Software product release planning through optimization and what-if analysis

Marjan van den Akker
Sjaak Brinkkemper
Guido Diepen
Johan Versendaal

Department of Information and Computing Sciences, Utrecht University

Technical Report UU-CS-2006-63

www.cs.uu.nl

ISSN: 0924-3275

Software product release planning through optimization and what-if analysis

Marjan van den Akker [1] Sjaak Brinkkemper Guido Diepen [1]
Johan Versendaal

Department for Information and Computing Sciences,
Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands

Abstract

We present a mathematical formalization of release planning with a corresponding optimization tool that supports product and 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 maximal projected revenue against available resources. The input for the optimization is twofold. The first type of input data concerns the list of candidate requirements, estimated revenues, and resources needed. Secondly, managerial steering mechanisms enable what-if analysis in the optimization environment. Experiments based on real-life data made a sound case for the applicability of our approach.

Key words: Integer Linear Programming, Release Planning, Requirements Management, Optimization

1 Requirements selection for release planning

Determining requirements for upcoming releases is a complex process (see e.g. Potts (1995); Regnell and Brinkkemper (2005)). Often there are more requirements than can be implemented, the capacity of available resources is limited, requirements are unclear, important customers of the software products may put more pressure to include certain requirements, etc. Moreover,

Email addresses: j.m.vandenakker@cs.uu.nl (Marjan van den Akker), s.brinkkemper@cs.uu.nl (Sjaak Brinkkemper), g.diepen@cs.uu.nl (Guido Diepen), j.versendaal@cs.uu.nl (Johan Versendaal).

[1] Supported by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society)

requirements need to be collected from different sources. Typically, a product manager obtains functional, technical and usability requirement through the sales department, service and maintenance departments, ad hoc or organized customer contacts, user groups, the development department, and more (cf. Natt och Dag et al. (2005)).

In van den Akker et al. (2005) we discuss the basics of a solution for release planning using Integer Linear Programming techniques. As described many aspects influence the definition of a set of requirements for a next release. Several 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, Eberlein, and Pfahl (2003); Natt och Dag et al. (2004); Natt och Dag et al. (2005)).

In order to deal with the multi-aspect optimization problem in literature different techniques and procedures have been applied. The analytical hierarchy process (AHP; 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 the application of the analytical hierarchy process to large amounts of requirements using linear programming techniques. Ruhe and Saliu (2005) describe a method for release planning including stakeholders opinions on requirement importance, and use linear programming techniques. Carlshamre (2002) too used linear programming techniques on which a release planning tool was built and they added requirement interdependencies as an important aspect in release planning. 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, Eberlein, and Pfahl (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 the approach of Jung (1998) and Carlshamre (2002), our technique is based on the assumption that a release's best set of requirements is the set that results in maximum projected revenue against available resources in a given time period. We take into account dif-

Requirement	Revenues	Total	Team A	Team B	Team C
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

ferent aspects, including the total list of requirements, possible dependency between requirements, a requirement's projected revenue including additional revenues for complete realization of requirement packages (bundles), and a requirement's resource claim per development team. To further increase its practical applicability, we enhance our technique with managerial steering mechanisms, i.e. enabling of team transfers, conceding release deadline extension, and allowing extra resources. Both the listed aspects, and the list of managerial steering mechanisms are not complete. Others can be identified. The mentioned aspects and steering mechanisms however are recognizable from a product management (e.g. Nelson (2005)) and project management (e.g. CCTA (2001)) perspective. By introducing the aspects and managerial steering mechanisms into integer linear programming models for the release planning process we extend the work of Jung (1998) and Carlshamre (2002).

Table 1 depicts a simplified example 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 needs to be defined. Suppose for instance that the total amount of available man days in the three teams is 60, then we note 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 B or C, and/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.

As described above, several authors have already discussed and presented as-

pects, 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 and requirement dependencies. Secondly, we use unique yet practical managerial steering mechanisms that can aid product managers and project managers in release planning, notably enabling of team transfers, deadlines and extra resources. In the third place we show how to define and process aspects and managerial steering mechanisms using integer linear programming. The outcomes of our optimization tool are values for decision variables that maximize the estimated revenue 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 and in our domain. For the reader's convenience, we include a discussion on solving integer linear programming problems in Appendix A. In Section 3 we describe the software tool, called ReqMan, through which we demonstrate the practical applicability of our optimization approach. In closing, we conclude and provide areas for future research in Section 4.

2 Formalization of release planning

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 being assigned a number of tasks each with a specific duration and deadline. It receives 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,

an (near-)optimal solution of the integer linear program can often be found 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 release 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.

We present different models taking into account different aspects and managerial steering mechanisms within a product software company. We consider:

- one pool of developers (i.e. no different development teams)
- different teams without team transfers, each with its own capacity constraint
- different teams with team transfers allowed
- hiring external team capacity
- extension of the development project deadline
- requirement dependency (functional, revenue and cost related)

In the following sub-sections we discuss the above aspects and managerial steering mechanisms. For presentational reasons, each of these issues will be 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.

2.1 Development by 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. Such estimates could come from project managers (top-down) as well as developers (bottom-up). 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 Martello 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.

We realize that the determination of values for revenues of requirements can be hard. Various factors may influence these values. A revenue value can be tangible (like estimated return on investment) or even intangible (like user satisfaction; see for example Murphy and Simon (2001), regarding a discussion on intangible benefits for ERP implementation). We recognize two main categories of revenue value calculation: absolute value determination, and relative value determination. In absolute revenue value calculation e.g. the product manager of a company developing standard software products could determine an estimated value for the absolute revenue. In the former position of product manager, one of the authors of this paper wrote a business case for triggering the development of a new product software release, taking into account estimated sales of the new release, and drilling down to determine estimated values for each of the new to build requirements. Note that intangible values (like user satisfaction) were not made explicit. As for relative revenue value calculation, Karlsson and Ryan (1997) provide a useful approach: through AHP a relative value for the revenue per candidate requirement can be determined. Using AHP, each requirement can be assigned a value between for example 0-100. Ruhe and Saliu (2005) use stakeholders' opinions to determine the rel-

ative importance of requirements. In further finetuning they assign different levels of importance to stakeholders, and consequently their opinions of values for importance. The overall relative revenue for a particular requirement is subsequently calculated taking into account the assigned importance values (by the stakeholders) and the relative importance of each of the stakeholders individually.

Like with determining the revenue values, also the determination of values for costs can be difficult to calculate. At the software vendor two of the authors worked for in the past, the process was as follows. The product manager inserted the requirement in the requirements database, and determined the software modules for which the requirement had consequences (which modules needed to be updated for this requirement). The product architect(s) determined the workload per module using input from product/module consultants and software engineers. In many cases, to make the resource need more clear, conceptual solutions were written, detailing the requirement in a business context, and showing relations to other modules. In general top down resource calculation and bottom up resource calculation may provide a useful estimation for the costs in many software companies.

2.2 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 Team transfers

By allowing people to work in a different team, there is more flexibility which can increase revenue. We 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

$$\sum_{j=1}^{n} a_{ij}x_j \leq U_{cap}[y_{ii} + \sum_{k:k\neq i} \alpha_{ki}y_{ki}] \qquad \text{for }, i = 1, \ldots, m, \qquad (3)$$

$$\sum_{k=1}^{m} y_{ik} = m_i, \qquad \text{for } i = 1, \ldots, m, \qquad (4)$$

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

$$y_{ik} \text{ nonnegative and integral}, \qquad \text{for } j = 1, \ldots, n.$$

Inequality (3) 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 (4) 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, transfers 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 will not occur very often. However, we can prevent this by an extension of the model which includes binary variables indicating whether a team is extended by transfers or sends transfers to others. This extension is omitted for reasons of brevity.

2.4 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 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 in the number of man days. 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

$$\sum_{j=1}^{n} a_{ij} x_j \leq d(T) Q_i + \alpha_{ei} u_i, \qquad \text{for } i = 1, \ldots, m, \tag{5}$$

$$\sum_{i=1}^{m} q_i u_i \leq E \tag{6}$$

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

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

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 $\sum_{j=1}^{n} v_j x_j - C\delta_T$ and the right-hand side of inequality (5) is changed into $(d(T) + \delta_T)Q_i$.

2.5 Requirement dependencies

Until now we assumed that all requirements can be implemented independently. One of the few papers that addresses mutual dependency of requirements in release planning is from Carlshamre (2002). Based on the referred paper we identified five types of requirement dependencies: 1) Implication, 2) Combination, 3) Exclusion, 4) Revenue-based, and 5) Cost-based. The first three deal with functional dependencies: requirement R_j can only be implemented if requirement R_k is also implemented, requirement R_j only makes sense together with requirement R_k, and requirement R_j cannot be implemented in combination with requirement R_k, respectively. Revenue-based dependencies occur when combinations of requirements increase or decrease the revenue value. Finally, cost-based dependencies apply in case combinations of requirements increase or decrease the needed amount of resources. We will model the five types subsequently.

1) Implication
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

$$x_j \leq x_k. \tag{7}$$

2) Combination
If two requirements R_j and R_k are dependent in the sense that they cannot be implemented separately from each other, this can be dealt with by considering them as one requirement. However, if the company wants to treat the requirements separately we include this type of dependency by extending our model with the equation

$$x_j = x_k. \tag{8}$$

3) Exclusion
This type of dependency excludes a combination of requirements. If R_j and R_k cannot both be selected this can be modelled by the inequality

$$x_j + x_k \leq 1. \tag{9}$$

4) Revenue-based
A typical example is when two requirements together provide a 'package' in some area within the software product such as procurement within an ERP package. For instance: the ability to purchase order non-production goods (requirement 1), and a workflow approval engine for purchase ordering (requirement 2); the first has especially value when there is workflow supporting the non-production goods ordering.

Let R_j and R_k be a pair of complementing requirements. Since it is possible the R_j influences the revenue of R_k and vice versa, we choose to model these effects by one number w_{jk} denoting the additional revenue if both requirements are selected. We can include this type of revenue in the model by using a binary variable x_{jk}, which equals 1 if the R_j and R_k are both selected and 0 otherwise. The revenue that we want to maximize is then equal to

$$\sum_{j=1}^{n} v_j x_j + \sum_{(j,k)\in C} w_{jk} x_{jk},$$

where C is the set of pairs of complementing requirements. To ensure that the variable x_{jk} will only be equal to 1 if both the variables x_j and x_k are equal to 1 we have to add the following constraint:

$$2x_{jk} \leq x_j + x_k. \tag{10}$$

The above can be generalized to the situation with larger sets of requirements that form a package, in case the additional revenue is only obtained if the

complete package is selected. If part of the additional revenue is already obtained when a subset of the package is implemented, the model becomes much more complex a.o. because we would have to define variables for all subsets. If the packages are not disjoint, the additional revenue might be reduced if two 'overlapping' packages are selected, which again complicates the model. We consider this is outside the scope of the paper and assume that the set C defined above consists if disjoint pairs.

The revenue effect of combinations of requirements can also be negative. This is the case when two requirements have some overlap such as for example a paper user manual and an on-line help functionality. Let R_j and R_k be a pair of overlapping requirements. We again use the number w_{jk}, but it now has a negative value. Similar to the previous case, x_{jk} indicates if R_j and R_k are both selected and the term

$$w_{jk}x_{jk}$$

has to be added to the revenue function. Observe that in case of additional revenues the revenue function drives x_{jk} to 1 if possible. Therefore, we posed an upper bound on x_{jk} to make sure that it only gets equal to 1 when R_j and R_k are both selected. When we are dealing with additional cost, the revenue function drives x_{jk} to 0 if possible. Now we use a lower bound on x_{jk} to force it to be 1 when R_j and R_k are selected. We add the constraint:

$$x_{jk} \geq x_j + x_k - 1. \tag{11}$$

5) Cost-based
The last type of dependency concerns the fact the requirements can influence each others cost. For example, if all the output has to fit a 800*600-pixels screen, this may have a strong impact on a number of requirements. We model this by defining the number $a_{i,j\to k}$ as the amount of additional work in team G_i on requirement R_k and induced by requirement R_j. Unlike the previous case, we explicitly model which requirement influences the workload of which other requirement, since this information is very useful for the planning of the work in the teams. We again define a binary variable x_{jk} indicating if R_j and R_k are both selected. We now can include the additional work by changing constraint (2) into

$$\sum_{j=1}^{n} a_{ij}x_j + \sum_{(j,k)\in A} a_{i,j\to k}x_{jk} \leq d(T)Q_i, \quad \text{for } i = 1,\ldots,m, \tag{12}$$

where A is the set of pairs of requirements (j,k) such that R_j influences the amount of work to implement R_k. To make sure that x_{jk} gets equals to 1 if R_j and R_k are selected we have to add constraint (11).

Clearly, implementing some requirement R_j may also make the implementation of R_k less time-consuming. This can also be modelled by the numbers

$a_{i,j\to k}$ but these numbers now take a negative value. However, since setting x_{jk} to 1 is now favorable for the capacity constraints, we now have to use constraint (10) to make x_{jk} behave correctly. Suppose that two different requirements R_1 and R_2 both increase the amount of work of R_3. If in the implementation of R_3 we have to deal with issues induced by R_1 and by R_2, the additional work may be less than the sum of the amounts of work induced by R_1 and R_2 separately. Since such combination effects strongly complicate the model, we assume that the pairs in A are disjoint. The only possible exception is that if $(j, k) \in A$ then (k, j) may also be in A.

3 Experimentation

To obtain a proof-of-concept we implemented a prototype of a requirements selection system. Besides the computation of a requirement selection for a given optimization model, the prototype allows the user to generate alternative selections by fixing certain requirements beforehand. 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) Solver will notify that it is not able to solve the problem because the number of variables is just too big. To be able to also solve the larger problems a Java program was implemented. This prototype has a graphical interface and makes use of the callable library of ILOG CPLEX (see Cplex (2005)) for solving the ILP problems. CPLEX is one of the best known packages for integer linear programming.

3.1 One pool, different teams, and team transfers

For testing the program different types of data sets were used. The different types were:

- **small**: 9 requirements and 3 development teams.
- **AA, BB, and CC**: 24 requirements and 17 teams. Ratio of available and total required capacity approximately 50, 60 and 70 percent, respectively.
- **master**: 99 requirements and 17 teams.

All of the used data sets are available online [2] for research purposes.

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

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*

Table 2
Solution values for the three types of data sets

The Small data set consists of fictitious data already provided in Table 1. The AA, BB, CC and Master data sets were generated from larger real life data sets. All team values were kept the same, but team capacities and 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.

While no transfers between the teams were allowed all of the above problems (associated with the three types of data sets) were still solvable by the Solver in Microsoft Excel. As soon as team transfers were allowed, the number of variables in each of the problems, except for the small problem, became too big for this solver and we had to use CPLEX to solve these problems.

For the largest problem even CPLEX has some difficulties, because of the large number of variables it takes CPLEX too much time to solve the problem to full optimality. However, 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.

Table 2 presents the solution values, i.e., revenues for the computed requirements selection, for the different problems that were tested. The second column corresponds to the model introduced in Section 2.1 where there is just one big pool of resources. The third 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 2.3 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.

The problems marked with a * are the problems where the solving process was stopped after a certain time because CPLEX needed too much time 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, which indicates that

Data	Ratio rule	ILP optimum pool
small	182	182
AA	655	700
BB	805	810
CC	835	835
master	45290	46220

Table 3
Comparison between the ratio rule and optimal solution

the solutions are near-optimal.

Without mathematical optimization the product manager might use an intuitive rule for selecting requirements. A well-known approach for the problem with one pool of developers is to select the requirements in decreasing order of the ratio of the revenue v_j and the workload a_j. To get an idea of the difference between solutions obtained by such an intuitive approach and optimizing by integer linear programming, we compared the total revenue from both approaches for the different data sets. Table 3 shows that especially for larger data sets, the possible gain can be significant. If more teams are involved and possibly some steering mechanisms are included it is harder to have a good overview of the problem data, and hence intuitive optimization by hand becomes very difficult.

3.2 Perturbations and what-if analysis

To test the stability of a solution found by the ILP-solver we added random perturbation to the revenue input of some of the requirements. We choose to add the perturbation to the revenue since this is the component that is often the most difficult to determine. From the original *master*-problem with $U_{\text{cap}} = 10$ we created 1000 instances each having on average 10% of the requirements with perturbations of maximally $+/-10$, $+/-20$, and $+/-30$. When we look at the solutions found for these perturbated instances we see that only in 5 cases the set of selected requirements differ from the original set of selected requirements. When looking at the difference, this is very little: the majority of the selected requirements in the original solution is also selected in the solution for the perturbated problem.

Furthermore, we also looked at the effect of extending the deadline or hiring external resources to the total revenue of the selected requirements for the *master*-problem with $U_{\text{cap}} = 10$. The pattern we see when extending the

deadline is that at first the extension only increases the cost since it does not provide enough capacity to implement a better set of requirements. Only after the deadline is extended sufficiently, enough of capacity is available to implement a better set of requirements, and thus the objective value will increase. After that, extending the deadline even further will also first decrease the objective until there is enough of capacity available to implement an even better set of requirements.

In this way we obtain so-called jump points which are in fact the only interesting options. In Figure 1 we plotted the the total revenue against the

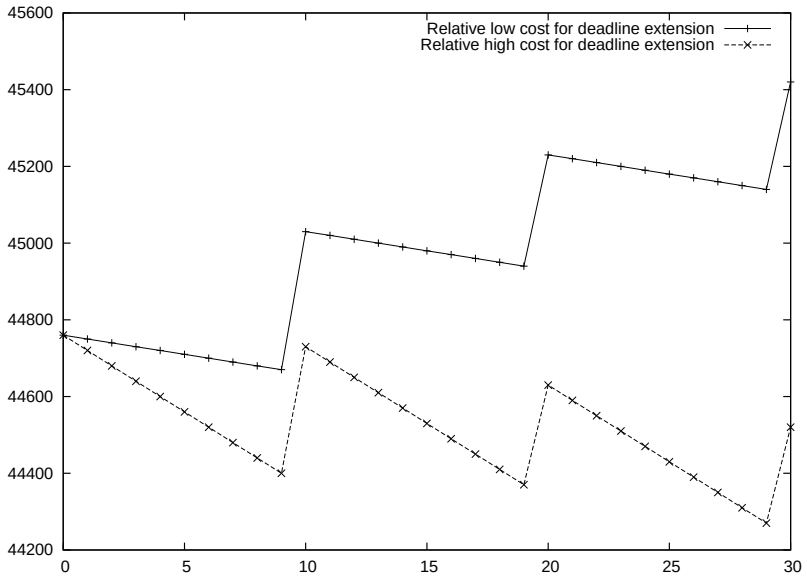


Fig. 1. Total revenue plotted against the number of days deadline is extended

number of days the deadline is extended for two different extension costs. The upper line shows that if the cost of extending the deadline is relatively low, the consecutive jump points have increasing revenues. One may expect that after some time the jump points will start to decrease and will not provide any interesting options anymore. The lower line shows that if the cost of extending the deadline are relatively high, the jump points will be decreasing from the beginning, which indicates that a deadline extension is not interesting. This implies that there is a certain threshold (maximum extensions costs) for applying deadline extension in order to be reasonable.

We also tested the effect of adding the possibility of hiring external resources at various costs per unit of hiring. Since in this situation extra resources are only hired if they are really used for implementing requirements we expect the results to differ from the results of extending the deadline because no extra resources will be hired as long as the gained capacity does not allow for selecting another requirement. This means that in theory the objective function is non-decreasing. In practice however, the resulting ILP's turned

out to be quite difficult to solve to optimality, which means that we had to deal with gaps up to 0.23% from the optimal solutions. This explains the small decreases found in Figure 2. As we may expect, Figure 2 indicates that the increase in revenue becomes smaller as more budget for external resources is available and eventually becomes 0.

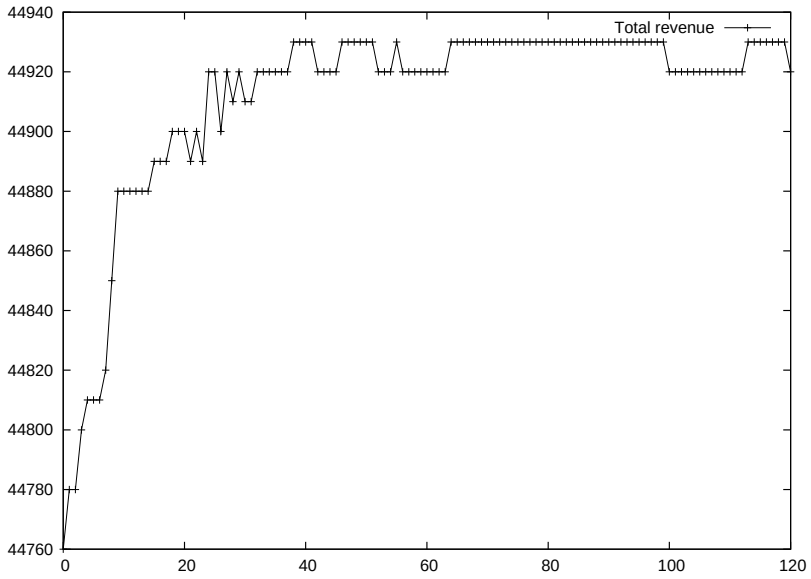
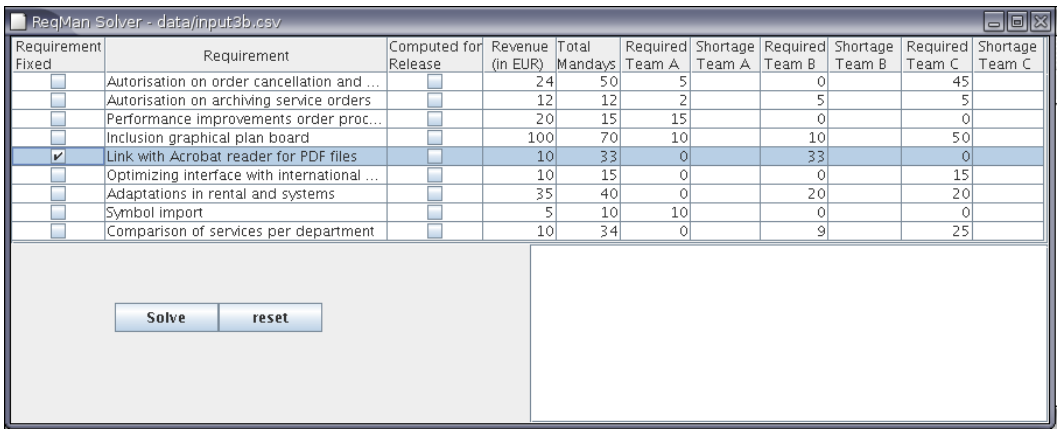


Fig. 2. Total revenue plotted against budget for external resources

3.3 Prototype tool

Screenshots of the Java-prototype based on CPLEX showing a specific run (based on a 'small' type of data set) can be seen in Figure 3 and Figure 4.



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	

Fig. 3. Screenshot of prototype before solving

Figure 3 shows the program just after it has started while the user has fixed the fifth requirement which means that this requirement must be selected

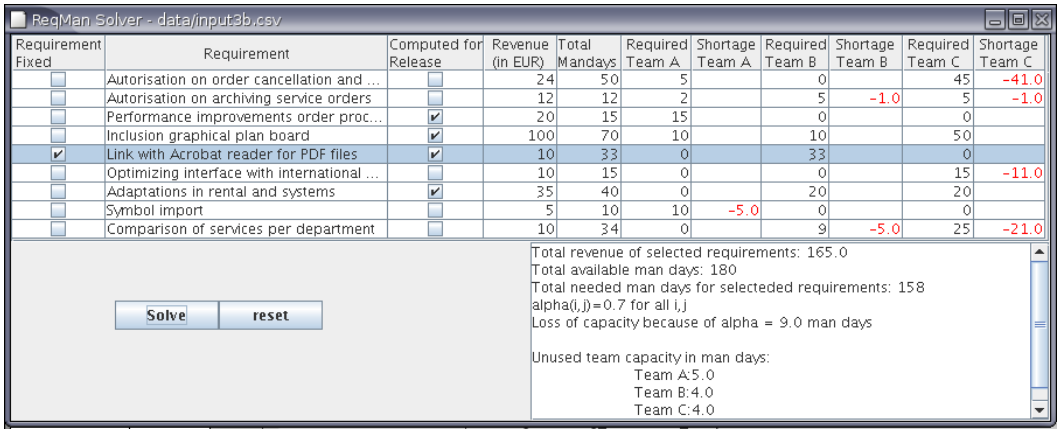


Fig. 4. Screenshot of prototype after solving

in the solution. The user can change the first column to select requirements that must be in the solution and in this way generate alternative solutions. The other columns are either data or results of the computation. Note that, although not indicated in Figure 3, also the team transfer steering mechanism is enabled.

Figure 4 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 in the solution. The fourth and fifth column show what the revenue of each requirement is 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 (unused) 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 α_{ik} parameters and how much capacity is lost because of transferring people between the teams.

4 Conclusion and Future Research

In this paper, we have presented a mathematical formalization of flexible release planning, using integer linear programming models and methods. We defined unique aspects and managerial steering mechanisms as input for our framework, modelling one pool of developers as well as different development

teams, allowing team transfers, considering dependent requirements including additional revenues for packages of 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 planning, and offers possibility to extend even further with additional aspects or mechanisms. 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. However, obviously more experimentation is needed.

We found that, despite (controlled and limited) revenue estimation perturbations, for large sets of requirements the resulting output does not differ too much. This encourages us that the model is useful even when (revenue) input is not fully correct. More research needs to further confirm this.

We assume that all team members start at the same date, and that all developers have the same productivity. A next step is to extend the model with issues regarding different team members, by adapting the team capacities in the right way. Furthermore, the combination of steering mechanisms needs to be worked out further and tested in the tool. 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.

Note that our approach supports the release planning for a fixed given time period. In practice the revenue value of requirements may evolve over time, as the release is being developed in a changing market; during release development the dynamics of the project workload turns out to be either overestimated or underestimated. When activities are scheduled in time replanning can be used to deal with these changes. This is a further extension of our current approach.

We further plan to test the validity of our model through multiple business cases. In such cases an appropriate approach is to use our model for release planning on the one hand, and compare its outcome with the company's proprietary way of release planning.

Acknowledgements

The authors wish to thank the organizers of the workshop REFSQ'05 and anonymous reviewers for their valuable comments on an earlier version of this paper.

References

van den Akker, J.M., Brinkkemper S., Diepen G. and Versendaal, J. (2005). Determination of the Next Release of a Software Product: an Approach using Integer Linear Programming. *Proceeding of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ'05*, eds. Erik Kamsties, Vincenzo Gervasi, and Pete Sawyer, Band 10, pp 247-262.

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.

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.

van den Akker, J.M. and J.A. Hoogeveen (2004A). Minimizing 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.

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.* To appear in Journal of Scheduling.

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).

Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Natt och Dag, J., (2001), An Industrial Survey of Requirements Interdependencies in Software Product Release Planning, Proceedings of the Fifth International Symposium on Requirements Engineering (RE'01).

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

Central Computer and Telecommunication Agency (CCTA) (2001), *Managing Successful Projects with PRINCE2* , Eleventh impression.

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

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

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

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.

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

Karlsson, J. and Ryan, K. (1997), A Cost-Value Approach for Prioritizing

Requirements, *IEEE Software*, September/October 1997 pp 67-74.

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

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

Murphy, K.E., and Simon, S.J., (2001), Using cost benefit analysis for enterprise resource planning project evaluation: A case for including intangibles, Proceedings of the 34th Hawaii International Conference on System Sciences.

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, pp 32-39, January/February 2005.

Natt och Dag, J., Gervasi, V., Brinkkemper, S. and Regnell, B. (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.

Nelson, B. (2005), Who Needs Product Management? *www.productmarketing.com*, vol 3, no 2, pp 12-15, March/April 2005.

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.

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.

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.

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.

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

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.

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

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

A Solving integer linear programming problems

For the sake of completeness, we elaborate in this appendix 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 near-optimal solution may be available. The described method is used in most integer linear programming (ILP) software packages. For more background information, the interested reader is referred to Wolsey (1998).

All the problems formulated in this paper 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 a 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 or more 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 in the node 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.

We are aware that with the application of ILP in the domain of release planning one of the key issues in finding a best set of requirements for development is the determination of revenues and costs.