Flexible Release Composition using Integer Linear Programming

Marjan van den Akker, Sjaak Brinkkemper,
Guido Diepen, Johan Versendaal

institute of information and computing sciences, utrecht university

technical report UU-CS-2004-063

www.cs.uu.nl

Flexible Release Composition using Integer Linear Programming[*]

Marjan van den Akker, Sjaak Brinkkemper, Guido Diepen, Johan Versendaal
Insitute of Information and Computing Sciences,
Universiteit Utrecht, P.O. Box 80089, 3508 TB Utrecht, The Netherlands.
e-mail: {marjan, s.brinkkemper, g.diepen, jversend}@cs.uu.nl

December 6, 2004

Abstract

For software vendors, the process to determine the requirements for the next release of a software product is often difficult. In this paper we present a mathematical formalization of release composition with a corresponding optimization tool that aims to support product managers and development project managers during release planning. The tool is based on integer linear programming and assumes that an optimal set of requirements is the set with maximum projected revenue against available resources in a given time period. The input for the optimization is twofold. Input data like the list of candidate requirements, estimated revenue and required team resources per requirement, whether or not a requirement is mandatory, comprise the first type of input. Secondly, several managerial steering mechanisms provide flexibility in the optimization environment: team composition, permitting of team transfers, extension of deadlines, and hiring external resources. Through experiments based on real-life data we make a sound case for the applicability of our proposal.

1 Requirements selection for release planning

Companies developing software products face complex challenges when determining requirements for upcoming releases (see e.g. Potts (1995); Regnell and Brinkkemper, (2005)). Often the wish-list of requirements extends the capacity of available resources, requirements may not be unambiguous, they may be difficult to prioritize, etc. As a matter of fact, many aspects influence the definition of an optimal set of requirements for a next release. Several

[*]This research was partially supported by the Bsik-ICT project BRICKS ('Basic Research in Informatics for Creating the Knowledge Society').

scholars have presented lists of such aspects including: importance or business value, personal preference of certain customers and other stakeholders, penalty if not developed, cost of development in man days, development lead-time, requirement volatility, requirement dependencies, ability to reuse, and requirements quality (e.g. Berander and Andrews (2005); Firesmith (2004); Ruhe et al. (2003); Natt och Dag et al. (2004); Natt och Dag et al. (2005)).

In order to deal with this multi-aspect optimization problem different techniques and procedures have been applied. The analytical hierarchy process (Saaty (1980); Karlsson and Ryan (1997); Regnell et al (2001)) assesses requirements according to certain criteria by taking all possible requirement pairs, relatively valuing each pair, and subsequently using matrix calculations to determine a weighted list of requirements. Jung (1998) extended the work of Karlsson and Ryan (1997) by reducing the complexity of applying the analytical hierarchy process to large amounts of requirements using linear programming techniques. Through cumulative voting (Leffingwell and Widrig (2000)) different stakeholders are asked to distribute a fixed amount of units (e.g. euros) between all requirements, from where an average weighted requirement list is constructed. With discrete event simulation the effect of development of requirements is modeled, allowing what-if analysis (Höst et al. (2001); Ruhe et al. (2003)). For more techniques, see for example Berander and Andrews (2005), who provide an extensive list of requirements prioritization techniques.

In this paper we develop and demonstrate an optimization technique, based on integer linear programming, to support software vendors in determining the next release of a software product. As with Jung's approach (Jung, 1998), our technique is based on the assumption that a release's best set of requirements is the set that results maximum projected revenue against available resources in a given time period. We take into account practical aspects, including the total list of requirements, whether or not requirements are dependent on one another, a requirement's projected revenue, and a requirement's resource claim per development team. To further increase its practicable application, we enhance our technique with managerial steering mechanisms, i.c. enabling of team transfers, conceding release deadline extension, allowing extra resources, and more. By introducing these aspects and managerial steering mechanisms into integer linear programming models for the release composition process we extend the work of Jung (1998).

Table 1 depicts a simplified representation of the problem domain. For nine requirements with given input (revenue in euros and required man days per team) the best set of requirements for a next release need to be defined. Suppose for instance that the total amount of available man days in the three teams is 60, then we see that team A has room for additional work, where team B and C have too much work. Then the set of requirements that brings the maximum revenue has to be determined, given the fact that there are several management options: extending the deadline so that each team has more than 60 days, transferring employees from team A to team C, or hiring external capacity. Each option has its pros and cons, and therefore a solution approach is required that comprises all options into one model. Note that in reality it is not uncommon to include tens to even hundreds of requirements, making release planning without tooling an extremely difficult task.

Requirement	**Revenues**	**Total**	**Team 1**	**Team 2**	**Team 3**
Authorization on order cancellation and removal	24	50	5		45
Authorization on archiving service orders	12	12	2	5	5
Performance improvements order processing	20	15	15		
Inclusion graphical plan board	100	70	10	10	50
Link with Acrobat reader for PDF files	10	33		33	
Optimizing interface with international Postal code system	10	15			15
Adaptations in rental and systems	35	40		20	20
Symbol import	5	10	10		
Comparison of services per department	10	34		9	25
Totals	**226**	**279**	**42**	**77**	**160**

Table 1: Example requirements sheet with estimated team workload and revenues

As described above, several authors have already discussed and presented aspects, techniques, and tooling for release planning. The novelty of our research, however, is threefold. Firstly, we take into account a unique set of aspects, among others needed team capacity per requirement. Secondly, we use unique yet practical managerial steering mechanisms that can aid product managers and development project managers in release planning, notably enabling of team transfers, deadlines and extra resources. In the third place we show how to define aspects and managerial steering mechanisms using integer linear programming. The outcomes of our optimization tool are values for so-called decision variables and include the list of requirements for the next release, needed team transfers (if enabled), needed additional team resources to include other requirements, and the needed deadline extension (if enabled).

In Section 2 our paper discusses integer linear programming models as applied in the optimization tool. Firstly we model our initial assumption: the best set of requirements for a next release is the set that maximizes projected revenue against available resources. Secondly, we describe different scenarios of managerial steering, including team transfers and deadline extension in our environment. We conclude Section 2 with a discussion on solving integer linear programming problems in general. In Section 3 we describe the software tool, called ReqMan, through which we demonstrate the practical application of our optimization approach. In closing, we conclude and provide areas for future research in Section 4.

2 Integer linear programming models

In this section we formulate requirements management in product software companies as a combinatorial optimization problem. In such a problem we have to find the best from a finite but very large number of solutions. These type of problems occur in many areas within production planning, logistics, transportation, personnel planning and telecommunication. The most famous example is the traveling salesman problem. An example related to requirements management is the problem of a person or company getting a number of tasks each with a specific duration and deadline. It gets more tasks than it can handle in time. However, the customer only pays if the work is completed in time. This means that a selection of tasks to be executed has to be made such that the selected tasks are completed on time and the revenue is maximized. In scheduling theory, this problem is known as minimizing the (weighted) number of tardy jobs. A survey on solution methods for these type of problems with a single resource, i.e. person or company, is given in van den Akker and Hoogeveen (2004A) and van den Akker and Hoogeveen (2004B) consider the case with a single resource and uncertainty in the processing times of the jobs.

Integer linear programming is a well-known technique for solving combinatorial optimization problems. In general, integer linear programming problems are NP-hard. However, using advanced algorithms and specialized software, these problems can be solved within a reasonable amount of time. For a general introduction to integer linear programming we refer to Wolsey (1998). For examples in the area of scheduling we refer to van den Akker, van Hoesel and Savelsbergh (1999), and van den Akker, Hoogeveen and van de Velde (1999).

We will discuss different variants of the problem of version definition, i.e., selecting requirements for the next release of the software product. We are given a set of n requirements $\{R_1, R_2, \ldots, R_n\}$. Suppose that for each requirement R_j we can estimate its revenue v_j. The implementation of each requirement needs a given amount of resources in the form of labor hours from the development teams. We assume that the date of the next release is given; hence we have to deal with a fixed planning period. Clearly, the available amount of resources is limited. Therefore, we have to make a selection of the requirements to be included in the next release, preferably, such that the revenue is maximal. This can be viewed as the following optimization problem: find the subset of requirements for the next release such that the revenue is maximal and the available capacity is not exceeded.

2.1 One pool of developers

In the first variant we only deal with the total amount of man days available in the company. We denote our planning period by T and define $d(T)$ as the number of working days in the planning period. Moreover, let Q be the number of persons working in the development teams of the company. The available capacity then equals $d(T)Q$ man days.

Moreover, we have an estimate a_j of the amount of man days needed for the implementation of requirement R_j. We model the requirements selection problem in terms of binary variables x_j $(j = 1, \ldots, n)$, where

$$x_j = \begin{cases} 1 & \text{if requirement } R_j \text{ is selected;} \\ 0 & \text{otherwise.} \end{cases}$$

The problem can be modeled as an integer linear programming problem in the following way:

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

subject to

$$\sum_{j=1}^{n} a_j x_j \leq d(T)Q, \tag{1}$$

$$x_j \in \{0,1\}, \qquad \text{for } j = 1, \ldots, n.$$

This problem is known as the binary knapsack problem (see in general Marthello and Toth (1990), and specifically Jung (1998) for its requirements analysis application). If the company decides that some of the requirements have to be included in the new release in any case, this can be achieved by fixing the corresponding variables x_j at 1, i.e. adding the equation $x_j = 1$ to the above model. If the number of working days in the planning period is different for different persons the total capacity is given by $\sum d_p(T)$, where $d_p(T)$ is the number of working days of person p in period T and the sum is over all persons in the company.

2.2 Different development teams

In the previous model, there was only one pool software developers. Usually there are different development teams within a software company, each having their own specialization. So the above model may be too optimistic, since it did not take the individual team capacities into account.

Let m be the number of teams and suppose team G_i $(i = 1, \ldots, m)$ consists of Q_i persons. We assume that the implementation of requirement R_j needs a given amount a_{ij} of man days from team G_i $(i = 1, \ldots, m)$. Now the team capacities can be included by replacing constraint (1) by:

$$\sum_{j=1}^{n} a_{ij} x_j \leq d(T)Q_i, \quad \text{for } i = 1, \ldots, m. \tag{2}$$

Note that for $m = 1$, this coincides with the model for one pool of developers. This problem is known as the binary m-dimensional knapsack problem (see Crescenzi and Kann). Clearly, the model can be adapted to the situation with different amounts of man hours per person.

2.3 Dependent requirements

Until now we have assumed that all requirements can be implemented independently. Now suppose that the functionality imposed by a certain requirement can only be effective if one or more other requirements are also implemented, say if we select requirement R_j we also have to select R_k. In the model, we have to ensure that

$$x_j = 1 \Rightarrow x_k = 1.$$

This can be done by extending our model with the linear inequality

$$x_j \leq x_k. \tag{3}$$

2.4 Team transfers

By allowing people to work in a different team, there is more flexibility which can increase revenue. We will call this transfers. For an individual person a transfer will probably result in a decrease of efficiency because the person has less experience with working in another team. We assume that if a person from team G_i works in another team G_k his contribution in terms of man days is multiplied by α_{ik} , i.e. if the person works one day this contributes only α_{ik} day to the work delivered by team G_k. Sometimes, transfers are not possible for all combinations of teams. For example a company with development teams in different locations may only transfer people within a location. We can use the factor α_{ik} to reflect the feasibility of a transfer as follows:

- $\alpha_{ik} = 0$ if a transfer from G_i to G_k is infeasible, for example because the specializations of the teams differ too much or because of the traveling distance between the locations of the teams.
- $\alpha_{ik} = 1$ if a person from team G_i can do the work from team G_k without any additional effort e.g. if the work in G_i and G_k is very similar
- $0 < \alpha_{ik} < 1$ otherwise.

Note that α_{ik} may be different from α_{ki}, for example if the work in teams G_i and G_k is in similar areas, but the work in G_i is more complicated than that in G_k. Then α_{ik} is considerably larger than α_{ki}.

We assume that the amount of time for which a person can be transferred is a multiple of the so-called *capacity unit* which is denoted by U_{cap}. If people can be transferred per day then U_{cap} will just be 1. The other extreme is that only full-time transfers are allowed. Then U_{cap} corresponds to the complete planning period, i.e., $U_{cap} = d(T)$. If people can only be transferred for (a number of) complete weeks then U_{cap} will be equal to 5.

Besides the variables x_j, we now define the variables y_{ik} as the number of capacity units from team G_i deployed in team G_k. Let m_i be the number of capacity units in team G_i. Then

$$m_i = \frac{d(T)}{U_{cap}} Q_i.$$

Including transfers results in the following model:

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

subject to

$$\begin{aligned}
\sum_{j=1}^{n} a_{ij}x_j \leq U_{cap}[y_{ii} + \sum_{k:k\neq i} \alpha_{ki}y_{ki}] & \qquad \text{for } , i = 1, \ldots, m, && (4)\\
\sum_{k=1}^{m} y_{ik} = m_i, & \qquad \text{for } i = 1, \ldots, m, && (5)\\
x_j \in \{0, 1\}, & \qquad \text{for } j = 1, \ldots, n,\\
y_{ik} \text{ nonnegative and integral}, & \qquad \text{for } j = 1, \ldots, n.
\end{aligned}$$

Inequality (4) states that the work of team G_i on the selected requirements is at most the capacity delivered by people working in their own team plus the capacity obtained from people outside the team. Equation (5) ensures that the number of persons from team G_i working in the different teams exactly equals the team size Q_i, i.e. nobody gets lost. The above model is a modification and extension of the model from Section 2.2. Clearly, tranfers are not applicable with one pool of developers so we must have $m > 1$.

Note that if only full-time transfers are allowed, then y_{ik} is just the number of persons from team G_i working in team G_k. By deleting the integrality constraints on the variables y_{ik} persons can get any fractional division over teams.

Observe that in the above model it is possible that for example 2 persons are transferred from team A to team B and 1 person from team B to team C, i.e. team B is extended by transfers and sends persons to other teams simultaneously. Since each transfer decreases total capacity, this situation is inefficient and it will occur in an optimal solution only if it still enables the maximal revenue. However, it is not desirable and we can exclude it in the following way. Define a binary variable z_i which equals 1 if people from team G_i are transferred to other teams and 0 otherwise. Now we can add for each team G_i the constraints:

$$\begin{aligned}
\sum_{k:k\neq i} y_{ik} &\leq m_i z_i, && (6)\\
\sum_{k:k\neq i} y_{ki} &\leq (M - m_i)(1 - z_i), && (7)
\end{aligned}$$

where $M = \sum_{i=1}^{n} m_i$. Inequality (6) ensures that team G_i can only send capacity to another team when $z_i = 1$ and inequality (7) ensures that other teams can only transfer capacity to team i if $z_i = 0$.

One can think of situations where the restrictions (6) and (7) are not desirable. Suppose that transfers from team A to B and from team B to C are feasible, but transfers from A to C are not. If there is lack of capacity in team C, this can be solved by transferring form B to C. When this leads to lack of capacity in team B, these can be compensated by transfers from A to B.

2.5 External resources or deadline extensions

To increase the capacity for the development of the next release, the company may consider to hire external personnel in some of the development teams. Clearly, this brings a certain cost. In our model, this can be included by to adding the external capacity to the right-hand side of constraints (2) and including the cost in the revenue function. For simplicity, we assume that the cost of external capacity are linear. We define q_i as the unit cost of hiring external capacity in team G_i, i.e., if u_i is the amount of additional man days hired in team G_i, then the cost are $q_i u_i$. The value of q_i depends on the team specialization. Similar to the case of transfers, we assume that the contribution of u_i external man days is given by $\alpha_{ei} u_i$, where $0 < \alpha_{ei} < 1$. Finally, we assume that there is a maximum available budget for external capacity; this budget is denoted by E. This results in the following model which is an extension of the model from Section 2.2:

$$\max \sum_{j=1}^{n} v_j x_j - \sum_{i=1}^{m} q_i u_i$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \leq d(T) Q_i + \alpha_{ei} u_i, \qquad \text{for } i = 1, \ldots, m, \tag{8}$$

$$\sum_{i=1}^{m} q_i u_i \leq E \tag{9}$$

$$u_i \text{ nonnegative and integral}, \qquad \text{for } i = 1, \ldots, m,$$

$$x_j \in \{0, 1\}, \qquad \text{for } j = 1, \ldots, n.$$

Clearly, this extension also applies to the case with one pool of developers.

Another possibility is postponing the delivery date for the new release. Suppose the delivery date is postponed by δ_T *working days*, and that the estimated cost are C per day. This can be included in a similar way. Now, δ_T is a (integer) variable in the integer linear program, the revenue function is $\sum_{j=1}^{n} v_j x_j - C\delta_{d_T}$ and the right-hand side of inequality (8) is changed into $(d(T) + \delta_T) Q_i$.

Summarizing, we have presented in the previous sections different models taking into account different aspects and managerial steering mechanisms within a software product company. The team structure of the company determines the number of man hour capacity constraints. In case of one pool of developers we have one such constraint (see Section 2.1) and in case of different teams we have a capacity constraint for each team (see Section 2.2). Based on the team structure, we obtained two basic models, where the model with different teams is a generalization of the model with one pool of developers. We showed how to extend the model dependent requirements. Moreover we studied team transfers, hiring external team capacity and deadline extensions. For presentational reasons, each of these issues were included separately. However, depending on the relations between the requirements and the used managerial steering mechanisms, a combination of the presented models can be applied.

2.6 Solving integer linear programming problems

For the sake of completeness, we elaborate in this section on the general solution method of integer linear programming problems and the fact that even if the problem is not fully solved to optimality a good solution may be available. The described method is used in most integer linear programming (ILP) software packages. For more information, the interested reader is referred to Wolsey (1998).

All the problem formulated in this section are NP-hard. In general, integer linear programming problems are NP-hard. This implies that it is very unlikely that there exists an algorithm that is guaranteed to find the optimal solution in time that is polynomial in the input size. Finding the optimal solution requires an amount of time which in the worst case grows exponentially with the problem size.

If in a given ILP we relax the integrality conditions, i.e., 'x integral' is replaced by $x \geq 0$ and $x \in \{0, 1\}$ by $0 \leq x \leq 1$, we obtain a linear program which is called the *LP-relaxation*. This problem can easily be solved by e.g. the simplex method. In case of a maximization problem, the LP-relaxation provides an upper bound on the optimal value of the ILP.

The first step to solve an ILP is always to solve the LP-relaxation. If the solution of the LP-relaxation is integral, we are done. Otherwise, we start with a branch-and-bound tree. The ILP is split into two or more subproblems corresponding to two nodes of a tree, for example by fixing a variable x_j to 0 in one node, i.e. omit requirement R_j in the release, and to 1 in the other node, i.e. select requirement R_j in the new release. (This is the branching part). The algorithm starts evaluating one of the nodes. First the LP-relaxation is solved. If the solution is integral, the node is finished and the best-known integral solution is updated, if necessary. If the LP-relaxation is infeasible, clearly the node is finished as well. If the values of the LP-relaxation is lower than the best known integral solution, the node can be skipped from further consideration since we have no hope of finding the optimal solution there. (This is the bounding part). Otherwise, new nodes are generated by branching, i.e. by selecting or omitting another requirement.

Since we maintain the best known integral solution and we have an upper bound from the LP-relaxation, we have a solution with a quality guarantee from the moment at which an integral solution is found. This allows us to stop if the solution is guaranteed to be within a certain margin from the optimum. This can be beneficial, because it occurs quite often that the optimal solution is found quickly and it takes a lot of time to prove that the solution is indeed optimal.

3 Experiments

To obtain a proof-of-concept we implemented a prototype of a requirements selection system. For testing the models we used two different ILP software packages. The first package we used to solve the models is the solver included in Microsoft Excel (professional version). This solver suffices for problems of small size, but as soon as the problems become bigger in the number of variables (for example by means of introducing transfer variables) the solver will notify that it is not able to solve the problem because the number of variables is just to big. To be able to also solve the larger problems a Java program was implemented. This prototype has a graphical interface and it makes use of the callable library of ILOG CPLEX (see [6]) for solving the ILP problems. CPLEX is one of the best known packages for integer linear programming.

For testing the program different data sets were used. These different data sets are the following:

- **Small**: 9 requirements and 3 teams.
- **AA, BB, CC**: 24 requirements and 17 teams. Ratio of available and total capacity approximately 50, 60 and 70 percent.
- **Master**: 99 requirements and 17 teams.

All of the used data sets are available online[1] for research purposes.

The Small data set consists of fictitious data. The AA, BB, CC and Master data sets were generated from larger real life data sets. All team values were kept, but revenues were randomly generated and afterwards some were edited to create more interesting problems. For confidentiality reasons it is not possible to expose the real data to the general public.

When no transfers between the teams are allowed all of the above problems are still solvable by the solver in Microsoft Excel. As soon as team transfers are allowed, the number of variables in the problem becomes too big for this solver and we have to use CPLEX to solve these problems.

For some of the bigger problems even CPLEX has some difficulties, in the sense that because of the large number variables it takes CPLEX too much time to solve the problem to full optimality. In these cases the solution that the CPLEX solver has found, has a maximum error with regards to the optimal solution that is very small.

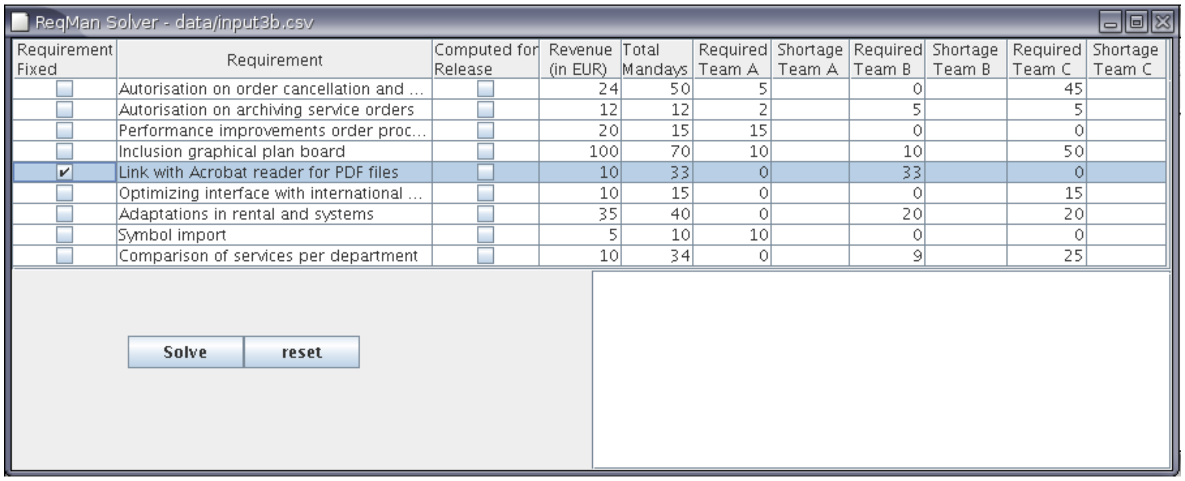
The following table gives the solution values for the different problems that were tested. The first column corresponds to the model introduced in Section 2.1 where there is just one big

[1]http://www.cs.uu.nl/~diepen/ReqMan

pool of resources. The second column corresponds to the model with teams but without transfers between the teams, introduced in Section 2.2. The last three columns correspond to the model introduced in Section 2.4 and each of the three columns corresponds to a different transfer unit. For the last three columns $\alpha_{ik} = 0.7$ for all $i \neq k$ was used.

Data	Pool	Teams	$U_{cap} = 10$	$U_{cap} = 5$	$U_{cap} = 1$
small	182	147	177	182	182
AA	700	510	620	685	685
BB	810	570	765	785	790
CC	835	670	765	805	810
master	46220	42730	44760*	44800	44810*

The problems marked with a * are the problems where the solving process was stopped after a certain time because CPLEX was not able to solve the problem to full optimality. The maximum error of the solution returned by CPLEX after being stopped with regards to the optimum was somewhere in the range between 0.01% and 0.04% for our problems.

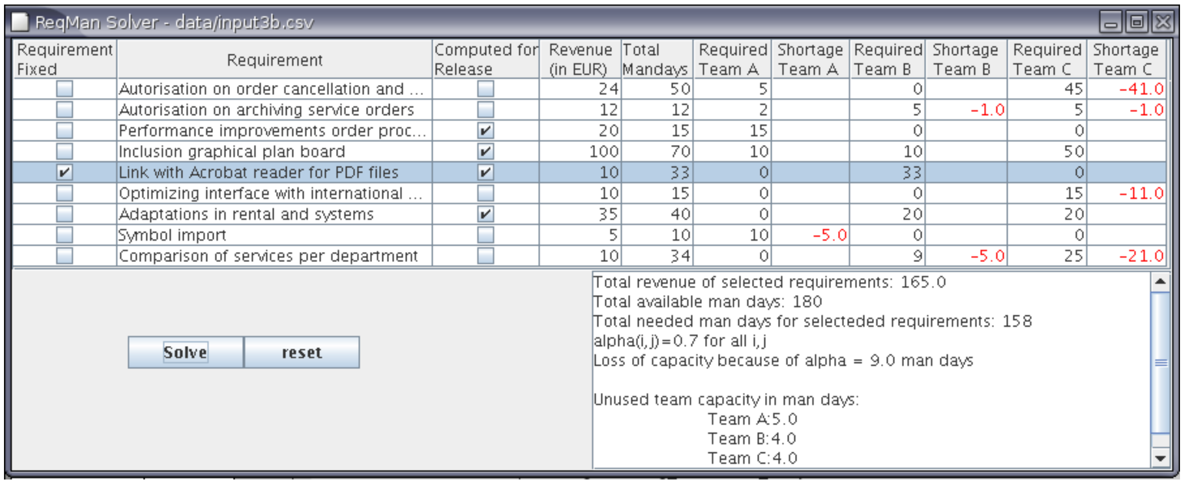

ReqMan Solver - data/input3b.csv

Requirement Fixed	Requirement	Computed for Release	Revenue (in EUR)	Total Mandays	Required Team A	Shortage Team A	Required Team B	Shortage Team B	Required Team C	Shortage Team C
☐	Autorisation on order cancellation and ...	☐	24	50	5		0		45	
☐	Autorisation on archiving service orders	☐	12	12	2		5		5	
☐	Performance improvements order proc...	☐	20	15	15		0		0	
☐	Inclusion graphical plan board	☐	100	70	10		10		50	
✓	Link with Acrobat reader for PDF files	☐	10	33	0		33		0	
☐	Optimizing interface with international ...	☐	10	15	0		0		15	
☐	Adaptations in rental and systems	☐	35	40	0		20		20	
☐	Symbol import	☐	5	10	10		0		0	
☐	Comparison of services per department	☐	10	34	0		9		25	

Figure 1: Screenshot of prototype before solving

In Figure 1 and Figure 2 screenshots of the prototype based on CPLEX are given. The data set that was used for these screenshots is the example that was introduced in Table 1. Figure 1 shows the program just after it has started while the user has fixed the fifth requirement which means that this requirement must be selected in the solution. The only thing that can be changed by the user is the first column, by which a user can select requirements that must be in the solution. Note that, although not indicated in Figure 1, also the team transfer steering mechanism is enabled.

Figure 2 shows the program after the situation of the first figure is solved. The program shows the requirements that are selected in the solution by showing a check in the third column and in this figure it can be seen that the fixed fifth requirement is indeed selected



Requirement Fixed	Requirement	Computed for Release	Revenue (in EUR)	Total Mandays	Required Team A	Shortage Team A	Required Team B	Shortage Team B	Required Team C	Shortage Team C
☐	Autorisation on order cancellation and ...	☐	24	50	5		0		45	-41.0
☐	Autorisation on archiving service orders	☐	12	12	2		5	-1.0	5	-1.0
☐	Performance improvements order proc...	✓	20	15	15		0		0	
☐	Inclusion graphical plan board	✓	100	70	10		10		50	
✓	Link with Acrobat reader for PDF files	✓	10	33	0		33		0	
☐	Optimizing interface with international ...	☐	10	15	0		0		15	-11.0
☐	Adaptations in rental and systems	✓	35	40	0		20		20	
☐	Symbol import	☐	5	10	10	-5.0	0		0	
☐	Comparison of services per department	☐	10	34	0		9	-5.0	25	-21.0

```
Total revenue of selected requirements: 165.0
Total available man days: 180
Total needed man days for selecteded requirements: 158
alpha(i,j)=0.7 for all i,j
Loss of capacity because of alpha = 9.0 man days

Unused team capacity in man days:
              Team A:5.0
              Team B:4.0
              Team C:4.0
```

Figure 2: Screenshot of prototype after solving

in the solution. The fourth and fifth column show what the revenue of each requirement are and how many total man days are required to implement each requirement. The remainder of the columns show for each of the requirements the total amount of work per team that is needed to implement it, and how much additional work would still be needed besides the already available capacity from each of the teams to still implement this requirement in case a requirement is not selected in the current solution.

In the text-area the user is presented with additional information about the total revenue, the total amount of available man days, the total needed available man days and the capacity per team that is still left. Furthermore information is given about the $\alpha_{i,k}$ parameters and how much capacity is lost by means of transferring people between the teams.

4 Conclusion and Future Research

In this paper, we have presented a mathematical formalization of flexible release composition, using integer linear programming models and methods. We defined unique aspects and managerial steering mechanisms as input for our tool, modelling one pool of developers as well as different development teams, allowing team transfers, considering dependent requirements, hiring external capacity, and extending the deadline. A large body of knowledge available for formalizing and solving ILP models can now be used to reason about release composition, and offers possibility to extend even further with additional aspects or mechanisms.

We implemented a subset of the managerial steering mechanisms in a prototype tool, which shows that with given constraints it is possible to select a practically optimal subset of requirements with maximized revenue. In addition, alternative selections can be identified by fixing certain requirements beforehand. Our prototype also gives information on the used capacity and on the additional effort required to implement requirements which are not yet selected. With the results of experiments on real-life data we are confident that our tool is of practical value for product managers and development project managers. There still are some limitations to the chosen approach, as we assume that all team members start at the same date, and that all developers have the same productivity. Furthermore, the combination of steering mechanisms need to be worked out both in the formalization, as well as in the tool. The companies involved in the experimentation indicated that our approach simplified the task of computing the set of release requirements to a great extend. They also expressed the wish for support during release development as in the dynamics of the project workload turns out to be overestimated or underestimated, and that the revenue projections are changing due to a changing market.

In our vision a product manager composing the set of requirements for the next release needs a dashboard that serves with all kinds of buttons and handles to evaluate the consequences of managerial options before making a decision. We therefore intend to extend our prototype with possibilities to analyze and visualize alternative selections including the effect of hiring external capacity and the extension of the deadline.

Moreover, we intend to consider more detailed models for requirements selection which results in more flexibility for the product manager. One extension is to make a planning for each person as part of a planning of complete teams. Another important extension is to schedule activities explicitly in time. This allows for example to take planning periods for different teams and persons into account, such as holiday season or temporal unavailability due to other projects. It also allows the inclusion of dependencies in the work, e.g. requirement R_i has to be implemented before R_k, thus providing insight in the critical path of a release project plan.

References

[1] van den Akker, J.M., C.P.M. van Hoesel, and M.W.P. Savelsbergh (1999). A polyhedral approach to single-machine scheduling problems. *Mathematical Programming 85*, 541-572.

[2] van den Akker, J.M., J.A. Hoogeveen, and S.L. van de Velde (1999). Parallel machine scheduling by column generation. *Operations Research*, Vol. 47, No. 6, 862-872.

[3] van den Akker, J.M. and J.A. Hoogeveen (2004A). Minizing the number of tardy jobs. In J. Y.-T Leung (ed.), *Handbook of Scheduling: Algorithms, Models and Performance Analysis,* pp. 227-243, CRC Press, Inc. Boca Raton, Fl, USA.

[4] van den Akker, J.M. and J.A. Hoogeveen (2004B). *Minimizing the number of late jobs in case of stochastic processing times with minimum success probabilities.* Submitted to INFORMS Journal on Computing.

[5] Berander, P. and Andrews, A. (2005), Requirements Prioritization. In: *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin (eds.), Berlin, Germany, Springer Verlag (Forthcoming).

[6] ILOG CPLEX, http://www.ilog.com/products/cplex.

[7] Crescenzi P. and V. Kann, eds. *A compendium of NP optimization problem.* http://www.nada.kth.se/ viggo/wwwcompendium/wwwcompendium.html

[8] Firesmith, D. (2004), Prioritizing Requirements, *Journal of Object Technology*, vol 3, no 8, September-October 2004, pp 35-47.

[9] Höst, M., Regnell, B., Natt och Dag, J., Nedstam, J., Nyberg, C. (2001), Exploring Bottlenecks in Market-Driven Requirements Management Processes with Discrete Event Simulation, *Journal of Systems and Software*, vol 59, pp 323-332.

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

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

[12] Leffingwell, D., and Widrig, D. (2000), *Managing Software Requirements - A Unified Approach*, Addison-Wesley, Upper Saddle River, NJ.

[13] S. Martello and P. Toth. (1990) *Knapsack Problems: Algorithms and Computer Implementations* Wiley-Interscience Series In Discrete Mathematics and Optimization.

[14] Natt och Dag, J., Gervasi, V., Brinkkemper, S. and Regnell, B. (2005), A Linguistic Engineering Approach to Large-Scale Requirements Management, *IEEE Software, Special Issue on Requirements Engineering*, vol 22, no 1, January/February 2005 (Forthcoming).

[15] Johan Natt och Dag, Vincenzo Gervasi, Sjaak Brinkkemper, and Björn Regnell (2004), Speeding up Requirements Management in a Product Software Company: Linking Customer Wishes to Product Requirements through Linguistic Engineering. *In: Proceedings of the 12th International Requirements Engineering Conference*, N.A.M. Maiden (Ed.), IEEE Computer Science Press, pp 283-294, September 2004.

[16] Potts, C. (1995), Invented Requirements and Imagined Customers: Requirements Engineering for Off-the-Shelf Software, *Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE'95)*, pp. 128-30.

[17] Regnell, B., Karlsson, L. and Höst, M. (2003), An Analytical Model for Requirements Selection Quality Evaluation Evaluation in Product Software Development, *Proceedings of the 11th International Requirements Engineering Conference*, IEEE Computer Science Press, pp 254-263.

[18] 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 (Forthcoming).

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

[20] Ruhe, G., Eberlein, A., Pfahl, D. (2003), Trade-off Analysis for Requirements Selection, *International Journal of Software Engineering and Knowledge Engineering,* vol 13, no 4, pp 345-366.

[21] Saaty, T.L. (1980), *The Analytic Hierarchy Process*, McGraw-Hill, New York, NY.

[22] Wolsey L.A. (1998) *Integer programming* Wiley-Interscience Series In Discrete Mathematics and Optimization.