Integrated Requirement Selection and Scheduling for the Release Planning of a Software Product

C. Li

J.M. van den Akker

S. Brinkkemper

G. Diepen

Department of Information and Computing Sciences
Utrecht University

Technical Report UU-CS-2006-059
www.cs.uu.nl

ISSN 0924-3275

Integrated Requirement Selection and Scheduling for the Release Planning of a Software Product

C. Li, J.M. van den Akker, S. Brinkkemper, and G. Diepen

Department of Information and Computing Sciences,
Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands
{cli, j.m.vandenakker, s.brinkkemper, diepen}@cs.uu.nl

Abstract. This paper integrates requirement scheduling issues into software release planning. Two integer linear programming models are presented—the first model can schedule the development of the requirements for the new release exactly in time so that the project span is minimized and the resource and precedence constraints are satisfied. The second model is for combined requirement selection and scheduling and it can not only maximize revenue but also calculates an on-time-delivery project schedule simultaneously. We also run two simulations to examine the influence of precedence constraints and compare the differences of the traditional prioritization models and the two new ones. The simulation results suggest that requirement dependency can significantly influence the project plan and the combined model for requirement selection and scheduling is better in the sense of efficiency and on-time delivery.

1 Introduction

Determining requirements for the upcoming release is a complex process [24]. With the evident pressure on time-to-market [22, 27] and limited available resources, often there are more requirements than can be actually implemented. The market-driven requirement engineering processes [6] have a strong focus on requirement prioritization [18]. The requirement list needs to fulfill the interests of various stakeholders and takes many variables into consideration. Several scholars have presented lists of such variables including: importance or business value, preference of different stakeholders, cost of development, requirement quality, development risk and requirement dependencies [8, 13, 14, and 27].

In order to deal with the multi-aspect optimization problem, different techniques have been applied. The analytical hierarchy process (AHP) [18, 22] assesses requirements by examining all possible requirement pairs and use matrix calculations to determine a weighted list. Jung [17] extended the work of Karlsson and Ryan [18] by using integer linear programming (ILP) to reduce the complexity of AHP to large amounts of requirements. Carlshamre [8] too used ILP on which a release planning tool was built and added requirement dependencies as an important aspect in release planning. Ruhe and Saliu [25] describe a method based on ILP to include stakeholder's opinions for release planning. Van den Akker et al [2] further extended the ILP technique by including some management steering mechanisms and ran a few simulations to test the influences of each factor. Besides ILP techniques, the cumulative voting method [19] allows different stakeholders to assign a fixed amount of units among all requirements, and an average weighted requirement list is constructed; Ruhe & Saliu [25] provide a method called EVOLVE to allocate requirements to incremental releases. For more techniques, Berander and Andrews [4], provide an extensive list of requirement prioritizing techniques.

The schedule of the requirements development is also suggested as an important issue in this field [13], unfortunately, few prioritization methods have taken it into account. Scheduling requirements is considered as a next step after requirement selection [8] and the two processes—selection and scheduling are often used iteratively to find a group of requirements with an on-time delivery project plan [24]. Compared to the extensive research on requirement selection, only little research has been performed for the scheduling part. Given the fact that 80% of software projects are late or over budgeted [10], a precise project plan which can synchronize every development team is necessary. The traditional critical path algorithm or Gantt chart method is widely used for project planning, but is often not able to include all factors. Different types of dependencies [7], which describe the relationships between requirements, also increase the complexity of making a project plan.

1.1 Example of release planning problem

Release Definition 5.1

Nr.	Requirement	Dependency	Revenues	Total man days	Team A	Team B	Team C
12	Authorization on order cancellation and removal	Imp 63, 25	24	50	5		45
34	Authorization on archiving service orders		12	12	2	5	5
63	Performance improvements order processing		20	15	15		
25	Inclusion graphical plan board	Com 66	100	70	10	10	50
43	Link with Acrobat reader for PDF files	Imp 25	10	33		33	
75	Optimizing interface with international Postal code system	Imp 25	10	15			15
35	Adaptations in rental and systems		35	40		20	20
66	Symbol import		5	10	10		
67	Comparison of services per department		10	34		9	25
	Total		226	279	42	77	160
	Available resources (number of developers)			3	1	1	1
	Release duration				60 days		
	Available team capacity for release			180	60	60	60

Table 1: Example requirements sheets of a release planning problem

Table 1 depicts a simplified example representation of the release planning problem. For nine requirements with estimated revenue (in euro) and cost (in man days), the available resources in different teams (or skills) within the given period, and the interdependencies between the requirements, the best set of requirements for a next release needs to be determined. Here we use the six types of dependencies suggested by Carlshamre [7]. These are given by: 1) *Combination*: two requirements are in need of each other; 2) *Implication*: one requirement requires another one to function; 3) *Exclusion*: two requirements conflicts to each other. 4) *revenue-based* and 5) *cost-based* dependency means one requirement influences the revenue / cost of another. And 6) *time-related* dependency means one requirement needs to be implemented after another.

The release planning problem has been modeled as a multi-dimensional knapsack problem [2, 8, 17, and 25]. Using ILP technique, five requirements are selected (marked in yellow) so that the total revenue is maximized against the available resources. It is also possible to include requirement dependency and some management steering mechanisms like hiring external personnel, deadline extension, etc in the model, we refer to van den Akker et al [2] for detail. To solve the ILP problem, we refer to Wolsey [28] for details.

The next step is to schedule the selected requirement exactly in time. Here we have to deal with dependencies that result in restrictions on time. For example, requirements pertaining to foundational components often need to be implemented before others. Similarly, certain capabilities (for example quality issues like safety and security) need to be architected and built into the system rather than added on later during development. Therefore, an optimal implementation order of the requirements is desired. In the next section, we will illustrate how precedence constraint can influence the project plan, the release date, as well as the requirement selection.

1.3 Problem illustration

Here we first formally define precedence constraint—if requirement $R_{j'}$ can only start after requirement R_j is completely finished, then there is a precedence constraint between R_j and $R_{j'}$, denoted as $R_j \prec R_{j'}$. Usually, precedence constraints result from dependencies. It is clear that the precedence constraint can influence the development sequence of the requirements. However, the question is: as we have already selected the requirements based on the available capacity, will the precedence constraint also influence the project deadline of the release?

When there are precedence constraints and different development teams, scheduling requirements becomes a complex problem. Figure 1, provides an example of a time-schedule for the release planning problem in Table 1.

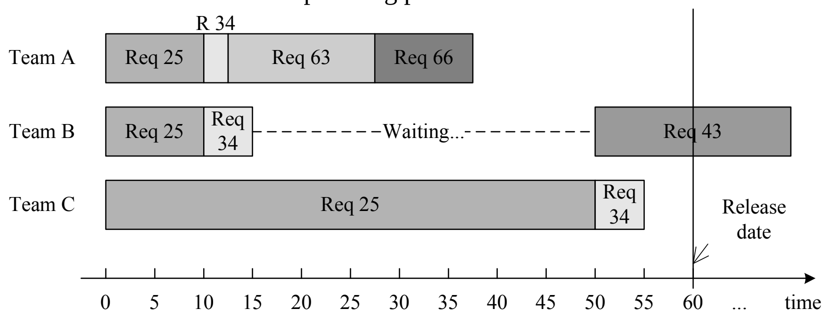

Figure 1: a numerical example of requirement scheduling problem

From Figure 1, it is clear that although the requirement selection does not exceed the teams' capacities, the project is delayed. The reason is that there is an *implication* dependency and hence a precedence constraints between requirement 25 and 43. Although team B finishes its task for R25 at day 10, it can not start to develop R43, which is dependent on R25's completion, because R25 is only available at day 50 when team C finishes its job. So, between day 10 and day 50, Team B only needs five days for R34 and the rest 35 days are wasted on waiting Team C. When R25 is finally available at day 50, it takes Team B another 33 days to develop R43, so the earliest date to finish the whole project is at day 83 instead of the expected day 60. Obviously, the time wasted on synchronization is not preferred. This raises an important issue how to design a schedule which makes teams not wasting time on waiting others? Or if this problem can not be eliminated (Results are shown later in chapter 6), how to minimize such waiting time and also minimize the total project span of the whole release project?

Another issue is: if we need to spend too much time on waiting for others, is that possible to re-select the requirements so that the release plan fits a predetermined deadline? For example, in the former case, if we still want to keep the 60 days as the deadline, then we need to re-select the requirements so that the newly selected requirements can be implemented within the time span. For the case in Figure 1, R43 has to be dropped to keep the project on time.

In this paper, we will focus on solving the two problems mentioned above: under the circumstances that there are both different development teams (or special skills) and precedence constraints:

1. How should we schedule the requirements to minimize the project make-span, i.e., the finishing time of the project?
2. How should we integrate the requirement selection and scheduling together so that the revenue is maximized and the project plan is on schedule?

The remaining of the paper is organized as follow. In Section 2, we first present the relationship between precedence constraint and the requirement dependencies. Then we show two special cases for requirement scheduling. Sections 3 and 4 provide ILP models for requirement scheduling and a combined method for requirement selection and scheduling. We discuss the prototypes we developed in Section 5. In Section 6, two simulations are presented to examine the influences of precedence constraint on requirement scheduling and the differences between the models. At last, we conclude the paper and provide future research directions in Section 7.

2 A first analysis

2.1 Precedence constraint & requirement dependency

Carlshamre et, al [7] identified six types of requirement interdependencies (listed in Table 2) for the release planning, and the first five are suggested and modeled as important factors for requirement selection [2, 8]. With respect to time, some of the dependencies can not only influence the requirement selection, but will also influence the requirement scheduling. For example, if requirement $R_{j'}$ requires R_j to function, it is normally better to start develop $R_{j'}$ after R_j is finished; or if requirement R_j influences the implementation cost of requirement $R_{j'}$, it is also considered better to implement R_j first [8]. So, together with the explicitly mentioned *time-related* dependency, also the *implication* and *cost-related* dependencies provide precedence constraints. Hence, when scheduling the requirements, we should take three out of the six types of requirement dependencies into consideration. Table 2 depicts the influence of dependencies on requirement selection and scheduling.

Dependency group	Dependency type	Influence requirement selection	Influence requirement scheduling
Functional dependency	*Combination*	✓	
	Implication	✓	✓
	Exclusion	✓	
Value-related dependency	*Revenue-based*	✓	
	Cost-based	✓	✓
Time-related dependency	*Time-related*		✓

Table 2: The influences of dependencies on requirement selection and scheduling

2.2 Scheduling with no precedence constraint

In Figure 1, we have illustrated the scheduling problem when there are precedence constraints and team divisions. However, scheduling will not be a problem if there are no precedence constraints between requirements. As each team works independently, and no synchronization is needed, they just need to randomly give a permutation of all the development tasks of the team, and perform them one after another. In this way, scheduling is not a problem and the deadline will not be exceeded.

2.3 Scheduling with no team division

If there are precedence constraints but no team or task division, scheduling the activities is also not a difficult issue. We can first draw a Directed Acyclic Graph (DAG) by setting the requirements R_j as vertexes and the precedence constraint $R_j \prec R_{j'}$ as a directed edge $(R_j, R_{j'})$. Then any topological sort [9] of the directed acyclic graph results in a feasible schedule. This sort provides a linear order of all the vertices such that if G contains an edge $(R_j, R_{j'})$, then R_j appears before $R_{j'}$. We can compute this sort in $O(V+E)$ time where V equals the number of requirements and E equals the number of dependencies. Because the development works continuously without interruption, the release deadline can also be kept.

3 An ILP model for requirement scheduling

To schedule the requirements exactly in time, there are two issues to consider: the available resources are limited and there are precedence constraints between the requirements. Within scheduling theory, the problem can be characterized as a special case of the resource constraint project scheduling problem (RCPSP) [21]. It is special because the resources all have capacity 1. RCPSP is an NP-Hard problem [5]. The problem complexity caused many scholars to development heuristics method [3] or exact algorithms [11]. Here, we present an ILP model of such problem.

3.1 Problem formulation

We are given a set of n requirements $\{R_1 \quad R_2 \quad \cdots \quad R_n\}$. Let m be the number of teams G_i $(i=1,2,\dots m)$. We denote a_{ij} as the amount of man days needed for Requirement R_j in team G_i. The development activity in team G_i for requirement R_j is considered as one individual job—each team works independently on one requirement and there is no predefined time restriction for the jobs within a requirement. Let us define a set $X=(J_1,J_2,\dots,J_k)$ of all the jobs with positive development time and there are k $(k \le m \times n)$ jobs in the set.

Because each job belongs to only one requirement, using this attribute, we can partition the set X into n disjoint subsets $\{X(R_1) \quad X(R_2) \quad \cdots \quad X(R_n)\}$ where $X(R_j)$ = { J_k | job J_k is for requirement R_j }, $(j=1,2,\dots n)$. Similarly, one job only belongs to one team, so we can partition the set X into m disjoint subsets $\{X(G_1) \quad X(G_2) \quad \cdots \quad X(G_n)\}$ where $X(G_i)$ = { J_k | job J_k is in team G_i } $(i=1,2,\dots m)$.

Assuming the number of developers in team G_i is Q_i, we find that the development time d_k for job J_k is a_{ij}/Q_j where $J_k \in X(R_j) \bigcap X(G_i)$.Here we assume that as soon as a team starts working on a job, it will continue work on it until the job is complete finished.

The precedence constraints

We can define a set $A=\{(R_j,R_{j'}) \mid R_j \prec R_{j'}\}$ which contains all the precedence constraints. We define the set H to show the precedence relationship between jobs:

$$H=\{(J_k,J_{k'}) \mid J_k \in X(R_j), J_{k'} \in X(R_{j'}), (R_j,R_{j'}) \in A\}$$

In this way, we set all the jobs of requirement $R_{j'}$ as the successors of the jobs of requirement R_j and we can make sure that any job in requirement $R_{j'}$ can only start after all the jobs for requirement R_j are finished.

We also need to introduce two virtual jobs, the start of the project and the end of the project. The job *START* must start before starting the jobs in X, the job *END* can only start when all the jobs X are finished. We consider the processing time of these two virtual jobs is 0. And the new job set with the two additional virtual jobs is X'.

If job J_k does not have any successor, then we set (J_k, END) in H. Or if job J_k does not have any predecessor then we put $(START, J_k)$ in H .

The precedent relationships between jobs can be represented by a directed acyclic graph $G=(X',H)$.

The upper bound of the project span

Let $T_{\max}$ be the upper bound of the project span. We can set the upper bound as $\sum_{j=1}^{n} \max(d_k \mid J_k \in X(R_j))$. The upper bound corresponds to developing requirements one after another, i.e. without any time overlap between different requirements.

The earliest start es_k and the latest start ls_k of each job J_k

For each job J_k, we can compute es_k (earliest possible start) and ls_k (latest possible start) as its time window to start. To compute the time interval, we first topologically sort the jobs, so that job J_k is before job $J_{k'}$ in the order if $(J_k, J_{k'}) \in H$.

We can use a longest path algorithm (forward recursion) to compute es_k . First, set $es_{START} = 0$, then we go through the jobs from *START* to *END* and set $es_k = \max_{(j,k)\in H} \left(es_j + d_j \right)$. Similarly, we can compute the latest start ls_k using a longest path algorithm (backward recursion). First, set $ls_{END} = T_{\max}$ then we go through the jobs from *END* to *START* and set $ls_j = \min_{(j,k)\in H} \left(ls_k - d_j \right)$

The (0,1) integer linear programming model

For the integer linear programming model we use a time-indexed formulation. This formulation has successfully been applied for machine-scheduling problems and is known to have a strong LP-relaxation lower bound (see e.g. [1] and [12]). We discretize time and the integer time t represents the period of $\left[t, t+1\right)$. For each job J_k we define a group of variable ξ_{kt} within the time interval $\left[es_k, ls_k\right]$, where t is the possible time for J_k to start. Now ξ_{kt} is a binary variable which equals 1 if and only if J_k starts at the beginning of period t . Then we can formulate the problem as follow:

$$\min \sum_{t=es_{END}}^{t=ls_{END}} t \cdot \xi_{ENDt} \qquad (3.1)$$

Subject to:

$$\sum_{t=es_k}^{t=ls_k} \xi_{kt} = 1 \ , \qquad \text{for all } J_k \in X' \qquad (3.2)$$

$$\sum_{t=es_k}^{t=ls_k} t \cdot \xi_{kt} + d_k \le \sum_{t=es_{k'}}^{t=ls_{k'}} t \cdot \xi_{k't} \qquad \text{for all } (J_k, J_{k'}) \in H \qquad (3.3)$$

$$\sum_{J_k \in X(G_i)} \sum_{\tau=\sigma(t,k)}^{t} \xi_{k\tau} \le 1 \qquad \text{for } t = \left(0, 1, \ldots T_{\max}\right), \ i = 1, \ldots, m \quad (3.4)$$

$$\xi_{kt} \in \left\{0, 1\right\} \qquad \text{for all } t \in \left[es_k, ls_k\right] \ , J_k \in X' \qquad (3.5)$$

where in constraint (3.4), $\sigma(t,k) = \max(0, t - d_k + 1)$. Constraint (3.1) shows the objective that we want to minimize the project span. Constraint (3.2) shows a job is started exactly once. Constraint (3.3) is the precedence constraint—one requirement can only start after its predecessor is finished. Constraint (3.4) means a development team can only develop at most one job at one time.

4 A combined model for requirement selection & scheduling

As we have seen, there is a risk that the selected set of requirements can not be scheduled in time. In most of the software development process models, the selection and scheduling are performed iteratively until a good solution is found [24]. However, doing it iteratively is not only difficult but also time-consuming because we need to constantly repeat the following 3 steps:

1. Drop some requirements so that the project plan is fit.
2. Re-fill in some requirements to take up the freed capacity.
3. Re-make project plan for the new group of requirements.

Because of the complexities of the knapsack model and the RCPSP model (they are both NP-Hard), without a proper search algorithm, it is very difficult to find a solution which can fulfill the goals of maximizing revenue and on time delivery. Even if such searching method is found, constantly calling these two NP-hard models will be very time consuming. A better method is demanded to solve this problem.

In this section, we will present a new ILP model which enables us to achieve the goals of maximizing revenue and on time delivery simultaneously. In the following section, we will present a model for combined selection and scheduling of the requirements when a fixed project deadline is given.

4.1 Formulating the ILP model

We can define the requirements R_j, the teams G_i, the jobs k and the dependency set A same as the in Section 3.1. In addition, each requirement R_j is associated with an expected revenue v_j. And we denote our planning period by T and define $d(T)$ as the number of working days in the planning period.

The precedence constraints

We can handle the precedence constraints similarly to Section 3.1, only that we do not need to introduce the two virtual jobs: *START* & *END* and do not need to link them to the jobs in X. It is because which requirements will be in the schedule is still uncertain and the release date is already fixed.

The earliest start es_k and the latest start ls_k of each job J_k

For the earliest start es_k, we can also use the longest path algorithm from Section 3.1. The only difference is since we do not have the virtual job *START* any more, we need to set the earliest start $es_k = 0$ for all the jobs which do not have predecessor. We can apply this lower bound because a requirement can only be selected and developed when all its predecessors are selected and developed. .

For the latest start ls_k, it equals $d(T) - d_k$. Please note that the method to compute ls_k is significantly different from the scheduling model. We can not lower this upper bound because we do not know whether the successors of a job will be selected.

It is possible that ls_k is less than es_k for a certain job *k.* It then means it can not fit in the project time span. So the requirement R_j which contains this job will also not be a candidate of the next release. Hence, we can eliminate these requirements beforehand and define a set X'' which only contain the feasible ones.

The (0,1) integer linear programming model

Like in [2], for each requirement R_j, we define a binary decision variable x_j associated to it, where $x_j = 1$ if and only if requirement R_j is selected. Moreover, for each job $J_k \in X''$, we define a group of binary decision variable ξ_{kt} within its possible time interval $t \in [es_k, ls_k]$, where $\xi_{kt} = 1$ if and only if job J_k starts at time t.

We can now model the combined selection and scheduling problem as follows:

$$\max \sum_{j=1}^{n} v_j x_j \tag{4.1}$$

Subject to

$$\sum_{t=es_k}^{t=ls_k} \xi_{kt} = x_j \qquad \text{for all } J_k \in X(R_j),\ j = 1,\ldots,n \tag{4.2}$$

$$x_{j'} \le x_j \qquad \text{for all } (R_j, R_{j'}) \in A \tag{4.3}$$

$$\sum_{t=es_k}^{t=ls_k} t \cdot \xi_{kt} + d_k \le \sum_{t=es_{k'}}^{t=ls_{k'}} t \cdot \xi_{k't} + (1 - x_{j'}) \cdot d(T) \qquad \text{for all } (J_k, J_{k'}) \in H,\ J_{k'} \in X(R_{j'}) \tag{4.4}$$

$$\sum_{k \in X(G_i)} \sum_{\tau=\sigma(t,k)}^{t} \xi_{k\tau} \le 1 \qquad \text{for } t = (0,1,\ldots T_{\max}),\ i = 1,\ldots,m \tag{4.5}$$

$$\xi_{kt}, x_j \in \{0,1\} \qquad \text{for all } t \in [es_k, ls_k],\ J_k \in X'',\ j = 1,\ldots,n \tag{4.6}$$

where in constraint 3.5, $\sigma(t,k) = \max(0, t - d_k + 1)$. The objective (4.1) shows that we want to maximize the revenue. Constraint (4.2) means that a requirement is selected if and only if all its jobs are planned. Constraints (4.3) and (4.4) deal with the precedence constraints. Constraint (4.3) means a requirement is only selected when its predecessor is selected. Constraint (4.4) means the jobs for the successor requirement can only start after all the jobs for its precedent requirements are finished. Please note that this constraint is different with the precedence constraint modeled in section 3.1, because the successor job is not guaranteed to be selected. (4.5) is the resource constraint that one team is only able to develop one requirement at a time. Constraint (4.6) is the binary constraint for all the variables.

Note that if we ignore the precedence constraints (4.3) and (4.4), it is another way to represent the multi-dimensional Knapsack problem.

4.2 Extensions of the model

Using the combined model, it is possible to model all the six types of requirement dependency listed in Table 2. *Combination*, *implication*, *exclusion* and *revenue-based* can be modeled the same way as in the knapsack model. Only the *cost-based* dependency is modeled differently. It is also possible to model the conditions when team G_i is only available for a certain time interval instead of the whole period, or there are holiday seasons within the period. For reasons of brevity, we refer to [20] for details.

5 Prototype

We have implemented three Java prototypes for requirement selection & scheduling based on the models available so far—the knapsack model, the scheduling model, and the combined model. These prototypes run in Linux environment and make use of the callable library of ILOG CPLEX [16] for solving the ILP problem. CPLEX is one of the best known packages for integer linear programming.

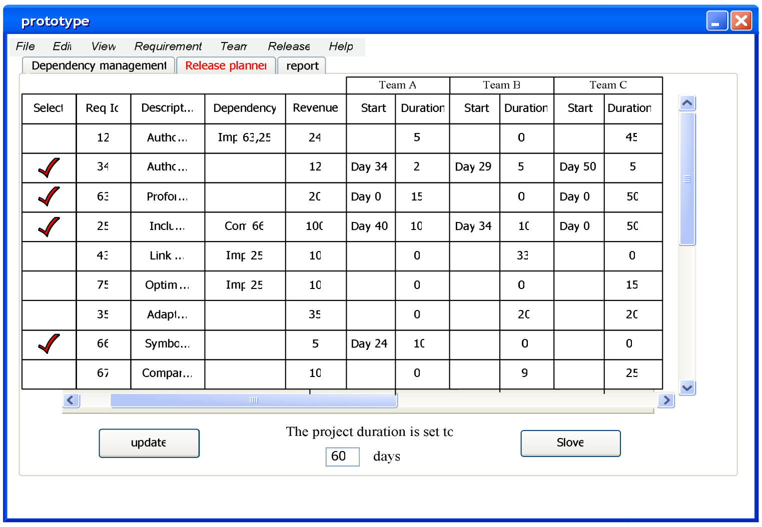

Figure 2: screen shot of the scheduling prototypes

Figure 2 shows a screenshot of the prototype for the combined model. The requirements are managed and stored in the database with estimated revenue, cost and dependency. This screenshot shows the interface of the model for combined requirement selection and scheduling. Based on the data attributes of the requirements and the expected release date, the requirements selection and a project plan for the next release are calculated simultaneously.

6 Simulation tests

In Section 1.3 we have shown that when there are different development teams and precedence constraints, the problem of synchronization can possibly delay the whole project. However, the size of this influence is still unknown. In addition, although the combined model for requirement selection and scheduling can guarantee on time delivery, the additional constraints will possibly cause a loss of revenue. The trade off between the time saving and the additional cost is also not clear. These concerns lead us to investigate the following questions through simulation tests:

Simulation 1: *What is the relationship between the number of time-related dependencies and the possibility of running out of time in the project planning?*

Simulation 2: *What are the differences when we select and schedule requirements at the same time, and when we select and schedule sequentially?*

For testing the programs and comparing the models, two types of datasets were used (available online [15] for research purpose). They were:

- **Small**: 9 requirements and 3 teams, release duration 60 days.
- **Master**: 99 requirements and 17 teams, release duration 30 days.

The Small dataset was the example dataset provide in Table 1. The Master dataset was generated from larger real life datasets. All team values were kept the same, but the team capacities and revenues were modified for confidentiality reasons.

In order to make the model not case specific, we randomly generated dependencies. We guaranteed that no cycle occurs within the dependencies. This is important because the requirements in the cycle would be inter-waiting others' completion and cause a deadlock. For the small dataset, we examine the situation with 1, 2, 3 and 4 dependencies, while for the master dataset, we check the situation with 0.5%, 1%, 2%, and 5% of the maximal number of possible dependencies (this equals $C_n^2 = n \cdot (n-1)/2$). Note that we here we mentioned the number of dependencies we explicitly generated. There may also be some additional dependencies induced by the generated dependencies, e.g. if R_i has to precede R_i and R_j has to precede R_k, then also R_i has to precede R_k. For every number of dependencies, we randomly generate 100 groups of dependencies and run 100 times.

6.1 Results of the simulation 1: the influence of dependencies on project plan

In this simulation, we want to exam how much precedence constraint can influence the project span. Given the small and master dataset, we first select requirement using the knapsack model, then we randomly generate a certain amount of dependencies and call the scheduling model to make a project plan. We then find the maximal, minimal and average make-span, i.e. duration of the project and count how many times the project is delayed within the 100 runs. At last, we compare the results with the lower bound. The lower bound is the maximum value of the project make-span without precedence constraints and the result of longest path algorithm, which relaxed the constraint on team difference (i.e. es_{END} in Section 3.1). Table 3 shows the results of the 100 runs each row.

Data Set	Dep ratio	No. Dep	The project span			Times of delay	The difference between lower bound		
			Max days	Min days	Average days		Max diff	Min diff	Average diff
Small-result (5 Reqs, 60 days)	10%	1	83	55	58.80	16	0.00%	0.00%	0.00%
	20%	2	93	55	63.70	40	27.27%	0.00%	0.93%
	30%	3	103	55	70.42	62	27.27%	0.00%	2.64%
	40%	4	108	55	75.32	76	14.55%	0.00%	2.12%
Master-result (76 Reqs, 30 days)	0.5%	14	40	30	30.93	33	30.00%	0.00%	2.70%
	1%	29	46	30	31.38	27	8.57%	0.00%	0.22%
	2%	57	69	30	36.92	76	22.58%	0.00%	2.13%
	5%	142	84	38	56.15	100	19.23%	0.00%	3.47%

Table 3: schedule results of the first simulation

To visualize the results, we plot the result of master data set in the following chart. The result of small dataset keeps the same trend as the master one.

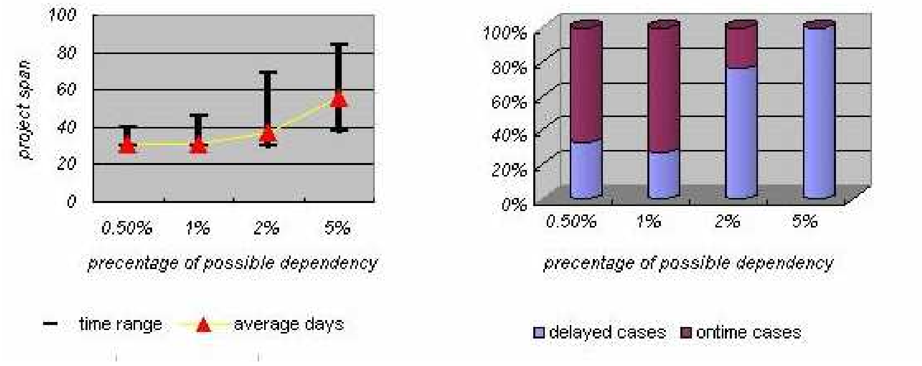

Figure 3: schedule results based on the master dataset

In figure 3, the left chart shows the dependency's influence on project span and the right chats shows the ratio of the delayed cases and on-time cases. Although the requirements selected using knapsack model are expected to finish within 30 days, the results vary a lot. When there are 0.5% or 1% of possible dependencies, the results of the 100 runs range within a few days, the average project span is close to the release date and the number of over-time cases is still low. The result starts to explode after 2%. Then the project span varies a lot based on different dependencies and is on average much higher than expected. Especially when there are 5% of possible dependencies, the minimal case requires 38 days which means none of the 100 run are on time.

It is not difficult to conclude that precedence constraints play an important role for release scheduling. When there are just a few dependencies, they can already greatly influence the project span. And as the number of dependencies grows, the project span also grows significantly. Based on the complexity of the system, the exact number of dependencies may vary a lot, but a former survey [8] has suggested that there are at least 80% of requirements are interdependent and most of them are *implications* and *cost-based*, then we can assume that the exact number of dependency is at least higher than the second row of the small and master dataset.

6.2 Results of the simulation 2: model comparison

In this simulation, we compare the differences between applying the knapsack and scheduling model subsequently (k&s), and the combined model (comb). We take the following three steps to compare the models. Step 1, based on the small and the master datasets, we randomly generate a group of dependencies. Step 2, we then use the knapsack model to select the requirements and record down the dependencies within the selected requirements, and we call the scheduling model to schedule the activities exactly in time. Step 3, for the same dataset and dependencies we call the combined model to select and schedule the requirement at the same time. Step 4, we compare the revenue difference between the knapsack model and the combined model; the time difference between the scheduling model and release date (which is the schedule result of the combined model) and the times of delay.

When analyzing the results, we found that when the combined model and the knapsack model select the same requirements, the scheduling model can always find a timely schedule. The result is not surprising but also of no interest since everything is the same. So we decided to also make a statistics only for the delayed cases. The computational results are shown in Table 4.

Data Set	Dep ratio	No. of Dep	Statistics for the 100 runs			Statistics **only** for the delayed cases					
			Average revenue (comb)	Average revenue (k&s)	Average project span (k&s)	No. of delay (k&s)	Average revenue (comb)	Average revenue (k&s)	Average project span (k&s)	Average revenue diff	Average time diff
Small (9 Reqs 60 days)	3%	1	139.17	141.27	56.62	9	123.67	147	73	15.87%	21.67%
	10%	3	128.06	132.53	58.15	17	110.53	136.82	76	19.15%	26.67%
	15%	5	114.81	121.45	59.25	22	99.27	129.45	76.59	22.92%	27.65%
	20%	7	105.59	110.87	57.72	24	104.02	126.14	76.07	16.84%	26.78%
Master (99 Reqs 30 days)	0.5%	24	40420.1	40429.5	30.48	17	40442.1	40493.5	32.82	0.13%	9.41%
	1%	48	39275.5	39479.1	32.62	45	38965.7	39400.9	35.82	1.15%	19.41%
	2%	97	35581.6	36103.1	36.41	68	35351.8	36118.7	39.43	2.11%	31.42%
	5%	242	26947.7	29127.3	45.61	95	26804.5	29098.8	46.43	7.84%	54.77%

Table 4: simulation results of model comparison

The results prove again that precedence constraints play an important role for requirement selection and scheduling. As the number of constraint increase, the average revenue of the two models decrease and the average project plan as well as the possibility of delay increase. To compare the models, we plot the computational results of master dataset in the Figure 4.

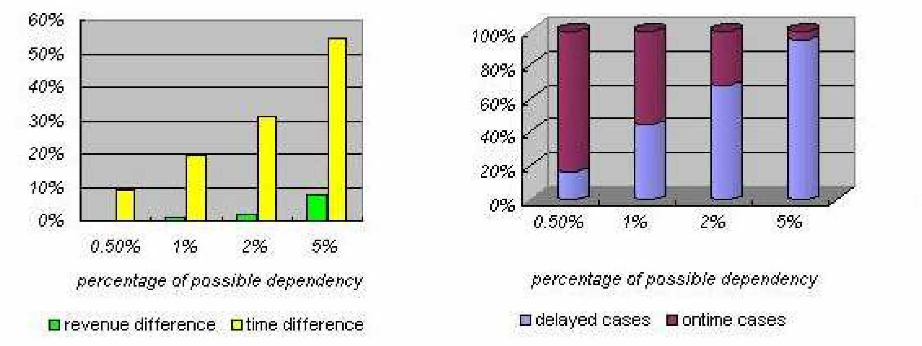

Figure 4: model comparison result based on master dataset

In Figure 4, the left chart shows the average revenue difference and cost difference for the delayed cases and the right chart shows ratio of on-time cases and delayed cases. It is clear that the combined model can not only guarantee on time delivery but also gain more efficiency. When follow the select and then schedule process, the project stand a high change of being delayed and this possibility grows larger and larger as the number of dependencies increases. The simulation result also suggests that it is more efficient to take the project plan issues into account when selecting the requirements, because even if we ignore the influence on missing the deadline, the revenue loss of the combined model is significantly less than the additional development time.

7 Conclusion and future research

In this paper we present two ILP models to include requirement scheduling issues into software release planning. The scheduling model can schedule the requirements so that the project make-span is minimized and the resource and precedence constraints are satisfied; the combined model maximizes revenue while ensuring on-time delivery of the project and simultaneously presents a project plan.

Simulations have demonstrated the application of the models. The results indicate that the model for combined requirement selection and scheduling can not only keep on-time-delivery but also be more efficient than the traditional knapsack model.
The results looks very promising, but some more works still needs to be done. The second simulation results show convincing figures to combine the requirement selection and scheduling together. More work is needed to evaluate this process improvement opportunity. The first simulation results also suggest that the optimal schedule found by integer linear programming is not far away from the critical path lower bound. It can be interesting to investigate if there are faster algorithms for scheduling that can get rather close to the optimum.

References

1 Akker J.M. van den, C.P.M. van Hoesel, and M.W.P. Savelsbergh (1999). A Polyhedral Approach to Single-Machine Scheduling Problems. *Mathematical Programming* 85 (3), 541-572.

2 Akker, J.M. van den, Brinkkemper, S., Diepen, G., & Versendaal, J.M. (2005). Flexible Release Planning Using Integer Linear Programming. In E. Kamsties, v. Gervasi, & P. Sawyer (Eds.), *Proceedings of the 11th International Workshop on Requirements Engineering for Software Quality (REFSQ'05)* (pp. 247-262)

3 Balakrishnan, R, and W. J, Leon (1993). *Quality and Adaptability of Problem-Space Based Neighborhoods for Resource Constrained Scheduling*, Working Paper, Department of Industrial Engineering, Texas A & M University, College Station, TX.

4 Berander, P., Andrews, A. (2005). Requirements Prioritization. In: *Engineering and Managing Software Requirement*s, A. Aurum and C.Wohlin. Berlin, Germany, Springer Verlag.

5 Blazewicz, J., J. K. Lenstra, and A. H. G. Rinnooy Kan (1983). Scheduling Projects Subject to Resource Constraints: Classification and Complexity, *Discrete Applied Math.*, 5, 11-24

6 Pär Carlshamre, Björn Regnell (2000). *Requirements Lifecycle Management and Release Planning in Market-Driven Requirements Engineering Processes*. International Workshop on the Requirements Engineering Process: Innovative Techniques, Models, and Tools to support the RE Process, 6th-8th of September 2000, Greenwich UK, the DEXA Conference

7 Carlshamre P, Sandahl K, Lindvall M, Regnell B, Natt och Dag J (2001). An industrial survey of requirements interdependencies in software release planning. In: *Proceedings of the 5th IEEE international symposium on requirements engineering*, pp 84–91

8 Pär Carlshamre (2002). Release Planning in Market-Driven Software Product Development: Provoking an Understanding. *Requirements Engineering*, Volume: 7, Issue: 3, pp: 139-151

9 Thomas H.Cormen, Charlse E. Leiserson, Ronald L. Riverst, Clifford Stein (2001). *Introduction to algorithms*, second edition. MIT Press, pp 549 -551

10 Cusumano, M.A. (2004). *The Business of Software*. Free Press.

11 Demeulemeester,E, W.Herroelen (1992). A Branch and Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem *Management Science* 38,1803-1818.

12 Dyer, M. and L. Wolsey (1990), Formulating the Single Machine Sequencing Problem with Release Dates as a Mixed Integer Program. *Discrete Applied Mathematics* 26, 255-270.

13 Firesmith, D. (2004). Prioritizing Requirements, *Journal of Object Technology*, vol 3, no 8, pp 35 – 47.

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

15 http://www.cs.uu.nl/~diepen/ReqMan

16 ILOG CPLEX, http://www.ilog.com/products/cplex

17 Jung, H. –W. (1998) Optimizing Value and Cost in Requirements Analysis, *IEEE Software*, July/August 1998 pp 74 – 78.

18 Karlsson, J and Ryan, K. (1997). A cost-Value Approach for Prioritizing Requirements, *IEEE Software*, September/October 1997 pp 67-74.

19 Leffingwell, D., Widrig, D. (2000) *Managing Software Requirements – A Unified Approach*, Addison-Wesly, Upper Saddle River, NJ.

20 C.Li. (2006) *An Integer Linear Programming Approach to Product Software Release Planning & Scheduling*". Master thesis of Utrecht University, pp 22 -71

21 A. Mingozzi, V. Maniezzo, S. Ricciardelli, L. Bianco (1998). An Exact Algorithm for the Resource-Constrained Project Scheduling Problem Based on a New Mathematical Formulation. *Management Science*, Vol. 44, No. 5. , pp. 714-729.

22 Novorita, R., Grube, G. (1998). Benefits of Structured Requirements Methods for Market-Based Enterprises, *Proceedings of International Council on Systems Engineering Sixth Annual International Symposium on Systems Engineering: Practice and Tools (INCOSE'96)*, Boston USA.

23 Regnell, B, Höst, M., Natt och Dag, J, Beremark, P. Hjelm. T. (2001) .An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software. *Requirement Engineering*, vol 6, no 1, pp 51-62

24 Regnell, b. and Brinkkemper, S. (2005), Market-Driven Requirements Engineering for Software Products. In: *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin (eds.), Berlin, Germany, Springer Verlag pp. 287-308

25 Ruhe, G., Saliu, M.O. (2005). The Art and Science of Software Release Planning, *IEEE Software*, vol 22, no 6, pp. 47-53

26 Sawyer, P., Sommerville, I., Kotonya, G. (1999). Improving Market-Driven RE Processes, *Proceedings of International Conference on Product Focused Software Process Improvement (PROFES'99*), Oulu Finland, June 1999

27 Inge van de Weerd, Sjaak Brinkkemper, Richard Nieuwenhuis, Johan Versendaal, Lex Bijlsma (2006) *A reference framework for software product management*. Utrecht University Technical Report UU-CS-2006-014.

28 Wolsey L.A. (1998). *Integer Programming*. Wiley-Interscience Series In Discrete Mathematics and Optimization. 1998