An Integer Linear Programming Approach to Product Software Release Planning & Scheduling

C. Li
J.M. van den Akker
S. Brinkkemper

Technical Report UU-CS-2006-065
August 2006
Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

An Integer Linear Programming Approach to Product Software Release Planning & Scheduling

Chen Li

Master Thesis

Master program of Business Informatics

Department of information & computing science

Utrecht University

August 10, 2006

Index

0. Introduction

0.1 Problem statement

The development processes of product software can be categorized into four phases: Requirement management→ Architecture/Design development → Deliver → Implementation Services[1]. In requirement management phase, the main activities are to generate requirements and to select requirements. Release planning is one of those activities- which refer to the process of selecting the right requirements for the coming release. This thesis is built around this topic of release planning.

"Release planning—the definition of upcoming releases in a product roadmap—fulfils a strategic role. Making incorrect choices for a release definition may significantly impact the competitiveness of software intensive companies in a market driven environment"[2].

It is always a challenge for software companies to determine the upcoming release due to the fact that the wish list of requirements gathered from different parties is so big that exceeds the capability of the company. Other constraints like time to market, cost, etc also restrict the scope of the coming release into a limited range. So how to select or prioritize the requirements becomes very important and can even play a strategic role.

Several techniques have been published for requirements selection and prioritization. Firesmith (2004) has presented a list of dimensions, and the priority of a requirement is determined by the average value of each dimension [3].Leffingwell and Widrig (2000) [4] has designed a voting mechanism to determine the average weight of each requirement through the voting of different stakeholders. Suzanne Robertson and James Robertson used a method called 'quality gateway' to determine on each requirement's go or not go [5]. Marjan van den Akker, et al (2004)[7] used a very intriguing selection method using Integer linear programming (ILP) however, there are several open questions left to answer. This thesis will address some of those questions.

Firstly, most of the releases planning methods try to balance the trade of between values and cost, for example, obtaining the maximal revenue with limited amount resources. However, whether these two factors—cost and value—are sufficient enough to determine a good release plan remains uncertain. This provides us the first opportunity to find out which factors should be considered in making a release planning.

Secondly, Marjan van den Akker et al (2004) have provided a knapsack model for requirement selection and some management steering mechanism for making a release plan, however, more functions are demanded. These functions include setting dependencies, modeling personal differences scheduling requirements, etc. These demands require us to enrich and extend the ILP model so as to include more functions for making a release plan.

At last, prototype tools for the new functions should also be implemented. Besides the technical issues, like the design, the implementation and integration, using these tools, we can not only test the mathematical models for the additional functions, but also check for how much the release planning factors can influence the final result. We can also search for whether there are opportunities for further process improvement with the help of the tools.

0.2 Research question

To address the problems mentioned above, my main research question will be:

*How to define a **profitable** and **practical** release definition which can fulfill the **different interests** of stakeholders in the release planning context?*

In this research question, three issues are specially emphasized:

- First, the key point is still to maximize the anticipated revenue of the requirements composition, that's why "*profitable*" is addressed.
- Second, *"practical"* means the release definition should not conflict to the external and internal constraints. For example, the resource and time are limited; the requirements are interdependent, etc.
- Third, *"fulfill different interests"* means it should provide more functions or management steering mechanism to fulfill the wishes from different stakeholders.

This research question can be divided into several sub-questions categorized into three scientific fields—Information science, Algorithms & Computational model, as well as Computer science.

0.2.1. Information science:

- *What are the factors for release planning?*
- *What are the key activities and processes to make a release plan?*

In this field, the research goal is to find which factors should be included in release planning, and what are he processes and key activities for release planning. The factors will be included in the later chapters as input parameters, and the key processes will guide the modeling and be integrated with other process supporting models.

0.2.2. Algorithms & Computational model:

Here, the basic assumption is the number of requirements is restricted within 200, and ILP model is capable enough to solve the problem [6] [8]. Based on this assumption, the research questions are:

- *How to model the functional extensions of the current linear programming model i.e. what-if analysis?*
- *How to model each type of requirement interdependency using ILP?*
- *How to schedule the requirements development exactly in time?*
- *How to integrate the new models with the original knapsack model?*

After we determined the factors for release planning and the key activities, we will focus on the mathematical modeling. As stated above, we will still use ILP for the modeling. We will try to include more management steering mechanisms in the model and solve at least the requirement dependency issue and the requirement scheduling issue. Needless to say, these new models should be compatible with the original knapsack model and it extensions.

0.2.3. Prototype design:

- *How to implement the computational mode?*
- *How to integrate it with the prototype of the knapsack model?*
- *How to adapt the tool with J2EE environment?*
- *How much can the release planning factors influence the result?*
- *Is there any possibility for process improvement?*

After the computational models are determined, the following issue is how to implement them. The new prototype should not only be capable of solving new problems, but also compatible with the prototype of the knapsack model. An additional constraint is the new prototype should work in J2EE (Java 2 Enterprise Edition) Environment.

After we implement the prototype, we will try to find out how much the release planning factors can influence the result, and also search for the opportunities for process improvement.

0.3 Thesis structure

Same as the sub-research questions, the thesis can be divided into three main parts. The first part includes chapter one & two, which is the information science part of the thesis. Chapter one discuss the factors needed for release planning, and chapter two discuss the processes and key activities for release planning.

The second part is the mathematical modeling part which includes the chapter three to chapter eight. Chapter three gives a general introduction to the mathematical modeling problem and some basic information on integer linear programming. Chapter four is based on the knapsack model for requirement selection and also presents several management steering mechanism for requirement selection. After the requirement selection, we present the requirement scheduling model in chapter five using integer linear programming. Based on the fact that the scheduling result may not always keep the deadline, we present a combined model for requirement selection and scheduling in chapter six. We also include two more extensions for the combined model, i.e. holiday seasons,

and different time availability in this chapter. In chapter seven and eight, we will talk about how to dynamic adjust release plan and the relationships between each mathematical models.

The third part—the prototype and tests—includes chapter nine and ten. In chapter nine, we discuss two prototypes for requirement scheduling and the combined model. It includes the general information, the software structure and the activity diagram of the two prototypes. Using these two prototypes, we will present two simulation tests in chapter 10. The first simulation test is to find how much the dependencies can influence the requirement scheduling. And in the second simulation, we will compare the two release planning processes i.e. whether one should select requirements first and then schedule them or select and schedule requirement at the same time.

Chapter 11 provides the lucid answer to the research questions and draws conclusion of the thesis .This section also accounts the limitations of the research and proposes some of the possible future dimension of research in this context.

The factors & processes of release planning

In this section, we will discuss the factors involved for release planning and the processes to conduct a release planning. There are two chapters. In the first chapter, we will discuss which factors should be involved in release planning and what are the relationships between them. In the second chapter, based on three case studies, we will discuss the general processes to determine which requirement should be in the next release.

Discussions in these two chapters refer to the information science part of this thesis and should be considered as the foundation of the “mathematical modeling” part. The factors mentioned here will be modeled in the mathematical modeling section, and the release planning processes will guide the relationships between the mathematical models. We also propose several questions in this section, and will try to find the results in later “tools and tests” section.

1. Release planning factors

1.1 Introductions

Software release planning is a complex process and includes many factors. In this chapter, we will discuss what factors should be involved in release planning, and what's the relationships between these factors.

The purpose of release planning is to find the suitable requirements for the next release given the constraints like time-to-market and limited available resources. Obviously, available time, and resources are two main factors. To evaluate requirements, we have identified six factors from literatures, which are: 1) importance or business value, 2) personal preferences of different stakeholders; 3) cost of development, 4) quality, 5), risk 6) requirement dependency. The following table shows the relationship among these factors:

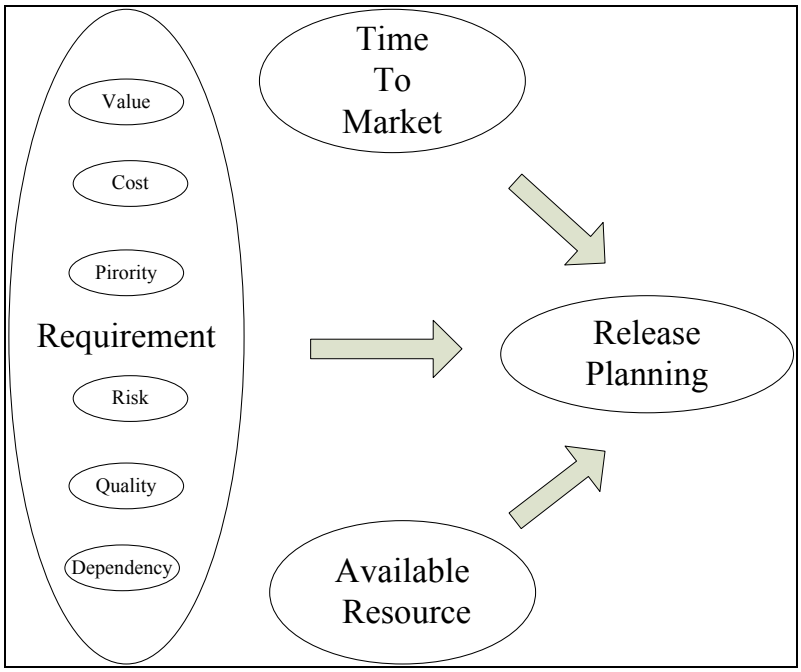


Figure 1.1: release planning factors

1.2 Importance or business value

Different requirements will have different values to the business. Some requirements will be critical, whereas others will be less important though still mandatory. Some potential requirements are not requirements at all but merely desirable though not necessary features or characteristics, and others will be merely characteristics that would be nice to have or items on someone's wish list. Also, some requirements have a tactical usefulness, whereas others have a more long-term

strategic value to the business.

A revenue value can be tangible (like estimated return on investment or estimated revenue) or even intangible (like user satisfaction). Two main categories of revenue value calculation are recognized: absolute value determination, and relative value determination[7]. In absolute value determination, the product manager of a company developing standard software products could determine and estimate value for the revenue. As for relative revenue value calculation, Karlsson and Ryan provide a useful approach through AHP[8] : each requirement can be assigned a value between for example 0-100. Ruhe and Saliu[9] used take the stakeholders' opinions into account, and assign each stakeholder its weight of importance. All the stake holders need to estimate the value of a requirement, and the final value of a requirement is computed as the average weighted value from different stakeholders.

1.3 Personal preference of different stakeholders

Different stakeholders (e.g., customers, users, marketing, operators, maintainers, and architects) will prefer certain requirements over others. This is especially true when practical reasons such as schedule and budget mean that all of the requirements cannot be implemented and released during the current build of an incremental development cycle.

Both internal and external stakeholders are identified[10]. The internal stakeholders include:

1. The **Company board** is responsible for the definition and communication of strategy, vision and mission to the rest of the company. It can occur that a requirement is sent directly to the product manager.
2. The **Research & innovation** has two core responsibilities: (1) doing research to new opportunities for product innovations and (2) finding ways to incorporate improvements or new features into the existing products. The first one results in requirements in the form of technology drivers that are communicated to the product manager
3. The **Service** department is responsible for the implementation of the software product at the customer organization. They need to be aware of new release features and they gather new requirements from the customers
4. The **Development** has as main responsibility the execution of the release plan. The release definition also includes functional explanation of the product requirements that serve as input for the functional and technical design. It may occur that during the development process new requirements can arise, due to more complex requirements than was anticipated
5. The **Support** stands for the helpdesk to answer questions (1st line support) and for small defect repair unit (2nd line support). Large defect repair is usually performed by Development.
6. The **Sales & marketing** is the first contact with a potential customer. Through these contacts new requirements can be gathered.

There are also some external stakeholders like:

1. The **Market** is an abstract stakeholder, standing for potential customers, competitors and analysts.

2. The **external partners**: the implementation partner, the development partner and distribution partner etc.
3. The **Customers** often have new feature requests in the process of closing the deal or during the usage of the product. These requests can be communicated to Services, Sales & marketing, Support, but also directly to the product manager

The opinions of stakeholders can influence the value of a requirement. Ruhe and Saliu[11] took the stakeholders' opinions into account by assigning each stakeholder a weight of importance. All the stake holders need to estimate the value of a requirement, and the final value of a requirement is computed as the average weighted value from different stakeholders.

1.4 Cost of development

The cost of a requirement can be presented by monetary cost and/or labor cost. Briand et al[12] have summarized and compared the common methods on software cost estimations. The typical variables includes: System type, organization type, application type, target platform, productivity factors and so on.

When use labor cost as the cost unit, i.e. the cost is presented by man days or man hours. A benchmarking number linking the cost and the line of code is : one man day = 20 line of codes[13]. When a requirement is transferred to software models and conceptual solutions, the top down resource calculation or bottom up calculations may provide a useful estimation for the costs[7].

Although the cost unit may be the same for different resources, the cost of labors is not the same. A software company may have specialists in different fields, for example, it may have Java developers and C++ developers. When record the cost of a requirement, we need to make clear which types of resource are needed. To make a clear estimation of the cost, we need to know not only how many but also which kind of resources are needed.

1.5 Quality

Quality is a complex and multifaceted concept[14]. From the user's view, the quality is the product characteristics that meet the user's needs. From the product point of view, the quality is more focused on the internal product properties that will result in improved product behavior. It is also difficult to evaluate the quality; in general, software quality includes the following attributes:

1. **Functionality**: a set of attributes that bear on the existence of a set of functions and their specified properties.
2. **Reliability**: a set of attributes that bear on the capability of software to maintain its performance level under stated conditions for a stated period of time.
3. **Usability**: a set of attributes that bear on the effort needed for use and on the individual assessment of such use by a stated or implied set of users.
4. **Efficiency**: a set of attributes that bear on the relationship between the software's performance and the amount of resources used under stated conditions.

5. **Maintainability**: a set of attributes that bear on the effort need to make specified modifications.
6. **Portability**: as set of attributes that bear on the ability of software to be transferred from one environment to another.

The quality of a requirement also influences its attractiveness. For example, if a requirement is highly reusable within a product line, then it might be wise to give it a higher priority so that no system within the product line has to wait for its implementation.

1.6 Risk

Software industry is risky: 80% of software project are late or over budgeted[15]. Shown in the financial market, software industry is one of the industries which have the highest expected rate of return[16]. For release planning, it may well make sense to prioritize requirements by the risks associated with their implementation. For example, one can attempt to implement those requirements having the highest risk first so as to deal with the resulting problems during development. On the other hand, it may make sense to implement the lowest risk requirements first in order to maximize the amount of the system implemented by ensuring that limited resources are not wasted on trying to implement high risk aspects of the system that may be impossible to successfully implement. Postponing the implementation of high risk requirements can also maximize the time available to research the risks and determine appropriate risk mitigation approaches.

It is also important to balance the overall risk of the whole release. Ruhe has provided a method based on generic algorithm to balance the risks of different releases[17]. A tool called EVOLVE+ is developed for decision support. It can help to determine which requirement in which release so that the trade of between risk and revenue are balanced. When determining only on release, Ruhe[18] consider risk in a way similar to how we consider cost: one requirement is associated with a number between 0 to 1 which stands for its risk, and the average risk of the selected requirements should be lower than a certain bound. When dealing with the risk of project plan issues, the risk reflects the uncertainty or probability of a value, for example the expected duration of a job. It is then very complex because a stochastic system is in need For example see , Marjan van den Akker (2004) [19] When there are task divisions to develop a requirement, which means the project are running concurrently in several groups, we can consult the stochastic model of resource constrained scheduling model[20].Research about Stochastic systems on planning are very new and so far models developed have found to be no practical application in the field of release planning.

1.7 Requirement dependency

Requirements are not isolated islands but have complex relationships within them. These relationships are requirement dependencies. In the filed of software release planning, Carlshamre[21] has found about 80% of requirements are interdependent, and only a few requirements are singular.

This practical data suggests that requirement dependencies play an important role for release planning. In the same paper, six types of requirement dependencies have been identified, which are

1. **Combination.** A printer requires a driver to function, and the driver requires a printer to function.
2. **Implication.** Sending an e-mail requires a network connection, but not the opposite.
3. **Time-related.** The function ***Add object*** should be implemented before ***Delete object.*** (This type is doubtful, which is discussed in section 3.1)
4. **Revenue-based.** A detailed on-line manual may decrease the customer value of a printed manual.
5. **Cost-based.** A requirement stating that "no response time should be longer than 1 second" will typically increase the cost of implementing many other requirements.
6. **Exclusion**. In a word processor, it can be either provided as integrated drawing model or a link of external drawing application.

Multiple types of relationships can be found between two particular requirements. For example, R1 may require R2 to function, and R2 also increase the value of R1. It is suggested to priority the dependencies and only consider the dependency with the highest one. The priority order is as follows: 1. Combination, 2, Implication, 3, Time-related, 4 Revenue-based & cost-based, 5, exclusion. For the above case, we may only consider the Implication dependency.

Not all the types of dependency appear equally frequent. Implication and cost-based are the most common types, which can take up 80% of the total dependencies. The least common one is the time-related dependency. Some papers [22] suggest ignoring this type of dependency, and leaving it to the project plan phase; some[23] specially picks this type as Implementation dependencies. A comparison of the two will come in later chapter.

1.8 Time to market

For software product, although the pressure of time to market is evident[24] [25] [1], 80% of software project are late or over budgeted[26]. From a former survey[27], the average time-to-market a new release is about 6 weeks, and within this period, the company can expect receiving around 80 requirements on average.

There are normally two ways to determine the time-to-market of a new release: internally and externally. When the project is budget or quality/scope oriented, the release date is normally internally defined, and the release date is the time when the requirements in the project scope are complete. When the project is market oriented, the release date is normally on the pressure of the external market condition. The focus of this thesis is on product software, so we will use the second method: the release date determined by market condition and not on company situations.

1.9 Resources of the company

The most valuable resource of a software company is the human resource or its specialties. For the release planning problem, to evaluate how many resources are available in the period includes two steps:

- What kinds of resources are available? Everyone has its own specialty, for example someone is good at system analysis, and someone is good at programming. A good understanding of what kind of skills are available is the first step to evaluate the resource of the company.
- How many resources are available? After knowing who are available for the new project, we need to know how long can they work for the project. Each developer may have different time availability, or need to go to holidays, or need to work for other projects. A clear understanding of how much resources are available for different type of resources is important.

2. Software release planning processes

2.1 Introduction

When developing software for a market-place, rather than for a single customer, the pressure on short time-to-market is evident. Market-driven Requirements Engineering processes have a strong focus on requirements prioritization and often deliver incremental releases of a continuously evolving product.

It takes several processes to determine which requirement to be included in the next release. In This chapter, we want to find processes that are typically included in release planning. As to the scope of the problem, we will consider the processes from the time when new requirements are issued until the time when requirements are ready for development.

Based on literature, we will compare three requirement engineering or release planning processes, and try to identify the common processes for release planning. The three cases are:

1. Requirement management process at Baan[28]
2. Requirement engineering process at Ericsson Radio Systems AB[29]
3. Requirement engineering for Time-to-Market Project[30]

2.2 The three cases

2.2.1 The requirement management processes at Baan

The requirement management processes at Baan is shown in the following chart:
When a customer wish related to future product comes into the company, it is recorded as a market requirement. Then the product managers need to link the market requirement to the business requirement, while the business requirement is a product requirement covered by Baan's product, and described in Baan's way. Then a conceptual solution is designed and linked for a business requirement. When the company's manager decided to start a new release, a release initiation document triggers the writing of the corresponding VD and CS. These are then used as input for the development processes, which include writing design documents and actual coding.

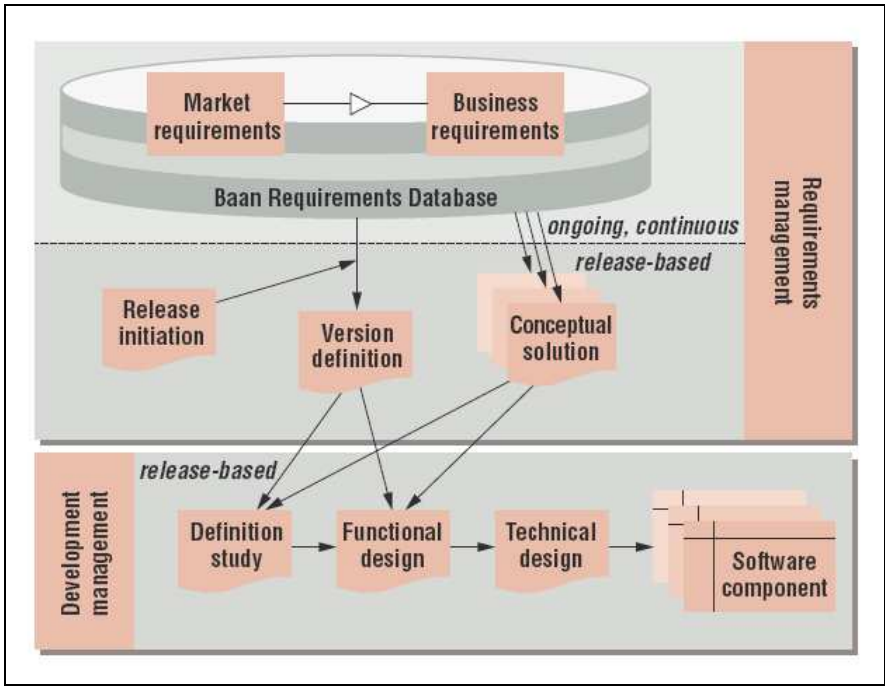


Figure 2.1: requirement management processes at Baan

2.2.2 The requirement engineering process at Ericsson Radio Systems AB

The requirement engineering process at Ericsson Radio Systems AB is as follow:

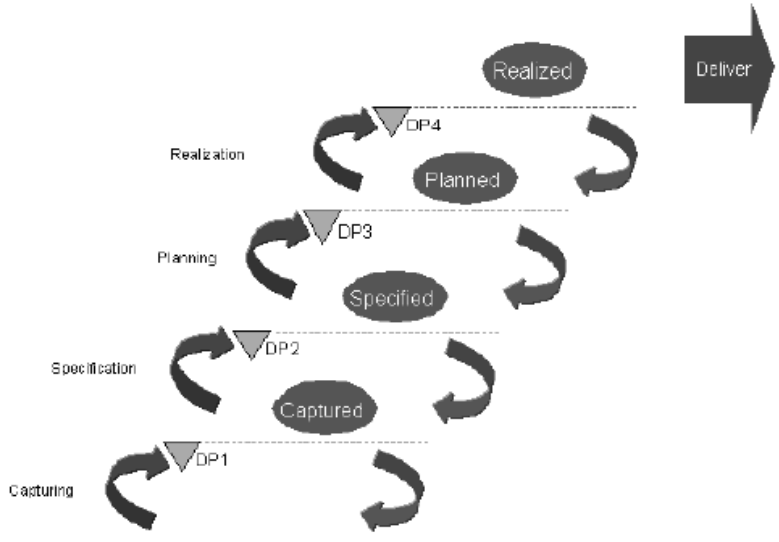


Figure 2.2 The requirement engineering process at Ericsson Radio Systems AB

The RE process at Ericsson is called RDEM model. After a requirement is captured by the product committee, the requirement goes to the specification stage when all information needed to proceed with implementation and verification from a narrow, system-oriented perspective is analyzed. However, in this stage , it does not yet hold any production-oriented information, e.g., when and

how the requirement is best implemented from a customer or product management perspective. If a requirement can be elevated to the planned stage, it will be implemented and verified, and after this is done, the requirement is the done and configured to the product.

2.2.3 Requirement engineering for Time-to-Market Project

The requirement engineering process for time-to-market project is as follow:

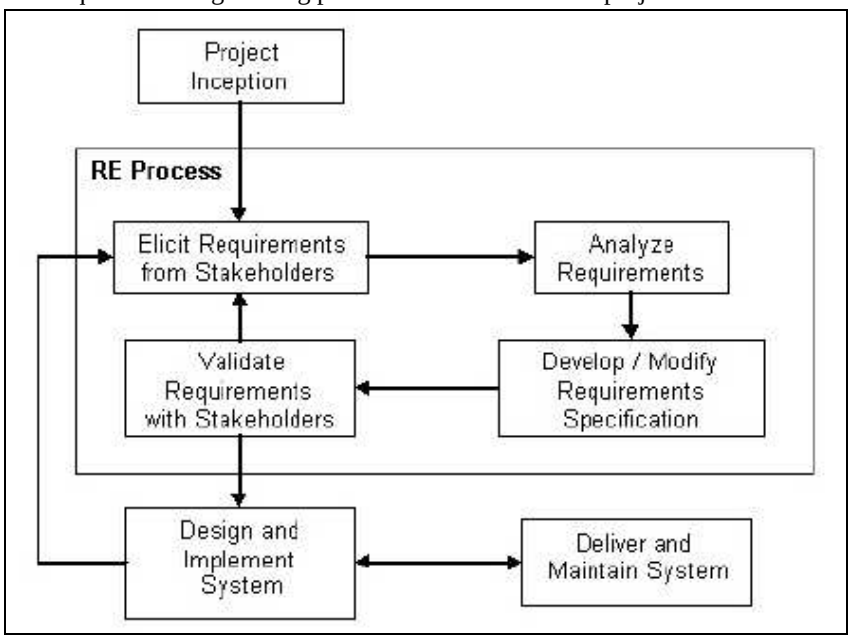


Figure 2.3: Requirement engineering for Time-to-Market Project

The RE process consists of four main activities: elicitation, analysis, specification, and validation. Elicitation is the activity of gathering the requirements from stakeholders. After gathering the requirements, they are analyzed to determine areas requiring clarifications, logical groupings, etc. After being analyzed, the requirements are documented and validated with the stakeholder to ensure that the product developed from the requirements will meet the needs of the stakeholder. Small development increments and formal requirement documentation or experimental prototype are also of high importance.

2.3 The comparison & conclusion

From the cases, we can conclude that:
There are two similarities of the models:

1. The processes “issued” and “specified” appear in the three models. In the Baan’s model, market requirement is the “issued” requirement and it is later refined into the business requirement. The same process to refine the raw requirement into a more structured, and more understanding way is in all the three models.
2. All the three models emphasize on building conceptual models. The Baan’s model has one process for it, the Ericsson’s model build the model on the “specification” phase, and the last

model do it in the “develop/modify requirement specification” phase.

There are also three differences in the model:

1. All the three models have the process to select requirement, but in different ways. The Baan’s case has a special process to select requirement based on release initiation, the Ericsson model select requirement every time when elevating and the last model do it in an iterative way. So, the selection must be a process in the release planning model, but how to do it is not clear yet.
2. The Baan’s model is not an iterative model, while the rest of the models are. The Ericsson model is an iterative model while the third model emphasize on quick iteration.
3. For the Ericsson model, in the “planned” stage, the project plan issues are mentioned, but not in the rest of the model. It is not a big problem because, when the requirements go to the constructed model, normally the first step is to make the project plan. It is only a matter of choice whether we should consider project planning as an issue in the release planning or we leave it when we construct the requirements. In later chapters, we will discuss this question in details.

To sum up, a requirement may goes through the following steps to turn a market wish in to a software product component:

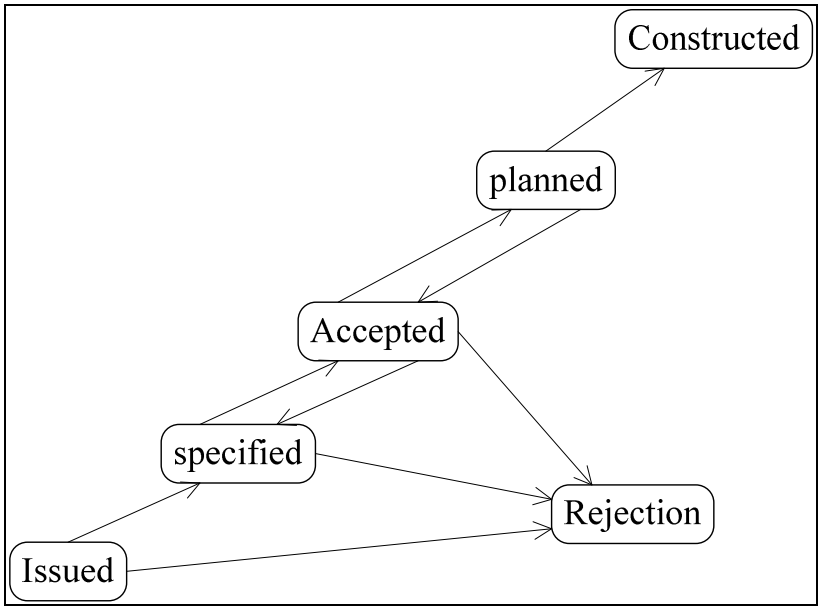


Figure 2.4: different stages for a requirement in release planning

A requirement is only called as a requirement when it is “issued”. The customer, the product manager or any stakeholders inside or outside of the company can issue requirements. For every issued requirement, if it is feasible and clear, further researches will be conduct to specify the requirement into a highly conceptual model or solution. When it is done, we can call this requirement as a “specified” requirement. All the “issued” requirements and “specified” requirements are stored in database for further re-use. When determining the next release, the company will find the suitable requirements for the new release against the available resource in a certain period. If a requirement is selected in this process, it is elevated to the phase of “Accepted”.

The next step is to make a project plan to implement them. In this process, it is also possible to drop some requirements because of the implementation dependencies within the requirements. If a requirement also fits the project plan, then this requirement is “planned”. When the project plan is accepted, the requirement will go to the final step for implementation.

The process is very much like the combination of Baan and Ericsson model. The main differences are: 1) we specially designed a process to select requirement against available resources in a given period of time and 2) make several processes iteratively. There are two reasons for it:

- First, when confronting hundreds or even thousands of requirements, without a proper selection, it is very difficult to conduct the succeeding processes. So, we designed a special process to reduce the scope of the problem.
- Second, in later chapters, we will try to find ways for tooling support. This provides opportunities to repeat some complex and tedious jobs, like select requirement or schedule requirement. Making the processes repeatable can help us to determine whether doing it iteratively can improve the result or not.

The mathematical models of release planning

In this section, we develop and demonstrate an optimization technique based on integer linear programming (ILP), to support software vendors in determining the next release. As with the approach of Jung[40] and Carlshamre[51], our technique is based on the assumption that a release's best set of requirements is the set that has maximum projected revenue against the constraints like available resources, planning period, dependencies, etc. In this section, we demonstrate how to include the factors for release planning in the linear programming model and present several models to realize different functions, like requirement selection, requirement scheduling, etc.

The first chapter gives a brief introduction to the integer linear programming. A simple example representation of the release planning problem is depicted afterward. The second chapter shows the knapsack model and its extensions for requirement selection. The third chapter shows the ILP model to schedule the requirement exactly in time. The Fourth chapter presents a new model which can select and schedule requirements at the same time. Its extensions, like holiday seasons, different time availability, etc are presented afterward. The fifth chapter shows how to dynamically adjust the release plan. The last chapter shows the relationships between the different models.

3. The mathematical modeling

3.1 The quantitative representation of the factors

In the former section, we have discussed the factors and processes of release planning. We have identified eight factors: the requirement's value, cost, priority, risk, quality, dependency, and the time to market as well as the available resources in the company. We have also identified two main processes: selection and scheduling, after the requirements are specified.

To present a general idea of the domain, the following table depicts a simplified example representation of a release planning problem.

Release Definition 3.1						
Nr.	Requirement	Revenues	Total	Team A	Team B	Team C
12	Authorization on order cancellation and removal	24	50	5		45
34	Authorization on archiving service orders	12	12	2	5	5
63	Performance improvements order processing	20	15	15		
25	Inclusion graphical plan board	100	70	10	10	50
43	Link with Acrobat reader for PDF files	10	33		33	
75	Optimizing interface with international Postal code system	10	15			15
35	Adaptations in rental and systems	35	40		20	20
66	Symbol import	5	10	10		
67	Comparison of services per department	10	34		9	25
	Total	226	279	42	77	160
	Available team capacity		180	60	60	60

Table 3.1: an example of a release planning problem (Source from Marjan van den Akker, et al (2004))[7]

For the nine requirements in the datasheet, the factors are estimated. Each requirement has **expected revenue** (in euros) and **expected cost** (required man days per team) associated to it. In addition, the priority of the model is also evaluated. Suppose for instance that the total amount of available man days in the three teams is 60, then we note that team 'A 'has some free capacities while team 'B' and team 'C' are overloaded. Then the set of requirements that brings the maximum revenue has to be determined.

Several scholars have discussed the trade off between the cost and revenue. A very famous model is to use the Integer Linear Programming (ILP). This ILP model is also adopted and extended in this thesis. We will give a clear introduction of ILP in the next chapter and propose the detail model afterwards.

The table (3.1) does not include all the factors we identified before. The factors: the requirement's

value and cost, the time to market and the team capacity are explicitly presented. We can also include the other four factors in the following way.

For the **stake holder's opinions** or **priorities**, Ruhe and Saliu[31] modeled it by assigning each stakeholder a weight of importance. For each requirement, all the stake holders need to estimate the value of it, and the final value of a requirement is computed as the average weighted value from different stakeholders. For example, stake holder 'A' has the weight of 0.4, and estimate value of requirement is 10; stake holder 'B' has the weight of 0.6, and assume the requirement has the value of 15, so the weighted value of this requirement is $0.4\times10+0.6\times15=13$.

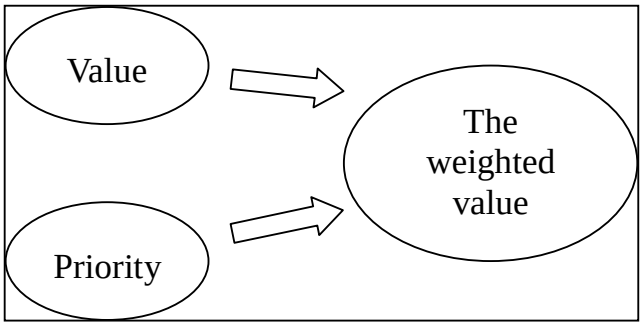


Figure 3.1: relationships between value, priority and the weighted value

The above figure shows the relationships between the value, the priority and the weighted value. When considering the value of a requirement, we can integrate the two factors: value and priority together as the weighted value.

As to the **requirement dependency**, we have identified six types of requirement dependencies in the former chapters, they should also be included when composing release plan. Theoretically, we need to consider the relationship between every pair of requirements, which is $n\times(n-1)/2$ times if we have n requirements to consider. In later chapter, we will show the detail model of requirement dependency.

The factor **quality** is very difficult to quantify. When a requirement requires a certain level of quality, like reliability or reusability, we can model it by issuing a new none-functional requirement[32] to show the influences (for example additional revenue or additional cost) and link the new none-functional requirement with the original one by setting dependencies between them. Clearly, it is an *implication* dependency, because this non-functional requirement for quality requires the original one to work.

Another important factor for release planning is **risk**. Unfortunately, the ILP model is a deterministic system which does not allow variances of the input data. In the former release planning tools, most of them do not include the attribute of risk; Ruhe[33] consider risk in a way similar to how we consider cost: one requirement is associated with a number between 0 to 1 which stands for its risk, and the average risk of the selected requirements should be lower than a

certain bound. In fact, the risk reflects the uncertainty or probability of a value, for example the expected revenue or expected man days in our case. If we want to handle risk in this way, we have to introduce a stochastic system in which all the values have a certain level of uncertainty (for example see Marjan van den Akker[34]).For the sake of simplicity, when the input data is not risk free, we can use the following empirical formula[35] to compute the expected value of a job's duration:

For a job k, we evaluate the optimistic time a_k, the pessimistic time b_k, and the most possible time m_k. Then the expected value of the job's duration is:

$$d_k = \frac{a_k + 4m_k + b_k}{6} \qquad (3.1)$$

We can then use d_k as a risk free value in the model so that we still can solve the problem in a deterministic system.

To sum up, We can use the value and priority to determine the weighted value of a requirement. Using the empirical formula, we can show the risk influence on the value and cost. The quality of a requirement can be modeled as an additional none-functional requirement as well as a dependency. We can show the relationship in the following chart.

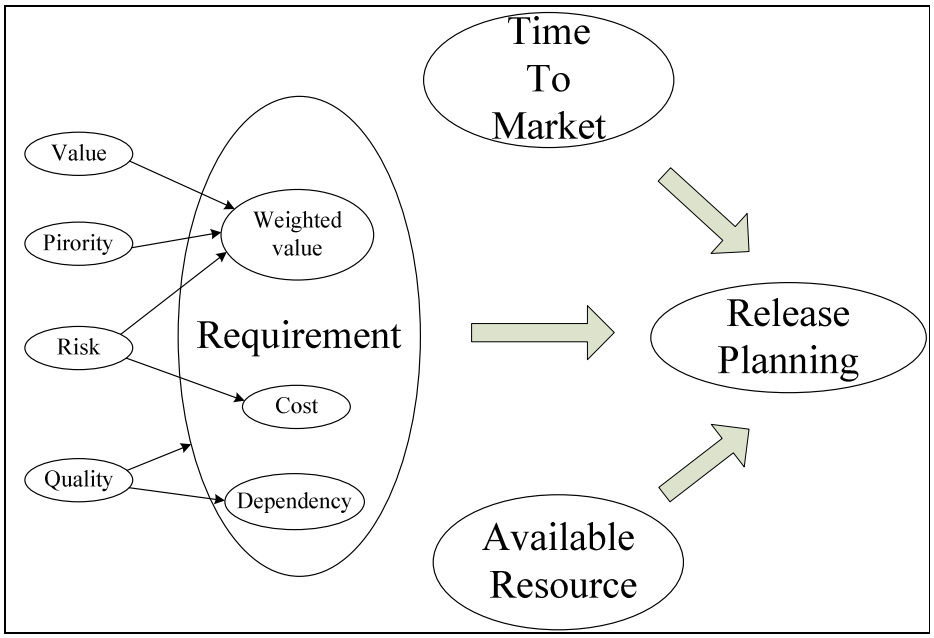


Figure 1.2: the factors and their relationships for release planning

After pre-solve some factors, we can reduce the input factors of the linear programming model. These factors include the weighted revenue, the cost (represented in expected man days) and the dependency for each requirement, the time to market and the available resource. The rest factors, like quality, priority and risk are indirectly included in the model. In the later chapters, we will present how to build the integer linear programming models using these factors.

3.2 Introduction to linear programming

For the sake of completeness, we present a small introduction of linear programming in this chapter. For more background information, we refer the readers to the book of Wolsey[36] (1998) as a reference book.

In mathematics, **linear programming** (LP) problems are optimization problems in which the objective function and the constraints are all linear.

Linear programming is an important field of optimization for several reasons. Many practical problems in operations research can be expressed as linear programming problems. Certain special cases of linear programming, such as *network flow* problems and *machine scheduling* problems are considered important enough to have generated much research on specialized algorithms for their solution. A number of algorithms for other types of optimization problems work by solving LP problems as sub-problems. Historically, ideas from linear programming have inspired many of the central concepts of optimization theory, such as *duality, decomposition,* and the importance of *convexity* and its generalizations.

3.2.1 The standard form

Standard form is the usual and most intuitive form of describing a linear programming problem. It consists of the following three parts:

A **linear function to be maximized**

e.g. maximize $c_1 x_1 + c_2 x_2$

Problem constraints of the following form

e.g. $a_{11} x_1 + a_{12} x_2 \leq b_1$

$a_{21} x_1 + a_{22} x_2 \leq b_2$
$a_{31} x_1 + a_{32} x_2 \leq b_3$

Non-negative variables

e.g. $x_1 \geq 0$

$x_2 \geq 0$

The problem is usually expressed in matrix form, and then becomes:

Maximize $\mathbf{c}^T \mathbf{x}$

Subject to $\mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0$

Other forms, such as minimization problems, problems with constraints on alternative forms, as well as problems involving negative variables can always be rewritten into an equivalent problem in standard form.

3.2.2 The algorithms to solve ILP problem

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.

We can first obtain a linear program which is called the *LP-relaxation*. 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. The simplex algorithm, developed by George Dantzig, solves LP problems by constructing an admissible solution at a vertex of the polyhedron, and then walking along edges of the polyhedron to vertices with successively higher values of the objective function until the optimum is reached. Although this algorithm is quite efficient in practice, and can be guaranteed to find the global optimum if certain precautions against *cycling* are taken, it has poor worst-case behavior.

The first step to solve an ILP is to solve the LP-relaxation. If the solution of the LP-relaxation is integral, it is done. If not, we start with a branch-and bound tree. The ILP is split into several sub-problems corresponding to two or more nodes of a tree. 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 there is not feasible integral solution, obviously, then the node is finished. If the value of LP-relaxation is lower than the best known integral solution (in case we are searching the maximal value), the node can be skipped. Otherwise, new nodes are generated by branching.

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 method is used in most of the ILP software packages.

4. The knapsack model for release composition

In this chapter, we introduce the knapsack model for release planning. Van den Akker[37] et al have presented a linear programming model in this field. For the sake of completeness, we repeat the models from section 4.2 to 4.6 and extend the models in later sections.

4.1 Problem description

In this chapter, we can formulate selecting requirements for the coming release as a combinatorial optimization problem. In such a problem, we have to find the best from a finite but very large number of solutions. From a former survey[38], a product software company gets minimal from 0 to 20 and maximum from 5 to 500 requirements a week. The most probable values range from 1 to 50 with a mean of 13.6 requirements/week. The survey also reported that the mean-time-to-market is about 6 weeks. So, we can expect to handle around 80 requirements every time for a new release. Given the time to market and the fixed resource in the company, it is not possible to develop all of these requirements. A selection is necessary here to determine the coming release, and this is a typical combinatorial optimization problem.

In the former chapter, we have introduced the Integer Linear Programming technique. We will use it to model release planning problem in the later chapters. Although ILP in general are NP-hard, using advanced algorithms and software, we can expect to find an (near-) optimal solution within a reasonable time.

We can model the problem i.e. selecting requirements for the next release as follow. We are given a set of n requirements $\{R_1 \quad R_2 \quad \cdots \quad R_n\}$. For each requirement R_j, we can estimate its revenue as v_j. The cost for a requirement is expressed in the number of man days required in different teams. We assume the time-to-market is given; hence we have to deal with a fixed planning period with limited resources. Therefore, we have to make a selection of requirements to be included in the next release, preferably, with maximal possible revenue. This can be considered as the following optimization problem: find the sub-set of requirements for the coming release such that the revenue is maximal and the available capacity is not exceeded.

We firstly present the basic selection model with team division and without team division. In the later sections, we present three managerial steering mechanisms: team transfer, hiring external team capacity and deadline extension. At last, we show the models to handle requirement dependencies.

4.2 One pool of developers

When there is no team division in the company, we only deal with the total amount of man days in the company. The planning period is T and available working days are $d(T)$ in the planning period. Moreover, let Q be the number of persons working on the release in the company. The available capacity then equals $d(T)Q$ man days.

Moreover, we estimate a_j as the amount of man days needed to implement requirement R_j. Such estimation could come from project managers (top-down) or developers (bottom-up). We model the requirements selection problem by defining binary variables $x_j \quad (j = 1, 2, \ldots n)$. Where:

$x_j = 1$ if requirement R_j is selected;

$x_j = 0$ otherwise.

We can model this problem as an integer linear programming model in the following way:

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

Subject to:

$$\sum_{j=1}^{n} a_j x_j \le d(T)Q \qquad (4.1)$$

$$x_j \in \{0,1\} \qquad \text{For} \qquad (j = 1, 2, \ldots n)$$

This problem is known as the binary knapsack problem[39]. We want to include as much as requirement in the "knapsack" to get maximal value. Jung in 1998[40] has presented the application on requirements analysis. If the company decides that some of the requirements have to be included in the new release in any case, we can add one more constraint that $x_j = 1$ if requirement R_j is fixed.

In this model, $d(T)Q$ is the total available man days in the period T. We assume every

developer has the same available working days $d(T)$ in the period. If the number of working days in the planning period is different from 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.

4.3 Development teams

In the previous model we have been too optimistic by not considering the team divisions. In practical usually, there are different development teams in the company with their own specialization. There are other reasons to form teams in the company, like geographic reasons or management reason. We can include the team-differences in the following way. Let m be the number of teams and suppose team G_i $(i = 1, 2, \ldots m)$ 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 $(i = 1, 2, \ldots m)$. Now we can replace capacity constraint (4.1) by:

$$\sum_{j=1}^{n} a_{ij} x_j \le d(T) Q_i , \qquad \text{for } (i = 1, 2, \ldots m) \qquad (4.2)$$

Note that when $m = 1$, this model is the same with model for one pool of developers. This model is known as binary m-dimensional knapsack problem[41]. Same as the model for one pool of developer, this model can be adapted to the situation with different amounts of man days in the planning period T. We can replace the team capacity $d(T)Q_i$ by $\sum d_p(T)$ where $d_p(T)$ is the number of working days of person p in team G_i in the period T.

4.4 Team transfers

When some teams are overloaded and some team's capacity is not fully occupied, we can consider transferring people to the overloaded team. This may result in additional revenues. We call this team transfers. A transfer will probably result in a decrease of efficiency because the person is not experienced in the new working environment. When a person is working in his own team, we assume he can perform 100% of his capacity, but when a person is transferred from team G_i to team G_k, his contribution in the new team turns to be α_{ik} per day. The factor α_{ik} also reflects the feasibility of a transfer:

$\alpha_{ik} = 0$ if a transfer from G_i to G_k is infeasible, for example, because the specialization of the teams differ too much of geographical reasons.

$\alpha_{ik} = 1$ if persons from team G_i can do the work in team G_k without any reduction in performance, e.g. if the work in the two teams is very similar.

$0 < \alpha_{ik} < 0$ if person from team G_i can work in team G_k. However, their productivity will reduce because of the new working environment.

Note that α_{ik} does not necessarily equal 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 difficult than that in G_k. Then α_{ik} is larger than α_{ki}. It is also clearly that when $i = k$, α_{ik} equals one. Because then the transfer is in the same team.

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}. This value ranges from 1 to $d(T)$. If people can be transferred per day then $U_{cap} = 1$, or if, on another extreme, a person can only be transferred the whole period, then $U_{cap} = d(T)$. If people can only be transferred for a number of complete weeks, then U_{cap} equals five.

Besides the variable x_j, we now define a new group of variable y_{ik} as the number of capacity units transferred from team G_i to team G_k. We can compute the number of capacity units m_i in team G_i equals:

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

New we can present the model with team transfers:

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

Subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \leq U_{cap}[y_{ii} + \sum_{k:k\neq i} \alpha_{ki} y_{ki}] \quad \text{for , } \ (i = 1,2,\ldots m) \qquad (4.3)$$

$$\sum_{k=1}^{m} y_{ik} = m_i \qquad \text{for , } \ (i = 1,2,\ldots m) \qquad (4.4)$$

$$x_j \in \{0,1\} \qquad \text{for} \quad (j = 1,2,\ldots n)$$

y_{ik} non-negative and integral, for $(j = 1,2,\ldots n)$

Constraint (4.3) shows the re-distribution of the capacities in the company. The team G_i 's actual capacity equals the capacity of its own plus the capacity obtained from other teams. Equation (4.4) ensures that total capacity in a team does not get lost.

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.

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. This situation is inefficient but will possibly occur in an optimal solution. 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:

$$\sum_{k:k\neq i} y_{ik} \leq m_i z_i \qquad (4.5)$$

$$\sum_{k:k\neq i} y_{ik} \leq (M - m_i)(1 - z_i) \qquad (4.6)$$

where $M = \sum_{i=1}^{n} m_i$ Constraint (4.5) ensures that team G_i can only send capacity to another team when $z_i = 1$ and inequality (4.6) ensures that other teams can only transfer capacity to team G_i if $z_i = 0$ One can think of situations where the restrictions (4.5) and (4.6) 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’.

4.5 External resources

When the teams are overloaded, the company may consider hiring external personnel in some teams. This decision will not only increase the teams’ capacity, but also bring in a certain cost. We assume the cost of external capacity is linear in the number of man days. We define q_i as the daily cost of hiring external capacity in team G_i, i.e. if u_i is the amount additional man days hired in team G_i, then the cost are $q_i u_i$. Please note that q_i can be different from team to team. Similar to the case of team transfers, we assume that the contribution of u_i external man days is given by $\alpha_{ei} u_i$, where $0 < \alpha_{ei} < 1$.Given the maximal budget E for hiring external personnel. This results in the following model which is an extension of the model from Section 4.3:

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

Subject to

$$\sum_{j=1}^{n} a_{ij} x_j \le d(T) Q_i + \alpha u_i \qquad \text{for , } (i = 1, 2, \ldots m) \qquad (4.7)$$

$$\sum_{i=1}^{m} q_i u_i \le E \qquad (4.8)$$

u_i non-negative and integral, for , $(i = 1, 2, \ldots m)$,

$x_j \in \{0,1\}$ for $(j = 1, 2, \ldots n)$

When $m = 1$, this model is also available. So, this extension also applies to the case with one pool of developers.

4.6 Deadline extension

When the deadline allows a bit range of variance, we can consider postponing the delivery date if

it is profitable to do so. Suppose the delivery date is postponed by δ_T working days, and the estimated additional costs are C per day. We can define δ_T as a (integer) variable in the integer linear program model. We will change the revenue function to $\sum_{j=1}^{n} v_j x_j - C\delta_T$ and the $d(T)$ on the right-hand side of constraints to by $\left(d(T)+\delta_T\right)$.

4.7 Requirement dependencies

4.7.1 Six types of requirement dependencies

In an industrial survey [21] above of requirement dependencies in software product release planning, six types of dependencies have been identified and prioritized. They are:

Example 1: **Combination.** A printer requires a driver to function, and the driver requires a printer to function.

Example 2: **Implication.** Sending an e-mail requires a network connection, but not the opposite.

Example 3: Time-related. The function ***Add object*** should be implemented before ***Delete object.*** (This type is doubtful, which is discussed in section 3.1)

Example 4: **Revenue-base.** A detailed on-line manual may decrease the customer value of a printed manual.

Example 5: Cost-based. A requirement stating that "no response time should be longer than 1 second" will typically increase the cost of implementing many other requirements.

Example 6: Exclusion. In a word processor, it can be either provided as integrated drawing model or a link of external drawing application.

The detail of the requirement dependencies and the prioritization of them have been discussed in former section.

4.7.2 The ILP model for requirement dependencies

Combination

R_i requires R_j, and R_j requires R_i. So, we should select either both of them or none of them. This can be done by add one more constraint:

$$x_i = x_j \qquad (4.9)$$

Implication

R_i requires R_j to function, but not vice-versa. So, we should only select R_i when R_j is selected. This can be done by adding one more constraint:

$$x_i \le x_j \qquad (4.10)$$

Time-related

Either R_i has to be implemented before R_j or vice-versa . As this type of dependency is purely for requirement scheduling not selection, ILP will not model Time-related dependency. In later chapter, we will present a new model which can include this type of requirement dependency.

Revenue based

R_i affects the value of R_j. In this case, if R_i is selected, the value of R_j will change, either positively, or negatively. The following table shows when this dependency will take effect:

R_i	R_j	Will it influence the value?
Not select	Not select	NO
Not select	Select	NO
Select	Not select	NO
Select	Select	YES

From the truth table above, you can see only if both R_i and R_j are selected in the coming release, you can obtain a certain amount of bonus, saying B_{ij} (if R_i decrease the value of R_j, then B_{ij} will be negative). To model this, we need to introduce a new integer variable c_{ij} which equals 1 when both of the requirements are selected.
We also need to add the following constraint:

$$c_{ij} \le (x_i + x_j)/2 \qquad \text{when } B_{ij} \text{ is positive} \qquad (4.11A)$$

$$x_i + x_j - 1 \le c_{ij} \qquad \text{when } B_{ij} \text{ is negative} \qquad (4.11B)$$

The truth table for the upper inequality is:

x_i	x_j	$x_i + x_j - 1$	$(x_i + x_j)/2$	c_{ij}
0	0	-1	0	0
0	1	0	0.5	0
1	0	0	0.5	0
1	1	1	1	1

Which is inline with the truth table of the revenue-based dependency.

Moreover, we also need to change the target equation from $\max \sum_{j=1}^{n} x_j v_j$ to:

$$\max(\sum_{j=1}^{n} x_j v_j + B_{ij} c_{ij}) \qquad (4.12)$$

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

Cost-based

The ICOST dependency means R_k influence the cost of R_j. In the linear programming framework, the cost is represented by man days. Same with revenue-based, this influence can either be positive or negative. The following table shows when this dependency will take effect:

R_k	R_j	Will it influence the cost?
Not select	Not select	NO
Not select	Select	NO
Select	Not select	NO
Select	Select	YES

Here, we can say if both R_k and R_j are selected in the coming release, we can save some man days, saying $S_{i,j\to k}$,(if R_k increase the man days needed for R_j, then $S_{i,j\to k}$ is negative). To model it, we first need to introduce a new variable i_{jk}. Same as revenue-based, a new constraint for i_{jk} will be added:

$$i_{jk} \le (x_k + x_j)/2 \qquad \text{when } S_{i,j\to k} \text{ is positive} \qquad (4.13A)$$

$$x_k + x_j - 1 \le i_{jk} \qquad \text{when } S_{i,j\to k} \text{ is negative} \qquad (4.13B)$$

We also need to adjust the original constraint for man days from $\sum_{j=1}^{n} a_{ij} x_j \le d(T) Q_i$ to:

$$\sum_{j=1}^{n} a_{ij} x_j - i_{jk} S_{i,j\to k} \le d(T) Q_i \qquad \text{for } (i = 1, 2, \ldots m) \qquad (4.14)$$

Exclusion

OR dependency means we need either R_i or R_j but not both. It is also possible that we need neither of them. To model this type of dependency, we can set one additional constraint as follow:

$$x_i + x_j \leq 1 \qquad (4.15)$$

4.8 Dependency generalization

So, far, we only apply the dependencies between pairs of requirements, however, the dependencies mentioned above can be generalized to the situation with larger sets of requirements which we call a package. For example, if we develop a complete package for marketing, we can obtain some extra value besides the revenues from individual requirements in the package. We can use the value-based dependency to model all the requirement pairs, but it is very difficult to determine: first, which requirement influences which, and second: how to divide the package bonus into each requirement pairs.

4.8.1 Construct a package

This problem can be represented in the following way:

The package P_t consists of a set of requirements R_j, $j = 1,\ldots,l$ and $l_t \leq n$ (the package contains at most all the requirements). If we implement all of them, we can obtain the bonus value of B_t from the package. To model this, we need a binary variable y_t to determine whether we have the package or not.

Now, we need to add the constraints:

$$y_t \leq \frac{\sum_{j=1}^{l_t} x_j}{l_t} \qquad \text{for} \quad j = 1,\ldots,l_t. \qquad (4.16)$$

At last, we can add:

$$B_t y_t \qquad (4.17)$$

to the revenue function.

When the value of a package is shown as cost-reduction rather than additional values, we can

model it in the following way. If implementing the package P_t can reduce the work in team G_i for a_{it}, we can include the cost reduction effect by changing constraint 4.2 into:

$$\sum_{j=1}^{n} a_{ij}x_j - y_t a_{it} \le d(T)Q_i \qquad (4.18)$$

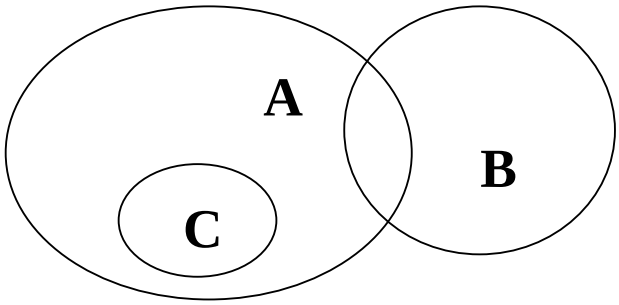
4.8.2 Requirement and Package

4.8.2.1 Dependencies

So far, we know how to construct a requirement package. Same as each requirement, a package also has its expected revenue B_t and its decision variable y_t. So, a package can be considered as a "special requirement". If we do so, the requirement dependencies mentioned above then can be generalized to the dependencies between requirements and packages or even between packages. Similar to how we use the binary variable x_j to model dependencies, we can use the binary variable y_t as well. For example, If the package P_t requires requirement R_j to function, it is typically an *implication* dependencies, then we can set $y_t \le x_j$ to model this dependency.

4.8.2.2 Package relationships

However, because a package is a group of requirements, it is actually a set not a basic element as requirement is. So, there are more issues to discuss about. Think of the following three packages, A, B, C. Each package contains several requirements. Package A include all the requirements that package C has, and package A also share some requirements with package B.



We would expect all the packages are like C and B, where the packages are disjoint. In this case, we can use the package freely, just like a requirement, and no further constraint is needed.

Another case is like A and C, where package A includes package C. For example, package C is a basic package of marketing, where only key requirements are selected; on another hand, package A is an extended version of marketing package, where we have not only the key requirements but also some supporting requirements. The packaging is nice and common, but this situation leads to

a problem. Because package C is just a part of package A, if we finally select package A, we actually automatically select the package C, and you will get the bonuses from both A and C. This should not be the case, because package C are in fact counted twice. We can solve this problem by adding a new *exclusion* constraint:

$$y_a + y_c \leq 1 \qquad (4.19)$$

If the condition is satisfied for both packages, the ILP can automatically select the higher bonus package, which is normally the one with more requirements.

The third case is like A and B, where two packages share some requirements together and have some others for their own. For example, one package focuses on route planning, another focuses on inventory management, and both package contain some requirements on saying handling of customer orders. If both the two packages are selected, it seems not reasonable to have all bonuses, because a part of the two packages is overlapping. We can deduct some of the value by using negative revenue-based dependency. For example, If this overlapping make a lose of w_{ab}, we need to:

- First, introduce a new variable y_{ab}
- Second, subtract $w_{ab}y_{ab}$ to the revenue function
- Third, set a new constraint $y_{ab} \geq y_a + y_b - 1$

If the two packages share large amount of requirements, it is also possible to set *exclusion* dependency between them. This is a management decision. Hereby we suggest using revenue-based dependency because it has a higher priority.

4.8.3 Penalty package

In requirement dependency, the revenue/cost-based dependency can model both positive and negative influence on revenue or cost. In the section above, we introduced the package with additional values or decreasing cost; in this section, we will introduce the package with negative values or additional cost which we call as a “penalty package”. This package is useful when overlapping happens. For example, a group of requirements have the similar function of providing user manual, but doing in different ways like through website, or electronic document or paper version. Having three of them may have some overlapping so as to reduce their overall values. Another case regarding to cost is when having the whole package requires additional work. For example, when the dependencies between the requirements in the package are very complex, implementing the whole package may needs additional cost to handle these dependencies. In these two situations, we need to construct a penalty package to show the value reduction or additional works. Unfortunately, we can not use the method in section 4.8.1 to construct penalty package.

When we use constraint 4.16 to construct a package, the selection variable y_t still has the freedom to be zero even when all the requirements in the package are selected. When then value of the package is positive or the package can reduce the implementing cost, the ILP model will turn y_t to one so that to obtain additional value or cost reduction. But when the package value is negative, the ILP model will let y_t to zero so that not to lose values or have additional cost. This problem requires us to construct a "penalty package" in a different way.

A penalty package can be constructed in a similar way as a bonus package, the differences are:
First the constraint 4.16

$$y_t \le \frac{\sum_{j=1}^{l_t} x_j}{l_t} \qquad \text{for} \quad j = 1,\ldots,l_t. \qquad (4.16)$$

Should be replaced by:

$$y_t \ge \sum_{j=1}^{l} x_j - l_t + 1 \qquad \text{for} \quad j = 1,\ldots,l_t. \qquad (4.20)$$

Second, we can not set compulsive requirements in the package, because these compulsive requirements are in fact modeled as *implication* dependencies between requirements and package.

The reason is because the ILP is searching for good results. If it is a bonus package, the system will automatically go for it when the condition is satisfied, because it increases the value. So, we only need to set an upper bound as a lunching condition, like in inequality (1). However, fulfilling the condition does not compulsively launching the package. The decision variable p_j still has the freedom to 0, because the condition is an upper bound. That is why we can set additional dependencies for the package. On the contrary, ILP system will not go automatically for penalty package, because it decreases the revenue. So we need to set the condition in (5) as a lower bound, which means if the condition is satisfied, the penalty will compulsively lunch. In this way, we can not set additional dependencies for a penalty package, because immediately after we construct the penalty package i.e. when the condition is fulfilled, the decision variable p_j will turn into a constant which is 1.

The conclusion for penalty package is although we can construct penalty package if necessary, we can not set additional dependencies for penalty package, neither between it with other requirements, nor within itself.

4.9 Model personal differences

4.9.1 Problem statement

So far, the ILP model is based on team capacity rather than individual members. So, it has made two assumptions to eliminate the difference between each team member. First, every team member has the same productivity; second, all team members work in the same period, from the beginning of the project to the end. In addition, as mentioned in former chapter, the personal preference is also an important issue for release planning. In order to show the personal differences in capability, available time, and preference, we need to extend the current team based model to a people based model.

One of the important scenarios in the knapsack model is team transfer. However, one important parameter is α_{ik}, which shows the contribution of a person in team G_i when moved to team G_k. This parameter is needed between every pair of team. However, when consider personal difference, this parameter is very difficult to evaluate. Considering the following case:

developer	Own team	Other team
Alice	1	0
Bob	1	1
Carol	1	0.6
David (team leader)	1	1.5

As shown in the table, Alice can not work in the another table; Bob works as well in his own team as in other team; Carol works only 60% when transferred to another team while David works even better in another team, but unfortunately, he is the team leader, and not allowed to move. In such situation, it is very difficult to evaluate the team transfer rate between these two teams, or not possible to get a precise one. There is a need of a new model to solve the problem.

4.9.2 The basic model

Let m be the number of teams and each team is $G_i (i = 1, \ldots, m)$. Assume there are n persons in the company, and each person is $H_k (k = 1, \ldots, n)$. The next to do is to create a $m \times n$ matrix showing each person's performance rate in each group. Lets use β_{ik} reflect the performance rate of the person H_k in the team G_i. There are several possibilities:

- $\beta_{ik} = 0$ If the person H_k is not capable of working in the team G_i. It can be technical reasons, geographic reasons, management reasons or something else.
- $\beta_{ik} = 1$ If the person H_k works in the team G_i at the standard performance rate. This standard rate can be the number of lines per person per day[1], or other standard the organization use. In this model, it should be available within the whole organization. Please also note that this standard is also the standard rate to estimate the development man days for a requirement.
- $\beta_{ik} = others$ If the person H_k's performance in the team G_i is considered to be better or poorer. If it is less than one, it means this person works poorer than the company's wish. It can also be higher than one, which means this person can do a better job there.

We can then assign our developers by introducing a new binary variable z_{ik} ($i = 1, \ldots, m$), ($k = 1, \ldots, n$), where:

$z_{ik} = 1$ if the person H_k works in the team G_i

$z_{ik} = 0$ if the person H_k does not work in the team G_i

If we assume one person must work full time and can only work in one team in the whole development period, we can add a group of constraints:

$$\sum_{i=1}^{m} z_{ik} = 1 \qquad \text{for } k = 1, \ldots, n \qquad (4.21)$$

This constraint makes sure that one person can only work in one team, and it applies for all the team members.

The team's capacity is the sum of the capacity of all the team members. Then, instead of using a fix number Q, the team G_i'capacity is:

$$d(T)\sum_{k=1}^{n} \beta_{ik} z_{ik} \qquad \text{for } i = 1, \ldots, m \qquad (4.22)$$

Now, we can replace team capacity of $d(T)Q$ by $d(T)\sum_{k=1}^{n} \beta_{ik} z_{ik}$, then the original model will be extended to a model based on people rather than teams.

[1] In Sjaak's information business course, the standard rate is 20 line of code per person per day. But this is the figure for the whole development process. I don't know whether this can also be the benchmarking figure here.

4.9.3 Working in different teams (team transfer)

Till now, we have built a fundamental people-based model. This model has taken members' different capacity into account. However, there are still some management issues to think about, for example scheduling people more flexibly, or modeling working periods for different members. They will come in this chapter as the extension of the basic model.

4.9.3.1 Working in different teams (team transfer)

Sometimes, a developer needs to work in different teams due to various reasons. For example, the team capacities are not in balance; delay in other teams or for management reasons. We can model this by introducing a new group of integer variable y_{ik} which is the number of days the person H_k works in the team G_i.

No, instead of using (4.21), we need to add a new group of constraints:

$$\sum_{i=1}^{m} y_{ik} = d(T) \qquad \text{for } k = 1,\ldots,n \qquad (4.23)$$

These constraints make sure that one developer will assign all his/her working days to at least one of the groups. Please note that $d(T)$ can differs from person to person.

Please note now y_{ik} is the number of days a developer works in a group. So, the total group capacity is now:

$$\sum_{k=1}^{n} \beta_{ik} y_{ik} \qquad \text{for } i = 1,\ldots,m \qquad (4.24)$$

We can then replace $d(T)Q$ by $\sum_{k=1}^{n} \beta_{ik} y_{ik}$ in the team based model and it will turn to be a people based model then.

In section 4.4, we introduced the concept of Capability Unit for team transfer. This concept is also applicable here. We can define the variable y_{ik} is the number of Capability Unit a developer works in a certain team. To include this, we need to replace the constraint 4.23 by :

$$\sum_{i=1}^{m} y_{ik} = \frac{d(T)}{U_{cap}} \qquad \text{for } k = 1,\ldots,n \qquad (4.25)$$

And the team capacity (4.24) by::

$$U_{cap} \sum_{k=1}^{n} \beta_{ik} y_{ik} \qquad \text{for } i = 1, \ldots, m \qquad (4.26)$$

4.9.3.2 Management issue for team transfers

More often, a team member is not willing to transfer or only accept a limited times of changes. For example, a team member is not will to work for more than 2 teams within the development period. Sometimes, it is also necessary to set a lower bound for the number of days one developer works in a team. Like we only transfer a person to another group if he/she needs to work more than 5 days there. We also need to deal with personal reasons, for example two developers always want to work together or do not want to work together. All these management issues will be discussed in this section.

4.9.3.3 Basic constraint

Before we model the management issues, we first need to set a new group of constraints to link two variables z_{ik} and y_{ik} together. The definition of z_{ik} and y_{ik} can be found in 4.9.2 and 4.9.3.

$$y_{ik} \le z_{ik} \times d(T) \qquad \text{for } k = 1, \ldots, n$$

$$\text{for } i = 1, \ldots, m \qquad (4.27)$$

This constraints means: if a person works in a certain group, he can work no more than the whole project period there. The constraints on z_{ik} and y_{ik} will be explained next depending on management choices. Even though it might seem unnecessary here, however, we will need it for all the further extensions.

4.9.3.4 Limit the number of transfers

If we want to limit the number of teams one person works in, we need to use the variable z_{ik} again. (The definition of z_{ik} can be found in section 4.9.1) If we only want a developer H_k works in no more than N teams. We can add the following constraint:

$$\sum_{i=1}^{m} z_{ik} \le N \qquad (4.28)$$

Please note that N can be different from person to person.

4.9.3.5 Lower bound of working days

We can also set the lowest working days a developer works in a team. If a developer H_k need to work more than M days in a team, we need to set a new group of constraints:

$$M \times z_{ik} \le y_{ik} \qquad \text{for } i = 1, \ldots, m \qquad (4.29)$$

4.9.4 Personal preferences

4.9.4.1 Preference to team

A developer may only want to work in a few teams. To model this, we can only define the variables z_{ik} and y_{ik} for the teams G_i where person H_k prefer to work.

4.9.4.2 Working with others

When a developer has person preference to other developers, like he/she wants to work with someone else or he/she does not want to work with another one. We can use the variable z_{ik} to model his preference. When developer $H_{k'}$ only want to work with H_k, we can add the a group of constraints that:

$$y_{ik} = y_{ik'} \qquad \text{for} \qquad i = 1, \ldots, m$$

Or when developer $H_{k'}$ does not want to work with H_k, we can add the a group of constraints that:

$$y_{ik'} + y_{ik} \le 1 \quad \text{for} \qquad i = 1, \ldots, m$$

4.9.4.3 Key team members

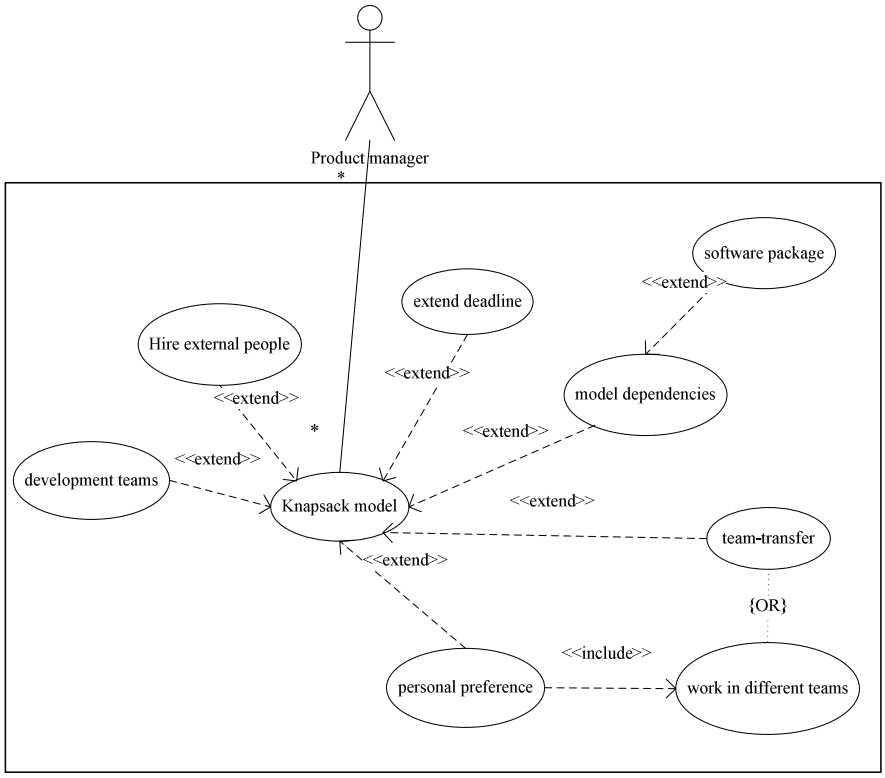
Some team members are very important for a team, for example the team leader. When make the people planning, it is better to fix these people in the team. If we want to fix the person H_k in the team G_i, we can set the decision variable z_{ik} to 1.

4.9.4.4 Different time availability

In the team based model, every team member has the same working period which is equal to the planning period $d(T)$. In the people based planning model, it is possible to assign each people a different working period. This will be represented as $d(T_k)$ ($j = 1, \ldots, n$) for the developer H_k. Then you need to replace $d(T)$ by $d(T_k)$ in the constraint (4.23) and (4.27).

4.10 Chapter conclusion

In this chapter, we introduced the knapsack model for release planning. Based on the knapsack model, we also introduced several management steering mechanism as the extensions of the knapsack models. The relationships between these models are depicted in the following table.



Use case chart of release planning

As shown in the picture, the heart of the model is the knapsack model. And all the other models are the extensions of this one. The combinational use of any of the models is applicable except only one case: when we need to include personal preferences, we introduced a new model for transferring people and this one has the same function as team transfer described in section 4.4. These two models are not compatible, and it is only possible to use one based on whether the personal preference model is included. In the picture, this conflict is shown as an OR constraint between the two models.

5. Scheduling the requirements

5.1 Problem statement

After we select the requirements, a very important process in the release planning is to schedule the activities exactly in time. In this chapter, we will discuss what will influence the scheduling process, and how to make a project plan with minimal time span.

5.1.1 Precedence constraints

In the former chapter, we have indicated five types of requirement dependencies. These requirement dependencies will continue influencing the schedule of the development processes. When schedule the requirement, we should take two out of five types of requirement dependencies into consideration—*implication* and *cost-related***.** They are considered as implicitly mentioned precedence constraints [51]. If requirement R_j influences the implementation cost of requirement $R_{j'}$, or if requirement $R_{j'}$ requires R_j to function, it is better to start develop $R_{j'}$ after R_j is finished. Let us denote this precedence constraint by $R_j \prec R_{j'}$.

According to a former survey[21], *implication* and *cost-related* dependencies take up a great portion in practice: three out of five cases reported them as the most common dependencies and took up to about 80% of the total requirement dependencies. After influencing the requirement selection, these dependencies are inherited and will also influence the project schedule.

Besides the inherited precedence constraints, it is also possible to set *time-related* dependencies for project plan purposes. This dependency express project plan issues like: "we need to develop $R_{j'}$ after R_j". For example, it is better to develop the function "delete an item" after develop "add an item".

Although Carlshamre[21] suggested only taking one type of requirement dependency between a pair of requirements, but in fact in his discussion he interpreted more than one. For example, if requirement $R_{j'}$ requires R_j to function, this *Implication* dependency means not only that $R_{j'}$ logically require R_j to function, but also that $R_{j'}$ need to be developed after R_j. So this relationship is in fact an *implication* plus a *time-related*. Theoretically, there can be more relationships between a pair of requirements, for example, $R_{j'}$ requires R_j to function and influences its cost, then, they have *implication* and *cost-based* dependencies.

If it is allowed to set multiple dependencies between a pair of requirements, not all combination of the six are valid. The first exception is that: exclusion is not compatible to any other dependencies, because we can at most have one of the two requirements, so building more relationships between them is not necessary. The second exception is Combination and Implication, because Combination means $R_{j'}$ require R_j, and R_j also requires $R_{j'}$. The rest of the types can work together without any problems.

We can divide the requirement dependencies into three groups-

- The functional dependency including Combination, Implication and Exclusion;
- The value-related dependency including revenue-related and cost-related dependency
- The time related dependency.

The following table shows how the requirement dependencies influence the requirement selection and requirement scheduling. The functional and value-related dependencies can influence the requirement selection, while the *implication*, *cost-related* and *time-related* dependencies will influence the requirement scheduling. For simplicity reason, we can define these three types of requirement dependencies as precedence constraints. A precedence constraint is denoted as $R_j \prec R_{j'}$, if R_j need to finish before requirement $R_{j'}$ starts.

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

Table 5.1: the influence of requirement dependencies on requirement selection and scheduling

It is clear that the precedence constraint can influence the development sequence in a team. However, the question is: as we have already selected requirement based on our capability, why should we still consider scheduling activities as an important issue in release planning? Can the precedence constraint also influence the deadline of the project?

5.1.2 No precedence constraint

It is not a problem if there are no precedence constraints between the requirements. As each team works independently, they just need to randomly give a permutation of all the jobs, and develop

them one after another. In this way, we can guarantee that the project will be on time.

Proof: We have selected requirements based on the constraint $\sum_{j=1}^{n} a_{ij} x_j \le d(T) Q_i$ for all the teams G_i $(i = 1, \ldots, m)$. So, in the release plan, we can get $\sum a_{ij} \le d(T) Q_i$ for all the team G_i. The development time for requirement R_j in team G_i equals the man days a_{ij} divided by the number of developers Q_i. Because each team work independently and continuously, the total development time is $\sum \frac{a_{ij}}{Q_i}$ in team G_i. Given the constraint $\sum a_{ij} \le d(T) Q_i$ we can get that $\sum \frac{a_{ij}}{Q_i} \le d(T)$.

5.1.3 One pool of developer

If we have time-related requirement interdependencies, when there is only one team i.e. the requirements are developed by one pool of developers, scheduling the activities is also not a difficult issue. We can first draw a Directed Acyclic Graph (DAG) by setting the requirements R_j as vertexes and setting the precedence constraint $R_j \prec R_{j'}$ as a directed edge $(R_j, R_{j'})$. Then the schedule of the development is the topological sort of the directed acyclic graph. A topological sort of a DAG is a linear ordering of all its vertices such that if G contains an edge $(R_j, R_{j'})$, then R_j appears before $R_{j'}$ in the order. The topological sort algorithm is as follow:

Topological-Sort (G):

1. Call depth-first search (G) to compute finishing times $f[R_j]$ for each vertex R_j.
2. As each vertex is finished, insert it onto the front of a linked list.
3. Return the linked list of vertices

We can compute this sort in $O(V + E)$ time where V equals the number of requirements and E equals the number of dependencies[53]. We can also prove that the project will finish on time.

Proof: Let π be a topological sort of the requirements based on the precedence constraint. Because the team can develop requirements continuously, the total time span to finish them is

$\sum \frac{a_j}{Q}$. As we selected requirements based on $\sum_{j=1}^{n} a_{ij} x_j \le d(T) Q_i$, we will get that $\sum a_j \le d(T) Q$. This yields the conclusion that $\sum \frac{a_j}{Q}$ is less or equal to $d(T)$.

The following figure shows one example of topological sorting. In the chart, the nodes are the requirements and the arrows represent the precedence constraints, which point to the immediate successor of the requirements.

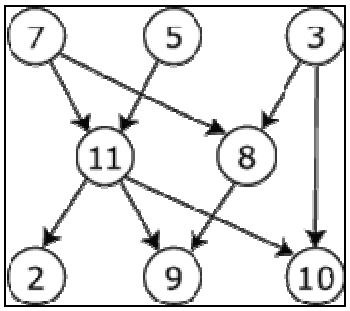


Figure 5.2 example of topological sorting

The topological sorting gives a linear sequence order of the requirements so that when there is a precedence constraint between R_j and $R_{j'}$ then R_j appears before $R_{j'}$. Using the algorithm above, we can get the order of 7,5,11,2,3,8,9,10. Please note that the topological sorting of a chart is not necessary to be unique, and does not have to be depth-first search. The following two orders are both valid topological sorting of the chart:

- 7,5,3,11,8,2,9,10 (width-first search)
- 7,5,11,2,3,10,8,9

This order can be used as the schedule for development.

5.1.4 Schedule with team and precedence constraint

When there are precedence constraints and there are multiple development teams in the project. The scheduling problem becomes very complex.

Let's have a look again at the small release sample:

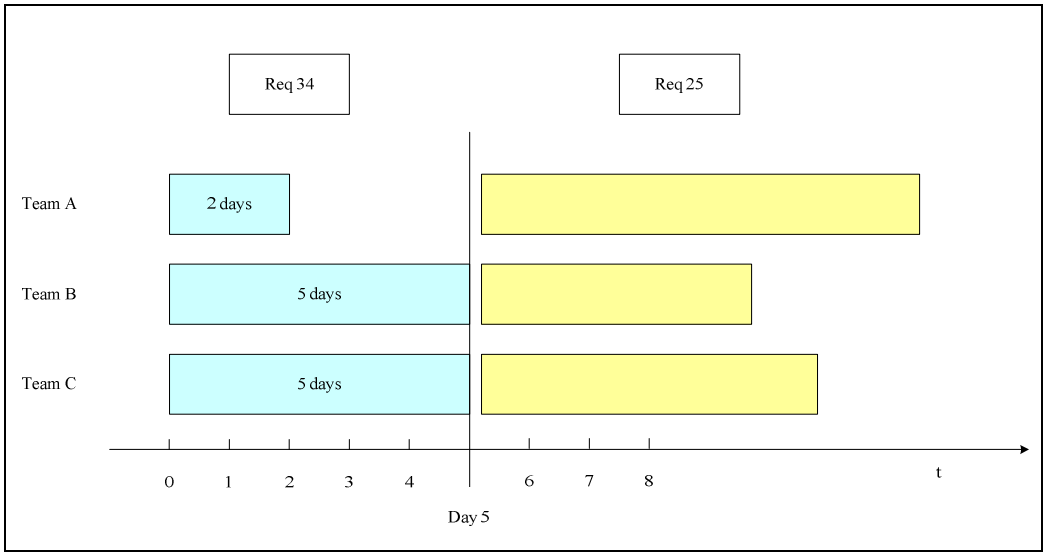
Release Definition 5.1

Prio	Nr.	Requirement	Revenues	Total	Team A	Team B	Team C
1	12	Authorization on order cancellation and removal	24	50	5		45
1	34	Authorization on archiving service orders	12	12	2	5	5
1	63	Performance improvements order processing	20	15	15		

1	25	Inclusion graphical plan board	100	70	10	10	50
1	43	Link with Acrobat reader for PDF files	10	33		33	
1	75	Optimizing interface with international Postal code system	10	15			15
1	35	Adaptations in rental and systems	35	40		20	20
1	66	Symbol import	5	10	10		
1	67	Comparison of services per department	10	34		9	25
		Total	226	279	42	77	160
		Available team capacity		180	60	60	60

If we use the knapsack model, the solution with maximal revenue is to select, requirement 34, 63, 25, 43 and 66 for the next release. In this way we can compose a release plan with the highest revenue of 147.

If there is a precedence constraint between them, for example, between Requirement 25 and Requirement 34, what will happen?



In this chart, the blue bar shows the time for Requirement 34, and the yellow bar represents the time for Requirement 25. We can see from the chart that actually at day two, Team ‘A’ has already finished their work for Requirement 34, however, because Team ‘B’ and Team ‘C’ still need another three days to finish their job, so the Requirement 34 does not finish at day 2, but actually finish at day 5, when Team ‘B’ and Team ‘C’ finish their jobs for this requirement. So, if Requirement 25 needs to be development after Requirement 34, the earliest start time is at day 5. So, here comes a problem - Team ‘A’ has wasted three days on waiting Team ‘B’ and Team ‘C’ to finish their jobs. It is possible that during these days, team ‘A’ can do something else, for example, developing some other requirements which do not depend on Requirement 34. However, the risk of waiting others still exist, and this risk raises an important issue that how shall one design a schedule which make teams do not waste time on waiting others or if this problem can not be

eliminated, how should one minimize such waiting time and also minimize the total project span of the whole release project?

Another issue is: if we need to spend too much time on waiting others, is that possible to re-select the requirements so that the release plan fits a predetermined deadline? For example, if Requirement 43 depend on Requirement 25. Even if requirement 25 starts at the first day, it takes Team 'C' 50 days to finish their job, and even if we immediately start to develop Requirement 43, it will still take Team 'B' another 33 days to complete this requirement, so the project span will not be less than 50 + 33 = 83 days. If we still want to keep the 60 days as the deadline, we then need to re-select the requirements. This issue is even more important than to schedule the selected requirements, because it is a more market orientated approach [42] and the pressure on time-to-market is evident[43] [44].

In this chapter, we will focus on solving the two problems mentioned above: under the circumstances that there are both development teams in the company and precedence constraints between requirements:

1. How should we schedule the selected requirements to minimize the total development time span when there are precedence constraints between the requirements?
2. Given a predetermined release date, how should we select requirements with precedence constraints to maximize the revenue of the new release?

In addition, the assumption so far is that all the teams are available for the whole development period. However, what if some teams have other activities to do or what if there are pre-arranged holidays during the development time? How should we synchronize them? It may be highly relevant and normal for international companies, because the public holidays in each country are significantly different.

5.2 Scheduling with team & precedence constraints

In this section, we will try to answer the question that if there are precedence constraints between requirements, how can we make a schedule with minimal project time span?

5.2.1 Four basic assumptions:

1. Each development team works independently on one requirement. If one requirement needs the efforts in several teams, there are no predefined sequences between the jobs in these teams.
2. One requirement is available to use only after all its components are finished. If one requirement needs the effort for multiple teams, it is considered ready to use only after all the teams finish their jobs for this requirement.
3. One team can only develop one requirement at one time. If a team wants to parallel develop requirement, we can divide the team into several sub-teams, where in these sub-teams, they work on just one requirement.

4. One team will continue its work until it is done, we do not allow a team to switch to develop another requirement before finishing the current one. However, it does not mean that the development can not be interrupted, the team can go to holidays or be interrupted for other reasons, but after the break, they still need to continue with the unfinished requirement.

5.2.2 The RCPSP model

This schedule problem is so unique that it does not fall into any traditional machine scheduling problem. It is not a multi-stage problem because there is no predefined sequential order in a requirement. It is hardly a parallel-machine problem because the jobs for a development team have already been given. It is not a single machine problem as the schedule of a team also depends on that of other teams. Most likely, it is a relaxed job shop problem with no prescribed route, but it is not very efficient to model it as a job shop problem, because it is regarded as one of the hardest in combinatorial optimization[45].

One widely used technique to solve precedence constraints is PERT (program evaluation and review technique). We can compute the minimal project span by identifying the critical path [35] , and also compute the earliest start and latest start for each job. However, one problem in PERT is that it does not consider the resources it use at a time, in our case, the resource is our team capacity, and a team is only capable of development one requirement at a time. So, the result of PERT is not very practical since it can plan a team to develop a couple of requirements at a time. To include both resource constraint and precedence constraint, the Resource Constrained Project Scheduling Problem (RCPSP)[46] is a good reference to use.

The RCPSP model is often used when a project plan is given limited amount of resources available at a time. Normally, the amount of resource is not necessarily fixed at one like modeled in our case. Here, we model the requirement scheduling problem as a special case of RCPSP problem because it is too unique to fit in a machine scheduling problem category.

RCPSP is an NP-Hard problem[47]. The problem complexity caused many scholars to development heuristics method[48] or exact algorithms[49]. One such was proposed by Aristide Mingozzi et al (1998), and gave solution to the problem with hundreds of jobs[46].

5.2.3 Problem description

We can now model the problem in the following way:

We are given a set of n requirements $\{R_1 \quad R_2 \quad \cdots \quad R_n\}$. Let m be the number of teams G_i $(i = 1, 2, \ldots m)$. We denote a_{ij} is the man days needed for Requirement R_j in team G_i.

We can consider the development process in team G_i for requirement R_j as one individual job.

Because for most of the time, one team does not need to develop all the requirements and one

requirement normally only need a few teams to develop, correspondingly, we can find lots of a_{ij} are zero, which means team G_i is not involved in the development of requirement R_j. So, we can simplify the model by only considering the jobs with positive man days.

Let us define a set $X = (1, 2, \ldots, k)$ of all the jobs with positive development time and there are at most k ($k \le m \times n$) jobs in the set.

Because each job belongs to only one requirement, use this attribute, we can partition the set X into n disjoint subsets $\{R_1 \quad R_2 \quad \cdots \quad R_n\}$ where $R_j = \{k \mid$ job k is for requirement $R_j\}$, $(j = 1, 2, \ldots n)$. So, now we consider a requirement is a set of jobs to do in different teams.

And we can get that $\bigcup_j R_j = X$.

Similarly, one job only belongs to one team, so we can partition the set X into m disjoint subsets $\{G_1 \quad G_2 \quad \cdots \quad G_m\}$ where $G_i = \{k \mid$ job k is in team $G_i\}$ $(i = 1, 2, \ldots m)$. We consider a team is a set of jobs to do for different requirements. And we also can get that $\bigcup_i G_i = X$.

Assuming the number of developers in team G_i is Q_i, we can determine the development time d_k is equal to $\frac{a_{ij}}{Q_i}$ for job k where job $k \in R_j \bigcap G_i$.

5.2.4 Precedence constraints

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

The precedence constraints are set between two requirements, not between tasks. According to our former definition, the precedence constraint $R_j \prec R_{j'}$ is considered as precedence constraint between two sets of jobs R_j and $R_{j'}$. We can model them in the following way.

With each job k is associated with a set $\Gamma_k^{-1} \subseteq X'/\{k\}$ of immediate successors: jobs that can only start after the completion of job k.

We set the precedence constraint one after another, If $R_j \prec R_{j'}$,

For $\forall k \in \mathrm{R}_j$, $\Gamma_k^{-1} = \left\{\Gamma_k^{-1} \bigcup \mathrm{R}_{j'}\right\}$.

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

We can define a set $A = \left\{(R_j, R_{j'}) \middle| R_j \prec R_{j'}\right\}$ which contains all the precedence constraints. After we set all the precedence constraints, if $k \in X \Big/ \bigcup_{(R_j, R_{j'}) \in A} \mathrm{R}_j$, it means job k does not have any successor, then we set $\Gamma_k^{-1} = \{END\}$. Or if $k \in X \Big/ \bigcup_{(R_j, R_{j'}) \in A} \mathrm{R}_{j'}$ it means job k does not have any predecessor then we set $\Gamma_{start}^{-1} = \left\{ k \middle| k \in X \Big/ \bigcup_{(R_j, R_{j'}) \in A} \mathrm{R}_{j'} \right\}$.

The precedence constraints can be represented by a directed acyclic graph $G = (X', H)$ where

$H = \left\{(k, l) \middle| k \in X', l \in \Gamma_k^{-1}\right\}$

In this graph, the nodes are the jobs and the directed edges show the precedence constraint between jobs. This graph G is different from the graph we presented in section 5.1.3, because this graph shows the relationships within jobs while the chart in section shows the relationships between requirements.

5.2.5 The upper bound

Let $T_{\max}$ be the upper bound of the completion time. We can set the upper bound as $\sum_{j=1}^{n} \max(d_k \middle| k \in \mathrm{R}_j)$. It happens if we process requirement one by one.

5.2.6 The time window

For each job k, we can compute es_k (earliest start) ls_k (latest start) as its time window. Before we compute the time interval, we can first topological sort the jobs, so that job j is before job k in the order if $(j,k) \in H$.

Compute es_k:

1) Set the earliest start $es_{START} = 0$.

2) Use critical path algorithm (forward recursion) to compute the es_k for the rest jobs.

Critical path algorithm (forward recursion):

1. $es_k = \max_{(j,k)\in H} \left(es_j + d_j \right)$
2. Perform 1 from the *START* to the *END* according to the topological order of the jobs.

Compute ls_k:

1) Set $ls_{END} = T_{\max}$.

2) Use critical path (backward recursion) to compute the ls_k.

Critical path (backward recursion):

1. $ls_j = \min_{(j,k)\in H} \left(ls_k - d_j \right)$
2. Perform 1 from the *END* to the *START* according to the topological order of the jobs.

5.2.7 The (0,1) integer programming model

Let ξ_{kt} be a (0-1) binary variable that is equal to 1 if and only if activity k starts at the beginning of period t. We can formulate the problem as:

$$\min \sum_{t=es_{END}}^{t=ls_{END}} t\xi_{ENDt} \qquad (5.1)$$

Subject to:

$$\sum_{t=es_k}^{t=ls_k} \xi_{kt} = 1 \ , \ k \in X \tag{5.2}$$

$$\sum_{t=es_k}^{t=ls_k} t \cdot \xi_{kt} + d_k \le \sum_{t=es_{k'}}^{t=ls_{k'}} t \cdot \xi_{k't} \quad \text{for} \quad (k,k') \in H \tag{5.3}$$

$$\sum_{k \in G_i} \sum_{\tau=\sigma(t,k)}^{t} \xi_{k\tau} \le 1$$

$$t = (0,1,\ldots T_{\max}), \quad \sigma(t,k) = \max(0, t - d_k + 1)$$

$$i = 1,\ldots,m \tag{5.4}$$

The (5.2) means one job must be selected once. It also shows that each job has the same priority and there is no preemption between them.

The (5.3) is the precedence constraint—one requirement can only start after its predecessor is finished.

The (5.4) means a development team can only develop at most one job at one time.

6. Select and Schedule the requirements

6.1 Introduction

Given a fixed release date T, we know the available amount of working days $d(T)$ within the period, and if during this period the number of developers in team G_i is fixed to Q_i, we can compute the available team capability equals $Q_i d(T)$ man days in team G_i. We set this figure as the capability constraint in our knapsack model. However, using the method of resource constrained project scheduling problem (RCPSP), it is possible that the project needs more working days than $d(T)$ and finishes after the release deadline T. If this happens, we will face a new problem: namely how to modify the original plan?

It is not difficult to think of using the knapsack model for selection and RCPSP model for scheduling iteratively until a good solution is found. In fact, this method is used in most of the software engineering methods. However, doing it iteratively is not only difficult but also time-consuming.

To use the two models iteratively, we need to repeat the 3 steps until a satisfied solution is found:

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

Because RCPSP problem is NP-Hard, it is difficult to find a fast solution to determine which requirements to drop in order to make the deadline. More importantly, if we drop some requirements to fit the project plan, we actually will free some capabilities. It is very wasteful to ignore these free capabilities because there are still large piles of requirements waiting to be developed in our repository. So we need to re-fill in some requirements using the knapsack model. Then we need the RCPSP model to schedule them again to see whether they fit. The problems of doing it iteratively are: first of all, this searching method is difficult to find, and secondly, even if we find one, the knapsack model and RCPSP model are both NP-hard, which means we need to spend lots of time on solving them. A better method is demanded to solve this problem.

This results in the following research question: is it possible to find a method to select and schedule requirements at the same time so that we can define a profitable and practical release plan? In this release plan, we still want to have the maximal revenue, but we also want the project to finish before the fixed given deadline. To achieve these two goals, this model should not only be able to include functional and value-related requirement dependencies, but also include the

precedence constraints. In the following section, we will present the model to select and schedule the requirements when a fixed project deadline is given.

6.2 The integer linear programming model

6.2.1 Problem description

The first step is to mathematically define the requirement, the team, the job, etc. These definitions are exactly the same as what stated in section 5.2.3. For the sake of conciseness, we will not repeat the definitions here. Please refer to section 5.2.3 for details.

6.2.2 Precedence constraints

The precedence constraints are set between two requirements, not between tasks. According to our former definition, the precedence constraint $R_j \prec R_{j'}$ is considered as precedence constraint between two sets of jobs R_j and $R_{j'}$. We can model them in the following way.

We can define a set $A = \left\{ (R_j, R_{j'}) \middle| R_j \prec R_{j'} \right\}$ which contains all the precedence constraints.

With each job k is associated with a set $\Gamma_k^{-1} \subseteq X / \{k\}$ of immediate successors: jobs that can only start after the completion of job k.

We set the precedence constraint one after another, If $R_j \prec R_{j'}$,

For $\forall k \in \mathrm{R}_j$, $\Gamma_k^{-1} = \left\{ \Gamma_k^{-1} \bigcup \mathrm{R}_{j'} \right\}$.

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

The precedence constraints can be represented by a directed acyclic graph $G = (X', H)$ where $H = \left\{ (k, l) \middle| k \in X', l \in \Gamma_k^{-1} \right\}$

6.2.3 Compute the earliest start and the latest start

For each job k, there is a time window (es_k , ls_k) associated to it. This time window defines the possible time interval for this job to start. We can compute the time window in the following way:

(2) Compute es_k:

- Set the earliest start $es_k = 0$ for all the jobs k which do not have predecessor.
- Use critical path algorithm to compute the es_k for the rest jobs.

  Critical path algorithm:

  1. Give a topological sort of all the jobs, so that if $(j,k) \in H$, then j appears before k in the order.
  2. $es_k = \max_{(j,k)\in H} \left(es_j + d_j \right)$
  3. Perform 2 from the start to the end according to the topological order of the jobs.

(3) Compute ls_k.

- For each job k, ls_k is equal to $d(T) - d_k$. We can not lower this upper bound because we do not know whether its successor will be selected or not.

(4) If $ls_k < es_k$, which means this job k can not fit in the time span of the project, and the requirement R_j which contains this job will not be the candidate of the coming release, so $X'' = X / \mathrm{R}_j$.

6.2.4 The Objective function & the constraints

6.2.4.1 Define the variables:

- For each requirement $\mathrm{R}_j \subset X''$, we define a binary decision variable x_j associated to it, $x_j = 1$ if and only if requirement R_j is selected.
- For each job $k \in X''$, there is a binary decision variable y_k associated to it. $y_k = 1$ if and only if job k is selected in the new release.

- For each job $k \in X''$, we define a group of binary decision variable ξ_{kt} where $t \in [es_k, ls_k]$. $\xi_{kt} = 1$ if and only if job k starts at time t.

6.2.4.2 The objective function

We can model it as follow:

$$\max \sum_{j=1}^{n} v_j x_j \qquad (6.1)$$

Subject to

$$x_j \le \frac{\sum_{k \in R_j} y_k}{m_j} \quad \text{for} \quad j = 1, \ldots, n \qquad (6.2)$$

$$\sum_{t=es_k}^{t=ls_k} \xi_{kt} = y_k \quad \text{for} \quad k \in X' \qquad (6.3)$$

$$x_{j'} \le x_j \quad \text{for} \quad (j, j') \in A \qquad (6.4)$$

$$\sum_{t=es_k}^{t=ls_k} t \cdot \xi_{kt} + d_k \le \sum_{t=es_{k'}}^{t=ls_{k'}} t \cdot \xi_{k't} + (1 - y_{k'}) \cdot d(T)$$

$$\text{for} \quad (k, k') \in H \qquad (6.5)$$

$$\sum_{k \in G_i} \sum_{\tau=\sigma(t,k)}^{t} \xi_{k\tau} \le 1$$

$$\sigma(t,k) = \max(0, t - d_k + 1) \quad \text{for} \quad t = (0, 1, \ldots T_{\max}),$$

$$i = 1, \ldots, m \qquad (6.6)$$

6.2.5 The explanation of the model

(6.1) is the objective function, we want to maximize the revenue of the requirement in the time span. v_j is the revenue of a requirement, and x_j is a binary selection variable of that requirement.

(6.2) means that a requirement is only selected when all the sub-jobs in related teams are also selected. In the formula, y_k is a binary selection variable for the jobs k for requirement R_j. Here m_j is the number of jobs for requirement R_j, which is a constant. Please note that $m_j \leq m$ because we do not count the jobs with no development time.

(6.3) Means a job is only selected when it is planned. ξ_{kt} is a binary selection variable, which equals 1 when the job k start at time t.
(6.4) and (6.5) deals with precedence constraint. (6.4) means a requirement is only selected when its predecessor is selected. (6.5) means the jobs for the successor requirement can only start after all the jobs for its precedent requirements finished. In this constraint, $d(T)$ is the number of available working days in the release plan project.
(6.6) is the resource constraint that one team is only able to develop one requirement at one time.

6.2.6 Transformation:

- If we ignore the precedence constraints (4) and (5), it is another way to represent the multi-dimensional Knapsack problem which we used to solve the requirement selection problem.

- If we ignore the resource constraint (6), the method will turn to be a normal project plan problem without specific team capacities. Using Gantt Chart or Network Chart, we can solve it in Polynomial-time.

6.2.7 Requirement dependencies:

In this model, we introduced a new group of variables which deal with the time issues. These variables provide us the opportunities to include the time-related requirement dependencies. In the knapsack model, we have introduced five types of requirement dependencies: 1) Implication, 2) Combination, 3) Exclusion, 4) Revenue-based and 5) Cost-based[50]. These five types of requirement dependencies have been modeled in former chapter using the knapsack model.

Besides the functional and revenue-related requirement dependencies, there are also time-related requirement dependencies: we have to implement requirement R_j before requirement $R_{j'}$ [21]. The standalone time-related interdependencies draw little attention when compose the release plan, however, this dependencies usually come together with other dependencies like *Implication* and *cost-based*[51]. For example, if we need requirement R_j to implement requirement $R_{j'}$, we

probably need to implement R_j before $R_{j'}$. Similarly, if R_j influence the implementation cost of requirement $R_{j'}$, we probably also need to implement R_j before requirement $R_{j'}$. We can conclude that although the time-related dependency does not come alone, it is associated with the *implication* and *cost-based* dependencies.

The time-related dependency expresses more process knowledge rather than product knowledge. To fit the pressure on time-to-market, considering the time-related dependencies can help product managers to deal with the project plan issues the same time as they select the requirements. This model may sacrifice some revenue to fit the more strict constraints, but on return, the selection result will be more practical to fit the release date, and a project plan for the coming release will be made simultaneously.

The Implication, combination, exclusion, revenue-based dependencies are same in the knapsack model, please refer to section 2.7 for details.

cost based

In this model, we assume the development time for a certain requirement in a certain team is a deterministic figure which equals the expected man days divided by the number of developers in the team. The cost based requirement dependencies will change this assumption because then the development time d_k for job k is not deterministic but is influenced by other requirements. This will turn the model into a non-linear one. To restrict the model in a linear way, we need to model it differently.

If requirement R_j influence the implantation cost of requirement $R_{j'}$, after implement R_j, the development cost of the jobs k' ($k' \in \mathrm{R}_{j'}$) for requirement $R_{j'}$ will change from $d_{k'}$ to $d_{k''}$ man days. So we can virtually define a new requirement called $R_{j''}$, and this requirement is a copy $R_{j'}$ only that this development cost has been influenced by requirement R_j, and the durations of the jobs k' ($k' \in \mathrm{R}_{j'}$) change from $d_{k'}$ to $d_{k''}$. We can define these jobs as a group of new jobs called k''. So the newly created requirement $R_{j''}$ has the same expected revenue as $R_{j'}$, and the job k' and k'' belongs to the same team. Only the durations of the two tasks are different.

For the newly created requirement $R_{j''}$, there is a selection variable $x_{j''}$ associated to it, and for each jobs k'' in $R_{j''}$, there are a selection variable $x_{k''}$ and the time variables $\xi_{k''t}$ associated

to it.

We can now analyze the relationship among R_j, $R_{j'}$ and the virtually created requirement $R_{j''}$.

a) If we want to obtain the cost benefit, i.e. to have requirement $R_{j''}$ we must have requirement R_j selected first.

b) If we have selected requirement R_j, then we can not select requirement $R_{j'}$ any more, because the requirement R_j will change the development cost of $R_{j'}$, and actually turn $R_{j'}$ to $R_{j''}$.

c) It is obvious that it is not possible to select both $R_{j'}$ and $R_{j''}$, because $R_{j''}$ is not a real requirement, but just another version of $R_{j'}$ which shows the influence of the cost-related dependency between R_j.

.

It can be seen clearly from the following truth table:

R_j	$R_{j'}$	$R_{j''}$	**T/F**	Explanation
0	0	0	**T**	It is possible to select neither R_j nor $R_{j'}$
0	0	1	**F**	Not possible, can not obtain the cost influence without selecting R_j.
0	1	0	**T**	It is possible to select only $R_{j'}$.
0	1	1	**F**	Not possible, can not select both $R_{j'}$ and $R_{j''}$.
1	0	0	**T**	Possible. We can just select R_j.
1	0	1	**T**	Possible, when we selected R_j, then we can get the cost influence on $R_{j'}$ so as to select $R_{j''}$
1	1	0	**F**	Not possible. When we selected R_j, we can not ignore the cost influence on $R_{j'}$.

1	1	1	**F**	Not possible, we can not select both $R_{j'}$ and $R_{j''}$.

From the relations we analyzed among R_j, $R_{j'}$ $R_{j''}$ and from the truth table we can get that:

1. Requirement $R_{j''}$ has *implication* dependency on requirement R_j .So, $x_{j''} \le x_j$.
2. Requirement R_j has *exclusion* dependency on requirement $R_{j'}$. So, $x_j + x_{j'} \le 1$
3. Requirement $R_{j'}$ has *exclusion* dependency on requirement $R_{j''}$. So, $x_{j'} + x_{j''} \le 1$.

Based on the above result, we can model the *cost-related* requirement dependency by creating a virtual requirement $R_{j''}$ and adding three new constraints 1) $x_{j''} \le x_j$, 2) $x_j + x_{j'} \le 1$, 3) $x_{j'} + x_{j''} \le 1$ in the model.

To model this dependency, we have created a virtual requirement $R_{j''}$. This has created a problem -which one should we use if we need to model dependencies between $R_{j'}$ and other requirements, if we do not know whether $R_{j''}$ or $R_{j'}$ is actually selected. We can define a new variable $x_{j'j''}$ where $x_{j'j''} = x_{j'} + x_{j''}$. We can use this variable to model the dependencies between requirement $R_{j'}$ and other requirements. For example, if requirement $R_{j'}$ *exclusion* requirement R_m then we can set $x_{j'j''} + x_m \le 1$.

time-related

The time-related dependency always come along with implication or cost-based. If we need to implement requirement R_j before requirement $R_{j'}$, we can set a time-related dependency between them, denoted as $R_j \prec R_{j'}$.

With each job k is associated with a set $\Gamma_k^{-1} \subseteq X'/\{k\}$ of immediate successors: jobs that can only start after the completion of job k. If $R_j \prec R_{j'}$, we need to set:

For $\forall k \in R_j$, $\Gamma_k^{-1} = \{\Gamma_k^{-1} \bigcup R_{j'}\}$.

In this way, we set all the jobs for requirement $R_{j'}$ as the successors of the jobs for requirement R_j so that requirement $R_{j'}$ can only start after all the jobs for requirement R_j is done.

The precedence constraints can be represented as the edges in a directed acyclic graph $G = (X', H)$ where $H = \{(k,l) \mid k \in X', l \in \Gamma_k^{-1}\}$.

At last, we add the constraints:

$$\sum_{t=es_k}^{t=ls_k} t \cdot \xi_{kt} + d_k \le \sum_{t=es_{k'}}^{t=ls_{k'}} t \cdot \xi_{k't} + (1 - y_{k'}) \cdot D \qquad (k,k') \in H$$

to the model.

Please note that the *time-related* dependency can not work alone in the model. It has to associate with either *Implication* or *cost-based*. As we modeled the *cost-based* dependency as one *implication* plus two *exclusions* dependencies (see the section above), we need to associate the time-related dependency on the *implication* relationship, i.e. between requirement $R_{j'}$ and R_j.

6.3 The different time availability for different teams

So far, we considered all the teams are available for the whole project period. However, sometimes a team G_i is only available for a certain interval $[lb_i, ub_i]$ where lb_i and ub_i are the lower bound and upper bound of the time interval. For the new release project, which lasts from day 0 until day $d(T)$, we assume, without loss of generality, that the time interval for each G_i $(i = 1, 2, \ldots m)$, $[lb_i, ub_i] \subseteq [0, d(T)]$.

When a team can not work full time on the project, it not only reduces its capacity on the project, but also influences the schedule of other teams. It pops up a synchronization problem among different groups and also changes the time interval $[es_k, ls_k]$ for a job. In the following chapter, we will show how to calculate the new time interval $[es_k, ls_k]$ for the all the jobs.

The earliest start es_k

A team G_i is available for the project from the time lb_i on. We can create a virtual requirement $R_s = (lb_1, lb_2, \ldots, lb_m)$ as the start of the project. This requirement contains the jobs $s1$, $s2 \ldots sm$ where $d_{si} = lb_i$ $(i = 1, 2, \ldots m)$. As the start of the project, R_s is the predecessor of all the requirements R_j which does not have a predecessor. Let $A = \{(R_j, R_{j'}) | R_j \prec R_{j'}\}$ be the set which contains all the precedence constraints. We can create a new set $A' = A \bigcup \left\{ (R_s, R_{j'}) \middle| R_i \subseteq \left(X \Big/ \bigcup_{(R_j, R_{j'}) \in A} \mathrm{R}_{j'} \right) \right\}$, so that it also contains the precedence constraints between R_s and R_j.

Instead of using A, we now use the set A' to construct the set H in section 5.3. Because the virtual requirement R_s is the predecessor of all the requirement, all the earliest starts of the jobs $es_{si}, (i = 1, 2, \ldots m)$ equal zero. In this way, this virtual requirement takes up the time interval $[0, lb_i - 1]$ when the team is not available for the project. Please note that it is mandatory to

select the virtual requirement R_s.

The latest start ls_k

The latest start of job k is determined by the available time of the team where this job belongs. So the latest start of job k equals $ub_i - d_k$, where $k \in G_i$. This day is the time when team G_i has just enough time to complete job k before its last available day.

In the same way as we discussed before: if $ls_k < es_k$, it means this job k can not fit in the time span of the project, and the requirement R_j which contains this job will not be the candidate of the coming release, so $X'' = X / R_j$.

6.4 Model the holiday seasons

Sometimes, a development team is temporally unavailable while other teams are still working, for example, one development team needs to work on another project for a while or simply because of the holidays. This model is especially useful for international companies, since the holidays in each country are significantly different.

Based on our assumption before, if a team goes on holiday before finishes the job at hand, this team will continue to develop this job until it is complete. The holidays can influence in two fields:
First, if holidays interrupt a job, the completion time of this job will be delayed and it will also influence the start time of its successors (if there is any).
Second, if a team is on holiday, obviously, the team capacity is zero during this period. It is neither possible to proceed a job nor to start a new one.

Without losing generality, we assume the holidays are in the team's available time. i.e.

$$lb_i \le Hs_i \le He_i \le ub_i . \qquad (6.7)$$

6.4.1 The model

If we want to include the holiday period in a team, we need to set the following constraint:

$$\sum_{k \in G_i} \sum_{t=Hs_i}^{He_i} \xi_{kt} = 0 \qquad (6.8)$$

We also need to modify the development duration of job k from d_k to d'_k:

$$d'_k = d_k + (He_i - Hs_i) \sum_{\rho=\pi(i,k)}^{Hs_i-1} \xi_{k\rho}$$

$$k \in G_i \quad \text{and} \quad \pi(i,k) = \max(0, Hs_i - d_k + 1) \qquad (6.9)$$

We need to change d_k in constraint (5) to d'_k.

In the resource constraint (6), we need to modify the model to:

$$\sum_{k \in G_i} \sum_{\tau=\sigma(t,k)}^{t} \xi_{k\tau} \le 1$$

$$t = (0,1,\ldots Hs_i), \quad \sigma(t,k) = \max(0, t - d_k + 1)$$

$$i = 1,\ldots,m \qquad (6.10A)$$

$$\sum_{k \in G_i} \sum_{\tau=\varphi(t,k)}^{t} \xi_{k\tau} \le 1$$

$$t = (He_i,\ldots d(T)),$$

$$\varphi(t,k) = \max(0, t - d_k - (He_i - Hs_i) + 1) \qquad \text{if} \quad t - d_k \le He_i$$

$$\varphi(t,k) = \max(0, t - d_k + 1) \qquad \text{if} \quad t - d_k > He_i$$

$$i = 1,\ldots,m \qquad (6.10B)$$

The holiday will also influence the latest start of a job. The latest start ls_k of job k equals:

$$d(T) - d_k \qquad \text{if} \quad d(T) - d_k \ge He_i$$

$$d(T) - d_k - (He_i - Hs_i) \qquad \text{if} \quad d(T) - d_k \le He_i$$

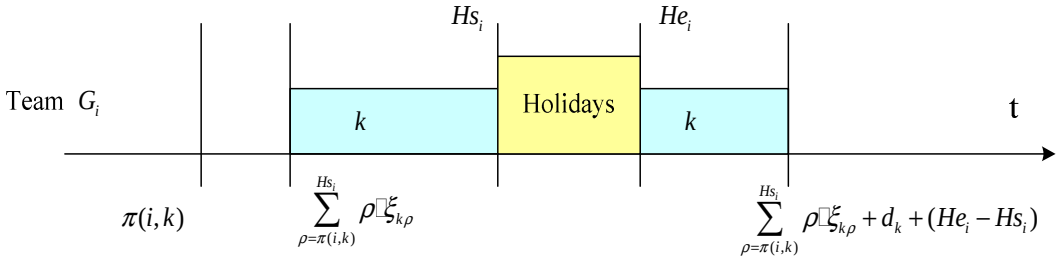
If $ls_k < es_k$, it means this job k can not fit in the time span of the project, the requirement R_j

which contains this job k will not be the candidate of the coming release, so $X'' = X / R_j$.

6.4.2 Explanations of the constraints

Constraint (6.8) means the team G_i can not start to develop a new requirement when this team is on holiday. In this constraint, Hs_i is the time when team G_i starts the holidays and He_i is the end of the holidays

Constraint (6.9) deals with the jobs in the team G_i if they are interrupted by the holidays. If job k does not finish before the holiday starts, the team should continue its development job after the holidays. If we count the holiday time in, the team needs to spend more time on job k and it will influence the start time of its successors (if there is any).
The following picture depicts such situation:



If the team G_i starts to develop job k after the time $\pi(i,k)$, it will not be able to finish its development job before the holiday starts. We can use $\sum_{\rho=\pi(i,k)}^{Hs_i} \xi_{k\rho}$ to determine whether it is the case. $\xi_{k\rho}$ is a binary decision variable which equals one only if the development job k starts at time ρ. The formula $\sum_{\rho=\pi(i,k)}^{Hs_i} \rho \cdot \xi_{k\rho}$ can tell us when the development starts. And it will finish at $\sum_{\rho=\pi(i,k)}^{Hs_i} \rho \cdot \xi_{k\rho} + d_k + (He_i - Hs_i)$. In this case, we can use d'_k which equals : $d'_k = d_k + (He_j - Hs_j) \cdot \sum_{\rho=\pi(i,k)}^{Hs_j} \xi_{k\rho}$ as the development duration for job k instead of d_k

The holidays also influence the resource constraints. Separated by the holiday, we can divide the project into two parts, the one before the holiday and the one after it.

Before the holiday starts, it has no influence on any of the jobs. So, the resource constraint before the holidays (6.10A) remains the same as what we set in the original model.

After the holidays, however, a team also needs to deal with the left jobs before the holiday. At any time t, a team is devoting its time on job k if and only if this job starts less than d_k working days ago. If d_k working days ago is before the holidays start($t - d_k \le He_i$), we need to include the holidays time in. If not ($t - d_k > He_i$) we can just ignore the influence of the holiday.

Please note that we do not set resource constraint within the holiday period. We can ignore it because we have already set constraint (6.8) so that no team can start a new job within the holiday season.

If we can not complete job k after the holidays end, i.e. $d(T) - He_i \le d_k$, we have to start this job somewhere before the holidays to keep the project deadline. So the latest start ls_k of job k equals $d(T) - d_k - (He_i - Hs_i)$ if $d(T) - He_i \le d_k$. The time $d(T) - d_k - (He_i - Hs_i)$ is the time where you have d_k working days left for the project.

In this way, d_k' is still linear to d_k because Hs_i, He_i and $\pi(i,k)$ are constant for each development job k.

7. Dynamic adjustment of the release

Until now, 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 the development phase, the expected working man days can be either overestimated or underestimated. It can also happen that one very important customer places an order after the release is determined, and some of the new features must be added in the coming release. This section will answer how to deal with these changes i.e. how to modify these data, and how to set up a new model.

The following picture depicts a general example of the problem. Team A and Team B were assigned with a couple of jobs to do in the release period. After the project started, re-planning was needed due to over/under-estimations or important new order. Then the product manager needs to decide which jobs to continue and which to drop if necessary.

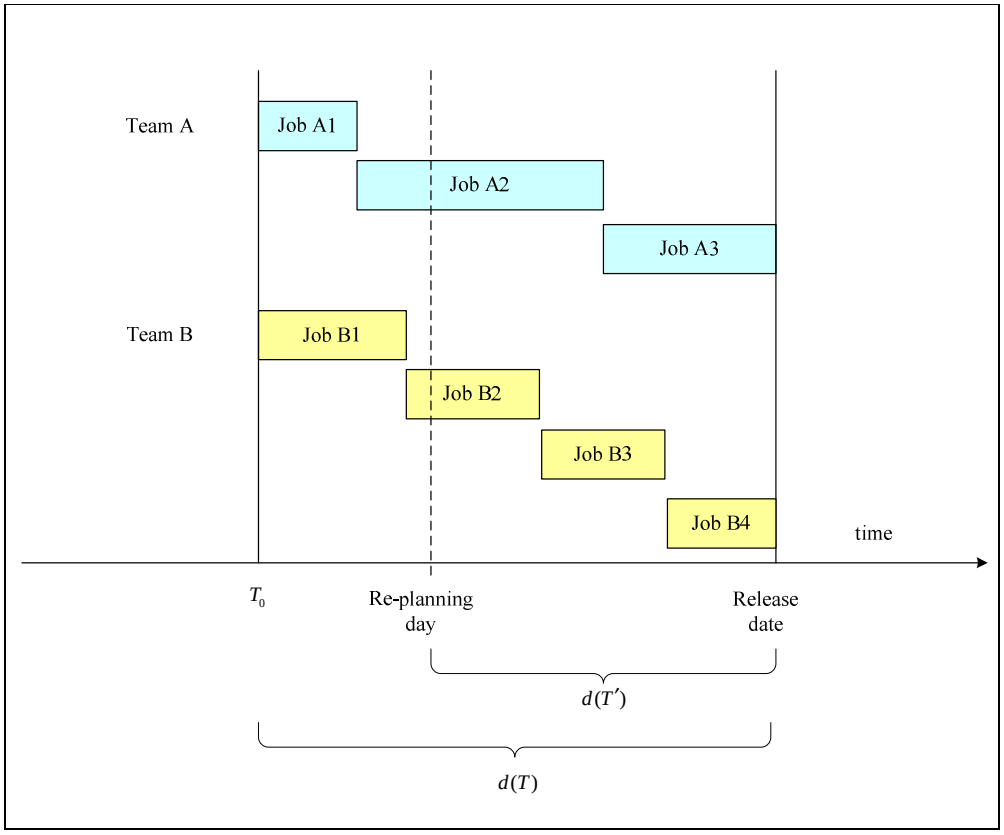


Figure 7.1 an example of release adjustment problem

Change release time

In our model, the planning period is T , and $d(T)$ is the number of working days in the planning

period. If the date of release changes, you need to modify the constant T. Please notice that the new T' will be the date between your release date and your decision date, (the date when you change the T).

Change expected revenue

You can change the expected revenue of any requirement freely, even the ones which you have already developed or under development. For example, if you want to change the expected revenue of requirement R_j from v_j to v_j', it is free to do at any time.

Change man days or dependency

When we need to change the expected man days for a requirement or want to add additional dependencies, we first need to divide the requirements into third groups.

- The first group is contains all finished requirements, for example, the job A1 and B1 in figure 7.1
- The second group contains the requirements currently under development, for example, the job A2, B2 in figure 7.1.
- And the third group contains all the rest requirements. For example, the job A3, B3, B4 in figure 7.1.

First group

The requirements in this group have already been implemented. So it is not necessary to adjust the expected man days for these requirements.. However, it is still meaningful for these requirements to appear in additional dependencies. We just need to add one additional constraints that $x_j = 1$ because these requirements have already been implemented.

Second group

For the second group of requirements, the expected man days should be the remaining days to complete the requirement. R_j. Please note that we do not deduct the revenue of R_j although we have already developed something for this requirement. It is because we believe the expected revenue can only be obtained after the whole development is complete.

Using this method, it is possible to terminate a requirement which is under development at the decision time. There are mainly three reasons why to model in this way:

1. First, from financial[52] point of view, the effort put on the requirement before the decision date is sunk cost. It should not influence the decision.
2. Second, the requirements under development have higher probability to be selected again because we have already deduced a part of the development cost while the revenue remains the same.
3. Third, The ILP model does not guarantee that the requirements currently under development are better than the requirements in the waiting list. It is possible that the requirements in the

waiting list have higher ROI than the ones under development, even when parts of the development cost have already been deducted.

Still, it is a management decision. If a manager decide to continue with the current development in terms of morale or other reasons, it is still possible. Then we just set the decision variable x_j in this group as 1.

If the requirements in this group are re-selected, we would rather continue the development. So when defining the timing variables ξ_{kt} for the jobs in this group, we only define ξ_{k0} instead of within the whole interval. In this way, we guarantee we will continue with the undergoing development if the requirements are re-selected.

Thi**rd group**

The third group constrains all the requirements in the waiting list. The requirement in this group has no difference with the requirements in the repository. There is nothing to worry about when changing the requirement data or the dependencies.

Important new orders

It is possible that a very important customer places an order after the release plan has been made and his order have to be included in the coming release. If it happens, we can set the decision variables x_j for these requirements as 1 and place these requirements in the third group.

Launch

After modifying the data and fixing several variables, put all the requirements in three groups back into the requirement repository. Run the ILP model again, and it will decide which to drop or which to add, and make a new project plan when it is necessary.

8. Relationships between the models

8.1 Structure of the models

The ILP models are not isolated islands. The following use case diagram shows the relationships among different models.

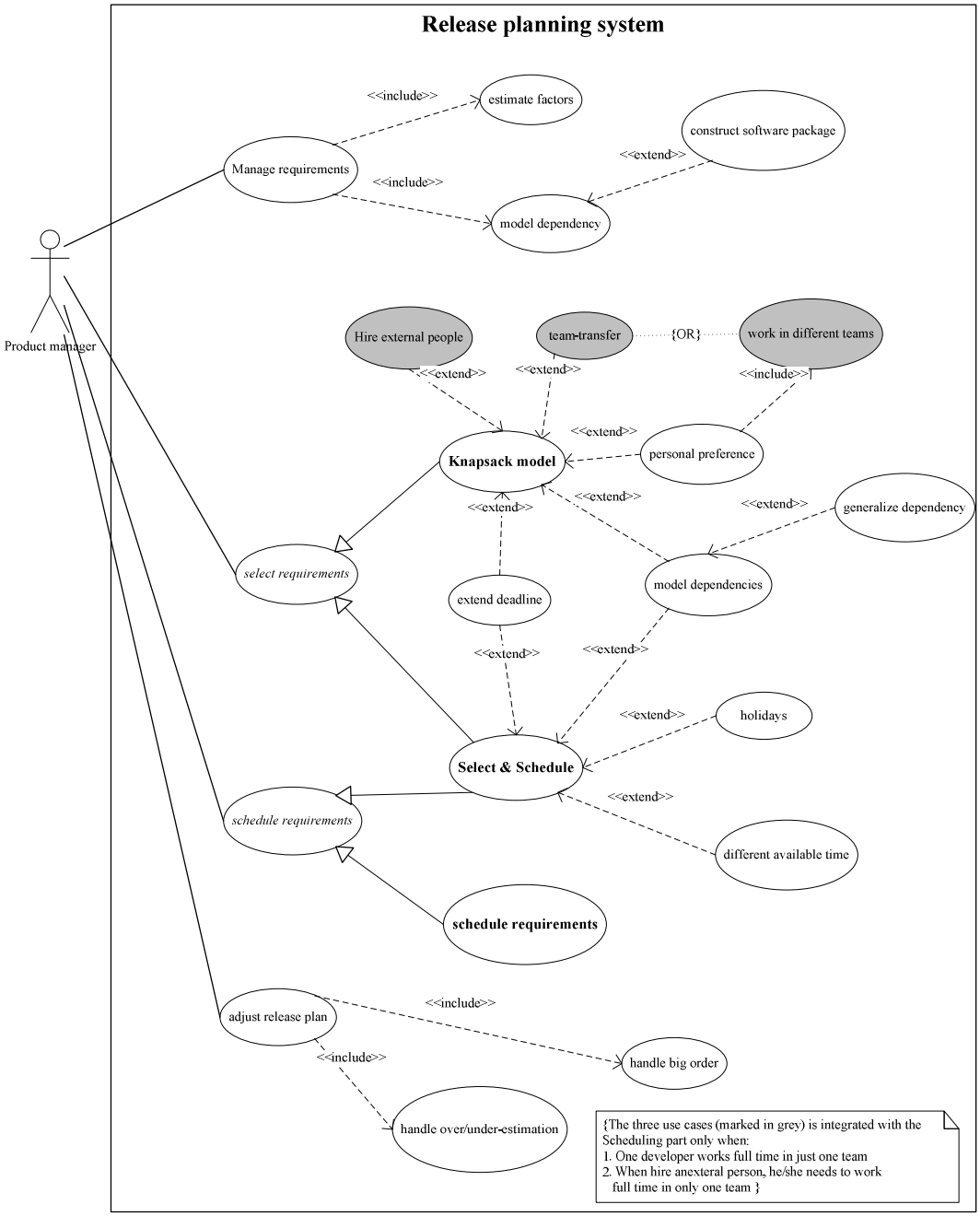


Figure 8.1: the structure of the models

This use case diagram describes the interaction of different models. There are mainly four functions in the system. The “manage requirements” is the part where requirements are issued and the relationships between them are modeled. The second and third parts are ILP models for requirement selection and requirement scheduling. The last part “adjust release plan” is used when we need to adjust the release plan.

Requirement management

In the “manage requirement” parts, the factors for every issued requirements are estimated and the requirement dependencies are set. In general, the factors include, expected man days, expected revenue for each requirement, requirement dependencies, release date, priority, risk, personal preference and so on. The details of these factors are discussed later in the section of “*the factors and process of release planning* “

Requirement selection

The “select requirement” is based on the knapsack model. There are also several management steering scenarios modeled as extensions of the knapsack model. We can “hire external personnel”, “extend deadline”, “model dependencies”, “team transfer”, and “model personal preferences”. The details and explanations of these models are discussed in the section of “*the mathematical models of release planning*”. These extensions can work simultaneously only with one exception: If we want to include personal differences, we have to use a new ILP model “work in different teams” to handle team transfer, therefore the old “team transfer” model is not available any more. This confliction is shown as an “OR” relationships in the use case diagram.

The input of the model is a collection of requirements with estimated factors and relationships, after selecting the scenarios a product manager wants, the output of the model is as a group of requirements for the next release.

.

Requirement scheduling

After the selection of the requirements, the next step is to schedule them. It is also possible to consider them as consecutive processes, i.e. the output of the requirement selection part is the input of the requirement scheduling part. Unfortunately, this connection is not seamless. In the scheduling part, we assume the development time for job k is d_k which equals the expected man days divided by the number of developers in the team, i.e. $d_k = \frac{a_{ij}}{Q_i}$. So the Q_i has to be a constant number for the whole planning time. This adds additional constraints for the requirement selection model:

1. If hire external personnel, he/she will have to work for the whole period in one team.
2. If enable team transfer model, a person can only be transferred for the whole period, i.e. $U_{cap} = d(T)$.

If the above mentioned constraints are satisfied, we can link the two models together with no worries.

It is also possible to only use the scheduling model. In that case, the input of the model is a collection of the requirements as while as the requirement dependencies between them; the output is a project plan with minimal project span.

Select & Schedule

It is also possible to select requirements and schedule them as the same time. In this way, we introduced a new model “Select & Schedule”. This model combines the two processes together so that the out put of the model is a group of requirements for the next release as well as a project plan to implement them. This model has four extensions. Two of them, “extend deadline” and “model dependencies”, are similar with the ones for knapsack model, it also has two more extensions, “holiday seasons” and “different time availability”, specially used for timing issues. All four of the scenarios are compatible with others.

Adjust release plan

After the release plan is set, it is also possible to adjust it due to the changes of external environment. Mostly, there are two reasons for it: first, as the requirements are developed in a dynamic environment, the estimated values of the factors are changing under market conditions. So we need to handle over/under-estimations. Second, when an important customer proposes some orders, it is also necessary to adjust the release plan to make time for these unexpected orders. These two conditions are modeled as two use cases in the diagram. The input of the model is the requirement dataset, as well as the changes of the factors, and the output of the model is a new release plan.

8.2 Processes to use the models

The following picture shows how to use the models to make a software release plan. The first step is to manage the requirements. In this phase, we try to gather requirements from different stakeholders and estimate the values of requirement factors. These factors include business value, cost, dependencies, priority, risk, quality and so on. In the first section, we will introduce several methods to estimate them. These factors will then be used as the input of the optimization models.

Then the next step is to select the right requirement for the release. We introduced the knapsack model and its extensions to compose a release plan so that to achieve the maximal revenue. Based on different company preferences, we have provided several management steering mechanism to improve the profitability, like team transfer and deadline extension. After the selection, we can make a project plan of these requirements using the RCPSP model. Another way is to combine the selection and scheduling processes together so as to find a group of profitable requirements with a suitable implementation plan. Using this method can guarantee that the project will finish on time but will lose some management steering mechanism, like hiring external people. No matter choosing which processes, we can expect to find the best group of requirements for the next release.

When the market condition changes or the original estimation is not very precise, we need to

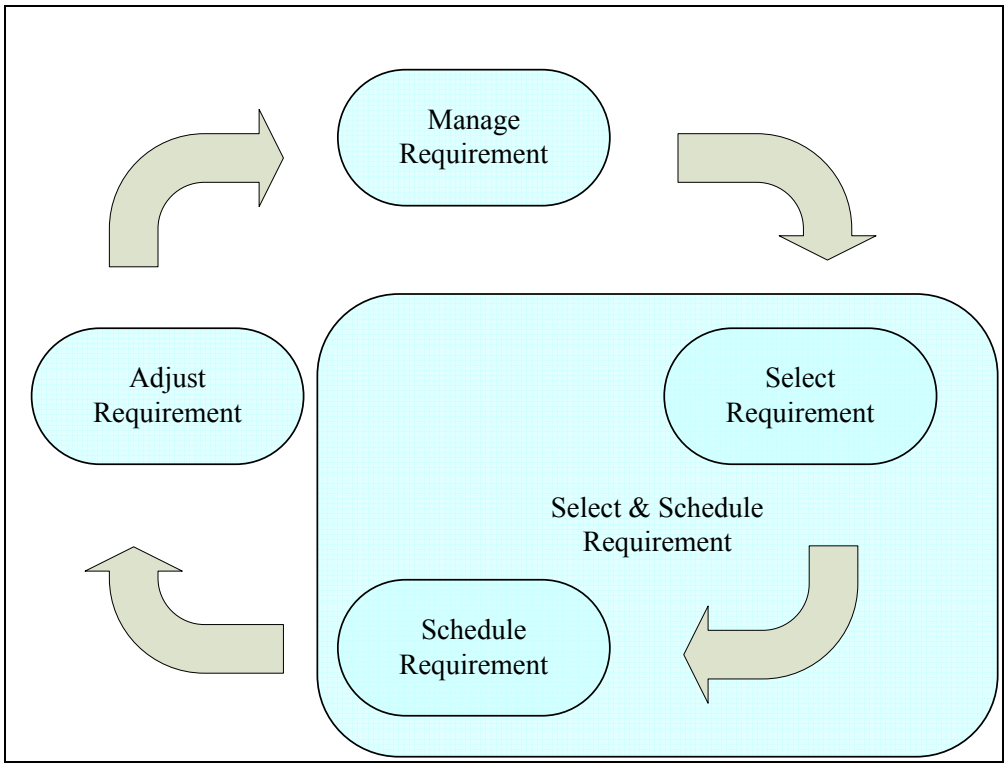


Figure 8.2: the processes to use the models

Adjust the release plan. In this phase, we may need to change the original estimations or even set more figures or relationships for a requirement. It is also possible to receive additional requirements from other stakeholders. These adjustments require us to re-manage the requirements and then re-optimize the selection. These processes will continue iteratively so as to find the best choice for the next release.

8.3 The comparison of the models

The release planning model contains three main ILP models—the knapsack model, the RCPSP model for scheduling and the model to combine these two processes together. Each of them has it advantages and disadvantages.

The knapsack model is good at finding the most profitable solution and has the most management steering mechanism options. These extensions provide additional way to find a profitable solution, and, for most of time, will increase the value of the release plan. The disadvantage however, is that the result of the model might appeared to be bit too optimistic, and this may lead to delays of the project.

The RCPSP model can help us to find a project plan with the minimal project span. This model is very much connected to the knapsack model, because the output of the knapsack model is

normally the input of this model. This model is easy to use and follow, but the function of this model is a bit limited.

The ‘Select & Schedule’ model can help us to find the requirement selection as well as the project plan at the same time. This model focuses on the on time delivery and is very useful when the release deadline is very strict. On the other hand, this model does not have as much extensions as the knapsack model does. So from the functional point of view, this model has less functions then the knapsack model. Without the help of hiring external persons, or team transfer, the result found by this model might be not as reliable as the knapsack model.

Appendix 1: Sets, variables and parameters:

Name	Description	Type	Note
R_j	Requirement j. It is considered as a set which contains all the jobs for this requirement	Set	
G_i	Team i. It is considered as a set which contains all the jobs for in this team.	Set	
Q_i	Number of developers in team G_i.	Parameter	
a_{ij}	Man days for Requirement R_j in team G_i.	Parameter	
k	Job k. We consider the development task for requirement j in team i as an individual job.		We only define a job when $a_{ij} > 0$
d_k	The duration of job k.	Parameter	
X	The set of all the jobs.	Set	
A	The set of all precedence constraints $(R_j \prec R_i)$	Set	
Γ_k^{-1}	The immediate successors of job k	Set	Every job is associated with one set Γ_k^{-1}.
H	A set which contains all the precedence constraints between dual jobs.	Set	
es_k	Earliest start time of job k		Compute using critical path algorithm (forward)
ls_k	Latest start time of job k		Compute using critical path algorithm (backward)
ξ_{kt}	Binary decision variable. Equals 1 if and only if job k starts at time t	Variable	For each job k, ξ_{kt} is defined from es_k to ls_k
Scheduling the requirement			
$T_{\max}$	The maximal time of the project span		
START	A virtual job which is the predecessor of all the jobs in X.	Job	Duration is zero.
END	A virtual job which is the successor of all the jobs in X	Job	Duration is zero.

X'	All the jobs X together with *START* and *END*	Set	
Select & Scheduling requirements			
$d(T)$	The deadline of the new release	Parameter	
v_j	The expected revenue of requirement R_j	Parameter	
x_j	Binary decision variable. Equals 1 if and only if requirement R_j is selected for the release	Variable	Every requirement is associated with one x_j
y_k	Binary decision variable. Equals 1 if and only if job k is selected for the release.	Variable	Every job is associated with one y_k
m_j	The number of jobs for requirement R_j	Parameter	
X''	The set of jobs which are the candidates for the new release	Set	$X'' \subseteq X$
Requirement dependency			
w_{ij}	The additional value from the *revenue-based* dependency between R_i and R_i	Parameter	
x_{ij}	Binary decision variable, equals 1 if and only if we obtain the additional value from the *revenue-based* dependency	Variable	$x_j + x_i - 1 \le x_{ij}$ $x_{ij} \le (x_j + x_i)/2$
R_i'	An artificial requirement. Created to show the cost changes of requirement R_i.	Requirement	Created when requirement R_j influence the cost of requirement R_i
Different time availability			
lb_i	The time when G_i starts to be available. The lower bound of the team's available time interval.	Parameter	$0 \le lb_i \le ub_i \le d(T)$
ub_i	The time when G_i is not available anymore. The upper bound of the team's available time interval.	Parameter	$0 \le lb_i \le ub_i \le d(T)$
Holidays			
Hs_i	The time when holidays start in team G_i.	Parameter	$lb_i \le Hs_i \le He_i \le ub_i$
He_i	The time when holidays end in team G_i.	Parameter	$lb_i \le Hs_i \le He_i \le ub_i$

The tools and the test results

We have developed two JAVA applications to test the ILP models proposed in the former section. The first one is called “scheduler” which can schedule the development activities exactly in time. The second is called “Planner” which can select and schedule the requirements at the same time. In the first chapter, we present the general information and the structures of the two applications. The source code and the detail model are not included in the thesis for the sake of brevity.

Using the tools we developed, we have conducted two tests. The first one is to examine how much requirement dependencies influence the project plan. The second one is to compare the two types of software development processes: i.e. Select→Schedule V.S Select & Schedule. The first type of planning processes is common in most of the software development processes models, like RUP, DSDM, Waterfall, etc. However, in the former chapters, we’ve shown that this model may have some problems, and introduced a new ILP model to combine these processes. These two types of software development processes are compared based on a simulation in chapter 10.

9. The tools

Two tools have been developed to test the models. The first one is called “Scheduler”, which can schedule the activities exactly in time based on the precedence constraints. The second one is called “Planner”, which can select and schedule the requirements at the same time. This tool is based on the linear programming model “schedule the requirement with fixed deadline” and the out put of the result is a collection of requirements for the coming release as well as the project plan to develop them.

9.1 General information

General Information	
Name	Scheduler, Planner
Developers	Chen Li
Platform	Linux
Languages	Java 1.4.2
Lib. Used	SWING, CPLEX 9.0, CSV
Interface	SWING
Input/output document format	CSV file

Table 9.1 the general information of the prototype

Both of the two prototypes are Java applications running in Linux environment. They use three libraries. The SWING is used for interface, the CPLEX is used for solving the integer linear programming model, and the CSV is used to read input document and write output document. The input and output document format is CSV (coma separated value) file. The CSV document can be easily transformed to an excel file.

9.2 Software structure

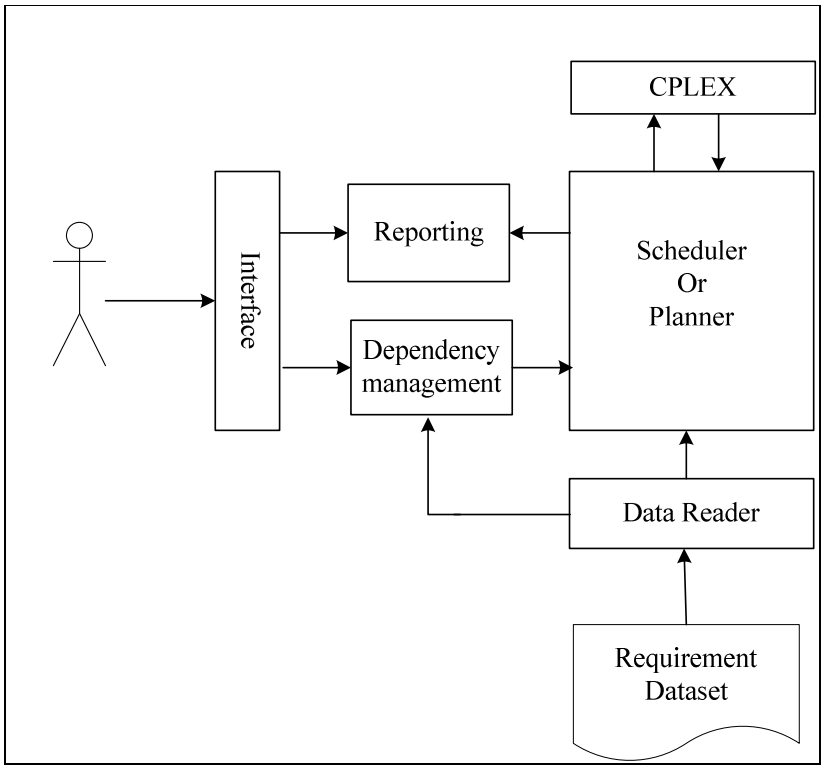


Figure 9.1 The software structure of the prototypes

The user needs to first set requirement dependencies manually or automatically through dependency management. The results then are sent to the Scheduler, which is the heart of the program. The scheduler will model the scheduling problem and then sent to CPLEX. When CPLEX get the results, it will send them to Scheduler and then to the Reporting system. The hard-copy of requirement dataset is stored in requirement dataset in a CSV file, and data reader will read all the information in the dataset.

The key functions of the models are described as follow:

- Interface: Give the program instructions and receive results
- Reporting: to report the result to the interface
- Dependency management: set dependencies among the requirements
- Data Reader: read date from the release planning dataset
- Requirement Dataset: Excel files or CSV files with all the requirement information
- Scheduler or planner: The heart of the program to process data and build up the ILP model
- CPLEX: the ILP library to solve the ILP models

The “Planner” and “Scheduler” are similar in structure. Only the heart of the program is different. The differences of the integer linear programming models are presented in the former chapters(chapter five & six). The following sections will show the software differences in activity

diagrams.

9.3 Screen shots

The following screen shots show how the prototypes look like. These interfaces are designed using Visio, so the final interfaces might look slightly different than what is shown here, but the general structure will be the same.

9.3.1 Dependency management

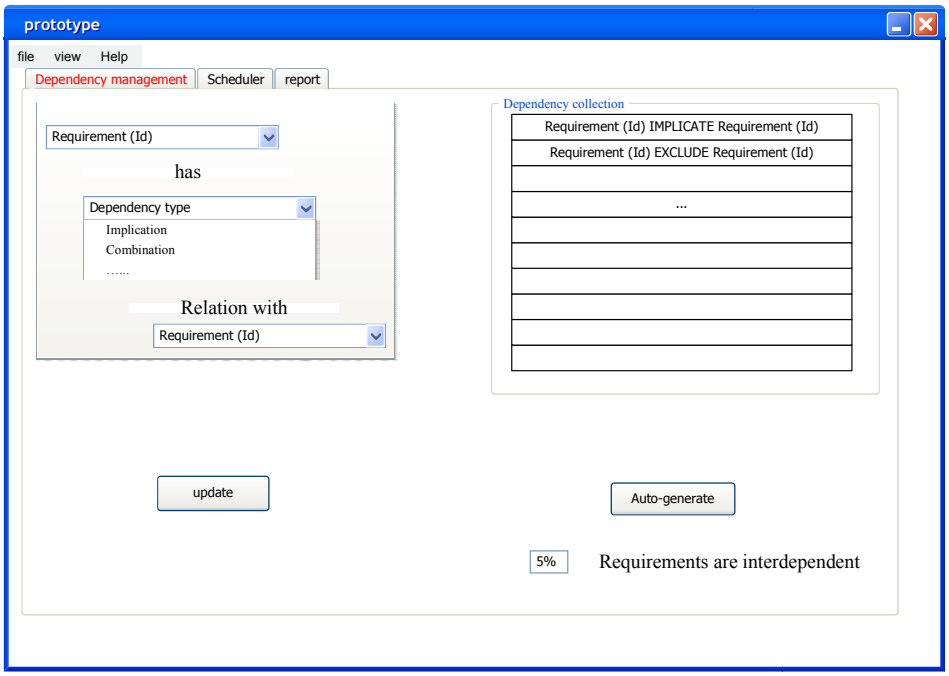


Figure 9.2 the interface of the dependency management

On this interface you can set requirement dependencies manually, or generate automatically. As there is no requirement dependency in the requirement dataset, for testing purpose, we provide the opportunity to automatically generate a certain amount of dependencies, for example, 5% of the theoretical maximal number of dependency. (The detail for automatically generating dependency will come in the next chapter).

9.3.2 Scheduler

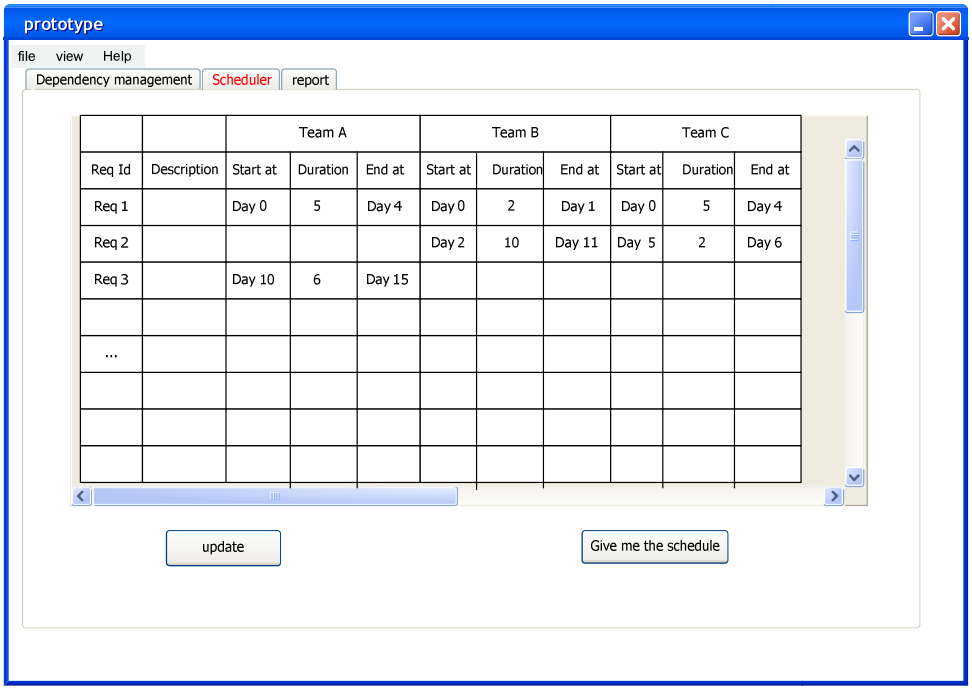


Figure 9.3 the interface for scheduler

The days needed for each requirement on each team comes from the requirement dataset. After set the requirement dependency, and click lunch, the scheduler can present you with the result of starting and finishing date of particular jobs.

9.3.3 Reporting

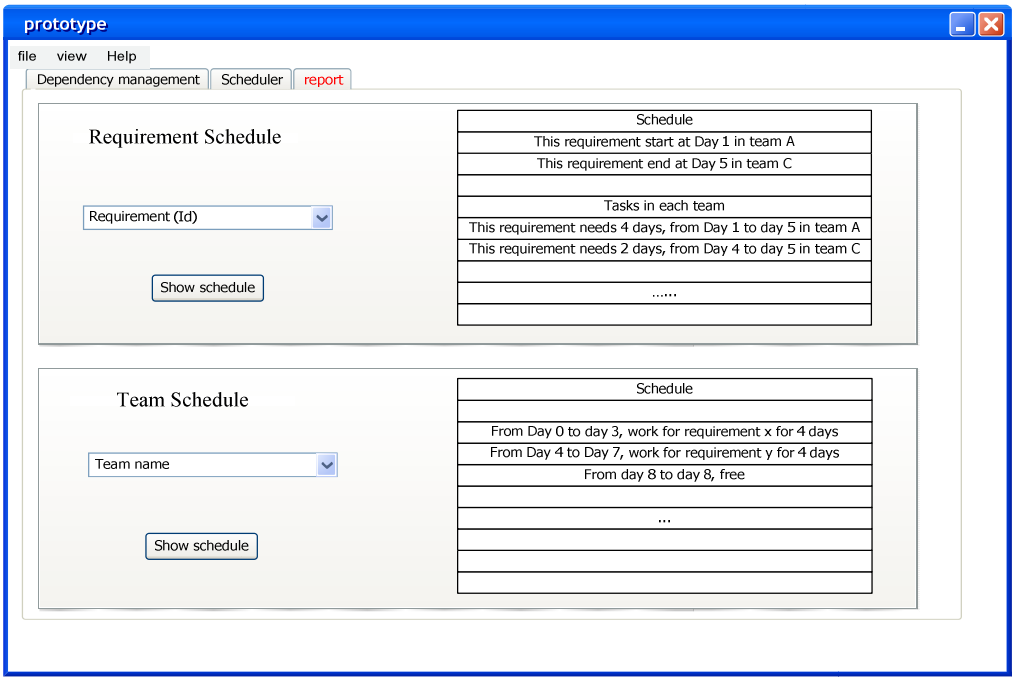


Figure 9.4 The interface for reporting system

This reporting interface details schedule of each requirement and each team. By clicking the requirement id or team name, we can get a clear schedule for each of the requirements.

9.4 The activity diagram of “Scheduler”

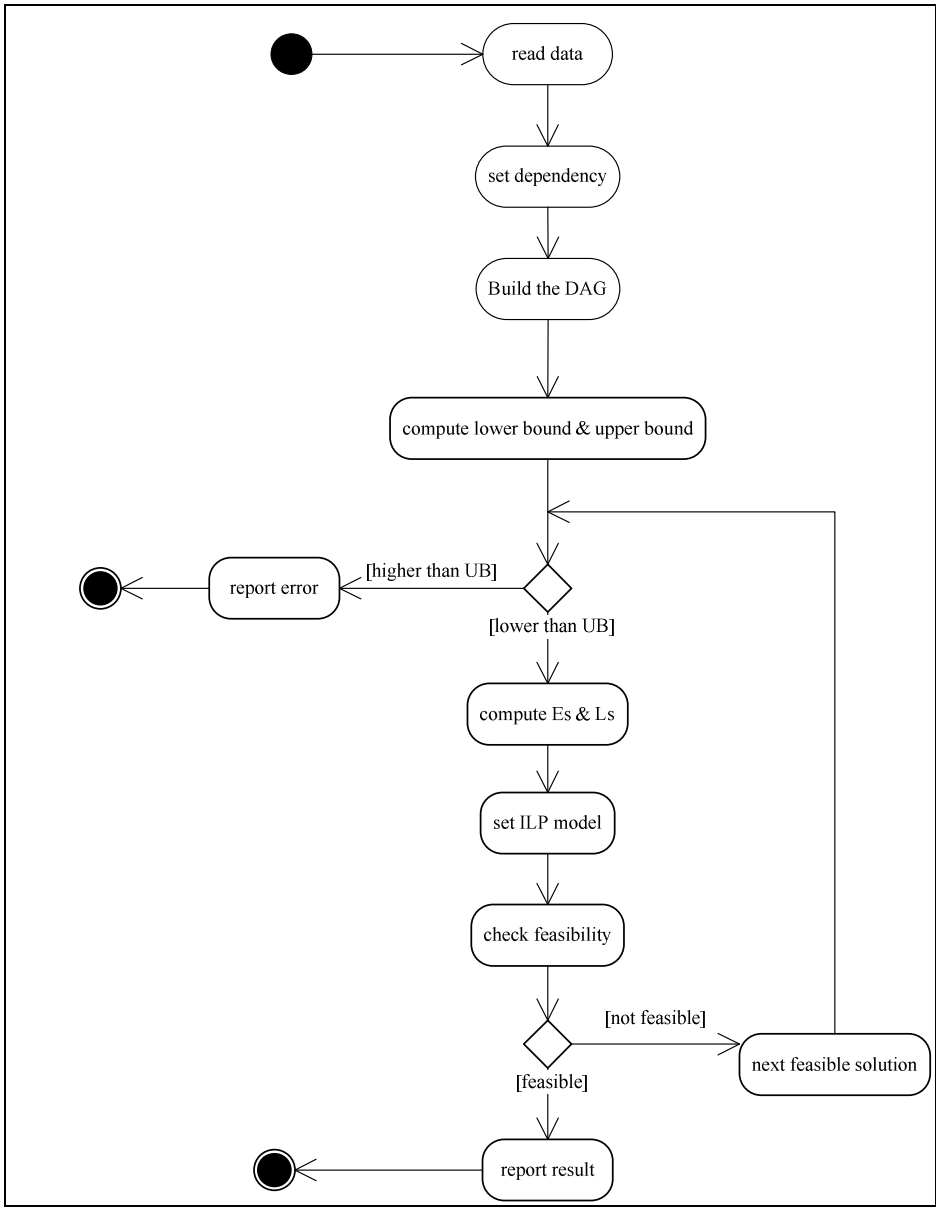


Figure 9.5 The activity diagram of “scheduler”

The function of “Scheduler” is to make a project plan of the selected requirements. So, the input of the model is the requirements selected by the knapsack model, and the output of the model will be the project plan to implement them.

The “Scheduler” first reads the requirement dataset through “data reader”; and set dependencies through “dependency management”. Based on the requirements and the dependencies, it builds up

a Directed Acyclic Graph to compute the lower bound (the maximal value of the critical path and release date) and the upper bound (when requirements are fully dependent and need to develop one after another) of the project span. The “scheduler” then checks the feasibility of every result starting from the lower bound (the larger one of the critical path and the deadline) to the upper bound (when fully dependent. i.e. serial one after another). Every time when checking the feasibility, the model first computes the earliest start and latest start for each job and then builds the linear programming model. When a feasible solution is found by the CPLEX library, the model stops and reports the result. Because the model checks the feasibility of the results from the lower bound to the upper bound, the first feasible result is the result with minimal project span.

9.5 The activity diagram of “Planner”

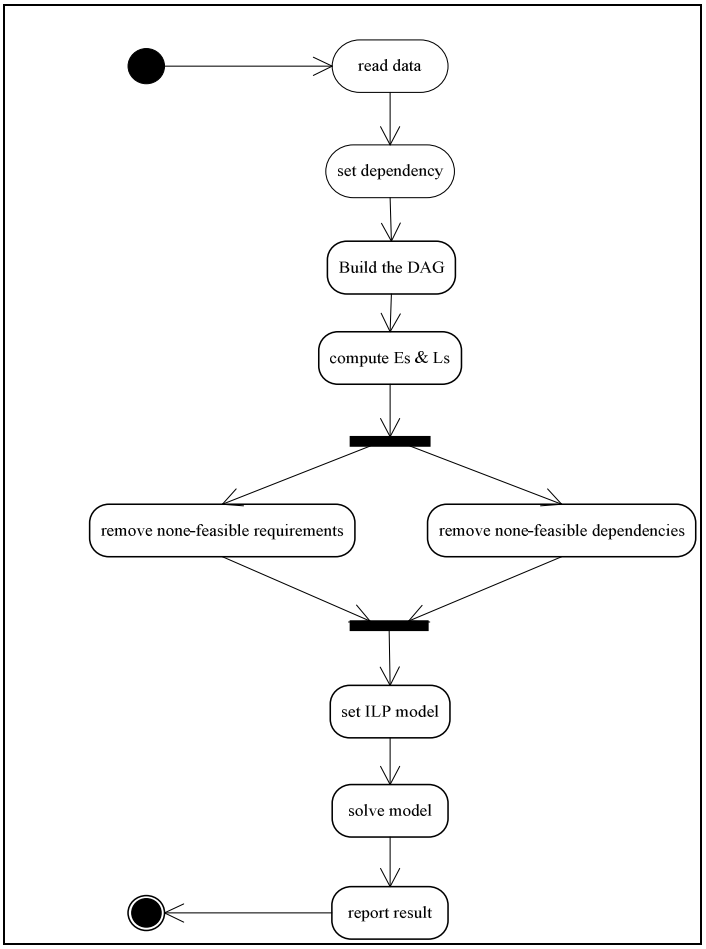


Figure 9.6 The activity diagram of planner

The function of the “planner” is to find the group of most profitable requirements as well as a project plan for implementation. So the input of the model is the requirement dataset with all candidate requirements, and the output is a selection of requirements and their project plan.

The “Planner” starts from reading the requirement data sets through “data reader”. It then gets the

dependencies from “dependency management”. Based on the requirement datasets and the dependencies, the model builds up a Directed Acyclic Graph and computes the earliest start and the latest start of each job. When some jobs are not feasible to be selected, for example, when the earliest start is larger than the latest start, the model can eliminate these none-feasible requirements as well the dependencies among them. The “Planner” then builds the ILP model based on the remaining requirements and dependencies. It is then solved by the CPLEX library and the result is documented to a CSV file when a solution is found.

10. Simulation tests

10.1 Test purpose

The purpose of this test is to answer the following two questions:

1. **What's the relationship between the number of time-related dependencies and the possibility of running out of time in the project planning?**
2. **What are the differences when we select and schedule requirements at the same time, and when we select and schedule sequentially?**

Currently, we consider the release composition problem as a knapsack problem. However, if we consider the precedence constraint (i.e. *time-related* dependency explicitly, and *implication* and *cost-related* implicitly), it is possible that the release composed using the knapsack method results in a schedule that exceeds the deadline. We want to find out how the time-related requirement dependencies influence the project span by answering the first question. When there are more dependencies, we would expect that the project stands a higher chance of being late, and the project span will be longer. We also want to find the difference between the optimal result and the lower bound. The lower bound of the model is the larger one of the maximal team workload or the project span computed using critical path method.

For the second question, we want to find out how the precedence constraints influence the requirement selection. In the knapsack model, we do not consider the time-related requirement dependencies, while the new model does. Given the same set of dependencies, the requirements selected by the original knapsack model are expected to have higher total revenue than the new model, because it does not include the time-related issues. However, there is a possibility that the release date will be delayed. So we will compare how much the project span may differ and how much the revenue may differ.

This comparison is in fact a comparison between the two release planning processes i.e. shall we follow the processes that we first *Select* requirement and then *Schedule* them or shall we combine these two processes together.

10.2 Test methods

10.2.1 Test tools

In this test, we use three prototypes.

1. The first one is the knapsack model for requirement selection.

2. The second one is requirement scheduling method based on RCPSP.
3. The third one is the prototype to select and schedule requirement at the same time.

The descriptions of the three tools can be found in chapter 9. In general, they are all Java applications based on integer linear programming and running in Linux environment. They also use the same library CPLEX to solve the ILP model.

10.2.2 Test data

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

Small: 9 requirements and 3 development teams.
Master: 99 requirement and 17 teams.

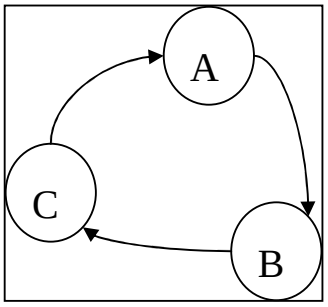
All of the used data sets are available online for research purposes[2].

10.2.3 The requirement dependency

Because there is no dependency in the requirement dataset, we have tested the result by generating random dependencies.

10.2.3.1 PERT & DAG

We use the PERT (program evaluation and review technique) to model the time-related requirement dependencies. The requirements are Vertexes in the graph, and the if requirement R_i REQUIRE requirement R_j, then there is an directed edge (R_j, R_i) which shows that the requirement R_j must be done before R_i. A basic requirement for PERT is that the PERT chart must be a DAG (Directed Acyclic Graphs). The reason is because if there are cycles in the chart, the jobs in a cycle can form a deadlock situation since they are always waiting for others to finish. Like in the following situation, none of the job can ever start because they are inter-waiting for each others.



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

Figure 10.1 the example of a deadlock situation

10.2.3.2 Topological sorting of DAG

For any directed acyclic graph, there is at least one topological sort of the DAG $G=(V,E)$. A topological sort of a DAG is a linear ordering of all its vertices such that if G contains an edge (u,v), then u appears before v in the order[53]. We can get the sort of a DAG in $O(V+E)$ time by performing a depth-first search of G. (The details of topological sorting is shown in section 5.1.)

In order to randomly generate a DAG, we can use the results above. Assume we have already got a topological order, if we add a new edge (u,v) to G, where u appears before v in the order, then new DAG G' is still a directed acyclic graph and the original topological sort is still a topological sort of G'.

Proof: Assume G' is not a directed acyclic graph, so the new edge (u,v) has created a cycle in G'. Then there must be a directed path in G from v to u so that to complete a cycle with the new edge (u,v). If this path exists in G, then vertex v is an ancestor of vertex u in the depth-first forest, and should be place before vertex u in the topological order of G. This yield a conflict, So G' is a DAG.

We can repeatedly add new edge in the above mentioned method, which guarantee the new graph is still a directed acyclic graph. If there are n vertexes, we can add maximally $C_n^2 = n\cdot(n-1)/2$ edges in the graph.

10.2.3.3 Randomly generate dependencies

1. First, give a random permutation of all the requirements as their topological sort. This can be achieved by shuffling algorithm in $O(n)$ time.
2. Use the above mentioned method to generate requirement dependencies. Maximally, we can generate $C_n^2 = n\cdot(n-1)/2$ dependencies between the requirements.

Theoretically speaking, when we randomly generate n dependency, the actual number may be more than n because of the implied dependencies. For example, if ‘A’ need to be before ‘B’, and ‘B’ need to before ‘C’, these two dependencies also implies one dependency that ‘A’ is before ‘C’. So the actual number of dependency is larger than we may think. In this model, we do not count

the implied dependencies. If this topic is an interesting issue for the readers, the following method provides the possibility to find the total number.

The following chart shows an example:

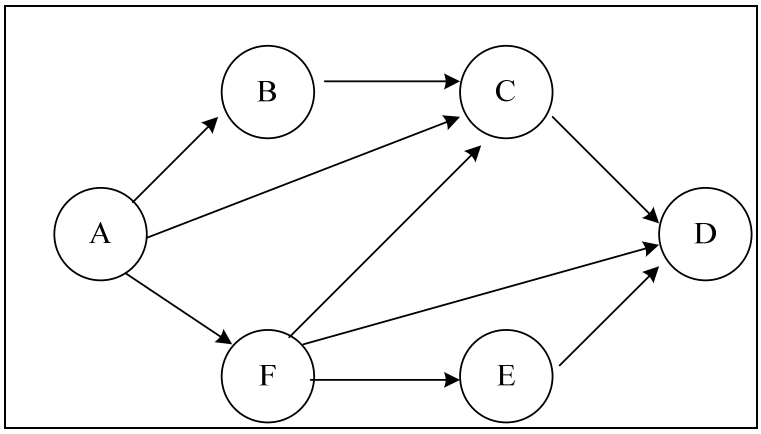


Figure 10.2 An example of requirement dependencies

In the chart, the nodes represent the requirement and the arrows represent the dependency. We can find the actual number of dependency in the following way:

1. Draw the conjunction matrix.

For the example graph, the conjunction matrix is:

G0	A	B	C	D	E	F
A	0	1	1	0	0	1
B	0	0	1	0	0	0
C	0	0	0	1	0	0
D	0	0	0	0	0	0
E	0	0	0	1	0	0
F	0	0	1	1	1	0

In the matrix, if there is an edge pointing from one node to another node, then the corresponding place is shown as 1, otherwise, it is zero. For example, there is an edge from A to C, then in the matrix, you can find the number in the first row and third column is 1.

2. Compute the connectivity of graph.

The connectivity of a graph is shown as the matrix $G*$. Where $G* = \sum_{x=1}^{n} G^x$ and n equals the number of node in the graph G. For the example case, the $G*$ is:

G*	A	B	C	D	E	F
A	0	1	3	5	1	1
B	0	0	1	1	0	0
C	0	0	0	1	0	0
D	0	0	0	0	0	0
E	0	0	0	1	0	0
F	0	0	1	3	1	0

The value in $G*$ equals the number of paths from one node to another node. For example, the number in the first row and third column is 3, which means there are three paths from A to C.

3. Count the number of none-zero numbers.

When we compare the difference between G and $G*$, we find three new non-zero unit, (marked in blue). These dependencies are the implied dependencies.

10.2.4 Rounding

Because the duration of a job equals the expected man days divided by the number of developers in the team, it is possible to get fractional numbers. The rounding is done in the following way:

1. When the duration is between 0 and 1, we round up to 1. Since when the duration of a job is zero and will be removed from our calculation, rounding up helps protecting the loss of valuable data (i.e. estimated value).
2. When the duration is larger than 1, it is rounded off.

10.2.5 The results format

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

We can use the method of scheduling model based on RCPSP (Resource Constrained Project Scheduling Problem) to test the result. This method can tell the minimal time span of the whole release plan.

If we have n requirements selected in the release composition, theoretically, we can set at most, $n \cdot (n-1)/2$ dependencies. In this test, we can find out as the number of dependencies grows, how much and how often it will influence the time span of the release date?

The test datasets are the release plans selected by the knapsack model.

In this table, dependency ratio shows how many time-related interdependencies exist compared with the maximal possible amount. For example for the Small dataset, there are five requirements selected by the knapsack model out of 9, so theoretically, there are at most $5 \times (5-1)/2 = 10$ dependencies possibly in the dataset. We can use the Dependency ratio * largest possible number = Number of Dependencies exist in the dataset.

For the master dataset, we have selected 76 requirements out of 100. Using the same method we

can compute that there are at most $76\times(76-1)/2=2850$ dependencies.

Data Set	Dep ratio	Number of Dep	The project span			Times of delay	The difference between lower bound		
			Max days	Min days	Average days		Max diff	Min diff	Average diff
Small-result (5 Reqs, 60 days)	10%	1							
	20%	2							
	30%	3							
	40%	4							
Master-result (76 Reqs, 30 days)	0.5%	14							
	1%	29							
	2%	57							
	5%	142							

Table 10.1 the result format of the first test

For each row, we will run 100 times based on 100 sets of random dependencies. For every run, we can compute the time span of the project. The *Average finishing time* is the average time span of the 100 tests, and the *Times of delay* shows how many times the project is late in the 100 runs. The *Maximal and Minimal days* record the largest project span and the minimal project of the simulation, and the *Difference from the lower bound* shows how much the result is different from the lower bound.

The second test: **what are the differences when we select and schedule requirements at the same time, and when we select and schedule sequentially?**

We will also use the Small requirement dataset (9 requirements) and Master requirement (99 requirements) data set for this test. Using the method mentioned in section 10.1, the theoretical maximal number of interdependencies in the requirement dataset are 36 for the Small data set and 4851 for the Master dataset.

For each row, we will run 100 times based on 100 groups of randomly-generated dependencies. The following activities diagram shows the processes of every run.

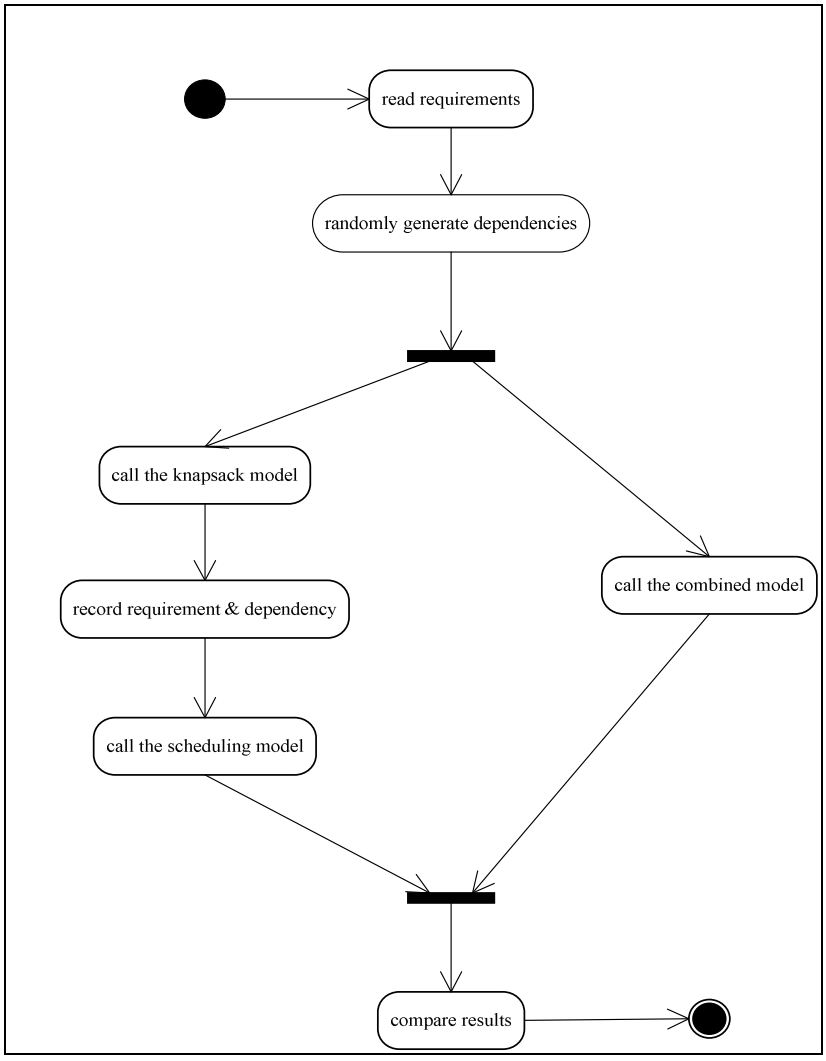


Figure 10.3 the activity diagram for model comparison

For every run, we use the combined model to select and schedule the requirement at the same time. For the dependencies, we not only consider their logical relationships, but also the timing relationships. For example, if R_j requires $R_{j'}$, then we need to first select $R_{j'}$ then R_j, and secondly schedule R_j after $R_{j'}$ is done. We will document the revenue of every run and compute the average revenue of the hundred samples in *the average revenue of the combined model*.

Data Set	Dep ratio	No. of Dep	Statistics for the 100 cases			Statistics only for the delayed cases					
			Average revenue (combined)	Average revenue (knapsack)	Average project span	Times of delay	Average revenue (combined)	Average revenue (knapsack)	Average project span	Average revenue difference	Average time difference
Small (9 Reqs, 60 days)	3%	1									
	10%	3									
	15%	5									
	20%	7									
Master (99 Reqs, 30 days)	0.5%	24									
	1%	48									
	2%	97									
	5%	242									

Table 10.2: the result format of the second test

Based on the same dependencies, we will also compute the average revenue using the original knapsack model. But in the knapsack model, we only consider that the implication dependency have logical meanings so that if R_j requires $R_{j'}$, then $x_j \le x_{j'}$. We will leave the time related issues to the project plan phase. After selecting the requirements, we will compute the time span of them using the scheduling method—"schedule the requirement with RCPSP", and compute the average time span of the 100 samples, and how many time the project is late.

Based on the simulation results, we will do two statistics. The first one is the statistics for the 100 runs, like the average revenues of the two models and the average of the project span. But in the 100 runs, it is possible that in same cases, the two models will select the same group of requirements, then these runs will not be of no interest because the revenue difference and time difference will be both zero. It is more interesting to see the difference for the projects which can not finish on time, so we will make the second statistics only based on the projects that run out of time.

10.3 Test result

10.3.1 The first group of results

The first test is to find how the time-related requirement dependencies influence the project span. The computational results are shown in the following table.

			The project span		The difference between lower bound

			Max days	Min days	Average days	delay	Max diff	Min diff	Average diff
Small-result (5 Reqs, 60 days)	10%	1	83	55	58.80	16	0.00%	0.00%	0.00%
	20%	2	93	55	63.70	40	27.27%	0.00%	0.93%
	30%	3	103	55	70.42	62	27.27%	0.00%	2.64%
	40%	4	108	55	75.32	76	14.55%	0.00%	2.12%
Master-result (76 Reqs, 30 days)	0.5%	14	40	30	30.93	33	30.00%	0.00%	2.70%
	1%	29	46	30	31.38	27	8.57%	0.00%	0.22%
	2%	57	69	30	36.92	76	22.58%	0.00%	2.13%
	5%	142	84	38	56.15	100	19.23%	0.00%	3.47%

Table 10.3 The test result of the first test

The following four figures visualize the result.

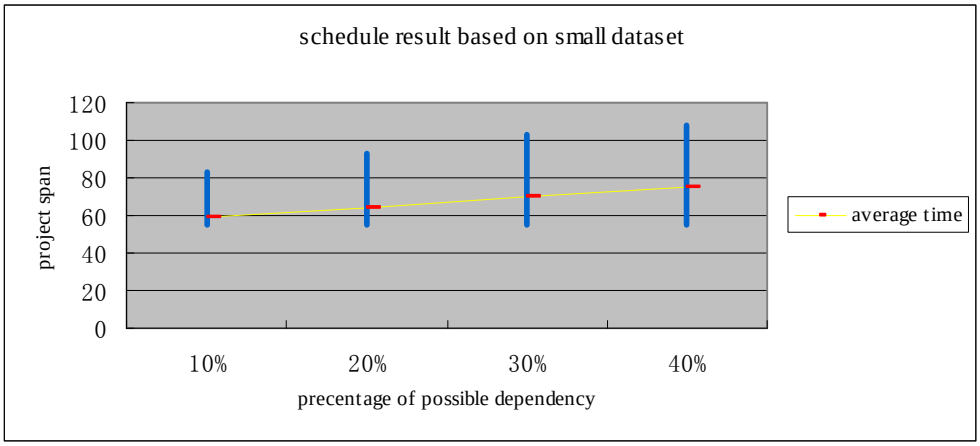


Figure 10.4 the schedule result based on small dataset

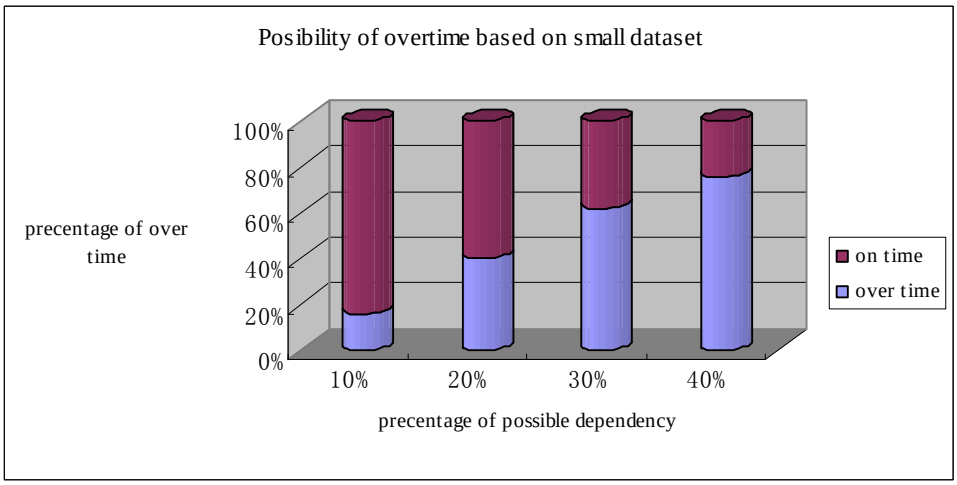


Figure 10.5 the possibility of overtime based on small dataset

For the small data set, we can find that as the number of requirement dependencies increases, the

maximal project span, the average project span and the number of overtime project keep on increasing (see figure 10.4 and 10.5). The average project span grows from 58.8 to 75.32, the number of delayed projects grows from 16 to 76 and the maximal project span grows from 83 to 108 days. However, the minimal project span remains at 55 days, which means even there are a large number of dependencies, it is still possible to keep the project on time, but the chance of meeting the deadline keep reducing. From the last column, we can find the computed result is not far away from the lower bound. The difference is within 3%.

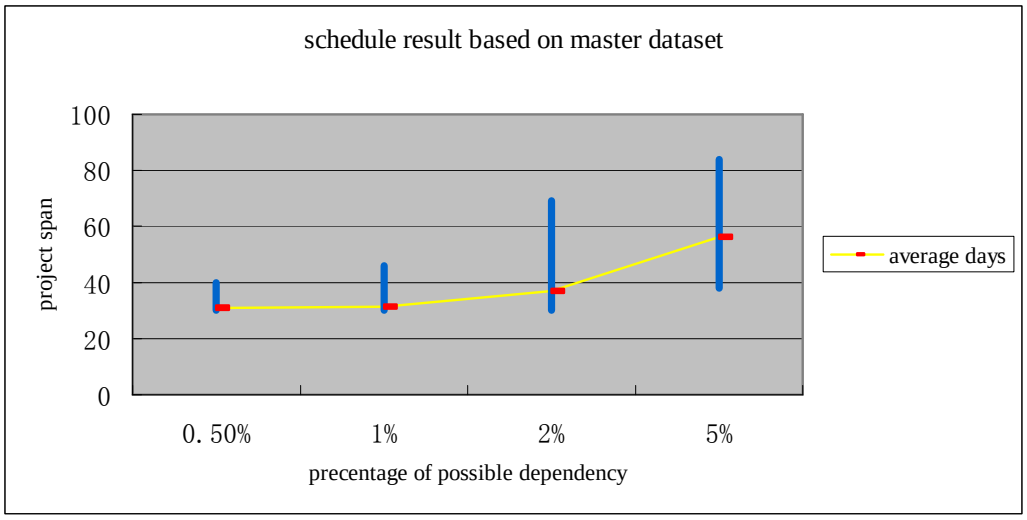


Figure 10.6 the schedule result based on master dataset

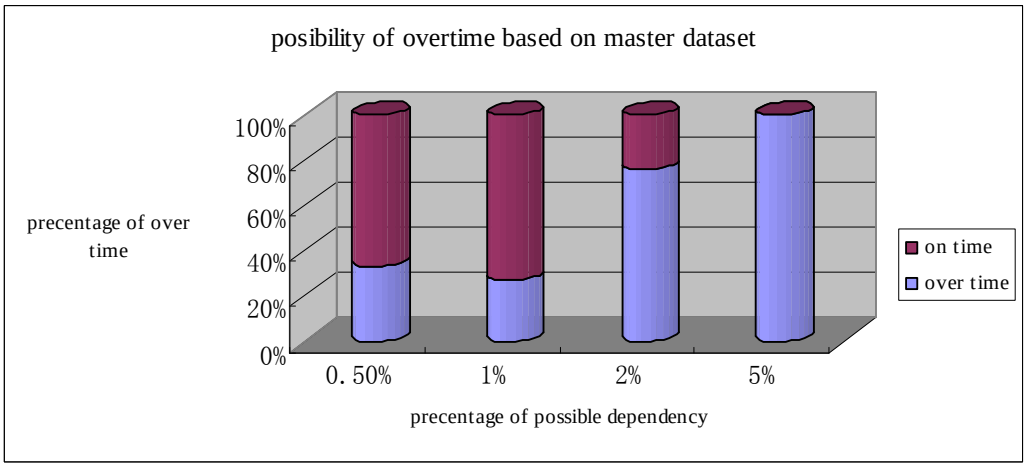


Figure 10.7 the possibility of overtime based on master dataset

The same trend can also be found in the master dataset (see figure 10.6 & 10.7). Special attention is needed for the last row. When there are 142 dependencies, which are only 5% of the maximal feasible number, the result explodes. The minimal time to finish the project is at 38 days, and the project is 100% late. The average project span is at 56.15, which is almost twice as much as in the project plan. Theoretically, it is still possible that the project can be completed on time, but

unfortunately, it did not happen within 100 time of computation in our case.

We have also used other dataset for this test. For example, for the small dataset, if the release planning period is 50 days or 70 days instead of 60days, what will be the scheduling result? We also modified the planning date for the master dataset, and schedule them. Based on the simulation, the figures are stable and very similar to this group of result. For the sake of conciseness, these results are documented in Appendix 2.

From the result, it is clear that the requirement dependencies greatly influence the project plan of the release. When there are only a few dependencies, the delay is not significant and does not happen often. As the number of requirement dependencies increases, the chance that the project will be delayed is very high. Unfortunately, how many dependencies can exactly exist between the requirements remains unknown, however, from a former survey[21], about 80% of the requirements are interdependent, and most of the requirements dependencies are precedence constraints (*Implication* or *cost-related* dependencies). We can expect the number of dependencies is at least higher than the second row of the small or master data set. (To set dependencies between 80% of requirements, we need at least $n \times 0.8/2$ dependencies, where n equals the number of requirement)
We can also find that the difference between the actual project span and the lower bound is not significant. From our computation, the difference is just about from 0% to 3.47%. This figure may trigger the interest of a new searching algorithm for this problem, since the project span is very close to the lower bound which can be found in polynomial time.

10.3.2 The second group of result

The second test is to compare the requirements selection using the knapsack model and the combined model. We will compare first: the revenue difference between the two models; second: the time difference to completely implement these selected requirements.

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

Table 10.4 the result of the second test

The test data is documented in the above table. It can be divided into two parts: the statistics for the whole 100 cases (from the fourth column to the sixth column), and the statistics only for the delayed projects (from seventh column to the twelfth column).
The following 2 charts present the results.

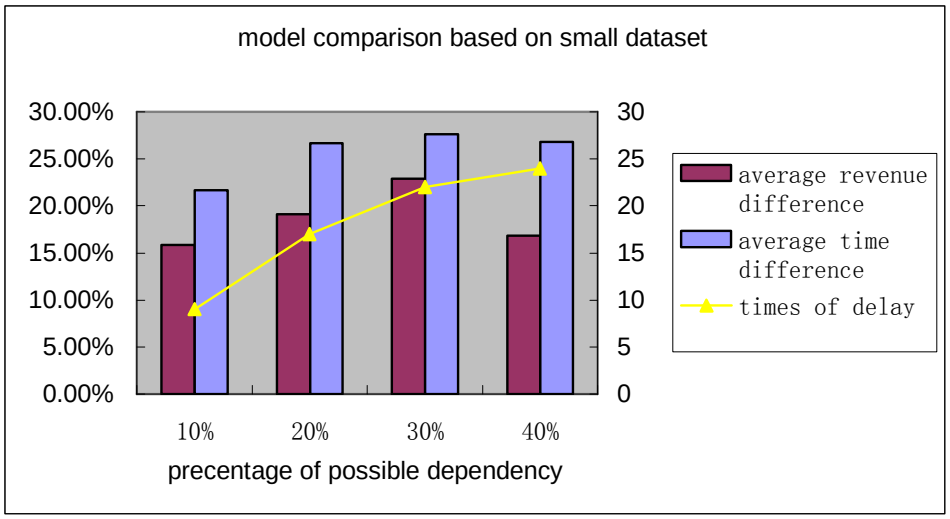


Figure 10.8 the model comparison based on small dataset

In the small data set, it is clear that the average revenue of the knapsack model is lower than the new model. We can also find both the revenue of the two models decreases as the number of dependencies increase. In the delayed projects, as we expected, following the Select→ Schedule processes, the more dependencies we have, the more the possibility of the project being delayed. However, this trend does not appear for the average project span, and although the average revenue difference is lower than the average time difference, they do not differ too much.

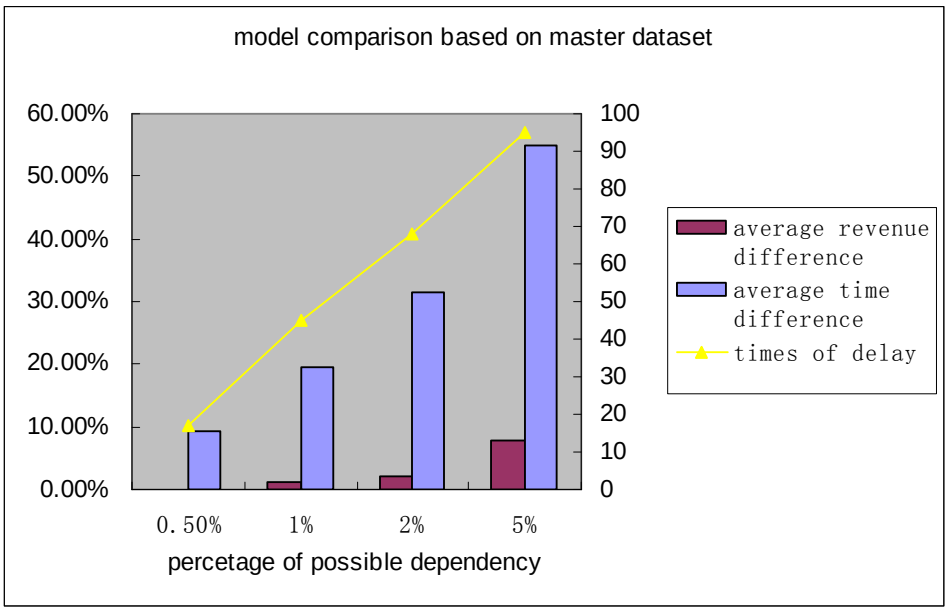


Figure 10.9 the model comparison based on master dataset

In the master data set, it is also clear that the revenue of the new model is lower than the revenue of the knapsack model. As the number of dependencies increase, following the Select→ Schedule processes, the average project span extends and the chance that the project is delayed becomes higher and higher. The same trend exists on the average revenue difference and the average time difference, but this time, the difference between these figures are significant (see figure 10.9). For example when there are 48 dependencies, the revenue in the new model is only 1.15% lower than the knapsack model, but to implement the requirements selected by the knapsack model, we need to spend 19.41% more time than planed. Because there are more requirements in the master dataset, we consider the result is more representative than the result from the small dataset.

We can draw two conclusions from this test:

- First: the precedence constraint significantly influence the requirement selection & scheduling and it is more efficient to consider the project plan issues when select the requirements. From the test result, when ignore the timing issues on the requirement selection, the project stand a high change of being delayed. The simulation result also suggests that it is more efficient to take the project plan issues into account when selecting the requirements, because the revenue loss of the new model is significantly less than the additional time we would spend on the implementation.
- Second, in a market oriented approach, the original Select→Schedule processes are challengeable. In order to fulfill the pressure on time to market, we also need to consider the project plan issue when we select the requirements. When the processes are separated the project stands a higher chance of being late, and from the above conclusion it is quite evident that this process is also not quite efficient. So far, none of the release planning process or software engineering process takes selection and timing as a joint process. A new approach in release planning process is therefore needed and more research, attention and time will be required to achieve considerable results.

11. Conclusion & future research

11.1 conclusions

This thesis has investigated software release planning from the perspective of three different scientific fields—the information science part on the factors and processes for release planning; the methodical modeling part on Integer Linear Programming models; and the computer science part on prototype design and testing. The following figure shows the components and relationships within the three parts.

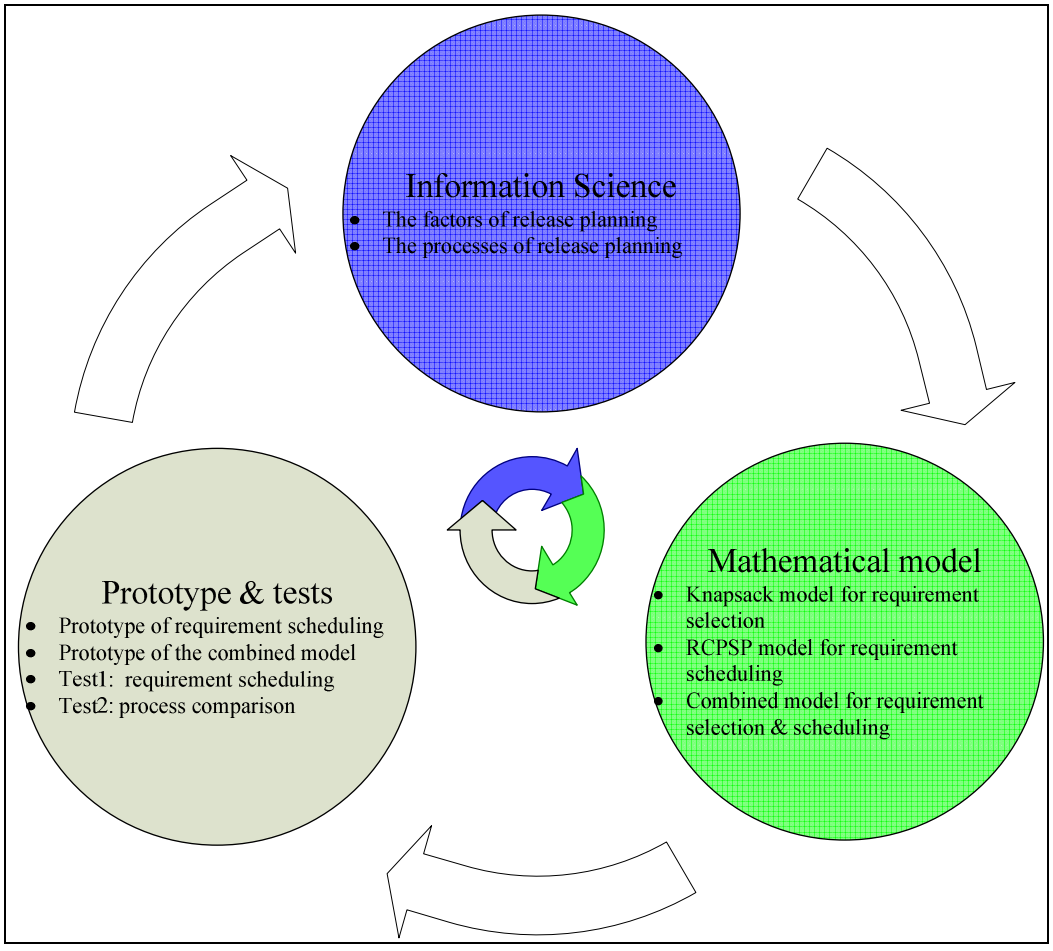


Figure 11.1 relationships within the three fields

In the field of information science, we searched for what are the factors and processes for making a software release plan. We have identified eight factors: value, cost, priority, risk, quality and dependency for each requirement, and the time to market and the company resources based on the market and company's current situation. Regarding to the release planning processes, it follows

five key processes issue→specify→select→schedule→construct to determine whether a requirement should be constructed. The factors are estimated during the “specify” phase and will be used as selection and scheduling criteria in the follow two processes.

Mostly, a product manager needs to deal with hundreds or even thousands of requirements. Without proper tooling support, it will be a tedious job and almost impossible to find the best solution. The complexity of the problem requires us first to model the problem properly through mathematical means. The second part of the thesis has been devoted to this issue. Using integer linear programming, we have presented three main models: the knapsack model for requirement selection, the RCPSP model for requirement scheduling and a combined model for requirement selection and scheduling at the same time. We have also modeled some management steering mechanisms as extensions of the basic models, like requirement dependencies, different time availability the holiday seasons, etc. Because all the models are based on integer linear programming, most of the models and extensions are compatible with others, only with few exceptions. The details of the relationship between each model can be found in chapter 8.

Based on the mathematical models in the second part, we have implemented two prototypes using Java programming language—the requirement scheduling models based on RCPSP and the combined model for requirement selection and scheduling. Using these two prototypes, we ran two simulation tests. The first test is how much the requirement dependencies can influence the project. Based on the simulation result, we found out the requirement dependencies that have significant influence on the project span. The second simulation test was to compare the result when we select and schedule the requirement separately or when we select and schedule requirement simultaneously. The simulation result suggests combining the two processes together, can not only guarantee the project to finish on time, but will also increase efficiency.

The simulation results have suggested an opportunity for process improvement on release planning. So far, the requirement selection and requirement scheduling are separated in most of the release planning process models and the marketed oriented software development models. In fact, combining these two processes appears to be better. It can guarantee the completion of the project within the set deadline as well as will increase efficiency. This may trigger more investigations on the market oriented software development processes model.

11.2 future researches

As the thesis has investigated in three scientific fields, the future research also includes three parts.

- In the information science field, more attentions are required on the release planning process optimization. The simulation results in this thesis show convincing figures to combine the requirement selection and scheduling together. This has suggested a potential field for process improvement in the future.
- In the mathematical modeling field, there are still two possibilities for improve the models. First, ILP is not a very efficient way for scheduling. Some other techniques like Constraint Satisfaction Problem (CSP) and local search method may appear to be more efficient. The simulation results also suggest using some searching mechanism for scheduling, because the

optimal value found by ILP does not differ too much from the lower bound, which can be computed on polynomial time. Another opportunity is to better integrate the different models. The scheduling model and the combined model for selection and scheduling do not fully integrated with the knapsack model at this moment. For example, we need to set restrictions on hiring external personnel if we want to use the scheduling mode later. Better models, or additional extensions are required to enrich the functionality of the models and the compatibility of the models.

- In the computer science field, we can try to find better tools for release planning support. For example to show the results in the Gantt chart is a good extension to visualize the result. It is also better to design and database system for requirement management instead of using hard-copy data.

Appendix 2 the experiment result based on other sample

Schedule result (Small)

Data Set Small-result	Dep ratio	Number of Deps	Max days	Min days	Average finishing time	Times of delay	Difference from the lower bound
(4 Reqs, 50 days)	10%	N/A					
	20%	1	83	50	57.78	45	0.00%
	30%	N/A					
	40%	2	98	50	69.24	84	1.72%
Small-result (5 Reqs, 60 days)	10%	1	83	55	58.80	16	0.00%
	20%	2	93	55	63.70	40	0.93%
	30%	3	103	55	70.42	62	2.64%
	40%	4	108	55	75.32	76	2.12%
(5 Reqs, 70 days)	10%	1	83	70	72.21	17	0.00%
	20%	2	103	70	75.06	34	2.48%
	30%	3	118	70	81.62	59	2.64%
	40%	4	118	70	87.17	65	4.33%

Schedule result (Master)

Data Set Master-result	Dep ratio	Number of Deps	Max days	Min days	Average finishing time	Times of delay	Difference from the lower bound
(63 Reqs, 20 days)	0.5%	9	20	20	20	0	0.00%
	1%	19	29	20	21.17	32	5.85%
	2%	39	31	20	21.41	39	3.27%
	5%	97	51	22	33.76	100	8.11%
(76 Reqs, 30 days)	0.5%	14	40	30	30.93	33	2.70%
	1%	29	46	30	31.38	27	0.22%
	2%	57	69	30	36.92	76	2.13%
	5%	142	84	38	56.15	100	3.47%
(84 Reqs, 40 days) [3]	0.5%	17	44	40	40.18	13	0.32%
	1%	34	50	40	40.52	13	0.30%
	2%	69	*50*	*40*	*43.95*	*14*	*2.12%*
	5%	174	*90*	*52*	*71.7*	*20*	*4.93%*

[3] The last two rows (2% & 5%) are based on 20 runs each.

References

[1] Xu, L., & Brinkkemper, S. (2005). Concepts of Product Software: Paving the Road for Urgently Needed Research. In J. Castro & E. Teniente (Eds.), *The first International Workshop on Philosophical Foundations of Information Systems Engineering (PHISE'05)* (pp. 523-528). FEUP Press

[2] The Slide of MBI colloquium—ITEA project proposal

[3] Donald Firesmith. "Prioritizing Requirements". Software Engineering Institute, U.S.A. JOURNAL OF OBJECT TECHNOLOGY Vol. 3, No.8, September-October 2004

[4] Leffingwell, D., and Widrig, D. "Managing Software Requirements - A Unified Approach", Addison-Wesley, Upper Saddle River, NJ. 2000

[5] Suzanne Robertson, James Rovertson. "Mastering the requirements process". ACM Press, 1999

[6] Michael R. Garey and David S. Johnson "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H. Freeman. 1979.

[7] Marjan van den Akker, Sjaak Brinkkemper, Guido Diepen, Johan Versendaal. "Software product release planning through optimization and what-if analysis" …

[8] Karlsson, J and Ryan, K. "A cost-Value Approach for Prioritizing Requirements". IEEE Software, September/October 1997 pp 67-74.

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

[10] Inge van de Weerd, Sjaak Brinkkemper, Richard Nieuwenhuis, Johan Versendaal, Lex Bijlsma "A reference framework for software product management". Utrecht UniversityTechnical Report UU-CS-2006-014, 2006.

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

[12] Lionel C. Briand, Khaled El Emam, Dagmar Surmann, Isabella Wieczorek, Katrina D. Maxwell "An assessment and comparison of common software cost estimation modeling techniques" . Proceedings of the 21st international conference on Software engineering, 1999 Pages: 313 - 322

[13] Sjaak, Brinkkemper, : slides of "information business" course.

[14] Kitchenham, B. Pfleeger, S.L. "Software quality: the elusive target", Software, IEEE Publication Date: Jan 1996,pp 12-21

[15] Cusumano, M.A.: The Business of Software. Free Press (2004)

[16] Brealy, R.A, S.C. Myers, A.J. Marcus "Fundamentals of Corporate Finance." McGraw-Hill/Irwin, New York, 2004, 4th edition.

[17] D. Greer, G. Ruhe. "Software release planning: an evolutionary and iterative approach" Information and Software Technology 46 (2004) 243–253

[18] Günther Ruhe. "Software release planning". Handbook Software Engineering and Knowledge Engineering - Vol. 3. 2005

[19] Marjan van den Akker and Han Hoogeveen. "Minimizing the number of late jobs in case of stochastic processing times with minimum success probabilities". UU technical report UU-CS-2004-067. 2004

[20] Dimitri Golenko-Ginzburg, Ahron Gonik, Zohar Laslo, "Resource constrained scheduling simulation model for alternative stochastic network projects". Mathematics and Computers in Simulation 63 (2003) 105-117

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

[22] Pär Carlshamre, "Release Planning in Market-Driven Software Product Development: Provoking an Understanding". Requirements Engineering, Volume: 7, Issue: 3 (September 1, 2002), pp: 139-151

[23] Donald Firesmith: "Prioritizing Requirements", in Journal of Object Technology, vol. 3, no. 8, September-October 2004, pp. 35-47

[24] Novorita, R., Grube, G., "Benefits of Structured Requirements Methods for Market-Based Enterprises", Proceedings of International Council on Systems Engineering Sixth Annual International

Symposium on Systems Engineering: Practice and Tools (INCOSE'96), Boston USA, July 1996

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

[26] Cusumano, M.A.: The Business of Software. Free Press (2004)

[27] Björn Regnell, Lena Karlsson, Martin Höst. "An Analytical Model for Requirements Selection Quality Evaluation in Product Software Development". *RE'03 - IEEE 11th International Conference on Requirements Engineering, September 8-12, Monterey Bay, California USA, 2003*

[28] JN och Dag, V Gervasi, S Brinkkemper, B Regnell, "A Linguistic-Engineering Approach to Large-Scale Requirements Management". IEEE Software, January 2005, pp 32-39

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

[30] Christopher McPhee, Dr. Armin Eberlein , "Requirements Engineering for Time-to-Market Projects" Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems. 2002

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

[32] Suzanne Robertson, James Robertson, "Mastering the requirements process". ACM press book. 1999

[33] Günther Ruhe. "Software release planning". Handbook Software Engineering and Knowledge Engineering - Vol. 3. 2005

[34] Marjan van den Akker and Han Hoogeveen. "Minimizing the number of late jobs in case of stochastic processing times with minimum success probabilities". UU technical report UU-CS-2004-067. 2004

[35]樊耘，等。 "管理学" 陕西人民出版社， 2003。 Pp 313

[36] Wolsey L.A. "Integer programming". Wiley-Interscience Series In Discrete Mathematics and Optimization. 1998

[37] Marjan van den Akker, Sjaak Brinkkemper, Guido Diepen, Johan Versendaal. "Software product relese planning hrough optimization and what-if analysis"

[38] Björn Regnell, Lena Karlsson, Martin Höst. "An Analytical Model for Requirements Selection Quality Evaluation in Product Software Development". *RE'03 - IEEE 11th International Conference on Requirements Engineering, September 8-12, Monterey Bay, California USA, 2003*

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

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

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

[42] Yeh, A., "Requirements Engineering Support Technique (REQUEST) – A Market Driven Requirements Management Process", Proceedings of *Second Symposium of Quality Software Development Tools* , pp. 211-223, New Orleans USA, IEEE Computer Society Press, May 1992.

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

[44] Novorita, R., Grube, G., "Benefits of Structured Requirements Methods for Market-Based Enterprises", Proceedings of *International Council on Systems Engineering Sixth Annual International Symposium on Systems Engineering: Practice and Tools* (INCOSE'96), Boston USA, July 1996.

[45] Emile Aarts, Jan Karel Lenstra. " Local Search In Combinatorial Optimization". Wiley 1997. pp 361 – 415.

[46] Aristide Mingozzi, Vittorio Maniezzo, Salvatore Ricciardelli, Lucio Bianco. "An Exact Algorithm

for the Resource-Constrained Project Scheduling Problem Based on a New Mathematical Formulation". Management Science, Vol. 44, No. *5*. (May, 1998), pp. 714-729.

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

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

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

[50] Marjan van den Akker, Sjaak Brinkkemper Guido Diepen, Johan Versendaal, "Software product release planning through optimization and what-if analysis"

[51] Pär Carlshamre, "Release Planning in Market-Driven Software Product Development: Provoking an Understanding". Requirements Engineering, Volume: 7, Issue: 3 (September 1, 2002), pp: 139-151

[52] , R.A., S.C. Myers, A.J. Marcus: Fundamentals of Corporate Finance. McGraw-Hill/Irwin, New York, 2004, 4th edition

[53] Thomas H.Cormen, Charlse E. Leiserson, Ronald L. Riverst, Clifford Stein . Introduction to algorithms ", second edition. MIT Press, 2001, pp 549 -551