Binary Linear Programming-based Release Planning for Multi-tenant Business SaaS

Mubarak Alrashoud
Department of Computer Science,
Ryerson University
350 Victoria Street, Toronto,
Ontario,M5B 2K3, Canada
mubarak.alrashoud@ryerson.ca

Lubaid Ahmed
Department of Computer Science,
Ryerson University
350 Victoria Street, Toronto,
Ontario,M5B 2K3, Canada
lubaid@scs.ryerson.ca

Abdolreza Abhari
Department of Computer Science,
Ryerson University
350 Victoria Street, Toronto,
Ontario,M5B 2K3, Canada
aabhari@scs.ryerson.ca

ABSTRACT
In multi-tenant Software as a Service (SaaS) business software, the degree of tenants' satisfaction is a significant indicator of the success of the SaaS system. Tenants' satisfaction can be achieved by continuously fulfilling their evolving needs. Usually, SaaS providers frequently deliver new releases of the application. Each release contains new or enhanced features. However, SaaS providers have limited resources, which makes it difficult to them to incorporate all of the tenants' requests in the next release. Therefore, some requirements shall be postponed to later releases. In order to achieve the highest possible level of tenets' satisfaction, SaaS providers shall include the most common requirements in the next release, which guarantee the satisfaction of highest possible number of tenants with less effort. Additionally, tenants' priorities and preferences about the requirements must be considered. Besides maximizing tenants' satisfaction, it is crucial to meet different types of constraints such as resource, technical, and contractual constraints. This paper identifies the factors that govern the release planning process for multi-tenant business software, which are contractual constraints, commonality of requirements, tenants' preferences and decision weights, risk, technical constraints. The first two factors are suggested by this paper, while the remaining factors are inherited from the traditional release planning process. Moreover, this paper proposes a framework that deals with the uniqueness of the release planning process in multi-tenant SaaS system. In this framework, Binary Linear Programming (BLP) is employed to optimize the selection process of the requirements that will be implemented in the next release. An experiments section is provided to illustrate the degree of satisfaction that can be achieved using the proposed framework.

Categories and Subject Descriptors
D.2.9 [**Software Management**]: Software development, Software process.

G.1.6 [**Optimization**]: Constrained optimization, Integer programming

C3S2E '14, August 03 - 05 2014, Montreal, QC, Canada
Copyright 2014 ACM 978-1-4503-2712-1/14/08…$15.00.
http://dx.doi.org/10.1145/2641483.2641542

General Terms
Management, Measurement, Human Factors.

Keywords
Software release planning, requirements prioritization, SaaS development, SaaS requirements engineering, applications of binary linear programming

1. INTRODUCTION
In the last few years, cloud computing has emerged as a computation model that increases resources utilization, eliminates the cost of hardware expenses and software licenses, and provides on-demand computation resources. Multi-tenant Software as a Service (SaaS) is a cloud service model where many clients (called tenants) use thin client software (for example web browser) to access SaaS applications, which run on a cloud infrastructure [1]. Multi-tenant SaaS systems are beneficial to both the SaaS providers and the tenants. Because SaaS providers offer their services via the internet, they can serve large numbers of tenants through a single shared and centralized instance of the application, thus increasing revenue [2]. In typical SaaS applications, The SaaS provider is required to manage only one single codebase, which decreases the cost of the upgrade and maintenance of the application. From the perspective of the tenants, SaaS applications help to eliminate the cost of software licenses, infrastructure' cost and reduce management and maintenance overhead. Nowadays, many successful Multi-tenant SaaS business software exist in the market (for example, Salesforce, NetSuite). The success of these SaaS applications is due to their popularity. High popularity can be achieved by continuously fulfilling the different requirements of the tenants [3, 4], which are extremely evolving due to the dynamics of the market. Therefore, SaaS providers frequently deliver new releases of the application, in order to cope with the tenants and market's needs as figure 1 shows. For example, Salesforce delivers four major releases per year [5]. However, when developing a new release, it is not possible for the SaaS provider to incorporate all the required changes in one release. In other words, usually SaaS provider can implement fewer requirements than required. This is due to resources and technical constraints. Thus, it is significant to SaaS providers to select a set of features among all those requested that will maximize the tenants' satisfaction, while taking into account the resources, technical, and contractual constraints [6]. To achieve this goal, the release planning (RP) process is conducted as a primary step in the development cycle of each new release. The output of release planning is the release plan, which includes the list of the most promising features that will be implemented in the next release. To construct an effective release plan for a SaaS application, many variables must be taken

into account, such as different tenants' preferences and priorities, the required and available resources, risk, the technical relationships among requirements. In addition to these factors that are inherited from traditional release planning process, this paper suggests two additional factors, which are commonality of requirements, and contractual constraints which are represented by Service Level Agreement (SLA) document. All of these variables are discussed in details in the further sections. Release planning is an optimization problem [7,8,9]; therefore, integral linear programming-based framework is proposed in this paper to optimize the selection of the requirements. To the best of our knowledge, no research works have discussed the release planning process in the business SaaS systems

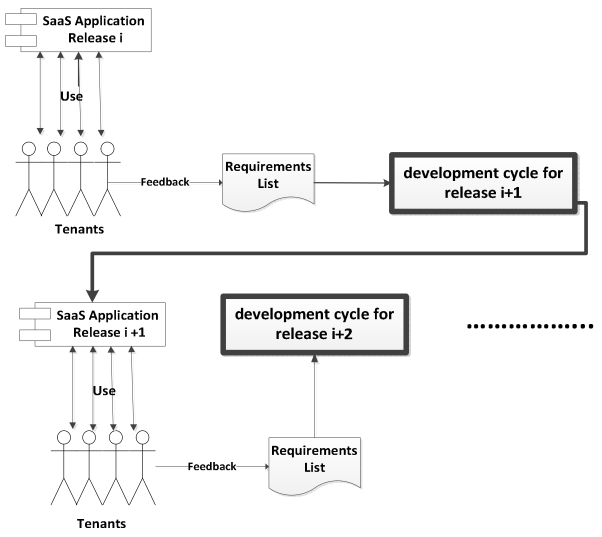
SaaS Application Release i
Use
Tenants
Feedback
Requirements List
development cycle for release i+1
SaaS Application Release i +1
development cycle for release i+2
....
Use
Tenants
Feedback
Requirements List

Figure 1. Continues Delivery of the SaaS Application

The rest of the paper is organized as follows: related work is shown in section 2. The problem statement for release planning process for SaaS applications is defined in section 3. In section 4, the proposed solution is presented. Section 5 shows the applicability and the validation of the proposed model, and finally the conclusions are shown in section 6.

2. RELATED WORK

The release planning problem has been discussed by many authors. Ruhe and Saliu [7] state that release planning is an ill-defined problem; therefore, in addition to mathematical models, human intuition should play a significant role in resolving release planning problems. They present two approaches for release planning. The first approach relies on human factors, such as communication and the human capabilities to resolve conflicts, and they call this approach "the art of release planning." The second approach uses integer linear programming to model the problem, and they call this approach "the science of release planning." They combine these two approaches to present a solution for RP. Ruhe and Ngo-The [8] present an approach called OPTIMIZERASORP to allocate available human resources with different degrees of productivity to a group of tasks required to implement a set of features assigned to a certain release. OPTIMIZERASORP utilizes the strength of integer linear programming and genetic algorithm to construct the plans for resource allocation. Greer and Ruhe [9] state that three main factors should be considered in release planning: technical precedence, conflicts of stakeholders' priorities, and available resources. These factors are considered in the method they propose, which they call EVOLVE. It employs genetic algorithm to perform continuous planning during incremental software development. Ruhe and Ngo-The [6] has improved the EVOLVE method to EVOLVE*, which consists of three main phases: modeling, exploration, and consolidation. In contrast with the earlier method, EVOLVE* plans only for the next two releases in advance, and it gives high consideration to stakeholders' satisfaction. EVOLVE* considers three factors in developing a solution: time, benefit and quality. In [10] EVOLVE* is extended to FUZZY-EVOLVE*. The estimates of the available and the required resources are provided as fuzzy numbers. Then the comparison of fuzzy number methods is used in order to find how much the resource constraints are satisfied. Similarly, the degree of satisfying the objective function is represented as membership function. After that, the fuzzy objective and the fuzzy resource constraints are aggregated in order to obtain the sub set of the requirements that achieves the optimal satisfaction degree. Akker et al. [11] use integer linear programming to solve the release planning problem. The goal of their proposed technique is to generate a release plan that maximizes projected revenue of the software product using available resources in a specific time period. The authors present two models: the first schedules the development of the requirements in a way that minimizes the project duration; the second aims to maximize revenues, and calculates an on-time-delivery project schedule by combining requirement selection and scheduling. Ullah and Ruhe [12] discuss how to formally model release planning for Software Product Line (SPL). They have identified the specific variables that govern release planning in SPL (for example, resolving the conflicts between the evolution of core assets and the requirements of various products). Ngo-The and Ruhe [13] state that release planning can be seen as a "wicked planning problem," which means that they believe that it is difficult or impossible to completely formalize release planning. Furthermore, they state that the solutions of release planning can be categorized under good-or-bad solutions rather true-or-false ones. The authors categorize the constraints in release planning under "hard constraints," such as budget and technological issues, and "soft constraints" such as risk and resource consumption. They introduced EVOLVE+, which is an extension to EVOLVE*. EVOLVE+ can manipulate the soft constraints and objectives in the decision making process. It generates several candidate solutions, and by using ELECTRE (which is a multi-criteria decision aid method); the decision maker can select the final solution. Colares et al. [14] formalize release planning problems as multi-objective search problems. The objectives are to maximize the stakeholders' satisfaction and to minimize the project risks. Colares et al. take many factors into account, including stakeholders' satisfaction, costs, deadlines, available resources, efforts needed, risks management and requirements interdependencies. Their approach generates a number of suggested high quality solutions, and the decision makers select the most appropriate one. In [15] Ngo-The and Saliu model dependency constraints between requirements as fuzzy relations. Two relations are defined: the coupling and the precedence relationships. Coupling relationship occurs when a group of requirements should be implemented in the same release. Precedence relationship occurs when a requirement should be implemented prior to other requirement(s). In earlier stages of the project, these relations are not clear. Therefore, for a certain solution, which is the set of suggested requirements assigned to certain release, the degree of satisfaction with respect to the dependency relations is calculated using fuzzy membership functions. Ngo-The and Ruhe [16] use fuzzy logic to handle the uncertainty in the estimated and available efforts. Also they use fuzzy logic to model the satisfaction objective function by considering uncertainty in cost, benefits, and quality factors. Karlsson and Ryan use [17] Analytical Hierarchal Approach (AHP) for prioritizing requirements based on their estimated cost values. Users and development teams apply AHP's pairwise comparison to measure the relative values and the relative costs of candidate requirements. After that, the results of the AHP evaluation are plotted on a cost-value diagram.

As it is clear from above illustration, no research has been conducted related to the release planning process for SaaS applications. The unique characteristics of SaaS applications, such as seamless delivery, SLA, and internet-related risk, and huge number of customers have produced new variables that must be counted in the release planning process. These new variables are taken into account in the proposed model. In the next section, these variables are defined and discussed in details.

3. PROBLEM STATMENT

Let us define a set $T = \{t_1, t_2, \ldots\ldots, t_m\}$, which represents the tenants who use the SaaS application. While using the SaaS application, the tenants need to add or modify requirements. Let us define a family of sets of requirements $R = \{Rt_1, Rt_2, \ldots\ldots, Rt_m\}$, where Rt_i represents the requirements which are needed by tenant t_i. A correlation process is performed to come up with a unified new set that contains all requirements; let $R^* = \bigcup_{i=1}^{m} Rt_i = \{r_1, r_2 \ldots r_n\}$, where n is the total number of requirements such that $n \leq \sum_{i=1}^{m} |Rt_i|$ and $|Rt_i|$ is the cardinality of the set Rt_i. The SaaS management wants to construct the release plan of the next release by selecting the requirements that maximize the degree of tenants' satisfaction while taking into account the resources, technical, and contractual constraints. Let us represent the selected requirements (the release plan) as the set $R^{\wedge} = \{ r_1^{\wedge}, r_2^{\wedge}, \ldots . r_w^{\wedge} \}$, where $R^{\wedge} \subset R^*$. We define $G_{R^{\wedge}}$ as the characteristic function of the set $R^{\wedge}$ which means

$$G_{R^{\wedge}}(r_i) = \begin{cases} 1 & \text{for } r_i \in R^{\wedge} \\ 0 & \text{for } r_i \notin R^{\wedge} \end{cases}$$

The planning is only for the next release because of the extreme dynamics in SaaS applications, which highly increase the possibility of having extreme changes of requirements in a short time period.

Decision variables

Let us represent the release plan 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 requirement r_i is assigned to the next release; otherwise, it is postponed to a future release. This is equivalence to $x_i = G_{R^{\wedge}}(r_i)$.

Preferences of Tenants

The tenants provide their estimates about the importance of each requirement. Those estimates reflect the priority and perspective of each tenant on each requirement. To capture tenants' evaluation, we define $n \times m$ matrix, and let us call it E where the element e_{ij} is an integer that denotes the importance of requirement r_i from the perspective of tenant t_j, and $1 \leq e_{ij} \leq 10$, such that 1 and 10 are the lowest and the highest degrees of importance respectively.

Decision Weights of the Tenants

In many cases, some tenants are given higher decision weights than others. That is because of their loyalty or their volume of trade. Hence, in order to reflect the real importance of the requirements, the provided estimates are linked with the decision weights of the different tenants. Let W is a vector contains the decision weights of the different tenants. w_i is the weight of tenant t_i such that:

$$\sum_{i=1}^{m} w_i = 1$$

The decision weight can be extracted by many ways;for example, in [9] AHP is used. For sake of simplicity, this paper assumes that decision weights are given.

Contractual constraints (Service Level Agreement (SLA))

SLA is a document that describes the quality-of-service (QoS) attributes of the provided service, such as response time, capacity, up-time, etc. SLA shows the limitations and the guarantees of the provided service [2]. In multi-tenants SaaS, it is possible to have different tenants with different QoS requirements, which causes different SLAs. For example, one tenant is interested in having low-price service regardless of the performance, while other tenants need high performance service regardless of the cost. In order to meet the SLA when a tenant requests to add or modify a requirement, the SaaS provider should verify if the non-functional aspect of this requirement complies with the SLA of that tenant. For example, it is possible for a tenant to request enhancing a requirement, and this enhancement needs to add more processing time. In this case, the SaaS provider should verify whether the required additional processing time complies with the processing time found in SLA of that tenant. Let us take another example that is used in [18]. Suppose a SaaS provider has three types of services (Standards, Professional and Enterprise), and there is one tenant who has a subscription in the standard type service. After using the service, the tenant discovered that she/he needs more functionality. She/he sends a request to the SaaS provider to add these functions. The SaaS provider found that these features are only found in the professional levels, and only the subscribers in this level can ask to change or enhance them. In this case, the SaaS provider asks the tenant to upgrade his account (which means additional fees) in order to fulfill his request. If the tenant does so, his/her request for adding (or enhancing) the requirements will be included in the candidate requirements for the next release; otherwise his request will be omitted.

In this paper, we assume that there is no ambiguity in matching the requirements and the SLA of each tenant. In order to make the relation between the requirements and SLA of the tenants, we define S as n × m matrix where the element

$$s_{ij} = \begin{cases} 1 & \textbf{if requirement j complies with SLA of tenant i} \\ 0 & \textbf{if requirement j does not comply with SLA of tenant i} \end{cases}$$

However, what would happen if some tenants change their level of subscription just before starting the construction of the next release? In this case, it is significant to reflect any change has emerged on the S matrix. We define the a function $\text{ChangOfSLA(i,j)} \in \{0, 1\}$ such that if the compliance of r_j with the SLA of t_i has changed, then $\text{ChangOfSLA(i,j)} = 1$; else $\text{ChangOfSLA(i,j)} = 0$.

The new values of the elements of S can be calculated as flows: $s_{ij} = s_{ij} \oplus \text{ChangOfSLA(i,j)}$, where $\oplus$ is the exclusive disjunction operator.

Degree of Commonalities of Requirements

Fulfilling the needs of more tenants with less effort is a significant objective. Therefore, it is efficient to select the requirements that are common by the highest possible number of tenants, but under the condition that these requirements comply with their SLA. We define the commonality of a requirement as follows:

$$\mathbf{Com(r_i)} = \sum_{j=1}^{m} ((\mathbf{r_i} \in \mathbf{Rt_j}) * \mathbf{s_{ij}})$$

If r_i is in the set Rt_j then the predicate $(r_i \in Rt_j) = 1$, otherwise it is 0. This equation shows that there are two factors that increase the degree of commonality of a requirement: the number of tenants whose SLA complies with the requirement and the number of tenants that need this requirement. Also the equation shows that the commonality of a requirement can range between 0 to m, i.e. $0 \leq \text{Com}(r_i) \leq m$. We define C as a vector that contains the commonalities of requirements, where $c_i = \text{Com}(r_i)$.

Required and available resources (resources constraints)

Many resources are required for implementing a requirement, such human resources, budget, and time. For the sake of simplicity, we assume that the required resources are provided in the form of person-days. Let us define the variable ReleaseCapacity as the available resources allocated to deliver the release [9, 16]. We represent the required effort to implement the requirements as a vector and let us call it F. The element f_i denotes the effort needed to implement requirement r_i. Given F and ReleaseCapacity, the flowing constraint shall be satisfied.

$$\sum_{i=1}^{w} \mathbf{f_i} \leq \mathbf{ReleaseCapacity}$$

Such that w is the total number of the selected requirements. This constraint states that the total required effort needed to implement the selected requirements shall be less or equal to the available effort.

Risk

The attributes of requirements, such as the complexity, completeness, and clarity, play significant role in defining the riskiness of the requirements [19]. In addition, the lack of some experience in the development team may increase the risk [20]; for example, working in a new development environment can reduce the velocity and the quality of the implementation process. In addition, risk can happen if there is high probability that the implementation of the selected requirements will affect the quality of the current operational software (for example, negative effects on the performance or tenants data). In this context, the risk of a requirement is estimated by the development team. We assume that development team is one stakeholder that provides one agreed-on risk estimation. We define a vector and we call it K where the element k_j denotes the estimated risk of the requirements r_j from the perspective of the development team.

Technical Relationships

The relationships between requirements are another main player in the release planning process. In this paper, two types of relationships are considered: I) coupling relationship where requirement r_i and r_j are coupled when it is beneficial to implement them in the same release, and II) precedence relationship where requirement r_i is a precedence of requirement r_j when it is beneficial to implement r_j after a full completion of implementing and testing r_i [9,16]. Formally, we define the relationships between requirements as two binary relations **Coup** and **Prec**, where:

$$\text{Coup} \subset R^* \times R^* \text{ and } \forall (r_i, r_j) \in \text{Coup} \rightarrow x_i = x_j$$

$$\text{Prec} \subset R^* \times R^* \text{ and } \forall (r_i, r_j) \in \text{Prec} \rightarrow x_i \leq x_j$$

Problem Statement for multi-tenant SaaS release Planning

Given above variables, from R^* , we want to optimize the selection of the sub set of requirements that will be included in the next release. The selected requirements shall achieve the following objectives: I) maximizes tenants' preferences about the importance, II) minimizes the risk, II) maximize degree of commonality, while considering the SLA for each tenant, and the effort and requirements' relationships constraints. Formally

$$\text{Max } F(x)$$

Such that $F(x) = = ((((E .*S)*W).*C)./k)^t * X$

Subject to,

1) $\sum_{i=1}^{w} \mathbf{f_i} \leq \mathbf{ReleaseCapacity}$
2) $x_i - x_j = 0 \ \forall (x_i, x_i) \in \text{Coup}$
3) $x_i - x_j \leq 0 \ \forall (x_i, x_i) \in \text{Prec}$
4) $x_i \in \{0,1\} \ \forall i = 1,2,\ldots\ldots,n$

.* and ./ are element wise multiplication and division operations.

The objective function shows that element wise multiplication is applied on E and S in order to eliminate the estimates provided by tenants who are not eligible to have the feature according to their SLA. For example, suppose tenant t4 gives 8 as the importance of the requirement r3 , which means e34 =8. However, when analyzing tenants' SLA, the development team finds that it does not comply with the needed QoS of this requirement, which means sij = 0. In this case, the estimate will be not counted. In addition, we can see from the objective function that the filtered estimates are multiplied with the decision weight vector. The reason behind this is to reflect the decision weight of the tenants on the final importance of the requirement. Furthermore, the objective function aims to maximize the commonality of a requirement by multiplying the final estimate of each requirement with its degree of commonality. Moreover, the objective function minimizes the risk by dividing the result obtained from the previous step by the estimated risk of the requirement. Thus, the risky requirements should have a lower chance of being included in the next release.

4. PROPOSED SOLUTION

Release planning can be formalized as an integer linear programming (ILP) problem, where the objective function and the constraints are is linear functions, and the decision variables are integer [7]. In many works, ILP has been applied to solve release planning problem [7, 11, 21] 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, BLP can be employed in our case. BLP is a special case of integer linear programming where the variable can take binary values.

BLP can be used in selection problem, where the decision makers have many alternatives and they want to eliminate the inappropriate ones. Also BLP can be used in yes/no problems, where the solution is a set of selected choices. This approach is suited to planning the next release, where the SaaS provider shall select a sub set of requirements among all those requested. The solution in BLP is represented using tree. Each path from the root node to a leaf node is a potential solution. With n decision variables, there are 2n possible solutions. When n is a big number, there will be huge solution space, which means huge solutions tree. Therefore, branch and bound technique is applied. The idea of branching is that the growth of the tree is only towards the most promising nodes. The upper bound value of the objective function is calculated at the root node. Before any further move, the upper bound is recalculated, and only the child that belongs to the feasible solutions and gives the highest possible upper bound is selected. This procedure is repeated until having the values of all variables. More details about BLP can be found in [22]. The next section illustrates how to manipulate the variables of the release planning and use BLP to generate release plan.

5. THE APPLICABILITY OF THE MODEL

Table 1 shows 20 requirements with 4 tenants. Each column j of this table represents the characteristic function of the set Rt_j, if r_i is required by r_j then the cell i,j =1; otherwise, it equals 0. Decision weights for the four tenants are captured by DW= [0.36 0.43 0.1 0.1]. Tables 2, 3, 4 show the compliance of SLA of each tenant with the requirements (matrix S), the evaluation provided by each tenant of each requirement (matrix E), and the commonality of the requirements (vector C) respectively. Additionally, the risk and the required resources in person/days format are provided in tables 5 and 6. We assume that the available resources (ReleaseCapacity) is 36 person/days. Also we assume that no changes are applied on the S matrix before starting the implementation of the release, which means $\sum_{i=1}^{m}\sum_{j=1}^{n}\mathbf{ChangOfSLA(i,j)=0}$. The relationships between requirements are given in relations Coup and Prec as follows:

Table1. The Requirements List Requested by Each Tenant

Requirements	Rt1	Rt2	Rt3	Rt4
r1	1	1	0	1
r2	0	1	0	1
r3	0	1	0	1
r4	1	1	1	0
r5	1	1	1	1
r6	0	1	0	1
r7	1	1	0	1
r8	0	1	0	1
r9	1	1	1	1
r10	0	1	1	1
r11	1	1	1	0
r12	1	1	1	0
r13	1	1	1	1
r14	1	1	1	1
r15	1	1	0	1
r16	0	1	1	1
r17	0	1	0	0
r18	0	1	1	1
r19	0	0	0	1
r20	1	1	0	0

Table2. The Compliance of SLA of Each Tenant with the Requirements

Requirements	SLA of t1	SLA of t2	SLA of t3	SLA of t4
r1	1	1	1	1
r2	0	1	1	1
r3	1	1	1	1
r4	1	1	0	0
r5	1	1	1	1
r6	1	1	0	0
r7	1	1	1	1
r8	1	1	1	1
r9	1	1	0	1
r10	1	1	1	1
r11	1	1	1	0
r12	1	1	0	1
r13	1	1	0	1
r14	1	0	1	0
r15	1	1	1	1
r16	1	1	1	1
r17	1	1	1	1
r18	1	0	1	1
r19	1	1	1	1
r20	1	1	1	1

Table3. The Estimates of the Importance of the Requirements

Requirements	t1	t2	t3	t4
r1	4	3	10	4
r2	1	6	7	6
r3	9	5	2	7
r4	6	4	1	1
r5	1	2	3	4
r6	3	4	1	1
r7	6	5	8	7
r8	8	7	7	5
r9	5	3	1	9
r10	6	2	5	7
r11	4	1	4	1
r12	7	1	1	6
r13	10	1	1	8
r14	7	1	3	1
r15	6	6	9	1
r16	10	6	7	6
r17	8	10	7	1
r18	6	1	7	2
r19	4	5	2	2
r20	2	2	2	4

Table4. The Commonalities of Requirements

Requirements	Commonality
r1	3
r2	2
r3	2
r4	2
r5	4
r6	1
r7	3
r8	2
r9	3
r10	3
r11	3
r12	2
r13	3
r14	2
r15	3
r16	3
r17	1
r18	2
r19	1
r20	2

Coup= {(r_3,r_5), (r_{13},r_{14})}

Prec= $\{(r_1,r_{12}), (r_2,r_6), (r_{17},r_{16})\}$

Coup relation is converted to the following equalities: $x_3 = x_5$ $x_{13} = x_{14}$

and Prec relation is converted to the following inequalities : $x_1 \leq x_{12}$ $x_2 \leq x_6$ $x_{17} \leq x_{16}$

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

x = bintprog (Rank,A,b,Aeq,beq)

Where Rank= (((E.*S)*DW).*C)./K

Table5. The Estimated Risk of the Requirements

Requirements	Risk
r1	1
r2	2
r3	2
r4	6
r5	4
r6	9
r7	8
r8	6
r9	1
r10	6
r11	8
r12	5
r13	1
r14	9
r15	2
r16	3
r17	6
r18	8
r19	9
r20	2

Table6. The Estimated Required Effort for the Requirements

Requirements	Required Effort (person/day)
r1	3
r2	5
r3	2
r4	1
r5	1
r6	6
r7	6
r8	1
r9	2
r10	3
r11	3
r12	5
r13	1
r14	3
r15	1
r16	7
r17	1
r18	6
r19	4
r20	1

Table7. Release Plan is Represented Using Binary Values (0 is not Assigned, 1 is Assigned)

Requirements	Assigned or not
r1	1
r2	1
r3	1
r4	1
r5	1
r6	0
r7	1
r8	1
r9	1
r10	1
r11	0
r12	0
r13	0
r14	0
r15	1
r16	1
r17	1
r18	0
r19	0
r20	1

Rank is a vector with n elements contains the coefficients of the objective function.

As described in Matlab documents: "A is a matrix contains the coefficients of the linear inequality constraints. b is a vector represents the right-hand side of the linear inequality constraints. Aeq is a matrix contains the coefficients of the linear equality constraints beq is the right-hand side of the linear equality constraints".

A and Aeq are n×n matrices, b and beq are n elements vectors. Each inequality constraint is represented by rows in A and b. Each row in A contains n elements (20 in our example), where the constrained elements are given values 1 and -1 (in order to represent $x_i - x_j \leq 0$), and the other elements are given 0 values. For example, the first constraint $x_1 \leq x_{12}$ is represented in A as the first row with element values [**1** 0 0 0 0 0 0 0 0 0 0 **-1** 0…………0], and represented in b as the first row with the element value 0. Also, A holds the effort constraint where the left-side of the effort constraints is represented in A as the last row with element values F^t, and the right-side is represented in b as the last row with the element value ReleaseCapacity. In the same way, each equality constraint is represented by rows in Aeq and beq. For example, the constraint $x_3 = x_5$ is represented in Aeq as the row with element values [0 0 **1** 0 **-1** ………..0].

After calling the **bintprog** function, vector X contains binary values that represent the release plan. If x_i is equal to 0 then r_i will be not assigned to the release plan, otherwise it will be assigned. Table 7 shows the x vector.

6. TENANTS' SATISFACTION

This section shows how much the tenants are satisfied if the proposed model is applied. In this context, the satisfaction of a tenant is measured by how many of the tenant's requested requirements will be implemented in the next release. For example, if a tenant has asked to add 10 requirements and the release plan includes 8 from these, then the degree of the satisfaction of that tenant is 80%.

In this experiment, the proposed model is used to generate 30 release plans using simulated data. The average release capacity is 41.5 person/days. The decision weights of the tenants are given in DW= [0.5 0.2 0.15 0.15]. Each iteration has 20 requirements to plan. For each release plan, the satisfaction of each tenant and the weighted satisfaction are calculated. The weighted satisfaction represents the overall satisfaction of all tenants depending on their decision weights.

Figure 2 shows the degree of satisfaction of every tenant for each release.

We can see from the figure that most of the release plans have achieved a higher degree of satisfaction for tenant 1. The reason behind this is that tenant 1 has a higher decision weight which provides his/her estimates with higher values.

Figure 3 shows the overall satisfaction of all tenants, which approximately ranges from 0.5 to 0.9. It is obvious that a higher value of the resource capacity (available resources) leads to a higher degree of satisfaction. For example, if there are infinite resources (no effort constraint) then we will have 100% satisfaction. Figure 4 shows the average degree of each tenant's satisfaction in all release plans. It is clear that the decision weights of the tenants have an influence on the SaaS provider's response to their requests. The tenant with a higher decision weight will be given more consideration when planning a release.

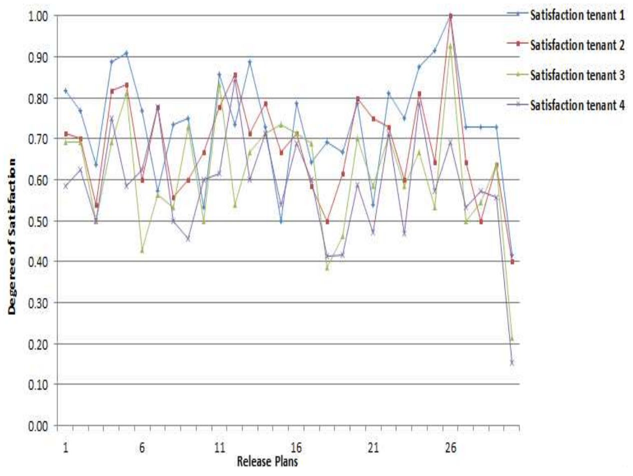

Figure 2. Degree of Tenants' Satisfactions in Each Release Plan

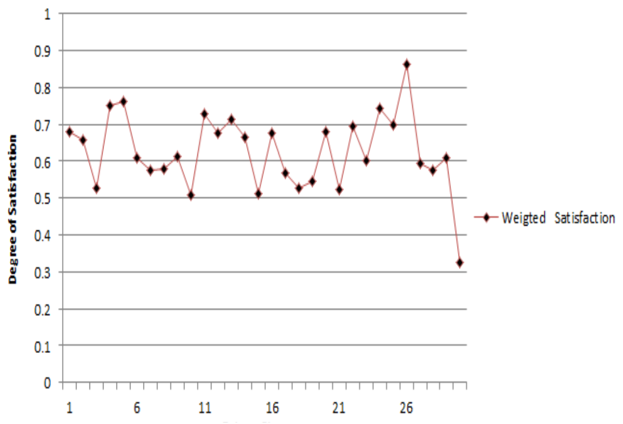

Figure 3. Degree of the Overall Satisfactions of All Tenants

In addition to the decision weights of the tenants, the degree of satisfaction is affected by the requirements founded in each list of each tenant. For example, suppose we relax the contractual constraint, and the same set of requirements are required by two tenants, then they will have the same degree of satisfaction regardless of their decision weight. Additionally, the SLA of each tenant plays significant role in the degree of satisfaction. In other words, if a tenant asks for adding requirements that he/she cannot have due his/her SLA, then the release plan will achieve low level of satisfaction for that particular tenant.

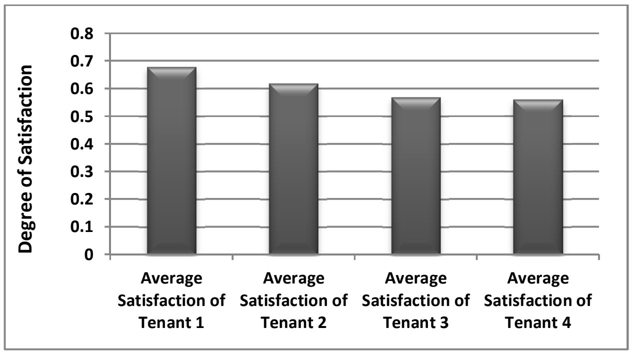

Figure 4. The Average Degree of Satisfaction of All Releases for Each Tenant

7. CONCLUSION

It is significant for the SaaS provider to increase the degree of tenants' satisfaction while considering resources and technical constraints. Therefore, SaaS providers deliver new releases very frequently in order to fulfill the evolving needs of their tenants. Many variables govern the release planning process for multi-tenant SaaS systems, which are the preferences of the tenants, SLA, degree of commonality, risk, available and required resources, and dependency relationships. Release planning is an optimization problem where we should select the requirements that maximize the commonality, importance. On the other hand, we want to minimize the risk and satisfy resources, dependency constraints, and contractual (SLA) constraints. Because of the extreme dynamic of the SaaS market, we choose to only plan for the next release. The release plan is represented using binary decision variables. Each decision variable x_i takes the 1 value if the requirement r_i is assigned to the release plan, and it takes the 0 value otherwise. Binary linear programming is used to optimize the selection process. Using experiments, we conclude that using proposed model can achieve acceptable degree of tenants' satisfactions. The degree of satisfaction depends on available resources, decision weights of the tenants, and the list of the required features for each tenant.

Further research is required to apply other optimization techniques to construct release plans for SaaS. Moreover, more research on release planning for SaaS with the use of a prioritization-based approach is suggested.

7. ACKNOWLEDGMENTS

The authors would like to thank King Saud University (KSU)[1] for their support of this research, and for their funding of the scholarship of the first author.

[1] **King Saud University, P.O. Box 2454, Riyadh 11451, Saudi Arabia**

8. REFERENCES

[1] Mell, P and Grance, T. the NIST Definition of Cloud Computing. National Institute of Standards and Technology (NEST), 2009, from http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[2] Erl ,T., Mahmood,Z., and Puttini, R., Cloud Computing: Concepts, Technology & Architecture. *Prentice Hall,2013.*

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

[4] Salesforce, "The 7 Secrets of SaaS Start-up Success", URL: http://www.salesforce.com/assets/pdf.

[5] Fry, C. and Greene, S. Large Scale Agile Transformation in an On-Demand World. In Anonymous Agile Conference (AGILE), 2007. (). , 2007, 136-142.

[6] Ruhe, G and An Ngo-The. Hybrid Intelligence in Software Release Planning. Journal of Hybrid Intelligent Systems, 1(2004), 99-110.

[7] Ruhe, G. and Saliu, M. O. The art and science of software release planning. Software, IEEE, 22, 6 (2005), 47-53. DOI=10.1109/MS.2005.164.

[8] An Ngo-The and Ruhe, G. Optimized Resource Allocation for Software Release Planning. Software Engineering, IEEE Transactions on, 35, 1 (2009), 109-123. DOI=10.1109/TSE.2008.80.

[9] Greer, D. and Ruhe, G. Software release planning: an evolutionary and iterative approach. Information and Software Technology, 46, 4 (2004), 243-253. DOI=10.1016/j.infsof.2003.07.002

[10] Shen, W. Software release planning with fuzzy objectives and constraints. M.Sc. Thesis, University of Calgary (Canada), Canada, 2005.

[11] van den Akker, M., Brinkkemper, S., Diepen, G. and Versendaal, J. Software product release planning through optimization and what-if analysis. Information and Software Technology, 50, 1-2 (2008), 101-111.

[12] Ullah, M. I. and Ruhe, G. Towards comprehensive release planning for software product lines. In Anonymous Proceedings of the First International Workshop on Software Product Management, IWSPM'06. (). , 2006, 51-55.

[13] Ngo-, T. and Ruhe, G.
A systematic approach for solving the wicked problem of softwar erelease planning. Soft Computing, 12, 1 (2008), 95-108.

[14] Colares, F., Souza, J., Carmo, R., Padua, C. and Mateus, G. R. A New Approach to the Software Release Planning. In Anonymous Software Engineering, 2009. SBES '09. XXIII Brazilian Symposium on. (). , 2009, 207-215.

[15] An Ngo-The and Saliu, M. O. Fuzzy Structural Dependency Constraints in Software Release Planning. In Anonymous Fuzzy Systems, 2005. FUZZ '05. The 14th IEEE International Conference on. (). , 2005, 442-447.

[16] An Ngo-The, Ruhe, G. and Wei Shen. Release planning under fuzzy effort constraints. In Anonymous Cognitive Informatics, 2004. Proceedings of the Third IEEE International Conference on. (). , 2004, 168-175.

[17] Karlsson, J. and Ryan, K. A cost-value approach for prioritizing requirements. Software, IEEE, 14, 5 (1997), 67-74. DOI=10.1109/52.605933.

[18] Linlin Wu, Garg, S. K. and Buyya, R. SLA-Based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments. In Anonymous Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on. (). , 2011, 195-204.

[19] Sharma, A and Kushwaha, D.S. A Complexity Measure Based on Requirement Engineering Document. Journal of Computer Science and Engineering, 1,1(2010), 112 -117.

[20] Vinod, V, Dhanalakshmi, J and Sahadev,S. Software Team Skills on Software Product Quality.Asian Journal of Information Technology, 8,1(2009), 8-13.

[21] Li,C, van den Akker, J. M. , Brinkkemper,S and Diepen ,G. Integrated Requirement Selection and Scheduling for the Release Planning of a Software Product. In Requirements Engineering: Foundation for Software Quality, 13th International Working Conference, REFSQ 2007, Trondheim, Norway, June 11-12, 2007.

[22] Chinneck ,J. W. PRACTICAL OPTIMIZATION: A GENTLE INTRODUCTION. Systems and Computer Engineering, Carleton university, 2012, available online at www.sce.carleton.ca/faculty/chinneck/po.html.