Release Planning For Multi-tenant Software as a Service (SaaS) Applications

by

Mubarak Alrashoud

B.S. in Information and Computer Science, King Saud University, Saudi Arabia, 1996

M.S. in Computer Science, King Abdulaziz University, Saudi Arabia, 2008

A dissertation

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy in the program of Computer Science

Toronto, Ontario, Canada, 2015

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

Mubarak Alrashoud

Release Planning For Multi-tenant Software as a Service (SaaS) Applications

Mubarak Alrashoud

Doctor of Philosophy program in Computer Science, 2015

Ryerson University

ABSTRACT

In multi-tenant Software as a Service (SaaS) applications, the providers are required to regularly deliver new releases of the software in order to satisfy the evolving requirements of tenants. The first step in a release development lifecycle is the release planning process.

This thesis formulates the problem of the "next release" planning for multi-tenant Software as a Service (SaaS) applications. Two variables that influence release planning in SaaS applications are introduced: the degree of commonality of features and the contractual constraints. The commonality of a feature denotes the number of tenants that have requested that feature. The contractual constraints denote the effects of service levels to which tenants have subscribed on the release planning process.

Furthermore, this thesis proposes three novel approaches in order to tackle the problem of the "next release" planning for multi-tenant SaaS applications. The first one is a prioritization approach that employs a Fuzzy Inference System (FIS) engine in order to speed up the release planning process and overcome the uncertainty associated with the human judgment. In this approach, the human expertise, which is represented by fuzzy rules, is considered automatically in the release planning process. The second and third approaches consider release planning as an optimization problem. The second approach uses an exact optimization method (Binary Linear

Programming (BLP)) in order to generate an optimal release plan, while the third approach uses heuristic optimization method (Genetic Algorithm (GA)). All of the three approaches aim to generate a plan for the next release that maximizes the degree of overall tenants' satisfaction, maximizes the degree of commonality, and minimizes the potential risk while taking into account contractual, effort, and dependencies constraints.

Moreover, the thesis presents an experimental study of the proposed approaches in order to determine which approach is best suited to different sets of scenarios. In this experiment, the performance of the proposed approaches is evaluated using four criteria: the overall tenants' satisfaction, the commonality, the adherence to the risk, and the running time. Additionally, the thesis presents an experiment that compares the proposed approaches with a compared model that is selected from the literature.

ACKNOWLEDGEMENTS

I would like to express my special appreciation and thanks to my supervisor, Professor Abdoloreza Abhari for his tremendous support, valuable advice, and for the time and effort that he spent to make this thesis possible. His encouragement and guidance gave me the confidence and motivation needed to finish this research. I would also like to express my gratitude to my thesis committee members, Professor Alex Ferworn, Professor Andriy Miranskyy, Professor Cherie Ding, Professor EIbrahim Bagheri, Professor Soosan Beheshti, Professor Vojislav Misic for the time and effort they spent to review my thesis and for their helpful and useful advice and comments. My sincere appreciation is extended to Professor Jean-Pierre Corriveau, Carleton University, for agreeing to be in my committee and provide me with his valuable comments.

I am very grateful to my family for their support and patience. Without their assistance, I would not have finished this thesis.

I thank King Saud University for financial support.

Last but not least, I thank all the members of our research group in the distributed system and multimedia processing (DSMP http://dsmp.ryerson.ca/) lab for their support

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AHP	Analytical Hierarchy Approach
BLP	Binary Linear Programming
CP	Constraint Programming
CRM	Customer Relationship Management
CSP	Constraint Satisfaction Problem
FIS	Fuzzy Inference System
GA	Genetic Algorithm
IaaS	Infrastructure as a Service
ILP	Integer Linear Programming
MISO	Multi Input Single Output
NIST	National Institute of Standards and Technology
OWA	Ordered Weighted Averaging
PaaS	Platform as a Service
QFD	Quality Function Deployment
RP	Release Planner
SaaS	Software as a Service
SDP	Service Delivery Platform
SEDS	Software Engineering Decision Support
SLA	Service Level Agreement
SPL	Software Product Line
WS-BPEL	Web Services Business Process Execution Language

CHAPTER 1: INTRODUCTION

When undertaking software engineering projects, many decisions must be made on the basis of uncertain, incomplete, volatile, and/or conflicting information [1]. These decisions must consider varied and even contradictory goals (such as performance, time to market, and customer satisfaction). In addition, they have to take into account resources and technical constraints [1]. Because of the complexity of software engineering processes, human intelligence cannot deal with the range of interrelated and complex decision factors. Therefore, computational intelligence must support human intelligence and knowledge in the decision making process. The area of Software Engineering Decision Support (SEDS) [2] has emerged to deal with decisions related to software engineering activities. SEDS is concerned with providing decision makers with the necessary aids to analyze the available alternatives and select those that are optimal (or near optimal). In SEDS, human knowledge and intelligence, along with well-established methodologies from other disciplines, are employed in order to manipulate hard and soft decision factors to reach the best possible decisions [3].

The research described in this thesis is located under the umbrella of utilizing SEDS in the engineering of multi-tenant Software as a Service (SaaS) systems. More precisely, it deals with the problem of how SaaS managers can most effectively plan their next software release. Release planning can be defined as the process of selecting the features that should be implemented in a certain release (in this research, the next release of an SaaS application)

In this thesis, we propose a new formulation of the problem of the next release planning for multi-tenant Software as a Service (SaaS) applications. After that, we propose three novel approaches that support SaaS release managers in planning for the next release.

The objective of the proposed approaches is to maximize tenants' satisfaction, maximize degree of commonality (selecting the features that are required by the highest number of tenants), and minimize the risk, while taking into account the effort, technical, and contractual constraints. The first approach exploits the simplicity and strength of Fuzzy Inference System (FIS) in representing human knowledge to assign a rank for each software feature. Hereafter, we call this approach FIS-based. Ranks of features represent their priorities and importance. The features are prioritized according to their ranks. The features with the highest ranks have better chance to be assigned to the next release. The novelty of the proposed FIS-based approach is that it deals with uncertainty associated with human judgments, which depend on approximation rather than exactness. FIS depends on the concept of fuzzy reasoning which mimics the way of human reasoning by manipulating human judgements using predefined linguistic rules. These rules use linguistic terms to represent the knowledge of the domain experts. In all of the previously published works on release planning, human expertise has been involved manually as the final step (when the choice is made between alternatives that have been generated by release planning models). Blending the human expertise automatically with release planning models is a more appropriate way to address release planning problems for two reasons: 1) it allows human expertise to influence the release planning models implicitly, which will increase the applicability of these models, and 2) when there is a very large number of requirements and stakeholders, it is of great benefit to be able to use an automatic method to consider human knowledge (which can be expressed in linguistic terms) in determining the final solution. Adjusting final solutions manually is very difficult in such situations. Moreover, FIS-based approach is fast, simple, intuitive, and can be adjusted easily according to the changes in the

management's policies. For example, if the release management wants to consider the importance of features more than other factors, the rules can be tuned in order to cope with this policy. The second proposed approach considers the "next release" planning for multi-tenant SaaS as an optimization problem. A release plan is represented as a vector of decision variables $X = [x_1\ x_2 ...\ x_n]$ where $x_i \in \{0,1\}$. If $x_i = 1$ then the feature f_i is assigned to the next release; otherwise, it is postponed to a future release. This approach utilizes Binary Linear Programming (BLP) in order to generate an optimal plan for the next release of an SaaS application. BLP-based solution deals with release planning as an integer linear programming with adding binary constraints on the problem variables. The third approach considers the "next release" planning for Multi-tenant SaaS as an optimization problem, and provides Genetic Algorithm-based (GA-based) solution (heuristic optimization). GA-based approach depends on the concept of the evolution to better solutions through a set of crossover, mutation operations on many generations. This thesis claims that each one of these three approaches is suitable for certain circumstances. In order to determine that, we conduct a set of experiments that measures the effectiveness of each approach under different conditions.

The rest of this chapter is organized as follows: Section 1.1 presents the background information about SaaS applications. The problem statement is sated briefly in Section 1.2. Section 1.3 summarizes the methodology that is used in this research. The contributions and the outline of the thesis are stated in sections 1.4 and 1.5 respectively.

1.1 Software as a Service (SaaS)

The National Institute of Standards and Technology (NIST) defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a

shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [4]. From this definition, one can infer that cloud computing is an on-demand service where the cloud's consumers can reach the computation resources without any direct interaction with the cloud provider. The consumers can use heterogeneous devices and software interface to select, subscribe, and immediately use the service. This makes cloud computing highly effective in saving consumers time and effort. Cloud computing depends on the concept of resource pooling; i.e., resources are pooled to serve multiple consumers using a multi-tenant model [5]. There are three main cloud-computing models: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). Cloud providers in IaaS have their own physical resources, such as servers, storage, and processing time. Using virtualization mechanisms, virtual resources are offered to cloud consumers [5,6]. In PaaS, a provider offers ready-to-use programming and deployment resources, such as integrated development environment (IDE), testing approaches, database management systems and deployment approaches. Cloud consumers in PaaS can use these resources to build their applications without any modification to the provided platform services. In SaaS, the cloud provider offers software applications as a service to customers. In a multi-tenant SaaS application, many tenants use a thin client (such as web browser) to access a SaaS application. This application runs on a service delivery platform (SDP) (network, servers, end, etc.). The SaaS provider leases resources from an SDP provider. In some cases, the SaaS provider owns the SDP. SaaS software can be developed by a third party, which is called a SaaS developer. For convenience, we assume in this thesis that “SaaS provider” refers to the organization

that develops and provides the software. Figure 1.1 shows the different roles and layers in multi-tenant SaaS applications.

Using SaaS applications allows tenants to eliminate the expenses of establishing IT infrastructure. They do not need to budget for huge up-front costs to purchase hardware, software licenses, and other IT infrastructures components. At the same time, SaaS providers can serve a huge number of customers using a single shared instance of an application. This guarantees for SaaS providers a recurring amount of revenue with less maintenance and management effort because they only maintain one codebase.

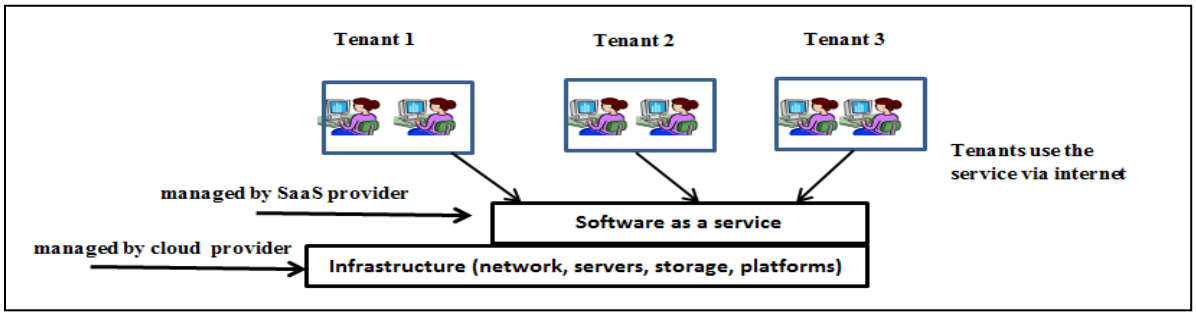


Figure 1.1: Multi-tenant SaaS Application Layers and Roles

Popularity is a significant indicator of successful SaaS applications. Highly popular software means that more customers are attracted; consequently, more profit is achieved. Ongoing satisfaction of evolving tenants' needs can attract more tenants, and guarantee the loyalty of the current ones [6]. In order to achieve that, SaaS providers frequently deliver new and high quality releases of the application during its lifecycle, as show in Figure 1.2. Each new release includes new or enhanced features. For example, Salesforce releases four major versions of their CRM software annually [7]. The provided feature shall attract highest possible number of tenants, which means SaaS providers shall include the features that are important to tenants with considering the quality. However, sometimes because of

limitations in time and resources, technical constraints, and risk, SaaS providers can implement fewer features than are requested by tenants. Therefore, SaaS providers must decide which features will be assigned to the early releases, and which ones will be postponed to later releases. Release planning is the process that carries out this planning endeavor. Release plans are the output of the release planning process [8]. Each plan for a release contains the features that will be implemented in that release. SaaS applications have many distinctive characteristics that should be considered in release planning process. Table 1.1 shows these characteristics with their effects on the release planning process.

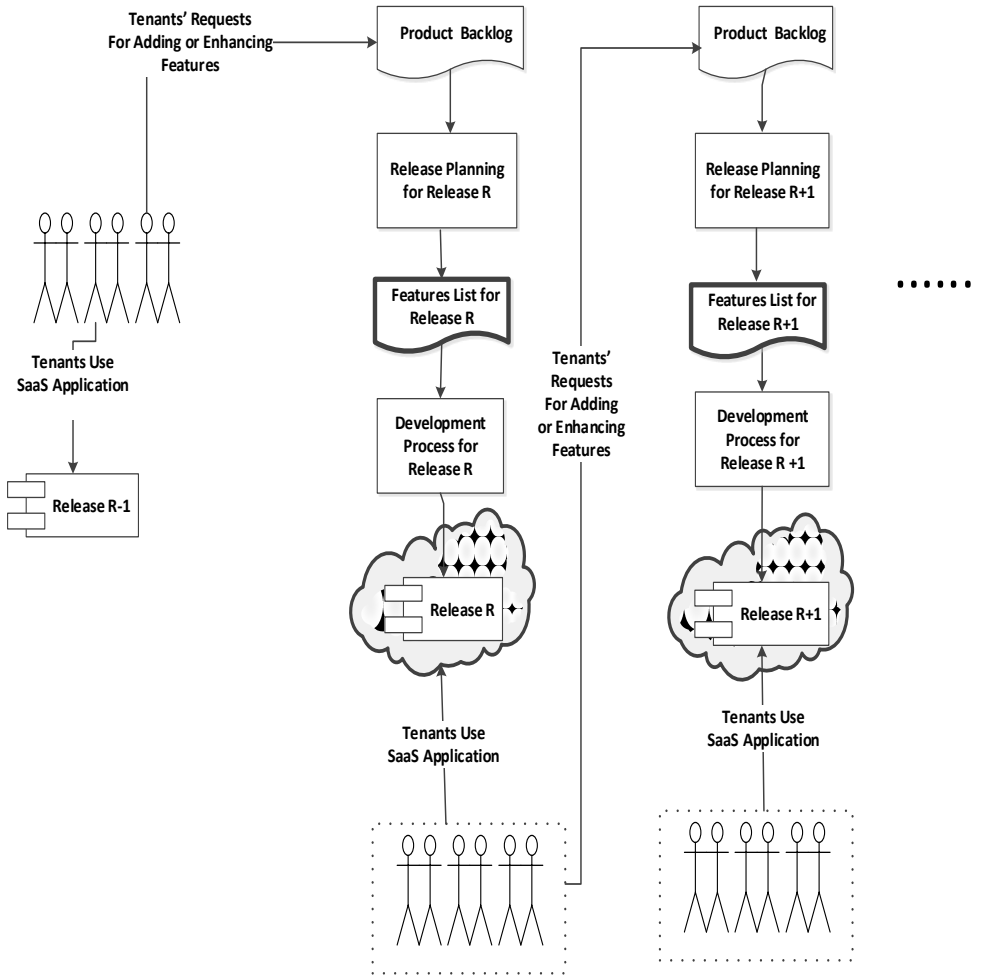


Figure 1.2: Incremental Development for SaaS Applications

This research assumes that release management plans only for the next release. Next release planning was introduced first in [108]. It deals with delivering a set of features that will be implemented in the next release. The reason behind the assumption of planning for the next release is that the extreme dynamics involved in SaaS applications, which greatly increases the possibility of a wide range of changes being required within a short time period.

Table 1.1: Characteristic of SaaS Applications and their Effects on Release Planning

SaaS Characteristics	The Effects on the Release Planning Process
Software is shared among huge number of tenants and accessed via internet.	1) Risk associated with data integrity and security [9, 10]. 2) The features that are requested by high numbers of tenants are preferred to be delivered in the early releases [6, 11]; therefore, the commonality of features should be considered.
SaaS is offered in different service levels and the tenants can change from a service level to another service instantly [12].	The service level to which tenants subscribe should be considered.
Seamless delivery of the software.	Release planning should be continuous, fast, and for short period of time [13].

For example, during a short time period, new tenants with new needs may subscribe to the SaaS application, and others who are already using the service may unsubscribe, which may change the tenants list in a very fast base. Additionally, in huge SaaS applications, hundreds of features may be added to the product backlog in a very short period. This means that the state of the product backlog can extremely change. This increase the chance of changing the priorities of features during the time between delivering a release and implementing a release after.

The next release plan shall:

- Maximize the tenants' satisfaction, by selecting the features that are important to the highest possible number of tenants.
- Consider the tenants' decision weights (the importance, volume of trade, or loyalty of each tenant to the SaaS application).
- Maximize the degree of commonality, by selecting the features that are required by the highest possible number of tenants).
- Minimize the risk of delivering low-quality release by selecting the features that have the lowest possible risk.
- Fulfill the dependencies constraints among features (for example, perhaps for technical reasons a feature A can only be delivered after a feature B).
- Fulfill contractual constraints, which are documented in the service level agreement (SLA) of each tenant.
- Fulfill the effort constraint by ensuring that the total effort required to implement the generated release plan is less than or equal to the available effort.

This thesis considers all of these factors in the problem statement and the proposed approaches.

1.2 Problem Statement

Given a set of tenants $T = \{t_1, t_2, \dots, t_m\}$, let $F = \{F_{t1}, F_{t2} \dots, F_{tm}\}$ be a family of sets of features that are requested by the tenants, where F_{ti} represents a set of features which are requested by a tenant t_i. Let $F^* = \bigcup_{i=1}^{m} F_{ti} = \{f_1, f_2 \dots f_n\}$ be the unified set that contains all features, where n is the total number of features, and $n \leq \sum_{i=1}^{m} |F_{ti}|$ and $|F_{ti}|$ is the cardinality of the set F_{ti}. It is required to find $F^{\wedge}$, which is a set of features that represents the release plan for the next release, such that $F^{\wedge} \subseteq F^*$. $F^{\wedge}$ can be defined by its characteristic function $G_{F^{\wedge}}: F^* \rightarrow \{0,1\}$ such that

$$G_{F^{\wedge}}(f_i) = \begin{cases} 1 & for\ f_i \in F^{\wedge} \\ 0 & for\ f_i \notin F^{\wedge} \end{cases} \qquad i = 1 \dots n \qquad (1.1)$$

A release plan $F^{\wedge}$ shall achieve three objectives: 1) maximizing the tenants' satisfaction; 2) maximizing the degree of commonality; and 3) minimizing the risk. A release plan $F^{\wedge}$ shall satisfy three constraints: 1) the dependencies among features, 2) contractual constraints (which are documented in the service level agreement of each tenant), and 3) the effort constraint by ensuring that the total effort required to implement the generated release plan is less than or equal to the available effort. These objectives and constraints are stated more formally in the next two sections.

1.2.1 Planning Objectives

$F^{\wedge}$ shall maximize the degree of overall satisfaction of all tenants. The degree of a tenant's satisfaction is calculated by the function $Sats(t_i, F_{ti}^{\wedge}, F^{\wedge}) \in [0\ 1]$, where $F_{ti}^{\wedge}$ is the

desired release plan to t_i (the features that achieve the maximum degree of satisfaction to tenant t_i). The degree of overall satisfaction ($OSat$) can be calculated as the additive weighting of the degree of satisfactions of the tenants.

$$OSat = \sum_{i=1}^{m} Sats(t_i, F_{ti}^{\wedge}, F^{\wedge}) \times W(t_i) \qquad (1.2)$$

where $W(t_i)$ is the decision weight of a tenant t_i and m is the number of tenants.

$F^{\wedge}$ shall maximize the degree of commonality; that is, it shall include the highest possible number of features that are required by the highest possible number of tenants. The commonality of the release plan is calculated by the function $Com\_RP(F^{\wedge}) \in [0,1]$. Furthermore, $F^{\wedge}$ shall minimize the potential risk by including the features that have the lowest possible risk. There are many risk factors that shall be considered when planning for the next release of an SaaS application. These factors are described in details in section 3.1.3. The most significant risk factor in SaaS applications is the data integrity and security. Let $Ad\_risk\_factor\ (F^{\wedge}) \in [0,1]$ be the function that calculates the degree of adherence to the risk factor. The risk of $F^{\wedge}$ can be calculated by the function $Risk(F^{\wedge})$, where

$$Risk(F^{\wedge}) = 1 - Ad\_risk\_factor\ (F^{\wedge}) \qquad (1.3)$$

1.2.2 Planning Constraints

A release plan $F^{\wedge}$ shall fulfill the contractual constraints. Let $S = \{S_1, S_2, \ldots S_p\}$ represents the levels of service of an SaaS application. Different tenants can subscribe to

different levels of service. Each service level S_i can be considered as a set of features that are included, or can be included in that level. Let $F\_Se\_L\ (f_i) \in \mathcal{P}(S)$ be a function that returns the service levels that will include the feature f_i such that $\mathcal{P}(S)$ is the power set of S. Let $Get\_Se\_L(t_i) \in S$ be the function that returns the service level to which the tenant t_i has subscribed. Then $F^{\wedge}$ shall satisfy the following constraint:

$$\forall\ f_i\ \in F^{\wedge}\ \rightarrow \exists\ t_j \in T:\ Get\_Se\_L(t_j)\ \in\ F\_Se\_L\ (f_i) \qquad (1.4)$$

which means for each feature assigned to the next release, at least the SLA of one tenant must comply with that feature. The features that comply with the service levels of high number of tenants have higher chance to be included in the next release. The effect of this constraint on the release plan process is discussed in details in section 3.2.1. Effort is another significant factor that $F^{\wedge}$ shall satisfy [8]. The required effort to implement the next release shall be less than or equal to the available effort. Formally, $Required\_Effort \leq Available\_Effort$ such that $Required\_Effort$ is the total effort needed to implement the selected features, and $Available\_Effort$ is the effort that release management can afford. The dependencies constraints [8] are technical constraints that significantly affect the content of a release plan $F^{\wedge}$. This thesis considers two types of dependencies:

- **Coupling** :Two features or more are described as "coupled" when they should be delivered in the same release.
- **Precedence :**One feature f_i precedes another feature f_j when f_i should be delivered (or at least implemented and tested) prior to feature f_j.

1.3 Methodology and the Proposed Approaches

In order to address the problem of the next release planning for SaaS applications, three approaches are proposed: The first proposed approach is the FIS-based approach. As Figure 1.3 shows, FIS-based approach consists of the following processes:

- **Raw data collection**: In this stage, the estimates that are provided by stakeholders about the different release planning factors are collected. As Table 1.2 shows, three types of stakeholders are involved in this process: tenants, development team, and release managers. Each type of stakeholder provides certain types of information. For example, development team is the party responsible for estimating risk of features [14], release management is responsible for determining the decision weights of tenants [8] and the required and available efforts [15], and tenants (customers) provide the estimates of the importance of features [1].

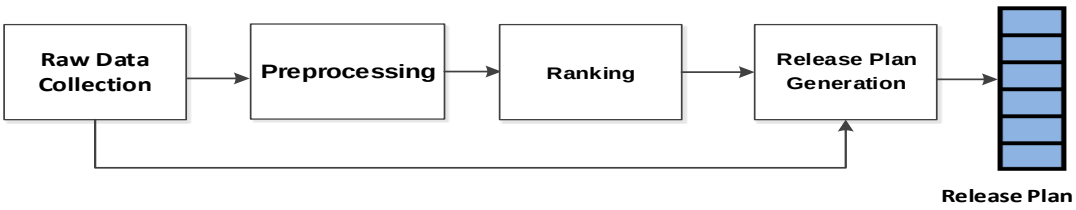


Figure 1.3: The Stages of the Proposed FIS-based Approach

Table 1.2: The Stakeholders Participating in SaaS Next-Release Planning

	Importance	Risk	Available Effort	Required Effort	Dependencies	Tenants Decision Weights	Commonality	Service Level
Tenants	X							
Development Team		X		X	X			
Release Management			X	X		X	X	X

- **Preprocessing**: This intermediate stage performs the required manipulation in order to make the raw data ready for the ranking process. The following sub-processes are performed during this stage:
  - The compliance of contractual constraints is considered when calculating the commonality and the importance of each feature.
  - The weighted importance of each feature is calculated.

  The output of preprocessing stage is two augmented data structures (vectors) that contain the following information:
  - A vector containing the weighted importance of the features.
  - A vector containing the commonality of the features after considering the contractual constraints.
- **Ranking**: As shown in Figure 1.4, the FIS-based approach employs the knowledge obtained from experts (represented by fuzzy rules) in order to rank each feature.

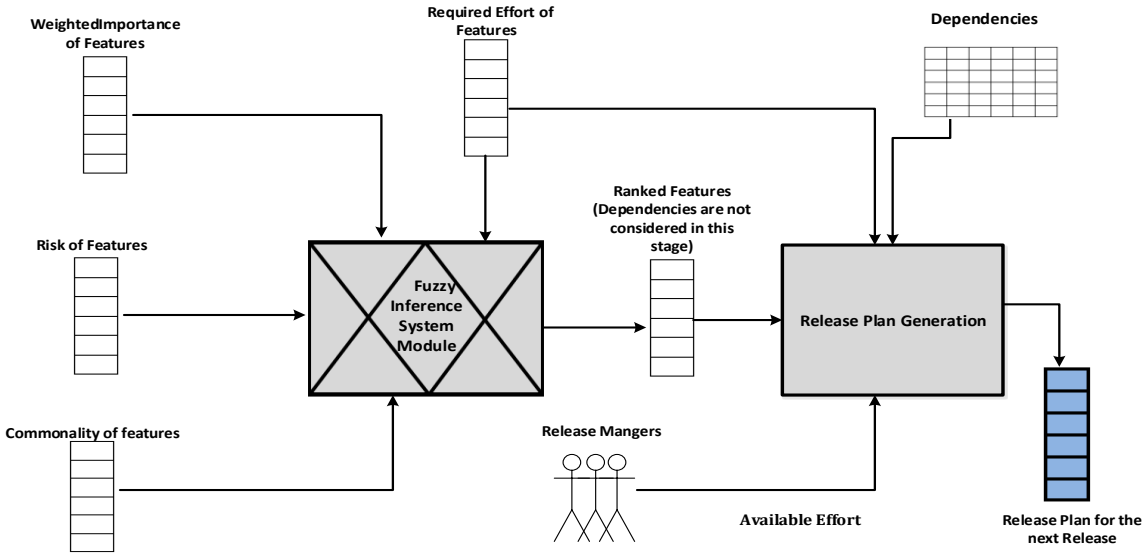


Figure 1.4: Ranking and Generating Release Plan Processes (FIS-based Approach)

- **Release Plan Generation**: In this process two steps are performed:
  - The ranks of the features are tuned in order to satisfy dependencies constrains. For example, if a feature f_i is a precedent of feature f_j, the rank of feature f_i shall be greater than or equal to the rank of feature f_j.
  - The features are sorted according to their ranks. The features that have highest ranks are assigned to the next release plan. Note that the dependencies constraints are fulfilled in the previous step. The effort constraint is taken into account in this step; such that, the total required effort of the selected features is less than or equal to the available effort. Figure 1.4 shows the inputs and the output of release plan generation process.

The second and third proposed approaches are optimization approaches. Two optimization methods are used: BLP and GA. As Figure 1.5 shows, both approaches consist of the following processes:

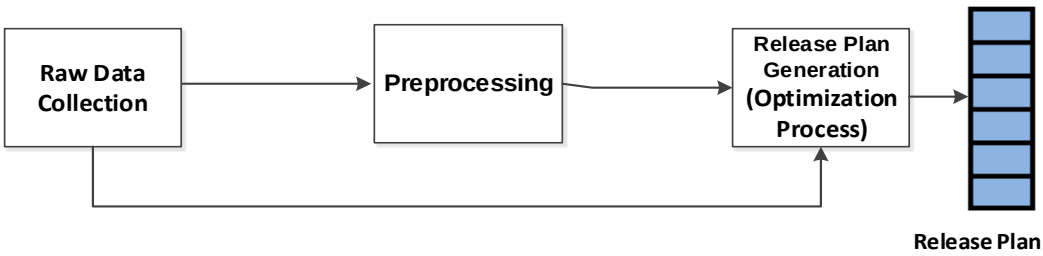


Figure 1.5: The Stages of the Proposed Optimization Approaches (BLP and GA)

- **Raw data collection**: this process is the same as the one in the FIS-based approach.

- **Preprocessing**: this process is the same as the one in the FIS-based approach.
- **Release Plan Generation**: Figure 1.6 shows the inputs to this process. In the BLP-based approach, the optimization capability of BLP is utilized in order to assign the most promising features to the next release plan. The next release plan is represented as a vector of decision variables $X = [x_1 x_2 \ldots\ldots\ldots x_n]$ where $x_i \in \{0,1\}$. If $x_i = 1$ then the feature f_i is assigned to the next release; otherwise, it is postponed to a future release. In the GA-based approach, GA with binary variables is used to optimize release-planning process. BLP and GA approaches used the same objective function and problem variables. GA is used in order to speed up the optimization process. More details about these approaches are presented in Chapter 5.

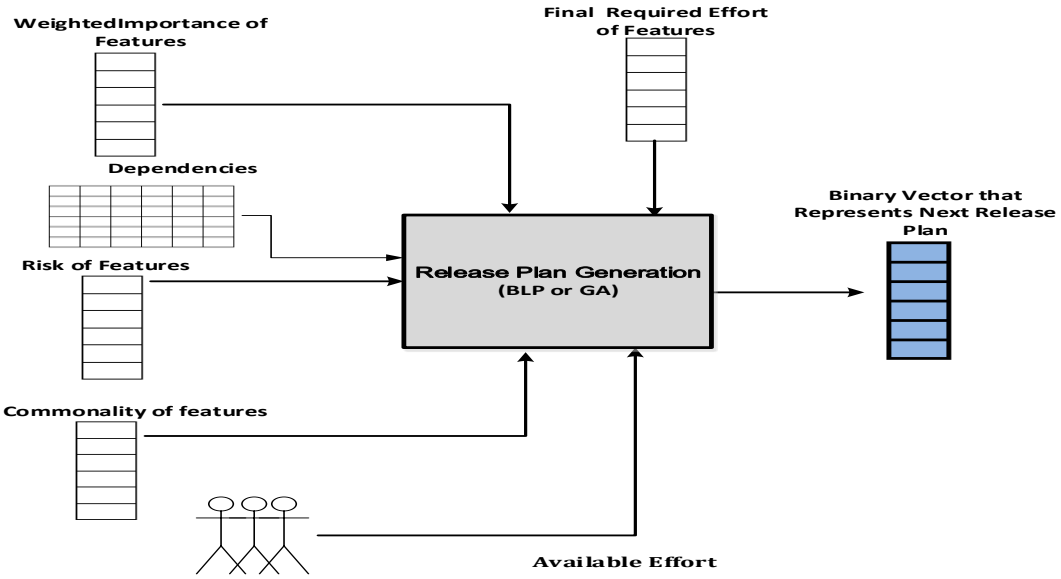


Figure 1.6: Release Plan Generation Processes in Binary Linear Programming

1.4 Contributions

This research provides novel approaches to tackle release planning problems in SaaS applications. In an SaaS application many tenants use thin client to use an application

running on cloud infrastructure. Tenants can subscribe or unsubscribe to SaaS applications easily and instantaneously; therefore, release management shall highly consider the satisfaction of tenants in order to maintain their loyalty, while at the same time fulfilling the needs of other tenants (new or less important ones) in order to increase the popularity of SaaS applications. In addition, the dimension of data integrity and security is very crucial when SaaS providers want to deliver new release of their applications [9, 10]. This risk factor must be considered by the release management because it can significantly affect the success of SaaS applications. Furthermore, SaaS providers can obtain the tenants' comments and criticisms very fast, which can enormously increase the list of the required features in a short period of time. Moreover, SaaS providers usually offer their application in different types of service levels. Tenants can change their service level immediately. Therefore, when planning for the next release, the service level agreements of tenants shall be verified.

The following are the key contributions provided by this thesis:

1. Formulating the problem of the “next release” planning for multi-tenant Software as a Service (SaaS) applications. The main goal of this formulation is being simple and fast enough to address release planning problem with huge number of features and tenants. Two new factors are suggested by this thesis:
   - Commonality of features: In order to maximize tenants’ satisfaction and increase the efficiency of release planning, the features that are requested by highest number of tenants are preferred to be assigned to the next release.
   - Consideration of the service level agreement: In order to meet the SLA, when a tenant requests to add or modify a feature, the SaaS provider should verify if the functional and non-functional aspects of this feature comply with the SLA

of that tenant. If that tenant is not eligible to have this feature, then his vote for this feature is omitted unless he subscribes to the required service level.

2. Providing three novel approaches that generate next release plans for SaaS systems:
   - Fuzzy-inference-system-based (FIS-based) approach, which is fast, simple, intuitive, and depends on fuzzy reasoning in order to deal with uncertainty associated with human judgments. In FIS, the knowledge of the experts is converted to fuzzy rules. Then, the estimates that are provided by stakeholders about the uncertain attributes of the features are manipulated using these rules in order to generate a rank for each feature. This rank of a feature shows its priority among other features. As a part of this approach, two algorithms are proposed in order to satisfy the dependencies constraints among features. These two algorithms are responsible for adjusting ranks of features in order to apply the influence of the dependencies constraints.
   - Binary Linear Programming-based (BLP-based) approach: This approach considers release planning as an optimization problem with binary variables. In BLP-based release planning, we aim to maximize the degree of tenants' satisfaction and degree of commonality, and minimize the degree of potential risk. The dependencies and effort constraints are dealt with in this approach as inequality and equality constraints.
   - Genetic Algorithm-based (GA-based) approach: This approach utilizes genetic algorithm in order to optimize release planning. When the number of features is huge, the optimization of release planning process using GA-based

approach is faster than using BLP-based approach. The same objective function and constraints that are used in BLP-based approach are used in the GA-based approach.

3. Finding out the situations in which each approach is suitable to be used. The proposed approaches are validated, using different scenarios, from the perspective of the degree of the overall tenants' satisfaction, the degree of the commonality of release plans, the degree of the adherence to the risk, and the scalability. Additionally, In order to find out the probability distribution that can fit the data about the importance of features in release planning process, a statistical analysis is conducted on datasets collected from the literature. This study helps researchers in the field of release planning to validate their models using the proper synthetic data.

1.5 Thesis Outline

The thesis is organized as follows: Chapter 2 presents a comprehensive review of the range of issues related to release planning for SaaS systems. It discusses the development lifecycle, and methodologies that are used in SaaS development. It explains the importance and the location of the release planning process during SaaS development. In addition, it presents the nature of release planning processes and explains previous approaches to their development. Chapter 3 states in detail the problem statement and the variables that govern the release planning process in multi-tenant SaaS applications. Chapter 4 introduces the proposed FIS-based approach. Chapter 5 presents the BLP-based and GA-based release planning approaches. In Chapter 6, the results of experiments are discussed. Chapter 7

includes conclusions and recommendations of issues for further study. Figures 1.7 depicts the structure of the thesis.

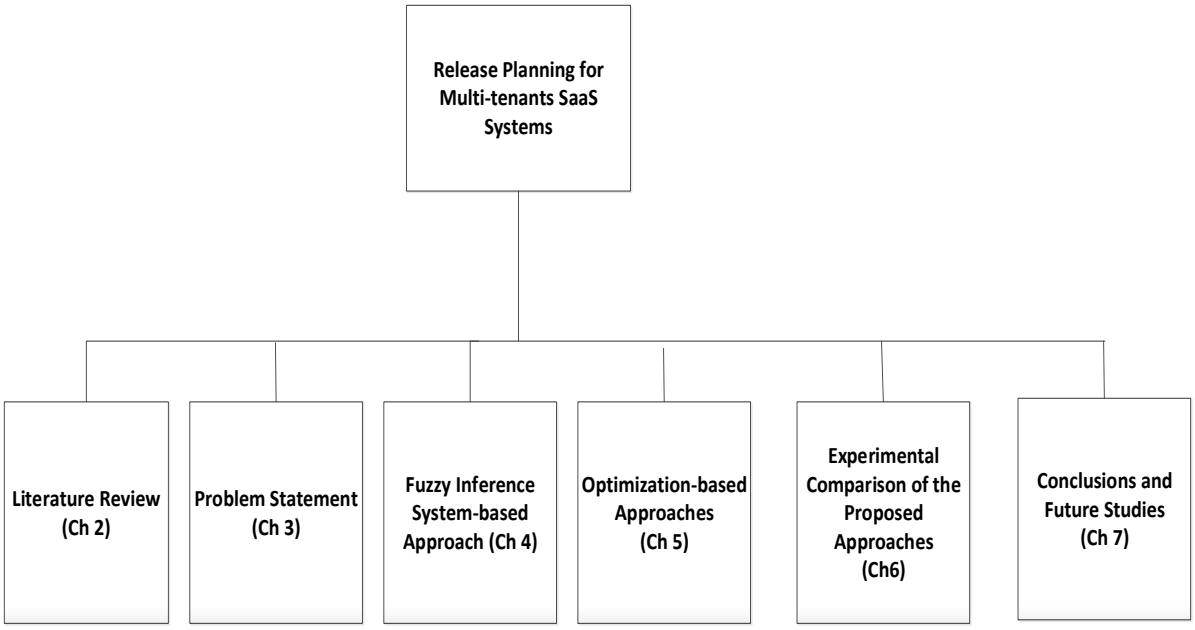


Figure 1.7: Thesis Outline

CHAPTER 2: LITERATURE REVIEW

This chapter presents the research, studies, and tools that are related to this thesis. The chapter includes four sections:

1) The development lifecycle of SaaS applications: This section explores works that discuss the processes that are performed to develop, deliver, and maintain SaaS applications.

2) The nature of the release planning process: This section shows that the release-planning problem is an ill-defined, uncertain decision-making problem.

3) The next release planning problem: This section the works in the literature about next release planning are explored**.**

4) State-of-the-art models and approaches for solving release-planning problem: This section presents the solutions and approaches along with the tools that have been proposed to generate software release plans.

2.1 SaaS Development Lifecycle

As explained in the many papers that explore the development cycle of SaaS applications, most SaaS-application developers rely heavily on Agile and incremental/iterative development methodologies. According to the Agile manifesto [16], Agile methodologies concentrate on having working software over comprehensive documentation, maximize the value of individuals, and have less planning and high responses to the software change. The following processes can be applied iteratively in order to deliver a SaaS product in a set of continuous releases [17, 18]: 1) Envisioning, during which the scope and goal of the application are determined, 2) Planning, during which the schedule, budget, and quality-assurance and control procedures are determined, 3)

selecting delivery platform (SDP), where the reliability, availability, scalability, and performance of different cloud SDPs are evaluated, after which a specific SDP is selected to host the SaaS application, 4) The development of the application, which involves all development activities (analysis, design, implementation, testing and deployment), and 5) Operation, where the SaaS is operational and customers can subscribe according to a certain service level agreement and use the service. In [18], the authors discuss the design criteria of SaaS applications. They divide these criteria into two groups: the special characteristics of SaaS applications (such as supporting commonality, internet-based operation, etc.), and the properties of SaaS applications (such as availability, scalability, security, and reusability). Furthermore, in that research the traditional development process is used to identify the activities that are required during the creation of SaaS applications. In addition, the authors discuss the importance of maximizing reusability via commonality/variability analysis. They define commonality "as the number of potential applications which need a specified feature such as a component or a service" [18]. They state that it is more advantageous if the features with high commonality are included in the target SaaS application. In [19], the authors state that the uniqueness of SaaS applications requires that new processes be added to traditional development processes. These new processes include establishing pricing policies, SDP evaluation, and close consideration of customization and configuration. In [20], the techniques of software product line (SPL) are exploited in order to model the variability in SaaS applications. The variability in SaaS can be customer-driven, which matches the external variability model of SPL, or it can be realization-driven, which is similar in its principle to the internal variability of SPL.

Scrum [21] is used in the development of many SaaS applications. Scrum is an Agile methodology that develops software in a set of Sprints (iterations). In each Sprint, new functionalities are added to the software product. The first step in Scrum is the planning phase which carries out the definition of a new release based on currently known backlog. In [13], continuous Scrum (which is an extension of Scrum development [21]) is used to build a SaaS product. As in the regular Scrum development, the product is developed in series of Sprints. Each Sprint lasts three weeks. Three types of activities are performed during a Sprint: fixing bugs, minor enhancements, and key enhancements. As many software products may be developed at the same time, the Sprints of one product may overlap with the Sprints of other products. The Sprint is divided into three stages: planning, development, and quality assurance. Accordingly, the development team is divided into three sub-teams. Each sub-team is specialized to carry out a specific phase of the Sprint. At a certain point, the three sub-teams may work in three different overlapped Sprints. For example, while the first sub-team is working on the planning of Sprint k, the second sub-team is working on the development of Sprint k-1 and the third sub-team is working on quality assurance of Sprint k-2. The authors in [22] relate their expertise of developing a SaaS application using Scrum in a small software industry. Because of unrealistic estimates, the team could not deliver the first release in the specified time. For the later releases, due to the experience gained from the first release, the estimates were more accurate.

Extreme Programming (XP) [23] is an Agile methodology that considers the following principle in the software development: rapid iterations, rapid feedback, rigorously tested code, team courage, high communication with customers, and simple design. In [24], new concepts and techniques are added to the current XP practices, in order to address the

challenges associated with SaaS applications. For example, an information page is used by the project parties (development team and customers) to track the current status of the project. Customers can add or change requirements using this online page. The concurrent version system is also used in [24]. This system is a tool that allows the members of the development team to share their knowledge and expertise by adding new solutions, suggestions, or recommendation. A tracker tool is also used, which allows the project parties to track the history of changes on the diagrams and algorithms.

In [25], Agile manufacturing along with Toyota's lean manufacturing system are used to improve the quality of seamless delivery in SaaS applications. For example, the Poka Yoke "mistake-proofing" concept is used. This concept states that the possibility of mistakes is decreased by automating the number of reproducible repeated tasks, which reduces the effort needed to track the dependencies between different tasks and activities in the development environment. An additional level of quality is provided to the application by using the Jidoka "stop the line" practice. Jidoka is concerned with fixing the error when it happens, and is automated through the use of human heuristics. The third concept is Kaizen, which is a principle of continuous improvement during the entire life of the application. Applying these principles can increase the quality of the delivered SaaS application.

In [26], the authors discuss how to use a service delivery platform (SDP) to build an SaaS application. In [6], the authors address the evolving nature of the SaaS applications; more specifically, how SaaS providers can handle the issue of tenant-driven evolution, where the SaaS providers change the SaaS according to the needs of tenants. SaaS providers try to maximize the commonality of requirements of different tenants in order to minimize

the cost of upgrades to applications. However, there are still specific tenants' needs that should be fulfilled. A fixed set of customization options to tenants sometimes is not enough to address the tenant-driven evolution; therefore, the authors present some techniques that can be used to change the SaaS according to the needs of tenants.

In [27], the authors state that in order to satisfy tenants' needs, SaaS applications shall provide a set of variant points that can be modified according to each tenant's needs. For example, tenants shall have the ability to configure some fields in a user interface. In addition, the concept of a "variability describer" is introduced. A variability describer sets out the locations, constraints, and dependencies of the different variation points. The concept of a variability describer is incorporated to the Web Services Business Process Execution Language (WS-BPEL) [28] process model. WS-BPEL is a language for specifying business process behavior based on Web Services. In [29], the five levels of customization of the SaaS applications to the tenants are discussed. These levels include: GUI, workflow, service, data, and QoS, which is represented by the SLA. In addition, [29] sets out the methods that are used for customizations; these include: source code, configuration, and workflow composition. After that, a tenant-based, semi-automated customization approach is proposed.

2.1.1 Discussion

In this section, we concentrate on two main points: the appropriateness of Agile methodologies to the development of SaaS applications, and the location of the release-planning process in the development lifecycle of a SaaS application.

2.1.1.1 Agile Development of SaaS Applications

From above illustration we can infer that the following points must be considered when SaaS applications are developed:

- Popularity is a significant indicator of the success of a SaaS application.
- To maintain the popularity of the application and increase the profit, different tenants' needs must be satisfied while continuing to place a priority on quality.
- To fulfill the evolving and new requirements of the tenants, SaaS providers must frequently deliver new releases of the software.

To build a successful SaaS application, it is essential to use the appropriate development methodology. Agile is a paradigm that has been increasingly used in recent years. One major aspect of Agile development that should be considered is that Agile delivers the application in a set of short releases. Each new release is a working version of the software with additional features. At the beginning of development of each release, release planning must be conducted in order to select the features that will be included in that release. We can see that there is compatibility between the nature of SaaS applications (which should be developed envisioning short lifecycles and stressing high quality) and Agile methodologies (which concentrate on delivering high quality products in a set of short releases). Also we can see from above literature that the release planning process is a key aspect of the development of a SaaS application.

2.1.1.2 SaaS Development Lifecycle

According to [17, 18, 19], the development lifecycle of SaaS applications consists of the following processes (see Figure 2.2):

1. **Envisioning:** The senior management of the SaaS provider studies new markets, new business opportunities, and the feasibility of offering the service.

2. **Analysis:** The following tasks are performed:

   a. **Domain exploration:** The service domain is explored and analyzed in order to specify the initial set of features that will attract the targeted clients;

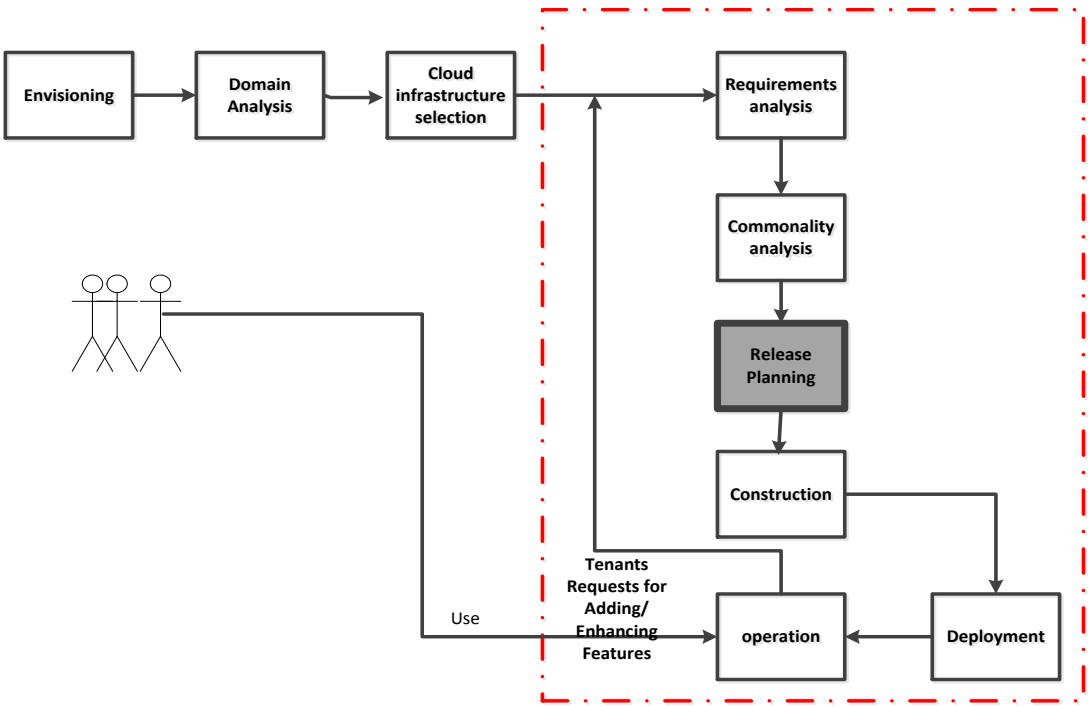


Figure 2.1: The Development Lifecycle of SaaS Applications

   b. **Cloud provider selection:** The SaaS provider chooses a cloud provider from which to lease infrastructure and platform resources.

   c. **Commonality and variability analysis:** The commonality analysis shows the common features that tenants share. These features are usually encountered in the common codebase of the SaaS application.

   d. **Release planning:** Depending on resources and technical constraints, the most promising set of features is selected to be implemented in the next release.

3. **Construction:** This phase includes the design, the implementation, and the testing of the service.
4. **Deployment:** The SaaS application is deployed onto the cloud infrastructure.
5. **Operations:** According to a pricing policy, the tenants start subscribing to the service and have the authorization to use it.
6. **Requirements Elicitation:** during the use of the service, the tenants send their requests for additions or modifications of features. They also submit their evaluations about the service. The feedback can be provided using linguistic terms which allows the tenants to qualitatively and naturally express their opinions about the provided service, and about their future needs. In addition, the requirements of market (such as new promising features) are gathered.

We can call the steps 1, 2a, 2b the start-up stages. The steps from 2c to 6 are applied periodically when developing a new release of the SaaS application. It is clear that release planning process is a key process in SaaS development, and the effectiveness of this process increases the possibility of maintaining or increasing tenants' satisfaction.

2.2 The Nature of the Release Planning Process

Release planning is a decision-making problem that has high degree of uncertainty. The release management team is required to make decisions about assigning sub-sets of features to a sequence of releases using uncertain, human-based information. In [30], it is stated that when planning a software release in an Agile environment, management has difficulties in making decisions because of the high degree of uncertainty associated with the business value and the size of each user story, and the available resources. Carlshamre [31] and Ruhe et.al [32] consider release planning as a "wicked planning problem."

“Wicked problems” are defined as those that are difficult or impossible to completely formalize because of their incomplete, ambiguous, and contradictory attributes. Furthermore, there are no crisp ("true" or "false") solutions for release planning; instead, the solutions of release planning can be categorized as "good" or "bad" solutions. Moreover, the constraints that govern release planning can be “hard constraints,” such as budget and technology, or “soft constraints” such as risk and resource consumption [32]. Ruhe and Saliu [33] describe release planning as an ill-defined problem; hence, it is necessary for the suggested solutions to combine mathematical models with human knowledge and expertise. Al-Emran et al. [34] state that the uncertainty in operational release planning can arise from many factors, including the arriving features while the release is developed, the effort required to deliver the features, and the availability and productivity of human resources (developers).

In XP, a user story (or a story) is informal way that users describe their requirements to the development team. To estimate the effort required to implement a story, story points are used. A story point [35] is a metric that measures the complexity and difficulty needed to deliver a feature. For example, a feature that requires 4 story points is double in difficulty and complexity the feature that requires 2 story points. Logue et al. in [36] state that release planning in XP is an uncertain problem. The authors enumerate and discuss some factors that cause this uncertainty. These factors include: the velocity of the development team (how many story points can be completed during an iteration), story size (how many story points or ideal working days are needed to implement and deliver a story), and the business value of the stories that are included in the release.

Ruhe [37] state that release planning can be considered as a multi-criteria/ multi-person decision-making problem. Different stakeholders who have different requirements and different (and sometimes contradictory) objectives need to participate in the planning endeavor. In addition, the planning effort must take into account many decision criteria, such as value, quality, cost, and time. In [8], Greer et al. state that release planning can be seen as an optimization problem, where the release management intends to construct release plans for future releases in a way that: 1) minimizes the penalties that will arise from not meeting the dependencies constraints (coupling and precedence) among requirements, and not satisfying the requirement priorities of the different stakeholders, 2) maximizes the benefits of satisfying the dependencies and priorities factors, and 3) stratifies the resources constraints. Similarly, Akker et al. [38] deal with release planning as an optimization problem, where the planning process aims to maximize the projected revenue using available resource.

2.2.1 Discussion

The release-planning problem can be seen from different views:

- It is a multi-person/multi-criteria decision-making problem. Many decision makers participate in the planning effort, including management, developers, and customer representatives. In addition, many factors (criteria) need to be taken into account when planning a release, such as the business value of the planned release, the risk, the available and required resources, the decision weight of the stakeholders, and the technical and managerial constraints.
- It is a problem of "decision making under uncertainty." The uncertainty in release planning happens because of many factors, including incomplete and ambiguous

information, human factors (expertise and knowledge of the developers and customers), and dynamics of the market.

- It is a multi-objective optimization problem. The release management tries to plan the future releases in a way that maximizes certain objectives, such as stakeholders' satisfaction, release value, and quality, and minimizes other objectives such as risk and cost, while satisfying certain managerial and technical constraints.
- It is a prioritization problem. The release management can prioritize the requirements according to certain criteria (stakeholders' satisfaction, release value), and then assign the requirements with the highest priorities to the early releases.

2.3 The Next Release Planning Problem

The next release planning was introduced in [108]. In this type of release planning, release management plan just for the next of the software. In [109], multi-objective next release planning problem is discussed. The authors consider that problem as search-based problem. They consider two conflicted objectives: maximize customer satisfaction and minimize required cost. After that, the results of an empirical study about the suitability of weighted and Pareto optimal genetic algorithms, together with the Non-dominated Sorting genetic algorithms (NSGA-II) algorithm are presented. In [110], three state of the art multi-objective metaheuristics (two genetic algorithms, NSGAII and MOCell, and one evolutionary strategy, PAES) are applied to solve the next release planning problem. Two objectives are considered: maximizing customers' satisfaction and minimizing the cost. The

result of experiments shows that NSGA-II has generated the highest number of optimal, MOCell has produced the widest range of different solutions, and PAES is the fastest.

2.3.1 Discussion

In this research, we select to plan for the next release due to the nature of SaaS applications. In SaaS applications, the tenants' needs are evolved quickly because the number of tenants may change in a very short period of time because of the simplicity in subscribing to or unsubscribing of the applications; consequently, the features in the product backlog can also change in a very short period of time. This may change the priorities of features during the lifecycle of one release. Hence, we find that it is more efficient if the planning is performed only for the next release.

2.4 The Approaches to Solve the Release-Planning Problem

In this section, we illustrate the different models, approaches, and tools different researchers have proposed for solving release-planning problems. We will discuss five main techniques from the literature:

- Integer linear programming
- Combination of linear programming and genetic algorithm
- Analytical Hierarchy approach (AHP)
- Constraint programming
- Fuzzy logic

2.4.1 Integer Linear Programming-based (ILP) Solutions

In this approach, the release planning is considered as an optimization problem [32, 36]. It is required to assign a set of features to a sequence of releases in a way that

maximizes some objectives such as value, priority, stakeholders' satisfaction and profit, and minimizes other objectives such as risk and cost. In its general form, release planning can be formalized as follows:

Given a set of features $f_1, \ldots . f_n$, it is required to assign these features to a sequence of releases $r_k, r_{k+1} \ldots . , r_m$ in a way that maximizes certain objectives $O_1, O_2 \ldots . O_q$ while taking into account certain constraints $C_1, C_2 \ldots , C_p$. A decision vector X is defined as $X = [x_1, \ldots . , x_n]$, x_i is an integer, $k \leq x_i \leq m$, and k is the number of the next release (release $k-1$ is already delivered). If $x_i = y$ then the feature f_i is assigned to the release y. For example, if we are planning only for the next release and our objective is to maximize the overall stakeholders' satisfaction while taking into account resource constraints, then the problem can be formalized as: $max\ O(x)$ such that

$$O(x) = \sum_{i=1}^{n} (Sat(f_i) \times x_i) \qquad (2.1)$$

subject to

$$\sum_{i=1}^{n} (Res(f_i) \times x_i) \leq Ava\_Res(k)$$

$x_i \in \{0,1\}$ and $Sat(f_i)$ denotes the overall stakeholders' satisfaction of feature f_i, $Res(f_i)$ denotes the required resources to deliver feature i, and $Ava\_Res(k)$ denotes the available resources to deliver the release K (in this case, the next release). Much research has approached release planning as a linear programming problem. Ruhe et al. [33] introduce two approaches for planning software releases. The first approach (which they call "the art of release planning") depends on human skills and capabilities to tackle the

problems and resolve conflicts. The second approach (which they call “the science of release planning”) employs integer linear programming to generate optimal solutions. Akker et al. [38] present an integer linear programming-based model (ILP-based) with its corresponding tool to help release management to conduct release planning. The inputs to their model are: a set of requirements, the estimates of their revenue along with the required resources, and the managerial steering mechanism that depends on what-if analysis. Freitas et al. [39] compare some metaheuristics approaches (genetic algorithm and Simulated Annealing) with the exact optimization approach (such as simplex method) when generating release plans. The comparison shows that exact optimization has achieved better results (higher value of the objective function); however, exact optimization takes more time to find the solutions. Ullah et al. [40] study release planning in software product line (SPL) development. They state that because of the special characteristics of SPL, new variables must be involved in the release planning process (for example, resolving the conflicts between the requirements of core assets and the requirements of various products). They use ILP to formalize and optimize the solution of release planning. Li et al. [41] exploit ILP to generate optimal solutions for release-planning problems. The objective is to select the requirements that maximize the projected revenue while considering the constraints of available resources and the allocated time. The authors propose two models to achieve their goals. The first model carries out the scheduling of the development processes. This model concentrates on minimizing the project duration. The second model combines requirement selection and the development teams scheduling in order to find the optimal (maximum) revenues value. Akker et al. [38] employ ILP to help requirements engineers to plan for the next release. The input to their model includes the requirements, estimated revenue per a

unit of requirement, and available resources. Some flexibility is added to the planning effort in order to deal with related factors such as team composition, team members' transfers from a project to another project, extension of deadlines and hiring external resources.

2.4.2 Combination of Linear Programming and Genetic Algorithm

In this approach, release planning is formalized as an integer linear programming problem; however, because it is difficult and too expensive to explore all feasible solutions using traditional linear programming techniques [42], genetic algorithm (GA) is used to find the optimal or near-optimal solutions. GA [43] emulates the evolution phenomenon in biological life. The feasible solutions are represented as chromosomes. Solutions are evaluated by calculating the value of their fitness functions. If the desired solutions are not found, another generation (iteration) is created by applying reproduction, crossover and mutation operations. The new generation is evaluated again, and this loop keeps going until the desired solution is reached. Greer et al. [8] utilize a combination of ILP and GA to solve release planning problems. They consider three factors: technical precedence, conflicts of stakeholders' priorities, and available resources. They propose an approach for solving release planning problem that they call EVOLVE, which depends on ILP in formalizing the problem and uses the strength and practicality of genetic algorithms to generate the solutions. Ruhe et al. [37] have added more capabilities to the EVOLVE method by proposing EVOLVE*, which consists of three main phases: 1) modeling, where the problem is formalized as an ILP problem 2) exploration, where a genetic algorithm is used to produce a set of potential solutions and 3) consolidation, where the solutions produced in Stage 2 are evaluated by the release management. Depending on their expertise, the management members select (and may adjust) the most promising solution manually.

EVOLVE* produces two release plans for two future releases in order to be elastic for any market changes. In addition, EVOLVE* tries to maximize stakeholders' satisfaction. Ngo-The et al. [32] propose EVOLVE+, which is an approach that takes into count both hard and soft constraints and objectives in the planning process. Hard constraints are those that can be evaluated accurately (such as budget), while soft constraints are vague and difficult to measure using a crisp evaluation (such as risk). EVOLVE+ utilizes ELECTRE (which is a multi-criteria decision-aid method) to generate several potential solutions. The final decision regarding a solution is made by release management. Ngo-The et al. [44] discuss the resource-allocation problem in operational release planning. They study how to allocate available human resources to the tasks that are required to deliver a set of features that have already been assigned to the release under consideration. Their allocation process also takes into account the different degrees of productivity of the members of the development team. The authors present an approach which they call it OPTIMIZERASORP in order to tackle this issue. OPTIMIZERASORP utilizes the strength of ILP and GA to plan the resource allocation.

2.4.3 Analytical Hierarchy Approach (AHP)

The Analytical Hierarchy Approach (AHP) [45] is a decision-making technique that is used in multi-criteria decision-making processes. To apply AHP, four steps are performed: 1) determine the objectives, the decision criteria, and the possible alternatives, 2) calculate the values of the relative importance of the decision criteria. These values are captured in a vector, 3) they determine preferences regarding each alternative criterion over others in order to calculate the value of the relative ranks of each alternative. These values are stored in a matrix, and 4) calculate the weight of each alternative by multiplying the

vector from stage 2 by the matrix from stage 3. Karlsson et al. [46] present an AHP-based approach for prioritizing software requirements. Cost and value are the prioritization criteria. AHP's pair-wise comparison is used to measure the relative value and the relative cost of each requirement. Then, the requirements are plotted on the cost-value diagram. The cost-value diagram is divided into three regions: high-priority requirements (low cost and high value), medium-priority requirements (medium cost and medium value), and low-priority requirements (high cost and low value). In [47], fuzzy AHP is used in order to deal with the uncertainty associated with stakeholders' concerns in the process of requirements prioritization. Requirements prioritization is considered as multi-person decision making problem where many stakeholders participate in the prioritization process. Instead of using crisp numbers, triangular fuzzy numbers are used in the comparison matrix provided by each stakeholder.

2.4.4 Constraint Programming (CP)

A constraint satisfaction problem (CSP) " is a problem that is composed of a finite set of variables, each of which is associated with a finite domain, and a set of constraints that restricts the values that the variables can simultaneously take" [48]. Regnell et al. [49] consider release planning as a CSP. Priorities and dependencies between features are formulated as relations among release-planning variables (feature priorities, stakeholder preferences, and resource availability). They use these relations and variables to solve release planning using CP.

2.4.5 Fuzzy-theory-based Release Planning

Fuzzy set theory [50] has been used to interpret and represent uncertainty [51, 52]. In the context of decision making, when the objectives and/or the constraints are fuzzy and

uncertain, we can say that the decision is made in a fuzzy environment [53]. Because release planning is considered as an "under-uncertainty decision making" problem, some research works have used fuzzy theory-based approaches to model it. Shen [54] extends EVOLVE* to FUZZY-EVOLVE*. The available and the required resources are represented as fuzzy numbers. In addition, the objective function is considered as a fuzzy membership function. Fuzzy aggregation is applied to the fuzzy objective function and fuzzy resource constraints in order to find the release plan that achieves the optimal degree of satisfaction. Although FUZZY-EVOLVE* considers the fuzziness of the available and the required resources, it does not consider human expertise in the planning process. In other words, fuzzy logic is just used to reflect the fuzziness in the input data and not in the process itself. In this thesis, the fuzzy logic is used to represent the human knowledge that is incorporated implicitly in the planning process. The fuzziness is involved internally in the planning process. Ngo-The et al. [55] define two dependency among requirements: 1) coupling, in which two or more requirements should be developed in the same release; and 2) precedence relationship, in which a requirement should be developed (and sometimes delivered) prior to the other requirement(s). In early phases of the software project, it is difficult to define these relations precisely. Therefore, representing these dependencies as fuzzy relationships helps to capture the uncertainty associated with their definitions. Additionally, fuzzy membership functions are used to calculate the degree to which dependences constraints are satisfied. Ngo-The et.al. [56] state that there is an uncertainty in estimates of available effort and required effort. Additionally, uncertainty can be present in defining the objectives that are related to cost, benefit, and quality; hence, they use fuzzy logic.

2.4.6 Discussion

In this section we discuss the proposed models from two different angles: the variables that control release planning, and incorporating human knowledge.

- **The variables that control release planning**

When undertaking release planning, many decisions must be made on the basis of uncertain, incomplete, volatile, and/or conflicting information. These decisions must consider varied and even contradictory goals (such as business value, profit, stakeholders' satisfaction, quality, and delivery time). In addition, they have to take into account many hard and soft constraints. Hard constraints include technical constraints, budget and cost, and resources. Soft constraints include factors involving human influence, risk, resource consumption, and quality. In addition to these factors, contractual constraints must be considered. Contractual constraints , which are hard constraints, are those related to the contract between the development organization and the customer about the level of service and support that will be available after the software is delivered.

- **Incorporating human knowledge**

In all of the previously published works about release planning, human knowledge and expertise have been involved only in the final step – when the choice is made between alternatives that have been generated by the computational models. When there are huge number of features and stakeholders, it is difficult to compare between the generated release plans. Thus, it is more practical in such cases to automatically incorporate human expertise in the computational model. The difference between the FIS approach presented in this thesis and the fuzzy approaches in [54,55,56] is that, in this thesis, the fuzzy logic is used as a means to represent experts' knowledge in the planning process. The planning is performed

according to a set of rules that reflect the perspective of the experts. In other words, practically, experts' knowledge is considered as the function that is used in the planning process. In [54,55,56] the fuzzy logic is used to reflect the fuzziness associated with the inputs, and there are no fuzzy reasoning involved. The other difference is that this thesis uses fuzzy reasoning to prioritize the features according to certain inputs, while in the previous works, an optimization approaches are used with fuzzy inputs.

2.5 Some Tools for Constructing Release Plans

In this section, we introduce some tools that are used for planning software releases. Release Planner (RP) [57] helps software organizations to carry out the release planning process. RP is implemented as an optimization approach, and it covers a wide range of activities related to software planning; for example, it allows the users to maintain a features repository, to apply a proactive what-if-analysis, and to generate reports. In [49], MiniZinc [58] is utilized to solve release planning as a CSP. MiniZinc is a special case of Zinc language. Zinc allows defining variables, domains, and use predicates to represent the constraints on the variables. ScrumDo [59] is a planning tool for Scrum development. It allows the management to create repositories of stories (features), and assign them to certain iterations. The data about stories includes: estimated effort, the priority, the developers' information, and the tasks that are required to implement the story. AgileTrack [60] is a tool for managing XP projects. It allows for the creation of stories and their associated tasks, the planning of iterations, the planning of a release depending on iterations' plans, and the tracking of the development process. VersionOne [61] is a tool for managing Agile projects. Similar to other tools, it allows the user to create new releases with their attributes such as duration, velocity, and the required and available resources.

2.6 Fuzzy Theory in Software Engineering Decision-Making Problems

In this section, we review the research that has used fuzzy theory to tackle decision-making problems in software engineering. Lee [62] uses fuzzy theory to build a multi-person decision-making model for evaluating risk during software development activities. Bajaj et al. [63] employ fuzzy theory to estimate the effort needed to implement a software component. Wang and Li [64] develop a multi-group decision-making model that employs fuzzy sets to make decisions related to the selection of software-configuration items. Chen [65] proposes an algorithm that utilizes fuzzy theory to evaluate the rate of aggregative risk during software development. This algorithm does not require fuzzy assessment matrices for attributes. Also, it avoids the complexity of a defuzzification process that depends on the centroid method. Kwong and Bai [66] use fuzzy AHP to compute the weight of the importance of customers' requirements in the quality function deployment (QFD) method. Praynlin and Latha [67] use an Adaptive Neuro Fuzzy Inference System (ANFIS) in the analysis phase to estimate the effort required in the software development process. Amindoust and Saghafinia [68] use FIS to handle the uncertainty and the subjective judgment of the decision makers when they want to select suppliers in manufacturing and service industries. Kutlu et al. [69] use FIS to evaluate the jobs in an organization in order to help the management to build an appropriate pay structure based on the value of the work to the organization. Palomares et al. [70] use ordered a weighted averaging (OWA) operator (a fuzzy aggregation operator) to build a consensus system for large multi-person decision-making problems.

2.5.1 Discussion

In software engineering, "uncertainty is inherent and inevitable" [71]. The uncertainty can arise in the problem domain or the solution domain, or as a results of human

perception [71]. In this environment of uncertainty, any decision must be undertaken with unknown probabilities of the outcome of the available alternatives. This occurs in many software projects because of the limited information (historical data) or the uniqueness of the software under development. In those cases, human reasoning and approximation are used. Human reasoning is expressed using linguistic terms that reflect the perception of the evaluator/decision makers. Perception cannot be represented or measured using crisp values. Any values provided for the attributes of the decision-making problem are an approximation of reality [72]. It is important to map these approximate values to their corresponding linguistic values. This mapping can be performed using membership functions and fuzzy rules aggregation, which are the components of FIS. FIS considers decision attributes as qualitative rather than quantitative, which makes it an appropriate method for handling uncertainty [73]. One of the contributions of this thesis to the literature is that we use FIS to synthesize human knowledge and heuristics, and to measure the degree of membership of each estimated value to certain fuzzy sets. These fuzzy sets represent the terms used by humans in the estimation.

2.7 Summary of the Chapter

SaaS applications are developed using Agile methodologies due to the compatibility between the nature of SaaS applications and Agile methodologies. Release planning is a core process in Agile development; hence, it is a significant process in SaaS development. Release planning can be considered as a multi-criteria, multi-person, uncertain decision-making problem. It can be solved using optimization or prioritization techniques. Many variables that control release planning have high degree of uncertainty. This uncertainty can be dealt with using fuzzy theory.

CHAPTER 3: PROBLEM STATEMENT

This chapter introduces the variables and the problem statement that govern release planning for multi-tenant SaaS applications. As Figure 3.1 shows, the variables of release planning in SaaS are divided into two groups:

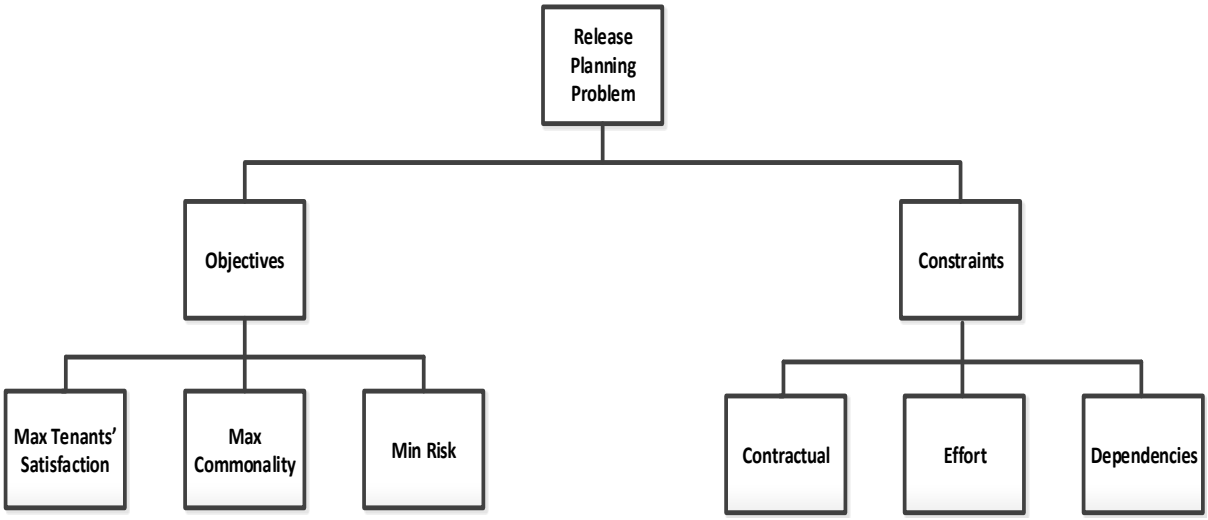


Figure 3.1: Release Planning Variables

- The objectives that the release planning process will maximize or minimize. The current research considers three objectives: maximizing tenants' satisfaction, maximizing tenants' commonality, and minimizing the risk of the release under consideration.
- The release planning constraints which include
  - Contractual constraints, which are related to the level and quality of services that a SaaS provider guarantees to his tenants. These are usually documented in the service level agreement document (SLA).

- Dependencies constraints, which are related to the technical dependencies between features. The current research takes into account two types of technical constraints: coupling, precedence.
- Effort constraints, which are the constraints related to the available and required effort to deliver the next release.

As Figure 1.2 shows, at the beginning of the planning of each release, different tenants ask for adding, fixing, or enhancing features. A feature is "a set of logically related requirements that provide a capability to the user and enable the satisfaction of business objectives" [74]. A feature can be seen as a bridge between the problem domain and the solution domain. From the perspective of tenants, the importance of features can be classified as either mandatory or optional. “Mandatory features” are those which tenants need to have, and they are related to the core business logic or to the quality of service (QoS) attributes of the product. For example, in human resource (HR) systems, the ability to add information about an employee is a mandatory feature. A security system is a mandatory feature in SaaS applications. “Optional features” are those that tenants want to have in their applications, but if they are not included, there will be no negative effects on the functionalities or the QoS of the product. For example, in HR systems the capacity to generate reports about the productivity of employees might be considered an optional feature. Usually, conflicts occur among the stakeholders about the importance of features. These conflicts are resolved by the release planning process, where release management should consider all the perspectives of the different stakeholders. In release planning process, the release managers are required to select the features that will be included in the next release. If management has adequate resources, it will include all the requested features

in the next release; however, most often, the resources available to management are limited, which means it must select just a subset of the required features.

Let $T = \{t_1, t_2, \ldots . t_m\}$ be the set that represents the tenants who subscribe to the SaaS application. While using the SaaS application, the tenants send requests to management to add or modify some features. Let $F = \{F_{ti} | \, i = 1,2, \ldots , m\}$ be a family of sets of features that are requested by the tenants, where F_{ti} represents the features which are requested by a tenant t_i. For example, for a tenant t_i, F_{ti} can be denoted as F_{ti}=$\{ f_i^1 , f_i^2, \ldots \ldots , f_i^{li}\}$ such that li is the total number of features that are requested by t_i. Let $F^* = \bigcup_{i=1}^{m} F_{ti} = \{f_1, f_2, \ldots f_n\}$ be the unified set that contains all features, where n is the total number of features, such that $n \leq \sum_{i=1}^{m} |F_{ti}|$ and $|F_{ti}|$ is the cardinality of the set F_{ti}. The release management is required to plan the next release by selecting the features that achieve the highest possible degree of tenant satisfaction, and the lowest possible degree of risk. In the planning endeavor, release management must consider resource limitations as well as technical and contractual constraints. Let $F^{\wedge}$ be a set that represents the release plan, such that $F^{\wedge}$= $\{ f_1^{\wedge}, f_2^{\wedge}, \ldots . f_w^{\wedge} \}$, where $F^{\wedge} \subseteq F^*$, and $w \leq n$. We define $G_{F^{\wedge}}: F^* \rightarrow \{0,1\}$ as the characteristic function of the set $F^{\wedge}$such that

$$G_{F^{\wedge}}(f_i) = \begin{cases} 1 & for \; f_i \in F^{\wedge} \\ 0 & for \; f_i \notin F^{\wedge} \end{cases} \qquad i = 1 \ldots . . n \qquad (3.1)$$

Because release planning is a continuous process in SaaS applications [13], it is more efficient for the release management to plan only the next release. This will cope with the nature of SaaS applications, which are extremely volatile due to the dynamics of the market. For example, because it is easy to subscribe or unsubscribe to the service, many tenants may

join or leave the service in a short period of time, which may significantly change the features list at the beginning of each release.

3.1 Planning Objectives

SaaS providers aim to maximize tenants' satisfaction and commonality (selecting the most common features), and to minimize risk.

3.1.1 Maximizing Tenants' Satisfaction

The success of a SaaS application can be measured by its popularity. Popularity can be achieved by incessantly meeting the expectations of tenants [6, 75]. Reaching a minimum level of tenant satisfaction requires managers to include an acceptable number of the most important features requested by that tenant in the next release. The maximum degree of tenant satisfaction is attained by including all of the features that the tenant has requested in the next release. The definition of importance varies from a context to other. "Importance could, for example, be combination of urgency of implementation, importance of a requirement for the product architecture, strategic importance for the company, etc." [76]. In [8, 14,76] , the importance of a feature is determined on the basis of two criteria:

- Business value, which is the expected value that the feature will add to the business of the tenant. Let $Value(f_i, t_j)$ be the function that captures the value of feature f_i from the perspective of tenant t_j [8].

- Urgency, where the importance of the feature is associated with the time frame within which the feature is released. In some cases, a tenant asks for a new feature as soon as possible in order to meet some unexpected changes in the market or fix

unexpected bug in the system. Let $Priority(f_i, t_j)$ be the function that captures the priority of feature f_i from the perspective of tenant t_j [8].

In this thesis, we assume that importance is a combination of value and priority. Let $Imp(f_i, t_j)$ be a function that returns the importance of features f_i from the perspective of a tenant t_i, such that $1 \leq Imp(f_i, t_j) \leq 9,\ 1 \leq i \leq n,\ 1 \leq j \leq m,\ Imp(f_i, t_j), i, j \in Z^+$ (positive integers),1 and 9 are the lowest and the highest possible degrees of importance respectively. Depending on the definition of the value and the priority of a feature, we define $Imp(f_i, t_j)$ as follows:

$$Imp(f_i, t_j) = \alpha\, Value(f_i, t_j) + (1 - \alpha)\, Priority(f_i, t_j) \qquad (3.2)$$

where $Value(f_i, t_j), Priority(f_i, t_j) \in \{1,2, \dots ,9\}$ and α is a weighting factor that controls the weights of the $Value$ and $Priority$ functions, and $0 \leq \alpha \leq 1$. In this research, unless stated to the contrary, we assume that $\alpha = 0.5$. As stated earlier in Chapter 2, release planning is considered as multi-person decision making problem. According to [77], in a multi-person decision making process, the decision result is influenced by the decision weights of the decision makers. Therefore, in release planning of multi-tenant SaaS applications, tenants are given different weights where the opinions of certain tenants are considered more seriously than other tenants because of their business volume or loyalty. Hence, in order to measure the actual importance of a feature, the provided estimates about the importance of features are associated with tenants' decision weights [8, 76]. The decision weight of a tenant determines his importance to the SaaS provider, and is determined by the management [8]. There are many ways to determine the decision weights of the tenants. For example, in [8], AHP is used to calculate the relative importance of the

tenants. Each tenant is given a real value in the interval [0,1]. In [14], a scale from 1 to 9 is used. In this research we define $W(t_i)$ as a function that returns the decision weight of the tenant t_i such that: $W(t_i) \in [0,1]$ and $\sum_{i=1}^{m} W(t_i) = 1$. $W(t_i)$ can be calculated depending on tenants' business volume or loyalty. We will not elaborate in how to calculate $W(t_i)$ because it is out of the scope of this research. In order to reflect the decision weight of a tenant on the importance of various features, we define $Weighted\_Imp(f_i, t_j) \in R^+$ (positive real numbers) as the function that calculates the importance of the feature f_i from the perspective of the tenant t_j while taking his decision weight into consideration, such that $Weighted\_Imp(f_i, t_j) = Imp(f_i, t_j) \times W(t_j)$. According to [37], when planning for the next release, management aims to maximize the degree of tenants' satisfaction. In other words, we can say that the quality of the generated release plan can be evaluated by calculating the satisfaction of each tenant and the overall satisfaction of all tenants. Let ds_i denote the degree of satisfaction for tenant t_i. We define $Sats: T \times P(F^*) \times P(F^*) \rightarrow [0\ 1]$ as a function that calculates the degree of satisfaction for a tenant as follows:

$$ds_i = Sats(t_i, F_{ti}^{\wedge}, F^{\wedge}) = \frac{\sum_{k=1}^{n} \left(Imp(f_k, t_i) \times (f_k \in F^{\wedge}) \times \left(f_k \in F_{ti}^{\wedge} \right) \right)}{\sum_{j=1}^{n} \left(Imp(f_j, t_i) \times \left(f_j \in F_{ti}^{\wedge} \right) \right)} \qquad (3.3)$$

such that $P(F^*)$ is the power set of F^*, $\in$ returns 1 if an element is a membership of a set and returns 0 if it is not, and $ds_i \in [1,0]$. In this calculation, $F_{ti}^{\wedge}$ is the release plan that is generated by fulfilling technical, contractual and effort constraints, and making t_i the only decision maker, which means that $W(t_i) = 1$. In other words, $F_{ti}^{\wedge}$ is only generated according to the perspective of t_i. Note that $F_{ti}^{\wedge}$ is just used to measure the degree of satisfaction of a tenant t_i about the generated release plan ($F^{\wedge}$). Equation 3.3 calculates the

ratio of the cumulative importance of the features that are found in the set $F_{ti}^{\wedge} \cap F^{\wedge}$ to the cumulative importance of the features that in the set $F_{ti}^{\wedge}$. If $F_{ti}^{\wedge} \subseteq F^{\wedge}$, then the degree of satisfaction of tenant t_i is 1 (100%), and if $F_{ti}^{\wedge} \cap F^{\wedge} = \emptyset$, then degree of satisfaction is 0.

Example 3.1: Assume that $F^{*}= \{f_1, \ldots\ldots, f_{20}\}$, $F^{\wedge}= \{f_{10}, f_{15}\}$, $F_{t2}^{\wedge}=\{f_1, f_{15}, f_{20}\}$,

$Imp(f_{10}, t_2) = 3$, $Imp(f_{15}, t_2) = 9$, $Imp(f_1, t_2) = 8$, $Imp(f_{20}, t_2) = 8$.

It is required to calculate the degree of satisfaction of tenant t_2.

The degree of satisfaction of tenant t_2 (ds_2) can be calculated as follows:

$$\sum_{k=1}^{20}\left(Imp(f_k, t_i) \times (f_k \in F^{\wedge}) \times \left(f_k \in F_{t2}^{\wedge}\right)\right) = 0 + \ldots\ldots Imp(f_{15}, t_2) + 0 \ldots\ldots = 9$$

$$\sum_{j=1}^{20}\left(Imp(f_j, t_i) \times \left(f_j \in F_{t2}^{\wedge}\right)\right) = Imp(f_1, t_2) + 0. \ldots + Imp(f_{15}, t_2) +$$

$$0 \ldots Imp(f_{20}, t_2) = 8+9+8=25$$

$$ds_2 = \frac{9}{25} = 0.36.$$

The overall satisfaction ($OSat$) can be calculated as the additive weighting of the degree satisfactions of the tenants.

$$OSat = \sum_{i=1}^{m} ds_i \times W(t_i) \qquad (3.4)$$

The maximum value $OSat$ can take is 1, which means that the degree of satisfaction is 100%. The lowest value is 0, which means a degree of satisfaction of 0%.

Example 3.2: Assume that we have two tenants whose decision weights are 0.3 and 0.7, and whose degrees of satisfaction are $ds_1 = 0.8$ and $ds_2 = 0.6$ respectively, then $OSat = 0.3 \times 0.8 + 0.7 \times 0.6 = 0.66$, which means the degree of overall satisfaction is 66%.

Algorithm 3.1 shows how to calculate the degree of satisfaction of each tenant and the degree of overall satisfaction.

MEASURING THE DEGREE OF SATISFACTION OF TENANTS AND DEGREE OF OVERALL SATISFACTION

Input:
1)AUGMENTEDIMPORTANCE //n * m matrix containing the importance of n features from the perspectives of m tenants.
2)F^ //a vector containing the plan for the next release.
3)Avail_E //a real variable containing the available effort.
4)EFFORT_Vector // n elements vector containing the required effort of features.
5)W // m elements vector containing the decision weights of the tenants.
6) m // is the number of tenants.
7) D // n*n matrix containing dependencies among features.

Output:
1)Sat //a vector containing the degree of satisfaction of the tenants.
2) OSat //a real variable containing the degree of overall satisfaction of the tenants.

```
Declare Fti^ //a vector containing the plan for the next release from the perspective of tenant ti.
Declare Tenant_Eval // two columns matrix. The first column contains the feature id (integer), and the second column contains the evaluation of features provided by a tenant.
for i ← 1 to m do
    for j ← 1 to n do
        Tenant_Eval[j, 1] = j
        Tenant_Eval[j, 2] = AUGMENTEDIMPORTANCE[j, i]
    Sort (Tenant_Eval, 2, descending)//sort in descending order according to column 2 (the estimates of the tenant)
    Tenant_Eval = ApplyDependencies( Tenant_Eval ,D); // Ensuring that The precedents and coupling are applied )
    k ← 1
    while Avail_E>0 do
        Fti^ [k]= Tenant_Eval[k, 1]
        Avail_E = Avail_E - EFFORT_Vector[Fti^ [k]]
        k + +;
    AdditiveIntersectedImportance ← 0
    FIntersection ← intersect(F^,Fti^ ) // Finding common features in the two list
    for j ← 1 to length(FIntersection) do
        AdditiveIntersectedImportance+ =
        AUGMENTEDIMPORTANCE[FIntersection[j], i];
    AdditiveImportanceForTanant ← 0
    for j ← 1 to length(Fti^ ) do
        AdditiveImportanceForti+ =
        AUGMENTEDIMPORTANCE[Fti^ [j], i]
Osat ← 0
for j ← 1 to m do
    Osat+ = Sat[j] * W[j]
return Osat,Sat
```

Algorithm 3.1

Analysis of Algorithm 3.1: From the algorithm, it can be observed that the growth rate of running time of Algorithm 3.1 depends on two inputs: n which represents number of features, and m which represents number of tenants. Therefore, the time of Algorithm 3.1 can be denoted as a function of n and m. Let $Time(n, m)$ be the function that represents the growth rate of running time of the algorithm. We can observe that there are one outer loop with m iterations and 4 inner loops with n iterations for each (in the worst cases). Also, there are two functions: sort with complexity $(n\ log\ n)$[1] since the quick sort is used, and $ApplyDependencies$[2] function with complexity (n^3). Depending on this, the worst-case time of Algorithm 3.1 is as follows:

$$Time(n, m) = O(m * (4n\ +\ n\ log\ n\ +\ n^3) + m) =\ O(mn^3)$$

3.1.2 Maximizing Commonality

When planning for the next release, it is more efficient to select the features that are common by the highest possible number of tenants [6, 18]. This helps the release management to fulfill the requests of more tenants with less effort. We define the commonality of a feature f_i as follows:

$$Com(f_i) = \sum_{j=1}^{m} (f_i\ \in\ F_{tj}^{\wedge}) \qquad (3.4)$$

where $Com(f_i)$ is an integer, and $1 \leq Com(f_i) \leq m$. Let $F_{com}^{\wedge}$ be the release plan produced by fulfilling technical, contractual, and effort constraints, and achieves the highest

[1] The Complexity of sort function in Matlab (Matlab documentation http://www.mathworks.com/help/matlab/)

[2] This function call algorithms 4.2 and 4.3 which are shown in Chapter 4

possible degree of commonality. We defined the degree of the commonality of a release plan $F^{\wedge}$ as follows:

$$Com\_RP(F^{\wedge}) = \frac{\sum_{i=1}^{n} \left((f_i \in F^{\wedge}) \times (f_j \in F_{com}^{\wedge}) \times Com(f_i)\right)}{\sum_{j=1}^{n}\left((f_j \in F_{com}^{\wedge}) \times Com((f_i)\right)} \quad (3.5)$$

Equation 3.5 calculates the ratio of the cumulative commonality of the features that found in the set $F_{com}^{\wedge} \cap F^{\wedge}$ to the cumulative commonality of the features that in the set $F_{com}^{\wedge}$.

If $F_{com}^{\wedge} \subseteq F^{\wedge}$, then the degree of commonality of is 1 (100%), and if $F_{com}^{\wedge} \cap F^{\wedge} = \emptyset$, then the degree of commonality is *0*.

Example 3.3: Assume that $F^{*}= \{f_1, \ldots\ldots, f_{20}\}, F^{\wedge} = \{f_{11}, f_{15}, f_{20}\}, \ m = 10, F_{com}^{\wedge} = \{f_{11}, f_{15}, f_{19}\}, \ Com(f_{11}) = 9, Com(f_{15}) = 7, \ Com(f_{20}) = 1, Com(f_{19}) = 6.$

The degree of commonality of $F^{\wedge}$ is calculated as follows: $Com\_RP(F^{\wedge}) = \frac{9+7}{9+7+6} = 0.66.$

Algorithm 3.2 shows how to find the degree of commonality of a release plan $F^{\wedge}$.

Analysis of Algorithm 3.2: From the algorithm, it can be observed that the growth rate of running time of Algorithm 3.2 depends on the number of features (n). It can be observed that there are 4 loops with n iterations for each. Also, there are two functions: sort with complexity ($n \ log \ n$) since the quick sort is used, and $ApplyDependencies$[3] function with complexity (n^3). Depending on this, the worst-case time of Algorithm 3.2 is as follows:

$$Time(n) = O(4n + n \log n + n^3) = O(n^3)$$

$ApplyDependencies$ is the dominant function in this algorithm.

[3] This function call algorithms 4.2 and 4.3 which are shown in Chapter 4

MEASURING THE DEGREE OF COMMONALITY OF RELEASE PLANS

Input:
1)COMMONALITY //n elements vector containing the commonality of n features
2)F^ //a vector containing the plan for the next release.
3)Avail_E //a real variable containing the available effort.
4)EFFORT_Vector //n elements vector containing the required effort of features.
5) D // n*n matrix containing dependencies among features.
Output:
1)Commonality_Degree //a scalar containing the degree of commonality of a release plan.

```
Declare FCom^ //a vector containing the plan for the next release ;this release plan contains the features with highest degree of commonality
Declare CommTemp // Two columns matrix. The first column contains the feature id (integer), and the second column contains the commonality of features.
for j ← 1 to n do
    CommTemp[j, 1] = j
    CommTemp[j, 2] = COMMONALITY[j]
Sort(CommTemp, 2, descending)//sort in descending order according to column 2.
CommTemp = ApplyDependencies( CommTemp, D); // Ensuring that The precedents and coupling are applied. )
k ← 1
while Avail_E>0 do
    FCom^ [k]= CommTemp[k, 1]
    Avail_E = Avail_E - EFFORT_Vector[FCom^ [k]]
    k++;
AdditiveintersectedComm ← 0
FIntersection = intersect(F^,FCom^ ) // Finding the common features between the two lists.
for j ← 1to length(FIntersection) do
    AdditiveintersectedComm+ =
    COMMONALITY[FIntersection[j]]
AdditiveComm ← 0
for i ← 1 to length(FCom^ ) do
    AdditiveComm+ = COMMONALITY[FCom^ [i]])
Commonality_Degree = AdditiveintersectedComm/AdditiveComm
return Commonality_Degree
```

Algorithm 3.2

3.1.3 Minimizing Risk

According to [78], risk is a factor that release management tries to minimize when planning for the next release. The risk can be defined as “a measure of the probability and severity of adverse effects inherent in the development of software that does not meet its intended functions and performance requirements" [79]. In the release-planning context, the risk of a feature is the possible negative effects of the implementation of that feature on the quality, delivery time, and cost of the release under consideration. The risk of each feature must be estimated before planning is undertaken. Release management aims to minimize the risk of the overall release by selecting those features that have the lowest possible risk. The potential risk of a feature can be estimated by analyzing the following risk factors:

- Data related risk: According to [80] data security and integrity is the highest possible risky factor in SaaS applications. Therefore, the negative effects of features on the security of SaaS applications must be carefully considered. "A new model targeting at improving features of an existing model must not risk or threaten other important features of the current model" [9]. According to [9,10], the following are the key security aspects that should be carefully taken into account in SaaS applications:
  - Data security: SaaS providers must apply strong encryption and very restricted authorization techniques on tenants' data
  - Network security: Secure data flow must be ensured in order to prevent leakage of important tenants' information. This can be achieved by applying strong network traffic encryption techniques. .

- Data integrity: SaaS providers must ensure the validity and consistency of data. This issue becomes more critical when all tenants use the same database instance. "The lack of integrity controls at the data level (or, in the case of existing integrity controls, bypassing the application logic to access the database directly) could result in profound problems. Architects and developers need to approach this danger cautiously, making sure they do not compromise databases' integrity in their zeal to move to cloud computing" [9].
- Data access: SaaS providers must ensure that a user can only access the components that he is authorized to use. For example, in many multi-tenant SaaS applications, all tenants use the same database instance, which increases the probability that a tenant can access to an unauthorized data. "Role Based Access Control" is a technique that can be used.

For each feature, all of these security and data integrity factors shall be analyzed carefully. The features that have high risk on data integrity and security should have low chance to be assigned to the next release.

- Software components' related risk: The criticality of the components that will be modified when the feature is implemented should be considered. Usually, the critical components are those that comprise the codebase of the SaaS application. According to [13], SaaS providers try to avoid customizations that lead to changes in the codebase because of the severe potential consequences. Instead, configuration is used as possible. Additionally, the severity of the consequence may be even greater if the component that will be changed is tightly coupled with other components. Moreover, the quality of the component that will be modified when the feature is implemented

should be considered. According to [76], the quality of a component can be measured by analyzing the defects of that component in the previous releases. If a feature is related to software components that had many issues in the previous release, then the risk of implementing this feature may cause other issues for those components. More information about the effects of quality of components on estimates of feature risks can be found in [76].

- Features' attributes related risk: In this factor, the risk is associated with the degree of complexity, ambiguity, incompleteness, and volatility of the requirements of which the feature consists. Increasing the level of one or more of these factors leads to more risk [81].
- Development team related risk: The low level of expertise of the members of the development team can be considered as a source of risk [82]. For example, in some cases it may be necessary to adopt new technology (such as new programming language) in order to implement certain features. Special training may be necessary in order to make the development team familiar with this technology, and such training requires additional time and effort.

In order to estimate the risk of a feature, we must measure the risk exposure of that feature. Let RE_S, RE_Q, RE_A, RE_D be the risk exposures (RE) that are related to data integrity and security, software components' quality, the attributes of the requirements that compose the features, and development team expertise respectively. Then we define the risk exposure of a feature f_i as follows:

$$RE(f_i) = \alpha RE_S(f_i) + \beta RE_Q(f_i) + \gamma RE_A(f_i) + \delta RE_D(f_i) \qquad (3.6)$$

where α, β, γ, ε are the weighting factors, which give either more or less weight to each source of the risk of a feature f_i, and α + β + γ + ε =1. Using the definition of the risk exposure (RE) that stated in [83], the risk of any of these risk sources can be denoted as follows:

$RE_X(f_i) = Prob\big(Risk_X(f_i)\big) + damage(Risk_X(f_i))$ such that x denotes the source of risk ($x \in \{S, Q, A, D\}$). $Prob\big(Risk_X(f_i)\big)$ is the probability of the occurrence of the risk from the source x when feature f_i is implemented, and $damage(Risk_X(f_i))$ is the potential consequences of that risk. We assume that $RE(f_i) \in \{0, \dots, 9\}$ as in [78], where 0 denotes no risk, and 9 denotes the highest possible risk. The risk estimation of a feature is assumed to be an agreed-upon value that is provided by the designer and development team as stated in [14]. The quality of the projected release plan can be evaluated by measuring the degree to which it considers the risk factor. Maximum adherence to the risk factor is desired. Let $F^{\wedge}_{Risk}$ be the release plan produced by fulfilling technical, contractual, and resource constraints, and by completely considering the risk factor, which means guaranteeing that the features with the lowest risk are assigned to the next release. We define the adherence to the risk factor (Ad_risk_factor) as follows:

$$Ad\_risk\_factor = \frac{|\, F^{\wedge}_{risk} \cap F^{\wedge} \,|}{|F^{\wedge}_{risk}|} \qquad (3.8)$$

Ad_risk_factor is the ratio of the number of features in $F^{\wedge}_{risk} \cap F^{\wedge}$ to the number of features in $F^{\wedge}_{risk}$. We say that a release plan has completely adhered to the risk factor when Ad_risk_factor = 1.

Example 3.4: Let $F^{\wedge} = \{f_1, f_5, f_7, f_{11}, f_{15}, f_{17}\}$ and $F^{\wedge}_{risk} = \{f_{17}, f_4, f_7, f_{10}, f_5, f_2\}$. Adherence to the risk factor can be calculated as $Ad\_risk\_factor = \frac{|\{f_{17}, f_5\}|}{|F^{\wedge}_{risk}|}$ = 33%.

Algorithm 3.3 shows how to calculate the degree of adherence to risk of a release plan $F^{\wedge}$

MEASURING THE ADHERENCE TO RISK

Input:
1)RISK_Vector //a vector containing the risk of n features
2)F^ //a vector containing the plan for the next release.
3)Avail_E //a variable containing the available effort.
4)EFFORT_Vector //a vector containing the required effort of features.
5) D // a matrix containing dependencies among features.
Output:
1)Adh_Risk_Degree //a scalar containing the degree of the Adhered to the risk factor

```
1  Declare FRisk^ //a vector containing the plan for the next release ;this
   release plan contains the features with lowest possible risk
2  Declare RiskTemp // Two columns matrix. The first column contains the
   feature id (integer), and the second column contains the risk of features
3  for j ← 1 to n do
4  |   RiskTemp[j, 1] = j
5  |   RiskTemp[j, 2] = RISK_Vector[j]
6  Sort(RiskTemp, 2, ascending)//sort in ascending order according to
   column 2.
7  RiskTemp = ApplyDependencies( RiskTemp, D); // Ensuring that The
   precedents and coupling are applied. )
8  k ← 1
9  while Avail_E>0 do
10 |   FRisk^ [k]= RiskTemp[k, 1]
11 |   Avail_E = Avail_E - EFFORT_Vector[FRisk^ [k]]
12 |   k + +;
13 AdditiveintersectedRisk ← 0
14 FIntersection = intersect(F^,FRisk^ ) // Finding the common features
   between the two lists.
15 Adh_Risk_Degree = length(FIntersection)/length(FRisk^ )
16 return Adh_Risk_Degree
```

Algorithm 3.3

Analysis of Algorithm 3.3: From the algorithm, it can be observed that the growth rate of running time of Algorithm 3.3 depends on the number of features (n). From the algorithm, it

can be observed that there are two loops with n iterations for each. Also, there are two functions: sort with complexity ($n \log n$) since the quick sort is used, and $ApplyDependencies$[4] function with complexity (n^3). Depending on this, the worst-case time of Algorithm 3.3 is as follows:

$$Time(n) = O(2n + n \log n + n^3) = O(n^3)$$

$ApplyDependencies$ is the dominant function in this algorithm.

3.1.4 Discussion

The challenge for release management is that most of time the above three objectives are in conflict with one another. For example, it is typical to find that a feature that is very important to the tenants is also very risky. Furthermore, many tenants may require a feature that has significant impact on a critical component of the codebase, which may negatively affect the functionality or performance of other components. Therefore, a trade-off must often be undertaken in order to generate an effective release plan.

3.2 Planning Constraints

This section presents three types of constraints that must be taken into account in a planning endeavor: contractual, effort, and dependencies among features.

3.2.1 Contractual Constraints

Contractual constraints are related to the level and quality of services that a SaaS provider guarantees to his tenants [5]. The service level agreement (SLA) is the document that includes all the descriptions and limitations of the service. Usually, tenants differ in their QoS criteria; for example, one tenant may place a high priority on receiving a low-

[4] this algorithm is shown in Chapter 4

priced service regardless of the performance, while other tenants may give high consideration to the performance regardless of the cost. Thus, SaaS providers offer different levels of service with different pre-defined SLAs. Each SLA is associated with a certain level of the service [12, 84]. The release management needs to take the SLA of tenants into account when planning for the next release. Tenants' requests to add (or enhance) a feature are verified against their SLAs. If the feature that is required by a tenant is compliant with his SLA, then his estimate of the importance of this feature is counted; otherwise, he will be notified that he needs to upgrade to the service level that includes this feature. If he agrees to do the upgrade, then his estimate for the importance of this feature will be considered; otherwise, it will be ignored (set to 0). For example, suppose a SaaS application is offered at two different levels: standard and premium. The standard level allows its subscribers to make changes to the user interface (UI), but does not allow any changes in the business logic. On the other hand, the premium level allows the subscriber to add functionalities to both UI and business logic interfaces. Requests by standard-level tenants that relate to additions or enhancements to business logic functions will not be included in the planning. Formally, let $S = \{S_1, S_2 \dots S_p\}$ be the set that represents the levels of service of the SaaS application. We represent each level of a service S_i as a set of the features that are offered (or allowed to be offered) by this level of service, so that $S_i = \{ f_i^1 , f_i^2, \dots\dots, f_i^u\}$, such that $i \in \{1,2, \dots . p\}$ and f_i^j means feature j of the service level S_i. In addition, we define the operator "≤" on the set S, such that $S_i \leq S_k \leftrightarrow S_i \subseteq S_k$, which means the upper service level includes all the features of the lower service levels. Let $F\_Se\_L\ (f_i) \in P(S)$ be a function that returns the levels that will include the feature f_i, such that $P(S)$ is the power set of the set S. Also, let $Get\_Se\_L(t_i) \in S$ be the function that returns the service level to which the

tenant t_i has subscribed. Depending on the two functions above, the following constraint must be satisfied by the generated release plan:

$$\forall f_i \in F^{\wedge} \leftrightarrow (\exists\, t_j \in T : Get\_Se\_L(t_j) \in F\_Se\_L(f_i)) \qquad (3.8)$$

which means that for each feature assigned to the next release, at least the SLA of one tenant must comply with of the level that will include the feature. The contractual constraint implies that the estimate of the importance of features provided by a tenant that is not eligible to have this feature is omitted. Formally,

$$\forall f_i \in F^* \forall t_j \in T\ (\sim(Get\_Se\_L(t_j) \in F\_Se_L(f_i)) \rightarrow Imp(f_i, t_j) = 0) \qquad (3.9)$$

such that $(\sim)$ is the negation operation. Depending on (3.9), we define the function $Augmented\_Imp$, which is the function that returns the importance of a feature from the perspective of a tenant taking into account contractual constraints.

$$Augmented\_Imp(f_i, t_j) = Imp(f_i, t_j) \times (Get\_Se\_L(t_j) \in F\_Se\_L(f_i)) \qquad (3.10)$$

The aim of $Augmented\_Imp(f, t_j)$ is to count the estimates of only the tenants whose SLAs comply with the requested features. Moreover, the commonality function is re-defined in order to fulfill the contractual constraint as:

$$Augmented\_Com(f_i) = \sum_{j=1}^{m} ((f_i \in F_{tj}^{\wedge}) \times (Get\_Se\_L(t_j) \in F\_Se\_L(f_i))) \qquad (3.11)$$

$Augmented\_Com(f_i)$ considers the compliance of a feature to the SLA of tenants in the calculation of the commonality of a feature.

In summary, the SLA of each tenant is considered in the calculation of the importance of each feature; such that, the estimate of the importance of a feature provide by a tenant that is not eligible to have this feature will be omitted. In addition, when a tenant

has a request to add a feature and he is not eligible to have this feature, his request will not be considered in calculating the commonality of that feature.

3.2.2 Effort Constraints

Estimating available and required effort is an essential task in release planning [8, 33, 37, 14]. First, let us define two terms that will be used in this section:

- Story points [35]: An abstract relative metric that measures the complexity and difficulty needed to deliver a feature. For example, a feature that requires 4 story points is double in difficulty and complexity to the feature that requires 2 story points.

- Velocity [35]: The number of story points that the development team can finish in a time-boxed period (for example 2 weeks). In XP, term iteration is used to denote this time-boxed period.

In order to measure the available effort, the following steps are performed:

- Set the beginning and ending dates of the release
- Set your velocity (it can be obtained from previous releases)
- Calculate the number of iterations (an iteration is a period of x weeks)

$$Number\ of\ iterations\ = (release\ delivery\ date - release\ start\ date)/7x \quad (3.11)$$

- Calculate available effort as:

$$Avail\_E\ =\ Number\ of\ iterations\ \times velocity \quad (3.12)$$

The release managers are responsible for estimating the available and required effort [76]. They shall select the features whose total required effort fits the available effort. Formally, the following constraint must be satisfied by the generated release plan:

$$\sum_{i=1}^{n}\left(Re\_Effort(f_i) \times (f_i \in \hat{F})\right) \leq Avail\_E \qquad (3.13)$$

such that $Re\_Effort(f_i) \in \mathrm{R}^+$ represents the effort needed to implement a feature f_i measured by story points. It is important to point out that story points are derived using different techniques [35]:

- Expert opinion: the estimates depend on intuition and expertise.
- Analogy: the feature is estimated with other features, which means relative estimation is performed between the features.
- Disaggregation: a feature is split into smaller features, which will be easier to estimate.
- Planning poker: where the all three of the above techniques are used to estimate the effort. Planning poker requires many developers to work together in the estimation endeavor.

3.2.2 The Constraints of Dependencies among Features

The dependencies among features require significant consideration in the release planning process. According to [85], most features depend on one another in many ways. There many times of constraints such as either or, at least one, at most one, coupling and precedence [85]. Many research about release planning consider the last two types of constraints [8,14,32,33,37,40]. Therefore, In current research, we consider only these two

types of constraints in order to maintain simplicity of the proposed formulation and be compatible with other release planning works.

3.2.2 .1 Coupling

Two features or more are coupled when they should be delivered in the same release. This can occur for several reasons. For example, if we have features f_1 and f_2, then the following scenarios can make these two features coupled:

- The importance of delivering the two features together (in the same release) is more than the sum of the importance of the features if each one is delivered separately (in different releases).
- The effort needed to deliver the two features together is less than the sum of the effort if they are delivered individually.
- Each one of the two features cannot be functional without the existence of the other feature, which means $f_1\ is\ functional \leftrightarrow f_2$ is functional.

When planning for the next release, the coupling constraints shall be satisfied. Formally, let $Coupling \subset F^* \times F^*$ be a binary relation, such that $\forall\ (f_i, f_j) \in Coupling \rightarrow G_{F^\wedge}(f_i) = G_{F^\wedge}(f_j)$, $G_{F^\wedge}$ is the characteristic function of the release plan set ($F^\wedge$), as defined in equation (3.1). This relation denotes that either the two features assigned to the next release or both of them are assigned to a future release.

3.2.2 .2 Precedence

A feature f_i precedes a feature f_j when f_i should be delivered (or at least implemented and tested) prior to feature f_j. There are many reasons why a feature should precede other features, as in the following scenarios:

- f_j cannot be functional without the existence of f_i, which means

  $f_j \, is \, functional \rightarrow f_i \, is \, functional$

- Marketing policies play a significant role in the precedence between features. For example, it is common in software industries to deliver features in different releases in order to keep their customers attracted to their product.

There are two types of precedence: weak and strict. In a weak precedence, f_i and f_j can be implemented in the same release; however, f_j should not be implemented in an earlier release than f_i. In strict precedence, f_i and f_j cannot be implemented in the same release, and f_i should be implemented in an earlier release than f_j. Formally, we define these relations as follows: 1) $Weak\_Precedence \subset F^* \times F^*$ as a binary relation such that $\forall (f_i, f_j) \in Weak\_Precedence \rightarrow G_{F^\wedge}(f_i) \geq G_{F^\wedge}(f_j)$, and 2) $Strict\_Precedence \subset F^* \times F^*$ as a binary relation such that $\forall (f_i, f_j) \in Strict\_Precedence \rightarrow G_{F^\wedge}(f_i) > G_{F^\wedge}(f_j)$. For the purpose of this research, we assume that the precedence between features is of the first type (weak precedence). For simplicity, we call this relation $precedence$.

3.3 Summary of the Chapter

Planning for the next release for multi-tenant SaaS applications aims to maximize stakeholders' satisfaction, to maximize commonality of features, and to minimize the potential risk. In addition, release planning shall consider effort constraint, where the required effort to implement the selected features shall be equal to or less than the available effort. Additionally, release planning shall comply with the contractual constraints. Moreover, release planning shall take dependencies constraints into account. In this thesis, we consider two dependencies constraints: 1) coupling and 2) precedence. Table 3.1 shows

all the variables discussed in this chapter with the design goals and motivation of including them in the release planning for multi tenants SaaS applications

Table 3.1: The Design Goals of the Variable of Release Planning in SaaS

Problem Variables	Type	Design Goals
Tenants satisfaction	objective	Popularity of SaaS applications is a significant indicator of the of success of the software [6]. It can be increased by maximizing tenants' satisfaction.
Risk	Objective	The high quality of software can increase its popularity. High quality means including all the required functional requirements while considering the non-functional requirements such as security and performance. Postponing the features that have risk on non-functional or functional requirements to later releases increase the quality of the software.
Commonality	Objective	selecting the features that are required by the highest possible number of tenants helps the release management to fulfill the requests of more tenants with less effort.
Contractual constraints	Constraint	SaaS are offered is different service levels, which makes it important to fulfill the needs of tenants according to their SLA.
Effort (required and available)	Constraint	SaaS providers have limitations in effort that prevent them to fulfill all the requested features. Therefore, they select subset of features while taking into account their effort limitations.
Dependencies constraints	Constraint	Most of features depend on one another in different ways. The quality of the delivered release can decrease if features are implemented with violating their dependencies since the functionalities of these features may not be completed.

CHAPTER 4: FUZZY INFERENCE SYSTEM-BASED APPROACH

4.1 Introduction

Chapter 3 explored the variables that affect the release planning process for SaaS applications. Most of these variables are subject to uncertainty, for two reasons [34, 86]

- **Insufficient knowledge about the features on the part of stakeholders**: For example, a developer may estimate the risk of a feature as "low," when in fact it is not, or a tenant may consider a feature as “highly urgent,” when in fact it can be postponed to a later release;.
- **The nature of features:** Some features are ambiguous or poorly analyzed, making them difficult to evaluate accurately.

Because of this uncertainty, release planning problems cannot be resolved without human intuition. As stated in [32] "Any formalized computational technique in isolation is unlikely to determine meaningful results because only a subset of the reality can be taken into account." Many existing approaches to release planning use human expertise as a final stage in order to tune the solutions that have been generated by computation models [37]. Those approaches may be practical in systems with limited numbers of features and stakeholders. However, such approaches are not feasible in the development of many SaaS applications, where thousands of tenants located in different locations participate in the planning endeavor. Therefore, it is imperative that a system be created that will allow release management to automatically and implicitly incorporate human expertise into the formulized solution of release planning problems. In many decision making applications [87, 88], linguistic rules have proved highly effective in capturing human expertise. In linguistic rules, the antecedents and premises are described using linguistic terms. These

terms can be considered as an approximation of the values that the estimated factors can take. In the current work, we use Mamdani types FIS [89] to build the linguistic rules, and then we synthesize the stakeholders' input using these rules in order to produce a rank for each feature. The features with the highest ranks have the greatest chance of being assigned to the next release.

The rest of this chapter is organized as follows: The preliminaries of fuzzy set theory are presented in Section 4.2. The process of Mamdani FIS is described in Section 4.3. Section 4.4 presents the proposed approach. The applicability of the proposed approach is illustrated by a proof-of-concept example in Section 4.5. The summary of the chapter is stated in Section 4.6.

4.2 Preliminaries

4.2.1 Basics of Fuzzy Set Theory

Fuzzy set [50] is a generalization of crisp set. The elements of a crisp set A in the universe of discourse U can be characterized using characteristic function as:

$$\mu_A(x) = \begin{cases} 1 & \leftrightarrow x \in A \\ 0 & \leftrightarrow x \notin A \end{cases} \quad (4.1)$$

In a fuzzy set, the characteristic function is generalized to the membership function. Within a set A, the different elements have different grades; a higher grade means a higher degree of membership. A fuzzy set A is defined using its membership function as:

$$A = \{(x, \mu_A(x)) | \mu_A(x) \in [0\ 1]\} \quad (4.2)$$

such that $\mu_A(x)$ shows the membership degree of element x to A. $\mu_A(x)$ can be described as follows:

$$\mu_A(x) = \begin{cases} 1 & if\ x\ \ is\ completely\ in\ A \\ 0 & if\ x\ is\ not\ in\ A \\ 0 < \mu A(x) < 1 & if\ x\ is\ partially\ in\ A \end{cases} \qquad (4.3)$$

Example 4.1: Assume a fuzzy set A (shown in Figure 4.1) is defined using the following membership function:

$$\mu_A(x) = \begin{cases} 0 & if\ x\ \in\ (-\infty, 0) \\ \frac{x}{6} & if\ x\ \in [0,6) \\ \frac{12-x}{6} & if\ x\ \in [6,12) \\ 0 & if \in\ (12, +\infty) \end{cases} \qquad (4.4)$$

We can see that $\mu_A(4) = 0.66$, $\mu_A(6) = 1$, and $\mu_A(13) = 0$, which means that the elements 4, 6, and 13 are members of A in degrees of 0.66, 1, and 0 respectively.

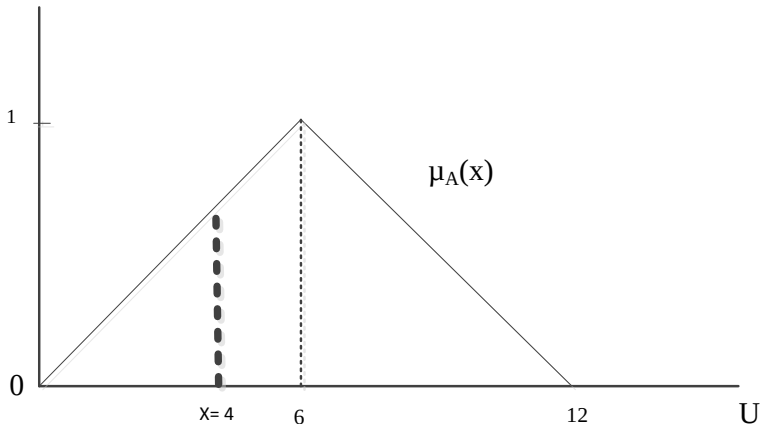


Figure 4.1: Example of a Fuzzy Set A

The operations such as the complement, union, and intersection of basic crisp sets can be generalized to fuzzy sets. If A, B are fuzzy sets in a universe of discourse U then the following operation are defined:

Complement: $\mathrm{A}^{\mathrm{c}} = \{(x, \mu_{\mathrm{A}}{}^{\mathrm{c}}(x))|\ \mu_{\mathrm{A}}{}^{\mathrm{c}}(x) = 1 - \mu_{\mathrm{A}}(x)\}$

Union: $A \cup B = \{(x, \mu_{\mathrm{A}\cup\mathrm{B}}(\mathrm{x}))|\ \mu_{\mathrm{A}\cup\mathrm{B}}(x) = max(\mu_{\mathrm{A}}(x), \mu_{\mathrm{B}}(x))\}$

Intersection: $A \cap B = \{(x, \mu_{\mathrm{A}\cap\mathrm{B}}(x))|\ \mu_{\mathrm{A}\cap\mathrm{B}}(x) = min(\mu_{\mathrm{A}}(x), \mu_{\mathrm{B}}(x))\}$

Fuzzy subset, equality, support and core of fuzzy sets are defined as follows:

Fuzzy subset: $A \subseteq B \rightarrow \mu_{\mathrm{A}}(x) \leq \mu_{\mathrm{B}}(x)$

Fuzzy sets equality: $A = B \leftrightarrow \mu_{\mathrm{A}}(x) = \mu_{\mathrm{B}}(x)$

Support of a fuzzy set: $Support\ (A) = \{x|\ \mu_{\mathrm{A}}(x) > 0\}$

Core of a fuzzy set: $Core(A) = \{x|\ \mu_{\mathrm{A}}(x) = 1\}$

A fuzzy singleton is a fuzzy set whose support is a single point in the universe of discourse.

The height of a fuzzy set $Height\ (A)$, is the maximum grade of membership in A, which is defined as: $Height(A) = supremum\ (\mu_{\mathrm{A}}(x))$.

A is **a normal fuzzy set** when $Height(A) = 1$

A is **a convex fuzzy set** if $\forall \lambda \in [0,1]\ (\mu_{\mathrm{A}}(\lambda \mathrm{x}_1 + (1-\lambda)\mathrm{x}_2) \geq \min(\mu_{\mathrm{A}}(x_1), \mu_{\mathrm{A}}(x_2)))$

A **fuzzy number** [90] is a normal and convex fuzzy subset in the universe of discourse R (real numbers). A fuzzy number F can be identified by the tuple $[a, b, c, d, w]$ and the membership function μ_F as follows:

$$\mu_F(x) = \begin{cases} \mu_F^L(x), a \leq x < b \\ w, b \leq x < c \\ \mu_F^R(x), c \leq x \leq d \\ 0\ d < x \end{cases} \qquad (4.5)$$

such that $\mu_F^L(x): [a, b] \rightarrow [0,1]$ and $\mu_F^R(x): [c, d] \rightarrow [0,1]$. The height of the fuzzy number is w. If $w = 1$, then F is a normal fuzzy number. The characteristics of $\mu_F(x)$ are as follows:

- $\mu_F(x): R \rightarrow [0\ ,1]$ is continuous.
- $\mu_F(x)$ is monotonic increasing on $[a, b]$ and monotonic decreasing on $[c, d]$.

When $b \leq c$, then F becomes a trapezoidal fuzzy number. F becomes a triangular fuzzy number if it is trapezoidal and $b = c$.

The $\boldsymbol{A^{\alpha}}$ **(**$\boldsymbol{\alpha}$**-cut of** $\boldsymbol{A}$**)** is a subset of A, such that $A^{\alpha} = \{x|\ \mu_A(x)\ \geq\ \alpha\ \}$

If A is a trapezoidal fuzzy number then we can write $A^{\alpha} = [a\ + \alpha\ (b\ -\ a), d\ -\ \alpha\ (d\ -\ b)], \forall \alpha\ \in [0,1]$.

4.2.2 Fuzzy Numbers Arithmetic

In fuzzy numbers arithmetic [90], the interval arithmetic on the α-cut of fuzzy numbers is used as the arithmetic on fuzzy numbers. Let $\otimes$ denote any of the four main arithmetic operations $(+,-,\div,\times)$, and let A and B be two linear fuzzy numbers; then $(A \otimes B)^{\alpha} = A^{\alpha} \otimes B^{\alpha}$ such that $\alpha\ \in [0\ 1]$. This Equation is not applicable when $0 \in {}^{\alpha}B$ and $\otimes = \div$. Let A and B are trapezoidal fuzzy numbers that are defined as follows: $A = [a_A\ , b_A\ , c_A\ , d_A\ ],\ B = [a_B\ , b_B\ , c_B, d_B]$, then the four basic operations are applied as follows:

$$A\ +\ B\ =\ [a_A + a_B\ , d_A + d_B]$$

$$A\ -\ B\ =\ [a_A - d_B\ , d_A - a_B]$$

$$A\ \times\ B\ =\ [\ min(\ a_A a_B\ , a_A d_B\ , d_A aB\ , d_A d_B),\ max\ (a_A a_B\ , a_A d_B\ , d_A a_B\ , d_A d_B)]$$

$$A\ /B\ =\ [\ min(\frac{a_A}{a_B}, \frac{a_A}{d_B}, \frac{d_A}{a_B}, \frac{d_A}{d_B})\ , max\left(\frac{a_A}{a_B}, \frac{a_A}{d_B}, \frac{d_A}{a_B}, \frac{d_A}{d_B}\right)],\ 0\ \notin\ [a_B, d_B]$$

Example 4.2: Figure 4.2 shows the four arithmetic operations for fuzzy numbers

$$A = [1\ 2\ 3\ 4], B = [4\ 5\ 6\ 7].$$

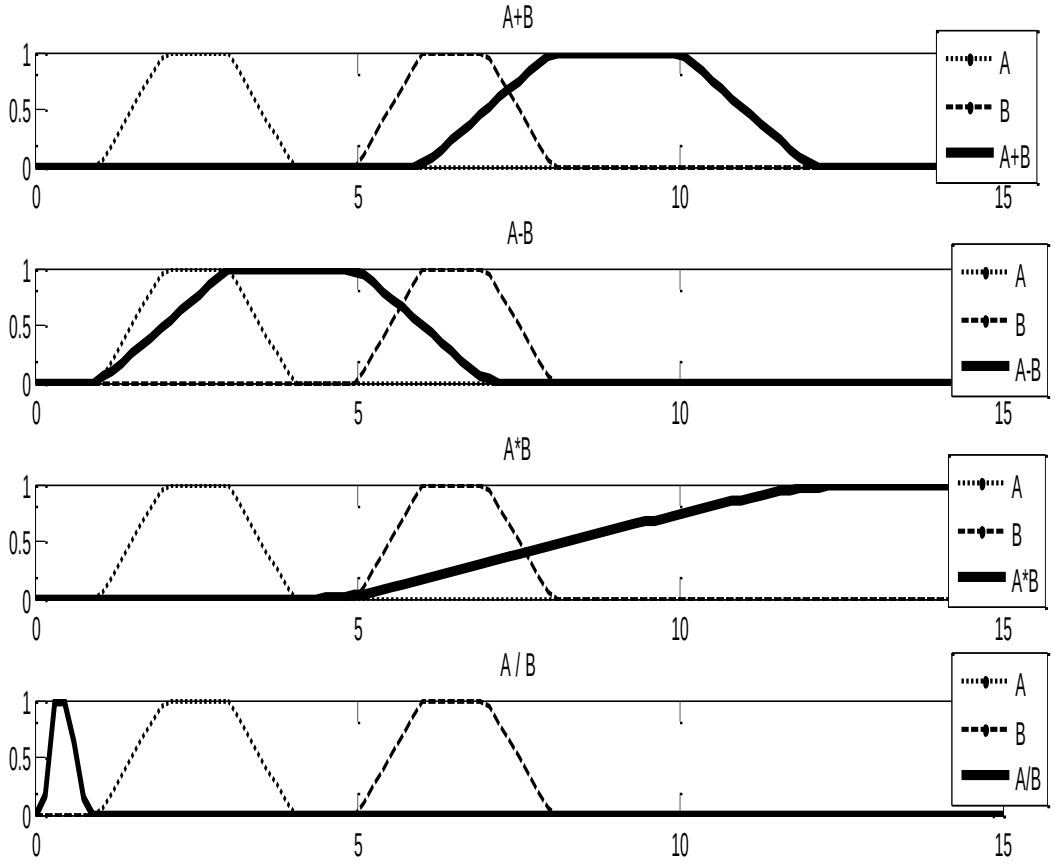


Figure 4.2: Fuzzy Numbers Arithmetic

4.2.3 Linguistic Variables

A linguistic variable [91] is one in which the values are linguistic terms (words). This is the natural way in which humans express their knowledge about most variables in their daily life. As stated in [91] "A particularly important area of application for the

concept of a linguistic variable is that of approximate reasoning, by which we mean a type of reasoning which is neither very precise nor very imprecise". For example, “distance” is a linguistic variable that can take linguistic values in the set {close, not far}. "A linguistic variable can be defined by a quintuple $\{X, T(X), U, G, M\}$ in which X is the name of the variable, $T(X)$ is the values set (term-set) of X, U is the universe of discourse, G is a syntactic rule which produces the term-set $T(X)$, and M is the semantics rule which associates each linguistic value to its meaning" [91]. The base variable for a linguistic variable is a numerical variable that can take numbers (real, integer, etc.).

Example 4.3: As figure 4.3 shows, distance is the fuzzy variable that may take two fuzzy values $\{“Close”, “Not\ Far”\}$. These two values are restrictions on the base value (distance in Km).

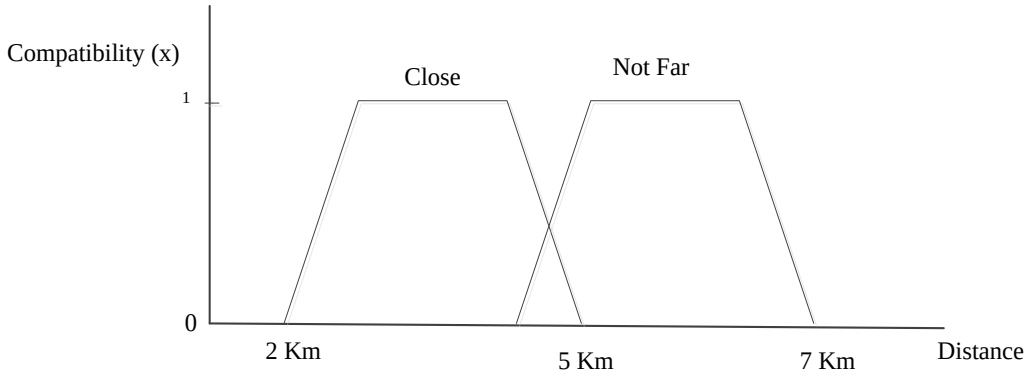


Figure 4.3: Example of a Fuzzy Variable

4.3 The Mamdani Fuzzy Inference Systems Process

A fuzzy inference system (FIS) process can be described as a function that receives input and maps it to output depending on certain rules. The main characteristic of the FIS process is that it mimics the human way of reasoning, meaning that a set of linguistic

variables interact with each other to generate the output [92]. There two well-known FIS types: Mamdani [89] and Sugeno [93]. The consequence in Mamdani is a fuzzy variable that takes fuzzy variables (linguistic terms), and defuzzification is used in order to calculate the output. In Sugeno, the consequence is a linear function, and the output is calculated using the weighted average of the firing strength of the rules. We have used Mamdani in this research due to the following reasons:

- Mamdani has the ability to acquire the human knowledge in an intuitive and human-like manner [94].
- "Mamdani has expressive power, easy to formalize, intuitive and interpretable nature of the rules, and widely used in decision support applications" [95].
- "Mamdani fuzzy model is more interpretative than Sugeno fuzzy models from a human perspective" [96].
- Mamdani is more transparent than Sugeno from the perspective of representing human knowledge. Therefore, Mamdani models are usually used in modeling human expert knowledge [97].

In order to design a Mamdani FIS, two components must be defined.

4.3.1 The Database

The database component contains the parameters relating to the input and output variables. For example, the database includes the name of the input and output variables, the linguistic values that input variables can take, and the meaning (parameters) of these linguistic terms. In detail, the main artifacts of the database component are as follows:

- Linguistic variables $X_1, X_2, \dots, X_n$ that are defined in the universe of discourses $U_1, U_2, \dots, U_n$ respectively. For the purpose of this research, we assume that$U_1, U_2, \dots, U_n \subset$ R. In Multi Input Single Output (MISO) FIS, there are many input variables and only one output variable; i.e. $\exists! X_j \in \{X_1, X_2, \dots\dots, X_n\}$ which represents the output variable of the FIS module.
- Linguistic terms contained in $T(X_1), T(X_2), \dots, T(X_n)$, such that $T(X_i)$ is a set of linguistic terms associated with the linguistic variable X_i.
- $M(X_i)$ which is a set that contains the semantics of linguistic terms. The semantics of linguistic terms are the fuzzy number associated with each linguistic term (see the definition of linguistic variables in section (4.2.3)). Each term t in $T(X_i)$ is associated with a fuzzy number in $M(X_i)$. This fuzzy number represents the semantics of the term t. In other words, $\forall\, t \in T(X_i)\ \exists\, \mu_t \in M(X_i)$ such that μ_t is the meaning of t.
- The parameters that define the fuzzy numbers. For example, if μ_t is a trapezoidal fuzzy number, and represents the semantics of t, then the database contains the values of $[a, b, c, d, w]$ of μ_t. The parameters of fuzzy numbers can be determined in several ways, such as direct rating, polling, interval estimation, membership function exemplification, pairwise comparison, and reverse rating [97].

4.3.2 The Rule Base

The rule base contains the IF-Then fuzzy rules. An IF-Then fuzzy rule can take the form of: $IF\ X\ is\ A\ THEN\ Y\ is\ B$ where $A \in T(X)\ and\ B \in T(Y)$, $T(X)$ and $T(Y)$ are the sets that contain the linguistic terms for the variables X and Y respectively. Fuzzy rules are

evaluated by turning them into fuzzy implications. A fuzzy logic implication is a generalization of a crisp logic implication. A fuzzy implication can be denoted by the function $I: [0,1] \times [0,1] \rightarrow [0,1]$ such that $I(x,y) = Sup\{z \in [0,1] | \ min\ (z,x) \leq y\}$, and x, y are the result of the defuzzification processes of $X\ is\ A$ and $Y\ is\ B$ respectively. The defuzzification process is discussed in detail in the next section.

4.3.3 The Inference Process

Assume that a FIS has n input variables $X_1, X_2, \dots, X_n$ and one output variable Y, and receives n crisp inputs $x_1, x_2, \dots\dots, x_n$ in the universe of discourses $U_1, U_2, \dots, U_n$ respectively. Let L be the number of the rules such that $1 \leq L \leq \prod_{i=1}^{n} |T(X_i)|$. A rule $R_k, (k = 1 \dots L)$ can be defined as follows:

$$R_k = \ (X_1^k\ is\ t_{X1}^k \ \circledast\ X_2^k\ is\ t_{X2}^k \ \dots\dots\dots\dots \circledast\ X_m^k\ is\ t_{Xm}^k) \ \rightarrow Y^k\ is\ t_y^k \qquad (4.6)$$

such that X_i^k represents the premise number i in the rule k, $\{X_1^k, X_2^k, \dots\ X_m^k\} \subseteq \{X_i, X_2, \dots\dots\dots X_n\}, t_{Xi}^k \in T(X_i^k)$, and $\circledast$ is T-norm or T-conorm operators. We will use $Minimum\ (Min)$ T-norms operator. The following processes are applied in order to map inputs to an output using FIS:

- Evaluation of the output (conclusion) of each rule: We define $O(R_k)$ as the a function that calculates the output of the rule R_k where

$$O(R_k) = \ Min\ \left(Min\left(\mu_{t_{X1}^k}(x_1^k), \mu_{t_{X2}^k}(x_2^k), .., \mu_{t_{Xm}^k}(x_m^k)\right), \mu_{t_y^k}(y)\right) \qquad (4.7)$$

$\mu_{t_{Xi}^k}(x_i^k)$ is the fuzzificaztion process. This process aims to find how much the base variable x_i^k (which is crisp) is a member of the fuzzy set t_{Xi} and t_{Xi} is a linguistic term that Xi (the linguistic variable) can take. The inner Min calculates the firing strength of the rule. The outer Min applies the implication operation of the rule. The

output of the implication is a fuzzy set that results from truncating the fuzzy set in consequence of the rule. The variable y represents the support value of the membership function $\mu_{t_y^k}$.

- Aggregating all the rules: The output of the previous step is L fuzzy sets that represent the conclusions of L rules. In the aggregation process those L generated fuzzy sets are unified in one set; let us call this set Z which can be defined as follows:

$$Z = \bigcup_{i=1}^{L} O(R_i) \qquad (4.8)$$

that means: $\mu_Z(x) = Max\ \left(\mu_{O(R1)}(x), \mu_{O(R2)}(x), \dots, \mu_{O(RL)}(x)\right)$.

- Defuzzification: In this step, we calculate the result of the inference process, which is obtained by calculating the centroid of Z

$$Centroid\ (Z)\ = \frac{\int_R x\mu_Z(x)}{\int_R \mu_Z(x)} \qquad (4.9)$$

Example 4.4: Assume importance, risk, and rank of software features are three linguistic variables. The terms of these to variables are defined as follows: $T(importance) = \{LowI, MidI, HighI\}$, $T(risk) = \{LowR, MidR, HighR\}$, $T(rank) = \{Lowrank, Midrank, Highrank\}$.

Assume the fuzzy rules are defined as follows:

$R1$: $if\ importance\ is\ low\ \ and\ risk\ is\ low\ \ then\ rank\ is\ medium.$

$R2$: $if\ importance\ is\ medium\ and\ risk\ is\ low\ then\ rank\ is\ medium.$

$R3$: $if\ importance\ is\ high\ and\ risk\ is\ low\ then\ rank\ is\ high.$

Figure 4.4 shows the process of the FIS when the $risk = 0.151$ and $importance = 0.868$.

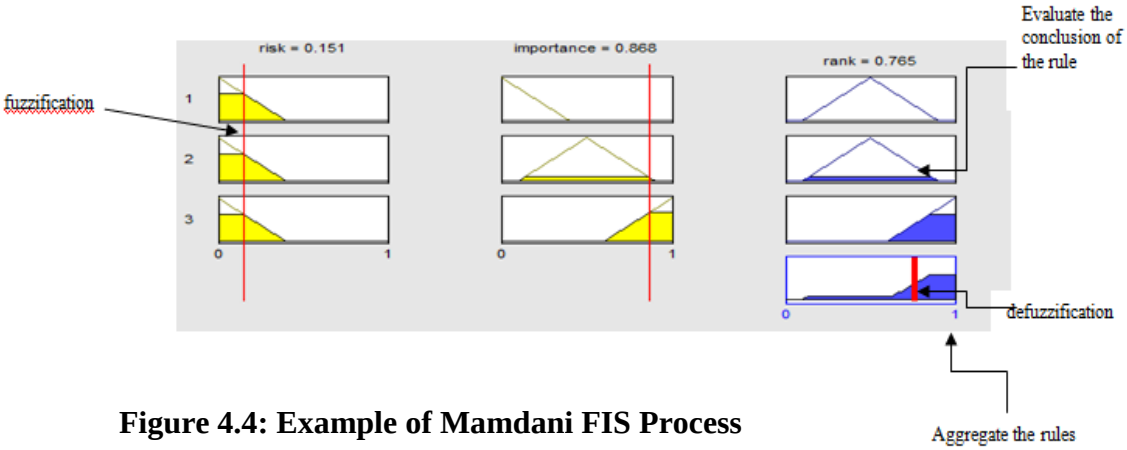


Figure 4.4: Example of Mamdani FIS Process

4.4 The Proposed FIS-based Approach

As Figure 4.5 shows, the proposed approach consists of four processes: raw data collection, preprocessing, ranking, and release plan generation. The output of a process is considered as the input of the next process.

4.4.1 Raw Data Collection

As Figure 4.6 shows, in this process, the release management collects and organizes the data about the variables that control the planning for the next release. The data is collected through eight different processes (the boxes with bold borders in Figure 4.6). The output of the "Raw Data Collection" process is comprised of the following eight data structures:

- Tenants' decision weights: Each tenant is given a decision weight. This weight represents the importance of this tenant to the SaaS provider. For example, tenant weight may be calculated depending on tenant loyalty or

volume trade. Release management obtains the information about decision weights from the tenants' profiles. Let W be an m elements vector that contains the decision weights of the tenants, such that $W[i]$ contains the decision weight of the tenant i. The decision weights of the tenants are normalized to 1, which means $\sum_{i=1}^{m} W[i] = 1$.

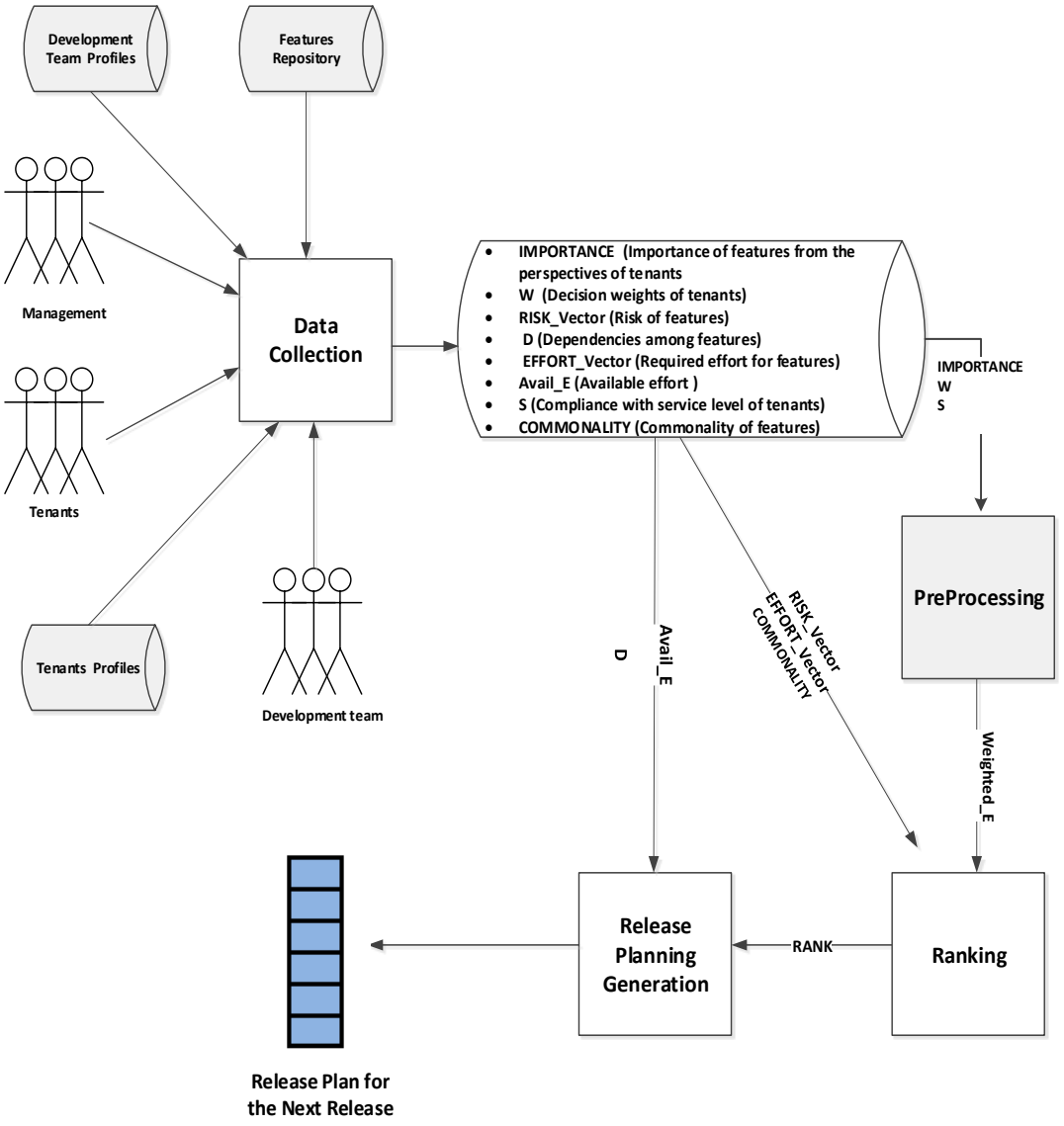


Figure 4.5: The Proposed FIS-based Approach

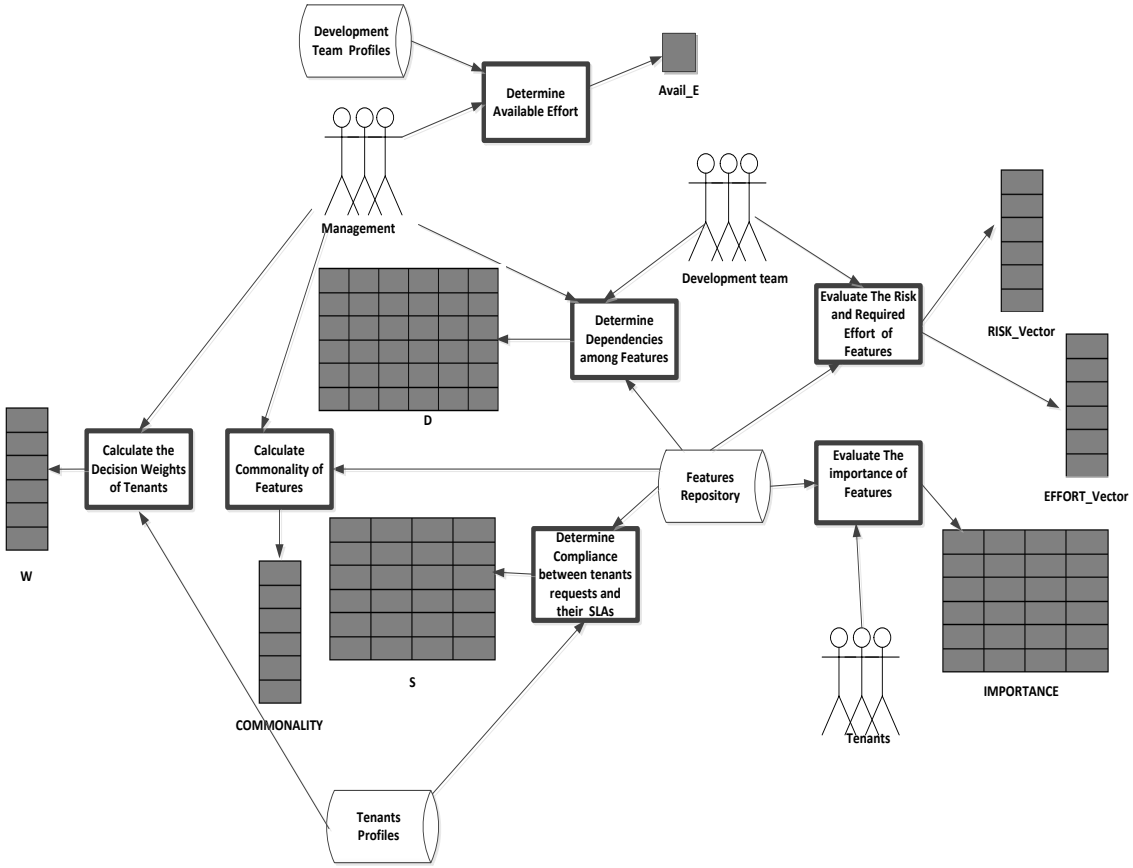


Figure 4.6: The Raw Data Collection Process

- Compliance between tenants' requests and their SLAs: As stated in Section 3.2.1, the SLAs of the tenants must be considered in the planning process. A request of a tenant t_i to include a feature f_j in the next release must be validated against the SLA of that tenant. The requests of tenants can be obtained using a user interface that is usually part of SaaS applications. Let $F\_Se\_L\ (f_i)$ be the function that returns the service levels that will include the feature f_i, and $Get\_Se\_L(t_j)$ is the function that returns the service level to which the tenant t_{i} has subscribed, then we have to validate that $Get\_Se\_L(t_j) \in F\_Se\_L\ (f_i)$ (see

Section 3.2.1.). To capture the data about the features and their compliance to the SLAs of the tenants, we define S as $n \times m$ matrix, such that $S[i,j] = (Get\_Se\_L(t_j) \in F\_Se\_L\ (f_i))$.

- Importance of the features: Each tenant provides his or her estimate about the importance of each feature. A user interface can be used to obtain the estimates of tenants about the importance of features. As defined in Chapter 3, $Imp(f_i, t_j)$ is the estimate of the feature f_i from the perspective of tenant t_j, where $i = 1 \ldots n$ and $j = 1 \ldots . m$. As in [8], we assume that the estimates are provided in the range from 1 to 9 where 1 and 9 are the lowest and the highest degrees of importance respectively. We define $IMPORTANCE$ as $n \times m$ matrix, such that $IMPORTANCE\ [i,j] = Imp(f_i, t_j)$

- Risk of the features: The risk of each feature is estimated. We assume that the risk of a feature is an agreed-up-on estimate provided by the members of the development team. As shown in Chapter 3, $RE(f_i)$ is the function that captures the risk exposure of a feature f_i . We assume that the highest possible degree of risk takes the value 9 while the lowest take the value 1. We define $RISK\_Vector$ as n elements vector, such that $RISK\_Vector\ [i] = RE(f_i)$ and $i = 1 \ldots n$.

- Required and Available Effort: As explained in Chapter 3, $Re\_Effort(f_i)$ is the function that captures the effort of a feature f_i. We assume that the estimate of the required effort of a feature f_i is provided in the form of real numbers, and represents the required person/days to implement that feature. We define $EFFORT\_Vector$ as n elements vector, such that $EFFORT\_Vector\ [i] = Re\_Effort(f_i)$ and $i =$

1.…n. The available effort $Avail\_E$ is captured as a crisp value at the beginning of each release planning. The steps for calculating $Avail\_E$ are stated in Section 3.2.3.

- Dependencies among features: As stated in Section 3.2.3, we consider two types of dependencies: coupling and precedence. If f_i and f_j are coupled, then they should be included in the same release, and if f_i precedes f_j then the former feature will be implemented before the later one. As described in Section 3.2.2, the dependencies among features can be due to managerial or technical issues. Therefore, those dependencies are defined in tandem by release management and the development team. Let D be a n$\times n$ matrix that represents the dependencies among features, such that the element $D[i,j]$ is defined as follows:

$$D[i,j] = \begin{cases} 'c' \quad when \quad (f_i, f_j) \in Coupling \\ 'p' \quad when \quad (f_i, f_j) \in precedence \\ '0' \quad otherwise \end{cases} \tag{4.10}$$

- Commonality of the features: As stated in Section 3.1.2, the commonality of a feature shows the number of tenants that have valid requests for this feature. We define $COMMONALITY$ as an n elements vector that includes the commonality of features. An element i of the vector $COMMONALITY$ is defined as follows $COMMONALITY\ [i]= Augmented\_Com(f_i)$, such that $Augmented\_Com(f_i)$ is the function that calculates the commonality of a feature f_i with taking into account the contractual constraint; i.e. the request of a tenant for adding a feature will not

be considered in the commonality calculation if that tenant is not eligible to have that feature.

4.4.2 Preprocessing

In this process, some of the data structures that resulted from the previous step are augmented in order to prepare them for the next process. As Figure 4.6 shows, the inputs for this stage are:

$IMPORTANCE$ (The importance of features from the perspectives of tenant)

W (The decision weights of tenants)

S (The compliance between the features and the SLA of the tenants)

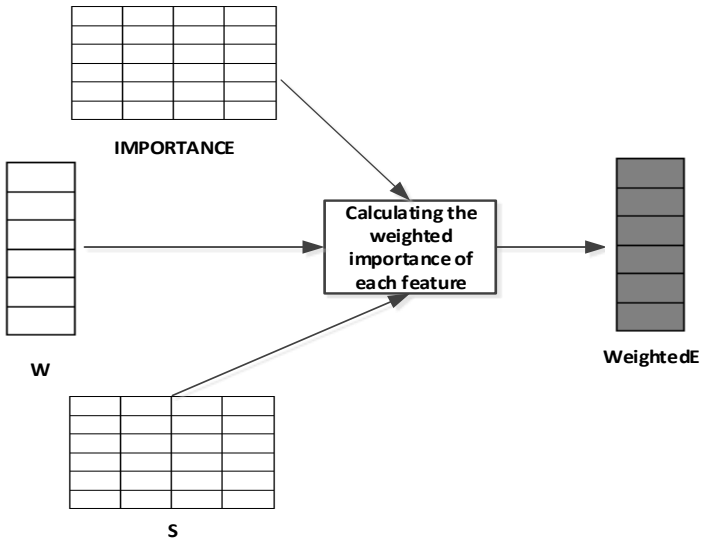


Figure 4.7: The Preprocessing Process

The output of this process is the $WeightedE$ vector. $WeightedE$ contains the importance of features with considering the decision weights of tenants. The following steps are performed at this process:

- **Calculating the weighted importance of each feature**

First, we define $AUGMENTEDIMPORTANCE$ as $n \times m$ matrix that includes the importance of features from the perspectives of tenants while taking into account the service levels of the tenants. An element of this matrix is defined as follows:

$$AUGMENTEDIMPORTANCE[i,\mathrm{j}] = IMPORTANCE[i,j] \times S[i,j] \qquad (4.10)$$

$$i = 1, \ldots . n \; j = 1, \ldots, m$$

As we can see from this Equation, if a features f_i is not complied with the SLA of a tenants t_j, then the estimates of t_j will not be counted for a feature f_i, which means that $AUGMENTEDIMPORTANCE[i,j] = 0$.

Using $AUGMENTEDIMPORTANCE$ and W (the decision weights of the tenants), we can calculate the weighted importance vector. The weighted importance of a feature is a real number that shows the overall importance of that feature by taking into account the decision weights of all tenants. The tenants with higher decision weights have more effect on the value of the weighted importance of a feature. The weighted importance of the features are included in a vector which we call $WeightedE$, such that

$$WeightedE = AUGMENTEDIMPORTANCE \times W \qquad (4.11)$$

($AUGMENTEDIMPORTANCE$ is $n \times m$ matrix and W is $m \times 1$ vector.).

Algorithm 4.1 shows the data collection and the Pre-processing processes.

Analysis of Algorithm 4.1: From the algorithm, it can be observed that the growth rate of running time of Algorithm 3.3 depends on the number of features (n) and the number of tenants m. The worst-case time of Algorithm 4.1 is as follows:

$$Time(n,m) = O(n^2 + n + m + mn) = O(n^2 m)$$

DATA COLLECTION AND PRE-PROCESSING

Input:
1)F* //Candidate features.
2) TenantsProfile // contains the information about the tenants.
Output:
1) D // a matrix containing dependencies among features.
2)WeightedE //n elements vector containing the weighted importance of features.
3)RISK_Vector(i)//n elements vector containing the risk of features.
4)EFFORT_Vector //n elements vector containing the required effort of features.
5)Avail_E // a scalar containing the available effort.
6)COMMONALITY // n elements vector containing the commonality of features.

```
Declare:
1) IMPORTANCE // n*m matrix that captures the importance of the features from the perspective of each tenant.
2)S // n*m matrix containing the compliance between tenants requests and their Service levels.
3)W // m element vector containing the decision weights of tenans.
4)T // a vector containing the tenets IDS.
Avail_E = getAvailableEffort();
n = |F* |; // number of candidate features.
m = |T|; // number of tenants.
T = getTenantsID(TenantsProfile);
for i ← 1 to n do
    for j ← 1 to n do
        if Coupled(fi, fj) then
            D(i, j) =' c';
        else if Precedent (fi,fj) then
            D(i, j) =' p';
for k ← 1 to n do
    COMMONALITY(k) = 0; WeightedE(k) = 0;
for j ← 1 to m do
    W(j) = getTenantweigt(TenantsProfile);
for i ← 1 to n do
    for j ← 1 to m do
        S(i, j) = isComplied(fi, tj); //1 if yes, 0 if No.
        IMPORTANCE(i, j) = Imp(fi, tj); // the importance of
        feature i from the perspective of tenant j.
        AGUMENTEDIMPORTANCE(i, j) =
        IMPORTANCE(i, j) * S(i, j);
        COMMONALITY(i) =
        COMMONALITY(i) + IsRequisted(fi, tj) * S(i, j);
        //IsRequisted(fi,fj)returns 1 if yes, 0 if Not.
        WeightedE(i) =
        WeightedE(i) + AGUMENTEDIMPORTANCE(i, j) * W(j);
    RISK_Vector(i)= Risk(fi); // returns the risk of feature i.
    EFFORT_Vector(i) = Re_Effort(fi);//returns the risk of feature i.
return D, WeightedE, RISK_Vector(i)
,EFFORT_Vector(i),Avail_E ,COMMONALITY
```

Algorithm 4.1

4.4.3 Ranking

The ranking process employs a FIS engine to generate a rank for each feature. As Figure 4.8 shows, the inputs to this process are:

$WeightedE$, $RISK\_Vector$, $EFFORT\_Vector$ and $COMMONALITY$ vectors.

The output is $n \times 3$ matrix, which we call $RANK$.

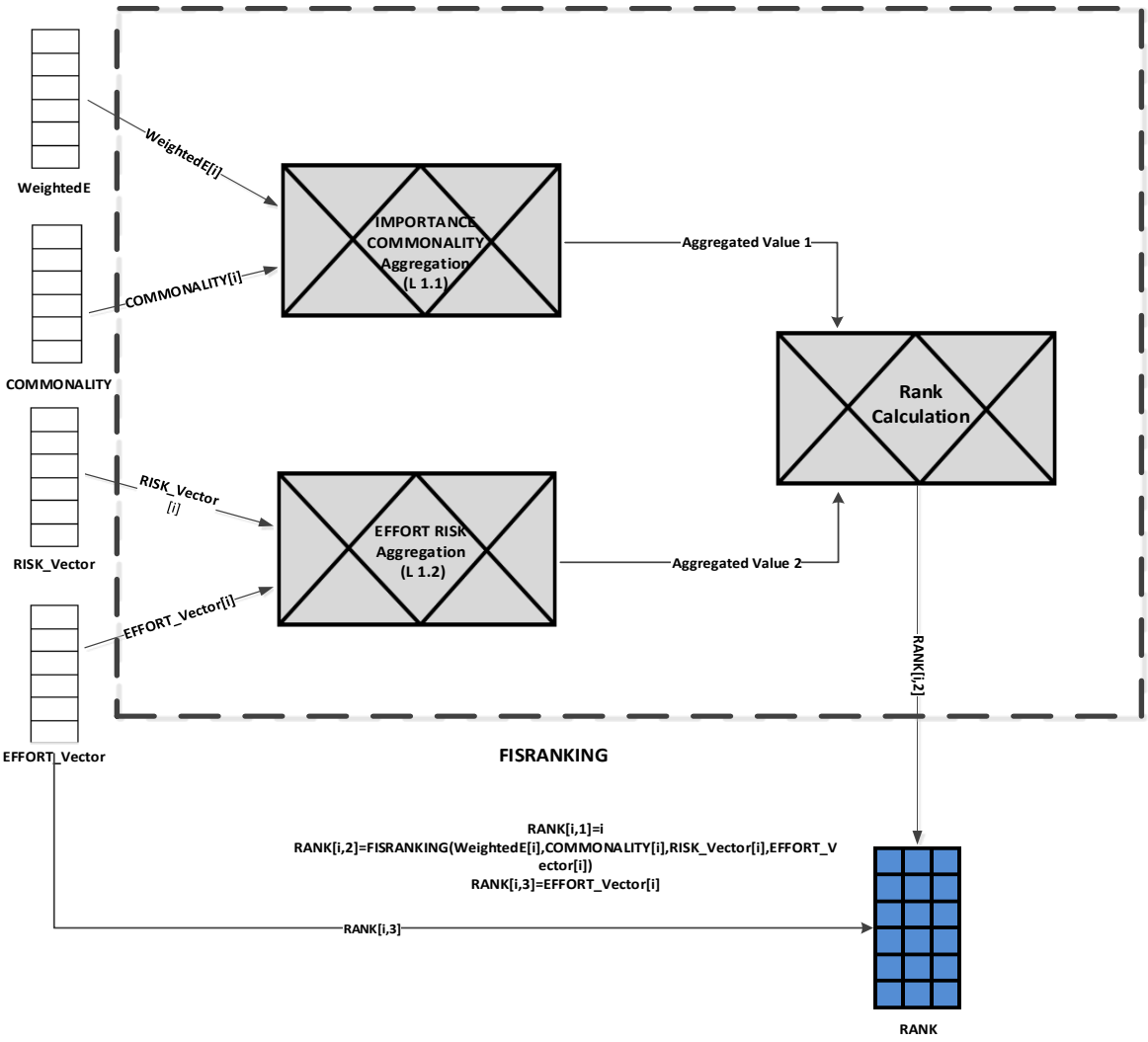


Figure 4.8: Ranking Process

The first column of $RANK$ contains features IDs (we use integer numbers in $1..n$), the second column contains ranks of the features, and the third column contains the required effort of the features. The core component in "Ranking" process is the FIS engine, which we call $FISRANKING$. It can be considered as a function that receives inputs about a feature f_i and generate a rank for that feature. The rank of a feature is a real number that shows the priority of the feature with taking into account the importance of that feature from the perspectives of the tenants, the risk associated with the feature, the required effort to implement the feature, and the commonality of the feature. The fuzzy rules play the major role in determining the rank of features. For example, if the fuzzy rules give the importance of features higher consideration than the risk, then the features with higher importance value will be given higher ranks regardless of the risk. This shows that the perspectives of the designer of the FIS engine is the main player in FIS approach. The elements of $RANK$ is defined as follows:

$RANK[i,1] = i,$

$RANK[i,2] =$

$FISRANKING(WeightedE[i], RISK\_Vector[i], EFFORT\_Vector[i], COMMONALITY[i]),$ and

$RANK[i,3] = EFFORT\_Vector[i].$

The artifices of $FISRANKING$ are defined as follows:

- Four input linguistic variables: $IMPORTANCEVAR, RISKVAR, EFFORTVAR,$ and $COMMONALITYVAR$. Each linguistic variable is associated with an input vector. The linguistic terms of these four variables and their meaning (parameters of the membership functions) are assumed to be identified by the domain experts. In order to deal with the disarmaments that may occur between the experts in defining the

membership functions, we use polling method. Using the polling method guarantees that the beliefs of the experts about the elements of the universe of discourse are interpreted in the membership functions. Polling method is explained in more details in Appendix II.

- IF-THEN fuzzy rules: The rules shall be constructed in a way that increases the final rank of the highly important and high commonality features, and minimizes the rank of high risk and high effort features. The if-then fuzzy rules are in this context are generated by the domain experts. We assume that all disagreements in building IF-then fuzzy rules are resolved, and the provided rules represent the opinion of all experts. When designing fuzzy rules, the likelihood of exponential increase in the total number of rules is a significant issue that should be taken into account. In our case, the maximum number of rules can be

$$\prod_{\forall X \in FISVAR} |X| \qquad (4.10)$$

$$FISVAR = \{IMPORTANCEVAR, RISKVAR, EFFORTVAR, COMMONALITYVAR\}$$

A huge number of rules "may damage the transparency and interpretation of FIS as humans are incapable of understanding and justifying hundreds or thousands of fuzzy rules and parameters" [98]. Therefore, it is necessary to use one of the techniques for rules reduction. In this thesis, we chose the hierarchical fuzzy inference system [99], in which the fuzzy system is built in a hierarchical manner. Hierarchical fuzzy rules have proved to reduce the number of rules without affecting the approximation ability of the FIS [100]. In the hierarchy of fuzzy units, the outputs of lower levels are the inputs for higher levels. There are many

structures of hierarchical fuzzy systems, including incremental, aggregated, and cascaded structures, and combinations of these three. More details about hierarchical fuzzy system are found in [98]. In our proposed model, we aggregate the inputs related to the development team and management using one FIS sub-module, and we aggregate the inputs related to the tenants using another FIS sub-module. After that, the outputs of these two intermediate FIS sub-modules are aggregated using a third FIS sub-module, which generates a rank for the feature under consideration. More precisely, as Figure 4.8 shows, $FISRANKING$ consists of three FIS sub-modules:

- Risk-Effort aggregation (Level 1.1): The input to this module is the risk and effort of a feature f_i.
- Importance-Commonality aggregation (Level 1.2): The input to this module is the weighted importance and the commonality of a feature f_i
- Rank calculation (Level 1.1-Level 1.2 aggregation): the two outputs of previous level are aggregated in order to generate the initial rank for the feature f_i

After applying the ranking process on all the features, we end up with the $RANK$ list, which is the input to the final process, "Release planning generation."

4.4.4 Release Planning Generation

The first step in this process is that the values of the $RANK$ matrix are tuned in order to satisfy the dependencies constraints. For coupling constraints, the ranks of the features that are coupled are adjusted to have the same rank, which means $(f_i, f_j) \in Coupling \rightarrow RANK[i,2] = RANK[j,2]$. Algorithm 4.2 shows how this adjustment is applied.

SATISFYING COUPLING CONSTRAINTS

Input:
1)RANK //n*3 matrix. first column contains the features IDs, the second contains the ranks of features, the third contains the required effort for the features.
2) D // a matrix containing dependencies among features.
Output:
1)RANK // n*3 matrix with adjusted ranks in order to reflect the coupling constraints.

```
1  for i ← 1 to n do
2      SumOfRanks ← RANK(i, 2);
3      CoupledCount ← 1;
4      for k ← 1 to n do
5          if D(RANK(i,1),RANK(k,1))=='c' then
6              SumOfRanks = SumOfRanks + RANK(k, 2);
7              CoupledCount + +;
8      if CoupledCount > 1 then
9          RANK[i, 2] = SumOfRanks/CoupledCount;
10         for k ← 1 to n do
11             if D(RANK(i,1),RANK(k,1))=='c' then
12                 RANK[k, 2] = SumOfRanks/CoupledCount;
13                 D(RANK(k, 1), RANK(i, 1)) =' 0';
14 return RANK;
```

Algorithm 4.2

Analysis of Algorithm 4.2: From the algorithm, it can be observed that the growth rate of running time of Algorithm 4.2 depends on the number of features (n). The worst-case time of Algorithm 4.2 is as follows: $Time(n) = O(2n^2) = O(n^2)$.

In order to satisfy the precedence constraints, it is required to adjust the ranks in a way that makes the superior features have higher ranks than dependent features, which means $(f_i, f_j) \in Precedence \rightarrow RANK[i, 2] \geq RANK[j, 2]$. This guarantees that f_j is assigned to the next release if and only if f_i is already assigned. Algorithm 4.3 shows how to adjust the ranks of the features in order to satisfy the precedence constraints.

SATISFYING PRECEDENCE CONSTRAINTS

Input:
1)RANK //n*3 matrix. first column contains the features IDs, the second contains the ranks of features, the third contains the required effort for the features.
2) D // a matrix containing dependencies among features.
Output:
1)RANK // n*3 matrix with adjusted ranks in order to reflect the coupling constraints.

```
1 RANK = Sort(RANK, 2, descend); // Sort according to second column
2 for i ← 1 to n do
3     for k ← 1 to n do
4         ϵ ← FindMinDifference(RANK(:, 2)); // finding the minimum
          different between the ranks of features
5         if D(RANK(i,1),RANK(k,1))=='p' and
          D(RANK(i,2)<RANK(k,2) then
6             RANK(i, 2) = RANK(j, 2) + ϵ;
7     RANK = Sort(RANK, 2, descend);
8 return RANK;
```

Algorithm 4.3

Analysis[5] of Algorithm 4.3: From the algorithm, it can be observed that the growth rate of running time of Algorithm 4.3 depends on the number of features (n). The worst-case time of Algorithm 4.3 is as follows:

$Time(n) = O\big(\, n(n\,(n)\, + nlogn)\big) = O(n^3 + n^2 logn) = \; O(n^3)$.

After that, a greedy approach is applied; that is, the features with the highest rank are added to the release plan. This process is continued for as long as the available effort is not

[5] $FindMinDifference$ is a function that finds the minimum difference between any two elements of a vector. The complexity of this function is $O(n)$

exceeded. In algorithmic form, the release planning generation process is shown in Algorithm 4.4.

RELEASE PLAN GENERATION

Input:
1) D // a matrix containing dependencies among features.
2)WeightedE //n elements vector containing the weighted importance of features.
3)RISK_Vector//n elements vector containing the risk of features.
4)EFFORT_Vector //n elements vector containing the required effort of features.
5)Avail_E // a scalar containing the available effort.
6)COMMONALITY // n elements vector containing the commonality of features.
Output: F^

```
1  for i ← 1 to n do
2      RANK(i,1) = i;
3      RANK(i,2) = FISRANKING(WeightedE(i), COMMONALITY(i), EFFORT_Vector(i), RISK_Vector(i)); // FIS engine to generate ranks for the features
4      RANK(i,3) = EFFORT_Vector(i);
5  RANK = SatisfyingCouplingConstraints(RANK, D);
   RANK = SatisfyingPrecedenceConstraints(RANK, D);
   Sort(RANK, descending, 2); //RANK is sorted according to the rank of features (second column)
6  k ← 1
7  while Avail_E>0 and k < n do
8      while Avail_E − RANK[k,3] < 0 and k < n do
9          k++;
10     Add(RANK(k,1), F^);//add feature id to the release plan list
11     Avail_E = Avail_E − RANK[k,3];
12 return F^ // The plan for the next release
```

Algorithm 4.4

Complexity of Algorithm 4.4: From the algorithm, it can be observed that the growth rate of running time of Algorithm 4.3 depends on the number of features (n). The worst-case time of Algorithm 4.4 is as follows:

$$Time(n) = O(n + n^2 + n^3 + nlogn + n^2) = O(n^3)$$

In the next section, the applicability of the proposed approach is illustrated, using a proof-of-concept example.

4.5 Proof of Concept

Assume we have 20 features that are requested by five tenants. The release management estimates the available effort as equal to 40 person-days. In order to use the proposed FIS-based release planning approach to build the plan for the next release, we apply the following stages:

1) Raw data collections

Using the profiles of the tenants, the release management calculates the decision weights $W = [0.1421 \;\; 0.1243 \;\; 0.2389 \;\; 0.2027 \;\; 0.2919]$. For each feature, the release management specifies the tenants who asked for that feature. The first part of Table 4.1 shows the features that are requested by each tenant. Each column j in this part of the table represents the characteristic function of the set F_{tj} (the set that represents the features that are requested by tenant t_i). If f_{i} is requested by t_{j} then the cell $i, j = 1$; otherwise, it equals0. Using the profiles of the tenants, the release management determines the validity of tenants' requests. The second part of Table 4.1 shows the compliance of the SLA of each tenant with the features (matrix S). Using the second and third steps, the commonality of each feature is calculated ($COMMONALITY$) which shown in the third part of Table 4.1. The development team and release management specify the dependencies among features. In this example, we assume that the dependencies are defined as follows: $Precedence = \{(3,12), (2,3), (5,6), (7,20)\}$

$$Coupling = \{(1,2)\}$$

Each tenant provides his estimates about each feature. Also, the development team provides the estimates about the required effort and the risk. The first part of Table 4.2 shows the $IMPORTANCE$ matrix. The second and third parts of table 4.2 show the $EFFORT\_Vector$ and the $RISK\_Vector$

Table 4.1: The Features List Requested by Each Tenant, the Compliance of SLA of Each Tenant with the Features, and the Commonality of Features

	The Lists of Features Requisted by Tenants					S					
	Ft1	Ft2	Ft3	Ft4	Ft5	SLA of t1	SLA of t2	SLA of t3	SLA of t4	SLA of t5	COMMONALITY
f1	1	1	1	1	1	1	1	1	1	0	4
f2	1	0	1	1	0	1	1	1	1	1	3
f3	1	0	0	0	1	1	1	1	1	1	2
f4	0	1	1	1	1	1	1	1	1	0	3
f5	1	1	1	1	1	0	1	1	1	1	4
f6	1	1	1	1	1	1	1	1	1	1	5
f7	1	1	1	0	1	1	1	1	1	1	4
f8	1	1	0	1	1	1	1	1	1	1	4
f9	1	1	1	1	1	1	1	1	0	1	4
f10	1	1	1	1	1	1	1	1	1	1	5
f11	1	1	1	0	1	1	1	1	1	1	4
f12	1	1	1	0	1	1	1	1	1	1	4
f13	1	1	1	1	1	1	1	1	1	1	5
f14	0	0	1	0	1	1	1	1	1	1	2
f15	1	1	1	0	1	1	1	1	1	1	4
f16	1	1	0	1	1	1	1	0	1	1	4
f17	1	1	1	1	0	1	1	1	1	1	4
f18	0	1	1	0	1	1	1	1	0	1	3
f19	0	0	1	1	1	1	1	1	1	1	3
f20	1	1	1	1	0	1	1	0	1	1	3

Table 4.2: The Estimates of the Importance, Risk, and Required Effort of the Features

	IMPORTANCE					EFFORT_Vector	RISK_Vector
f1	6	5	8	7	7	9	4
f2	2	2	4	3	5	4	8
f3	4	2	4	6	7	1	9
f4	7	2	5	2	2	3	7
f5	9	6	1	4	1	7	1
f6	2	8	5	7	5	3	3
f7	7	2	5	8	9	1	7
f8	2	5	5	8	2	5	5
f9	7	5	6	5	5	7	9
f10	9	9	9	8	8	5	4
f11	3	7	6	4	3	7	1
f12	2	5	4	5	8	8	9
f13	9	5	1	2	4	2	3
f14	9	3	3	8	1	7	9
f15	9	1	6	2	3	1	9
f16	6	5	4	4	2	5	1
f17	4	7	5	9	4	6	1
f18	1	6	5	8	8	4	7
f19	6	8	7	1	8	5	4
f20	4	6	2	9	2	2	1

2) Preprocessing

In this stage, we generate the $WeightedE$, which the vector that contains the weighted importance of features. In order to generate $WeightedE$ vector, the following steps are applied:

I) Generating $AUGMENTEDIMPORTANCE$ matrix by applying element wise multiplication between $IMMPORTANCE$ and S matrix. The first part of Table 4.3 shows the $AUGMENTEDIMPORTANCE$.

II) Generating $WeightedE$ by multiplying $AUGMENTEDIMPORTANCE$ by W (the decision weights of the tenants). The second part of Table 4.3 shows the $WeightedE$ in this example.

Table 4.3: AUGMENTEDIMPORTANCE Matrix, and WeightedE Vectors

	AUGMENTEDIMPORTANCE					WeightedE
f1	6	5	8	7	0	4.80
f2	2	2	4	3	5	3.56
f3	4	2	4	6	7	5.03
f4	7	2	5	2	0	2.84
f5	0	6	1	4	1	2.09
f6	2	8	5	7	5	5.35
f7	7	2	5	8	9	6.69
f8	2	5	5	8	2	4.31
f9	7	5	6	0	5	4.51
f10	9	9	9	8	8	8.51
f11	3	7	6	4	3	4.42
f12	2	5	4	5	8	5.21
f13	9	5	1	2	4	3.71
f14	9	3	3	8	1	4.28
f15	9	1	6	2	3	4.12
f16	6	5	0	4	2	2.87
f17	4	7	5	9	4	5.63
f18	1	6	5	0	8	4.42
f19	6	8	7	1	8	6.06
f20	4	6	0	9	2	3.72

3) Ranking

The ranking process is a structured hierarchical FIS engine. As Figure 4.8 shows, this FIS engine ($FISRANKING$) is composed of three modules. We use Matlab Fuzzy Logic Toolbox to build $FISRANKING$. Four experts participate in defining the membership functions of the input and output variables. Appendix III shows the data collected from these experts using polling method. Appendix IV shows the IF-then fuzzy rules used in these fuzzy modules. The output of Ranking process is $RANK$ list as shown in table 4.4. The features are sorted according to their ranks.

Table 4.4: The Output of Ranking Process

(RANK List before Applying Dependencies Constraints)

Rank		
Feature ID	Required Effort	rank
13	2	92.7176
6	3	89.1440
17	6	86.6403
20	2	81.5683
10	5	79.2165
7	1	77.7665
8	5	75.2500
11	7	71.2137
15	1	70.4289
16	5	68.2611
19	5	66.8848
1	9	62.8750
3	1	58.5445
4	3	55.6125
5	7	50.5000
18	4	44.4993
2	4	38.1243
12	8	33.8518
9	7	29.3239
14	7	25.7498

4) Release Planning Generation

During this stage, first algorithms 4.2 and 4.3 are used in order to tune the ranks of features to reflect the dependencies constraints. Table 4.5 shows the $RANK$ list after tuning the ranks of features. The last step is to apply algorithm 4.4 to select the features that will be implemented in the next release. In our example, those features are:

$$F^{\wedge} = \{f_{13}, f_5, f_6, f_{17}, f_7, f_{20}, f_{10}, f_8, f_{11}, f_{15}\}$$

Table 4.5: RANK List after Applying Dependencies Constraints

Rank		
Feature ID	Required Effort	rank
13	2	92.7176
5	7	89.1450
6	3	89.1440
17	6	86.6403
7	1	81.5693
20	2	81.5683
10	5	79.2165
8	5	75.2500
11	7	71.2137
15	1	70.4289
16	5	68.2611
19	5	66.8848
4	3	55.6125
1	9	53.1667
2	4	53.1667
3	1	53.1667
18	4	44.4993
12	8	33.8518
9	7	29.3239
14	7	25.7498

4.6 Summary of the Chapter

Release planning cannot be performed in isolation from human influences. Because of the human factors, release planning can be considered as an under-uncertainty decision-making problem. Therefore, human expertise must be taken into consideration while at the same time dealing with uncertainty. This chapter proposes a FIS-based approach that automatically incorporates human expertise with the computational solutions, and considers the uncertainty factors. The proposed approach is composed of four main processes: collection of raw data, preprocessing, ranking, and release planning generation. The dependencies among features are considered by adjusting the ranks of features; such that, the coupled features have the same rank and, in the precedence constraints, the superior features have higher ranks than dependent features. The effort constraint is taken into account by applying a greedy approach when features are assigned to a release plan; such that, the features list are sorted according to their ranks and then the features are assigned to the release plan until the total effort of the assigned features are equal to the available effort.

CHAPTER 5: OPTIMIZATION BASED APPROACHES (BLP and GA)

5.1 Introduction

This chapter proposes two optimization approaches for generating the next release plan for multi-tenant SaaS applications. The first one is a BLP-based approach. In many earlier studies about release planning, integer linear programming (ILP) has been applied to solve release-planning problems [33, 38, 101]. BLP is a special case of ILP, where the variable takes only binary values. BLP [102] can be used in selection problems, where the decision makers have many alternatives and they want to eliminate the inappropriate ones. BLP can also be used in yes/no problems, where the solution is a set of selected choices. Since we are planning for only the next release, we can make release planning more specific by restricting the decision variable on the set {0, 1}. Therefore, a BLP approach is suited to planning the next release in SaaS applications, where the SaaS provider needs to select a subset of features from among all those requested. However, it can be very expensive to find optimal solution using BLP; especially, when there is huge number of variables. Therefore, in the third approach we use GA, which is a heuristic optimization technique, in order to reduce the cost of release planning process. GA has been used in many studies to generate software release plans [8, 32, 37, 44]. The proposed optimization approaches help the release management to plan the next release in a way that maximizes tenants' satisfaction and release commonality, and minimizes the risk of having insecure and low-quality releases. In addition to those objectives, the proposed approaches takes into account the effort, the compliance of tenants requests with their SLA levels, and the dependencies constraints.

This chapter is organized as follows: Section 5.2 presents the general BLP model. Section 5.3 introduces the BLP-based approach. Section 5.4 presents the GA-based approach. Section 5.5 shows applicability of the proposed approaches using a proof-of-concept example. Section 5.6 presents the summary of the chapter.

5.2 Binary Linear Programming

Let $f(x)$: $\{0,1\}^n \rightarrow \mathbb{R}$ be the objective function that we want to minimize. Then the standard form of binary linear programming (BLP) problems is as follows:

$Min\ f(x_{1..n})$ such that

$$f(x_{1..n}) = \sum_{i=1}^{n} c_i\, x_i \qquad (5.1)$$

subject to :

1) p inequality constraints

$$h_j(x_{1..n}) = \sum_{i=1}^{n} a_{ij} x_i \geq b_j \qquad j = 1, \ldots . p \qquad (5.2)$$

2) k equality constraints

$$g_j(x_{1..n}) = \sum_{i=1}^{n} a_{ij} x_i = b_j \qquad j = 1, \ldots . k \qquad (5.3)$$

3) $x_i \in \{0,1\}, i = 1, \ldots . n.$

If we want to maximize the objective function then we rewrite the objective function as

$$f(x_{1..n}) = \sum_{i=1}^{n} - c_i\, x_i \qquad (5.4)$$

In order to simplify the inequality constraints we convert them to equality constraints by adding a slack variable z_i, such that the p inequality constraints can be rewritten as

$$h_j(x) = \sum_{i=1}^{n} a_{ij}x_{ij} + z_i = b_i \qquad j = 1, \ldots . p \qquad (5.5)$$

The objective function and equality and inequality constraints can be represented using matrices and vectors as follows:

$$f(X) = C\,X \qquad (5.6)$$

such that $X = [x_1\ x_{2,} \ldots, x_n]$, $C = [c_1\ c_2\ \ldots, c_n]$, C is $1 \times n$ raw vector and X is $n \times 1$ column vector.

subject to:

1) q inequality constraints, which can be written as $AX = b$, where A is $q \times n$ matrix, X is $n \times 1$ column vector, and b is a $q \times 1$ column vector. Note that q is the number of constraints and n is the number of variables.
2) k equality constraints, which can be written as $AeqX = beq$, where Aeq is $k \times n$ matrix, X is $n \times 1$ column vector, and beq is a $k \times 1$ column vector. Note that k is the number of constraints and n is the number of variables.

In BLP, one possible way to find the optimal solution is to enumerate all possible solutions and then to choose the one that is optimal in maximizing (or minimizing) the objective function. The problem with this strategy is that the number of possible solutions increases exponentially with the increasing of number of variables. If there are n variables, then there are 2^n possible solutions. The branch-and-bound algorithm addresses this problem [102]. Branch and bound is a "divide and conquer" strategy, which divides feasible regions into smaller, controllable regions. These new regions are divided recursively into smaller regions until the optimal solution is attained. This algorithm is described in more details Appendix I.

5.3 The Proposed BLP-based Approach

The proposed BLP-based approach consists of three processes: raw data collection, preprocessing, and release plan generation. The output of a process is considered as the input of the next process. The raw data collection stage is the same as is defined in Section 4.3.1. The preprocessing is the same as is defined in Section 4.3.2. The output of these two processes: $WeightedE$, $RISK\_Vector$, $EFFORT\_Vector$, $COMMONALITY$, and D (which is the matrix that has the dependency among features).

5.3.1 Release plan generation

The aim of release-plan generation is to optimize feature selection. The process aims to select the features that maximize tenants' satisfaction and commonality and minimize the risk, while taking into account the effort, the compliance of tenants' requests with their SLA levels, and dependencies constraints. The output of release plan generation process is the desired release plan which is represented as a vector of decision variables $X = [x_1\, x_2 \ldots, x_n]$, where $x_i \in \{0,1\}$. If $x_i = 1$, then the feature f_i is assigned to the next release; otherwise, it is postponed to a future release. This is equivalent to $x_i = G_{F^\wedge}(f_i)$, where $G_{F^\wedge}(f_i)$ the characteristic function of the set is $F^\wedge$ ($F^\wedge$ represents the release plan for the next release (see Chapter 3)). Depending on these decision variables, we can define the BLP model for release planning in SaaS as follows:

$$maximize\ O(X)$$

such that

$$O(X) = \sum_{i=1}^{n} \omega \left(\frac{WeightedE[i] \times COMMONALTY[i]}{RISK\_Vector[i]} \right) \times X[i] \qquad (5.7)$$

subject to

$$\left(\sum_{i=1}^{n} X[i] \times EFFORT\_Vector[i]\right) \leq Avail\_E \qquad (5.8)$$

$$X[i] - X[j] = 0 \; if \; D(i,j) =' c' \qquad (5.9)$$

$$X[i] - X[j] \geq 0 \; if \; D(i,j) =' p' \qquad (5.10)$$

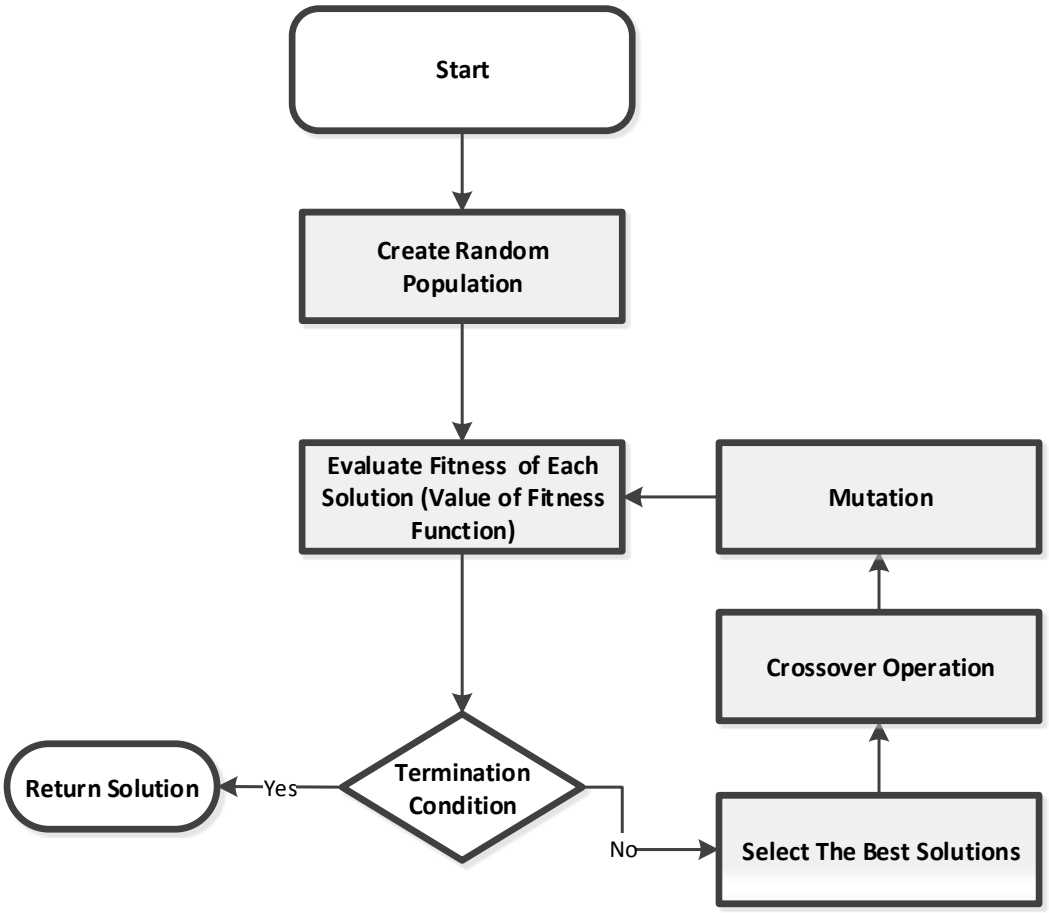
The objective function (Equation 5.7) shows that element-wise multiplication is applied on $WeightedE$ and $COMMONALTY$ vectors, which means that increasing the value of [6]$WeightedE[i]$ or $COMMONALTY[i]$ will increase the chance that a feature f_i will be included in the next release. Also, we can see that we divide the result of this multiplication by $RISK\_Vector[i]$, which means that increasing the $RISK\_Vector[i]$ will decrease the chance that f_i will be included in the next release; in other words, the risky requirements should have a lower chance of being included in the next release. ω is a scale factor. If $\omega > 1$, then more emphasis is given to the importance and commonality of features. If $\omega < 1$, then more emphasis is given to risk. Unless stated to the contrary, we assume that $\omega = 1$. The effort and dependencies constraints are dealt with using equality and inequality constraints. Equation (5.8) represents the effort constraint. The total required effort for the selected features (which means the associated decision variables are set to 1) is less or equal to the available effort ($Avail\_E$). Equations (5.10,5.11) represent the dependencies constraint. If $D(i,j) =' c'$ then f_i and f_j shall be included in the same release which means $X[i] = X[j]$ ($X[i], X[j]$ are the decision variables associated with the features f_i , f_j respectively). Also if $D(i,j) =' p'$ then f_i shall precede f_j, which means $X[i] \geq X[j]$.

[6] [i] denotes the element i of an array

5.4 Genetic Algorithm-based Approach

Genetic algorithms emulate the process of biological evolution in the natural life [43,]. The theory of biological evolution state that the population of species is improved during time. The new generations are an improvement of previous ones. From a population, only fit (healthy) organism can survive and participates in producing future generations. The offspring have mixed traits from the parent (previous generation) by the operations of crossover of parent's chromosomes. Each chromosome consists of gens and each gen contains a trait. Genetic Algorithm (GA) is population-based search algorithm that search for the optimal or near optimal solutions by producing a sequence of generations [42]. Figure 5.2 shows the steps of GA. At the beginning of GA, random population is generated. Two factors shall be balanced when determining the population size: 1) obtaining high quality solutions (more optimality), which is increased when the population size is increased, and 2) the computation time, which is less when low population size is used. In this context, each individual in the populations is represented using n binary numbers, where n refers to the number of variables (the length of $WeightedE$ vector). We use the default population size in Matlab (200 individuals). In the second step of GA, each solution in the population is evaluated using the fitness function (the objective function). If the optimal solution is found, the algorithm ends and returns the solution; otherwise, the fit solutions (the ones that have achieved the highest values of the fitness function) are selected for the reproduction process. Reproduction process involves two operations: crossover, and mutation. Crossover is the process of selecting two parents and combines the gens of these two parents. There are three basic types of crossover: one point, two points, and uniform crossovers. More details about crossover found in[104]. For examples if $chromosome\ 1 =$

100 001 and $chromosome\ 2 = 111\ 000$ then the result of one point crossover can be 111001. Mutation is the process of altering some gens values in order to avoiding trapping at a solution in the local optima. For example, the result of previous crossover operation can be changed to 111101. Mutation is performed according to mutation probability. The value of mutation probability depends on the type of the problem. The algorithm keeps running until the termination condition is reached. The termination condition can be related to predefined elapsed time, predefined number of iterations, or when there is no improvement in the produced solutions.



5.1: The Steps of Genetic Algorithm

In the case of the problems that restrict variables to integers, additional operations are added to the traditional GA algorithm. According to [104], the following operations are added:

- After crossover and mutation operations, the real values of the produced solution are truncated to integers.
- The fitness function is calculated by adding the penalties of the constraint violations to the objective function if the solution is in the invisible regions; otherwise, the value of fitness function is set to the value of the objective function.
- Equality constraints must be transformed to inequality constraints. Each equality constraint is replaced with two inequality constraints. For example, $x_1 + x_2 = 10$ is replaced with these two constraints: $x_1 + x_2 \geq 10$ and $x_1 + x_2 \leq 10$.

As in the BLP-based approach, the proposed GA-based approach for release planning consists of three processes: raw data collection, preprocessing, and release plan generation. The raw data collection and the preprocessing stages are the same defined in Sections 4.3.1 and 4.3.2. In the release plan generation process, we use the objective function defined by equation (5.7) and constraints defined by equations (5.8, 5.9, and 5.10) in order to evaluate the fitness of solutions.

In the next section, we present a proof-of-concept example to show the applicability of the BLP-based and GA-based approaches.

5.5 Proof of Concept

Recall the example that is used in Section 4.5. In it, 20 features are requested by five tenants whose decision weights are define as $W = [0.14\ \ 0.12\ \ 0.24\ \ 0.2027\ \ 0.3\ ]$, and the available effort is 40 person-days. The dependencies among features are defined by these two set $Precedence = \{(3,12),(2,3),(5,6),(7,20)\}$ and $Coupling = \{(1,2)\}$. As stated above, the raw data collection and preprocessing are the same as the one defined for the FIS-based approach. Table 6.1 shows the $RISK\_Vector$, $EFFORT\_Vector$, $WeightedE$, and $COMMONALITY$ vectors.

5.5.1 BLP-based Approach

In the BLP approach, the coupling relationship is represented using equality constraints that can be written as $Aeq * X = beq$ (matrix Aeq and vector beq represent the equality constraints). Since we have only one coupling relationship, the dimension of Aeq is 1×20 and beq is one element vector, which means $Aeq = [1\ -1\ 0 \dots \dots \dots 0]$ and $beq = [0\ ]$. The precedence relationship and the effort constraint are represented using inequality constraints that can be written as $AX = b$ (matrix A and vector b represent the inequality constraints). The dimension of A is 5×20 and b is a 5-elements vector since we have four precedence constraints and one effort constraint.

$$A = \begin{bmatrix} 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ -1\ 0 \dots \dots \dots \dots \dots 0 \\ 0\ 1\ -1\ 0\ 0\ 0\ 0\ 0\ 0 \dots \dots \dots \dots \dots \dots \dots \dots \dots 0 \\ 0\ 0\ 0\ 0\ 1\ -1\ 0\ 0\ 0\ 0 \dots \dots \dots \dots \dots \dots \dots \dots 0 \\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0 \dots \dots \dots \dots \dots \dots \dots \dots \dots -1 \\ 1\ 1\ 1\ 1 \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots 1 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -40 \end{bmatrix}$$

We use $bintprog$, which is a Matlab function that solves binary integer programming problems. $bintprog$ is called in the following form:

$$X = bintprog\ (-Rank, A, b, Aeq, beq)$$

such that $Rank = WeightedE .* COMMONALITY ./ RISK\_Vector .* X$, $(.*)$ and $(./)$ are element-wise multiplication and division operations. Note that $bintprog$ has been built to solve minimization problems; hence, we multiply $Rank$ by -1 in order to make the problem a maximization problem.

Table 5.1: EFFORT_Vector, RISK_Vector, WeightedE, and Commonality Vectors

	EFFORT_Vector	RISK_Vector	WeightedE	COMMONALITY
f1	9	4	4.80	4
f2	4	8	3.56	3
f3	1	9	5.03	2
f4	3	7	2.84	3
f5	7	1	2.09	4
f6	3	3	5.35	5
f7	1	7	6.69	4
f8	5	5	4.31	4
f9	7	9	4.51	4
f10	5	4	8.51	5
f11	7	1	4.42	4
f12	8	9	5.21	4
f13	2	3	3.71	5
f14	7	9	4.28	2
f15	1	9	4.12	4
f16	5	1	2.87	4
f17	6	1	5.63	4
f18	4	7	4.42	3
f19	5	4	6.06	3
f20	2	1	3.72	3

Vector X contains binary values that represent the release plan. If $X[i]$ is equal to 0, then f_i will be assigned to the next release; otherwise, it will be assigned to future releases. In this example, the output of $bintprog$ is $X = [0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1]$, which means that the features that are assigned to the next release are:

$F^{\wedge} = \{ f_5, f_6, f_7, f_{10}, f_{11}, f_{13}, f_{15}, f_{16}, f_{17}, f_{20}\}$.

5.5.2 GA-based Approach

The same data in Table 5.1 is used to run GA approach. The same objective function is used. As stated in Section 5.4, there are no equality constraints in GA with integer variables. Hence, the coupling relationship is represented using two inequality constraints. Depending on that, the dimension of A becomes 7×20 and b becomes 7-elements vector since we have four precedence constraints, one effort constraint, and two more constraints for coupling. We use ga, which is a Matlab function for genetic algorithm. ga is called in the following form:

$$x = ga(ObjectiveFunction, nvars, A, b, LB, UB, intr, option)$$

such that $ObjectiveFunction$ is a handle to the fitness function, $nvars$ is the number of variable which in this example 20, A and b represent inequality constraints, LB and UB are vectors that includes the lower and higher bounds of the variables, which are in our problem 0 and 1, $intr$ is a vector contains the indices of the integer variables, which are in our problem all the variables. $Option$ is a structure that defines the parameters of the ga function. We use the default parameters for the ga function. The output of ga function is $x = [0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1]$, which means that the features that are assigned to the next release are: $F^{\wedge} = \{ f_5, f_6, f_7, f_{10}, f_{11}, f_{13}, f_{15}, f_{16}, f_{17}, f_{20}\}$.

5.6 Summary of the Chapter

Release planning is an optimization problem. We propose a BLP-based and a GA-based approach in order to generate a plan for the next release in a way that maximizes tenants' satisfaction and release commonality, and minimize the risk of having insecure, low quality, and over-time releases. In addition to those objectives, the proposed approach takes

into account the effort, the compliance of tenants' requests with their SLA levels, and dependencies constraints. Bothe of the proposed approaches consists of three processes: raw data collection, preprocessing, and release planning generation. The generated release plans by both approaches are represented using vectors of binary values where each element of a vector represents a feature. If an element of this vector is set to 1 then the corresponding feature is selected, and if it is set to 0 then the corresponding feature is not selected.

In the next chapter, we compare FIS-based and BLP-based approaches. Four measurements are used to compare the three approaches: degree of tenants' satisfaction, degree of commonality, adherence to the risk factor, and growth in running time.

CHAPTER 6: EXPERIMENTAL COMPARISON OF THE PROPOSED APPROACHES

6.1 Introduction

This chapter evaluates and compares the proposed FIS-based, BLP-based, and GA-based approaches. The experiments presented in this chapter use different scenarios in order to compare the effectiveness of the three approaches from the perspective of the degree of overall tenants' satisfaction, the degree of commonality, the degree of the adherence to the risk factor, and the running time required to generate release plans. We want to figure out the situations in which each approach is suitable to be used. Since we could not find enough suitable real data for the experiments, we use synthetic data that are generated using the potential probability distributions.

This chapter is organized as follows: Section 6.2 provides statistical analysis of the data of some previous release planning approaches. Section 6.3 presents the results of a comparative study among the three approaches in different scenarios. Section 6.4 discusses the similarity between the release plans that are produced by the three approaches. Section 6.5 presents the results of a comparison between the proposed approaches and the release planning approach presented in [33]. Section 6.6 presents the summary of the chapter.

6.2 The Probability Distributions of Release Planning Data

The aim of this process is to find the distribution models for the data collected from tenants about the importance of features. This helps us to generate more general cases simulated data for validating the proposed approaches. Four datasets are used from different resources. Table 6.1 shows the description of these datasets.

Table 6.1: The Description of Dataset Samples

	Sample Size	Source	Data Type	max	min	mean	Standard deviation
DS1	149	[57]	integer	9	1	5.328	2.5
DS2	80	[105]	Integer	9	1	4.96	2.48
DS3	112	[57]	integer	9	1	4.96	2.64
DS4	80	[57]	integer	9	1	4.8	2.5

Chi-square test is used to compare the hypothesis distribution models with the empirical distribution (obtained from data samples). We test three null hypotheses: the data about importance of features can be generated using 1) discrete uniform, 2) Poisson, or 3) normal distributions. Chi-square test can be described as follows: Assume that we have an experiment that has m possible outcomes that can be categorized to n categories. Let O_i be the observed frequencies in the data category i, E_i is the expected frequency in the data category , and $i = 1 \ldots n$, then if $\chi^2 \leq \chi_\alpha^2$ then the null hypothesis is correct; otherwise, it is rejected, where:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \qquad (6.1)$$

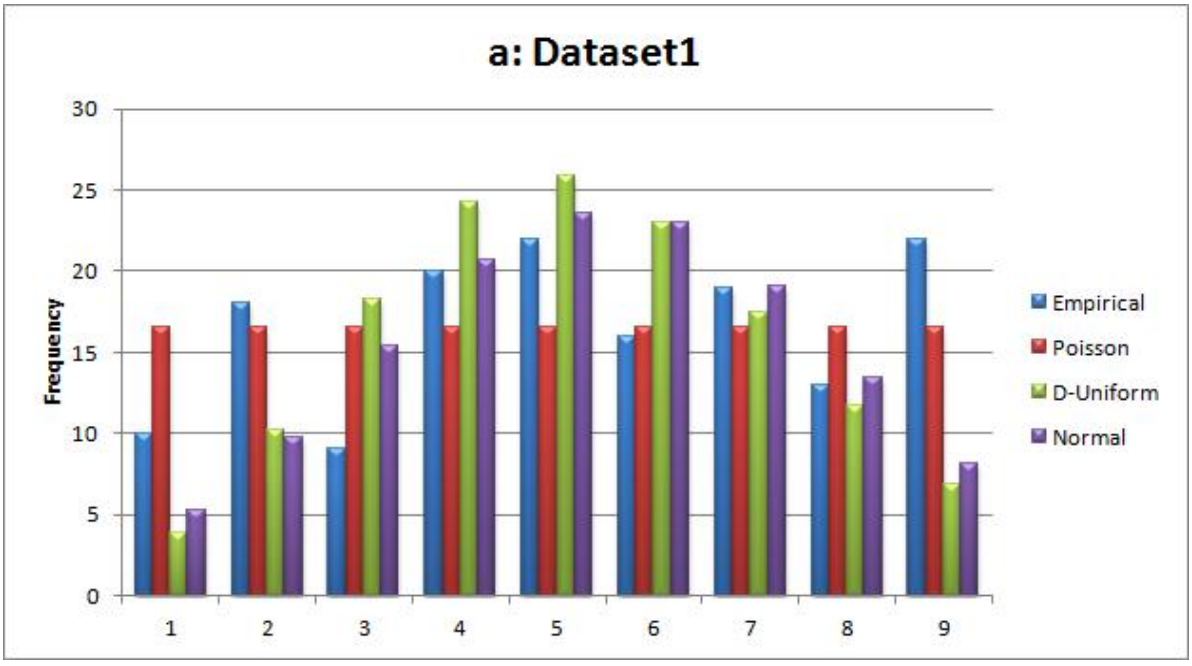
and χ_α^2 can be obtained from Chi-Square distribution table. α denotes the significance level. In this test, we use significance level of 0.05. Table 2 shows the Chi-square test results for the test of the three hypotheses. It is clear that the first null hypothesis is accepted for all datasets, while the remaining hypotheses are rejected. This confirms that discrete uniform

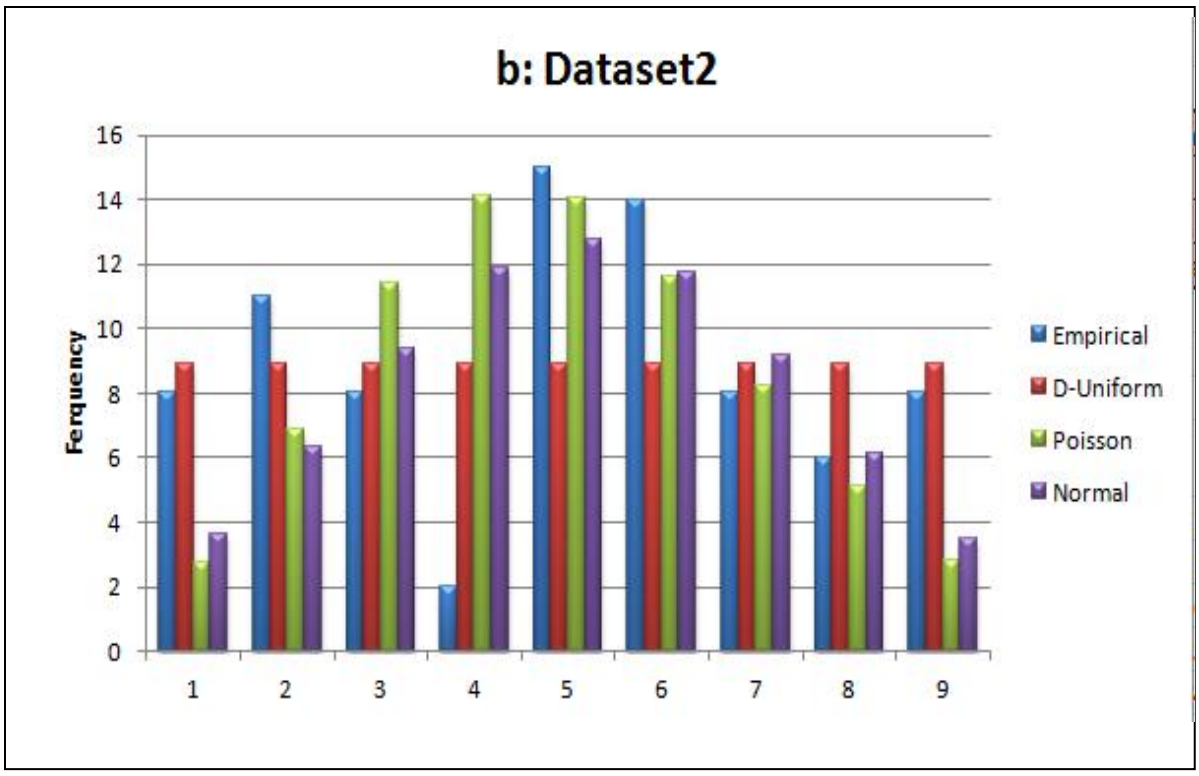
distribution can be used to generate the data about the importance of features. Figure 6.1 (a,b,c,d) shows the empirical, discrete uniform, Poisson, and normal distributions.

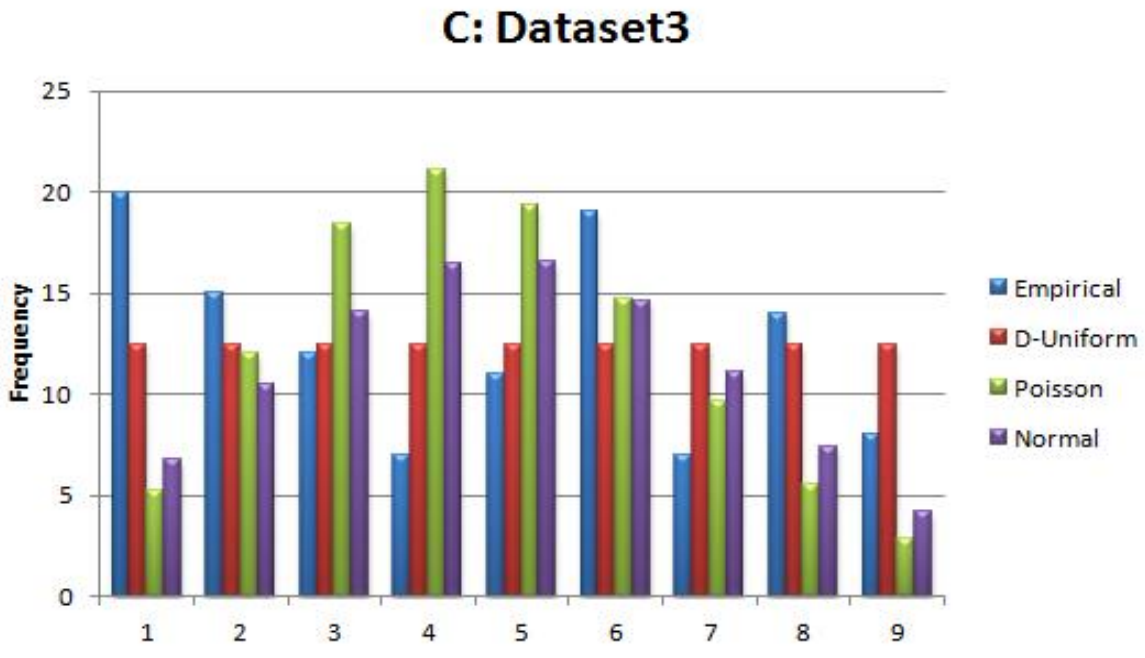
Table 6.2: Chi-square Tests for four Dataset Samples

	χ^2 (D-uniform)	χ^2 (Poisson)	χ^2 (Normal)	$\chi^2_{0.05}$
Dataset1	11.61077	57.11578283	39.9708497	15.507
Dataset2	14.28	33.98434777	24.00955583	15.507
Dataset3	14.19465	81.822	47.46957	15.507
Dataset4	15.17652	30.90713	26.38032	15.507

In order to generate the data for testing the proposed release planning approaches, a discrete uniform distribution-based random number generator is used to produce the data about the importance of features. Matlab *unidrnd* function is used. Because we do not have enough sample data for the other variables, normal distribution-based random number generator is used as in [106].




b: Dataset2
Ferquency
16
14
12
10
8
6
4
2
0
1
2
3
4
5
6
7
8
9
Empirical
D-Uniform
Poisson
Normal


C: Dataset3
Frequency
25
20
15
10
5
0
1
2
3
4
5
6
7
8
9
Empirical
D-Uniform
Poisson
Normal

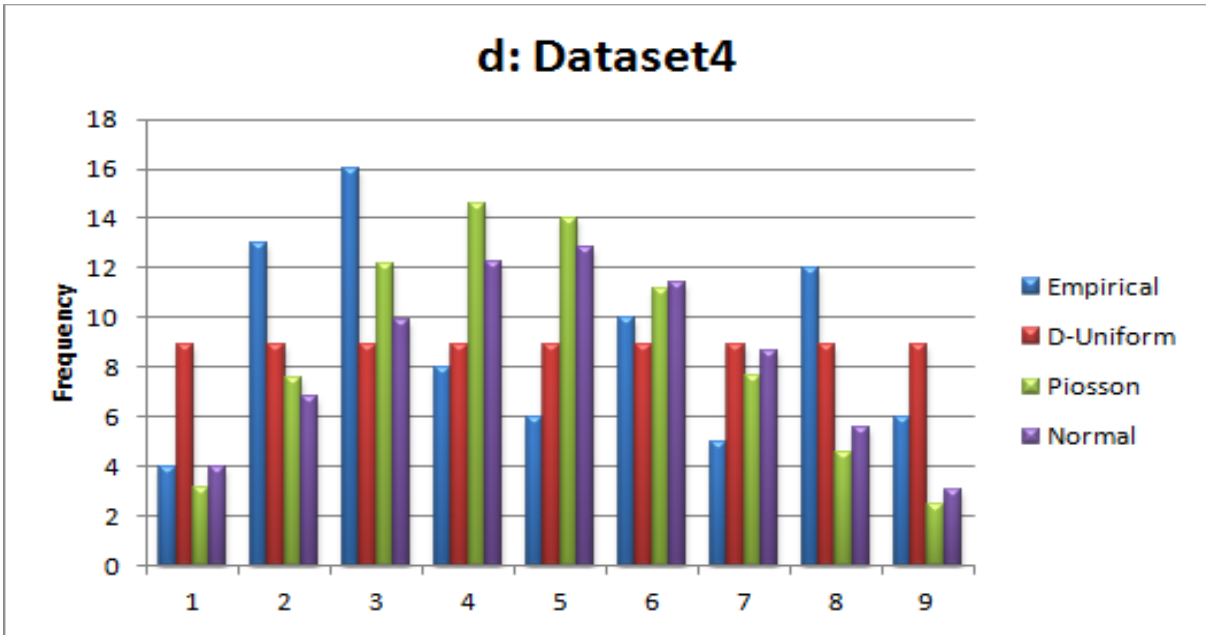


Figure 6.1: Empirical, D-Uniform, Poisson, and Normal Distributions for Four Data samples

6.3 Experimental Comparison of the Proposed Approaches

The effectiveness of the proposed approaches is compared by using four performance metrics: the degree of overall tenants' satisfaction, degree of commonality, degree of adherence to the risk factor, and the running time to generate the release plans. In this experiment, Matlab is used to implement FIS-based, BLP-based, GA-based approaches. The Matlab code is run using two groups of scenarios. In the first group, we fix the number of tenants and change the number of features. In the second group, we fix the number of features and change the number of tenants. We assume in all scenarios that the available effort takes a random value is in the range of 30% to 70% of the total required effort i.e., $0.30(sum\ (\ EFFORT\_Vector)) \leq Aval\_E \leq 0.70(sum\ (\ EFFORT\_Vector))$.

The compliance of service levels of tenants with the features is assumed to be in the range of 30% to 70% (for example, if the number of tenants is 10 and a feature f_i complies with 6

tenants, then we say the degree of compliance of f_i is 60%). These ranges of required effort and the degree of compliance of service levels are chosen because we want to test cases in which the available effort is very limited, and the cases in which the available effort is in the acceptable level. Furthermore, we want to cover the highest possible scenarios regarding the service levels of tenants. We generate many release plans (a release plan is the output of iteration). At the end of each iteration, we measure the degree of overall satisfaction, the degree of commonality, the degree of the adherence to the risk, and the time for each approach (FIS, BLP, and GA). We aim from these scenarios to figure out the effects of numbers of features and tenants on the performance of the proposed approaches, and we want to find the circumstances that suit each one of the three approaches.

Table 6.3: The Description of the First Group of Scenarios

Scenario #	Number of Iterations (release plans)	Number of tenants	Number of features
1	**500**	**5**	**20**
2	**500**	**5**	**50**
3	**500**	**5**	**100**
4	**100**	**5**	**700**

6.3.1 Variable Number of Features and Constant Number of Tenants

Table 6.3 shows the description of the first group of scenarios. In this group, the number of features is changed while the number of tenants is fixed.

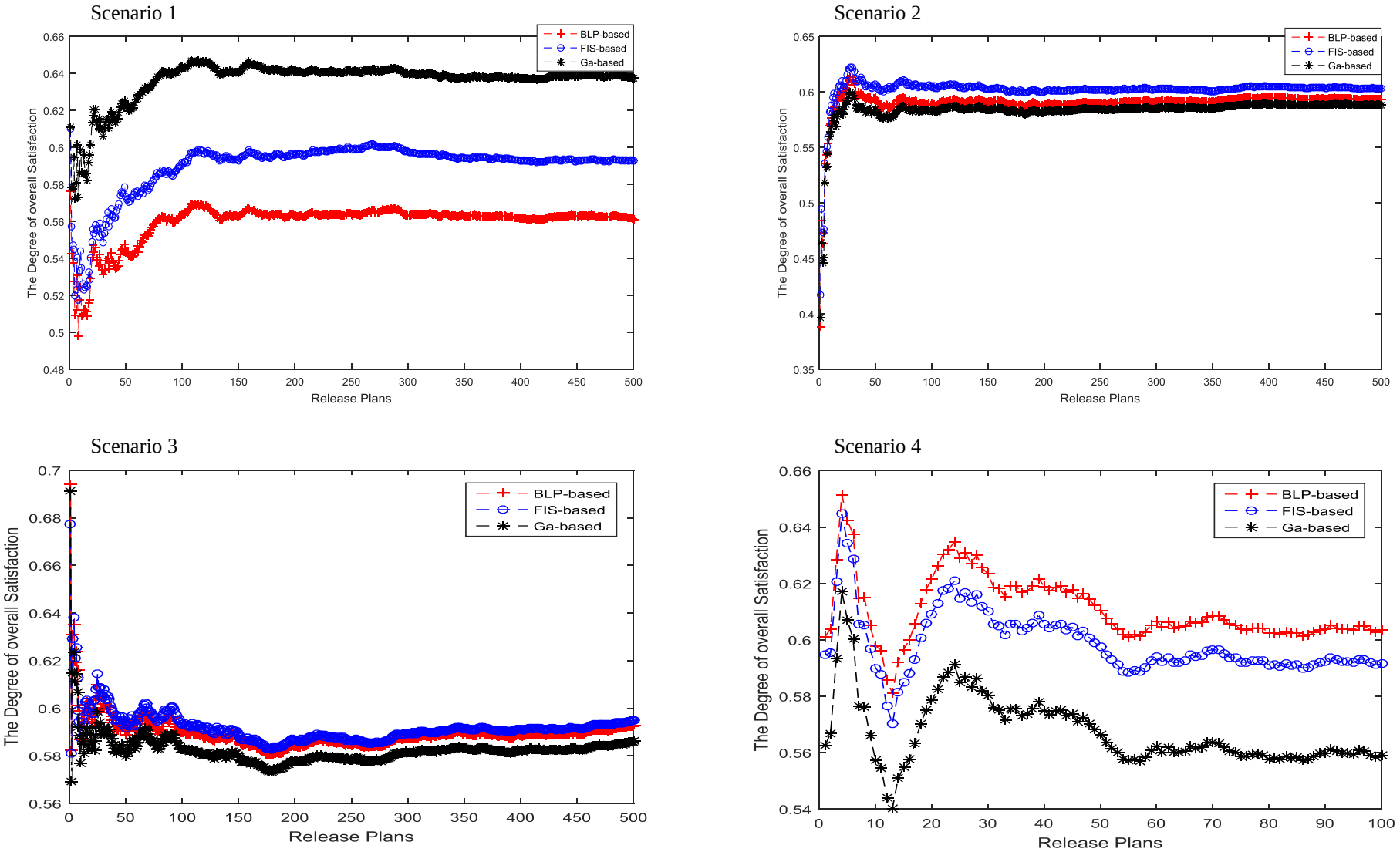
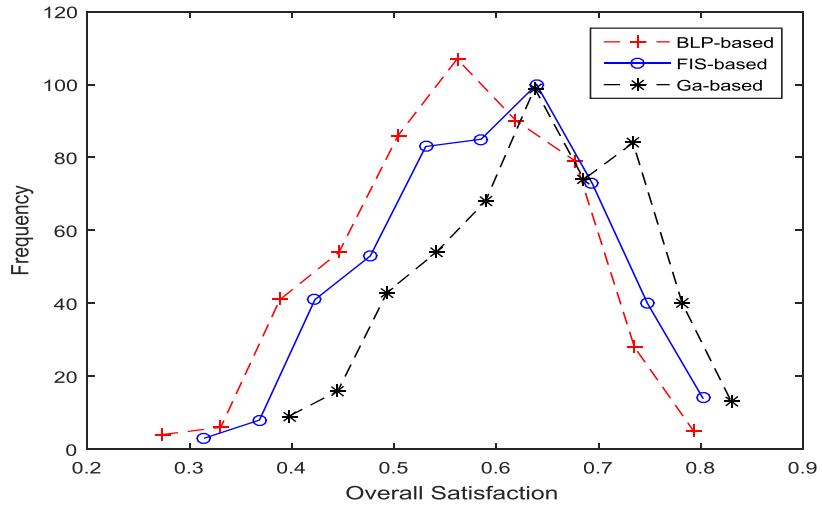
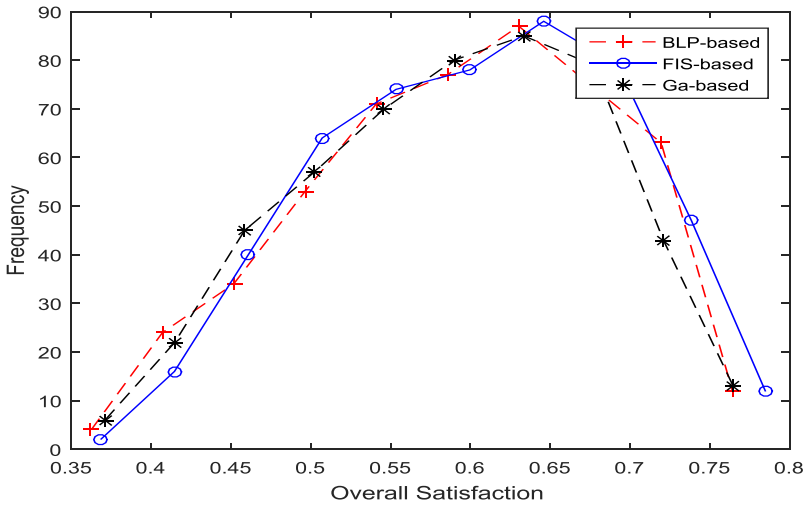


Figure 6.2: The Degree of Overall Satisfaction (Variable Number of Features and Constant Number of Tenants)

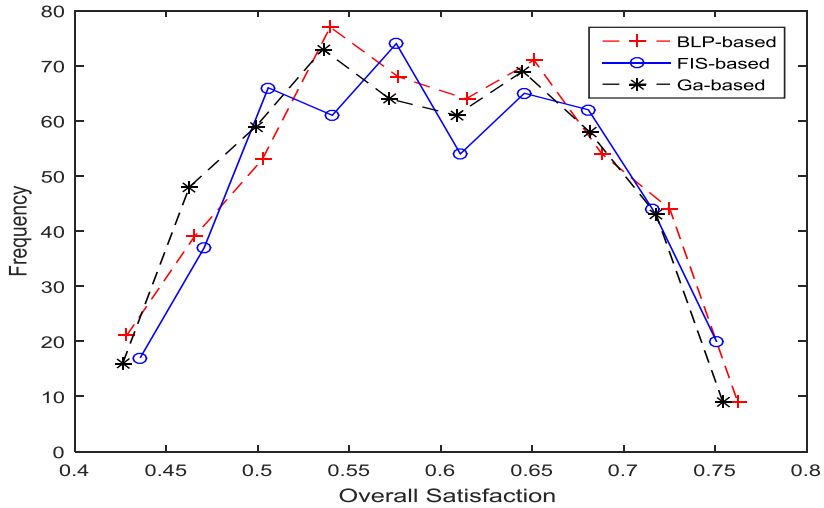
Scenario 1



Scenario 2



Scenario 3



Scenario 4

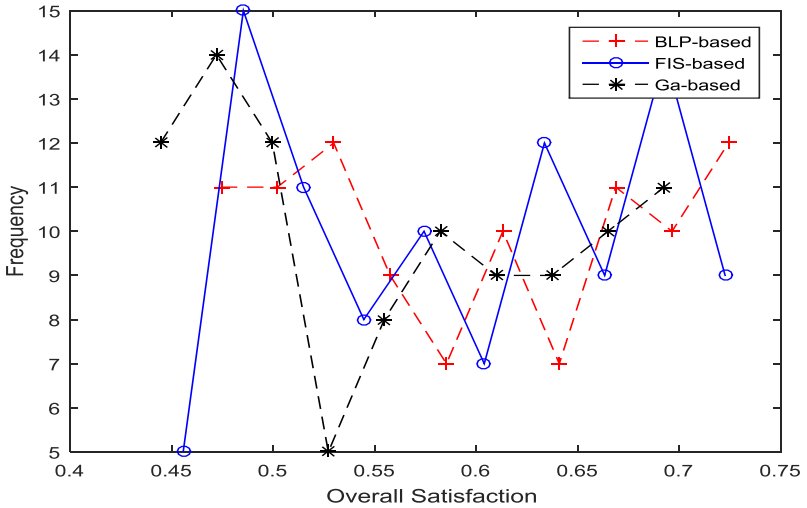
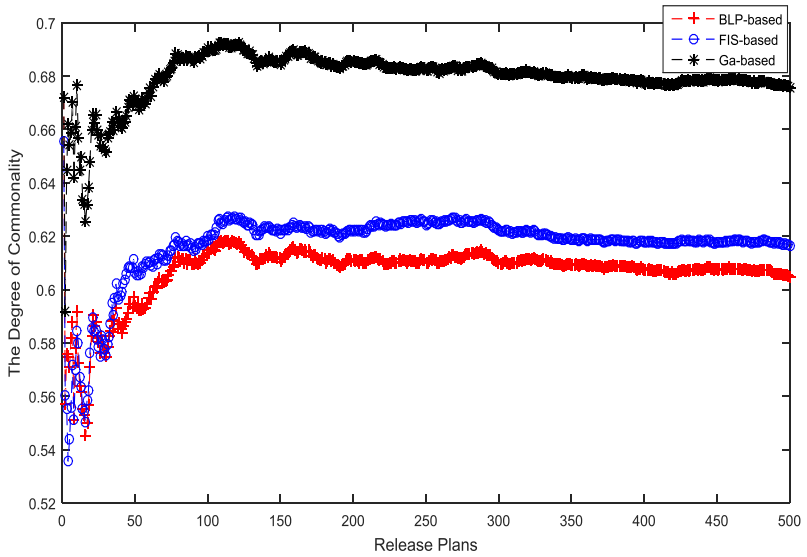


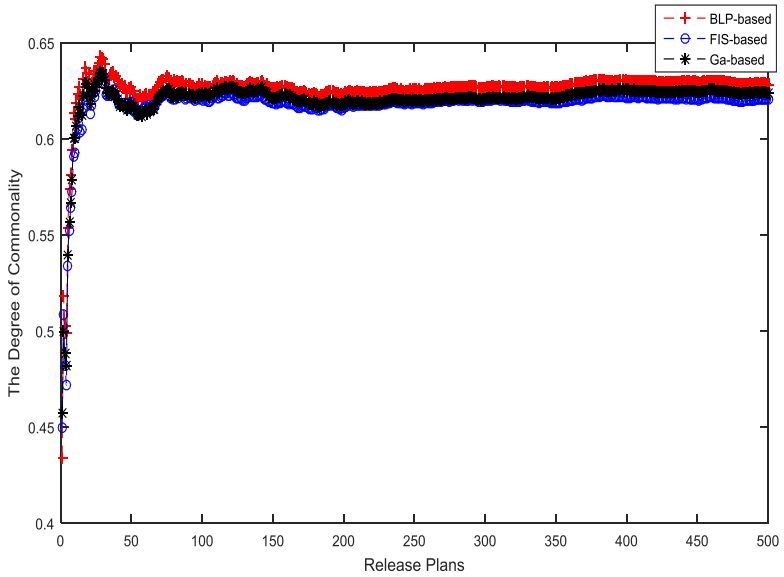
Figure 6.3: The Probability Distributions of the Degree of Overall Satisfaction

(Variable Number of Features and Constant Number of Tenants)

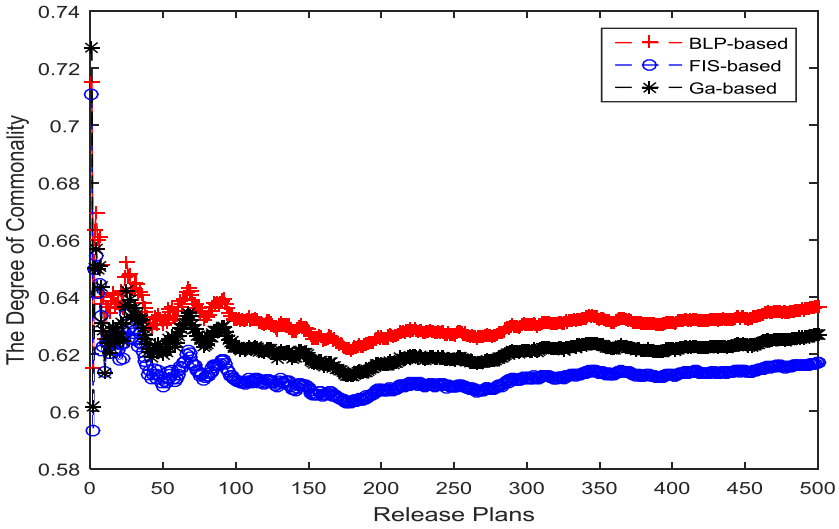
Scenario 1



Scenario 2



Scenario 3



Scenario 4

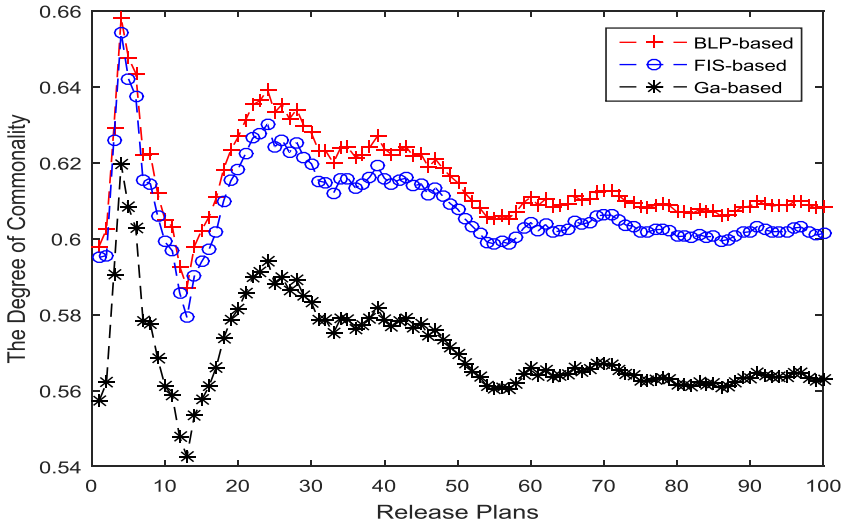


Figure 6.4: The Degree of Commonality (Variable Number of Features and Constant Number of Tenants)

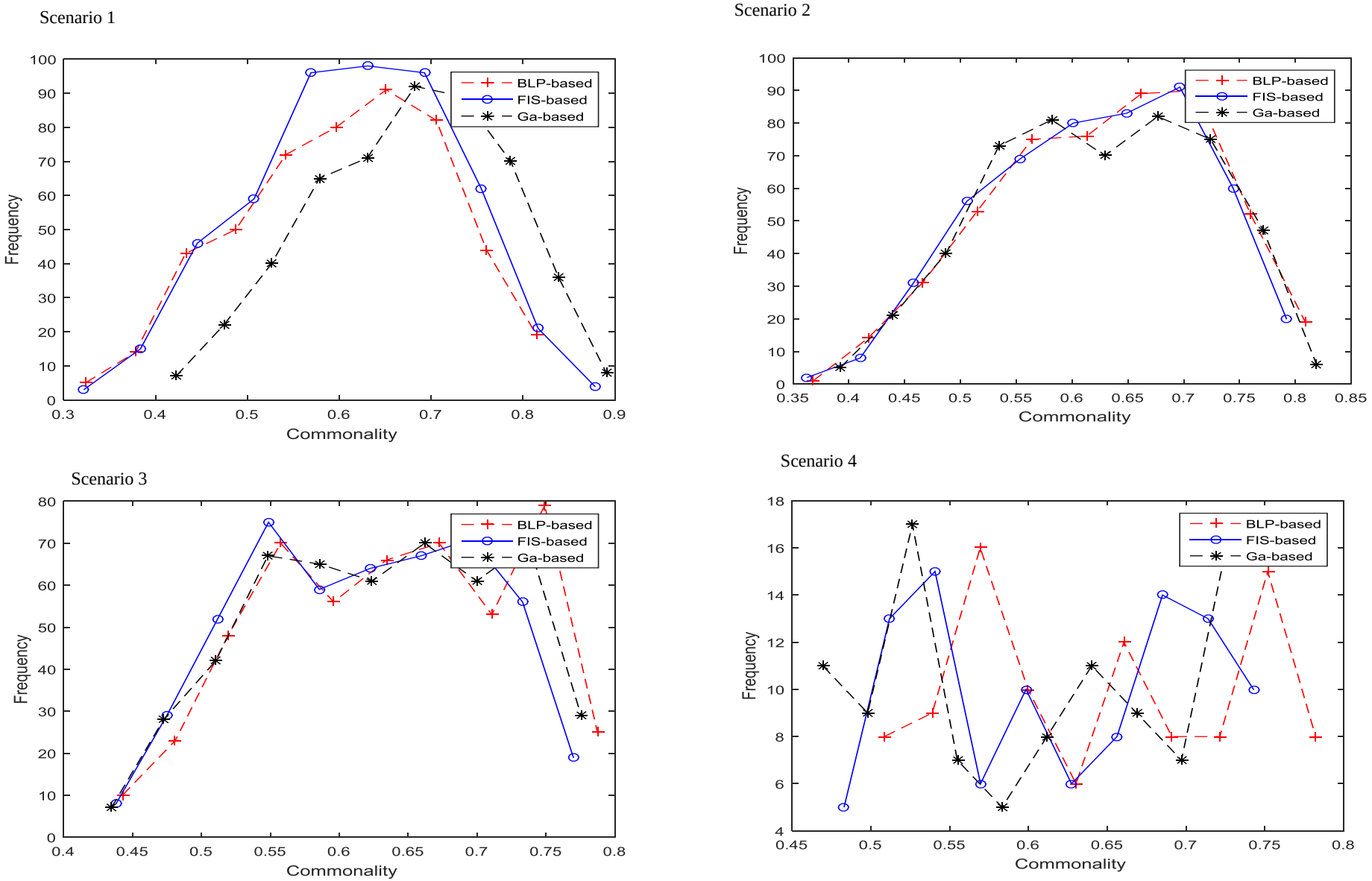


Figure 6.5: The Probability Distributions of the Degree of Commonality

(Variable Number of Features and Constant Number of Tenants)

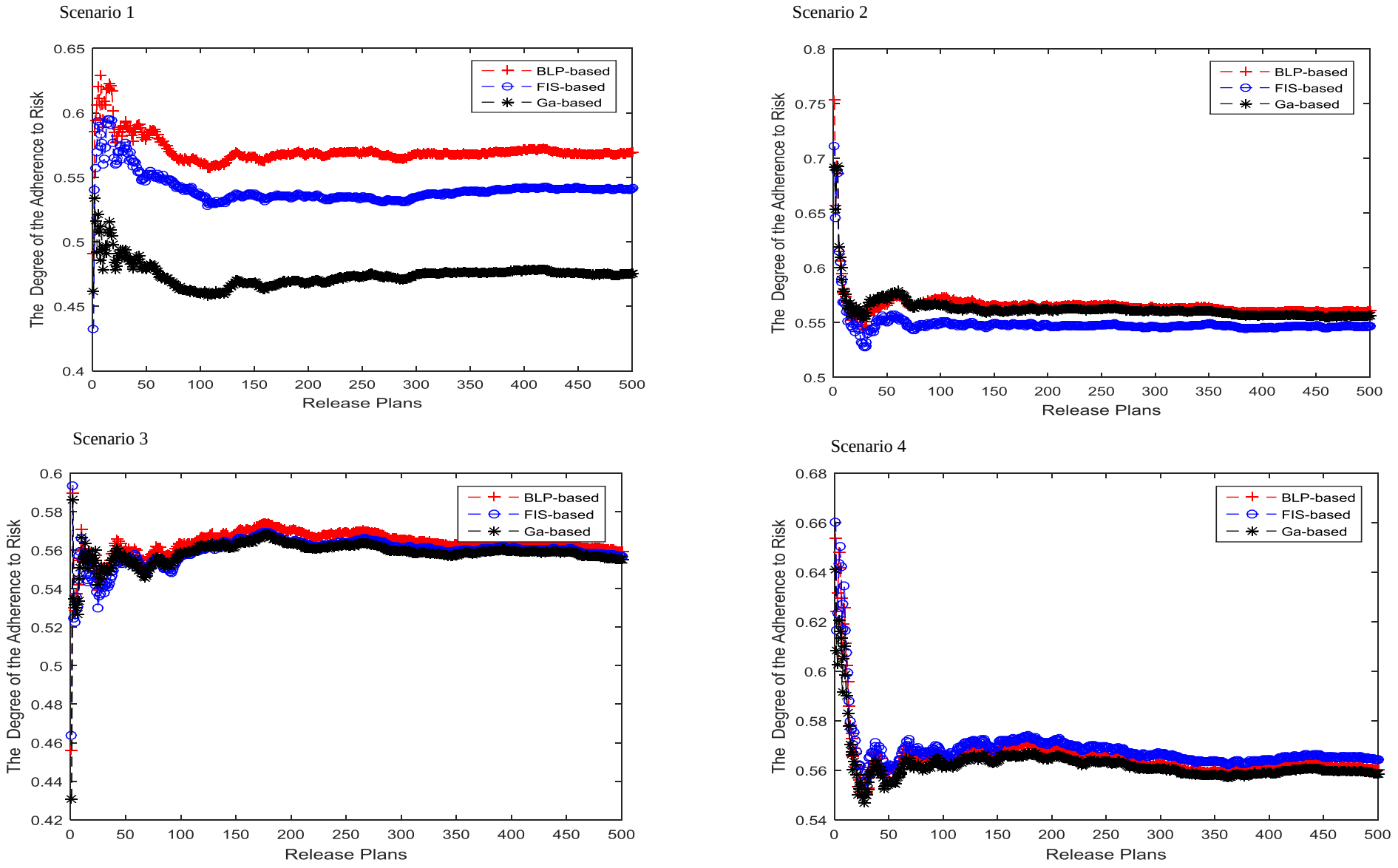


Figure 6.6: The Degree of the Adherence to Risk

(Variable Number of Features and Constant Number of Tenants)

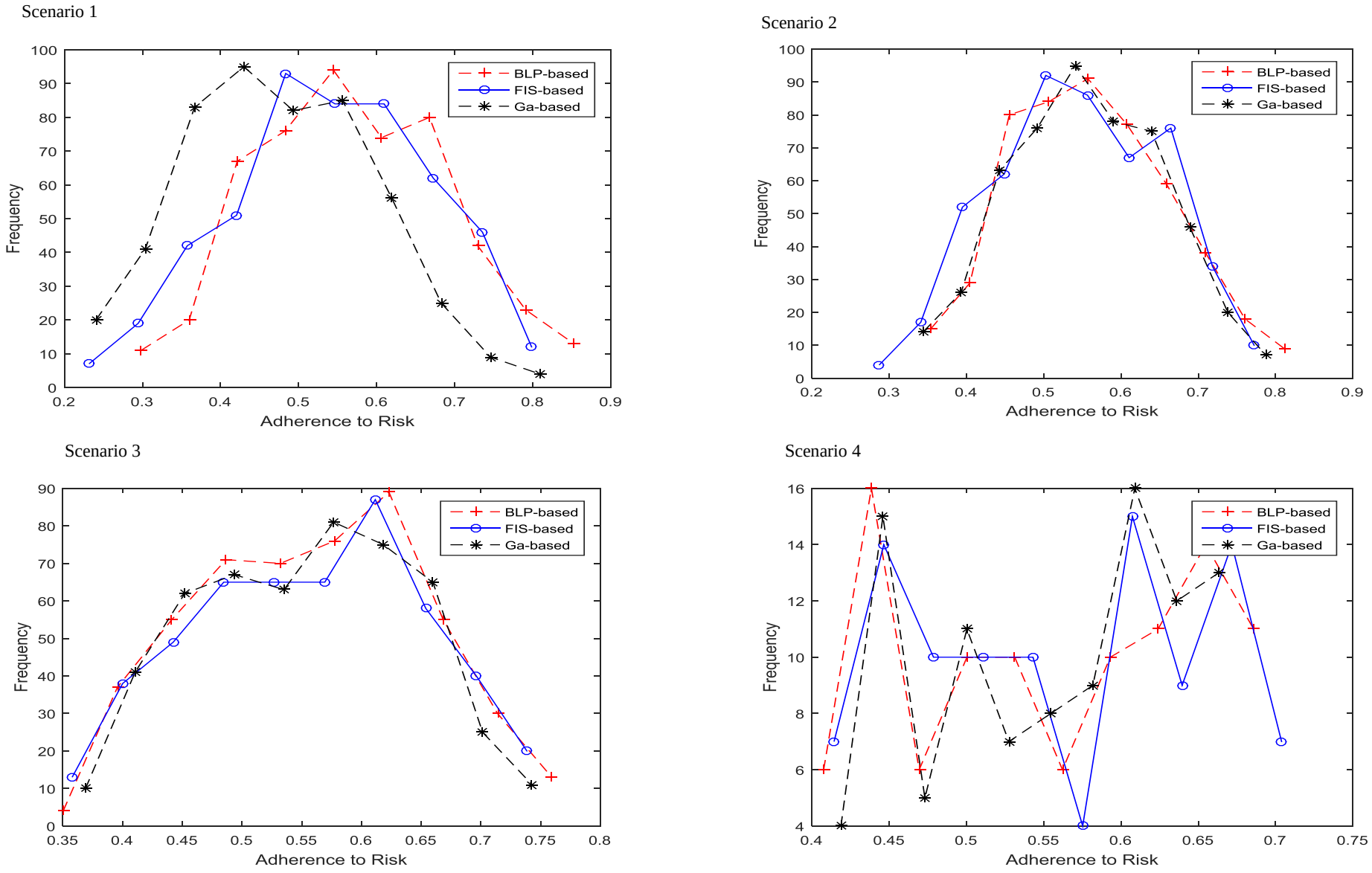


Figure 6.7: The Probability Distributions of the Degree of the Adherence to Risk

(Variable Number of Features and Constant Number of Tenants)

Table 6.4: Statistical Analysis of the First Group of Scenarios (Variable Number of Features and Constant Number of Tenants)

Scenario	Overall satisfaction			Commonality			Adherence to Risk		
		Mean	Standard Deviation	CI	Mean	Standard Deviation	CI	Mean	Standard Deviation	CI
1	FIS	0.593	0.104	[0.584, 0.602]	0.616	0.112	[0.603,0.63]	0.541	0.128	[0.53, 0.55]
	BLP	0.561	0.103	[0.552, 0.57]	0.605	0.110	[0.595,0.61]	0.569	0.125	[0.558, 0.58]
	GA	0.637	0.099	[0.628,0.646]	0.676	0.104	[0.667,0.68]	0.476	0.121	[0.465,0.47]
2	FIS	0.603	0.093	[0.592, 0.614]	0.620	0.095	[0.611,0.63]	0.547	0.108	[0.54, 0.56]
	BLP	0.593	0.093	[0.584,0.601]	0.628	0.097	[0.62,0.64]	0.561	0.103	[0.551, 0.57]
	GA	0.588	0.092	[0.58, 0.6]	0.624	0.096	[0.615,0.632]	0.556	0.099	[0.547, 564]
3	FIS	0.595	0.083	[0.59 0.6]	0.617	0.085	[0.61, 0.62]	0.557	0.096	[0.55, 0.57]
	BLP	0.593	0.085	[0.585 0.6]	0.637	0.090	[0.63, 0.645]	0.559	0.095	[0.55, 0.58]
	GA	0.586	0.084	[0.58, 0.6]	0.627	0.088	[0.62, 0.63]	0.555	0.090	[0.55, 0.56]
4	FIS	0.591	0.082	[0.57, 0.61]	0.615	0.082	[0.6, 0.63]	0.564	0.090	[0.56, 0.57]
	BLP	0.594	0.083	[0.58, 0.61]	0.640	0.085	[0.62, 0.66]	0.560	0.088	[0.55, 0.57]
	GA	0.576	0.082	[0.56, 0.6]	0.615	0.086	[0.6, 0.63]	0.559	0.080	[0.55, 0.57]

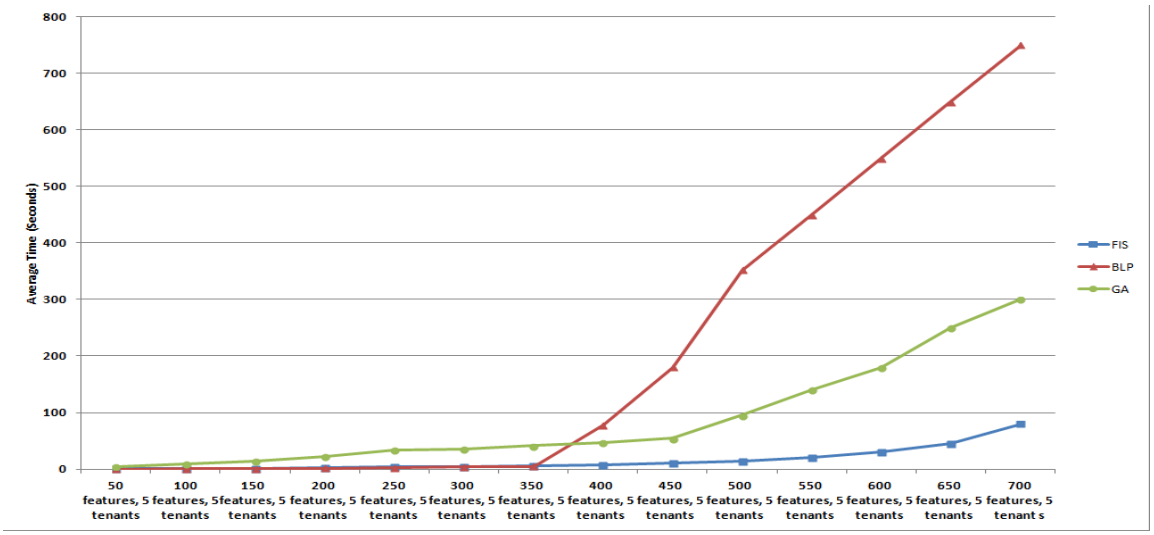


Figure 6.8: Running Time of the Three Approaches (Variable Number of Features and Constant Number of Tenants)

Figures 6.2 and 6.3 show the cumulative averages and the probability distributions of the degree of overall satisfaction of the release plans generated by the three approaches in each scenario. We can see from these figures and second column of Table 6.4 that GA has slightly achieved a higher degree of overall satisfaction than other approaches when the number of features is small, and BLP is slightly higher when there are a huge number of features. In general, the three approaches are comparable from the perspective of the degree overall satisfaction in most of scenarios. Figures 6.4 and 6.5 show the cumulative averages and the probability distributions of the degree of commonality of the three approaches. We can see from these figures and the third column of Table 6.4 that when there is a small number of features, GA has slightly achieved better results, and when there is huge number of features, BLP is slightly better. In general, there are no noticeable differences among the three approaches from the perspective of the degree of commonality. Figures 6.6 and 6.7 show the cumulative averages and the probability distributions of the degree of adherence to risk. We can notice that when there is the number of features is small BLP has achieved

better results than other approaches, and when there are huge number of features FIS has better results than other approaches. In general, the three approaches have approximately achieved comparable results in this measurement. In addition, we can conclude from this experiment that changing number of features has no clear influences on the performance of the three approaches from the perspectives of degree of overall satisfaction, commonality, and adherence to risk, which means that while the number of features is increased, all of the three approaches have almost the same performance from the these three metrics. However, the performance of the three approaches from the perspective of time is different. We run more points between 50 and 700 features in order to see how the running time grows by increasing the number of features. As Figure 6.8 shows, the time of BLP has extremely grown in after exceeding 350 features. We can also see that FIS is clearly the fastest. The time in FIS has slowly grown by increasing the number of features. When there are 700 features, the FIS is six times faster than BLP and 3 times faster than GA. GA is two times faster than BLP.

6.3.1 Variable Number of Tenants and Constant Number of Features

Table 6.5 shows the description of the second group of scenarios.

Table 6.5: The Description of the Second Group of Scenarios

Scenario#	Number Release Plans	Number of tenants	Number of features
5	**500**	**20**	**100**
6	**500**	**50**	**100**
7	**500**	**100**	**100**
8	**500**	**700**	**100**

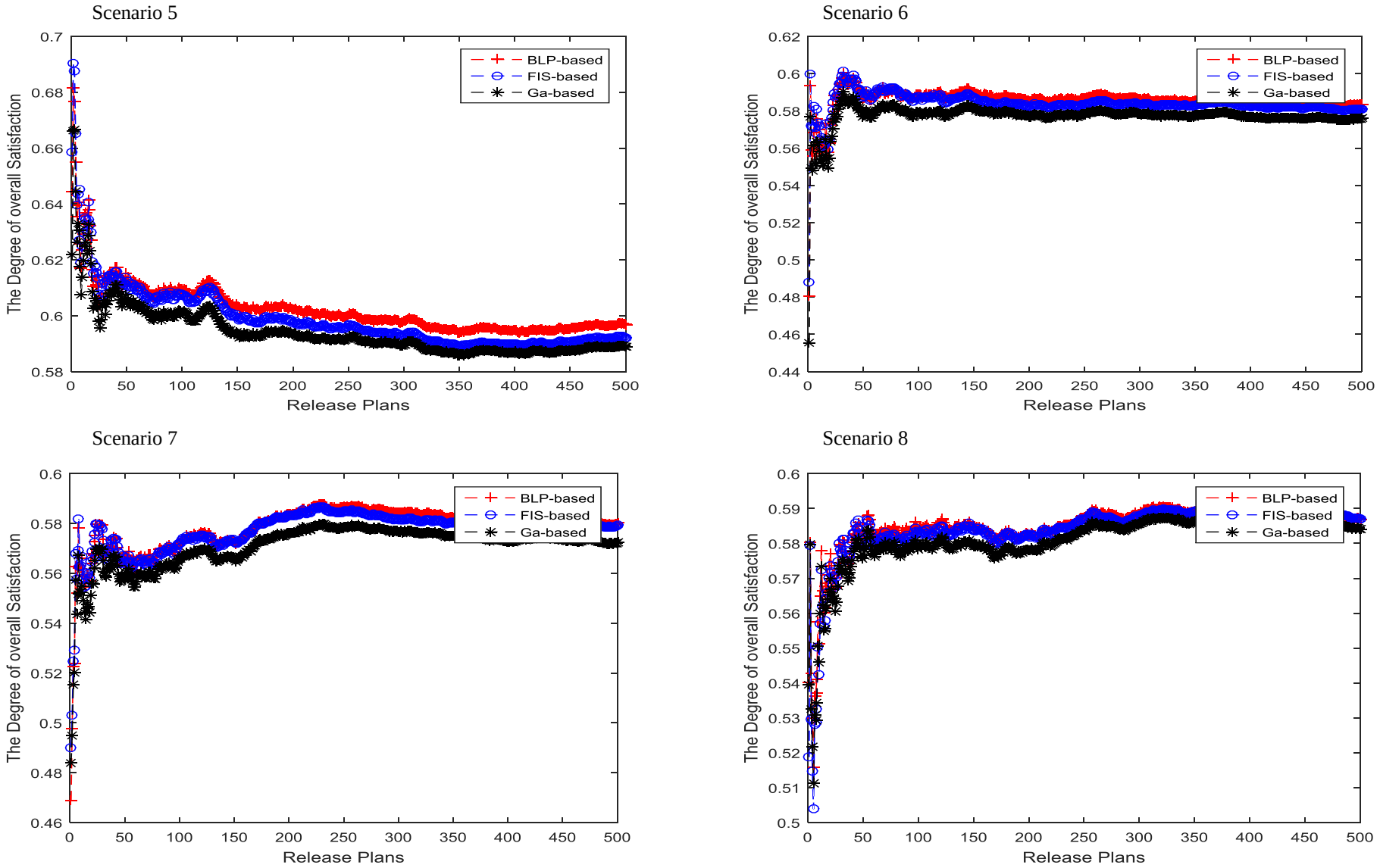
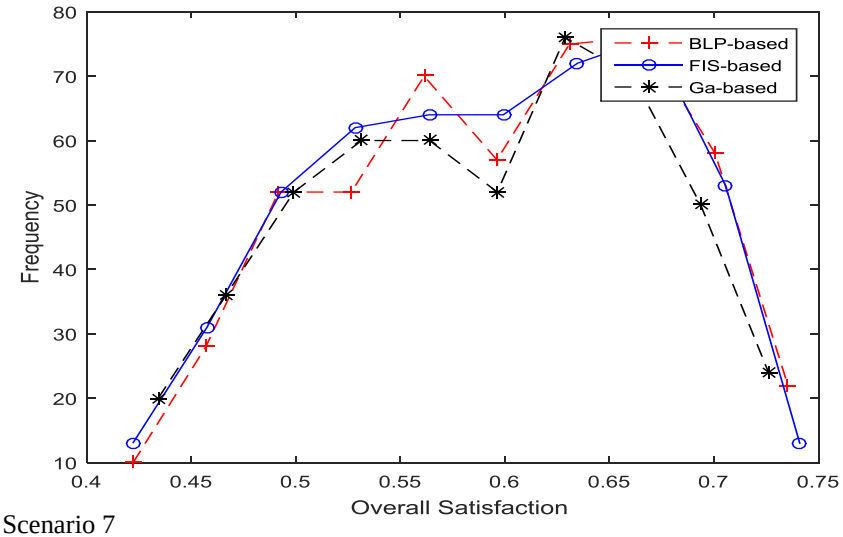


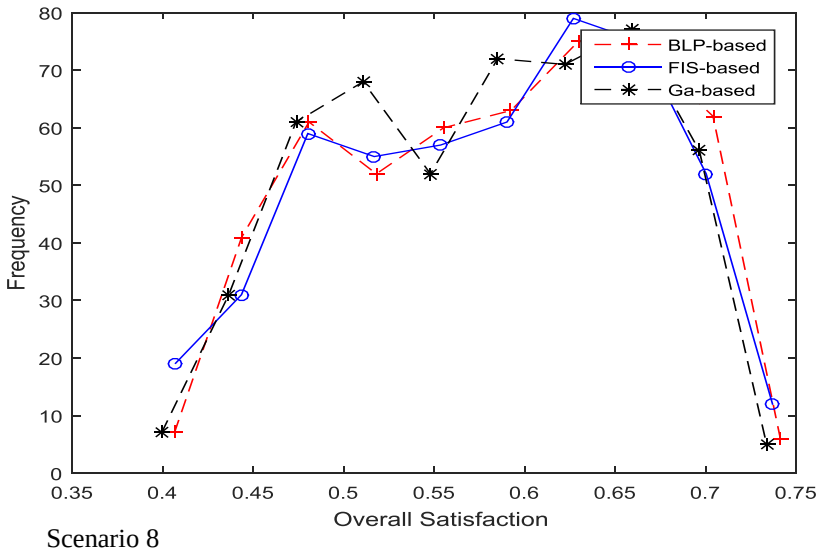
Figure 6.9: The Degree of Overall Satisfaction

(Variable Number of Tenants and Constant Number of Features)

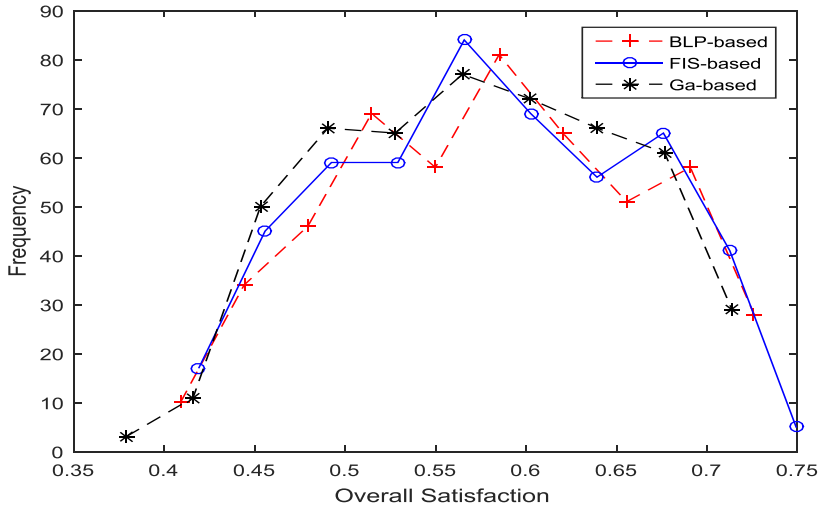
Scenario 5



Scenario 6



Scenario 7



Scenario 8

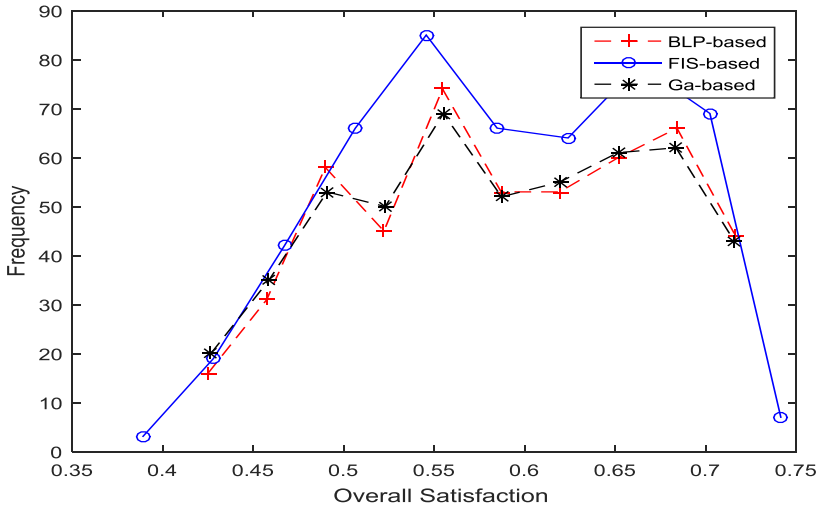


Figure 6.10: The Probability Distributions of the Overall Satisfaction (Variable Number of Tenants and Constant Number of Features)

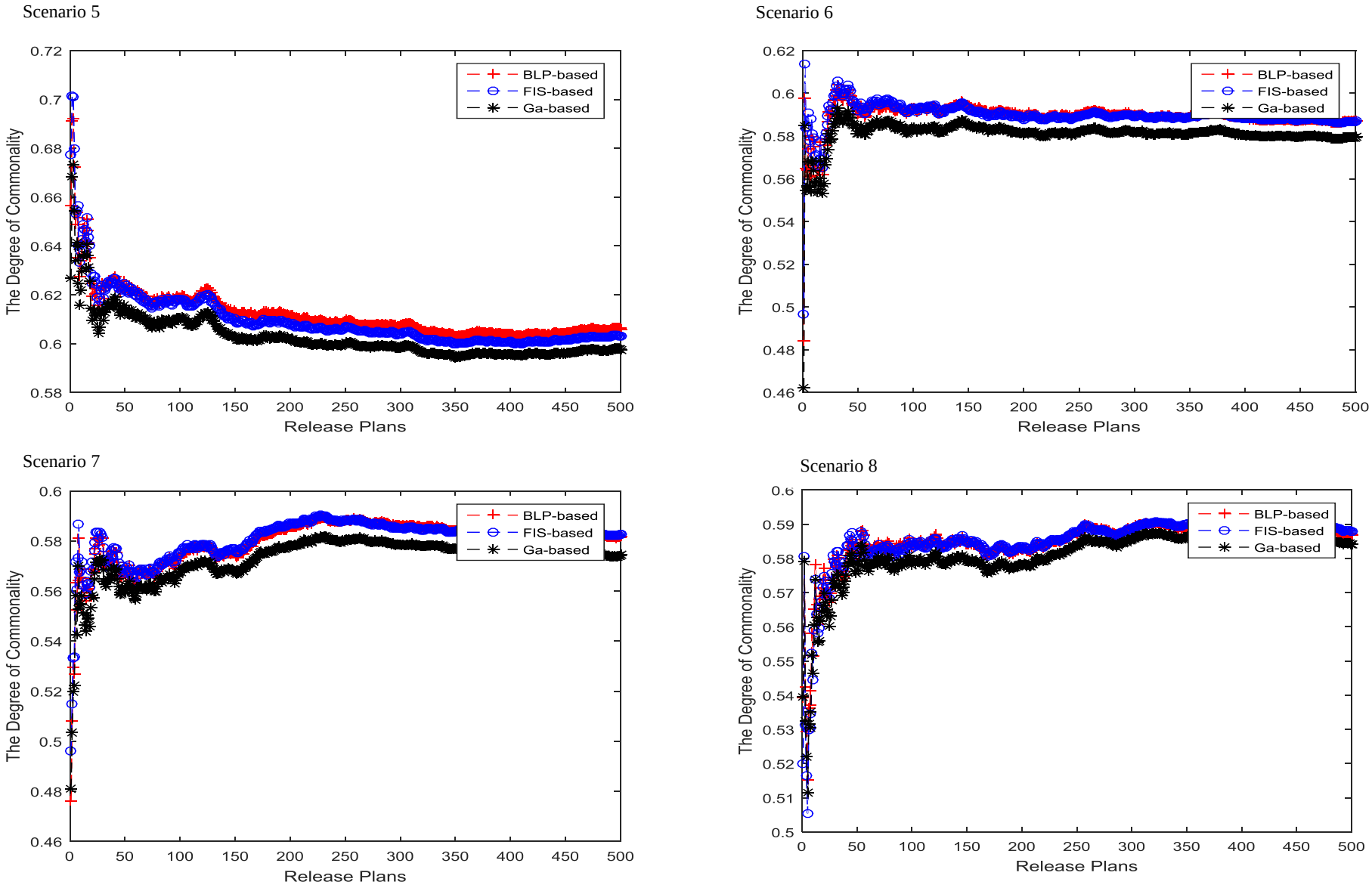


Figure 6.11: The Degree of Commonality (Variable Number of Tenants and Constant Number of Features)

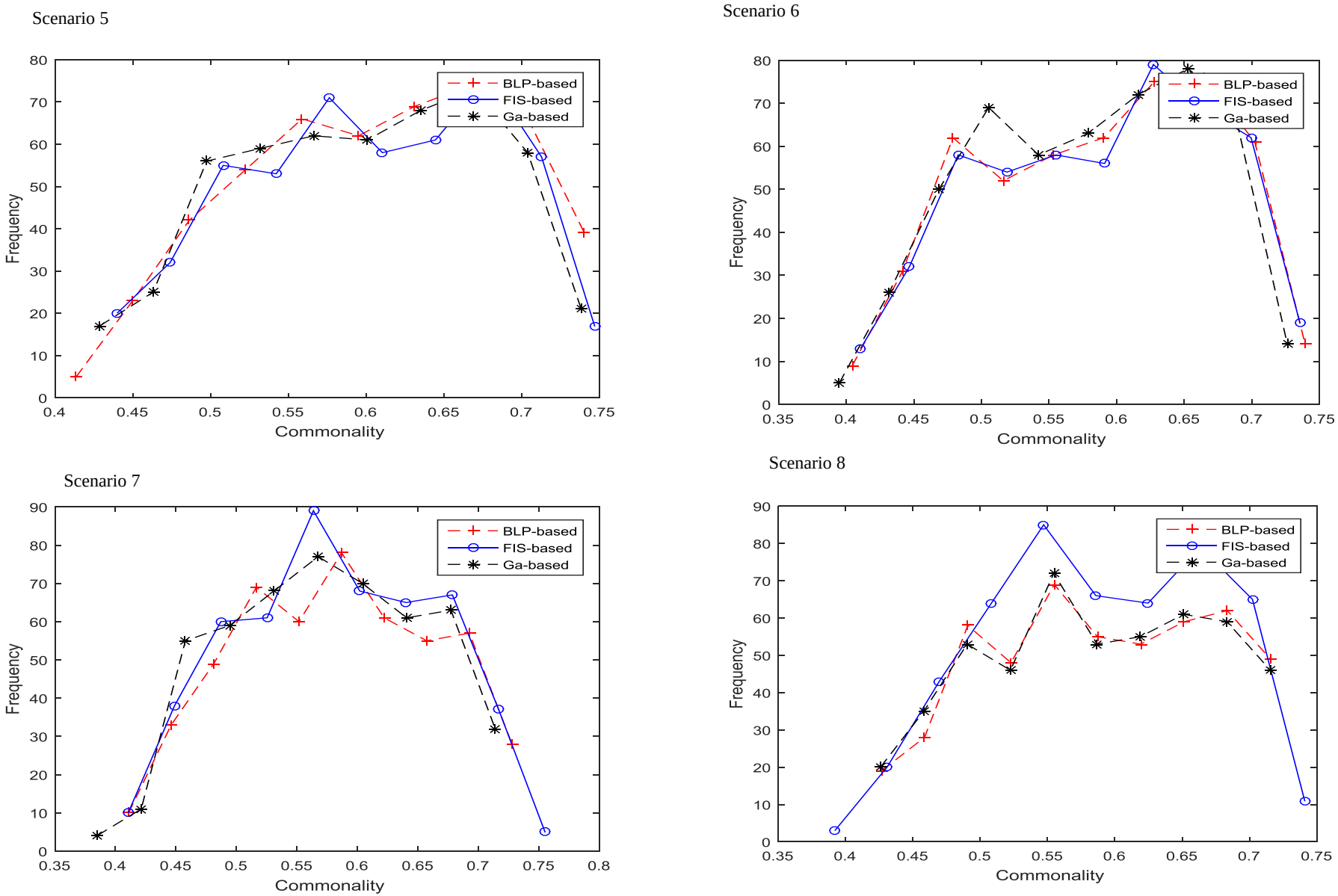


Figure 6.12: The Probability Distributions of the Degree of Commonality (Variable Number of Tenants and Constant Number of Features)

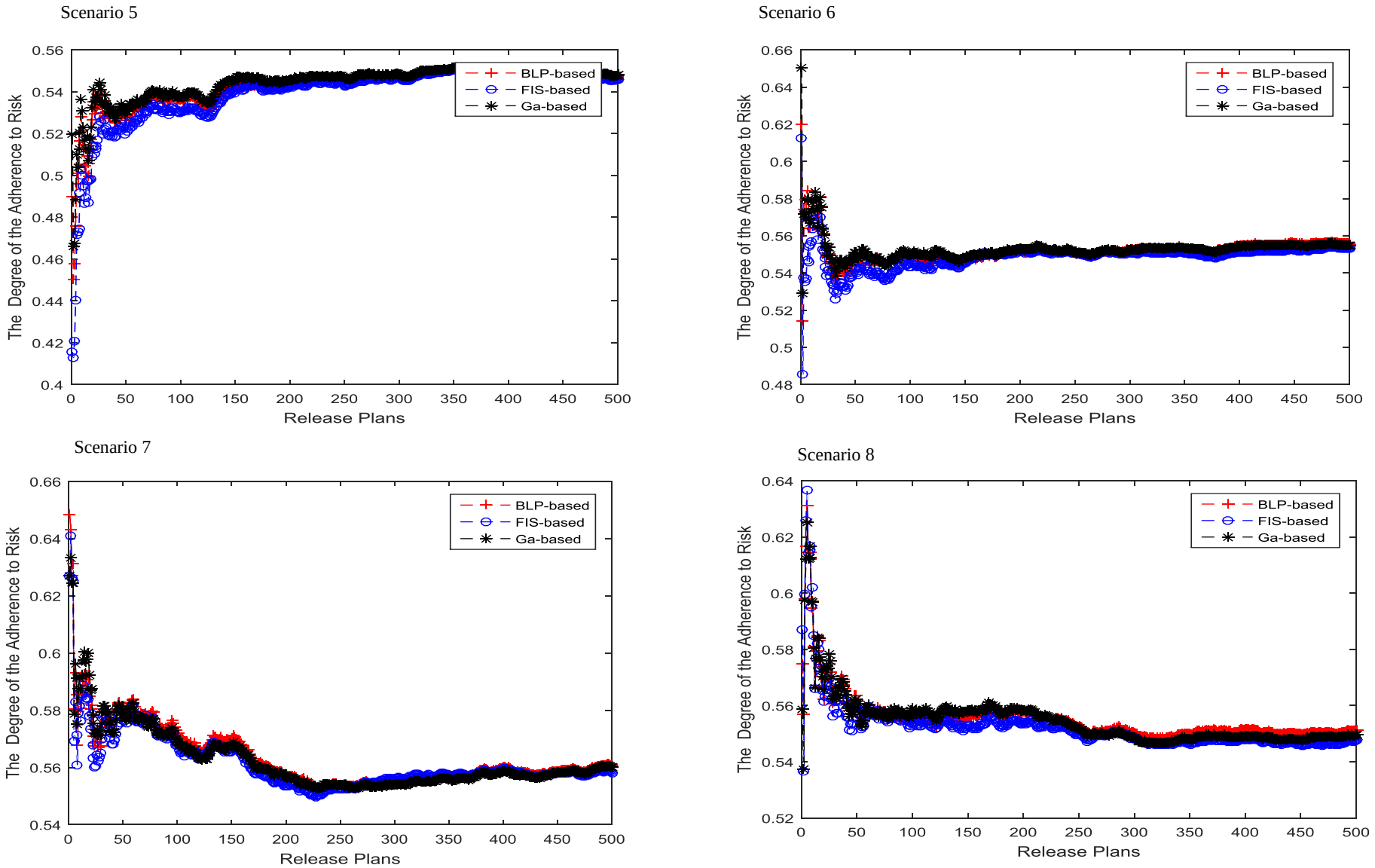


Figure 6.13: The Degree of the Adherence to Risk (Variable Number of Tenants and Constant Number of Features)

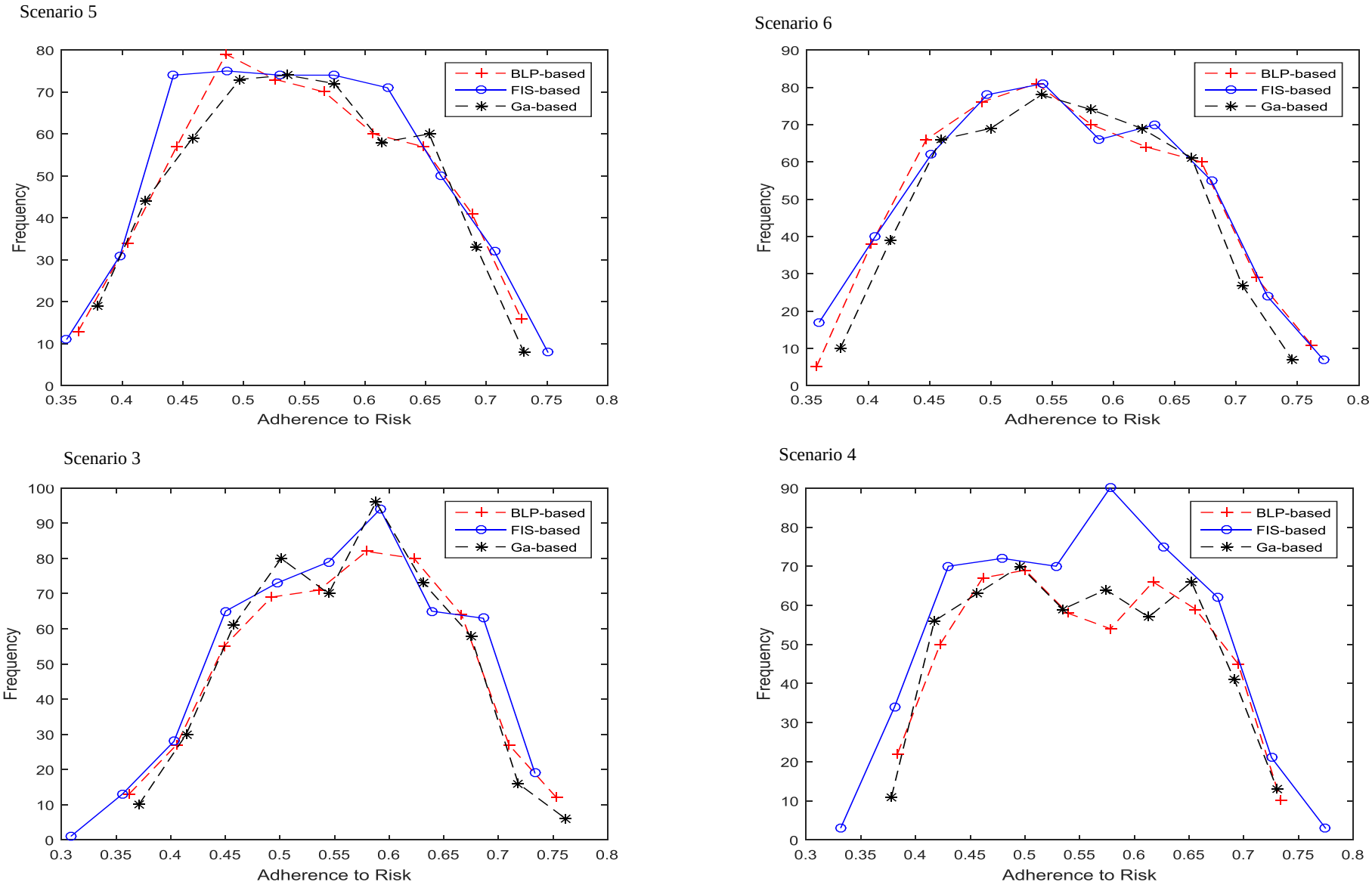


Figure 6.14: The Probability Distributions of the Degree of the Adherence to Risk (Variable Number of Tenants and Constant Number of Features)

Table 6.6: Statistical Analysis of the Second Group of Scenarios (Variable Number of Tenants and Constant Number of Features)

Scenario	Overall satisfaction			Commonality			Adherence to Risk		
		Mean	Standard Deviation	CI	Mean	Standard Deviation	CI	Mean	Standard Deviation	CI
5	FIS	0.592	0.082	[0.56, 0.60]	0.603	0.082	[0.6, 0.61]	0.546	0.094	[0.54, 0.55]
	BLP	0.597	0.081	[0.59, 0.6]	0.606	0.083	[0.6,0.61]	0.547	0.092	[0.54, 0.56]
	GA	0.589	0.080	[0.58,0.6]	0.597	0.082	[0.59,0.6]	0.548	0.088	[0.54,0.56]
2	FIS	0.581	0.086	[0.57, 0.59]	0.587	0.085	[0.58,0.6]	0.553	0.098	[0.54, 0.56]
	BLP	0.584	0.085	[0.58,0.6]	0.587	0.085	[0.58,0.6]	0.556	0.095	[0.55, 0.56]
	GA	0.576	0.083	[0.57, 0.58]	0.580	0.083	[0.57,0.59]	0.555	0.087	[0.55, 56]
3	FIS	0.579	0.082	[0.57, 0.59]	0.583	0.082	[0.575, 0.59]	0.558	0.094	[0.55, 0.57]
	BLP	0.580	0.082	[0.57, 0.59]	0.582	0.083	[0.575, 0.6]	0.560	0.093	[0.55, 0.57]
	GA	0.572	0.080	[0.57, 0.58]	0.574	0.081	[0.57, 0.58]	0.560	0.086	[0.55, 0.57]
4	FIS	0.587	0.084	[0.58, 0.6]	0.588	0.084	[0.58, 0.6]	0.548	0.097	[0.54, 0.56]
	BLP	0.587	0.083	[0.58, 0.6]	0.587	0.083	[0.58, 0.6]	0.551	0.093	[0.54, 0.56]
	GA	0.584	0.082	[0.58, 0.6]	0.584	0.082	[0.58, 0.6]	0.550	0.091	[0.54, 0.56]

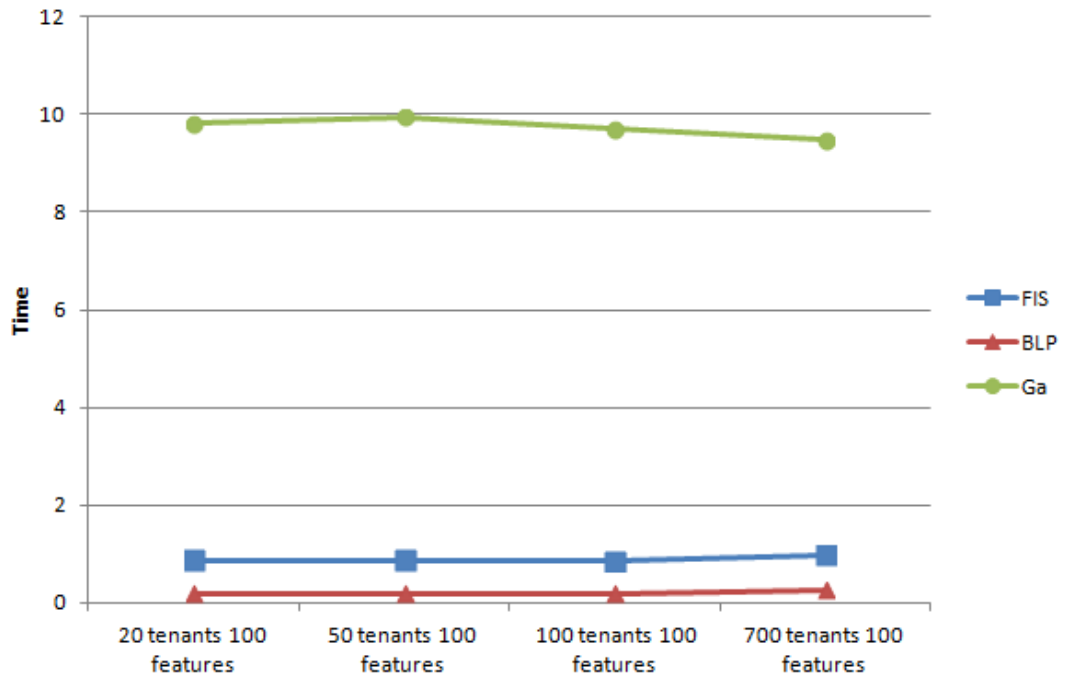


Figure 6.15: Running Time of the Three Approaches (Variable Number of Tenants and Constant Number of Features)

Figures 6.9 to 6.14 show that the degree of overall satisfaction, commonality, and adherence to risk are not affected by changing the number of tenants. We can observe that all of these three metrics have not considerably changed by the changes in the number of tenants. We can also see that they roughly have the same results from the perspectives of these three metrics. Figure 6.15 shows that the time of the three approaches has not changed by increasing the number of tenants. We can conclude that the number of tenants has no influences on the three approaches from degree of overall satisfaction, commonality, adherence to risk, and time.

The next section explains why the three approaches have achieved comparable results in the degree of overall satisfaction, commonality, and adherence to risk.

6.4 The Similarity of the Release Plans Generated by the Proposed Approaches

In this section, we measure the similarity of the release plans that are generated by the FIS-based, BLP-based and GA-based approaches. Let $F^{\wedge}_{FIS}$, $F^{\wedge}_{BLP}$, and $F^{\wedge}_{GA}$ be the release plans generated by the three approaches using the same inputs. That means, we feed the three approaches with data about importance, risk, commonality, available and required effort, dependencies, compliance with the service levels, and tenants' decision weights, and generate three release plans (a release plan from each approach). It is required to figure out how much these release plans are similar to each others. Each $F^{\wedge} \in \{F^{\wedge}_{FIS}, F^{\wedge}_{BLP}, F^{\wedge}_{GA}\}$ is represented using a a vector of decision variables $X = [x_1\ x_2 \dots\dots\dots, x_n]$ where $x_i \in \{0,1\}$ and n is the number of the candidate features ($n = |F^*|$). If $x_i = 1$ then the feature f_i is assigned to the next release; otherwise, it is not. Let X_{FIS}, X_{BLP}, X_{GA} are the decision vectors associated with the release plans generated by the three approaches. The similarity between two release plans X_1 and X_2 can be measured using hamming distance as follows:

$$Similarity\ (X_1, X_2) = 1 - \frac{H(X_1, X_2)}{n} \qquad (6.2)$$

Such that $H(X_1, X_2)$ is the hamming distance between the two vectors.

Example: 6.1: Suppose $n = 5$, $X_{FIS} = \{\mathbf{1}\ 1\ 1\ 0\ 0\}$, and $X_{GA} = \{\mathbf{0}\ 1\ 1\ 0\ 0\}$, then $Similarity\ (X_{FIS}, X_{GA}) = 1 - \frac{1}{5} = 80\%$, which means the release plans that are generated by FIS and GA approaches are similar to the degree of 80%.

Figure 6.16 and Table 6.6 show the average similarity between the release plans generated by the proposed approaches in different scenarios. The lowest degree of similarity

is 73% (the second scenario). This high degree of similarity among the proposed approaches explains why the results of the degree of overall satisfaction, commonality, and adherence to risk of these three approaches are close to each other.

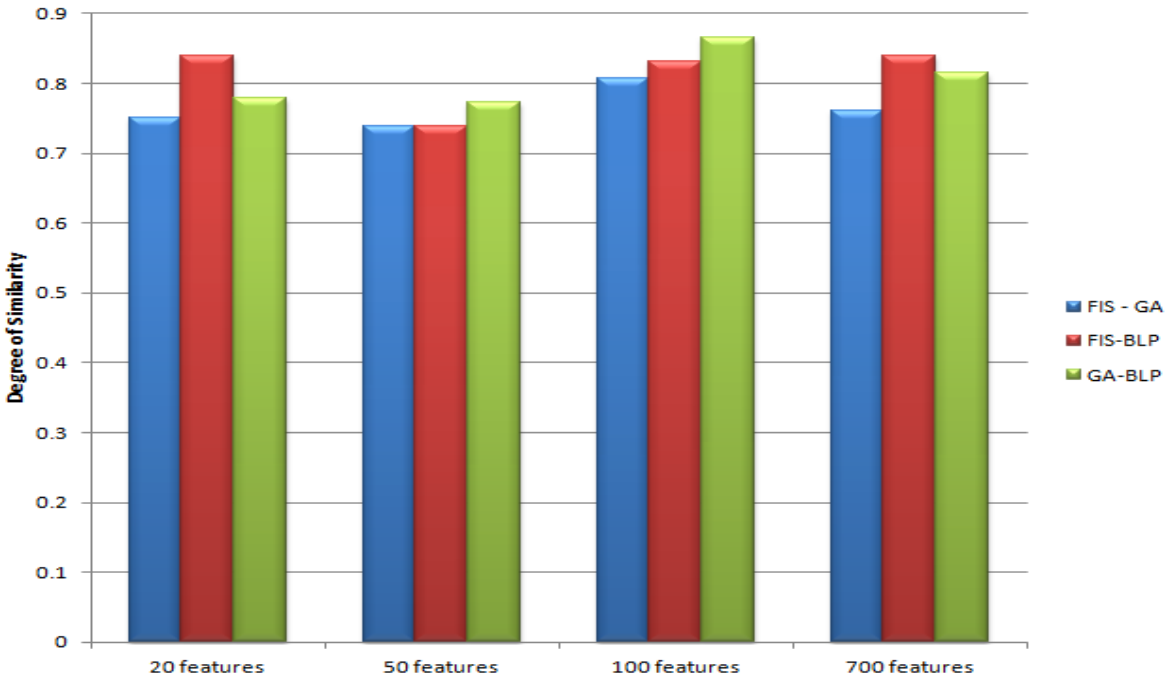


Figure 6.16: Similarity between the Proposed Approaches

Table 6.7: Statistics of the Similarity between the Proposed Approaches

		FIS-GA	FIS-BLP	GA-BLP
20 features	**mean**	0.75	0.84	0.779
	SD	0.0171	0.014	0.0212
	CI	[0.745, 0.754]	[0.835, 0.84]	[0.77 , 0.78]
50 features	**mean**	0.738	0.739	0.772
	SD	0.1002	0.0907	0.0782
	CI	[0.71, 0.77]	[0.71,0.76]	[0.75, 0.8]
100 features	**mean**	0.806	0.832	0.865
	SD	0.0313	0.0323	0.0361
	CI	[0.8, 0.81]	[0.82, 0.84]	[0.86, 0.88]
700 features	**mean**	0.7498	0.839	0.779
	SD	0.01697	0.01386	0.0203
	CI	[0.745, 0.754]	[0.835, 0.84]	[0.77,0.78]

6.5 Comparing the Proposed Approaches with an Approach from the Literature

In this section, we compare the proposed approaches to a GA approach that uses the fitness function that has been presented in [33] (hereinafter, we call this model the compared model). The compared model is chosen according to three criteria: 1) It is a well- known model and it has high similarity with many release planning models in the literature. 2) It is the closest model to the proposed formulation. 3) It is easy to implement. We run four Matlab modules with the same inputs: BLP, FIS, and two GAs modules with the same parameters but with different fitness functions. The first GA uses the proposed fitness function define by equation (5.7), and the second GA uses the fitness function in the compared model. The fitness function in that model is denoted as follows:

$$maximize \;\; F(X),$$

$$F(x) = \sum_{k=1}^{K} \sum_{i:x(i)=k} WAS(i,k) \;, \qquad (6.3)$$

where

$$WAS(i,k) = \xi(k) \left[\sum_{p=1}^{q} \lambda(p) \times value(p,i) \times Priority(p,i,k) \right],$$

such that:

K is the number of releases for which we are planning,

$x(i) = k$ is interpreted as feature i is assigned to the release number k,

$\xi(k)$ is the weight of release k,

$\lambda(p)$ is the weight of stakeholder p, and $p = 1 \ldots q$,

$value(p,i)$ is the value of feature i to the stakeholder p,

$Priority(p,i,k)$ the priority of a feature i for a stakeholder p in a release k.

In this research, since the planning is just performed for the next release, the first three parameters will take the following values:

$K = 2$ (next release and future releases).

$k \in \{0,1\}$, $x(i) \in \{0,1\}$ (1 for the next release and 0 for future releases).

$\xi(1) = 1$ and $\xi(0) = 0$ (because the planning is only for the next release, all the weight is given to it).

Depending on these values, the fitness function in (6.3) can be rewritten as follows:

$$maximize\ F(X)$$

$$F(x) = \sum_{i:x(i)=0..1} WAS(i,1) + WAS(i,0)$$

$$= \sum_{i:x(i)=0..1} \left[\xi(1) \left[\sum_{p=1}^{q} \lambda(p) \times value(p,i) \times Priority(p,i,1) \right] + \xi(0) \left[\sum_{p=1}^{q} \lambda(p) \times value(p,i) \times Priority(p,i,0) \right] \right]$$

$$= \sum_{i:x(i)=0..1} \left[\left[\sum_{p=1}^{q} \lambda(p) \times value(p,i) \times Priority(p,i,1) \right] + 0 \right]$$

$$= \sum_{i:x(i)=1} \sum_{p=1}^{q} \lambda(p) \times value(p,i) \times Priority(p,i,1) \qquad (6.4)$$

The problem constraints in the compared model are similar to the proposed approaches in dependencies and effort constraints. However, compared model does not include service level constraints.

The four modules are run for 100 iterations using the same inputs in each iteration. In each iteration, we measure the degree of overall satisfaction, commonality, and adherence to risk. We use random data for 100 features and 100 tenants in this experiment. As Figures 6.17 and 6.18 show, the proposed approaches have achieved better results than the compared model in the degree of overall satisfaction and commonality (approximately 2% for overall satisfaction and 4% for commonality). Note that the calculations of the degree of overall satisfaction and commonality take into account the service levels of the tenants. The compared model does not consider this factor when it generates release plans. It aims only to maximize the additive value of the multiplication between the priorities and values of the selected features regardless of the compliance with the service levels of the tenants. Therefore, the compared model may select features that do not have high degree of compliance with the service levels, which is reflected on the values of the degree of overall satisfaction and commonality.

Figure 6.19 shows that the proposed approaches have clearly shown better results (about 18% higher) in the degree of adherence to risk. This result is obtained because the proposed approaches consider the risk, overall satisfaction, and commonality simultaneously when the features are assigned to a release plan, while the compared model does not consider the risk.

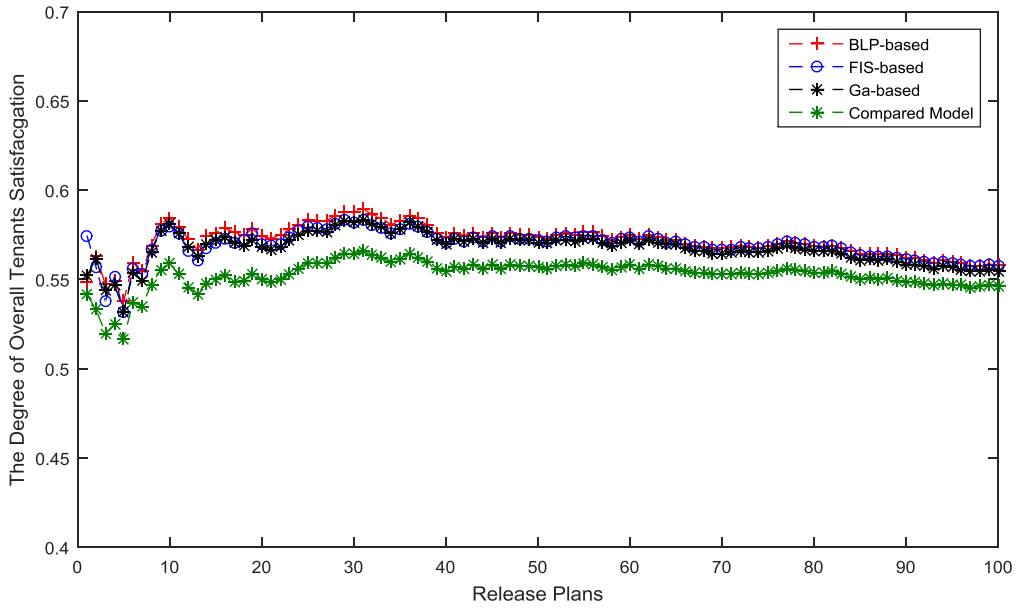


Figure 6.17: Comparison with the Compared Model (Degree of Overall Satisfaction)

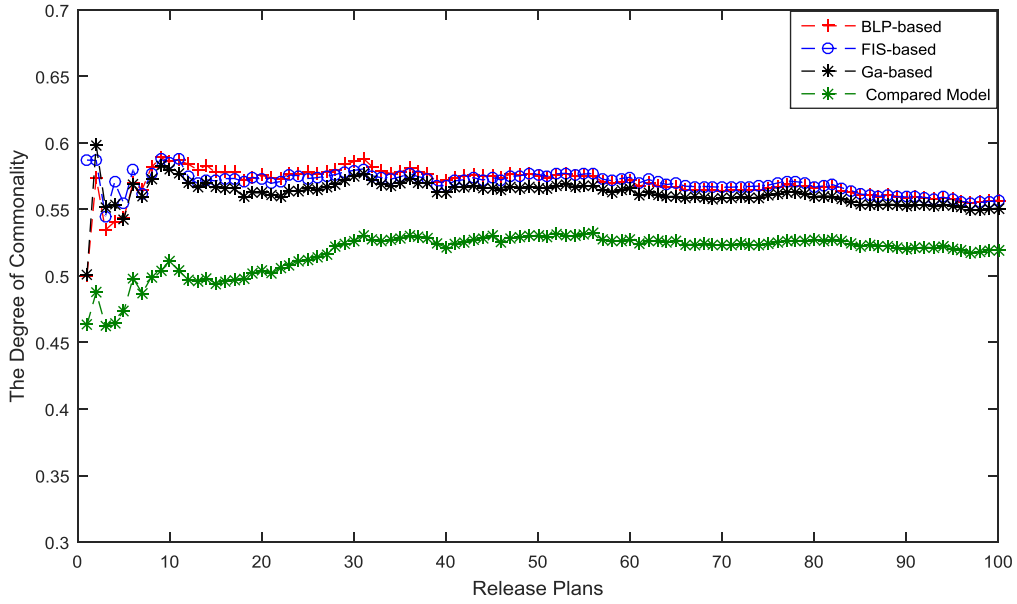


Figure 6.18: Comparison with the Compared Model (Degree of Commonality)

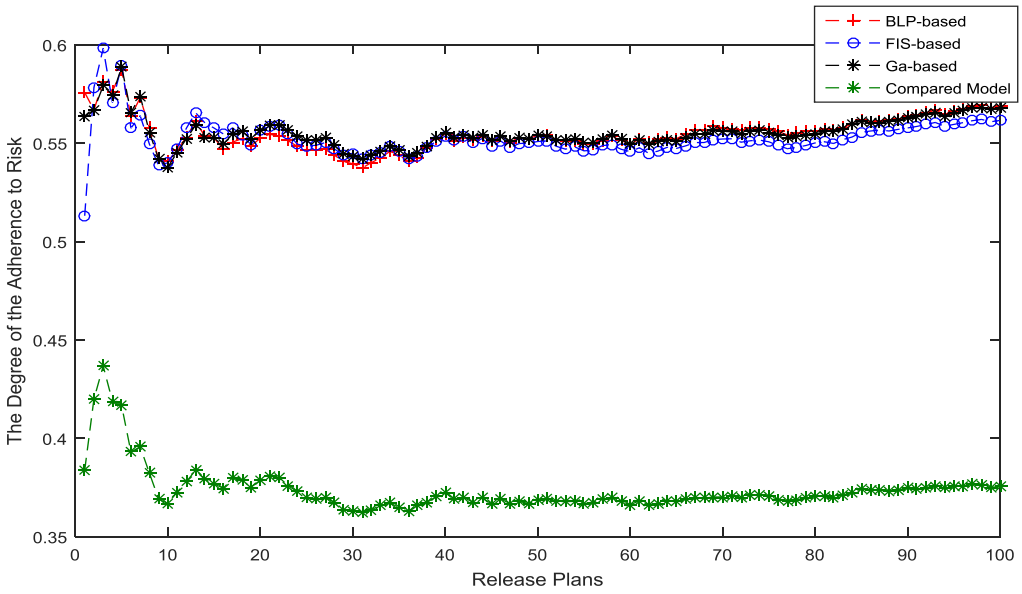


Figure 6.19: Comparison with the Compared Model (Degree of adherence to Risk)

Moreover, in order to show the growth of running time when the number of features is huge, we run an experiment with 500 features and 100 tenants for 50 release plans. Figure 6.20 shows clearly that the proposed FIS approaches has achieved better results in running time than the compared model (5 times faster), and the proposed GA also has achieved better results than the compared model (3 times faster) while the compared model has achieved better results than the proposed BLP (1.5 times faster).

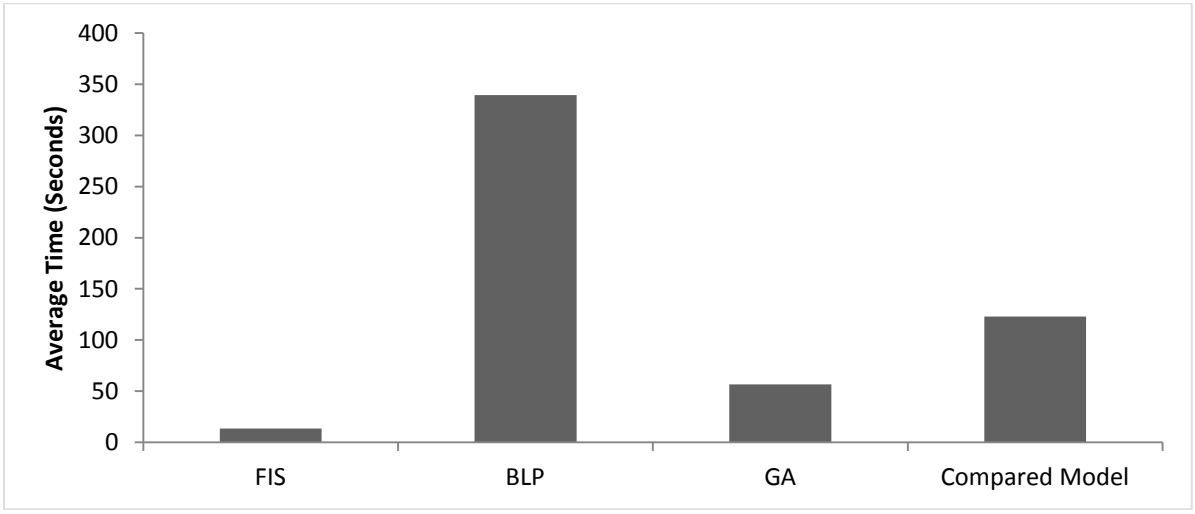


Figure 6.20: Comparison with the Compared Model (Running Time)

6.6 Summary of the Chapter

In general, the proposed approaches are comparable from the perspectives of the degree of overall satisfaction, commonality, and adherence to risk. GA approach is slightly better in the degree of overall satisfaction and commonality when the number of features is small. The BLP approach has achieved higher values than the other two approaches in the degree of overall satisfaction and commonality when the number of features is huge. The BLP has achieved better results in the degree of adherence to risk when the number of features is small. FIS has achieved better results than the other approaches in this measurement when there are a huge number of features. Changing the number of features or tenants does not have noticeable impact on the performance of the three approaches from the perspective of these three metrics. However, the running time of BLP has exponentially grown when the number of features exceeds certain threshold. FIS has shown good performance in the running time. The running time in FIS has slowly grown with increasing the number of features. The time in GA has grown by increasing number of features, but this growth in GA is not as significant as in BLP. In addition, the experiments show increasing number of tenants does not have any negative effects on the running time of the three approaches. The three approaches have shown high degree of similarity in the generated release plans. This explains why the first three metrics are roughly similar for the three approaches. In order to validate the proposed approaches, they are compared to a compared model that has been selected from the literature. The proposed approaches have shown slightly better performance in the degree of overall satisfaction and commonality, and they have achieved much better results in the degree of adherence to risk. Moreover, the running time in the proposed FIS and GA is faster than the running time in the compared model.

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

7.1 Conclusion

Release planning is a core process in the development cycle of multi-tenant Software as a Service (SaaS) applications. The aim of release planning is to assign the most promising features to the release under consideration. Effective release planning can increase the satisfaction of tenants by delivering those features that are important for most of the tenants with high degree of quality. In SaaS applications, It is more efficient to plan only for the next release due to their extreme dynamics, which increase the probability of there being significant changes in tenants' needs over short periods of time. The variables that control SaaS applications include: the importance of each feature as perceived by the different tenants, the decision weights of the tenants, the potential risks along with the required effort that are associated with each feature as estimated by the members of the development team, the available effort allocated to deliver the release as estimated by release management, the technical dependencies among features, contractual constraints which contained in SLA documents, and the degree of commonality of features.

This thesis has provided a formulation for release planning problem for muti-tenant SaaS applications. This formulation aims to I) maximize the degree of tenants' satisfaction by selecting the features that are important to the highest possible number of tenants, II) maximize the degree of commonality by selecting the features that are required by the highest possible number of tenants , and III) minimize the potential risk by selecting the features that have the lowest possible risk. These three objective shall be achieved while taking into account I) contractual constraints where for any feature that is included in the

next release plan there shall be at least one tenant who is eligible to have this feature. II) effort constraint where the required effort to implement the next release shall be less than or equal to the available effort, and III) dependencies constraints where some features have dependent relationship with other features. Two type of dependencies are considered: coupling where it is more beneficial if two features or more delivered in the same release, and the precedence where some features shall be delivered prior to other features.

This thesis proposed three release planning approaches to tackle the "next release" planning problem in SaaS applications. The first approach employs a Fuzzy Inference System (FIS) in order to utilize human expertise to aggregate the evaluations that are provided by the stakeholders and generate a rank for each feature. The rank of a feature represents its priority among other features. After that, two algorithms are applied in order to adjust the ranks of the features to satisfy precedence and coupling constraints. Then, the features are sorted and prioritized using a greedy approach. The features with the highest ranks are assigned to the next release. In the proposed FIS-based approach, linguistic rules are used to implicitly consider human expertise in the planning endeavor. The FIS-based approach consists of four processes: raw data collection, preprocessing, ranking, and release-plan generation. The second approach is an exact optimization approach that utilizes Binary Linear Programming (BLP) in order to optimize the planning process. The BLP-based approach consists of three processes: raw data collection, preprocessing, and release plan generation. The release plan is generated as a set of binary decision variables. Each variable is associated with a feature. If a decision variable is set to 1, then its corresponding feature is assigned to the next release; otherwise, it is postponed to future releases. The third approach is a heuristic optimization approach that employs GA. The objective function

and the constraints that are defined for the BLP approach are used as the fitness function for the GA approach. Because binary variables are used, some restrictions are applied on the proposed GA approach. For example, the equality constraints are transformed to inequality constraints.

In order to validate the effectiveness and efficiency of the proposed approaches, and find out the scenarios that are suited to each approach, experiments are conducted in order to investigate the following aspects: I) The impact of increasing number of features on the performance of the proposed approaches, II) The impact of increasing number of tenants on the performance of the proposed approaches, and III) the performance of the proposed approach when it is compared to the compared model that has been selected from the literature. The experiments show that the proposed approaches have in general comparable performance from the perspective of the degree of overall satisfaction, commonality, and adherence to risk. In more details, The BLP-based approach has shown slightly higher values in the degree of overall satisfaction and commonality than the other two approaches when the number of features is huge. Also it has shown better results compared to the other approaches in the degree of adherence to risk when the number of features is small. The GA-based approach has shown better results than the other two approaches in the degree of overall satisfaction and commonality when the number of features is small. The similarity test shows that there is high similarity between the release plans that are generated by the proposed approaches, which explains the high similarity in the values of the degree of overall satisfaction, commonality, and adherence to risk. In addition, the experiments show that the running time in BLP has exponentially grown when the features exceeds certain threshold, which makes BLP not the best choice when there are huge number of features and

the time is important factor in the release planning. Also, the time in GA has grown by increasing the number of features, but it is not as considerable as the growth in BLP. The FIS approach has shown promising results in running time. The running time in FIS has grown very slowly with increasing the number of features which makes FIS approach suitable when the time is an important factor in the planning process.

The proposed approaches are compared to a model that has been selected from the literature. They have achieved better performance in the degree of overall satisfaction and commonality. The reason behind this result is that when the degree of overall satisfaction and commonality are calculated, the service levels of the tenants are taken into account. The compared model tries to maximize the additive value of the multiplication between the priorities and values of the selected features regardless of the compliance with the service levels of the tenants. Therefore, the compared model may select features that do not have high degree of compliance with the service levels, which is reflected on the values of the degree of overall satisfaction and commonality. Furthermore, the proposed approaches have achieved much better results regarding the degree of adherence to risk. The reason behind this results is that the proposed approaches consider the risk, overall satisfaction, and commonality simultaneously when the features are assigned to a release plan, while the compared model does not consider the risk. Moreover, the growth of running time of the three approaches is compared with the compared model. The experiment shows that the proposed GA and FIS approaches have achieved better results in the running time than the compared model, while the BLP approaches is significantly slower than the compared model selected from the literature.

7.2 Future Work

As a further line of research, the process of collecting raw data for release planning in SaaS application can be investigated. For example, it would be of significant interest to conduct a study to investigate the use of sentiment analysis to measure the degree of tenants' satisfaction about the delivered release of an SaaS application. This data could then be used as input to the planning of the next release. Additionally, the effect of the architecture of SaaS applications on release planning can be investigated. For example, how to consider a request from a tenant for a feature that is provided by a third party. Additionally, the applications of computation with words [72] can be used. The stakeholders' estimates about the attributes of features can be obtained using linguistic terms. In this way, stakeholders can naturally and qualitatively express their opinion about software features. Moreover, the factor of reliability of information can be considered. This can be achieved by adding another variable that measures the reliability of the provided estimates. In order to address the reliability of information, the application of Z-numbers [107] can be used. A Z-number is made up of two linguistic terms. Each term is associated with a fuzzy number. The first component of a Z-number represents a restriction on the values that the evaluated object can take. The second component is a measure of the reliability (sureness) of the first component. Using Z-numbers allows the release management to consider the reliability of stakeholders' estimates in the release planning process.

APPENDIX I: BRANCH AND BOUND ALGORITHM

In BLP, one possible way to find the optimal solution is to enumerate all possible solutions and then to choose the one that is optimal in maximizing (or minimizing) the objective function. The problem with this strategy is that the number of possible solutions increases exponentially with the increasing of number of variables. If there are n variables, then there are 2^n possible solutions. The branch-and-bound algorithm addresses this problem [102]. Branch and bound is a "divide and conquer" strategy, which divides feasible regions into smaller, controllable regions. These new regions are divided recursively into smaller regions until the optimal solution is attained. The algorithm starts by "branching" process. In this process, a binary search tree is created. For each variable x_i two branches are created ($x_i = 1, x_i = 0$). Figure I.1 shows a binary search tree with three variables. After that, at each node, the algorithm solves the problem as a linear programming problem by relaxing the constraints $x_i \in \{0,1\}$ to $0 \leq x_i \leq 1$, where $i = 1 \ldots . n$ and n is the number of variables. In order to find optimal solution using the branch-and-bound algorithm, the following steps are applied:

1. Set current integer solution= $null$.
2. At any node, solve the problem as an LP problem and let Z=the value of the objective function.
3. If the variables are binary, and Z is more optimal than current integer solution then current integer solution=Z.
4. From the solution, pick a variable x_i that does not have an integer value ($0 \leq x_i \leq 1$) , and create two branches (new regions that represent possible solutions) by creating two constraints, $x_i = 0 \; and \; x_i = 1$.

5. Check the feasibility of the new regions (the new braches) and prune (remove) the branch if it does not satisfy the problem constraints.

6. Go to 2.

7. Return the values of current integer solution.

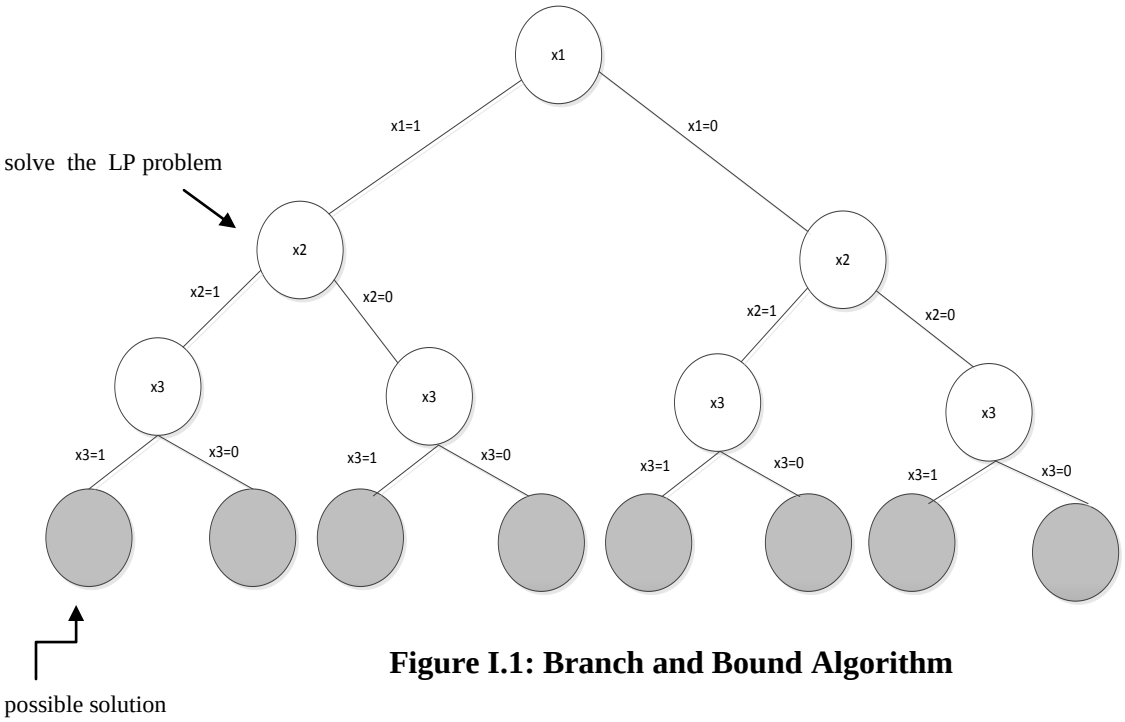


Figure I.1: Branch and Bound Algorithm

APPENDIX II: POLLING METHOD

In polling method, the expert is asked the following question: "do you agree that x is T", where x is a point in the universe of discourse and T is a linguistic term. In other words, each expert is asked to describe the points in the universe of discourse using predefined linguistic terms. After that $\mu(x)_T$ is calculated as the proportion of positive answers over the total number of answers. Let $E = \{e_1, e_2, \dots\dots, e_z\}$ is a set that represents the experts who will participate in defining the membership functions of a linguistic variable V. In order to use polling method, the following steps are applied:

- Define the linguistic values (terms) that the variable V can take. For example, V can take any value in the set $T(V) = \{low, medium, high\}$
- Each expert describes each element in the universe of discourse U using terms in $T(V)$. For this purpose, we define $Describe\_Point(u, e) \in T(V)$, where $u \in U$ and $e \in E$. For example, suppose $U = \{1, \dots 9\}$, $T(V) = \{low, medium, high\}$, and the expert e_2 describes number 4 as medium, then we write $Describe\_Point(4, 2) = medium$
- For $u \in U$ and $t \in T(V)$, $\mu_t(u)$ can be calculated as follows:

$$\mu_t(u) = \frac{\sum_{i=1}^{z}(Describe\_Point(u, i) = t)}{z} \qquad \text{(II. 1)}$$

where z is the number of experts, $t \in T(V)$ and $u \in U$

Example II.1 : $IMPORTANCEVAR$ is a variable that represents the importance of features, such that $T(IMPORTANCEVAR) = \{low, medium, high\}$, $E = \{e_1, e_2, e_3\}$, $U = \{1, \dots 9\}$. Assume that the experts are asked this question:"describe number 3 from the perspective of the importance of software features using the terms in

$T(IMPORTANCEVAR\ )$". Assume that in the following table three experts provide their answers about the degree of importance that number three represents.

Using Equation (II.1), we can see that $\mu_{Low}(3) = 0.66$, $\mu_{medium}(3) = 0.33$, $\mu_{high}(3) = 0$

	Low	medium	high
e1		✓	
e2	✓		
e3	✓		

APPENDIX III: MEMBERSHIP FUNCTIONS OF THE FUZZY VARIABLES

Defining the membership functions of IMPORTANCEVAR (the importance of features)

Universe of discourse	Expert1	Expert2	Expert3	Expert4
1	VL	VL	VL	VL
2	VL	VL	VL	VL
3	L	L	L	VL
4	M	L	M	L
5	M	M	M	M
6	M	M	M	M
7	H	H	H	H
8	H	VH	H	H
9	VH	VH	VH	VH

Defining the membership functions of RISKVAR (the risk of features)

Universe of discourse	Expert1	Expert2	Expert3	Expert4
1	VL	VL	VL	VL
2	VL	VL	VL	VL
3	L	L	L	L
4	M	L	L	M
5	M	M	M	M
6	M	M	M	H
7	H	H	H	H
8	VH	VH	H	H
9	VH	VH	VH	VH
10	VH	VH	VH	VH

Defining the membership functions of EFFORTVAR (the required effort of features)

Universe of discourse	Expert1	Expert2	Expert3	Expert4
1	VL	VL	VL	VL
2	L	VL	L	VL
3	L	L	L	L
4	M	L	L	M
5	M	M	M	M
6	H	M	M	H
7	H	H	H	H
8	H	H	H	VH
9	VH	VH	VH	VH
10	VH	VH	VH	VH

Defining the membership functions of COMMONALITYVAR (the required effort of features)

Universe of discourse	Expert1	Expert2	Expert3	Expert4
0% -10%	VL	VL	VL	VL
11% -20%	L	L	VL	VL
21% - 30%	M	M	L	L
31% - 40%%	M	M	M	M
41% -50%	M	H	H	M
51% - 60%	H	H	H	H
61%- 70%	H	VH	H	H
71- 80%	VH	VH	VH	VH
81-90%	VH	VH	VH	VH
90-100%	VH	VH	VH	VH

APPENDIX IV: FUZZY RULES FOR FIS-BASED APPROACH

IV.1: IMPORTANCE_COMMONALITY_Aggregation Sub Module

1. If (WImportance is VLI) and (Commonality is VLC) then (TEvaluation is VLE)
2. If (WImportance is VLI) and (Commonality is LC) then (TEvaluation is VLE)
3. If (WImportance is VLI) and (Commonality is MC) then (TEvaluation is LE)
4. If (WImportance is VLI) and (Commonality is HC) then (TEvaluation is ME)
5. If (WImportance is VLI) and (Commonality is VHC) then (TEvaluation is ME)
6. If (WImportance is LI) and (Commonality is VLC) then (TEvaluation is VLE)
7. If (WImportance is LI) and (Commonality is LC) then (TEvaluation is LE)
8. If (WImportance is LI) and (Commonality is MC) then (TEvaluation is LE)
9. If (WImportance is LI) and (Commonality is HC) then (TEvaluation is ME)
10. If (WImportance is LI) and (Commonality is VHC) then (TEvaluation is HE)
11. If (WImportance is MI) and (Commonality is VLC) then (TEvaluation is LE)
12. If (WImportance is MI) and (Commonality is LC) then (TEvaluation is ME)
13. If (WImportance is MI) and (Commonality is MC) then (TEvaluation is ME)
14. If (WImportance is MI) and (Commonality is HC) then (TEvaluation is HE)
15. If (WImportance is MI) and (Commonality is VHC) then (TEvaluation is HE)
16. If (WImportance is HI) and (Commonality is VLC) then (TEvaluation is ME)
17. If (WImportance is HI) and (Commonality is LC) then (TEvaluation is ME)
18. If (WImportance is HI) and (Commonality is MC) then (TEvaluation is HE)
19. If (WImportance is HI) and (Commonality is HC) then (TEvaluation is VHE)
20. If (WImportance is HI) and (Commonality is VHC) then (TEvaluation is VHE)
21. If (WImportance is VHI) and (Commonality is VLC) then (TEvaluation is ME)
22. If (WImportance is VHI) and (Commonality is MC) then (TEvaluation is HE)
23. If (WImportance is VHI) and (Commonality is HC) then (TEvaluation is VHE)
24. If (WImportance is VHI) and (Commonality is VHC) then (TEvaluation is VHE)
25. If (WImportance is VHI) and (Commonality is LC) then (TEvaluation is ME)

The meanings of the symbols used in the rules are as follows:

WImportance	the weighted importance for features
Commonality	the commonality of the features
TEvaluation	Tenant related evaluation
VLI	Very Low Importance
LI	Low Importance
MI	Medium Importance
HI	High Importance
VHI	Very High Importance
VLC	Very Low Commonality
LC	Low Commonality
MC	Medium Commonality
HC	High Commonality
VHC	Very High Commonality
VLE	Very Low Evaluation
LE	Low Evaluation
ME	Medium Evaluation
HE	High Evaluation
VHE	Very High Evaluation

IV.2: RISK EFFORT Aggregation Sub Module

1. If (Risk is VLR) and (Effort is VLE) then (DevEval is DVHE)
2. If (Risk is VLR) and (Effort is LE) then (DevEval is DVHE)
3. If (Risk is VLR) and (Effort is ME) then (DevEval is DHE)
4. If (Risk is VLR) and (Effort is HE) then (DevEval is DME)
5. If (Risk is VLR) and (Effort is VHE) then (DevEval is DME)
6. If (Risk is LR) and (Effort is VLE) then (DevEval is DVHE)
7. If (Risk is LR) and (Effort is LE) then (DevEval is DHE)
8. If (Risk is LR) and (Effort is ME) then (DevEval is DHE)
9. If (Risk is LR) and (Effort is HE) then (DevEval is DME)
10. If (Risk is LR) and (Effort is VHE) then (DevEval is DLE)
11. If (Risk is R) and (Effort is VLE) then (DevEval is DHE)
12. If (Risk is R) and (Effort is LE) then (DevEval is DME)
13. If (Risk is R) and (Effort is ME) then (DevEval is DME)
14. If (Risk is R) and (Effort is HE) then (DevEval is DLE)
15. If (Risk is R) and (Effort is VHE) then (DevEval is DLE)
16. If (Risk is HR) and (Effort is VLE) then (DevEval is DHE)
17. If (Risk is HR) and (Effort is LE) then (DevEval is DME)
18. If (Risk is HR) and (Effort is ME) then (DevEval is DLE)
19. If (Risk is HR) and (Effort is HE) then (DevEval is DLE)
20. If (Risk is HR) and (Effort is VHE) then (DevEval is DVLE)
21. If (Risk is VHR) and (Effort is VLE) then (DevEval isDME)
22. If (Risk is VHR) and (Effort is LE) then (DevEval is DME)
23. If (Risk is VHR) and (Effort is ME) then (DevEval is DLE)
24. If (Risk is VHR) and (Effort is HE) then (DevEvl is DVLE)
25. If (Risk is VHR) and (Effort is VHE) then (DevEval is DVLE)

The meanings of the symbols used in the rules are as follows:

Risk	the risk of features
Effort	the required effort for features
DevEval	Development team evaluation
VLR	Very Low Risk
LR	Low Risk
R	Medium Risk
HR	High Risk
VHR	Very High Risk
VLE	Very Low Effort
LE	Low Effort
ME	Medium Effort
HE	High Effort
VHE	Very High Effort
DVLE	Very Low Evaluation
DLE	Low Evaluation
DME	Medium Evaluation
DHE	High Evaluation
DVHE	Very High Evaluation

IV.3: Ranking Sub Module

1. If (DevEval is VLD) and (TEval is VLT) then (Rank is VLR)
2. If (DevEval is VLD) and (TEval is LT) then (Rank is VLR)
3. If (DevEval is VLD) and (TEval is MT) then (Rank is LR)
4. If (DevEval is VLD) and (TEval is HT) then (Rank is MR)
5. If (DevEval is VLD) and (TEval is VHT) then (Rank is HR)
6. If (DevEval is LD) and (TEval is VLT) then (Rank is VLR)
7. If (DevEval is LD) and (TEval is LT) then (Rank is LR)
8. If (DevEval is LD) and (TEval is MT) then (Rank is MR)
9. If (DevEval is LD) and (TEval is HT) then (Rank is MR)
10. If (DevEval is LD) and (TEval is VHT) then (Rank is HR)
11. If (DevEval is MD) and (TEval is VLT) then (Rank is LR)
12. If (DevEval is MD) and (TEval is LT) then (Rank is LR)
13. If (DevEval is MD) and (TEval is MT) then (Rank is MR)
14. If (DevEval is MD) and (TEval is HT) then (Rank is HR)
15. If (DevEval is MD) and (TEval is VHT) then (Rank is VHR)
16. If (DevEval is HD) and (TEval is VLT) then (Rank is LR)
17. If (DevEval is HD) and (TEval is LT) then (Rank is MR)
18. If (DevEval is HD) and (TEval is MT) then (Rank is HR)
19. If (DevEval is HD) and (TEval is HT) then (Rank is VHR)
20. If (DevEval is HD) and (TEval is VHT) then (Rank is VHR)
21. If (DevEval is VHD) and (TEval is VLT) then (Rank is MR)
22. If (DevEval is VHD) and (TEval is LT) then (Rank is MR)
23. If (DevEval is VHD) and (TEval is MT) then (Rank is HR)
24. If (DevEval is VHD) and (TEval is HT) then (Rank is VHR)
25. If (DevEval is VHD) and (TEval is VHT) then (Rank is VHR)

The meanings of the symbols used in the rules are as follows:

DevEval	Development team evaluation
TEval	Tenant related evaluation
Rank	Initial Ranks of features
VLD	Very Low Development team related evaluation
LD	Low Development team related evaluation
MD	Medium Development team related evaluation
HD	High Development team related evaluation
VHD	Very High Development team related evaluation
VLT	Very Low Tenants related evaluation
LT	Low Tenants related evaluation
MT	Medium Tenants related evaluation
HT	High Tenants related evaluation
VHT	Very High Tenants related evaluation
VLR	Very Low Rank
LR	Low Rank
MR	Medium Rank
HR	High Rank
VHR	Very High Rank

APPENDIX V: LIST OF PUBLICATIONS

[1] M. Alrashoud and A. Abhari, "Perception-Based Software Release Planning," *Intelligent Automation & Soft Computing, Taylor & Francis*, vol. 21(2), pp. 175-195, 2015.

[2] M. Alrashoud, L. Ahmed and A. Abhari, "Binary linear programming-based release planning for multi-tenant business SaaS," *in Proceedings of the International Conference on Computer Science & Software Engineering*, Montreal, 2014, pp. 118-125.

[3] M.Alrashoud, M.AlMeshary, A.Abhari, " Automatic Validation for Multi Criteria Decision Making Models in Simulation Environments," *in Communications and Networking Simulation Symposium (CNS), Spring Simulation Multi-Conference*, SpringSim2015, 2015, in press.

REFERENCES

[1] G. Ruhe, *Product Release Planning: Methods, Tools and Applications.* Alberta, Canada, CRC Press, 2010.

[2] G. Ruhe and S. L. Pfleeger, "Software engineering decision support," *in 40th Annual Hawaii International Conference On System Sciences*, Hawaii , HICSS 2007, 2007, pp. 282-282.

[3] G. Ruhe, "Software engineering decision support–a new paradigm for learning software organizations," *in Advances in Learning Software Organizations*, Springer, pp. 104-113, 2003.

[4] P. Mell, T. Grance, The NIST definition of cloud computing, *Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology,* Gaithersburg, MD. 2011

[5] T. Erl, R. Puttini and Z. Mahmood, *Cloud Computing: Concepts, Technology, & Architecture*. Pearson Education, 2013.

[6] B. Sengupta and A. Roychoudhury, "Engineering multi-tenant software-as-a-service systems," *In Proceedings of the 3rd International Workshop on Principles of Engineering Service-Oriented Systems* , New York, PESOS '11, NY, USA, 2011, pp. 15-21.

[7] C. Fry and S. Greene, "Large scale Agile transformation in an on-demand world," *in Agile Conference*, Washington, DC, AGILE, 2007, pp. 136-142.

[8] D. Greer and G. Ruhe, "Software release planning: an evolutionary and iterative approach," *Information and Software Technology*, vol. 46, pp. 243-253, 2004.

[9] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of Network and Computer Applications*, vol. 34, pp. 1-11, 2011.

[10] H. Takabi, J. B. D. Joshi and G-J.Ahn, "Security and Privacy Challenges in Cloud Computing Environments," *IEEE Security & Privacy*, vol. 8, pp. 24-31, 2010.

[11] H. J. La and S. D. Kim, "A systematic process for developing high quality SaaS cloud services," *Lecture Notes in Computer Science*, vol. 5931, pp. 278-289, 2009.

[12] T.Unger, R.Mietzne, and F. Leymann, "Customer-defined service level agreements for composite applications," *Enterprise Information Systems*, vol. 3, pp. 369-391, 2009.

[13] W.Sun, X.Zhang, C. J. Guo, P.Sun and H.Su, "Software as a service: Configuration and customization perspectives," *in Congress on Services Part II*, IEEE SERVICES-2., 2008, pp. 18-25.

[14] O.Saliu, G.Ruhe, "Software release planning for evolving systems,"*Innovations in Systems and Software Engineering*, vol. 1, pp. 189-204, 2005.

[15] S. Jantunen, L. Lehtola, D. C. Gause, U. R. Dumdum and R. J. Barnes, "The challenge of release planning," *IEEE Fifth International Workshop on Software Product Management*, Trento., IWSPM, 2011, pp. 36 - 45.

[16] K. Beck et al (2001), "Manifesto for Agile Software Development", Available on line http://agilemanifesto.org/.

[17] G. Kulkarni, P. Chavan, H. Bankar, K. Koli and V. Waykule, "A new approach to software as service cloud," *in 7th International Conference On Telecommunication Systems, Services, and Applications (TSSA)*, 2012, pp. 196-199.

[18] H. J. La and S. D. Kim, "A systematic process for developing high quality SaaS cloud services," *Lecture Notes in Computer Science*, vol. 5931, pp. 278-289, 2009.

[19] C. -. Guo, W. Sun, Z. -. Jiang, Y. Huang, B. Gao and Z. -. Wang, "Study of Software as a Service Support Platform for Small and Medium Businesses," *Lecture Notes in Business Information Processing*, vol. 74, pp. 1-30, 2011.

[20] R. Mietzner, A. Metzger, F. Leymann and K. Pohl, "Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications," *in ICSE Workshop on Principles of Engineering Service Oriented Systems, PESOS*, 2009, pp. 18-25.

[21] K. Schwaber, "Scrum development process," *in OOPSLA'95 Workshop Proceedings on Business Object Design and Implementation: Springer-Verlag*, 1997, pp. 117-134.

[22] M. Block, "Evolving to Agile: A story of Agile adoption at a small SaaS company," *in Agile Conference (AGILE)*, 2011, 2011, pp. 234-239.

[23] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2000.

[24] S.Tariq, M.Mohsin N. Saleemi and F.Saleemi, "Enhancement of XP for Cloud Application Development," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 3, pp. 295-301, 2012.

[25] R. Benefield, "Agile deployment: Lean service management and deployment strategies for the SaaS enterprise," *in 42nd Hawaii International Conference on System Sciences, HICSS '09*, 2009, pp. 1-5.

[26] G.Carraro, F. Chong, and E.Pace “, "Efficient Software Delivery Through Service-Delivery Platforms," *The Architecture Journal, Microsoft MSDN Architecture Center*, available on https://msdn.microsoft.com/en-us/library/bb735303.aspx,2007.

[27] R. Mietzner and F. Leymann, "Generation of BPEL customization processes for SaaS applications from variability descriptors," *in IEEE International Conference on Services Computing*, SCC 2008, 2008, pp. 359-366.

[28] J. Diane, E. John, "Web services business process execution language version 2.0," *OASIS standard*, available on https://www.oasisopen.org/committees/tc_home.php?wg_abbrev=wsbpel, accessed on 2007.

[29] W. -. Tsai and X. Sun, "SaaS multi-tenant application customization," *in IEEE 7th International Symposium on Service-Oriented System Engineering, SOSE 2013*, 2013, pp. 1-12.

[30] K. Logue and K. McDaid, "Handling uncertainty in Agile requirement prioritization and scheduling using statistical simulation," *in Agile Conference, AGILE'08*, 2008, pp. 73-82.

[31] P. Carlshamre, "Release planning in market-driven software product development: Provoking an understanding," *journal of Requirements Engineering*, vol.7, pp. 139-151, 2002.

[32] G. Ruhe, "A systematic approach for solving the wicked problem of software release planning," *journal of Soft Computing*, vol. 12, pp. 95-108, 2008.

[33] G. Ruhe and M. O. Saliu, "The art and science of software release planning," *IEEE Software*, vol. 22, pp. 47-53, 2005.

[34] A. Al-Emran, P. Kapur, D. Pfahl and G. Ruhe, "Studying the impact of uncertainty in operational release planning–An integrated method and its initial evaluation," *Information and Software Technology*, vol. 52, pp. 446-461, 2010.

[35] M. Cohn, *Agile Estimating and Planning*. Prentice Hall, 2005.

[36] K. Logue and K. McDaid, "Agile release planning: Dealing with uncertainty in development time and business value," *in Proceedings - Fifteenth IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS 2008*, 2008, pp. 437-442.

[37] G. Ruhe and A. N. The, "Hybrid intelligence in software release planning," *International Journal of Hybrid Intelligent Systems*, vol. 1, pp. 99-110, 2004.

[38] M. van den Akker, S. Brinkkemper, G. Diepen and J. Versendaal, "Software product release planning through optimization and what-if analysis," *Information and Software Technology*, vol. 50, pp. 101-111, 2008.

[39] F. G. Freitas, D. P. Coutinho and J. T. Souza, "Software next release planning approach through exact optimization," *International Journal of Computer Applications*, vol. 22, pp. 1-8, 2011.

[40] M. I. Ullah and G. Ruhe, "Towards comprehensive release planning for software product lines," *in Proceedings of the First International Workshop on Software Product Management, IWSPM'06*, 2006, pp. 51-55.

[41] C. Li, M. van den Akker, S. Brinkkemper and G. Diepen, "An integrated approach for requirement selection and scheduling in software release planning," *Journal of Requirements Engineering*, vol. 15, pp. 375-396, 2010.

[42] K. Deb and K. Pal, "Efficiently solving: A large-scale integer linear program using a customized genetic algorithm," *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3102, pp. 1054-1065, 2004.

[43] H. Holland, *Adaptation in Natural and Artificial systems*, University of Michigan Press, Ann Arbor, 1975.

[44] An Ngo-The and G. Ruhe, "Optimized Resource Allocation for Software Release Planning," *IEEE Transactions On Software Engineering* , vol. 35, pp. 109-123, 2009.

[45] T. L. Satty, *The analytic hierarchy process*. New York: McGraw Hill, 1980.

[46] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," *IEEE Software*, vol. 14, pp. 67-74, 1997.

[47] P. Bajaj and V. Arora. "Multi-person decision-making for requirements prioritization using fuzzy AHP," *ACM SIGSOFT Software Engineering Notes,* vol 38, pp. 1-6. 2013.

[48] E. Tsang, *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego, 1995.

[49] B. Regnell and K. Kuchcinski, "Exploring software product management decision problems with constraint solving-opportunities for prioritization and release planning," *in Fifth International Workshop On Software Product Management (IWSPM)*, 2011, pp. 47-56.

[50] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338-353, 1965.

[51] M. Laviolette and J. W. Seaman Jr., "The efficacy of fuzzy representations of uncertainty," *IEEE Transaction on Fuzzy Syst*, vol. 2, pp. 4-15, 1994.

[52] L. A. Zadeh, "Generalized theory of uncertainty (GTU) - Principal concepts and ideas," *journal of Advances in Soft Computing*, vol. 37, pp. 3-4, 2006.

[53] R.Bellman and L. A. Zadeh, "Decision-Making in a Fuzzy Environment," *journal of Management Science*, vol. 17, pp. b-141-b-164, 1970.

[54] W. Shen, "Software release planning with fuzzy objectives and constraints," M.S. thesis, Dept of Electrical and Computer Engineering, University of Calgary, Calgary, Canada, 2005.

[55] An Ngo-The and M. O. Saliu, "Fuzzy structural dependency constraints in software release planning," *in the 14th IEEE International Conference on Fuzzy Systems, FUZZ '05*, 2005, pp. 442-447.

[56] An Ngo-The, G. Ruhe and W. Shen, "Release planning under fuzzy effort constraints," *in the Third IEEE International Conference On Cognitive Informatics*, 2004, pp. 168-175.

[57]Expert Decisions Inc, ReleasePlanner, available on https://www.releaseplanner.com/rpApp/, 2004.

[58] Nethercote, P. Stuckey, R. Becket, S. Brand, G. Duck, and G. Tack, "Minizinc: Towards a standard cp modeling language," *Lecture Notes in Computer Science, C. Bessiere, Ed. Springer Berlin /Heidelberg*, vol. 4741, pp. 529-543, 2007.

[59] ScrumDo available on http://www.scrumdo.com/, 2015.

[60] AgileTrack Software, LLC, AgileTrack, available on http://agiletracksoftware.com/, 2014.

[61] VersionOne Inc, VersionOne SaaS application, available on http://www.versionone.com/, 2002.

[62] H. -. Lee, "Group decision making using fuzzy sets theory for evaluating the rate of aggregative risk in software development," *journal of Fuzzy Sets System*, vol. 80, pp. 261-271, 1996.

[63] N. Bajaj, A. Tyagi and R. Agarwal, "Software estimation: a fuzzy approach," *ACM SIGSOFT Software Engineering Notes*, vol. 31, pp. 1-5, 2006.

[64] J. Wang and Y. Lin, "A fuzzy multi-criteria group decision making approach to select configuration items for software development," *journal of Fuzzy Sets Syst*em, vol. 134, pp. 343-363, 2003.

[65] S. -. Chen, "Fuzzy group decision making for evaluating the rate of aggregative risk in software development," *journal of Fuzzy Sets Syst*em, vol. 118, pp. 75-88, 2001.

[66] C. K. Kwong and H. Bai, "A fuzzy AHP approach to the determination of importance weights of customer requirements in quality function deployment," *Journal of Intelligent Manufacturing*, vol. 13, pp. 367-377, 2002.

[67] E. Praynlin and P. Latha, "Software development effort estimation using ANFIS," *journal of Information (Japan)*, vol. 17, pp. 1325-1337, 2014.

[68] A.Amindoust and A.Saghafinia, "Supplier evaluation using fuzzy inference systems," *Studies in Fuzziness and Soft Computing*, vol. 313, pp. 3-19, 2014.

[69] A. C. Kutlu, H. Behret and C. Kahraman, "A fuzzy inference system for multiple criteria job evaluation using fuzzy AHP," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 23, pp. 113-133, 2014.

[70] I. Palomares, R. M. Rodríguez and L. Martínez, "An attitude-driven web consensus support system for heterogeneous group decision making," *journal of Expert System Applications*, vol. 40, pp. 139, 2012.

[71] H. Ziv, D. Richardson and R. Klösch, "The uncertainty principle in software engineering," the *19th International Conference on Software Engineering (ICSE'97)*, 1997.

[72] L. A. Zadeh, "From computing with numbers to computing with words. From manipulation of measurements to manipulation of perceptions," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 46, pp. 105-119, 1999.

[73] T. Elsayed, "Fuzzy inference system for the risk assessment of liquefied natural gas carriers during loading/offloading at terminals," *journal of Applied Ocean Research*, vol. 31, pp. 179-185, 2009.

[74] K. E. Wiegers and J. Beatty, *Software Requirements*. Microsoft Press, 2013.

[75] "The 7 secrets of SaaS startup success," *Salesforse white papers*, available on https://www.salesforce.com/assets/pdf/misc/WP_7Secrets_0408.pdf, 2008.

[76] O. Saliu and G. Ruhe, "Supporting software release planning decisions for evolving systems," *in Proceedings of the 2005 29th Annual IEEE/NASA Software Engineering Workshop, SEW'05*, 2005, pp. 14-24.

[77] Z. Yue, "An extended TOPSIS for determining weights of decision makers with interval numbers," *journal of Knowledge-Based System*. vol 24, pp. 146-153. 2011.

[78] G. Ruhe and D. Greer, "Quantitative studies in software release planning under risk and resource constraints," *in International Symposium on Empirical Software Engineering, ISESE 2003*, 2003, pp. 262-270.

[79] C. Chittister and Y. Y. Haimes, "Risk associated with software development: a holistic framework for assessment and management," *IEEE Transaction on System. Man Cybern*, vol. 23, pp. 710-723, 1993.

[80] A. Benlian and T. Hess, "Opportunities and risks of software-as-a-service: Findings from a survey of IT executives," *journal of Decision Support System*, vol 52, pp. 232. 2011.

[81] A. Sharma and D. S. Kushwaha, "A Complexity measure based on Requirement Engineering Document,"*International Conference on Computer and Communication Technology (ICCCT)*, 2010, pp 608-615.

[82] V.Viji, J. Dhanalakshmi, and S. Sahadev., "Software team skills on software product quality," *Asian Journal of Information Technology*, vol. 8, pp. 8-13, 2009.

[83] B. W. Boehm, "Software risk management: principles and practices," *IEEE Software*, vol. 5, pp. 32-41, 1991.

[84] L.Wu, S. K. Garg and R. Buyya, "SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments," *in 11th IEEE/ACM International Symposium On Cluster, Cloud and Grid Computing (CCGrid)*, 2011, pp. 195-204.

[85] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell and J. Natt och Dag, "An industrial survey of requirements interdependencies in software product release planning," *in Fifth IEEE International Symposium on Requirements Engineering*, 2001, pp. 84-91.

[86] C. Ebert and J. De Man, "Requirements uncertainty: Influencing factors and concrete improvements," *in 27th International Conference on Software Engineering, ICSE 2005*,2005, pp. 553-560.

[87] S. Cateni, M.Vannucci and V. Colla, "Industrial multiple criteria decision making problems handled by means of fuzzy inference-based decision support systems," *in 4th International Conference on Intelligent Systems Modelling & Simulation (ISMS)*, 2013, pp. 12-17.

[88] H. Vasudevan, N. C. Deshpande and R. R. Rajguru, "Multi criteria decision making using fuzzy inference system: A case in manufacturing," *in International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, 2014, pp. 1280-1286.

[89] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, vol. 7, pp. 1-13, 1975.

[90] A.Kaufmann, and M. M. Gupta, *Introduction to Fuzzy Arithmetic: Theory and Applications*. Arden Shakespeare, 1991.

[91] L. A. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning—I," *journal of Information Science*, vol. 8, pp. 199-249, 1975.

[92] T. J. Ross, *Fuzzy Logic with Engineering Applications*. England: Willy, 2005.

[93] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-15, pp. 116-132, 1985.

[94] M. Setnes, R. Babuska, U. Kaymak and H. R. van Nauta Lemke, "Similarity measures in fuzzy rule base simplification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol.28, pp. 376-386. 1998.

[95] A. Hamam and N. D. Georganas. "A comparison of mamdani and sugeno fuzzy inference systems for evaluating the quality of experience of hapto-audio-visual applications," *IEEE International Workshop on Haptic Audio visual Environments and Games, HAVE 2008*, 2008, pp 87 - 92.

[96] W. L. Tung and C. Quek, "A mamdani-takagi-sugeno based linguistic neural-fuzzy inference system for improved interpretability-accuracy representation," *in IEEE International Conference on Fuzzy Systems., FUZZ-IEEE,*2009, pp. 367-372.

[97] A. Sancho-Royo and J. L. Verdegay, "Methods for the Construction of Membership Functions," *International Journal of* Intelligent *Systems*, vol. 14, pp. 1213-1230, 1999.

[98] W.Di, X. Zeng, and J. Keane, "A survey of hierarchical fuzzy systems," *International Journal of Computational Cognition*, vol. 4, pp. 18-29, 2006.

[99] R. R. Yager, "On a hierarchical structure for fuzzy modeling and control," *IEEE Transactions* on *Systems, Man*, and *Cybernetics*, vol. 23, pp. 1189-1197, 1993.

[100] C. Wei and L. Wang, "A note on universal approximation by hierarchical fuzzy systems," *journal of Information Science*, vol.123, pp. 241-248, 2000.

[101] J. M. Van Den Akker, S. Brinkkemper, G. Diepen and J. Versendaal, "Determination of the next release of a software product: An approach using integer linear programming," *in Proceeding of the 11th International Workshop on Requirements Engineering*, 2005, pp. 119-124.

[102] J. W. Chinneck, *PRACTICAL OPTIMIZATION: A GENTLE INTRODUCTION.* Carleton University, 2012.

[103] S. N. Sivanandam and S. N. Deepa. *Introduction to Genetic Algorithms*. Springer Science & Business Media, 2007.

[104] K.Deep, K. P.Singh, M.L. Kansal, and C. Mohan, "A real coded genetic algorithm for solving integer and mixed integer optimization problems," *journal of Applied Mathematics and Computation*, vol.212, pp. 505–518, 2009.

[105] A. Al-Emran, D. Pfahl, and G. Ruhe, "Decision support for product release planning based on robustness analysis," *in 18th IEEE International Requirements Engineering Conference*, 2010, pp. 157-166

[106] L. M. Laird, M. C.Brennan, *Software Measurement and Estimation: a Practical Approach* .Wiley-IEEE Computer Society Press, 2006.

[107] L. A. Zadeh, "A Note on Z-numbers," *journal of Information Science*, vol. 181, pp. 2923-2932, 2011.

[108] A.J.Bagnall, A.J. Rayward-Smith, and I.M. Whittley, "The next release problem," *Information and software technology,* vol. 43, pp.883-890.2001.

[109] Z. Yuanyuan, M. Harman, and S.A. Mansouri, "The multi-objective next release problem," *Proceedings of the 9th annual conference on Genetic and evolutionary computation. ACM,* 2007, pp.1129-1137.

[110] J.J.Durillo, Y. Zhang, E.Alba, M.Harman, and A.J.Nebro, "A study of the bi-objective next release problem," *Empirical Software Engineering*, vol.16, pp. 29-60, 2011.