experimental study is conducted on seven real-world software projects. Two approaches are compared: *iRASPA* and a state-of-the-art two-stage approach. Henceforth, we will refer to the second approach as "the two-stage approach", which consists in solving the software release planning problem with an exact ILP solver, as explained in Subsection 5.2.1, and then tackling the software scheduling problem on the release plan produced in the first stage, this time with an exact QP solver, as described in Subsection 5.2.2.

5.4.1 Datasets

Seven projects, each from a different organisation, have been used to collect the data used for the experiments. They consists of five industrial projects [151] and two academia-industry cooperation projects. The latter were developed between October 2015 and May 2016 by students from University College London and engineers from a large public international IT company. Information on companies and people involved, as well as project descriptions, are protected by a Non-Disclosure Agreement (NDA). Raw data for the academia-industry cooperation projects was exported from Microsoft

Table 5.2: Project information including the following characteristics: number of requirements, dependencies, developers and skills, along with importance and effort estimations. Effort measured in person-hours for industrial projects and person-days for academia-industry cooperation projects.

Project ID	Reqs.	Deps.	Devs.	Skills	Import.	Effort
SoftChoice	245	247	N/A	22	N/A	6664
QuoteToOrder	60	64	N/A	9	N/A	547
DatabaseUpgrade	106	105	N/A	7	N/A	5390
SmartPrice	72	71	N/A	14	N/A	1570
CutOver	95	68	N/A	0	N/A	2356
Cooperation 1	98	279	7	33	189	417
Cooperation 2	46	37	5	20	119	135

Visual Studio Team Services.[1] Table 5.2 summarises key information concerning each project that can be disclosed without breaching the NDA.

Since neither information on the importance attached to each requirement nor the profile of developers is available for any of the industrial projects, we produce this information synthetically to simulate real data and cover as many real-world scenarios as possible, as described next.

First, the number of developers for each industrial project is set to 10. The expertise level of a developer for a project is measured as the percentage of the required skills mastered by the developer. Expertise levels ranging from 10% to 100% with a step of 10% are used to assess developer expertise. Developers mastering all the skills required by a project are able to work on the implementation of any of its requirements and receive an expertise level of 100%.

Second, the importance of a requirement is based on its correlation with the provided estimation of the requirement effort. In essence, this produces a value-to-cost ratio for each requirement. Pearson's correlation is used, following Harman et al. [22]. Correlation factors of 50%, 75%, 85%, 95% and 100% are considered.

Third, several studies report that the number of requirements dependencies in relation to the total number of requirements, can have a strong effect on requirements selec-

[1]Formerly, Microsoft Visual Studio Online.

tion [176, 177]. Dependency relations are generated with 20 different densities ranging from 5% to 100% with a step of 5% to study its impact in our approach.

Therefore, we have considered three independent dimensions, producing $(5+10+20)\cdot 7 = 245$ different configurations. Each configuration is an instance of a real project where we have basically filled the gaps of missing information with reasonable values covering a wide range of the possible values. Two full releases are planned in all instances.

5.4.2 Experimental Setup

Experiments were performed using the IBM CPLEX 12.6.2 Java API. A Java program builds each instance from our model and calls the solver as needed with the default CPLEX configuration. Ubuntu 14.04.4 LTS and Oracle JDK 1.8.0 have been used for development and the experiments have been run in a Microsoft Azure D4S-v2 virtual machine with one Intel E5-2673 CPU featuring 8 cores and 28 GB of RAM, at a cost of £0.8 per hour. Since the results produced by *iRASPA* are exact and deterministic, it only needs running each problem instance once, saving valuable experimentation time.

5.4.3 Research Questions

In order to evaluate *iRASPA* we compare it to a state-of-the-art implementation of the two-stage approach and assess the results on different impact factors. This motivates two research questions:

RQ1 How does *iRASPA* perform in comparison to the two-stage approach?

RQ2 What is the impact on *iRASPA* of different software project characteristics?

The first research question is a foundational question prior to adopting the *iRASPA* approach. We want to investigate how effective is *iRASPA* with respect to the two-stage approach as well as its efficiency in terms of execution time. As *iRASPA* guarantees the global optimality of the solutions, major differences between the solutions offered by both approaches could reveal information loss and suboptimality introduced by the two-stage approach.

The second research question is more concerned with scrutinising the impact of software project characteristics. It is possible that different project characteristics have a different impact on *iRASPA*. But, which are the relevant dimensions?

The effect of requirement dependencies is the first concern raised in our experiments, since several authors [170, 200] have consistently shown that they play a major role in the requirements selection process. In particular, we will investigate what is the impact of the density of the requirement dependencies.

The impact of the expertise of the developers is our second concern. On one hand, in traditional software development processes, e.g, CMMI and Waterfall, developers do specialise: each developer is charged with specific responsibilities. The skills needed to address those responsibilities are a subset of the skills required by the entire project. Allocating the developer to a task for which he is not skilled may result in project delays or even a defective product. On the other hand, in agile projects, each developer should be able to face any task and she is expected to become proficient in every skill required by the project. Therefore, we will investigate the impact of this factor, what could be of great help to project managers.

Last, but not least, the value-to-cost ratio of requirements have been shown to produce a strong impact in requirements selection for the next release problem [22]. It is important to know if this extends to the integrated release and scheduling software planning problem. We will use estimated efforts as a measure of cost and the perceived importance as a measure of value.

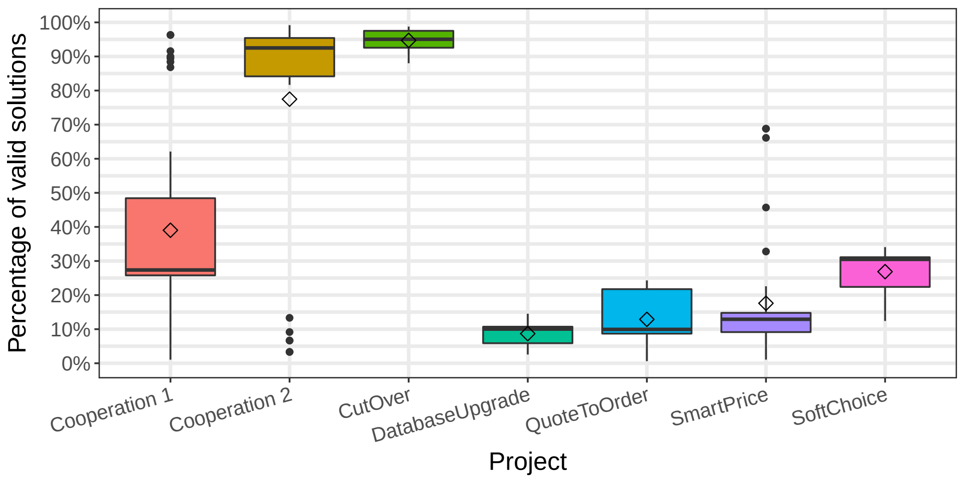


Figure 5.1: The percentage of valid solutions found by the two-stage approach.

5.4.4 Analysis of Results

We present the results of the experiments for all the instances of each project, the analysis of the results and the answers to the research questions.

RQ1: How does *iRASPA* perform in comparison to the two-stage approach?

This research question is answered by comparing the quality of the solutions found by the two-stage approach and *iRASPA*. We measure quality by analysing the solutions in the resulting Pareto fronts with four indicators: first, the percentage of valid solutions found by the two-stage approach; second, the percentage of optimal solutions found by the two-stage approach; third, the percentage of optimal solutions found by the two-stage approach in comparison with *iRASPA*; and, finally, the percentage of execution time needed to compute the Pareto fronts for the two-stage approach and *iRASPA*. For each project, the maximum execution time yielded by both approaches is used to normalise their percentages of execution time. Figures 5.1–5.4 exhibit the results of this study and summarise the relevant data to answer *RQ1*.

According to Figure 5.1, it is clear that the release plans generated by the two-stage approach are likely to be invalid when resources are going to be allocated, right at the beginning of the second stage. The two-stage approach may then fail to allocate

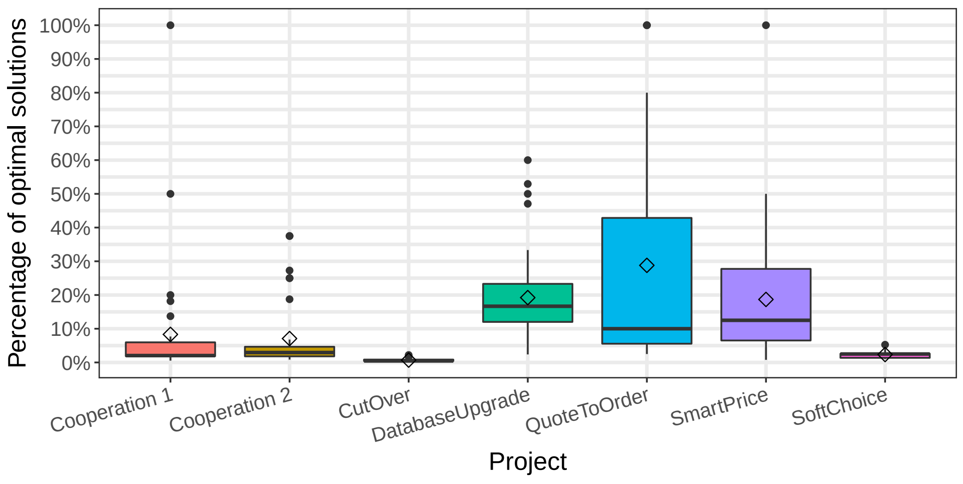


Figure 5.2: The percentage of optimal solutions found by the two-stage approach.

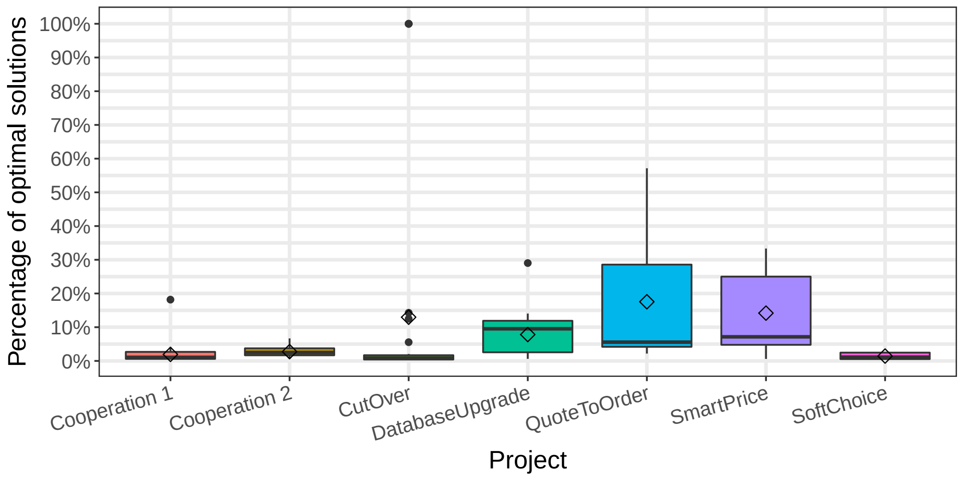


Figure 5.3: The percentage of optimal solutions found by the two-stage approach vs. *iRASPA*.

resources and we have found that this happens up to 99.34% of the time for some instances. This can lead to a situation where there are no developers who are able to effectively undertake the tasks planned and decision makers have to face project delays or defects in the product.

The two-stage approach may not only produce invalid solutions, but incomplete solutions too, missing most of the optimal solutions in the exact Pareto front, as shown in Figure 5.2. We have found that regarding the number of optimal solutions found by the two-stage approach, only an average of 12.16% are optimal. In addition, despite the two-stage approach can sometimes offer the right plans and, less frequently, even optimal plans, its incompleteness makes it difficult to explore the entire optimal solu-

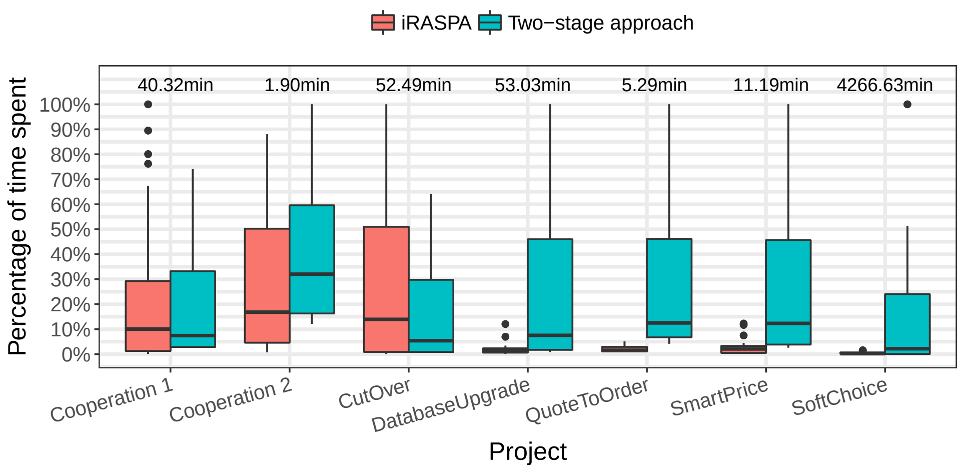


Figure 5.4: Comparison of execution times for computing the Pareto front of each instance. Maximum times on top.

tion set, as Figure 5.3 exposes. We have found that the two-stage approach can miss up to 93.38% of the optimal solutions when compared to *iRASPA*, which produces 100% of them. Even if the percentage of missing optimal solutions could be acceptable in practice for some instances, as it happens with project QUOTETOORDER, it still makes the two-stage approach to provide incomplete information for decision making.

Additionally, as shown in Figure 5.4, the two-stage approach takes often much longer than *iRASPA*, save for project CUTOVER. On average, *iRASPA* only consumes 71.77% of the execution time spent by the two-stage approach and gets every optimal solutions. On average, *iRASPA* saves 90.42 minutes when computing the Pareto fronts for every instance. As for project CUTOVER, *iRASPA* is still competitive, being only 6.02 minutes slower on average.

In summary, relying on the traditional two-stage approach cannot help software planning effectively as previously assumed by some researchers, even if a state-of-the-art implementation is provided. The above results increase our confidence in *iRASPA* as an effective and efficient approach to software release and schedule planning and provides a precise answer to *RQ1*.

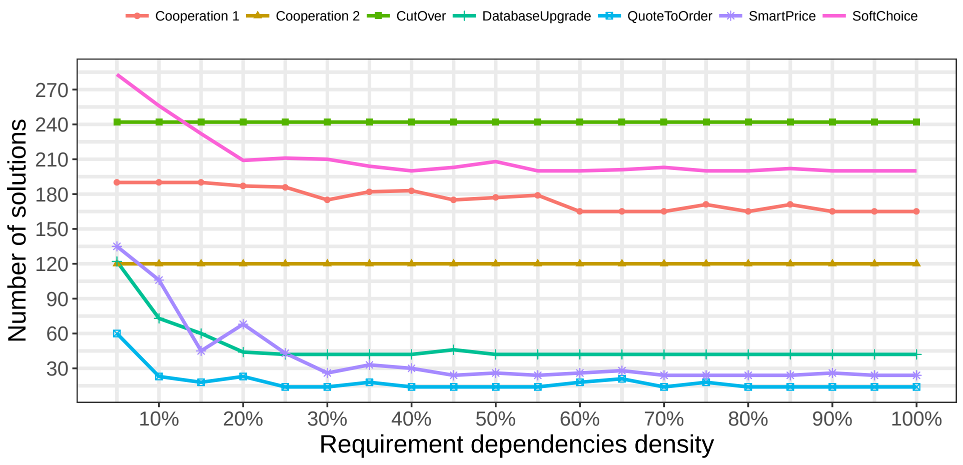
RQ2: What is the impact on *iRASPA* of different software project characteristics?

Outperforming the state-of-the-art two-stage approach seems promising, but it is necessary to assess the impact of project characteristics which could have the potential to impact the results obtained for *iRASPA*. Two metrics, number of optimal solutions and execution time, are used to quantify the impact of three different project characteristics: dependency density, developer expertise and value-to-cost ratio. Figures 5.5 and 5.6 present the results of the impact factors under consideration with respect to both metrics, the number of optimal solutions and the execution time, respectively, and provide the relevant data for *RQ2*.

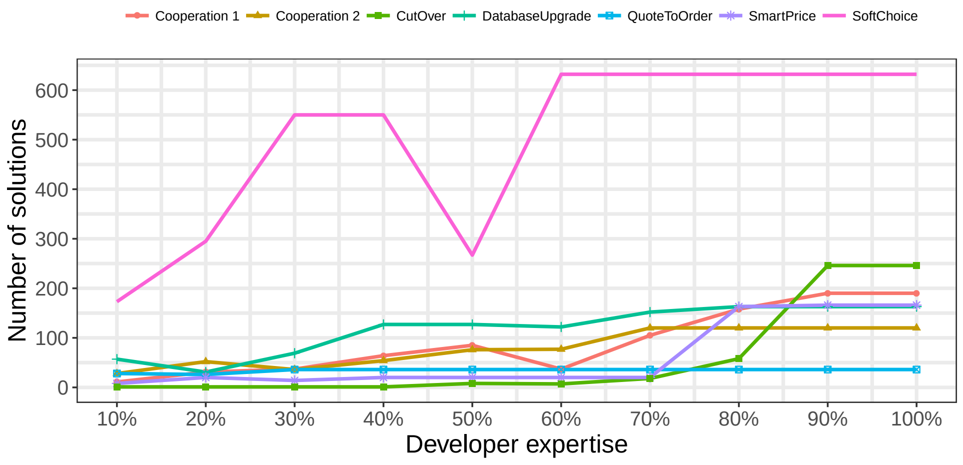
Regarding the dependency density, the higher the density, the lower the number of solutions that *iRASPA* can produce. This is not a defect in *iRASPA*, as it always produce all the existing solutions. It is just that the number of existing solutions decreases as the number of interactions grows and more constraints are imposed on requirements.

According to Figure 5.5a, it is clear that dependency density has a negative impact on the number of optimal solutions for most of the projects, with two exceptions: projects CutOver and Cooperation 2. We have found that the number of optimal solutions found is reduced up to 61.67% when the dependency density is increased by as little as 5%, even though the average reduction for this 5% increment is just 2.07%. Exceptions are possible because of some other factors that may counter the impact of a higher dependency density. For example, in project CutOver developers master all skills required and can be allocated to any task regardless how constrained the tasks are in terms of dependencies.

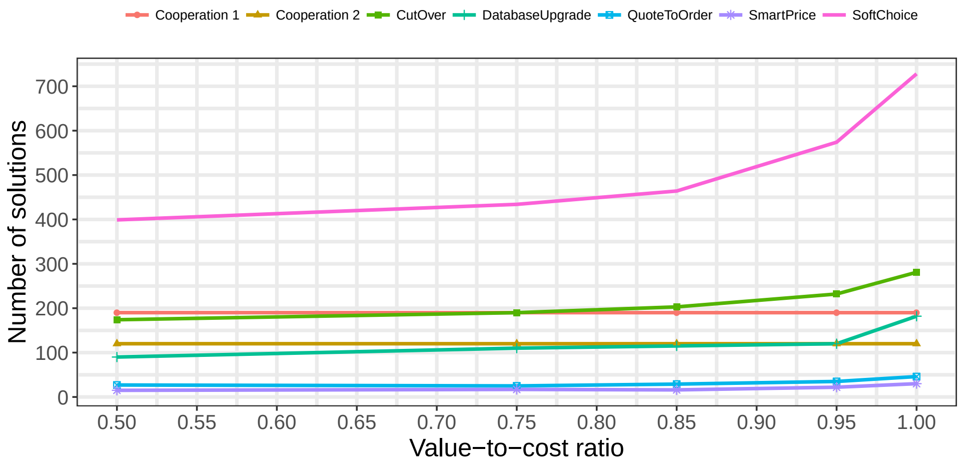
In contrast, we have found that increased developer expertise and value-to-cost ratio have a positive impact on the number of optimal solutions. In order to study the impact of developer expertise, we generate instances to simulate different developer skills for



(a) Impact of the density of the requirement dependencies.

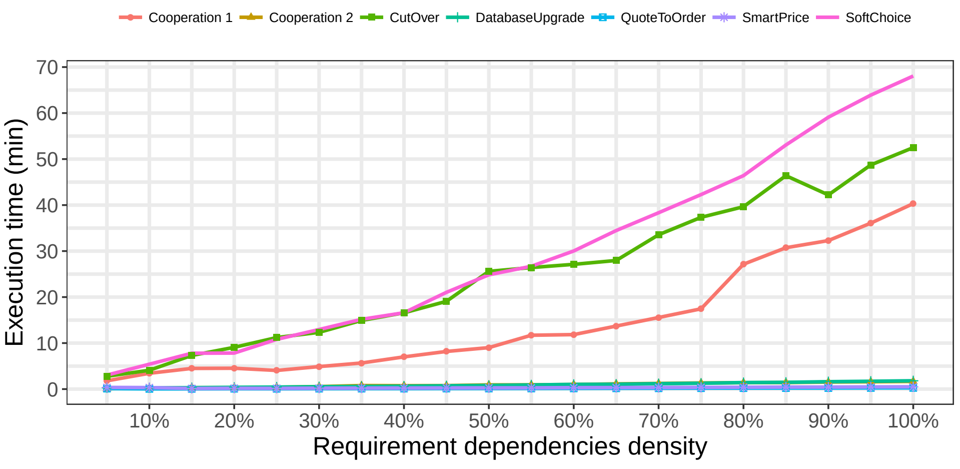


(b) Impact of the expertise of developers.

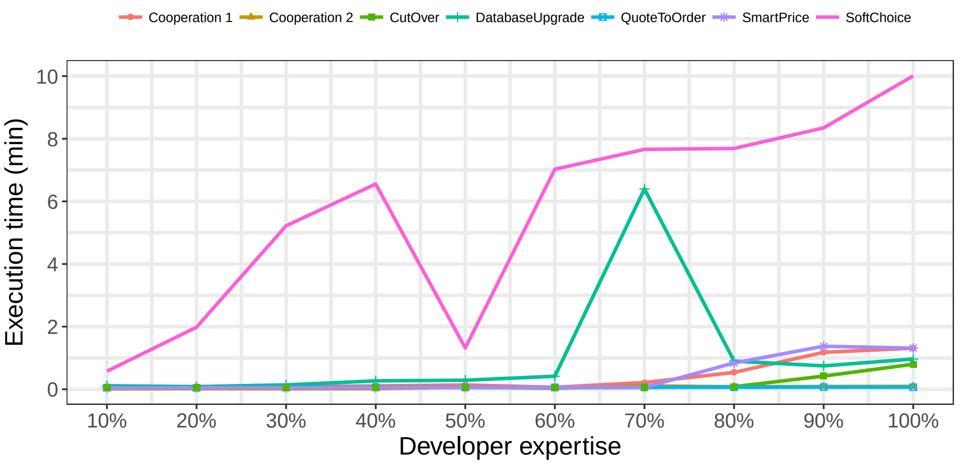


(c) Impact of the value-to-cost ratio of requirements.

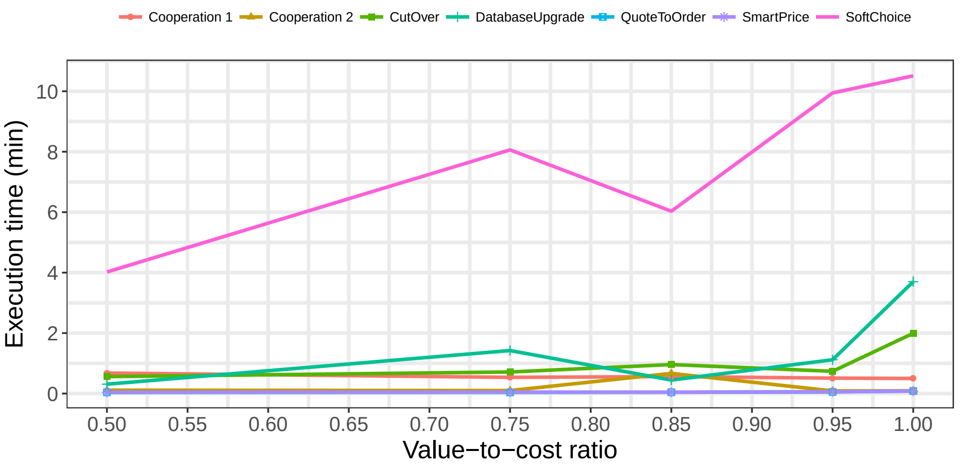
Figure 5.5: The impact of the project characteristics on the number of optimal solutions.



(a) Impact of the density of the requirement dependencies.



(b) Impact of the expertise of developers.



(c) Impact of the value-to-cost ratio of requirements.

Figure 5.6: The impact of the project characteristics on the execution time.

each project. However, in project CutOver every developer is fully qualified for all the skills demanded. Consequently, there is no skill information associated to the original project. We randomly generate 20 skills and assign them to the project requirements to enable the generation of the instances for CutOver.

As shown in Figure 5.5b, the number of optimal solutions that *iRASPA* can produce is positively correlated with developer expertise. Therefore, the more the skills mastered by developers, the higher the number of optimal solutions produced. For a 10% increment in developer expertise, the number of solutions is 54.62% higher on average and rises up to 715.00% in one extreme case: when developer expertise is increased from 70% to 80% in project SmartPrice. This circumstance can be explained by the fact that schedule planning can be more flexible if there are fewer skill constraints between developers and requirements.

Regarding the impact of the value-to-cost ratio in the number of optimal solutions, Figure 5.5c shows that an increase in this ratio produces an increase in the number of solutions too. For ratios in $[0.50, 0.95]$ the average increase is 8.34%, much smoother than for the extreme ratios in $[0.95, 1]$ where there is an almost perfect linear correlation between value and cost. For these extreme ratios, the average increase in the number of solutions rises to 24.38%.

In order to assess the impact of these factors on scalability, we analyse the execution time of *iRASPA* for increasing values of the impact factors. Figure 5.6 illustrates the results, which hint to a positive correlation between the values of the impact factors and execution time. The longest run is 68.06 minutes and corresponds to one instance of project SoftChoice, while every instance in projects Cooperation 2, QuoteToOrder, DatabaseUpgrade and SmartPrice takes less than 100 seconds. *iRASPA* can solve each instance in an average of 6.37 minutes.

However, to provide a significant statistical assessment of the dependence between impact factors and execution time, Kendall rank correlation coefficient was computed and statistical dependence was tested at 99% confidence level. Results are provided in

Table 5.3: Kendall's τ test for dependence between impact factors and execution time at **99**% confidence level.

Project ID	Density		Expertise		Value to cost	
	τ	p-value	τ	p-value	τ	p-value
SoftChoice	1.00	$\ll$0.001	0.87	$\ll$0.001	0.80	$\ll$0.001
QuoteToOrder	0.92	$\ll$0.001	0.69	$\ll$0.001	1.00	$\ll$0.001
DatabaseUpgrade	1.00	$\ll$0.001	0.78	$\ll$0.001	0.60	$\ll$0.001
SmartPrice	0.75	$\ll$0.001	0.90	$\ll$0.001	0.80	$\ll$0.001
CutOver	0.99	$\ll$0.001	1.00	$\ll$0.001	0.80	$\ll$0.001
Cooperation 1	0.98	$\ll$0.001	0.82	$\ll$0.001	0.80	$\ll$0.001
Cooperation 2	0.97	$\ll$0.001	0.78	$\ll$0.001	0.80	$\ll$0.001

Table 5.3. We have found a statistically significant dependence between all the three factors and execution time. Kendall's τ is always greater than 0.60 with associated p-values much lower than 0.001.

In summary, we have found three software project characteristics that impact both the number of optimal solutions and the time taken by *iRASPA* to find them. On one hand, our results indicate that dependency density hinders the flexibility of planning in real projects, while increased developer expertise and value-to-cost ratio have the opposite effect. Decision makers should then employ highly versatile developers instead of developers who are merely in possession of some specific talents. On the other hand, the highest execution times for *iRASPA* correspond to extremely high densities. However, we argue that such extreme densities are really infrequent in practice. According to our project data, dependency density is 2.87% on average with a peak value of 5.87% corresponding to project Cooperation 1. It would be plausible to state that dependency densities under 10% are quite common in practice. Therefore, all of this evidence indicates that the *iRASPA* approach is both useful and scalable.

5.5 Summary

In software engineering, assigning requirements to the right release and allocating resources for their implementation are key processes. Ignoring the interactions between these two software planning problems and the existing constraints makes software projects prone to cost overruns and delays. The release plan may be even invalid with respect to the resource constraints and the schedule plan would be very likely sub-optimal if it fails to account for such constraints. Trying to fix this in the final stages of project development is known to make the situation worse, leading to overworked employees and overall quality degradation [201].

In this chapter, in order to combine these two critical phases of software development in an optimal manner, we have introduced *iRASPA*, a holistic software release planning and schedule planning approach based on a QP model. An exact solver driven by an ε-constraint algorithm is used to produce optimal plans and complete Pareto fronts for two conflicting goals: maximising the overall release value and minimising the workload variance. This is the first time that QP has been used in this context and both problems have been solved simultaneously and optimally. Finally, *iRASPA* successfully addresses key challenges in software planning. This approach can provide decision makers with better alternatives and insight.

The experimental evaluation includes seven real-world software projects, coming from both academy and industry. Experiments on 245 instances derived from the projects under study show that *iRASPA* outperforms the current state-of-the-art in terms of both solution quality and execution speed. The instances take into account three different dimensions: the value-to-cost ratio of requirements, the density of interactions among requirements and the expertise of developers in terms of their skills. Experiments confirm the intuition that a higher number of dependencies can over-constrain the development and produce less flexibility in planning, while a higher value-to-cost ratio or wider range of skills in the developers produce the opposite effect.

Chapter 6

Conclusions and Future Work

In this chapter, we conclude the thesis, and review the outcomes and contributions of our introduction of exact analysis for requirements selection and optimisation. We also review the threats to validity of our research. At the end of this chapter, we sketch some potential future research directions.

6.1 Summary

In software engineering, requirements analysis and decision analysis are the critical foundations of the success of a software project. Uncertainty is essentially inevitable in early requirements engineering. It particularly comes from partially observable, stochastic environments, or ignorance. The requirements engineering community has demonstrated the success of research on quantitative multi-objective decision techniques and search-based approaches to produce optimal solutions to decision makers in the past decades [28, 30, 83, 98]. Decision makers are informed of possible trade-offs among conflicting objectives by visualising the Pareto optimal solutions generated by these quantitative methods. In these previous studies, uncertainty is either underestimated or completely overlooked [31, 83, 100, 202]. For instance, the attributes of requirements and stakeholders are quantified as explicit values, most of approaches used to deal with requirements engineering problems are approximate algorithms, and the uncertainty of resource constraints about schedule planning is entirely ignored in previous work on requirements selection and optimisation process.

Previous work on requirements selection and optimisation also suffers from important limitations from uncertainty. Little work has been done to consider uncertainties, interpret the consequences of those uncertainties, and to support decision makers in analysing the inherent characteristics of uncertainty [22, 116, 148]. Decisions have to be made under incomplete knowledge about software projects.

The goal of the thesis was to better support requirements engineers in understanding and analysing the inherent characteristics of uncertainty in the requirements selection and optimisation process. To achieve this goal, we proposed a simulation-based NRP to model requirements uncertainty, introduced an exact analysis approach to support requirements selection and optimisation in the presence of algorithmic and requirements uncertainty, and provided an integrated release and schedule planning approach to manage both algorithmic uncertainty and uncertainty of resource constraints. We also provided a more generic exact multi-objective integrated release and schedule planning approach to manage, simultaneously, both the algorithmic uncertainty and the uncertainty of resource constraints.

The contributions of the research work presented in this thesis can be summarised as follows:

6.1.1 Simulation based Robust Next Release Problem Model

We introduced a novel simulation based NRP (*sNRP*) model to provide requirements uncertainty management for requirement analysis and optimisation for the first time. We proposed to take requirements uncertainty as well as the probability of occurrence of uncertain events into account during the NRP optimisation. Compared to the original NRP formulation, we formulated one extra objective, "risk", which was simulated by

Monte-Carlo Simulation. Two notions of uncertainty were used to explain the "risk" inherit in a release plan. The first one is "Uncertainty Size", which offers decision makers a way to control the fluctuation range of *pay-off* for solutions. The second notion is "Failure Risk", which aims to help decision makers to explore solutions with lower *risk* of budget overrun.

We conducted two empirical studies to investigate the effectiveness and applicability of the proposed *sNRP*. The empirical study results reveal that the resulting risk-mitigated release plan solutions could capture the trade-offs among the three competing objectives: cost, value, and risk. Additionally, we found that the *sNRP* model could help decision makers to explore risk-less solutions, while sacrificing only a small quantity of the value in the pay-off. These validated the feasibility of *sNRP*, and provided valuable insights into the requirements uncertainty.

6.1.2 The Value of Exact Analysis in Next Release Problem

To manage and analyse uncertainty in requirements selection and optimisation, it is important for the decision maker to know that all uncertainty derives from the problem itself and not from the algorithm used to tackle it. We introduced a requirements optimisation and analysis decision support framework *METRO*, which uses an exact Next Release Problem solver, to manage both algorithmic uncertainty and requirements uncertainty.

Three experimental studies were conducted to evaluate the proposed decision support framework. In these three experimental studies, three synthetic NRP instances were derived from a real world NRP instance (according to the level of uncertainty), in order to account for the impact of estimation accuracy. There are two boundary scenarios, in which the uncertainty of a requirement is estimated either highly optimistically or pessimistically, and one 'in-between' scenario. The objectives of the experimental studies were to: 1) investigate the effectiveness of the proposed exact NRP solver, *NSGDP*, for eliminating algorithmic uncertainty; 2) analyse the impact of the requirements uncertainty; 3) help decision makers to understand the requirements characteristics and requirements inclusions.

According to the experimental results, we found that the proposed exact NRP solver (*NSGDP*) can effectively eliminate algorithmic uncertainty, and significantly reduce execution time. Also, we showed that requirements uncertainty would result in uncertainty for the overall software release plan. In order to minimise this risk, some loss of perceived utility must be accepted. Finally, requirement characteristics play an important role in their inclusion of solutions on Pareto-front. With respect to independent requirements, intrinsic uncertainty negatively correlates with inclusion when minimising solution's risk. For mutually exclusive requirements, the inclusion of either requirement relies on the dominance of this requirement's fitness value. Therefore, dominated requirements are seldom selected, compared to their conflicted twins.

6.1.3 Exact Analysis in Integrated Release and Schedule Planning Problem

Requirements selection and resource allocation are widely recognised as key aspects in software project management and, in software engineering, as a discipline [75, 107, 203]. The vast majority of the literature is concerned with managing both problems independently, dealing with the resource allocation stage after the requirements selection stage has been completed. The uncertainty of resource constraints is only taken into account in resource allocation, and thus ignored in requirements selection.

This study argued that the state-of-the-art two-stage approach may produce suboptimal results. Moreover, most of the existing researches relied solely on approximation meta-heuristic algorithms, and can only guarantee reasonable approximate solutions.

In order to address these two limitations in a holistic a systematic way, we have proposed an integrated model, *iRASPA*, which combines software release planning and software schedule planning. With *iRASPA*, an ε-constraint algorithm employs an exact solver to address both planning problems at once through a bi-objective QP model.

One unique advantage of this approach is that the exact solver can always find optimal results and the entire optimal Pareto front can be produced at the same time, thanks to the application of the ε-constraint algorithm.

By combining multiple objectives and exact methods, our approach guards us, not just against the loss of information caused (by the algorithmic uncertainty inherent to heuristic search-based techniques), but also produces the optimal Pareto front for the whole planning problem at hand, which enables decision makers to better understand the trade-offs among the various existing complex and conflicting demands.

We evaluated *iRASPA* on seven real-world software projects, instantiated as 245 instances augmented with synthetic data to cater for missing values. The experimental study shows that *iRASPA* can effectively generate the guaranteed exact Pareto front, unlike the current state-of-the-art, which misses 87.84% of the optimal solutions on average and up to 93.38% for some instances. In addition, it takes *iRASPA* 28.23% less time on average to solve all the instances under study.

6.2 Threats to Validity

In this section, the key threats to the validity of the research results presented in this thesis are discussed. Those threats are analysed with respect to three usual dimensions: construct validity, internal validity and external validity.

6.2.1 Construct validity

In this thesis, only a triangle probability distribution was used to represent requirements uncertainty. However, there are other kinds of probability distribution that might be used in risk analysis. For example, Gaussian distribution, uniform distribution, and discrete probability distribution. Catering for other distributions would not be a problem: our framework computes the estimation uncertainties of requirement attributes using MCS, and MCS can simulate most kinds of uncertainties straightforwardly. It merely needs to sample the scenarios based on input probability distribution directly. Therefore, *METRO* could use other kinds of uncertainty distribution to model the uncertainties of requirements.

The requirements selection and resource allocation model, as used throughout this research, do not contemplate all the factors that it is necessary to take into account to be able to reflect every possible real-world scenario. Some simplifications have been necessarily introduced to make the model easy to tailor to a variety of both real-world projects and synthetic data. For instance, regarding fine-grained resource allocation, the detailed time slots corresponding to the implementation of each requirement have not been considered. Developers are merely allocated to tasks in a specific release, regardless the implementation order of different tasks in the release.

However, providing fine-grain resource allocation should not be an insurmountable problem. In the approach presented in Chapter 5, the size of the constraint matrix can be augmented straightforwardly to introduce fine-grained resource constraints dealing with a higher time granularity. Therefore, we argue that the research undertaken in this thesis models the problem at a "reasonable" level of granularity and that the model can be easily extended to more detailed real-world scenarios, provided the necessary additional data is readily available.

6.2.2 Internal validity

Internal validity is concerned with any possible factor that may perturb the experimental evaluations. Typically, perturbations arise during experimental evaluations because of particular details in the implementation of algorithms and different parameter setting applied. This is especially true for heuristic and stochastic SBSE algorithms. However, by repeating the experiments a sufficient number of times, significant statistics can be extracted.

In addition, the solvers proposed in this thesis are all exact approaches, thus the stochastic properties of algorithms can be excluded. With respect to *METRO*, an exact NRP solver (*NSGDP*) is at the heart. *NSGDP* uses the Nemhauser-Ullmann algorithm, which is a dynamic programming algorithm, combined with conflict graph to solve specific instances in a decision tree solution space. Regarding *iRASPA*, it is a combination of the ε-constraint technique and an exact black-box solver. The former (the ε-constraint technique) is in charge of providing multi-objective optimisation for the problem under study by invoking the latter (an exact black-box solver) to solve a number of single-objective problems on demand.

The solver employed is CPLEX, a high-performance mathematical programming solver for linear programming, mixed integer programming and quadratic programming provided by IBM. [1] CPLEX is a state-of-the-art optimisation solver and, under appropriate conditions, it always produces exact results, provided enough computational resources are available. Although we can tune CPLEX parameters to change its behaviour, default settings have been used for the entire set of experiments and care has been taken so that exact results can be guaranteed.

The other threat to internal validity is concerned with the accuracy of the elicited probability distributions of requirements attributes. There are methods for eliciting the probability distributions of uncertainties, but such elicitation is sensitive due to the cognitive biases of the selected experts [181]. In the Chapter 4, due to the lack of uncertainty information within the RALIC data set, we generated uncertainty distributions for the estimation error of the requirement cost, informed by a literature survey. We cannot know the true estimate uncertainty for a project. Therefore, to minimise the impact of estimation accuracy, we study three synthetic NRP instances. There are two boundary scenarios, in which the uncertainty of a requirement is estimated either

[1] `www.ibm.com/software/products/en/ibmilogcpleoptistud`

highly optimistically or pessimistically, and one ‘in-between’ scenario.

Furthermore, in our experimental study, during the experiment, we excluded other system applications, so the experimental machine ran only our application. Therefore, we believe our approaches are exact and repeatable, and can provide valuable insights into decision analysis to support decision makers.

6.2.3 External validity

External validity is concerned with the extent to which it is possible to generalise the results. Namely, it may arise due to those choices of data set. In the experimental studies reported in the thesis, we evaluated different approaches on different data sets depending on what was available and appropriate for answering our research questions. Regarding *METRO*, we evaluated it on three synthetic data sets. These three data sets are derived from one real world data set from University College London, which contains two types of dependencies, and 143 raw requirements. Regarding *iRASPA*, we evaluated it versus the two-stage approach on 245 project instances with different characteristics. Those project instances are derived from seven real-world software projects: two of them are gathered from academia-industry cooperation projects and five of them are, in fact, real-world IT industry projects.

Nevertheless, there are pros and cons with either kind of project. Academic projects and academia-industry cooperation projects can provide detailed information on almost every aspect of the planning and development process but might not be fully practical and representative of truly industrial projects. Industry projects, on the other hand, do not usually provide complete project information as required by a research of these characteristics, so we have to synthetically generate the missing data in a best effort to accurately represent plausible instances.

Therefore, the results are not entirely genuine and we cannot claim that they generalise beyond the particular projects under study. For generalisation, more work is required to analyse different scenarios, models of uncertainty, as well as different NRP formulations. However, with regard to scalability, *METRO* can process the RALIC dataset (with 143 raw requirements and $10,000$ scenarios) within 40 seconds on average, and *iRASPA* can optimally plan the project instances that with more than 200 requirements, a similar number of dependencies and a major developer effort involved, within a horizon of two full releases in 6.37 minutes on average. We think that this is relevant and shows evidence of the usefulness of the approaches.

6.3 Future Work

This thesis established an initial work on exact analysis for requirements selection and optimisation. The findings of studies have implications for future investigations and potential research directions. In this section, we list several possibilities for future work.

One important direction for future work is to verify our findings by applying our approaches to more real world projects. In real world projects, there may be other formulations of the problem, objectives, or requirements interactions. In this thesis, we adopted only the basic (and classic) requirements selection and resource allocation formulation, and took three kinds of requirement dependencies into consideration. Expanding the proposed framework to more complex problem statements and requirements interactions raises a (significantly challenging) need to improve the generality of our exact technique.

The other topic in which the wider community may be interested concerns the scalability of exact requirements selection and optimisation solver. There is no doubt that the number of requirements and the number of dependencies are negatively correlated with algorithm speed. In non-trivial software projects, the number of requirements and requirements dependencies are large. This may be one of the major obstacles for using our framework in non-trivial software projects. Further work is required to investigate and improve the scalability of our approaches, especially, dealing with requirements uncertainty, algorithmic uncertainty, and uncertainty of resource constraints all together simultaneously. One solution might be to cluster requirements to reduce the size of the optimisation problem presented to the solver. Another solution might involve replacing the CPLEX solver with customised dynamic programming algorithm to further boost the implementation's speed.

Future work will also include considering other kinds of requirements uncertainty. For example, uncertainty about the extent of requirement fulfilment. Existing requirements optimisation and analysis work treats the fulfilment of requirements as an entirely discrete value. However, the requirement may be only partially fulfilled. Uncertainty about the extent of requirement fulfilment, and its impact on the overall solutions presented to decision makers have been largely ignored in requirements optimisation and analysis work, including ours. More work is required to extend our requirements uncertainty management frameworks to handle such requirements uncertainty.

Bibliography

[1] Lingbo Li, Mark Harman, Emmanuel Letier, and Yuanyuan Zhang. Robust next release problem: Handling uncertainty during optimization. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 1247–1254. ACM, 2014.

[2] Lingbo Li. Exact analysis for next release problem. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 438–443, Sept 2016.

[3] L. Li, M. Harman, F. Wu, and Y. Zhang. The value of exact analysis in requirements selection. *IEEE Transactions on Software Engineering*, 43(6):580–596, June 2017.

[4] Lingbo Li, Inmaculada Medina-Bulo, Francisco Palomo-Lozano, Mark Harman, and Yuanyuan Zhang. iRASPA: An exact multi-objective integrated release and schedule planning approach. *Submitted to IEEE Transactions on Software Engineering*, 2017. Under review.

[5] Haitao Dan, Mark Harman, Jens Krinke, Lingbo Li, Alexandru Marginean, and Fan Wu. *Pidgin Crasher: Searching for Minimised Crashing GUI Event Sequences*, chapter Search-Based Software Engineering: 6th International Symposium, SSBSE '14, Fortaleza, Brazil, August 26-29, 2014. Proceedings, pages 253–258. Springer International Publishing, Cham, 2014.

[6] Lingbo Li, Mark Harman, Fan Wu, and Yuanyuan Zhang. *SBSelector: Search Based Component Selection for Budget Hardware*, chapter Search-Based Software

Engineering: 7th International Symposium, SSBSE '15, Bergamo, Italy, September 5-7, 2015, Proceedings, pages 289–294. Springer International Publishing, Cham, 2015.

[7] Michail Basios, Lingbo Li, Fan Wu, Leslie Kanthan, Donald Lawrence, and Earl Barr. Darwinian data structure selection. *Submitted to IEEE Transactions on Software Engineering*, 2017. Under review.

[8] Michail Basios, Lingbo Li, Fan Wu, Leslie Kanthan, and Earl T. Barr. *Optimising Darwinian Data Structures on Google Guava*, pages 161–167. Springer International Publishing, Cham, 2017.

[9] Ian Sommerville. *Software Engineering: (Update) (8th Edition) (International Computer Science)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[10] Mian Li, Shapour Azarm, and Vikrant Aute. A multi-objective genetic algorithm for robust design optimization. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO'05)*, pages 771–778, New York, NY, USA, 2005. ACM.

[11] Stefan Wagner. Global sensitivity analysis of predictor models in software engineering. In *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, pages 3–3. IEEE, 2007.

[12] James C Felli and Gordon B Hazen. Sensitivity analysis and the expected value of perfect information. *Medical Decision Making*, 18(1):95–109, 1998.

[13] Bihuan Chen, Xin Peng, Yijun Yu, Bashar Nuseibeh, and Wenyun Zhao. Self-adaptation through incremental generative model transformations at runtime. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 676–687, New York, NY, USA, 2014. ACM.

[14] Xin Peng, Bihuan Chen, Yijun Yu, and Wenyun Zhao. Self-tuning of software

systems through dynamic quality tradeoff and value-based feedback control loop. *J. Syst. Softw.*, 85(12):2707–2719, December 2012.

[15] H. Yang, A. De Roeck, V. Gervasi, A. Willis, and B. Nuseibeh. Speculative requirements: Automatic detection of uncertainty in natural language requirements. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 11–20, Sept 2012.

[16] Alain Abran, Pierre Bourque, Robert Dupuis, and James W. Moore, editors. *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA, 2001.

[17] Ian F Alexander and Ljerka Beus-Dukic. *Discovering requirements: how to specify products and services*. John Wiley & Sons, 2009.

[18] J. Karlsson. Software requirements prioritizing. In *Requirements Engineering, 1996., Proceedings of the Second International Conference on*, pages 110–116, Apr 1996.

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

[20] D Greer and G Ruhe. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, 46(4):243–253, 2004.

[21] Mark Harman and Bryan F Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833–839, 2001.

[22] Mark Harman, Jens Krinke, Inmaculada Medina-Bulo, Francisco Palomo-Lozano, Jian Ren, and Shin Yoo. Exact scalable sensitivity analysis for the next release problem. *ACM Transactions on Software Engineering and Methodology*, 23(2):19:1–19:31, 2014.

[23] Hadar Ziv, Debra Richardson, and René Klösch. The uncertainty principle in

software engineering. In *Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, 1997.

[24] J.C. Helton, J.D. Johnson, C.J. Sallaberry, and C.B. Storlie. Survey of sampling-based methods for uncertainty and sensitivity analysis. *Reliability Engineering & System Safety*, 91(10-1):1175 – 1209, 2006. The Fourth International Conference on Sensitivity Analysis of Model Output (SAMO 2004) {SAMO}.

[25] Rick Salay, Marsha Chechik, Jennifer Horkoff, and Alessio Di Sandro. Managing requirements uncertainty with partial models. *Requirements Engineering*, 18(2):107–128, 2013.

[26] Christof Ebert and Jozef De Man. Requirements uncertainty: Influencing factors and concrete improvements. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 553–560, New York, NY, USA, 2005. ACM.

[27] Watts S. Humphrey. *A Discipline for Software Engineering.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1995.

[28] A. Finkelstein, M. Harman, S.A. Mansouri, Jian Ren, and Yuanyuan Zhang. "Fairness analysis" in requirements assignments. In *International Requirements Engineering, 2008. RE'08. 16th IEEE*, pages 115–124, Sept 2008.

[29] Des Greer and Günther Ruhe. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, 46(4):243–253, 2004.

[30] Yuanyuan Zhang, Mark Harman, and S. Afshin Mansouri. The multi-objective next release problem. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pages 1129–1137. ACM, 2007.

[31] Alexander Budzier. Why your it project may be riskier than you think. *Harvard Business Review*, 89(9):23–25, 2011.

[32] Douglas W Hubbard. *How to measure anything: Finding the value of intangibles in business.* John Wiley & Sons, 2014.

[33] Andrea Saltelli. Sensitivity analysis for importance assessment. *Risk Analysis*, 22(3):579–590, 2002.

[34] Andrea Saltelli, Marco Ratto, Terry Andres, Francesca Campolongo, Jessica Cariboni, Debora Gatelli, Michaela Saisana, and Stefano Tarantola. *Global sensitivity analysis: the primer.* Wiley. com, 2008.

[35] Mark Harman, Jens Krinke, Jian Ren, and Shin Yoo. Search based data sensitivity analysis applied to requirement engineering. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO'09)*, pages 1681–1688, New York, NY, USA, 2009. ACM.

[36] A. Saltelli, S. Tarantola, and F. Campolongo. Sensitivity analysis as an ingredient of modeling. *Statistical Science*, 15(4):377–395, 2000.

[37] Hans-Georg Beyer and Bernhard Sendhoff. Robust optimization–a comprehensive survey. *Computer Methods in Aplied Mechanics and Engineering*, 196(33):3190–3218, 2007.

[38] A. L. Soyster. Technical note: convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(5):1154–1157, 1973.

[39] Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Rev.*, 53(3):464–501, August 2011.

[40] Matheus Paixão and Jerffeson Souza. A scenario-based robust model for the next release problem. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation Conference (GECCO'13)*, pages 1469–1476, New York, NY, USA, 2013. ACM.

[41] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization.* Princeton University Press, 2009.

[42] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, MA, USA, 1992.

[43] Hans-Paul Schwefel. *Numerical Optimization of Computer Models.* John Wiley & Sons, Inc., New York, NY, USA, 1981.

[44] Franz Rothlauf. *Design of Modern Heuristics: Principles and Application*, chapter Optimization Methods, pages 45–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[45] P. Festa. A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems. In *Transparent Optical Networks (ICTON), 2014 16th International Conference on*, pages 1–20, July 2014.

[46] M. Pawlak. Application of evolution program to resource demand optimisation in project planning. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 1, pages 435–, Nov 1995.

[47] Chen Li, Marjan van den Akker, Sjaak Brinkkemper, and Guido Diepen. An integrated approach for requirement selection and scheduling in software release planning. *Requirements Engineering*, 15(4):375–396, 2010.

[48] Barry Boehm. A view of 20th and 21st century software engineering. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 12–29, New York, NY, USA, 2006. ACM.

[49] Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, Dec 1990.

[50] Frederick P. Brooks, Jr. No silver bullet essence and accidents of software engineering. *Computer*, 20(4):10–19, April 1987.

[51] Klaus Pohl. *The three dimensions of requirements engineering*, pages 275–292. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[52] Ian Sommerville and Gerald Kotonya. *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[53] Yuzo Yamamoto, Richard V. Morris, Christopher Hartsough, and E. David Callender. The role of requirements analysis in the system life cycle. In *Proceedings of the June 7-10, 1982, National Computer Conference*, AFIPS '82, pages 381–387, New York, NY, USA, 1982. ACM.

[54] MD Richter, JD Mason, MW Alford, IF Burns, and HA Helton. Software requirements engineering methodology. Technical report, DTIC Document, 1976.

[55] Merlin Dorfman. System and software requirements engineering. In *IEEE Computer Society Press Tutorial*, pages 7–22. IEEE Computer Society Press, 1990.

[56] Systems and software engineering – life cycle processes –requirements engineering. *ISO/IEC/IEEE 29148:2011(E)*, pages 1–94, Dec 2011.

[57] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 35–46, New York, NY, USA, 2000. ACM.

[58] R. J. Wieringa. *Requirements Engineering: Frameworks for Understanding*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[59] Mich Luisa, Franch Mariangela, and Inverardi Pierluigi. Market research for requirements analysis using linguistic tools. *Requir. Eng.*, 9(1):40–56, February 2004.

[60] Axel van Lamsweerde. Formal specification: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 147–159, New York, NY, USA, 2000. ACM.

[61] Marie-Claude Gaudel. Formal specification techniques (extended abstract). In *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, pages 223–227, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[62] Robert M. Hierons, Kirill Bogdanov, Jonathan P. Bowen, Rance Cleaveland, John Derrick, Jeremy Dick, Marian Gheorghe, Mark Harman, Kalpesh Kapoor, Paul Krause, Gerald Lüttgen, Anthony J. H. Simons, Sergiy Vilkomir, Martin R. Woodward, and Hussein Zedan. Using formal specifications to support testing. *ACM Comput. Surv.*, 41(2):9:1–9:76, February 2009.

[63] Ieee guide–adoption of the project management institute (pmi(r)) standard a guide to the project management body of knowledge (pmbok(r) guide)–fourth edition. *IEEE Std 1490-2011*, pages 1–508, Nov 2011.

[64] Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques.* Springer Publishing Company, Incorporated, 1st edition, 2010.

[65] Pei Hsia, David Kung, and Chris Sell. Software requirements and acceptance testing. *Ann. Softw. Eng.*, 3:291–317, January 1997.

[66] Q. Wang and X. Lai. Requirements management for the incremental development model. In *Proceedings of the Second Asia-Pacific Conference on Quality Software*, APAQS '01, pages 295–, Washington, DC, USA, 2001. IEEE Computer Society.

[67] James Herbsleb, David Zubrow, Dennis Goldenson, Will Hayes, and Mark Paulk. Software quality and the capability maturity model. *Commun. ACM*, 40(6):30–40, June 1997.

[68] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. Capability maturity model, version 1.1. *IEEE Softw.*, 10(4):18–27, July 1993.

[69] O. C Z Gotel and A C W Finkelstein. An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 94–101, Apr 1994.

[70] B. Ramesh, T. Powers, C. Stubbs, and M. Edwards. Implementing requirements traceability: A case study. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, RE '95, pages 89–, Washington, DC, USA, 1995. IEEE Computer Society.

[71] Raymond T Yeh and Peter A Ng. Software requirements?a management perspective. *System and Software Requirements Engineering*, pages 450–641, 1990.

[72] Yuanyuan Zhang. *Multi-Objective Search-based Requirements Selection and Optimisation*. University of London, 2010.

[73] T.L. Saaty. *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. Advanced book program. McGraw-Hill, 1980.

[74] Lai-Kow Chan and Ming-Lu Wu. Quality function deployment: A literature review. *European Journal of Operational Research*, 143(3):463 – 497, 2002.

[75] Joachim Karlsson and Kevin Ryan. A cost-value approach for prioritizing requirements. *IEEE Software*, 14(5):67–74, 1997.

[76] Joachim Karlsson, Claes Wohlin, and Björn Regnell. An evaluation of methods for prioritizing software requirements. *Information and Software Technology*, 39(14?15):939 – 947, 1998.

[77] Jason Eisner. State-of-the-art algorithms for minimum spanning trees - a tutorial discussion, 1997.

[78] W Issel. Aho, av, je hopcroft, jd ullman: Data structures and algorithms. addison-wesley amsterdam 1983. 436 s. *Biometrical Journal*, 26(4):390–390, 1984.

[79] Peter F Windley. Trees, forests and rearranging. *The Computer Journal*, 3(2):84–88, 1960.

[80] A.D. Booth and A.J.T. Colin. On the efficiency of a new method of dictionary construction. *Information and Control*, 3(4):327 – 334, 1960.

[81] Björn Regnell, Martin Höst, Natt Johan och Dag, Per Beremark, and Thomas Hjelm. An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software. *Requirements Engineering*, 6(1):51–62, 2001.

[82] Soo Ling Lim, Daniela Damian, and Anthony Finkelstein. Stakesource2.0: Using social networks of stakeholders to identify and prioritise requirements. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 1022–1024, New York, NY, USA, 2011. ACM.

[83] Yuanyuan Zhang, Anthony Finkelstein, and Mark Harman. Search based requirements optimisation: Existing work and challenges. In Barbara Paech and Colette Rolland, editors, *Requirements Engineering: Foundation for Software Quality*, volume 5025 of *Lecture Notes in Computer Science*, pages 88–94. Springer Berlin Heidelberg, 2008.

[84] M. S. Feather and T. Menzies. Converging on the optimal attainment of requirements. In *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*, pages 263–270, 2002.

[85] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[86] Christopher C. Skiścim and Bruce L. Golden. Optimization by simulated annealing: A preliminary computational study for the tsp. In *Proceedings of the 15th Conference on Winter Simulation - Volume 2*, WSC '83, pages 523–535, Piscataway, NJ, USA, 1983. IEEE Press.

[87] Günther Ruhe and An Ngo. Hybrid intelligence in software release planning. *Int. J. Hybrid Intell. Syst.*, 1(1-2):99–110, April 2004.

[88] G. Ruhe and M. O. Saliu. The art and science of software release planning. *IEEE Software*, 22(6):47–53, Nov 2005.

[89] An Ngo-The and Günther Ruhe. Optimized resource allocation for software release planning. *IEEE Trans. Softw. Eng.*, 35(1):109–123, January 2009.

[90] Mark Harman, Alexandros Skaliotis, Kathleen Steinhöfel, and Paul Baker. Search–based approaches to the component selection and prioritization problem. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 1951–1952, New York, NY, USA, 2006. ACM.

[91] He Jiang, Jingyuan Zhang, Jifeng Xuan, Zhilei Ren, and Yan Hu. A hybrid aco algorithm for the next release problem. In *Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on*, pages 166–171, June 2010.

[92] Paul Baker, Mark Harman, Kathleen Steinhofel, and Alexandros Skaliotis. Search based approaches to component selection and prioritization for the next release problem. In *Proceedings of the 22Nd IEEE International Conference on Software Maintenance*, ICSM '06, pages 176–185, Washington, DC, USA, 2006. IEEE Computer Society.

[93] Brian J. Ritzel, J. Wayland Eheart, and S. Ranjithan. Using genetic algorithms to solve a multiple objective groundwater pollution containment problem. *Water Resources Research*, 30(5):1589–1603, 1994.

[94] Carlos A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):269–308, 1999.

[95] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys*, 45(1):11:1–11:61, 2012.

[96] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective

genetic algorithm: NSGA-II. *Transaction on Evoluionary Computation*, 6(2):182–197, Apr 2002.

[97] A. Finkelstein, M. Harman, S. A. Mansouri, J. Ren, and Y. Zhang. Fairness analysis; in requirements assignments. In *International Requirements Engineering, 2008. RE '08. 16th IEEE*, pages 115–124, Sept 2008.

[98] Yuanyuan Zhang, Enrique Alba, Juan J. Durillo, Sigrid Eldh, and Mark Harman. Today/future importance analysis. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO'10)*, pages 1357–1364, New York, NY, USA, 2010. ACM.

[99] Yuanyuan Zhang and Mark Harman. Search based optimization of requirements interaction management. In *Proceedings of the 2nd International Symposium on Search Based Software Engineering*, SSBSE '10, pages 47–56. IEEE Computer Society, 2010.

[100] Juan J. Durillo, YuanYuan Zhang, Enrique Alba, and Antonio J. Nebro. A study of the multi-objective next release problem. In *Proceedings of the 2009 1st International Symposium on Search Based Software Engineering*, SSBSE '09, pages 49–58, Washington, DC, USA, 2009. IEEE Computer Society.

[101] Yuanyuan Zhang, Mark Harman, Anthony Finkelstein, and S. Afshin Mansouri. Comparing the performance of metaheuristics for the analysis of multi-stakeholder tradeoffs in requirements optimisation. *Information and Software Technology*, 53(7):761 – 773, 2011.

[102] Antonio J. Nebro, Juan J. Durillo, Francisco Luna, Bernabé Dorronsoro, and Enrique Alba. Mocell: A cellular genetic algorithm for multiobjective optimization. *Int. J. Intell. Syst.*, 24(7):726–746, July 2009.

[103] K. Praditwong and Xin Yao. A new multi-objective evolutionary optimisation algorithm: The two-archive algorithm. In *Computational Intelligence and Security, 2006 International Conference on*, volume 1, pages 286–291, Nov 2006.

[104] Yuanyuan Zhang, Mark Harman, Gabriela Ochoa, Guenther Ruhe, and Sjaak Brinkkemper. An empirical study of meta- and hyper-heuristic search for multi-objective release planning. *UCL Department of Computer Science Research Note*, 14(07), 2014.

[105] Alan W. Johnson and Sheldon H. Jacobson. A class of convergent generalized hill climbing algorithms. *Appl. Math. Comput.*, 125(2-3):359–373, January 2002.

[106] Peter Ross. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter Hyper-Heuristics, pages 529–556. Springer US, Boston, MA, 2005.

[107] Ho-Won Jung. Optimizing value and cost in requirements analysis. *IEEE Software*, 15(4):74–78, 1998.

[108] Pär Carlshamre. Release planning in market-driven software product development: Provoking an understanding. *Requirements Engineering*, 7(3):139–151, 2002.

[109] Marjan Van Den Akker, Sjaak Brinkkemper, Guido Diepen, Johan Versendaal, et al. Flexible release composition using integer linear programming. *UU-CS*, (2004-063), 2004.

[110] Marjan van den Akker, Sjaak Brinkkemper, G van Diepen, and Johan Versendaal. Flexible release planning using integer linear programming. *REFSQ'05*, 2005.

[111] Marjan van den Akker, Sjaak Brinkkemper, Guido Diepen, and Johan Versendaal. Software product release planning through optimization and what-if analysis. *Information and Software Technology*, 50(1?2):101–111, 2008. Special issue with two special sections. Section 1: Most-cited software engineering articles in 2001. Section 2: Requirement engineering: Foundation for software quality.

[112] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.

[113] C. Li, J. M. Van Den Akker, S. Brinkkemper, and G. Diepen. Integrated requirement selection and scheduling for the release planning of a software product. In *Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality*, REFSQ '07, pages 93–108, Berlin, Heidelberg, 2007. Springer-Verlag.

[114] Chen Li, Marjan Akker, Sjaak Brinkkemper, and Guido Diepen. An integrated approach for requirement selection and scheduling in software release planning. *Requirements Engineering*, 15(4):375–396, 2010.

[115] G. L. Nemhauser and Z. Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505, 1969.

[116] Emmanuel Letier, David Stefan, and Earl T. Barr. Uncertainty, risk, and information value in software requirements and architecture. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 883–894, New York, NY, USA, 2014. ACM.

[117] Nadarajen Veerapen, Gabriela Ochoa, Mark Harman, and Edmund K. Burke. An integer linear programming approach to the single and bi-objective next release problem. *Information and Software Technology*, 65:1 – 13, 2015.

[118] Hwang Ching-Lai and Syed Md Masud Abu. *Multiple objective decision making, methods and applications: a state-of-the-art survey.* Springer-Verlag, 1979.

[119] Y. Li, J. Chen, and L. Feng. Dealing with uncertainty: A survey of theories and practices. *IEEE Transactions on Knowledge and Data Engineering*, 25(11):2463–2482, Nov 2013.

[120] National Research Council (US). Committee on Risk-Based Analysis for Flood Damage Reduction. *Risk analysis and uncertainty in flood damage reduction studies.* Natl Academy Pr, 2000.

[121] Frank H Knight. Risk, uncertainty and profit. *New York: Hart, Schaffner and Marx*, 1921.

[122] Gabriele Bammer and Michael Smithson. *Uncertainty and risk: multidisciplinary perspectives*. Routledge, 2012.

[123] Shabnam Rasoulian and Luis Alberto Ricardez-Sandoval. Uncertainty analysis and robust optimization of multiscale process systems with application to epitaxial thin film growth. *Chemical Engineering Science*, 116:590 – 600, 2014.

[124] Donovan Chaffart, Shabnam Rasoulian, and Luis A. Ricardez-Sandoval. Distributional uncertainty analysis and robust optimization in spatially heterogeneous multiscale process systems. *AIChE Journal*, pages n/a–n/a, 2016.

[125] Michael D McKay. Sensitivity and uncertainty analysis using a statistical sample of input values. *Uncertainty analysis*, pages 145–186, 1988.

[126] Güzin Bayraksan. *Monte Carlo sampling-based methods in stochastic programming*. PhD thesis, University of Texas at Austin, 2005.

[127] A. Saltelli, K. Chan, and E.M. Scott. *Sensitivity Analysis*. Number no. 2008 in Wiley paperback series. Wiley, 2009.

[128] Eric Smith. Uncertainty analysis. *Encyclopedia of environmetrics*, 2006.

[129] Lakshmi S Dutt and Mathew Kurian. Handling of uncertainty–a survey. *International Journal of Scientific and Research Publications*, 3:2250–315, 2013.

[130] Z. Zi. Sensitivity analysis approaches applied to systems biology models. *Systems Biology, IET*, 5(6):336–346, Nov 2011.

[131] Ahmed Al-Emran, Puneet Kapur, Dietmar Pfahl, and Guenther Ruhe. Studying the impact of uncertainty in operational release planning - an integrated method and its initial evaluation. *Information and Software Technology*, 52(4):446–461, April 2010.

[132] Ahmed Al-Emran, Dietmar Pfahl, and Guenther Ruhe. Decision support for product release planning based on robustness analysis. In *Proceedings of the 2010*

18th IEEE International Requirements Engineering Conference (RE'10), pages 157–166, Washington, DC, USA, 2010. IEEE Computer Society.

[133] George L Nemhauser and Zev Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505, 1969.

[134] He Zhang, B. Kitchenham, and D. Pfahl. Software process simulation modeling: Facts, trends and directions. In *Software Engineering Conference, 2008. APSEC '08. 15th Asia-Pacific*, pages 59–66, Dec 2008.

[135] Aharon Ben-Tal and Arkadi Nemirovski. Robust optimization ? methodology and applications. *Mathematical Programming*, 92(3):453–480, 2002.

[136] A. L. Soyster. Technical note: convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(5):1154–1157, 1973.

[137] Aharon Ben-Tal and Arkadi Nemirovski. Robust optimization–methodology and applications. *Mathematical Programming*, 92(3):453–480, 2002.

[138] Laurent El Ghaoui and Hervé Lebret. Robust solutions to least-squares problems with uncertain data. *SIAM J. Matrix Anal. Appl.*, 18(4):1035–1064, October 1997.

[139] A. Ben-Tal and A. Nemirovski. Robust convex optimization. *Math. Oper. Res.*, 23(4):769–805, November 1998.

[140] A. Ben-Tal, A. Nemirovski, and C. Roos. Robust solutions of uncertain quadratic and conic-quadratic problems. *SIAM J. on Optimization*, 13(2):535–560, June 2002.

[141] Indraneel Das. Robustness optimization for constrained nonlinear programming problems. *Engineering Optimization*, 32(5):585–618, 2000.

[142] Sigrún Andradóttir. A review of simulation optimization techniques. In *Proceedings of the 30th Conference on Winter Simulation*, WSC '98, pages 151–158, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[143] Emre Kazancioglu, Guangquan Wu, Jeonghan Ko, Stanislav Bohac, Zoran Filipi, S Jack Hu, Dennis Assanis, and Kazuhiro Saitou. Robust optimization of an automotive valvetrain using a multiobjective genetic algorithm. In *ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 97–108. American Society of Mechanical Engineers, 2003.

[144] M. Papadrakakis, N.D. Lagaros, and V. Plevris. Design optimization of steel structures considering uncertainties. *Engineering Structures*, 27(9):1408 – 1418, 2005.

[145] J.W. Herrmann. A genetic algorithm for minimax optimization problems. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2, pages –1103 Vol. 2, 1999.

[146] Adrian Thompson and Paul Layzell. Evolution of robustness in an electronics design. In Julian Miller, Adrian Thompson, Peter Thomson, and TerenceC. Fogarty, editors, *Evolvable Systems: From Biology to Hardware*, volume 1801 of *Lecture Notes in Computer Science*, pages 218–228. Springer Berlin Heidelberg, 2000.

[147] Apurva Kumar, Andy J. Keane, Prasanth B. Nair, and Shahrokh Shahpar. Robust design of compressor fan blades against erosion. *Journal of Mechanical Design*, 128(4):864–873, 2006.

[148] W. Heaven and E. Letier. Simulating and optimising design decisions in quantitative goal models. In *Requirements Engineering Conference (RE), 2011 19th IEEE International*, pages 79–88, Aug 2011.

[149] Jaak Jurison. Software project management: The manager's view. *Commun. AIS*, 2(3es), November 1999.

[150] A. Ngo-The and G. Ruhe. Optimized resource allocation for software release planning. *IEEE Transactions on Software Engineering*, 35(1):109–123, 2009.

[151] Filomena Ferrucci, Mark Harman, Jian Ren, and Federica Sarro. Not going to take this anymore: Multi-objective overtime planning for software engineering projects. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 462–471. IEEE Press, 2013.

[152] Gunther Ruhe and Moshood Omolade Saliu. The art and science of software release planning. *IEEE Software*, 22(6):47–53, 2005.

[153] Guenther Ruhe and Joseph Momoh. Strategic release planning and evaluation of operational feasibility. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, HICSS '05, pages 313.2–313.2. IEEE Computer Society, 2005.

[154] Linda A. Macaulay. *Requirements Engineering.* Springer-Verlag, London, UK, UK, 1996.

[155] Luis Daniel Otero, Grisselle Centeno, Alex J. Ruiz-Torres, and Carlos E. Otero. A systematic approach for resource allocation in software projects. *Computers & Industrial Engineering*, 56(4):1333 – 1339, 2009.

[156] Silvia T. Acuna, Natalia Juristo, and Ana M. Moreno. Emphasizing human capabilities in software development. *IEEE Software*, 23(2):94–101, March 2006.

[157] F. Sarro, F. Ferrucci, M. Harman, A. Manna, and J. Ren. Adaptive multi-objective evolutionary algorithms for overtime planning in software projects. *IEEE Transactions on Software Engineering*, 2017. To appear.

[158] Tarek K. Abdel-Hamid. The dynamics of software project staffing: A system dynamics based simulation approach. *IEEE Trans. Softw. Eng.*, 15(2):109–119, February 1989.

[159] Tarek Abdel-Hamid and Stuart E. Madnick. *Software Project Dynamics: An Integrated Approach.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[160] G. Antoniol, M. Di Penta, and M. Harman. Search-based techniques applied to optimization of project planning for a massive maintenance project. In *21st IEEE International Conference on Software Maintenance (ICSM'05)*, pages 240–249, Sept 2005.

[161] Raymond Ho-Leung TSOI. Using analytic hierarchy process (ahp) method to prioritise human resources in substitution problem. *International Journal of the Computer, the Internet and Management*, 9(1):36–45, 2001.

[162] Santanu Kr. Misra and Amitava Ray. Article: Software developer selection: A holistic approach for an eclectic decision. *International Journal of Computer Applications*, 47(1):12–18, June 2012.

[163] Luis Daniel Otero, Grisselle Centeno, Alex J. Ruiz-Torres, and Carlos E. Otero. A systematic approach for resource allocation in software projects. *Comput. Ind. Eng.*, 56(4):1333–1339, May 2009.

[164] Carl K. Chang, Mark J. Christensen, and Tao Zhang. Genetic algorithms for project management. *Annals of Software Engineering*, 11(1):107–139, 2001.

[165] Jim Duggan, Jason Byrne, and Gerard J. Lyons. A task allocation optimizer for software construction. *IEEE Softw.*, 21(3):76–82, May 2004.

[166] Silvia T. Acuña and Natalia Juristo. Assigning people to roles in software projects. *Softw. Pract. Exper.*, 34(7):675–696, June 2004.

[167] Barry W. Boehm. *Software Engineering Economics.* Prentice Hall, Upper Saddle River, NJ, USA, 1st edition, 1981.

[168] John Michael Hammersley, David Christopher Handscomb, and George Weiss. Monte carlo methods. *Physics Today*, 18:55, 1965.

[169] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations research*, 52(1):35–53, 2004.

[170] Yuanyuan Zhang, Mark Harman, and Soo Ling Lim. Empirical evaluation of search based requirements interaction management. *Information and Software Technology*, 55(1):126–152, 2013. Special section: Best papers from the 2nd International Symposium on Search Based Software Engineering 2010.

[171] Alper Atamtürk, George L. Nemhauser, and Martin W.P. Savelsbergh. Conflict graphs in solving integer programming problems. *European Journal of Operational Research*, 121(1):40 – 55, 2000.

[172] Klaus Jansen and Sabine Öhring. Approximation algorithms for time constrained scheduling. *Inf. Comput.*, 132(2):85–108, February 1997.

[173] Karla L. Hoffman and Manfred Padberg. Solving airline crew scheduling problems by branch-and-cut. *Manage. Sci.*, 39(6):657–682, June 1993.

[174] Ulrich Pferschy and Joachim Schauer. The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications*, 13(2):233–249, 2009.

[175] Tobias Brunsch and Heiko Röglin. Improved smoothed analysis of multiobjective optimization. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 407–426, New York, NY, USA, 2012. ACM.

[176] Pär Carlshamre, Kristian Sandahl, Mikael Lindvall, Björn Regnell, and Johan Nattoch Dag. An industrial survey of requirements interdependencies in software product release planning. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, RE '01, pages 84–91. IEEE Computer Society, 2001.

[177] William N. Robinson, Suzanne D. Pawlowski, and Vecheslav Volkov. Requirements interaction management. *ACM Computing Surveys*, 35(2):132–190, 2003.

[178] Walker Royce. *Software project management.* Pearson Education India, 1998.

[179] Tom DeMarco and Tim Lister. *Waltzing with bears: Managing risk on software projects.* Addison-Wesley, 2013.

[180] Roy Schmidt, Kalle Lyytinen, Mark Keil, and Paul Cule. Identifying software project risks: An international delphi study. *J. Manage. Inf. Syst.*, 17(4):5–36, March 2001.

[181] Anthony O'Hagan, Caitlin E Buck, Alireza Daneshkhah, J Richard Eiser, Paul H Garthwaite, David J Jenkinson, Jeremy E Oakley, and Tim Rakow. *Uncertain judgements: eliciting experts' probabilities.* John Wiley & Sons, 2006.

[182] J. W. Pratt. Risk aversion in the small and in the large. *Econometrica*, 32(1/2):122–136, 1964.

[183] Soo Ling Lim and Anthony Finkelstein. Stakerare: Using social networks and collaborative filtering for large-scale requirements elicitation. *IEEE Trans. Softw. Eng.*, 38(3):707–735, May 2012.

[184] Michael Bloch, Sven Blumberg, and Jürgen Laartz. Delivering large-scale it projects on time, on budget, and on value. *Harvard Business Review*, 2011.

[185] Magne Jørgensen and Kjetil Moløkken-Østvold. How large are software cost overruns? a review of the 1994 {CHAOS} report. *Information and Software Technology*, 48(4):297 – 301, 2006.

[186] Jennifer Lynch. The standish group report. `http://www.standishgroup.com/sample_research_files/chaos_report_1994.pdf`, 1994. Accessed: 2015-01-12.

[187] N.D. Fogelstrom, M. Svahnberg, and T. Gorschek. Investigating impact of business risk on requirements selection decisions. In *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on*, pages 217–223, Aug 2009.

[188] Edward Yourdon. *Death March: The Complete Software Developer's Guide to Surviving Mission Impossible Projects.* Prentice Hall PTR, 1997.

[189] L. L. Minku, D. Sudholt, and X. Yao. Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis. *IEEE Transactions on Software Engineering*, 40(1):83–102, Jan 2014.

[190] X. Shen, L. L. Minku, R. Bahsoon, and X. Yao. Dynamic software project scheduling through a proactive-rescheduling method. *IEEE Transactions on Software Engineering*, 42(7):658–686, July 2016.

[191] Mark Harman. The current state and future of search based software engineering. In *2007 Future of Software Engineering*, FOSE '07, pages 342–357. IEEE Computer Society, 2007.

[192] Ahmed Al-Emran, Puneet Kapur, Dietmar Pfahl, and Guenther Ruhe. Studying the impact of uncertainty in operational release planning - an integrated method and its initial evaluation. *Inf. Softw. Technol.*, 52(4):446–461, 2010.

[193] George Mavrotas. Effective implementation of the ε-constraint method in multi-objective mathematical programming problems. *Applied Mathematics and Computation*, 213(2):455–465, 2009.

[194] J. Blazewicz, J.K. Lenstra, and A.H.G.Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[195] Pär Carlshamre. Release planning in market-driven software product development: Provoking an understanding. *Requirements Engineering*, 7(3):139–151, 2002.

[196] Filomena Ferrucci, Mark Harman, and Federica Sarro. *Search-Based Software Project Management*, pages 373–399. Springer-Verlag, 2014.

[197] Wei-Neng Chen and Jun Zhang. Ant colony optimization for software project scheduling and staffing with an event-based scheduler. *IEEE Trans. Softw. Eng.*, 39(1):1–17, January 2013.

[198] Jian Ren, Mark Harman, and Massimiliano Di Penta. *Cooperative Co-evolutionary Optimization of Software Project Staff Assignments and Job Scheduling*, pages 127–141. Springer-Verlag, 2011.

[199] Yv Haimes, Ls Lasdon, and Da Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-1(3):296–297, July 1971.

[200] Junjie Wang, Juan Li, Qing Wang, He Zhang, and Haitao Wang. A simulation approach for impact analysis of requirement volatility considering dependency change. In *Proceedings of the 18th International Conference on Requirements Engineering: Foundation for Software Quality*, REFSQ '12, pages 59–76. Springer-Verlag, 2012.

[201] Frederick P. Brooks, Jr. *The Mythical Man-Month: Essays on Software Engineering (Anniversary Ed.)*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[202] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. *Department of Computer Science, King?s College London, Tech. Rep. TR-09-03*, 2009.

[203] Linda Rising and Norman S. Janoff. The scrum software development process for small teams. *IEEE Software*, 17(4):26–32, 2000.

[204] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, 13(5):533–549, May 1986.

[205] J D Farmer, N H Packard, and A S Perelson. The immune system, adaptation, and machine learning. *Phys. D*, 2(1-3):187–204, October 1986.

[206] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[207] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*, pages 39–43, Oct 1995.

[208] Mark Harman, Edmund Burke, John Clark, and Xin Yao. Dynamic adaptive search based software engineering. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '12, pages 1–8, New York, NY, USA, 2012. ACM.

[209] Richard Fairley. Risk management for software projects. *IEEE Softw.*, 11(3):57–67, May 1994.

[210] Barry W. Boehm. Software risk management: Principles and practices. *IEEE Softw.*, 8(1):32–41, January 1991.

[211] Yudistira Asnar, Paolo Giorgini, and John Mylopoulos. Goal-driven risk assessment in requirements engineering. *Requir. Eng.*, 16(2):101–116, June 2011.

[212] Moshood Omolade Saliu and Guenther Ruhe. Bi-objective release planning for evolving software systems. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC-FSE '07, pages 105–114, New York, NY, USA, 2007. ACM.

[213] Marjan van den Akker, Sjaak Brinkkemper, Guido Diepen, and Johan Versendaal. Software product release planning through optimization and what-if analysis. *Inf. Softw. Technol.*, 50(1-2):101–111, January 2008.

[214] Dietmar Pfahl, Ahmed Al-Emran, and Günther Ruhe. A system dynamics simulation model for analyzing the stability of software release plans: Research sections. *Softw. Process*, 12(5):475–490, September 2007.

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

[216] Hugh W Coleman and W Glenn Steele. *Experimentation, validation, and uncertainty analysis for engineers.* John Wiley & Sons Incorporated, 2009.

[217] Günther Ruhe and Des Greer. Quantitative studies in software release planning under risk and resource constraints. In *Proceedings of the 2003 International Symposium on Empirical Software Engineering*, ISESE '03, pages 262–, Washington, DC, USA, 2003. IEEE Computer Society.

[218] Susan DP Harker, Ken D Eason, and John E 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, Jan 1993.

[219] L. Li. Exact analysis for next release problem. In *2016 IEEE 24th International Requirements Engineering Conference*, RE '16, pages 438–443, 2016.

[220] J. J. Durillo, Y. Zhang, E. Alba, and A. J. Nebro. A study of the multi-objective next release problem. In *2009 1st International Symposium on Search Based Software Engineering*, SSBSE '09, pages 49–58, 2009.

[221] Jian Ren, Mark Harman, and Massimiliano Di Penta. Cooperative co-evolutionary optimization of software project staff assignments and job scheduling. In *Proceedings of the Third International Conference on Search Based Software Engineering*, SSBSE '11, pages 127–141. Springer-Verlag, 2011.

[222] James E. Kelley, Jr and Morgan R. Walker. Critical-path planning and scheduling. In *Papers Presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference*, IRE-AIEE-ACM '59 (Eastern), pages 160–173, New York, NY, USA, 1959. ACM.

[223] Quentin W Fleming and Joel M Koppelman. *Earned value project management.* Project Mangement Institute, 2000.