Journal of the Society of Korea Industrial and Systems Engineering
Vol. 35, No. 2, pp.57－63, June 2012.

소프트웨어 제품라인의 출시 계획을 위한 최적해법

유 재 욱†

동아대학교 경영대학 경영학과 조교수

An Exact Solution Approach for Release Planning of Software Product Lines

Jaewook Yoo†

Department of Business Administration, Dong-A University

소프트웨어 개발에 있어서 소프트웨어를 시장에 출시하는 계획을 수립하는 것은 소프트웨어를 이루고 있는 기능들을 구현하는 데 제약이 되는 조건들(기술, 자원, 위험, 예산 등)을 만족하면서 계획된 출시기간에 이들 기능들을 할당하는 일이다. 이와 같이 소프트웨어 출시를 계획하는 것은 소프트웨어 제품라인에 대해서 고려할 때 더욱 복잡해진다. 본 연구에서는 소프트웨어 제품라인에 있어서 소프트웨어 출시 계획을 수립하기 위한 문제를 우선순위 제약하의 다수 0-1 배낭문제로 수리 모형화하고, 이를 풀기 위한 최적해법이 개발된다. 최적해법은 동적 계획법이 주가 되고, 문제의 크기를 줄이기 위하여 휴리스틱과 축소방법이 이용된다.

Keywords : Release Planning, Software Product Lines, Dynamic Programming, Precedence-Constrained Multiple 0-1 Knapsack Problem

1. Introduction

Software product lines are built of a reusable platform common to the whole software product family and specific variants sharing the platform. The concept of software product lines is considered as a viable and important software development paradigm allowing companies under certain conditions to realize order-of-magnitude improvements in time to market, cost, productivity, quality, and other business drivers [1].

Release planning for incremental software development is to assign features to release such that most important technical, resource, risk and budget constraints are met. Without good release planning critical features are jammed into the release late in the cycle without removing features or adjusting dates. This might result in unsatisfied customers, time and budget overruns, and a significant loss in market share [3]. Release planning for software product lines involves a number of new aspects, which are not applicable in traditional one. It has to consider multiple products being developed by multiple teams for different customers.

Currently, there are few models for release planning for software product lines that allows addressing the problem in a rigorous formalized manner finding optimal solutions. Although Ullah and Ruhe [6] developed a comprehensive and formalized model for software release planning in software product lines, they mapped release planning problem for software product lines to use existing functionality of

논문접수일 : 2012년 03월 06일　　게재확정일 : 2012년 03월 30일
† 교신저자 jyoo@dau.ac.kr
※ This work was supported by the Dong-A University research fund.

the ReleasePlanner™ decision support system developed for release planning of one product. Their solutions are not guaranteed to be optimal.

Release Matrix approach proposed by Taborda [5] provided a holistic view of software product lines' release management. However, it lacked the model and its solution approach needed to obtain the optimal solutions.

The main contribution of this article is the formulation of a mathematical model and development of a solution methodology for selecting and assigning features or requirements in sequence of releases for software product lines, along a specified planning horizon. The problem is formulated as a precedence-constrained multiple 0-1 knapsack problem.

The proposed solution methodology in this research can be viewed as a dynamic programming approach. The imbedded-state approach is used to reduce a multi-dimensional dynamic programming to a one-dimensional dynamic programming over a specified release planning horizon. The feasible solutions are generated by applying the backward dynamic programming approach to the single-release problem corresponding to each stage, as well as a pre-processing method is applied to reduce the problem size at each stage which can be solved easily by the dynamic programming approach.

In addition to this introductory section (Section 1), this article contains four more sections. Section 2 presents the mathematical formulation of the problem and the overall solution approach. Section 3 provides the description of each procedural component of the proposed solution methodology. Section 4 shows a numerical example and includes a short discussion of relevant aspects. Finally, section 5 consists of a summary along with conclusions and recommendations.

2. Model and Solution Approach

2.1 Model

In a software product line there are two types of features, i.e. core asset features of which a platform is composed, and product specific features. In this model it is assumed that these two types of features are already decided during the scoping process and core asset features are used by all the products while each product specific feature is required by only one of the products in the software product line. Therefore, the set of features which platform or product k consists of is given $F(k)$; $\cup_k F(k) = \{1, 2, \cdots, I\}$. These features can be assigned to one of the T possible release options. This is described by decision variable $x_{it} = 1$ if feature i is assigned to release $t \in \{1, 2, \cdots, T\}$ and $x_{it} = 0$ if feature i is postponed. The model considers the precedence relation among features which imposes to release features in a certain order. Let C_t be an available resource capacity at release t and w_{it} an amount of development resources which feature i requires at release t.

Stakeholders are extremely important in release planning. Assume a set of stakeholder $S = \{S(1), \cdots, S(q)\}$. Each stakeholder q can be assigned relative importance $\lambda(q) \in \{1, \cdots, 9\}$. $\lambda(q) = 1$ indicates the lowest and $\lambda(q) = 9$ the highest degree of stakeholder importance. Each stakeholder prioritizes every feature based on two criteria using a nine point scale. The first criterion is represented by value(q, i) and the second criterion is represented by urgency(q, i).

The objective is to maximize a function $P(x)$ among all release plans x subject to satisfaction of resource, precedence, and feature-selection constraints described in the model. The function $P(x)$ depends on a number of factors such as the value of the release, the importance of stakeholders, and the urgency of a feature and its value to stakeholders. It is defined as follows.

$$P(x) = \sum_t \sum_{i \in F(k)} p_{it} x_{it}$$

$$\text{with} \quad p_{it} = \xi(t)[\sum_q \lambda(q) \times value(q, i) \times urgency(q, i)] \quad (1)$$

In (1), $\zeta(t)$ expresses the relative (normalized to 1) importance of release t. The mathematical model is referred to as Problem (P) and is formulated in relationships (2)-(6).

Problem(P)

$$\max \sum_{t=1}^{T} \sum_{i \in F(k)} p_{it} x_{it} \quad (2)$$

$$s.t. \sum_{i \in F(k)} w_{it} x_{it} \le C_t \qquad \textit{for all } i, k, t \quad (3)$$

$$\sum_{t=1}^{T} (T+1-t)(x_{it} - x_{jt}) \ge 0 \qquad \textit{for all } i, j, t \quad (4)$$

$$\sum_{t=1}^{T} x_{it} \le 1 \qquad \textit{for all } i \quad (5)$$

$$x_{it} \in \{0, 1\} \qquad \textit{for all } i, t \quad (6)$$

In the formulation of Problem (P) the objective function (2) being maximized is defined as (1); the resource constraint set (3) indicates that the resource consumption by the selected features which platform or product k consists of cannot

exceed the available resource at each release *t*; the precedence constraint set (4) ensures possible dependencies between features in software product line over release planning horizon; the feature-selection constraint set (5) forces the model to assign a feature to any release planning horizon or delay its release; and constraint set (6) imposes the integrality of the decision variables.

2.2 Solution Approach

Dynamic programming approach is developed to solve the precedence-constrained multiple 0-1 knapsack problem to optimality, as well as a pre-processing method is applied to reduce the size of the problem at each stage.

In multi-release dynamic programming model, the problem is divided into smaller subproblems corresponding to each single-release problem. The feature of the proposed algorithm in this paper is its capability of reducing the state-space which otherwise would present an obstacle in solving multi-dimensional dynamic programming problems. This is due to the use of the imbedded-state approach, which reduces a multi-dimensional dynamic programming to a one-dimensional dynamic programming. Morin and Marsten [2] applied the imbedded-state approach to develop an algorithm for the solution of nonlinear knapsack problems.

Moreover, a dynamic programming approach for a single-release problem is used to identify feasible solutions to the next release problem corresponding to the next stage in the multi-release dynamic programming formulation. By dominance test, feasible solutions dominated by any other feasible solutions are eliminated and efficient solutions are selected from the remaining set of release strategies. Then the efficient solutions are used to generate potential solutions for the next stage.

A pre-processing approach is performed on the single-release problems corresponding to each stage in multi-release dynamic programming model in order to make the problem size reduced. It consists of a heuristic algorithm that finds quickly a lower bound to be used to reduce the problem size and a reduction method to make the size of the problem smaller. The latter makes use of the lower bound and fixes considerably many variables at either 0 or 1. Thus, eliminating the fixed variables, we are left with problems of much smaller size that can be solved easily by the dynamic programming algorithms at each stage.

The proposed approach can be divided into two procedures : pre-processing method and dynamic programming. The step 1 is pre-processing procedure, and the dynamic programming procedure consists of steps 2 and 3. A brief description of each major component of the methodology is provided as follows.

Step 1 : Pre-processing Procedure (heuristic and reduction algorithm)
A heuristic algorithm finds a lower bound to the current single-release problem corresponding to a stage in multi-release dynamic programming as well as a reduction method makes use of the lower bound and fixes numerous variables at either 0 or 1. Eliminating the fixed variables at each stage results in much smaller size of the problem that can be solved easily by the dynamic programming approaches. The pre-processing approach is applied to each stage in multi-release dynamic programming model.

Step 2 : Dynamic Programming for Single Release
A dynamic programming methodology is conducted to identify feasible solutions to single-release problems reduced by the pre-processing procedure corresponding to each stage in multi-release dynamic programming approach.

Step 3 : Dynamic Programming for Multiple Releases
Feasible solutions that are dominated by any other feasible solutions are eliminated. The remaining set of feasible solutions will be referred to as efficient solutions. The efficient solutions at the stage representing the current release planning are obtained and used to generate potential solutions for the next stage.

3. Description of the Solution Method

3.1 Dynamic Programming for Multiple Releases

The dynamic programming model for multiple releases is developed in a compact form of the separable nonlinear multi-dimensional knapsack problem (NKP) as shown in Problem (*D*), which is equivalent to Problem (*P*).

Problem (*D*)

$$P_t(c) = \max \sum_{t=1}^{T} P_t \widehat{X}_t$$

$$s.t. \sum_{i=1}^{T} W_{rt} \widehat{X}_t \le C_r \qquad 1 \le r \le T+I$$

$$\widehat{X}_t \in \Omega_t \qquad t = 1, 2, \cdots, T$$

where $\hat{X}_t$ is a release strategy for a software product line at release *t*; $\hat{X}_t = (x_{1t}, x_{2t}, \cdots, x_{It})$, P_t is a profit of strategy $\hat{X}_t$, W_{rt} is the amount of resource (resource and feature-selection) *r* taken by strategy $\hat{X}_t$ for a software product line at release *t*, C_r is an available resource *r*, and $\Omega_t = \{\hat{X}_t$: precedence constraint and 0-1 integrality constraint}.

Allocation of resources to a software product line using the dynamic programming approach results in the following recursive relationship:

$$P_n^*(S_n) = \max_{\hat{X}_n \in \phi_n} \{P_{n-1}^*(S_{n-1}) + P_n(S_{n,} \hat{X}_n)\}$$

where; ϕ_n={$\hat{X}_t$; resource constraint, precedence constraint, feature-selection constraint, and 0-1 integrality constraint} and the state variable S_n represents the amount of resource *c* which is available for allocation at release *n* and is a *T+I* dimensional vector. The vector is divided into two groups. The first group is represented by *T* dimensional vector corresponding to the resource in each release. The second group is represented by *I* dimensional vector corresponding to selection availability of features for the software product line.

Problem (*D*) can be decomposed into subproblems that can be considered as a single stage in the multi-release dynamic programming problem. In each single release, the feasible solutions are obtained by applying a single-release dynamic programming approach and the efficient solutions that are not dominated by any other feasible solutions are obtained by dominance testing.

Let $\tilde{X}_n^f$ denote the set of feasible solutions of Problem (*D*) until stage *n*. The feasible solution $\tilde{X}_n = (\hat{X}_{1,} \hat{X}_{2,} \cdots, \hat{X}_n) \in \tilde{X}_n^f$ is said to be dominated by the feasible solution $\tilde{X}_n' \in \tilde{X}_n^f$, if we have both

$$\Sigma_{t=1}^n W_{rt} \hat{X}_t' \le \Sigma_{t=1}^n W_{rt} \hat{X}_{t,} \qquad \forall r,$$

and

$$\Sigma_{t=1}^n P_t \hat{X}_t' \ge \Sigma_{t=1}^n P_t \hat{X}_{t,}$$

with at least one strict inequality. If $\tilde{X}_n \in \tilde{X}_n^f$ is not dominated by any other element of $\tilde{X}_n^f$, then we say that $\tilde{X}_n$ is efficient with respect to $\tilde{X}_n^f$. Then the efficient solutions are used to generate potential solutions for the next stage [2].

3.2 Dynamic Programming for Single-Release

In order to obtain feasible and efficient solutions in each release, a dynamic programming approach is applied to single-release problems corresponding to each stage in a multi-release dynamic programming. Precedence relations between features in the model can be represented by means of a direct acyclic graph, where nodes correspond to features in a one-to-one way.

In an acyclic graph $G_0 = (V_0, E_0)$, $v' \in V_0$ is a descendant of $v \in V_0$ if there exist an integer $k \ge 0$ and a sequence of vertices $v = v^0, v^1, \cdots, v^k = v'$ such that

$$(v^i, v^{i+1}) \in E_0 \qquad \text{for I = 0, 1, } \cdots, \textit{k}\text{-1.}$$

This is denoted as $v \rightsquigarrow v'$, and v is an ancestor of v'. Note that v is a descendant (ancestor) of itself. Let the sets of all descendants and all ancestors of v be denoted as

$$D_v = \{v' \in V_0 | v \rightsquigarrow v'\}$$
$$A_v = \{v' \in V_0 | v' \rightsquigarrow v\}.$$

For a subset of vertices $V \subseteq V_0$, $G_0|v$ denotes the subgraph of G_0 restricted to *V* with the corresponding edge set

$$E = \{e \in E_0 | e \in V \times V\}.$$

If there is no confusion, $G_0|v$ is simply denoted as *G* = (*V*, *E*). By top(*G*), we mean the vertex with the smallest index in *V*, namely,

$$\text{top}(G) = v_{min\{i | v_i \in V\}}$$

Subgraph *G* is said to be a singleton if |*V*| = 1, and is said to be empty if $V = \phi$.

Corresponding to an arbitrary subgraph *G* = *(V, E)*, we introduce a restriction of Problem (*P*) to *G* in the form of a single-release problem at stage *t* as shown in Problem (*G*, *c*) :

Problem (*G*, *c*) :

$$p(\hat{X}_t) = \max \Sigma_{i \in F(k)} p_{it} x_{it}$$
$$s.t.\ \Sigma_{i \in F(k)} w_{it} x_{it} \le C_t \qquad \textit{for all } k$$
$$x_{it} \ge x_{jt} \qquad \textit{for all(i, j)} \in E$$
$$x \in \{0, 1\} \qquad \textit{for all } v_i \in V$$

This is also referred to as subproblem G for simplicity. Note also that, if we fix

$$x_{top(G)} = 1$$

in Problem (G, c), the objective value is $p_{top(G)}$ plus the optimal value from the remaining nodes $V \setminus \{top(G)\}$ with the reduced resource capacity $c_t - c_{top(G)}$. On the other hand, if we set

$$x_{top(G)} = 0,$$

all the descendants of top(G) are excluded, and we will have the subgraph with node set $V \setminus D_{top(G)}$.

Define the left and right children of G by

$$G_L = G|_{V\setminus\{top(G)\}},$$
$$G_R = G|_{V\setminus Dtop(G)\}}.$$

Then, the optimal values from these children are

$$p^*(G, c) = \max\{p_{top(G)} + p^*(G_L, c - c_{top(G)}), p^*(G_R, c)\},$$

and the optimal decision for $x_{top(G)}$ is given by

$$x^*(G, c) = \begin{cases} 1, \text{ if } p_{top(G)} + p^*(G_L, c - c_{top(G)}) \geq p^*(G_R, c), \\ 0, \ otherwise \end{cases}$$

For the case of singleton, i.e., $V = \{v_i\}$, we have immediately

$$p^*(G, c) = \begin{cases} p_{it}, & \text{if } c_t \geq w_{it} \\ 0, & otherwise \end{cases}$$

with

$$x^*(G, c) = \begin{cases} 1, & \text{if } c_t \geq w_{it} \\ 0, & otherwise. \end{cases}$$

For the empty graph, we have

$$p^*(\phi, c) = 0.$$

3.3 Greedy Algorithm

The following is a greedy algorithm that tries to select features, one by one, as far as the resource is available at the current release t. After applying the greedy algorithm at release t, the input graph in consideration at the next release t+1 is one reduced by deleting the features released at release t obtained by a reduction method and single-release dynamic programming approach. The procedure of the greedy algorithm is shown as follows [4].

Let p_t and w_t be the current profit and weight of the graph G at release t.

Step 1 : Set $x_t = \{\}$, $p_t = 0$, $w_t = 0$.
Step 2 : Look for a feature j such that :

(i) $x_{jt} = 0$,
(ii) $x_{it} = 1, \quad \forall (v_i, v_j) \in E_0$,
(iii) $w_t + w_{jt} \leq c_t$

If such a feature j is found, go to Step 3.
Otherwise, stop.

Step 3 : Set $x_{jt} = 1$, $p_t = p_t + p_{jt}$, $w_t = w_t + w_{jt}$, and go to Step 2.

The output (p_t, w_t) from this algorithm is referred to as the value for applying reduction algorithm to reduce the problem size.

3.4 Reduction Algorithm

Let $x^* = x^*_{jt}$ be an optimal solution to the Problem (G, c). For each feature i, we define

$$\overline{w}_{it} = \Sigma_{v_j \in Av_i} w_{jt},$$
$$\tilde{p}_{it} = \Sigma_{v_j \notin Dv_i} p_{jt},$$

Then, the following proposition is straightforward.

Proposition. (i) $\overline{w}_{it} > c_t$ implies $x^*_{it} = 0$;
(ii) $\tilde{p}_{it} < p_{G_i}$ implies $x^*_{it} = 1$.

Proof.
(i) If $x^*_{it} = 1$, we have $x^*_{jt} = 1$ for all $v_j \in A_{v_i}$. Then,

$$\sum_{v_j \in V} w_{jt} x^*_{jt} \geq \overline{w}_{it} > c_t,$$

which is a contradiction.

(ii) If $x^*_{it} = 0$, we have $x^*_{jt} = 0$ for all $v_j \in D_{v_i}$. Then,

$$\sum_{v_j \in V} p_{jt} x^*_{jt} \leq \tilde{p}_{it} < p_t.$$

Therefore, x^* is not optimal, which is also a contradiction.

This proposition serves as a pegging test [4] to fix some of the variables at either 0 or 1, and thus reduce the size of the single-release problems.

4. Numerical Example

A numerical example has been developed to show how well the solution approaches proposed in this article work on it. Consider a software product line with two products (product A and product B), as well as a platform. There are eight candidate features in the product line. The assignment of features to products and platform and their interdependencies are shown in <Figure 1>.

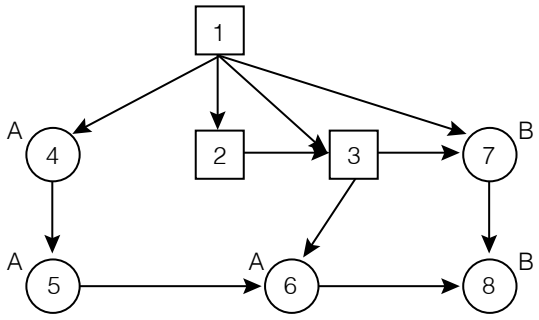
In this example we are considering T = 2 releases. There is one development team resource and its capacity is C_1 = 35 person days and C_2 = 31 person days for release t = 1 and 2, respectively. The relative importance of release are $\zeta(1) = 0.7$ and $\zeta(2) = 0.3$, respectively. There are two stakeholders $S(1)$ and $S(2)$ with relative weights $\lambda(1) = 8$ and $\lambda(2) = 5$. The resource requirements for the eight features are given in table 1. Stakeholders' votes on all of the features for value and urgency criteria are also shown in <Table 1>.

The proposed solution approaches in this article is applied to the above example as follows.

[Greedy Algorithm]
At release t = 1, feature 1, 2, 3, and 7 are selected with the profit of 1342 by using the greedy algorithm.

[Reduce the problem]
<Table 2> gives $\overline{w}_{i1}$ and $\tilde{p}_{i1}$. From this and $p_1 = 1342$, we can fix $x^*_{51} = x^*_{61} = x^*_{81} = 0$ (subscript ♭ in <Table 2>) and $x^*_{11} = x^*_{21} = x^*_{31} = 1$ (subscript # in <Table 2>) at release t = 1 according to the proposition mentioned in section 3.4. By eliminating the fixed features, the problem is reduced to the problem regarding feature 4 and 7 only.

[Dynamic programming]
To get feasible solutions at release t = 1, single-release dynamic programming approach is applied to the reduced problem consisting of feature 4 and 7. As the result, the feasible solutions at release t = *1* are $\hat{X}^1_1 = (11100000)$ and $\hat{X}^2_1 = (11100010)$. These two feasible solutions are efficient since they are not dominated by each other. Based on $\hat{X}^1_1$, the remaining problem considered at release t = 2 is the problem having five features (4, 5, 6, 7, and 8) and their interdependencies with C_2 = 31, while the remaining problem based on $\hat{X}^2_1$ considered at release t = 2 is the problem having four features (4, 5, 6, and 7) and their interdependencies with C_2 = 31.



①→② Feature 1 precedes feature 2.
ⓕ Product specific feature
[f] Platform feature

<Figure 1> Graphical View of Feature

<Table 1> Effort Estimates and Stakeholders' Votes

Feature i	w_i	Stakeholders S(1)		Stakeholders S(2)	
		Value (1, i)	Urgency (1, i)	Value (2, i)	Urgency (2, i)
1	5	6	7	4	2
2	14	5	8	7	6
3	7	6	7	5	6
4	16	4	2	6	7
5	15	7	6	6	4
6	10	6	7	6	3
7	5	5	7	7	7
8	8	6	6	5	8

The pre-processing approach is applied to these two remaining problems again in order to reduce the problems' size, as well as dynamic programming procedure is applied to find the optimal solution. Then, the procedure is terminated with an optimal solution $\tilde{X}^*_2 = (\hat{X}^*_1, \hat{X}^*_2)$, where $\hat{X}^*_1 = (1, 1, 1, 0, 0, 0, 1, 0)$ and $\hat{X}^*_2 = (0, 0, 0, 1, 1, 0, 0, 0)$.

<Table 2> Reduction of the problem

Features	Ancestors	Descendants	$\overline{w}_i$	$\tilde{p}_i$
1	1	1, 2, 3, 4, 5, 6, 7, 8	5	$0^{\#}$
2	1, 2	2, 3, 6, 7, 8	19	$774^{\#}$
3	1, 2, 3	3, 6, 7, 8	26	$1145^{\#}$
4	1, 4	4, 5, 6, 8	21	1342
5	1, 4, 5	5, 6, 8	$36^{\flat}$	1534
6	1, 2, 3, 4, 5, 6	6, 8	$67^{\flat}$	1853
7	1, 2, 3, 7	7, 8	31	1784
8	1, 2, 3, 4, 5, 6, 7, 8	8	$80^{\flat}$	2151

5. Summary

Software release planning model of software product lines was formulated as a precedence-constrained multiple 0-1 knapsack problem. The purpose of the model was to maximize the total profit of an entire set of selected features in a software product line over a multi-release planning horizon. The solution approach is a dynamic programming procedure. Feasible solutions at each stage in dynamic programming are determined by using backward dynamic programming approach while dynamic programming for multi-release planning is forward approach.

The pre-processing procedure with a heuristic and reduction algorithm was applied to the single-release problems corresponding to each stage in multi-release dynamic programming in order to reduce the problem size. The heuristic algorithm is used to find a lower bound to the problem. The reduction method makes use of the lower bound to fix a number of variables at either 0 or 1. Then the reduced problem can be solved easily by the dynamic programming approaches. These procedures keep on going until release $t = T$. A numerical example was developed to show how well the solution procedures in this research works on it.

Future work in this area could include the development of a heuristic to obtain lower bounds closer to the optimal solution to the model in this article, as well as computational test of the heuristic algorithm and the exact solution approach developed in this paper. Also, more constraints reflecting the characteristics of software product lines may be added to the model. For instance, other resources such as multiple teams, each developing one product or a platform in a software product line could be added to the model.

References

[1] http://www.sei.cmu.edu/productlines.

[2] Morin, T. L. and Marsten, R. E.; “An algorithm for nonlinear knapsack problems,” *Management science*, 22(10) : 1147-1158, 1976.

[3] Penny, D.; “An Estimation-Based Management Framework for Enhancive Maintenance in Commercial Software Products,” *Proceedings of the International Conference on Software Maintenance*, Montreal, Canada, 122-130, 2002.

[4] Samphaiboon, N. and Yamada, T.; “Heuristic and exact algorithm for the precedence-constrained knapsack problem,” *Journal of optimization theory and applications*, 105(3) : 659-676, 2000.

[5] Taborda, L.; “Generalized Release Planning for Product Line Architectures,” *Proceedings of the SPLC, The Third Software Product Lines Conference*, Boston, USA, 238-254, 2004.

[6] Ullah, M. and Ruhe, G.; “Towards Comprehensive Release Planning for Software Product Lines,” *Proceedings of the First International Workshop on Software Product Management*, Minneapolis/St. Paul, Minnesota, USA, 55-59, 2006.