2013

Release Planning through Functional Architecture

USING THE FUNCTIONAL ARCHITECTURE FRAMEWORK FOR PRODUCT SOFTWARE RELEASE PLANNING

Master Thesis
Version 1.01 (public version)
August 20, 2013

Jeffrey Breedijk (3728757)
Master Business Informatics
Institute of Information and Computer Science
Utrecht University

Universiteit Utrecht

MASTER THESIS

Thesis title: Release planning through functional architecture

Author:
Jeffrey Breedijk
Master Business Informatics
Utrecht University
j.c.breedijk@gmail.com

First supervisor:
Prof. dr. S. Brinkkemper
Department of Information and Computing Sciences
Utrecht University

Second supervisor:
K. Vlaanderen MSc.
Department of Information and Computing Sciences
Utrecht University

External supervisor:
Dr. H.W. van der Schuur
AFAS Software B.V.

External supervisor:
M. Amri
AFAS Software B.V.

ABSTRACT

Product software companies sell a standardized software product to a large market. Since a high number of customers use the product, product software companies cope with large numbers of requirements for the product they sell. One of the main activities of the software product manager is to select a subset of these incoming requirements, that will be developed in the next coming release. Therefore, software product managers must constantly balance between the highest customer satisfaction and the available development capacity within the firm.

A recent study on the Functional Architecture Framework (a framework for functional decomposition of a software product into functional components) provides software product managers with a formal method to identify and organize these large volumes of incoming requirements by functional component. But this is still a partial solution to the release planning problem.

In this research, a product software release planning method is presented which builds on the requirements management method of the Functional Architecture Framework, so a single method is created to manage the process from incoming requirement to planned release. For this method, different prioritization methods, cost estimation methods, and planning algorithms are studied, evaluated, adapted and combined which resulted in a method that uses linear optimization for product software release planning. The release planning method is eventually validated at a large product software company, by applying it on their release planning process.

INHOUD

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Product software is defined as *"a packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market"* (Xu & Brinkkemper, 2005)

In today's software industry, product software is becoming more complex and customers are becoming more demanding. Due to this change, the role of the software product manager, who is responsible for the lifecycle of the software product, becomes more complex and critical. Since there is little education and no extensive body of knowledge in the software product management (SPM) field, improving the quality of the SPM processes is often considered difficult (Ebert, 2007) (Bekkers, van de Weerd, Spruit, & Brinkkemper, 2010).

During the last years, research in the field of SPM has been conducted with different methods, for different software product management aspects as output. Bekkers, Weerd, Spruit, and Brinkkemper (2010) developed the Software Product Management competence model, which is a model that gives an overview of the software product management domain. In this competence model, the SPM domain is divided into four business functions (sorted from strategic level to operational level):

- Portfolio Management;
- Product Planning;
- Release Planning;
- Requirements Management.

Each business function contains of a number of focus areas, which are like activities for the product manager. Bekkers et al. (2010) created a focus area maturity matrix for the SPM focus areas, where there are a number of maturity capabilities defined per focus area.

However, this SPM reference framework is just a guideline for a product manager to improve the product management processes within a company. In a recent study, the Functional Architecture Framework (Salfischberger, van de Weerd, & Brinkkemper, 2011) has been developed in order to assist the software product manager in the processes belonging to the requirements management business function. The Functional Architecture Framework assists the software product manager by supplying a method that is in line with the capabilities described in the focus area maturity matrix.

The problem is that, at this moment, the Functional Architecture Framework assists in the requirements management process, and there isn't any formal software product management framework or method for the business functions release planning, product planning and portfolio management. The goal of this research is to find out how the Functional Architecture Framework can assist in the release planning processes (as described in the SPM Competence Model), in order to make the Functional Architecture Framework more valuable and useful in the software product management domain.

In chapter 1.1, the research questions are described and in chapter 1.2,, the context of the research is elaborated with a more extensive view on the SPM Competence Model, Focus Area Maturity Matrix, and the Functional Architecture Framework. In chapter 1.3, the research approach (methods and deliverables) is described. In chapter 2, different focus area methods from literature are compared and evaluated.

1.1. RESEARCH CONTEXT

In order to understand the context of this research, the next three background models of this research are elaborated:

- Software Product Management Competence Model (chapter 1.1.1); this model represents an overview of the business functions and focus areas involved with software product management.
- Focus Area Maturity Matrix (chapter 1.1.2); the focus areas from the Software Product Management Competence Model each have a set of maturity levels. These maturity levels are described in the Focus Area Maturity Matrix.
- Functional Architecture Framework (chapter 1.1.3); the Functional Architecture Framework is a framework developed to assist the software product manager in managing requirements.

1.1.1. SOFTWARE PRODUCT MANAGEMENT COMPETENCE MODEL

Much literature has been written about the domain of software product management (SPM). An overview of the SPM domain is provided in the form of a SPM Competence Model (Figure 1), where the SPM domain is split up in four business functions (Bekkers, van de Weerd, Spruit, & Brinkkemper, 2010). The contents of these business functions are considered the main baseline of software product management processes in this research. Each business function has multiple focus areas, which represents “a strongly coherent group of capabilities within a business function” (Bekkers, van de Weerd, Spruit, & Brinkkemper, 2010).

The business function *Portfolio Management* involves the strategic information gathering and decision making. This business function exists of the focus areas:

- Market analysis; which involves the gathering of external decision support information.
- Product lifecycle management; which involves the strategic decision making for a product portfolio.
- Partnering & contracting; which involves partnership establishment, product pricing, and decision making in product distribution.

The business function *Product Planning* involves gathering of information for building a roadmap for one or more product line(s). This business function consist of the focus areas:

- Roadmap intelligence; gathering of the decision support information for the product roadmap.
- Product road mapping; creation of a long-term product roadmap.
- Core asset road mapping; the planning of core asset (components used by multiple products) development.

The business function *Release Planning* is about creating and launching product updates/releases. This business function consists of the focus areas:

- Requirement prioritization; the prioritization of the incoming requirements.
- Release definition; the selection of requirements, and the creation of release documentation.
- Release definition validation; the validation of the release definition by the company board.
- Scope change management; acting on sudden changes in release development after decision making.
- Build validation; the validation of the developed release by different parties.
- Launch preparation; involves the necessary steps in order to successfully bring the release to the market.

The final business function is called *Requirements*[1] *Management* and involves the management of incoming software requirements. The focus areas belonging to this business function are:

- Requirements gathering; acquisition of requirements from external and internal stakeholders.
- Requirements identification; identifying the gathered requirements and translating requirements into concrete product requirements.
- Requirements organizing; structuring the identified requirements.

In Figure 1, a graphical representation of the SPM Competence Model is presented (Bekkers, van de Weerd, Spruit, & Brinkkemper, 2010). In both left and right side, the different stakeholders are presented. The middle part shows the four different business functions with their focus areas. The arrows between the stakeholders and the business function represents the interaction between the stakeholder and the processes within the business functions. Adjacent business functions have strong interrelationships, which is represented by the arrows between the business functions. The arrows between the different focus areas are representing the main process- and information flows.

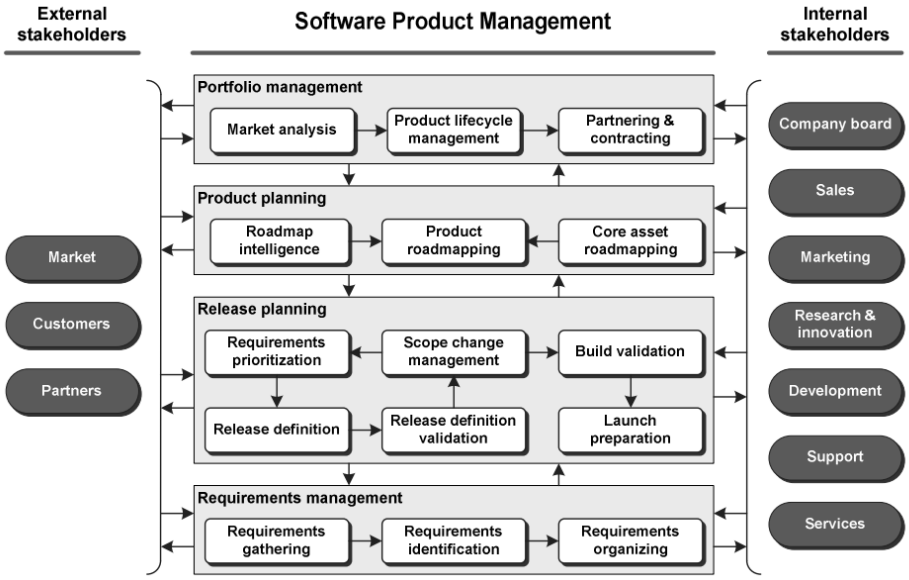


Figure 1: Software Product Management Competence Model

1.1.2. FOCUS AREA MATURITY MODEL

Based on the SPM Competence Model, the Focus Area Maturity Model, which is shown in Figure 2, was created by Bekkers et al. (2010). For each focus area of the SPM Competence Model, a maturity level between 1 and 10 can be reached. To reach a certain maturity level, a company must meet some capabilities which are based on best practices in the SPM domain. Depending on the focus area, there are three to six capabilities to meet in order to reach the highest maturity level. The Focus Area Maturity Model

[1] A requirement in software context can be defines as (Wiegers, 2003) (IEEE, 1990):

1. A condition or capability needed by a user to solve a problem or achieve an objective
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents

is used in this research to fully understand the possible process capabilities of each focus area within the SPM reference framework.

The capabilities in the focus area maturity model are marked with a capital letter. When a capability is met, the maturity level for that focus area is [maturity level of next capability] – 1. So for example, if capability B for the focus area *requirements gathering* is met (in Figure 2) , the maturity level for this focus area will be 3.

In the next paragraphs, each focus area and its maturity levels are elaborated. Due to the scope of this research, only the focus areas belonging to the Release Planning business function are elaborated.

	0	1	2	3	4	5	6	7	8	9	10
Requirements management											
Requirements gathering		A		B	C		D	E	F		
Requirements identification			A			B		C			D
Requirements organizing				A		B		C			
Release planning											
Requirements prioritization			A		B	C	D			E	
Release definition			A	B	C				D		E
Release definition validation					A			B		C	
Scope change management				A		B		C		D	
Build validation					A			B		C	
Launch preparation		A		B		C	D		E		F
Product planning											
Roadmap intelligence				A		B	C		D	E	
Core asset roadmapping					A		B		C		D
Product roadmapping			A	B			C	D		E	
Portfolio management											
Market analysis					A		B	C	D		E
Partnering & contracting						A	B		C	D	E
Product lifecycle management					A	B			C	D	E

Figure 2: Focus Area Maturity Model with the Release Planning focus areas highlighted

Requirements prioritization

The first focus area within the release planning business function, *requirement prioritization*, has five capabilities it must meet in order to reach the highest maturity level. In order to meet the highest maturity level, the organization must involve internal- and external (partners & customers) stakeholders in the prioritization process. The organization must also use a prioritization methodology that includes revenue (and preferably cost) consideration. There are numerous prioritization methodologies available for prioritization, some popular methodologies are MoSCoW, Binary Search Tree, and the Wiegers' Matrix (Racheva, Davena, & Buglione, 2008) (Bebensee, van de Weerd, & Brinkkemper, 2010).

Release definition

The second focus area, which is *release definition*, has also five necessary capabilities to reach the highest maturity level. In order to reach the maximum maturity level, the organization has to implement a computational support tool to select requirements, based on constraints like (at least) engineering capacity, priority, cost, and requirement interdependency. The computational support tool must also be able to select requirements for multiple releases. The selected requirements should be documented in a standardized release definition document (Bekkers, van de Weerd, Spruit, & Brinkkemper, 2010).

Release definition validation

The focus area, called *release definition validation* contains three capabilities it must meet in order to reach the highest maturity level. The first capability is the internal stakeholder validation of the release definition (which was made in the previous focus area). The second capability is the company board approval of the release definition. This capability is more like an extension of the first capability, since the board could be

considered an internal stakeholder. The third and last capability in this focus area is writing a business case for the release (before it is validated). The business case must include a ROI calculation in order to highlight the cost and benefit to the approvers of the release definition (Bekkers, van de Weerd, Spruit, & Brinkkemper, 2010).

Change management

The fourth focus area, called *scope change management*, has four capabilities. To cover all capabilities, an organization must implement change management processes like (Bekkers, van de Weerd, Spruit, & Brinkkemper, 2010) (Utrecht University, 2013).:

- Event (release) notification;
- Milestone monitoring; for monitoring the development progress
- Impact analysis; for handling problems occurring during the development of the release
- Scope change management; for handling release scope changes

Build validation

The fifth focus area within the release planning business function is *build validation*. To reach the maximum maturity level for this focus area, a company should let the release be validated by both internal-, as external stakeholders. If applicable, the release should also be certified by a third party (Bekkers, van de Weerd, Spruit, & Brinkkemper, 2010). A third party should be an independent software certification laboratory, which have their own resources to analyze and test different software components (Morris, Lee, Parker, Bundell, & Lam, 2001).

Launch preparation

The last focus area is *launch preparation.* For this focus areas there are capabilities for the internal- and external communication of the release launch. Also, formal board approval must be given before the official launch of the release. A launch impact analysis must be performed in order to analyze the time needed for release implementation. And last but not least, new training material must be prepared given in order to make the customer familiar with the new functionalities, and external expressions (like websites and help files) must be updated (Bekkers, van de Weerd, Spruit, & Brinkkemper, 2010).

1.1.3. FUNCTIONAL ARCHITECTURE FRAMEWORK

The Functional Architecture Framework (FAF) is a framework that formalizes the software requirements management business function. The goal of the FAF is to create functional components that function as categories that can be assigned to incoming requirements. This is useful because a requirement can immediately be assigned to a certain (functional) part of the software, which contributes in the ease of requirement identification and requirement organizing (Salfischberger, van de Weerd, & Brinkkemper, 2011).

The FAF uses Functional Architecture Modeling (FAM) (Pachidi & Brinkkemper, 2010) to visualize the product in different layers: the product context, the functional modules, and the functional sub-modules. The different layers are used to decompose a large and complex software product in smaller functional modules.

The highest modeling level is the *product context layer*. The product context layer depicts the product as used by the customers (Salfischberger, van de Weerd, & Brinkkemper, 2011). This layer exist of multiple

functional modules which always have interaction with each other. Within the product context, the product scope is defined. The product scope highlight(s) the functional module(s) that will be elaborated.

Each functional module within the product scope has multiple *sub-modules*. The sub-modules are modeled on a functional level, which means that they are modeled from a user's perspective instead of a technical perspective. Figure 3 shows an example of the product context, product scope and module level of the FAM.

In order to create functional components, the variability of the software product must be determined. Variability is "the ability of a system to be efficiently extended, changed, customized or configured for use in a particular context" (Salfischberger, van de Weerd, & Brinkkemper, 2011). Software variability can be for example the application on different operating systems, where the variability is for example Windows, Mac and Linux. Another example can be different markets where the software can be used. The software product from the example in Figure 3 can be applied in for example the book publishing, magazine publishing or newspaper publishing industry (Salfischberger, van de Weerd, & Brinkkemper, 2011). The functionality and purpose of the software product always stays the same, but product variability allows changes for a specific market.

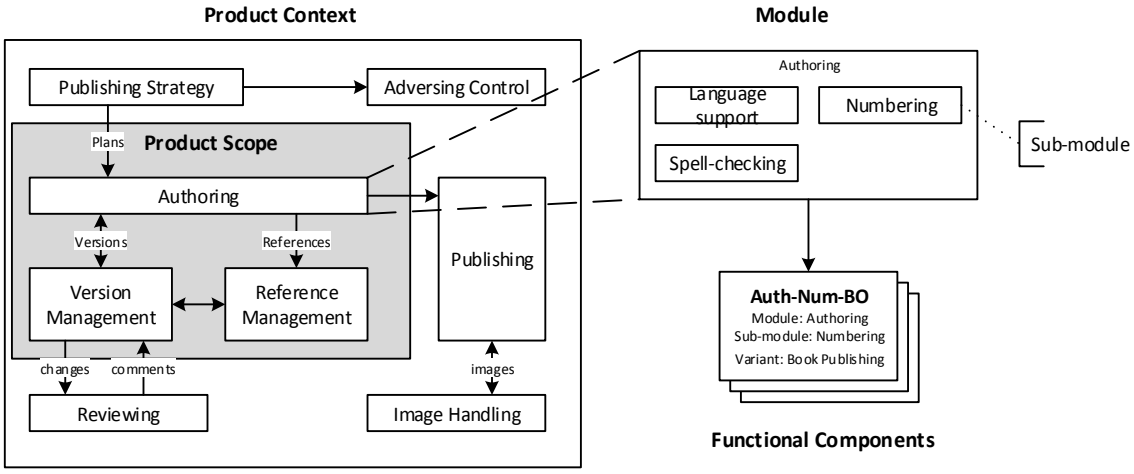


Figure 3: FAM example of a publishing application

A functional component is created for each combination of a sub-module with a product variant. For each combination, a unique code will be created which will be used to identify the requirement in the requirement database.

Table 1 shows the coding of functional components. This example uses a mnemonic coding for the functional components. According to (Salfischberger, van de Weerd, & Brinkkemper, 2011), mnemonic coding contributes to a common language between product managers and developers. Multiple functional components can be assigned to a requirement, so the effect of the requirement to the product is visible.

Module	Sub module	Variant	Functional Component
Version Management	History	Book publishing	VM-Hist-BO
Version Management	History	Magazine publishing	VM-Hist-MA
Version Management	History	Newspaper publishing	VM-Hist-NP
Version Management	Change recording	Book publishing	VM-Chrec-BO

Version Management	Change recording	Magazine publishing	VM-Chrec-MA
Version Management	Change recording	Newspaper publishing	VM-Chrec-NP

Table 1: Example of functional component coding

In Figure 4, a Process Deliverable Diagram (van de Weerd & Brinkkemper, 2008) is shown with the activities of the FAF on the left side, and the corresponding deliverables per activity on the right side. The main deliverable of the FAF is a, by means of a functional component, identified requirement.

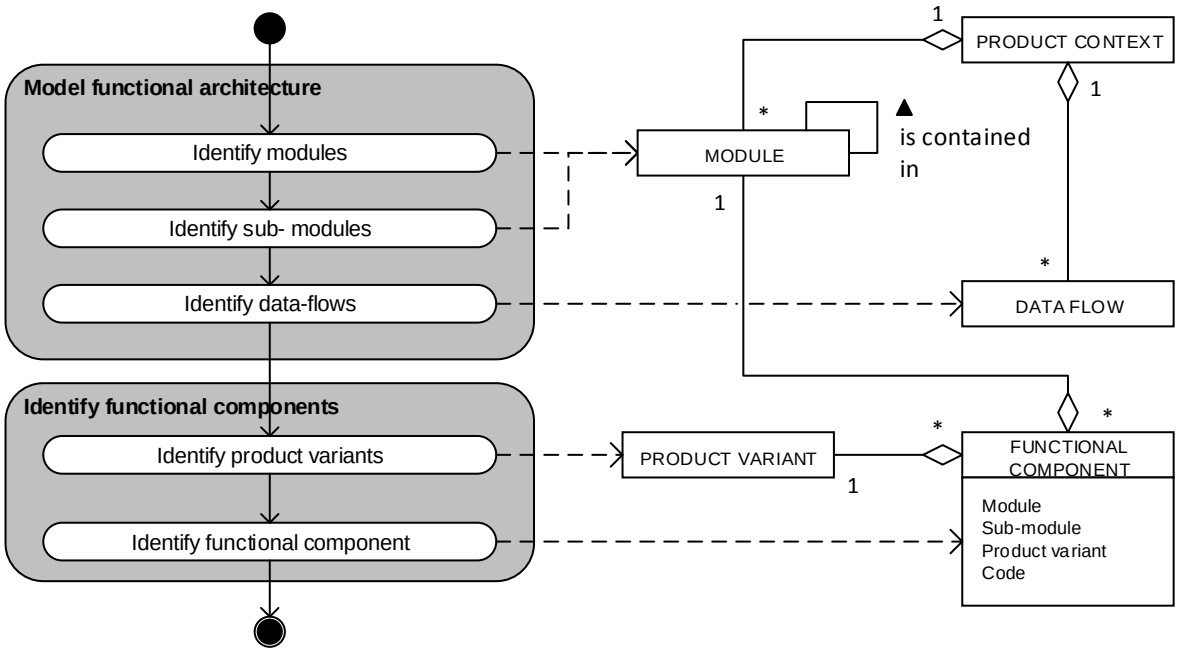


Figure 4: PDD of the Functional Architecture Framework process

1.2. RESEARCH QUESTIONS

The main goal of this research is to find out how the FAF can be used in product software release planning. In order to reach this goal, the next research question has been formulated:

How can the Functional Architecture Framework assist in a product software release planning?

In order to answer this research question, the definition of a product software release planning must be defined. In the previous chapter, the SPM Competence Model by Bekkers et al. (2010) was elaborated. One of the business functions in this model is release planning, and contains 6 focus areas like shown in Figure 5.

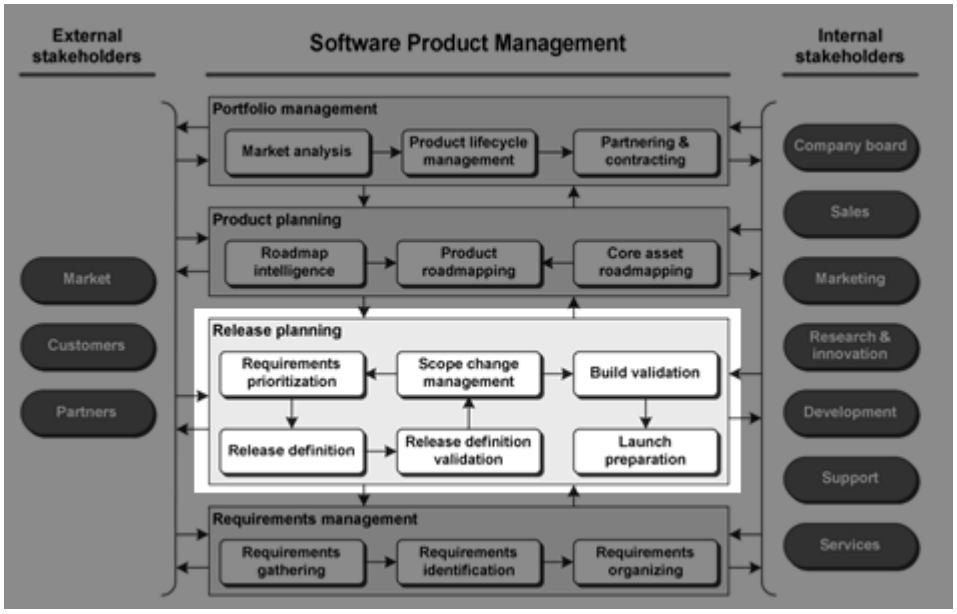


Figure 5: Focus areas of the release planning business function

The focus of release planning in this context is on the *planning of requirements into projects and all direct processes involved in order to create this planning*. Other release planning capabilities like communication, validation, approval, and other supporting capabilities are excluded from this research, since it is very company specific whether these capabilities will be implemented or not. This means that the focus areas requirements prioritization and release definition belong within the scope of this research, while release definition validation, scope change management, build validation and launch preparation are excluded from this research. The definition of a product software release planning in this research can therefore be defined as: *the value weighting (prioritization), cost weighting, and selection process of requirements to be included in (a) future release(s).*

Now the scope of the product software release planning is defined, aspects like activities and concepts of a product software release planning need to be defined. This is necessary in order to know how the FAF can assist in this process. This will be done by answering the first sub-question:

RQ 1 - What are the aspects of a product software release planning?

The expected output of RQ1 is a super product software release planning method, which covers as much release planning capabilities as possible. The super method will be constructed from method increments that were collected from:

- The evaluation of different methods that cover one or more capabilities from the Focus Area Maturity Matrix;
- An observation from the release planning process at a large product software company located in the Netherlands.

When the super product software release planning is developed, the FAF activities and concepts can be compared to the created super method, so the second sub research question can be answered:

RQ 2 - How do the aspects for product software release planning relate to the Functional Architecture Framework?

By answering the second research question, there will be more insight on how the FAF can assist in software release planning. These insights may result in either the FAF being integrated in the release planning method, or in the case that the FAF can't be integrated into the super release planning method, an adapted FAF which can be integrated with the super release planning method.

Based on the identified relation between the product software release planning and the FAF, the FAF assistance can be identified. This will be done through the third and final research question:

RQ 3 - How does the relation between product software release planning and the Functional Architecture Framework (RQ2), contribute to the assistance of the Functional Architecture Framework in product software release planning?

This research question also answers the main research question by describing a product software release planning process which uses the functional components of the FAF.

In Figure 6, the research objective is schematically explained. The main goal is to include the release planning business function into the FAF assistance scope, so software product managers have a clear and mature release planning process.

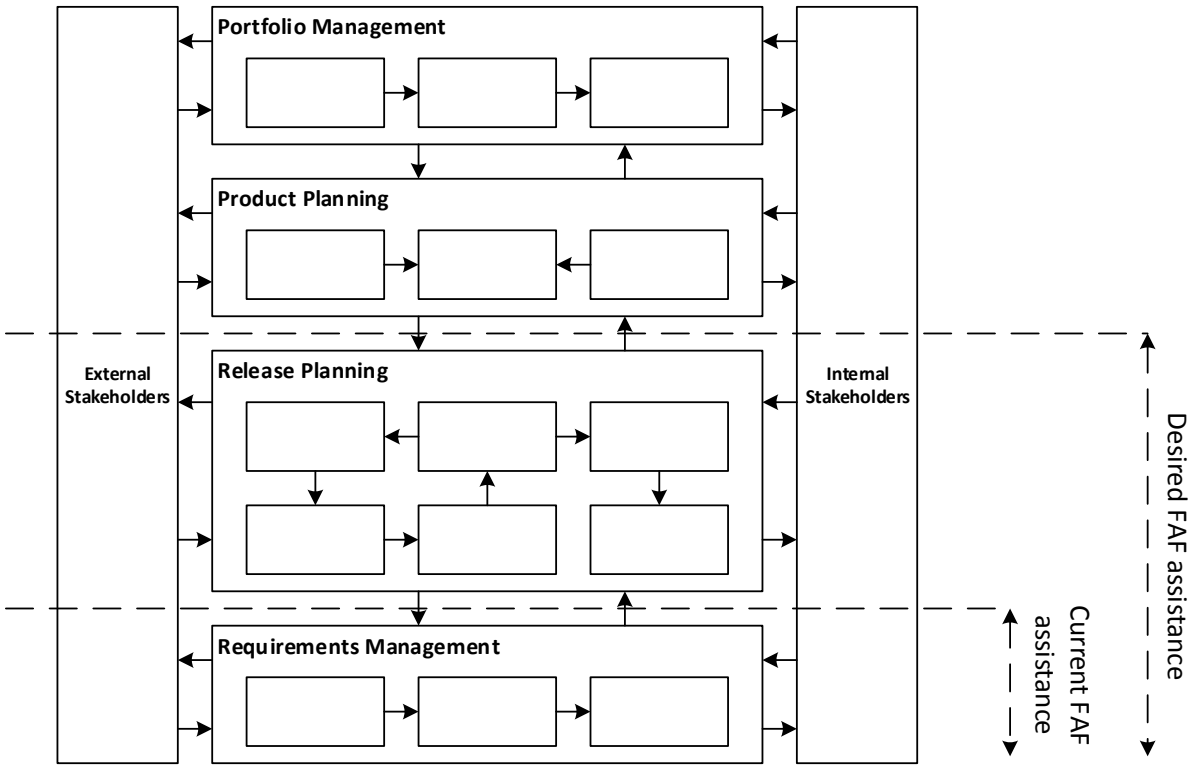


Figure 6: Research objective

1.3. RESEARCH APPROACH

There are two types of research that can be conducted in the information systems (IS) research field; behavioral-science and design-science (Hevner, March, Park, & Ram, 2004). Behavioral-science is mainly about developing theories to explain organizational or human phenomena about the analysis, design, implementation, management, and the use of IS. Design-science on the other hand, is mainly about creating artifacts (models, methodologies, constructs or instantiations) which optimize the analysis, design, implementation, management, and the use of IS (Hevner, March, Park, & Ram, 2004).

The goal of this research is to develop a release planning process which can be supported by the FAF. During this research, a super release planning method is constructed based on different sub-methods. The FAF (in its current state or in an evaluated state) will also be implemented into the supermethod. The main type of science that will be applied in this research is therefore design science. During this research, the release planning process of a large ERP vendor will be analyzed, so behavioral science does appear in this research, but the main artifact is created through design science.

Hevner et al. (2004) designed an IS research framework in order to give a formal body to the IS research. Figure 7 shows an instantiation of this model, where factors are added that are case specific to this research. The research has input from- and output to both the scientific domain and the business domain. In the right side of the figure, the theoretical foundation that are used are shown. The left side of the figure shows the involvement of AFAS, the case company which will be described later on in this chapter. The research itself is shown in the middle column. The research method that is used in this research is Design Research.

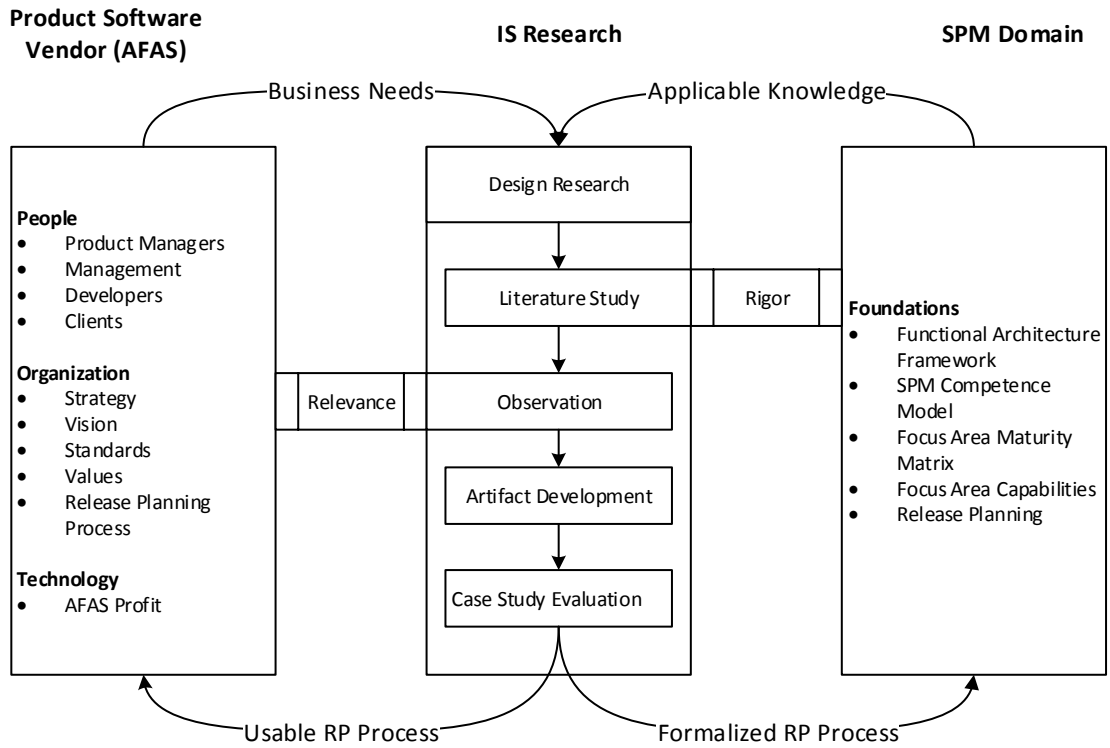


Figure 7: Research specific instantiation of the IS Research Framework

A generic release planning method, and possibly, an improved FAF are the final deliverables in this research. According to Hevner et al. (2004), the research must make a valuable contribution to both the business and the scientific domain. The (expected) contribution to the business domain is a release

planning method which assists companies in managing their software release planning processes. The expected contribution to the scientific domain is:

- An evaluation of the current methods covering the requirements prioritization and release definition focus areas;
- A release planning method, supported by the FAF, which covers as much capabilities as possible within the requirements prioritization and release definition focus areas.

Business Environment

To create and maintain focus on solving a relevant business need, this research will be performed at AFAS software, an ERP vendor with over 10.000 customers which operates in the Netherlands, Belgium and the Caribbean. The ERP software that AFAS sells is a software product.

The product management department of AFAS maintains a constant update cycle (at fixed times). Due to the fact that they sell product software, they receive so many software requirements, that it is impossible to develop them all in releases. Currently, AFAS has not adapted a formalized method to determine what (not) to include in each software release. The results of this research will be applied and validated at AFAS to ensure that this research will contribute to a relevant business need, which is one of the rules of IS research (Hevner, March, Park, & Ram, 2004).

1.3.1. RESEARCH METHODS

In this chapter, the methods used for this research are described. The methods that are used for this research are (like shown in Figure 7):

- Literature study;
- Observation;
- Artifact development;
- Case study evaluation.

The methods used for this research are explained in the next four sub-chapters.

Literature study

The background foundations for the super method are collected through a literature study. During this literature study, the necessary release planning capabilities were analyzed, whereupon methods and techniques were searched in order to cover these capabilities.

The methods and techniques were originally derived from the papers of (Karlsson, Wohlin, & Regnell, An evaluation of methods for prioritizing software requirements, 1997) and (Carlshamre P. , 2002), as they were mentioned as basic literature for release planning in the Utrecht University SPM course (Utrecht University, 2013). Based on the methods and techniques found in these papers, new and improved methods were searched. This is done based on the references and cited of these papers.

Another source that is used is a website published by Utrecht University, which is made to provoke an understanding of the SPM domain (Utrecht University, 2013). This website contains several potential useful methods that were not mentioned in the papers of Karlsson et al. (1997) and Carlshamre (2002). These methods were used as keywords to find scientific literature about these methods.

The methods and techniques that are found were modeled by the format of a Process-Deliverable Diagram (PDD) (van de Weerd & Brinkkemper, 2008). This format will be used to model all methods on the same

abstraction level, which is useful later on in the research when methods will be compared and combined to create a super release planning method.

Observation

The current release planning process of AFAS is being modeled through participation in the product management processes. The goal of this participation is to derive aspects and insights that contribute to the business relevancy of the artifact. The observation was done at AFAS Software in the Netherlands. General details about this company were given in the previous chapter.

Artifact development

The methods found in the literature study and the methods derived from observation are at this stage just some methods on a big pile. In order to construct an useful artifact out of it, all methods and method fragments (parts of a method) need to be evaluated, based on their usefulness. The usefulness is determined by the:

- Extent that the method supports requirements management and release definition capabilities;
- Extent that experts (in this case product managers from AFAS) find the method useful in the business environment (an example could be the time to execute the method, if the method takes too long to execute in the product manager's view, the method hasn't got a good fit for the business environment);

Based on this evaluation, a super release planning method (van de Weerd & Brinkkemper, 2008) will be constructed that has the highest release planning capability coverage. The activities and concepts of the super method are then compared to the FAF in order to identify the parts of the super method where the FAF can assist. If the FAF needs to be adapted in order to optimize the assistance in the super method, an adapted FAF also belong within the scope of the artifact development. The only requirements for the FAF adaption is that the requirements management processes remain the way they are (so the FAF can only be extended).

Case study validation

When the construction of super release planning method is completed, this method has to be validated in the business environment. A prerequisite for the validation is the FAF to be implemented in the case study company (since the goal is to test the FAF assistance in release planning). The validation will be done through a single case study at AFAS Software. The choice for a single case study has been made based on the perceived effort to apply the FAF and the new release planning method in the case study company, in combination with the available time for this research.

The expectation is that the case study will produce a realistic release planning. To test if the release planning is useful, it will be validated through an expert review. In this research, the expert roles are filled in by a number of product managers at AFAS Software, who will determine, based on their experience and knowledge of release planning process, if the created release planning method is realistic and if it actually has added value over the traditional release planning process.

1.3.2. DELIVERABLES

In order to develop the super release planning method, a number of activities need to be performed and sub-artifacts need to be developed. In order to show the full research process and deliverables, a PDD of the research process is modeled (Figure 8).

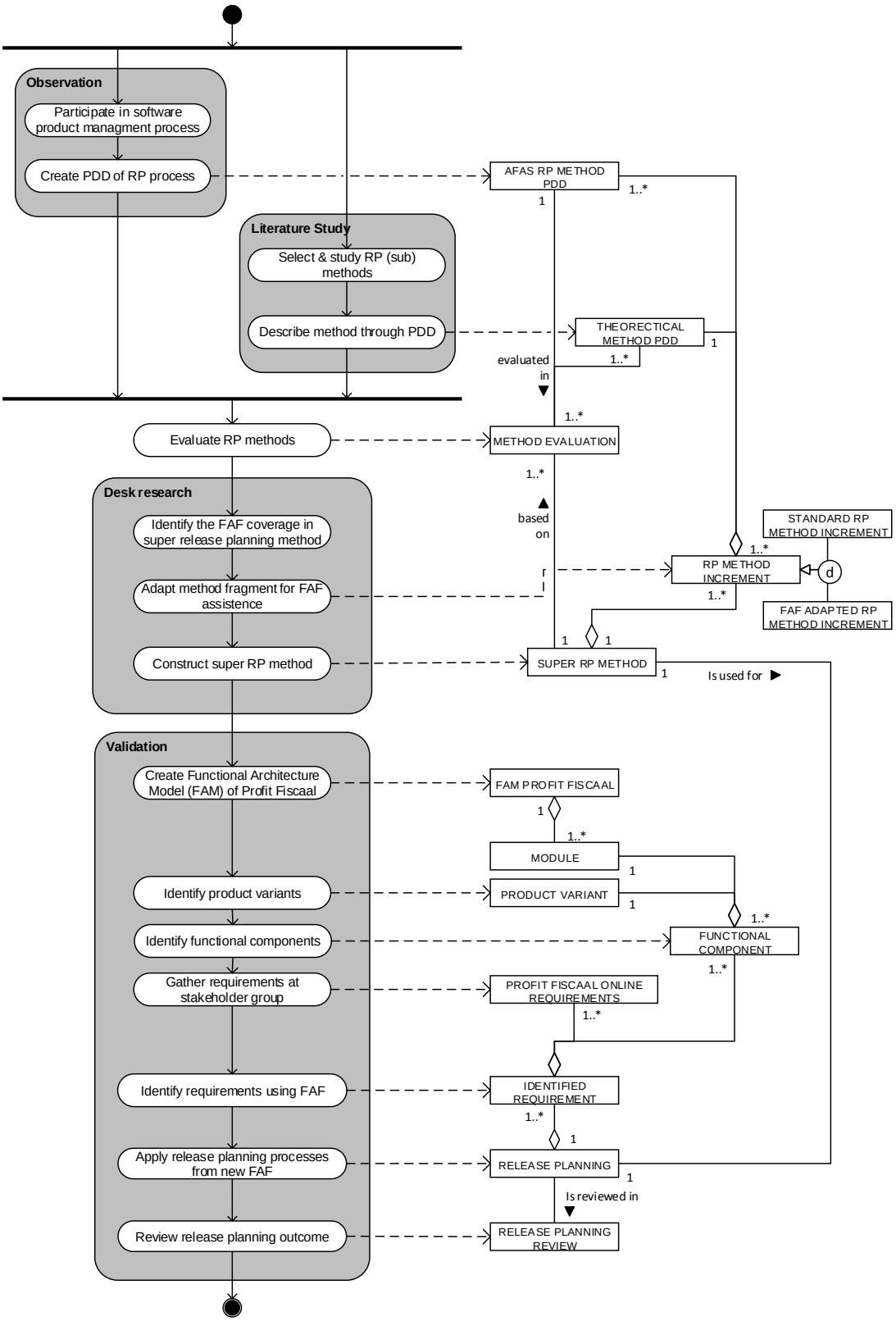


Figure 8: PDD of the full research process

The left side of the PDD shows the processes that was ran through during the research, and the right side shows the deliverables forthcoming from these processes. In Table 2, the deliverables/concepts that are produced during this research are described.

Research method	Concept	Description
Observation	AFAS RP METHOD	A PDD of the current release planning process at the case study company
Literature study	THEORETICAL METHOD PDD	PDD's of the requirements prioritization and release planning methods found in the scientific literature
Artifact development	METHOD EVALUATION	An evaluation of the AFAS RP METHOD and the different THEORETICAL METHOD PDD'S
Artifact development	RP METHOD INCREMENT	A collection of activities and concepts from the AFAS RP METHOD and the THEORETICAL METHOD PDD
Artifact development	SUPER RP METHOD	An optimal release planning PDD constructed from different RP METHOD INCREMENTS
Case study validation	FAM PROFIT FISCAAL	A functional architecture model of the AFAS Profit Fiscaal product
Case study validation	MODULE	A module of the AFAS Profit Fiscaal product
Case study validation	PRODUCT VARIANT	The identified product variants for AFAS Profit Fiscaal
Case study validation	FUNCTIONAL COMPONENT	The set of identified functional components for AFAS Profit Fiscaal
Case study validation	PROFIT FISCAAL ONLINE REQUIREMENTS	A list of the gathered AFAS Profit Fiscaal Online requirements
Case study validation	IDENTIFIED REQUIREMENTS	A requirement from the PROFIT FISCAAL ONLINE REQUIREMENTS, which is labeled with one or more FUNCTIONAL COMPONENT
Case study validation	RELEASE PLANNING	An overview of which IDENTIFIED REQUIREMENTS will be developed in which release. This concept will be produced through the application of the SUPER RP METHOD
Case study validation	RELEASE PLANNING REVIEW	A review of the RELEASE PLANNING by a software product manager at AFAS Software.

Table 2: The artifacts that will be produced during this research

2. OVERVIEW OF RELEASE PLANNING METHODS

In order to create a generic release planning method which is compatible with the FAF, a theoretical optimal release planning should be created (a method that covers as much release planning aspects as possible). Later on, this optimal will be adapted based on insights from the business environment, but this chapter focuses merely on the theoretical adequacy of the method.

The –to be constructed- super method is based on the focus areas requirements prioritization and release definition. The focus area of release definition could be split up in two parts; cost estimation and requirement selection. Therefore, this chapter is divided three sections in which a literature study is performed to gain more insights about the scientific background in the topic of the section.

- The first section focuses on the prioritization of requirements (chapter 2.1);
- The second section focuses on the cost estimation of requirements (chapter 2.2);
- The third section focuses on the selection process of requirements for a release (chapter 2.3).

2.1. REQUIREMENTS PRIORITIZATION

The need for requirements prioritization is not limited to the software industry. Requirements prioritization is an activity that always needs to be done when developing complex products for a large market segment (Hauser & Clausing, The House of Quality, 1988). Requirements prioritization is the first focus area in the release planning process.

Product software vendors usually receive more requirements than can be realized within the available cost and time (Berander & Andrews, 2005). Therefore, it is important for the software product manager to properly prioritize the requirements, so that the optimal set of requirements can be selected to develop for the next release.

Over the last years, numbers of prioritization methods have been developed. Most prioritization techniques are not especially designed for software requirements prioritization, but are very generalizable and applicable in multiple industries (Berander & Andrews, 2005) (Karlsson & Ryan, A cost-value approach for prioritizing requirements, 1997) (Saaty, 1990).

In the next chapters, a number of popular requirement prioritization methods are presented. The initial set of prioritization methods are derived from (van de Weerd & Brinkkemper, 2010) and (Utrecht University, 2013). The papers about the requirement prioritization methods are gathered through Google Scholar and Omega (Utrecht University). Based on the contents of the literature, other prioritization methods were found. In total, 9 prioritization methods were studies. The prioritization methods that were studied are divided into 3 different prioritization categories:

1. Pairwise comparison (chapter 2.1.1);
2. Numerical assignment (chapter 2.1.2);
3. Multi-dimensional (chapter 2.1.3).

2.1.1. PAIRWISE COMPARISON

During the literature study, three types of pairwise comparison prioritization methods were found:

- Analytical Hierarchy Process (AHP);
- Cost/Value approach (variant of AHP),
- Binary Search Tree.

In this chapter the Analytical Hierarchy Process is elaborated, since this method is also applied in the Cost/Value approach and is far more extensive and probably useful than the Binary Search Tree. The Cost/Value approach and the Binary Search Tree are described in short.

2.1.1.1. ANALYTICAL HIERARCHY PROCESS

The Analytical Hierarchy Process (AHP) is an extensive prioritization method that uses pairwise comparison of requirements, to eventually assign a certain priority on ratio scale to each requirement (Karlsson & Ryan, A cost-value approach for prioritizing requirements, 1997) (Saaty, 1990) (Karlsson, Berander, Regnell, & Wohlin, 2004) (Ruhe, Eberlein, & Pfahl, Trade-off Analysis for Requirements Selection, 2003) (Karlsson, Wohlin, & Regnell, An evaluation of methods for prioritizing software requirements, 1997) (Racheva, Davena, & Buglione, 2008). The AHP method exist of 3 steps:

- Performing pairwise comparisons
- Calculating priority vector per requirement
- Calculating consistency index of the process

Performing pairwise comparisons

For each pairwise comparison, the product manager assigns a value to both requirements. If the requirements are of equal value, both requirements get 1 point. If for example requirement 1 (r1) is slightly more important than requirement 2 (r2), r1 gets 3 points, where r2 gets $^1/_3$ point. The highest score that can be given to a requirement is 9 point, while the other requirement from the pairwise comparison gets the lowest possible score $^1/_9$ point.

In Table 3, an example is shown of a 4*4 matrix, where 6 pairwise comparisons are made (there are 6 unique pairs of requirements). Note that the comparison of identical requirements always get the value 1, and that the diagonally mirrored cells are the opposite value.

	R1	R2	R3	R4
R1	1	0.33	2	4
R2	3	1	5	3
R3	0.50	0.20	1	0.33
R4	0.25	0.33	3	1
Sum	*4.75*	*1.86*	*11*	*8.33*

Table 3: AHP ordinal result table

Calculating priority vector per requirement

The next step of the AHP process is to calculate the priority vector of each requirement. It is therefore necessary to sum the values of each column, and divide the value of each cell in that column by the column sum. Each cell had now a ratio score, which can be checked by summing the cell values in the columns (which should be 1 for each column).

After that, the values of each row must be summed up like the example in Table 4. The number of each row must then be multiplied by $^1/_n$, in the example case $^1/_4$ (0.25). The outcome variable is the priority vector of the specific requirement. So, in the example case it will be for R1 = $0.25 \cdot 1.05 = 0.26$, which means that R1 has 26% value of the total value.

	R1	R2	R3	R4	*Sum*
R1	0.21	0.18	0.18	0.48	*1.05*
R2	0.63	0.54	0.45	0.36	*1.98*
R3	0.11	0.11	0.09	0.04	*0.34*
R4	0.05	0.18	0.27	0.12	*0.62*

Table 4: AHP ratio result table

Calculating consistency index of the process

A strong feature of AHP is the consistency index (CI), which calculates the internal consistency of the pairwise comparison, in order to minimize misjudgments. CI is calculated as $CI = \frac{\lambda \max - n}{n-1}$. The Eigenvalue ($\lambda \max$) is calculated by multiplying the priority vector value of a requirement with the ordinal column total of the same requirement. So, λmax for the above example is:

$$\lambda max = (0.26)(4.75) + (0.50)(1.86) + (0.09)(11) + (0.16)(8.33) = 4.48$$

This means that $CI = \frac{4.48-4}{4-1} = 0.16$. In order to find out if 0.16 is acceptable, the consistency ratio ($CR = \frac{CI}{RI}$) must be calculated. To find RI, the random consistency index (Table 5) can be used (Saaty, 1990).

N	1	2	3	4	5	6	7	8	9
RI	0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45

Table 5: Random consistency index

Since we use four requirements in our example, we set RI as 0.90. The consistency ratio is now $CR = \frac{0.16}{0.9} = 0.17$. A value under 0.10 is considered consistent, but for example Karlsson and Ryan (1997) say that a CR between 0.1 and 0.20 is also acceptable.

The PDD describing the activities and deliverables of the AHP method is shown in Figure 9.

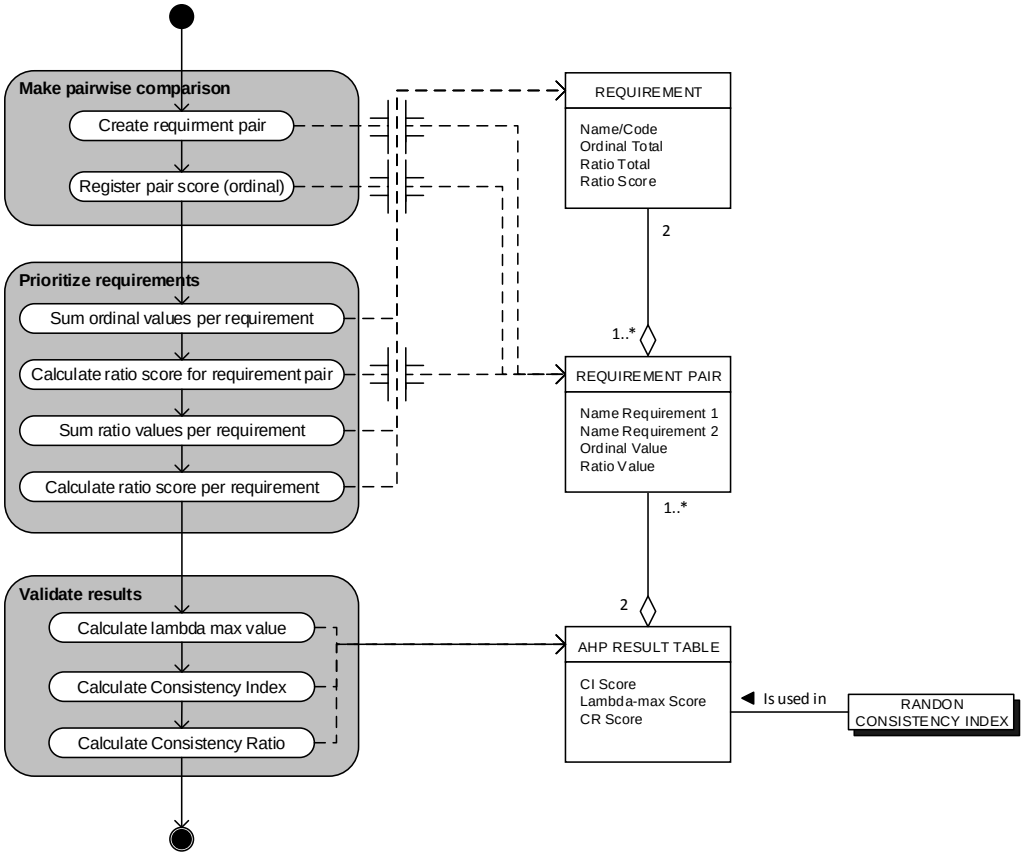


Figure 9: PDD of the Analytical Hierarchy Process

2.1.1.2. COST/VALUE APPROACH

A variant on the AHP method is the Cost/Value approach, where the AHP cycle is performed twice: once for determining the value of a requirement (like AHP), and once for determining the (relative) cost of a requirement. If both values are known for all requirements, they can be plotted in a graph where the cost is represented on the x-axis, and the value on the y-axis (Figure 10). This plot can help the product manager to see which requirements has the highest return on investment (Karlsson & Ryan, A cost-value approach for prioritizing requirements, 1997).

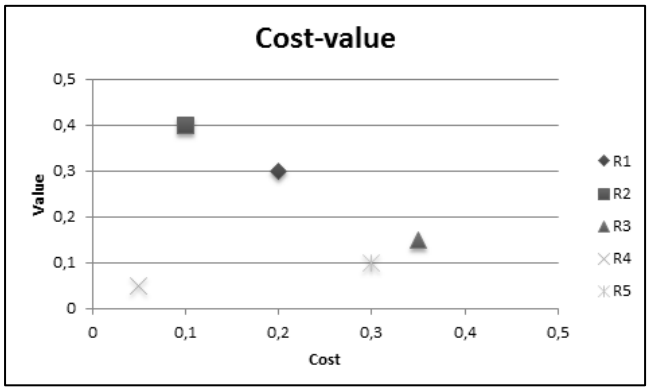


Figure 10: Example of the cost-value graphical plot

2.1.1.3. BINARY SEARCH TREE

A method which is frequently mentioned in literature, is the Binary Search Tree (Karlsson, Wohlin, & Regnell, An evaluation of methods for prioritizing software requirements, 1997) (Karlsson, Berander, Regnell, & Wohlin, 2004) (Bentley, 1975) (Bebensee, van de Weerd, & Brinkkemper, 2010), which is a method that allows to prioritize requirements without comparing them to all other requirements (like AHP).

In this method, the practitioner takes a random requirement from the requirement database and puts it in the top node of a tree. Then, a second requirements is picked and is placed as a child node to the tree. If the requirement is more valuable than the parent node, the requirement should be placed as a right-hand child node. If the requirement is less valuable than the parent node, it should be placed on as a left-hand child node.

When there is already a child node placed under the top node, this child node is considered the top node, and another comparison will be made. Figure 11 shows an example of the Binary Search Tree, with the numbers representing the value of a node.

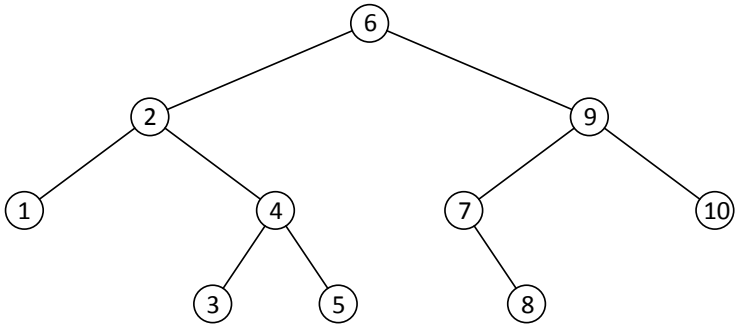


Figure 11: Example of the Binary Search Tree

2.1.2. ONE-DIMENSIONAL REQUIREMENT PRIORITIZATION

The one-dimensional category consist of methods which use a single ordinal or ratio value for requirements prioritization. Compared to the other two categories, these methods are rather simple, but quick to apply. In this chapter, the hundred dollar prioritization method is fully elaborated through a PDD. The other methods in this chapter (numerical assignment and the top-ten method) are described globally, since they are quite similar to the hundred dollar method.

2.1.2.1. HUNDRED-DOLLAR

In the hundred dollar method (also known as cumulative voting) (Berander & Andrews, 2005) (Leffingwell & Widrig, 2000) (Racheva, Davena, & Buglione, 2008), different stakeholders get a number of points (usually 100) to divide over each candidate requirement. The total points per requirement shows, in ratio scale, which requirements are preferred and which not.

An example of the hundred dollar method is shown in Table 6, and the PDD is shown in Figure 12.

	Stakeholder 1	Stakeholder 2	Stakeholder 3	**Total**
R1	40	25	10	**75**
R2	20	25	0	**45**
R3	20	25	80	**125**
R4	20	25	10	**55**
SUM	**100**	**100**	**100**	

Table 6: Hundred-test example

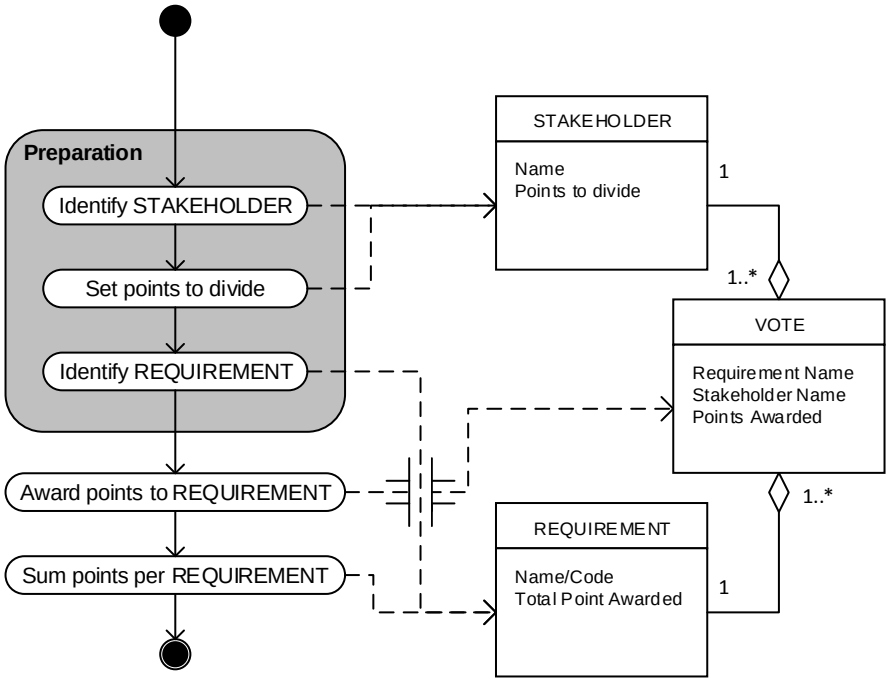


Figure 12: PDD of the hundred-dollar test

2.1.2.2. TOP-TEN

The top-ten method is a simple prioritization method where each stakeholder select their own top ten list from a pile of requirements. Based on the average score, the product manager can make a list of the most valuable requirements (Berander & Andrews, 2005).

Compared to the hundred dollar method, the top ten method is very limited. The hundred dollar test allows more than 10 requirements to be prioritized, and gives a ratio score for each requirement (where the top ten method gives an ordinal score).

2.1.2.3. NUMERICAL ASSIGNMENT

Another method similar to the hundred dollar test is the numerical assignment method (Berander & Andrews, 2005), which is an ordinal prioritization method, where the stakeholder assigns the requirement into a priority group (e.g. 1 to 5, where 1 is not important and 5 is critical). A popular variant is the MoSCoW technique, where stakeholders can categorize the requirements into four groups: must have, should have, could have and would have.

2.1.3. MULTI-DIMENSIONAL REQUIREMENT PRIORITIZATION

The multi-dimensional prioritization methods are methods that use multiple viewpoints to determine the priority of a requirement (where one-dimensional only uses perceived value), and are therefore more complex than one-dimensional prioritization methods. In this chapter, three different multi-dimensional methods are elaborated.

2.1.3.1. EVOLVE

EVOLVE is a relatively new and extensive method, which has numerical assignment as base method. The EVOLVE method is a hybrid release planning method which uses both absolute and ratio scale. The hybrid release planning model identifies itself through multiple release planning support (Ruhe & Saliu, The Art and Science of Software Release Planning, 2005) (Saliu & Ruhe, 2005).

This method starts with determining which resource constraints are necessary. These constraints can be for example man-hours of a development department or budget per development department in euros. There is no maximum number of resource constrains. The product manager must determine the available resources per constraint for the first and second release, and must determine the weight for release 1, release 2 or a further release (postponing). The weight of a release can be a value between 0.00 and 1.00, but the sum of the three release weights must be 1.00. The product manager also need to select the stakeholders which will be included into the prioritization process. The product manager can assign an importance weight to each stakeholder. This weight can vary from 1 (very low importance) to 9 (very high importance).

For each requirement, the product manager determines the required resources. Each stakeholder rates the requirement with a value from 1 (very unimportant) to 9 (very important). The stakeholder also determines the urgency of the requirement. Per requirement, the stakeholder has 9 points to divide over release 1, release 2 or a further release. The division of point must be written as (r1, r2, r3), where r1 is release 1, r2 is release 2, and r3 is postponing the requirement. If for example a stakeholder divide the urgency points as (3,3,3), that means that the stakeholder has no preference for the first release, second release or postponing the requirement. If the stakeholder for example divide the urgency as (9,0,0), the stakeholder finds it very important to have the requirement in the first coming release. An example of the data structure is shown in Table 7.

		Resources			Stakeholder S(1)		Stakeholder S(2)	
		Designers (hours)	Developers (hours)	Budget (€ '000)	Value	Urgency	Value	Urgency
ID	Requirement							
1	Example 1	200	300	200	6	(5,4,0)	6	(0,9,0)
2	Example 2	150	250	100	4	(5,0,4)	3	(0,8,1)
3	Example 3	100	200	200	5	(9,0,0)	4	(2,4,3)
4	Example 4	50	150	100	7	(7,2,0)	9	(2,7,0)
	Total	**300**	**900**	**600**				
Capacity rel1		**260**	**750**	**400**				
Capacity rel2		**200**	**500**	**400**				

Table 7: Hybrid release planning example

As all the values are known, the calculation can start. The goal is to find the weighted average satisfaction (WAS) per requirement and per release. The WAS is a score for one requirement in one particular release (so eventually you have three different WAS values for one requirement).

$$WAS = release\ weight \cdot \big[(weight\ S(1) \cdot value\ S(1) \cdot urgency\ S(1)) + \big(weight\ S(2) \cdot value\ S(2) \cdot urgency\ S(2)\big) + (weight\ S(n) \cdot value\ S(n) \cdot urgency\ S(n))\big]$$

If we use the example of Table 7, and the weight of release 1 = 0.7, the weight of S(1) = 5, and the weight of S(2) = 4, we calculate WAS for requirement 1 in release 1 as:

$$WAS = 0.7 \cdot [(5 \cdot 6 \cdot 5) + (4 \cdot 6 \cdot 0)] = 105$$

If the weight of release 2 = 0.3, the WAS of release 2 is:

$$WAS = 0.3 \cdot [(5 \cdot 6 \cdot 4) + (4 \cdot 6 \cdot 9)] = 100.8$$

So even though S(2) has indicated that they strongly prefer to have the requirement in release 2, release 1 has a higher WAS, which indicates that overall, the stakeholders prefer the requirement in release 1.

When all the WAS values are calculated (three WAS values for each requirement). The product manager applies mathematical optimization to calculate the ideal planning of minimizing the total cost and maximizing the total benefit (WAS) for each release. The PDD for the EVOLVE method is shown in Figure 13.

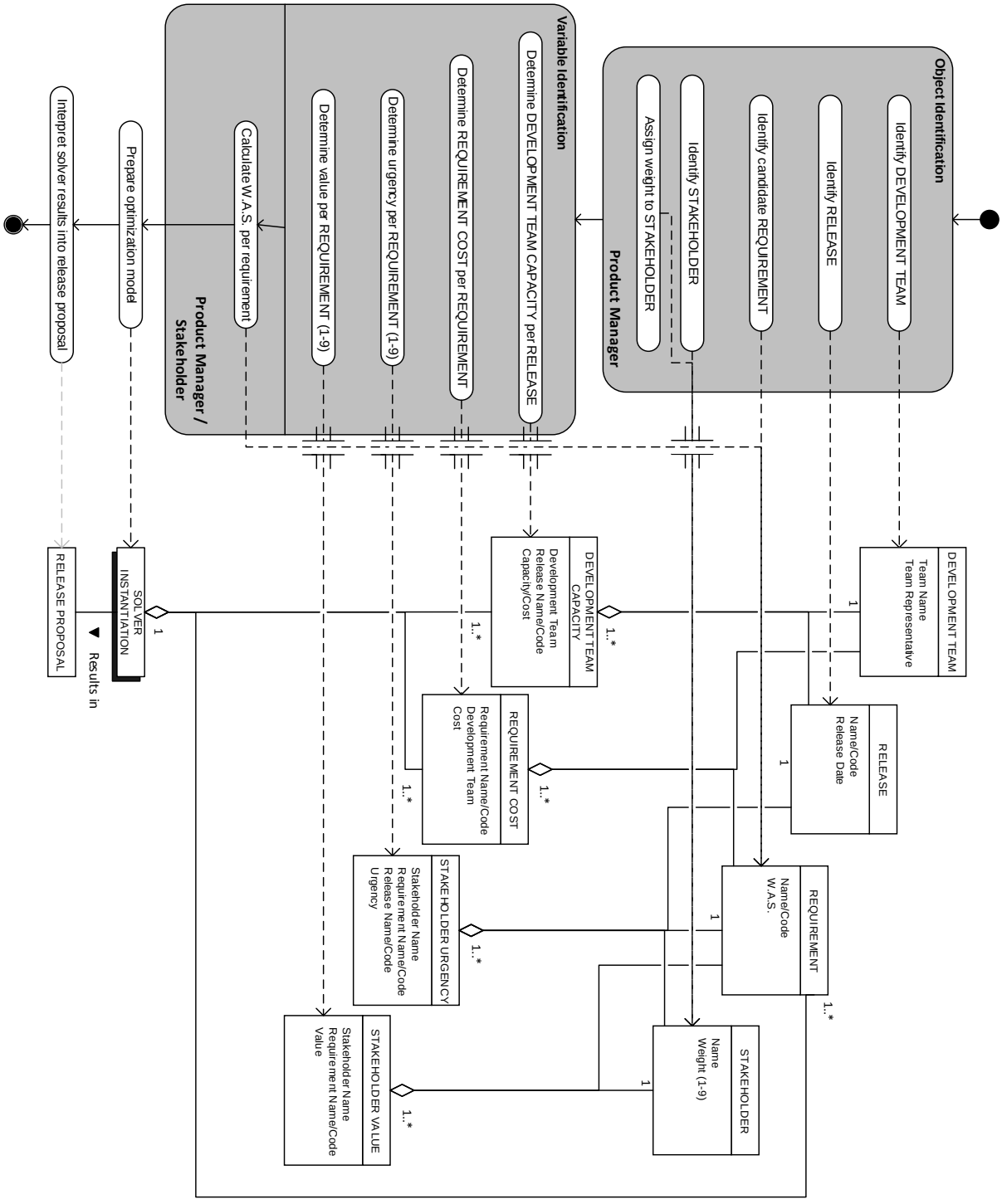


Figure 13: PDD of the EVOLVE method

2.1.3.2. FEATURES PRIORITIZATION MATRIX

The features prioritization matrix uses ordinal and ratio measurements to calculate the priority of a requirement (Wiegers, Software requirements : practical techniques for gathering and managing requirements throughout the product development cycle, 2003) (Berander & Andrews, 2005) (Racheva, Davena, & Buglione, 2008). The product manager must determine the relative benefit, relative penalty (impact when NOT implementing the requirement), relative cost and relative risk of the requirement. These four factors are determined on an ordinal scale from 1 (very low) tot 9 (very high).

For each of these four factors, the product manager must also determine the weight. For example, if the benefit of the requirement must count heavier than the other factors, and risk must count less than the other factors, the product manager can set the weight of the benefit to 2, and the weight of risk to 0.5.

For each requirement, the product manager calculates the ratio value, ratio cost, and ratio risk

$$\boldsymbol{ratio\ value} = 100 \cdot \left(\frac{total\ value\ =\ (relative\ benefit\ \cdot\ benefit\ weight) + (relative\ penalty\ \cdot\ penalty\ weight)}{total\ value(1) + total\ value(2) + \cdots + total\ value(n)}\right)$$

$$\boldsymbol{ratio\ cost} = 100 \cdot \left(\frac{relative\ cost}{relative\ cost(1) + relative\ cost(2) + \cdots + relative\ cost(n)}\right)$$

$$\boldsymbol{ratio\ risk} = 100 \cdot \left(\frac{relative\ risk}{relative\ risk(1) + relative\ risk(2) + \cdots + relative\ \ risk(n)}\right)$$

After all variables are known, the product manager can calculate the priority per requirement. The priority is calculated as:

$$\boldsymbol{priority} = \frac{ratio\ value}{[(ratio\ cost\ \cdot\ cost\ weight) + (ratio\ risk\ \cdot\ risk\ weight)]}$$

An example of the application of the Features Prioritization Matrix is shown in Table 8, the PDD of the Features Prioritization Matrix is shown in Figure 14.

Weight	2	1		1		0.5		
	Relative benefit	Relative penalty	Ratio value	Relative cost	Ratio cost	Relative risk	Ratio risk	Priority
Requirement1	2	7	14	4	24	2	8	0,5
Requirement2	5	2	15	3	18	3	12	0,62
Requirement3	8	3	24	5	29	5	20	0,61
Requirement4	9	4	28	2	12	7	28	1,07
Requirement5	5	4	18	3	18	8	32	0,52
Total	29	20		17		25		

Table 8: Example of the Features Prioritization Matrix

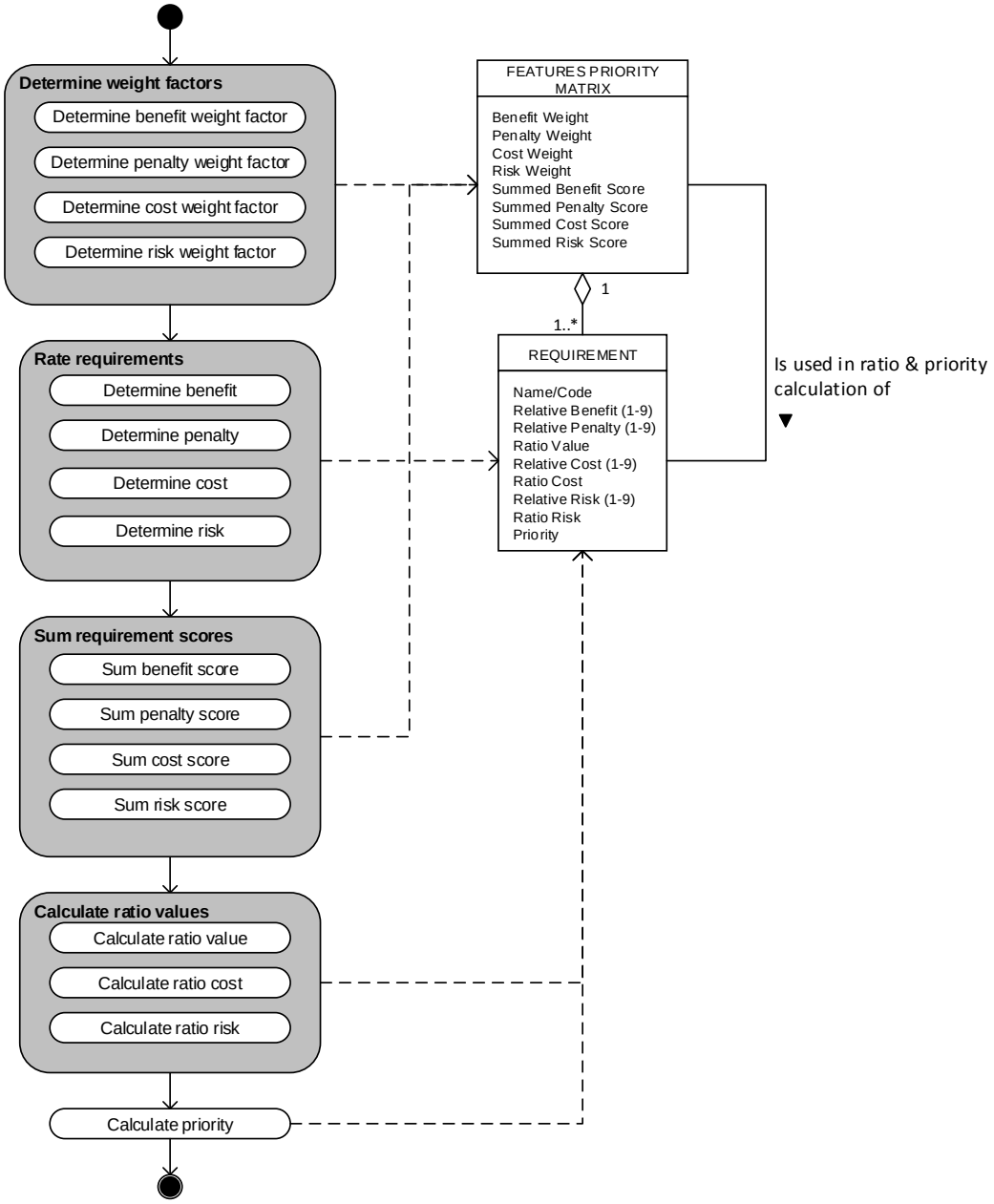


Figure 14: PDD of the Features Prioritization Matrix

2.1.3.3. QUALITY FUNCTION DEPLOYMENT

The Quality Function Deployment (QFD) is an extensive method that includes customer requirements and quality requirements. The model that the QFD produces is called the House of Quality (Hauser & Clausing, The House of Quality, 1988) (Racheva, Davena, & Buglione, 2008).

In the scientific literature, there are multiple variants of the QFD method. The QFD that will be briefly elaborated in this chapter is based on the version described by Hauser and Clausing (1988), because this version is published in a high quality journal (Harvard Business Review), and is cited nearly 2900 times at the moment of writing this chapter.

The House of Quality is a multi-dimensional schema to compare requirements in five dimensions. The first comparison dimension is the functional requirement against the quality requirement. A functional

requirement can contribute to a certain quality requirement, work against a certain quality requirement or is neutral to a quality requirement. The second dimension is the comparison of each quality requirement against each other quality requirement, because it can occur that a quality requirement improves or blocks another quality requirement. The third dimension is the comparison of each functional requirement with the current product, and the product of the main competitors. In this comparison, stakeholders can assign a score (1 to 5) to the presence of the functional requirement in the current version of the product, and the products of two or three main competitors. This leads to an overview of the current market position per functional requirement. The fifth dimension is the comparison of the quality requirements between the current product and the product of the main competitors. This comparison is done by measuring with absolute values. For example, if there is a quality requirement of lowering the loading time of an application, the actual loading time of the current product need to be measured, but also the loading time of the competitor product need to be retrieved.

Based on the quality of the competitor products, a goal need to be set for each quality requirement in order to create or maintain competitive advantage. For each quality requirement target, the technical difficulty end the estimated cost must be determined (on ordinal scale 1 to 10). Then, the product manager searches for the functional requirements where the current product has the most disadvantage against the competitor product, the product manager then selects the quality requirements that contributes best with the functional requirement. These quality requirements get a high priority, while quality requirements that are linked to functional requirement where the current product scores better than the competitor get a lower priority.

2.2. DEVELOPMENT EFFORT ESTIMATION

In the previous chapter, requirement prioritization techniques were discussed. Prioritization of requirements is necessary in order to assign an added value to each requirement. In order to select requirements for a specific release, the magnitude of each requirement must be defined and necessary (development) effort/cost must be linked to the requirements.

There are multiple methods to allocate resources to a development project (Briand & Wieczorek, 2002). Literature in the software project cost estimation domain distinct five categories of methods (Boehm, Abts, & Chulani, 2000):

- Model based; cost estimation based on static models, situational variables, and predefined coefficients.
- Expertise based; cost estimation based on views, opinions and experiences from field experts.
- Learning based; cost estimation based on historical data of similar projects.
- Dynamics based; cost estimation based on the assumption that the factors used will change over time.
- Regression based; cost estimation based on statistical linear regression.

Since there are many different software development cost estimation methods available in literature, a complete evaluation of all those would exceed scope and time limits for this research. Therefore, a selection is made of three different methods, each from a different category. The selected software development cost estimation methods are:

- COCOMO (model based);
- Wideband Delphi (expertise based);
- PROBE (learning based).

2.2.1. MODEL BASED ESTIMATION

Model based estimation methods have a high degree of formalization. In model based estimation methods, development effort is calculated through a set of predefined factors .

A model based software development cost estimation methods is for example the COCOMO method (Boehm, et al., 1995) (Clark, Devnani-Chulani, & Boehm, 1998). The COCOMO method calculates the required person months, required number of people, and the development time in months to develop a software product (Pressman & Ince, 1992). There are three kinds of COCOMO models: basic, intermediate, and expert. Due to the level of detail of the expert COCOMO, this chapter only elaborates on the basic and intermediate COCOMO.

The COCOMO method distinct three kinds of software development projects:

1. Organic; this project is considered easy and routine
2. Embedded; this project is considered difficult and complex
3. Semi-detached; this project is considered difficult, but there is some experience in handling this kind of projects

Basic COCOMO

The main input variables for the basic COCOMO are an estimation of the number of thousands lines of code (KLOC) that need to be developed and four predefined variables that are derived from the COCOMO literature (Pressman & Ince, 1992) (Boehm, et al., 1995). The four variables differ per type of software development project. Table 9 shows the four variables per project type.

Project type	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

Table 9: Variables for the basic COCOMO method

The required development effort in person months (E) can be calculated through the formula $E = a(KLOC)^b$. The development time in months (D) can be calculated through $D = c(E)^d$. The people required (P) for a development for a software development project can be calculated through the formula $P = E/D$.

Intermediate COCOMO

Like basic COCOMO, the intermediate variant uses the KLOC and the four (slightly changed, see Table 10) variables as input. But in this variant, there is the effort adjustment factor (EAF) as extra input. The EAF value is calculated through answering 17 questions about the product to be developed, product quality, personnel capacity & experience, and project management tools. Each question has to be answered on a scale from 1 to 6 (very low – very high). For each answer to a question, a corresponding value can be searched in the EAF table (University of Southern California, 2013). The product of all 17 values is the EAF value (which is typical between 0.9 and 1.4) (Pressman & Ince, 1992).

Project type	a	b	c	d
Organic	3.2	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.2	2.5	0.32

Table 10: Variables for the intermediate COCOMO method

If all variables are known, the required development effort in person months (E) can be calculated through $E = a(KLOC)^b . EAF$. The formulas for the total development time and the people required are identical to the basic COCOMO equations (Pressman & Ince, 1992). Figure 15 shows an PDD of the intermediate COCOMO method.

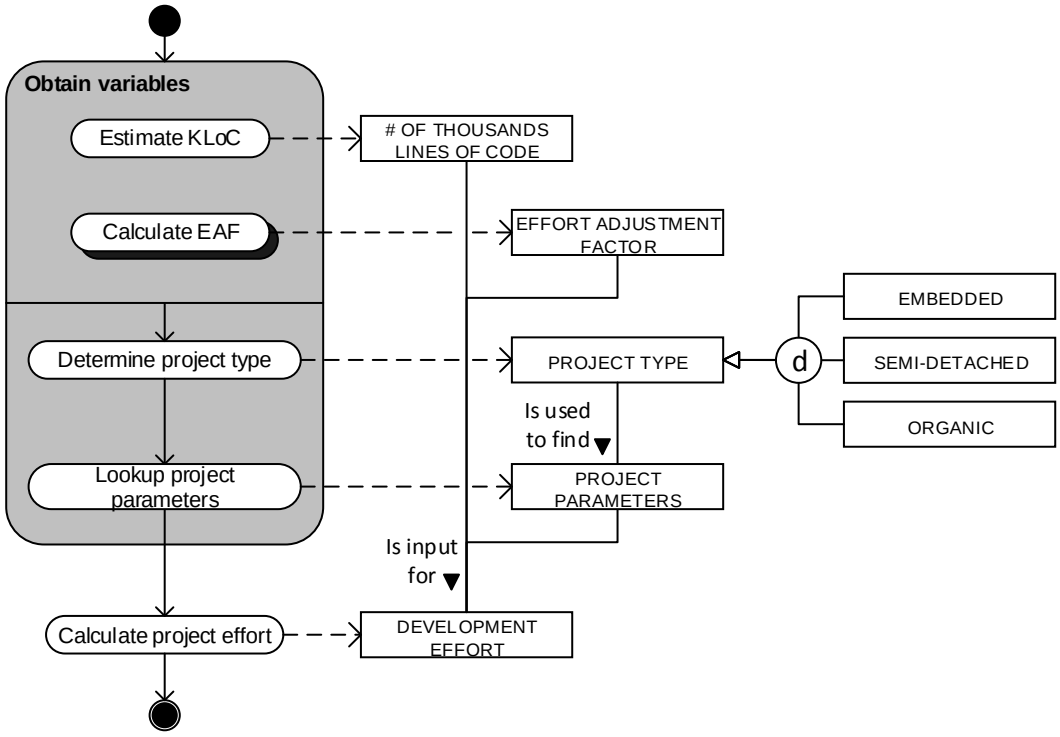


Figure 15: PDD of the intermediate COCOMO method

2.2.2. EXPERTISE BASED ESTIMATION

Expertise based estimation is based on the expertise of lead developers or development team leaders. A popular and widely used expertise based estimation method is the Wideband Delphi method (Wiegers, Stop promising miracles, 2000) (Stellman & Greene, 2005). The strength of this method is that the estimation is done by a team of multiple persons. Stellman & Greene (2005) mention that this is an important factor, because a single person estimation tend to be tighter an unrealistic

The Wideband Delphi methods starts with the project manager constructing an estimation team with representatives from different development teams or departments. With this team, a kickoff meeting will be held, in order to clarify the goal of the project, set measuring units (i.e. development cost in €, or man-hours), and to create a Work Breakdown Structure (WBS) of the development project.

After the kickoff meeting, each team member individually create an effort estimation for its development team for each requirement. It is important that each team member makes its own estimation independently from other team members, so without any influences and pressures from the outside world.

After the individual estimations are made, the estimation team meet again to discuss the assumptions, issues and question that came up during the individual estimation. The project manager plots all estimations in an estimation chart. The process of estimation, plotting and discussing, is repeated for a maximum of four rounds. In each round, the team members try to adjust the estimation based on the discussed topics. In Figure 16, an example of an estimation chart is shown, with 5 teams (represented with "X") and 3 estimation rounds.

Each round in the estimation meeting will take about 15 to 20 minutes. The estimation meeting is over when:

- There are four completed estimation rounds;
- The estimations have converged to a narrow range;
- The meeting time is over;
- The team members are unwilling to alter their estimates.

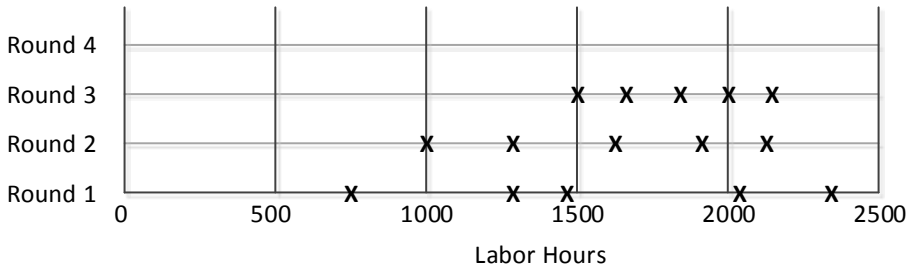


Figure 16: Example of an estimation chart

After the estimation session, the project manager summarizes the results into a master task list and reviews the results with the different teams. Figure 17 shows a PDD of the Wideband Delphi method.

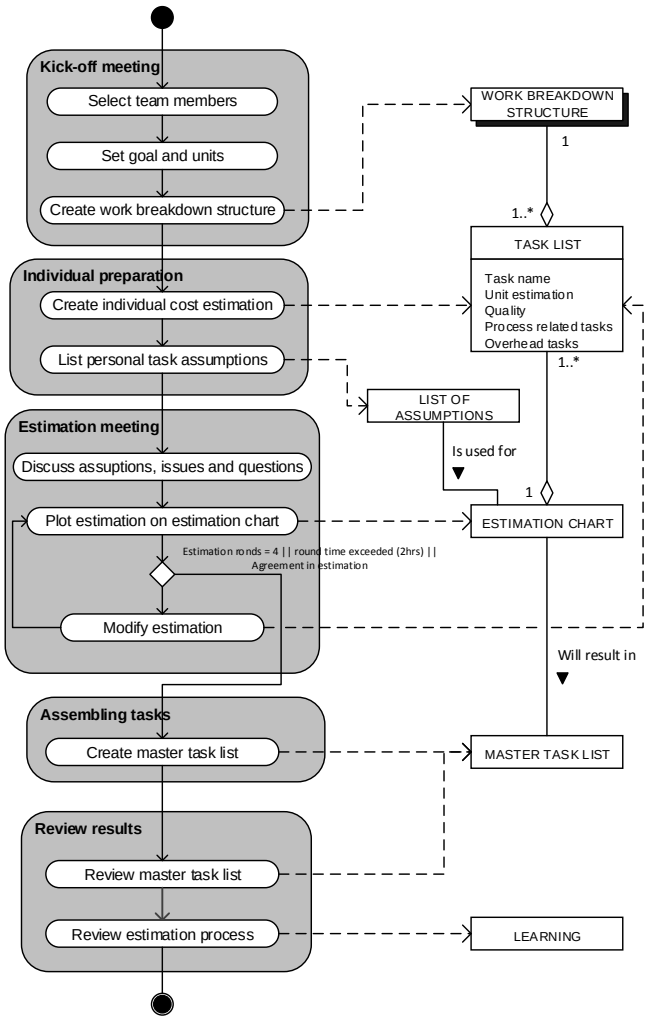


Figure 17: PDD of the Wideband Delphi method

2.2.3. LEARNING BASED ESTIMATION

Learning based estimation methods are methods that creates estimations based on earlier and similar development tasks.

Proxy Based Estimating, or PROBE (Stellman & Greene, 2005) (Humphrey, 2000), is a learning based estimation method, which is based on the assumption that when an engineer is building a software component that he has built before, it takes about the same time to develop it again.

The main principle is that after each software development project, the project is split up in small tasks. These task are logged in a database, together with information about the development effort and time required for the specific task. When one task is registered at least three different times, the product manager can use this data about the effort in future development cost estimation.

When the effort calculation is initiated for a software requirement, a conceptual solution will be developed and split up in different sub-tasks. For each task, the product manager selects the effort value from a similar task out of the historical task database. A manual estimation must be made when a task is not yet stored in the historical task database, or when there are not at least three different estimation records available of the specific task. All estimations will be added through linear regression (so without

consideration that task A influences the effort of task B), which results in a final development cost estimation. In Figure 18, the PDD for the PROBE method is shown.

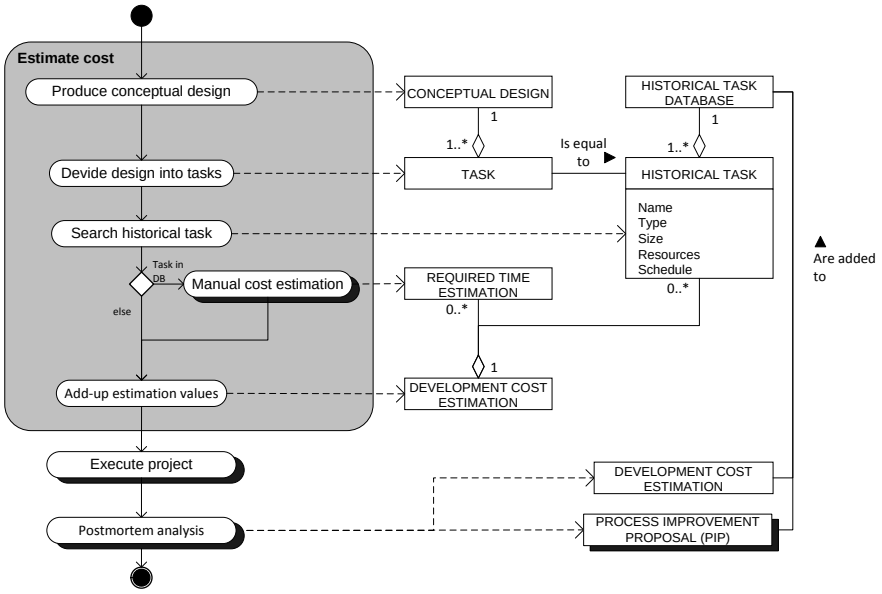


Figure 18: A PDD of the proxy based estimation (PROBE) method

2.3. REQUIREMENT SELECTION

After all requirements are prioritized, and the costs to develop a requirement are determined, the product manager selects the requirements that will be developed in the next release(s) (Bekkers, van de Weerd, Spruit, & Brinkkemper, 2010). When coping with a large number of requirements and limited development, the product manager has to select the right sub-set of requirements in order to make the highest possible contribution to the software product (Carlshamre P. , 2002).

Selecting the right sub-set of requirements is a form of the multiple knapsack problem (Kellerer, Pferschy, & Pisinger, 2004) (Li, van den Akker, Brinkkemper, & Diepen, 2007). The multiple knapsack problem is a mathematical optimization problem. In literature, the knapsack problem is often illustrated as the knapsack with the limited carrying capacity, where a consideration has to be made of what to include in the knapsack in order to have the highest content value within its capacity. The multiple knapsack problem involves more knapsacks, where the challenge is to optimally consider what contents to put in what knapsack in order to achieve the highest value. The multiple knapsack problem is illustrated in Figure 19.

Software product release planning is somewhat like the knapsack problem, when you replace the knapsacks with a number of releases which have a limited number of development resources. The product manager has to pick the right combination of development projects to develop, from a large set of candidate development projects.

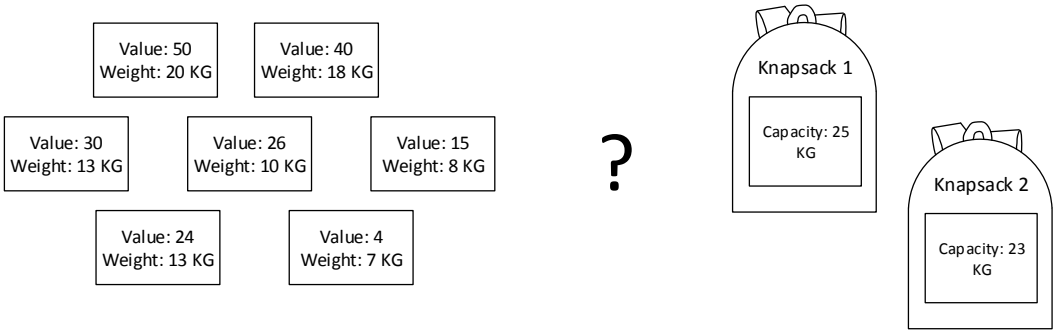


Figure 19: Multiple knapsack problem

Ullah & Ruhe (2006), Jung (1998), and Russel, Norvig, Canny, Malik, & Edwards (1995) talk about applying computational optimization[2] algorithms for automatically selecting the right sub-set of requirements. Linear programming (LP) is a computational optimization solution to solve the (multiple) knapsack problem. There are a number of studies that have linked LP to the application of automated release planning (Ruhe, Eberlein, & Pfahl, Trade-off Analysis for Requirements Selection, 2003) (Carlshamre, Sandahl, Lindvall, Regnell, & Natt och Dag, 2001) (Ruhe & Saliu, The Art and Science of Software Release Planning, 2005) (van den Akker M. , Brinkkemper, Diepen, & Versendaal, 2008) (van den Akker J. M., Brinkkemper, Diepen, & Versendaal, 2005) (Li, van den Akker, Brinkkemper, & Diepen, 2007).

Interdependencies

Requirements seldom are independent from each other (Ruhe, Software release planning, 2005). There are six types of requirement interdependencies identified from literature, which are shown in Table 11 (Carlshamre, Sandahl, Lindvall, Regnell, & Natt och Dag, 2001) (Carlshamre P. , 2002). In the literature about Linear Programming, these interdependencies were not taken into account, while this is a relative important factor in selecting the optimal set of requirements.

Interdependency	Description
AND	This interdependency requires R_1 to be implemented in the same release as R_2
OR	This interdependency means that either R_1 or R_2 will be implemented
REQUIRES	This interdependency requires R_1 to be implemented in the same release, or a release before R_2.
TEMPORAL	This means that either R_1 has to be implemented before R_2, or vice versa (not both in a single release)
CVALUE	This interdependency is used to indicate which requirement affect the customer value of another requirement (value can be affected positively or negatively)
ICOST	This interdependency is used to indicate which requirement affect the implementation cost of another requirement (cost can be affected positively or negatively)

Table 11: Overview of the requirement interdependencies

[2] Bertsekas (2009) describes optimization as "a principle underlying the analysis of many complex decision or allocation problems"

3. RELEASE PLANNING IN THE BUSINESS ENVIRONMENT

Due to confidentiality reasons, the contents of this chapter are hidden in this public version

4. OPTIMAL RELEASE PLANNING METHOD

Based on the insights derived from the literature study and the lessons learned from the business environment, a super method is constructed which meets the capabilities from the SPM competence model as good as possible. The method is constructed around an automated optimization model, which will be described further in this chapter. The contents of this chapter are supported by method fragments, which are part of a larger release planning PDD that can be found in Appendix A (Full PDD of the Optimal Release Planning Method)

4.1. METHOD CONSTRUCTION

Based on the lessons learned from the literature study and the lessons learned from the business environment, a new (optimal) release planning method is constructed. The goal of the release planning method was to cover as much capabilities of the Focus Area Maturity Model (Bekkers, van de Weerd, Spruit, & Brinkkemper, 2010) as possible. In order to achieve this goal, the new release planning method covers the following requirements:

- Multiple stakeholders must be involved in the release planning method
- The release planning has to be based on a cost/value consideration
- The release planning method must support the planning of multiple releases at once
- The optimal release planning has to be calculated through a computational support tool

The selection of which requirement should be included in a release is a difficult task, where the product manager has to take many variables (value, development cost per development team, available development team capacity per release) into account. Finding the right combination of requirements that produces the highest value with respect to the limited capacity could require thousands of combinations that have to be made. Earlier studies have shown that this problem can be tackled by using an optimization method. In order to use an optimization method in the release planning context, three questions need to be answered:

- *What are the required parameters, in order to plan requirements in multiple releases?*
- *What method(s) can be used to derive these parameters?*

The goal of release planning is to select the optimal subset of development projects in order to reach the highest possible customer value, while not exceeding the available development capacity. Which is also known as the (multiple) knapsack problem, which is mentioned in chapter 2.3.

In order to assist the product manager in the selection process, the use of a computer based solver should be included in the selection process. A solver is an application that can solve mathematical problems (like the multiple knapsack problem) based on an optimization model. The multiple knapsack problem can be solved through linear programming, which is a constraint satisfaction optimization technique that calculates the most optimal solution, within the boundaries of certain predefined constraints (Ullah & Ruhe, 2006) (Jung, 1998).

In order to learn the main principles of building an optimization model, and what variables are necessary, more knowledge about the use and implementation of a solver was required. Therefore, the solver from Microsoft (Microsoft Solver Foundation) was used in order to experiment with the different optimization techniques. There are a number of solvers available, such as Gurobi or IBM's CPlex. The reason to use Microsoft Solver Foundation (MSF), was due to the number of available resources like examples and the in-depth explanation on the implementation of the solver.

Linear programming is based on a single goal, which can be either maximizing or minimizing the sum of a certain variable. This goal can be reached within a predefined limitation. In Figure 20, an example is given in order to simplify the explanation of linear programming. Suppose there are 5 entities (named A – E), with each a cost constraint (red number) and a value constraint (green number). The goal of the solver is to select the right entities so the summed value is as high as possible, while the total cost does not exceed the predefined limitation of 15. In this example, the highest value that can be reached is 21.

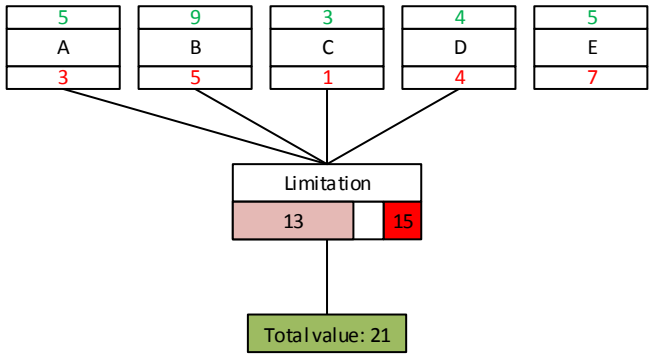


Figure 20: Linear programming

There are some differences that makes the product software release planning more complex than the example from Figure 20. First of all, the lessons learned from the business environment observation have shown that there can be more than one development team within an organization. This means a requirement may require effort from multiple development teams (so a requirement can have one to infinite cost constraints), with each team having a different capacity. Second, it is very likely that the development capacity per team is different in each release. This can be due to simple reasons like a holiday, or an increase or decrease of developers within a team. These factors change the available team capacity, so the capacity for each team in each release can be variable.

It can occur that when a stakeholder has to prioritize requirements for multiple releases, finds a requirement which could be very valuable, but not has to be developed in the first coming release (i.e. a VAT change what will occur in one year from now would be a valuable requirement, but not urgent to develop in for release 5 weeks from now). It is therefore necessary to produce individual priority values for each requirement, in each timeframe (release). So the solver also has to handle different priority values for a single release.

For the development cost of a requirement, the assumption is made that they do not change over time. There are of course factors like the increase of developer knowledge over time or future development patterns that reduce the development effort, but since these factors are not quantifiable, these are not included. Another effort influencing factor are cost affecting interdependencies. An example of this interdependency is for example requirement 1 influencing the development cost of development team 3 for requirement 2 by -20%. This interdependency and other interdependencies are covered in the next paragraph. So for now, the assumption is made that development effort does not change over time.

Interdependencies

Requirements seldom are independent from each other (Ruhe, Software release planning, 2005), therefore, the third difference in the release planning context of the solver is the use of different types of requirement interdependencies (chapter 2.3). During experimenting with the solver, it was not possible to program interdependencies directly into the solver. This was due to the risk of model infeasibility, which means that the model cannot satisfy all programmed rules. An example of infeasibility: suppose all requirements have

an AND interdependency (they have to be in the same release), and the combined effort is 100. However, the solver is programmed to find a solution with an effort limitation of 75. This results in an infeasibility error of the solver because it has to cope with conflicting rules.

The only interdependency that is successfully programmed in the model is the TEMPORAL interrelationship. Requirements with this interrelationship can be in any release, but not in the same release. The reason that this interrelationship is the only one that works, is due to the flexibility of which requirement to allocate in which release.

The reason that a release planning with the other interdependencies cannot (yet) be solved, is that it requires a more complex optimization algorithm using non-linear programming. Linear programming is done with the assumption that the values used in the optimization process do not change during the planning process. Using interdependencies (and thus non-linear programming), the effort and WAS values do change during the planning process.

An example of the problem is given using the multiple knapsack figure from Figure 21. Suppose that the circled blocks have an interdependency that when they're implemented at the same time, the weight of the upper block is reduced by 10 kilo's. A linear programming algorithm does not allows the value or the weight of the blocks to change during planning (since the calculation is not linear anymore), while a (more complex) non-linear algorithm should allow thus complex decision making.

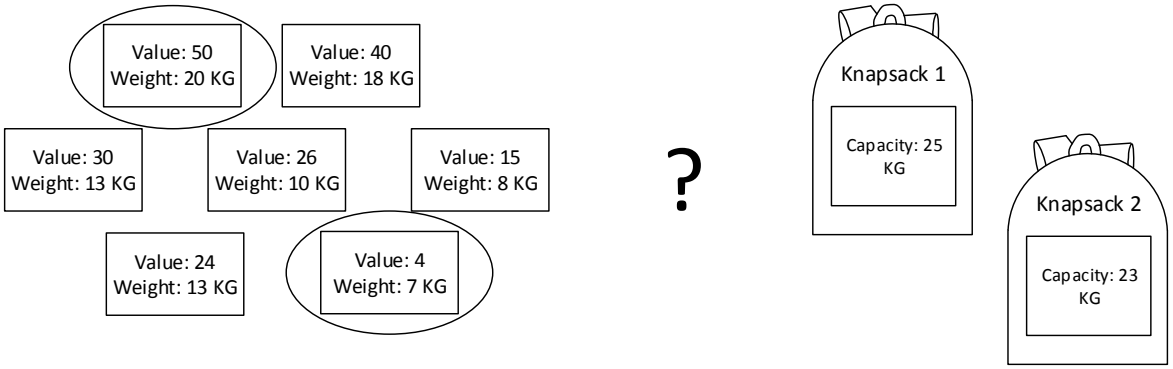


Figure 21: Multiple knapsack problem

Due to the research scope, it was not possible to do a research about using non-linear optimization algorithms in the release planning context, although the interdependencies are extremely viable in release planning. Future research is required in order to develop an optimization algorithm for software release planning purposes. A workaround for this shortcoming in the release planning process is presented in the next paragraph.

Development projects

In order to include the use of requirement interdependencies in the to be constructed release planning method, a transition step has been designed in order to cover most of the interdependencies before the solver calculates the optimal release planning. This transition step involves clustering requirements which have an interdependency into so-called development projects, so each development project is relatively independent from another (Herrmann & Davena, 2008). In Table 12, an overview is made with the clustering approach for each interdependency.

Interdependency	Clustering approach
AND	Requirements sharing an AND interdependency should always be clustered within a development project
OR	For requirements with an OR interdependency, the product manager has to determine which requirement to include in a development project. This manual tradeoff is based on the cost and value of a requirement.
REQUIRES	Requirement with a REQUIRES interdependency should be merged into a single development project. When this is not possible, the development project containing the requirement that should be implemented first, should be developed before the development project containing the second requirement. In this case, the interdependency remains on development project level, and the product manager has to plan these projects manually.
ICOST	The ICOST interdependency is not as straightforward as the three interdependencies described before. When a requirement has a positive effect on the cost of a requirement, these requirements should be grouped together (if the value of both requirements are high enough). When a requirement has a negative effect on the cost of a requirement, the least beneficial requirement should be excluded from the selection process.
CVALUE	The CVALUE interdependency requires an approach similar to the ICOST interdependency.

Table 12: Interdependency clustering approach

Observations from the business environment shows that there are more reasons to group requirements into development project:

- Creating development projects produce a more complete description of the perceived functionality and its context, and are therefore better to understand for stakeholders, which results in:
    - A better cost estimation, since the developers have an overview of the context and what to develop;
    - A better priority rating, since there will probably be less stakeholder confusion;
- Creating development projects reduces the list of options to include in a release.

For these reasons, the release planning method will not be requirement based, but development project based. This means that before the release planning process starts, the product manager creates development projects. Based on these development projects, the different cost and value parameters are determined.

In Figure 22, a conceptual solver model is shown on how the solver uses linear programming in the release planning context. This model contains sample data for example purposes. In this example there are five identified development projects (D1 – D5), two identified releases (R1 & R2), and two identified development teams (T1 & T2). Each development project has a priority parameter for each release (in green font), and a required effort parameter for each development team (in red font). Each development team has an available capacity which is registered per release. The goal of the solver is to find the right combination of development projects in the right release, so that the development effort per team does not exceed the capacity of that development team. For this example, the solver calculated that the release planning is optimal when D1, D3, and D5 are developed in release 1, and D3 is developed in release 2.

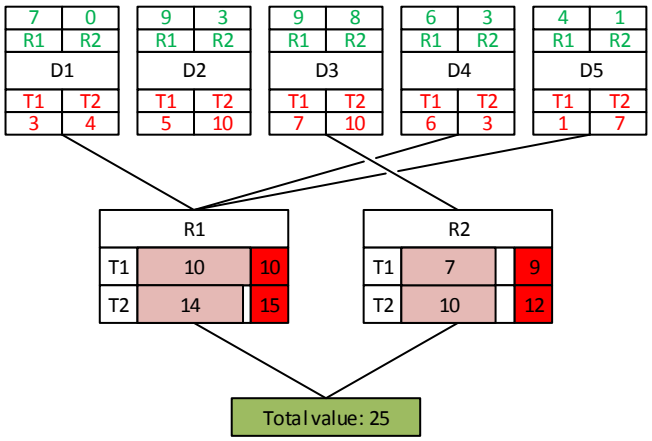


Figure 22: Linear Programming in the release planning context

With this this in mind, the answer to the question "*What parameters are needed in order to plan requirements in multiple releases?*" is:

- Development project value per release;
- Team effort per development project;
- Team capacity per release.

4.1.1. METHOD EVALUATION

The second question "*What methods can be used to derive these parameters?*" is formulized in order to evaluate the prioritization- and software development cost estimation methods from chapter 2.1. The goal of the method evaluation was to find the method, or combination of methods, that produce the required parameters for the solver.

Team effort per development project

The boundaries of the solver solution is based on the total required team effort parameter and the available team capacity parameter. For this reason, those parameters need to be absolute, since it is useless to create boundaries based on relative values. It is therefore best to adopt a method like COCOMO (chapter 2.2.1), Wideband Delphi (chapter 2.2.2) or PROBE (chapter 2.2.3), which all have an absolute effort value as output.

The business environment of AFAS Software show a constantly improving software architecture. This software architecture is focused on standardizing architectural components and attributes, so they can be reused much more easily for new functionality, what will reduce the required development effort. For this reason, a model based estimation method like COCOMO is not suitable for effort estimation. These models do not take custom architectures into account, and are often developed in another era of software development. A learning based estimation method like PROBE could be an accurate method to estimate the required effort, but it requires a large amount of detailed software development data. In a company like AFAS, historical development data is not collected on a detailed level. Besides, estimating based on historical data requires a similar functionality to be developed earlier, which is most of the time not the case at AFAS. Estimation based on historical data is more a method that fits in the tailor-made software industry, where certain functionalities used for a product of customer A, can be used for the product of customer B. An expertise based method like Wideband Delphi has the best fit to adopt in the optimal release planning method. Experts can estimate the required effort with respect to the latest changes in development techniques and software architecture.

Development project value per release

Like mentioned earlier in this chapter, the value of a development project for multiple releases is determined by the priority and the urgency. An important aspect of development project valuation is the involvement of multiple stakeholders (Bekkers, van de Weerd, Spruit, & Brinkkemper, 2010) (Davis, 2003). The method that is required to identify the necessary parameters must meet the following requirements:

- The method must assign a form of value to the development project
- The method must assign a form of urgency to a development project
- The method must have the possibility to involve multiple stakeholders (e.g. customers or partners) into the prioritization process

In chapter 2.1, five different prioritization methods have been presented. In this chapter, these prioritization methods are be evaluated, where after the method is selected that has the best fit for the optimal release planning method. Table 13 shows an overview of the comparison of each methods with the three defined method requirements. After Table 13 the comparison is elaborated per method.

Method	Priority	Multi-release	Multi-stakeholder
AHP	X		X
Cost/Value	X		X
Binary Search Tree	X		X
Hundred-dollar	X		X
Top Ten	X		X
Numerical Assignment	X		X
EVOLVE	X	X	X
Features Prioritization Matrix	X		
Quality Function Deployment	X		X

Table 13: Overview of the method comparison

The AHP method (or cost-value approach) is a very easy and effective method to determine the relative cost and benefit of a requirement. A strong aspect of this method is that it allows consistency checking. This way, the product manager check whether the stakeholder participated seriously in the voting process or the stakeholder voted randomly. A weaker aspect of this method, especially when coping with a lot of requirements, is that it takes a long time to perform all pairwise comparisons (comparing 10 requirements require 44 comparisons, comparing 20 requirements require 189 comparisons). This could be a risk factor for the stakeholder participation.

The hundred-dollar and top ten method are both popular prioritization methods because they are easy to understand, and are relatively quick to execute compared to the other evaluated prioritization methods. The ease of use makes these methods good for multiple stakeholder participation, but the output of these methods are only limited to a prioritization for a single release.

The EVOLVE method is a relative new method designed for requirements prioritization in the software industry. In the base, EVOLVE is an improved version of the “basic” prioritization methods like hundred-dollar or top-ten. A powerful aspect of this method is that it has multiple stakeholder involvement and supports multiple release planning by including an urgency factor, which produces a single value for a development project in each release.

The Features Prioritization Matrix (or the Wieger’s Model) is a method that uses different aspects like benefit, penalty, cost and risk. But these factors make the method too complex to support multiple stakeholders (a customer of a software vendor is not able to determine the risk or cost). Adopting the

Weighted Average Satisfaction technique from the EVOLVE method would allow multiple release planning, but due to the penalty, cost and risk factor, this method is unsuitable to meet the requirements for adoption into the optimal release planning method.

The Quality Function Deployment is a relative complex method, where the main goal is to select requirements that have the most competitive advantage. With a minor method adaption for including stakeholder weight, this method meets the capability for multiple stakeholder support. There is limited literature available about the Quality Function Deployment method, and this method is relatively old (developed in the late 80's). No literature was found about using the Quality Function Deployment in the software industry. Taking these factors into account, makes the Quality Function Deployment not the right method to adopt in the optimal release planning method.

For the optimal release planning method the EVOLVE method is adopted, because this method meets the three requirements for a requirements prioritization method. The EVOLVE method could be combined with the AHP method to support pairwise comparison of development projects. The pairwise comparisons of development projects improve the internal consistency of the development project valuation, which gives more trustworthy results. The negative side to AHP adoption is the great amount of additional questions that a stakeholder has to answer (which is already 45 more question when valuating only 10 requirements, and 105 additional questions when valuating 15 requirements). To keep the release planning method simple and easy to process, only the EVOLVE method is adopted.

Team capacity per release

The release capacity parameter represents the available man-hours of a development team in a particular release. The release capacity is dependent on the interval of the release and the size of the development team, and therefore cannot be defined through a model or method. Therefore, an assumption made in this optimal release planning method is that the available capacity is determined by a manager of the development team, or that these parameters are registered in the product roadmap.

4.2. METHOD PRESENTATION

In the previous chapter, the methods and techniques were evaluated and selected in order to create the optimal release planning method. In this chapter the optimal release planning is presented as a whole. The constructed release planning method can be brought down to four high level activities[3], which are shown in Figure 23.

[3] The closed activity *Identify Requirements* does not belong to the super method, but is included for context purposes.

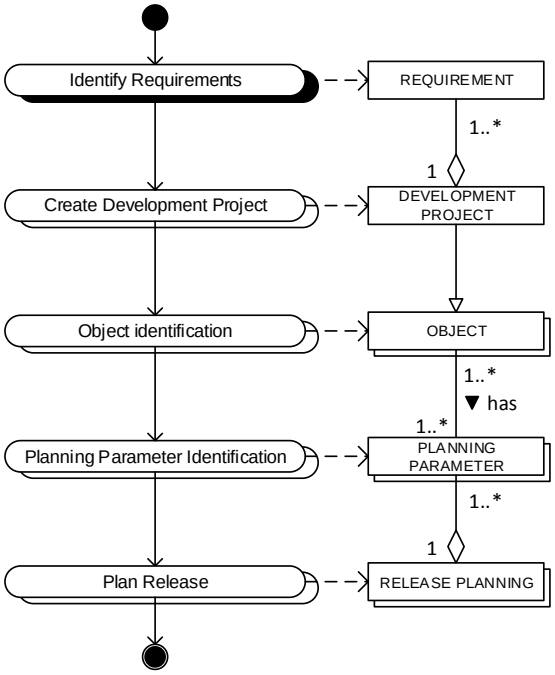


Figure 23: An overview of the super release planning process

In the coming subchapters of this chapter, each activity from the release planning method is elaborated. To give an introduction on each high-level activity, a short description of each activity is listed in Table 14.

Activity	Description
Create Development Project	Grouping requirements which addresses similar functionality, or have a coherent interdependency
Object Identification	In preparation to identifying the planning parameters, the objects need to be identified. An object in this context are considered an entity that remain unchanged over multiple releases. The types of objects that are used in this release planning process are: • Development team • Upcoming release • Stakeholder • Development project Each object can have infinite instances. For instance, the release planning could include one development team, but can also include five different development teams.
Planning Parameter Identification	One of the most important stages in this release planning process is the identification of planning parameters. A planning parameter is a numeric value that results from a combination of at least two instances from different objects. An example of a planning parameter is the available man-hours for each development team in each release. Depending on the number of identified development teams and release instances, each combination of a development team instance and a release instance produces a (numeric) parameter (i.e. team A, release 1 = 40 hours)
Plan Release	Based on the identified planning parameters, a computational tool can calculate

	the optimal release planning with respect to certain predefined constraints (more on that in the sub-chapter). The optimal release planning is considered the combination of development projects that need to be developed that product the highest value (WAS), while not exceeding the engineering capacity.

Table 14: Release planning activity description

During chapter 4.2 and 4.3., the main research artifact is incrementally build up in five different stages, where each state represents a topic in the new release planning method. In each artifact state representation the updated artifact is shown, with the new increments marked in a gray area like shown below.

4.2.1. CREATE DEVELOPMENT PROJECT

Both scientific (Herrmann & Davena, 2008) and practical insights point out that grouping requirements into development projects is an important prerequisite in order to create a release planning. This is necessary in order to:

- Reduce the number of interdependencies;
- Reduce the list of options of what to include in a release;
- Ease the process of cost/value estimation for the development teams.

Figure 24 shows the PDD of identifying development projects. When describing this process, the assumption is made that the requirements are already identified and organized through one or more functional components from the Functional Architecture Framework.

The first activity for the product manager is the candidate requirement consideration, and its purpose is to judge whether a requirement is a candidate requirement or not. This is done based on the process knowledge of the product manager, and the vision for the product for the near future. When a standard requirement becomes a candidate requirement, the status property of the requirement changes to candidate. It is situational what to do when a requirement is discarded; the product manager can decide to delete the requirement from the requirement database, or include the requirement in the next release planning session.

Based on the candidate requirement list, one or more development projects can be made. A development project is like a group of requirements and/or bug fixes involving the same goal or theme. An example can be the introduction of a new product variant within an existing product. For this product variant, many large scale (high level, involving many changes) and small scale (very specific, like adding a specific text field) requirements have been gathered. Requirements involving similar functional goals are grouped into development projects.

When grouping requirements into development projects the product manager has to take requirement interdependencies into account. Requirements with a coherent interdependency should be included into the same development project, while requirements with an incoherent interdependency should not be in the same development project. An overview of grouping criteria per interdependency can be found in Table 12 on page 44.

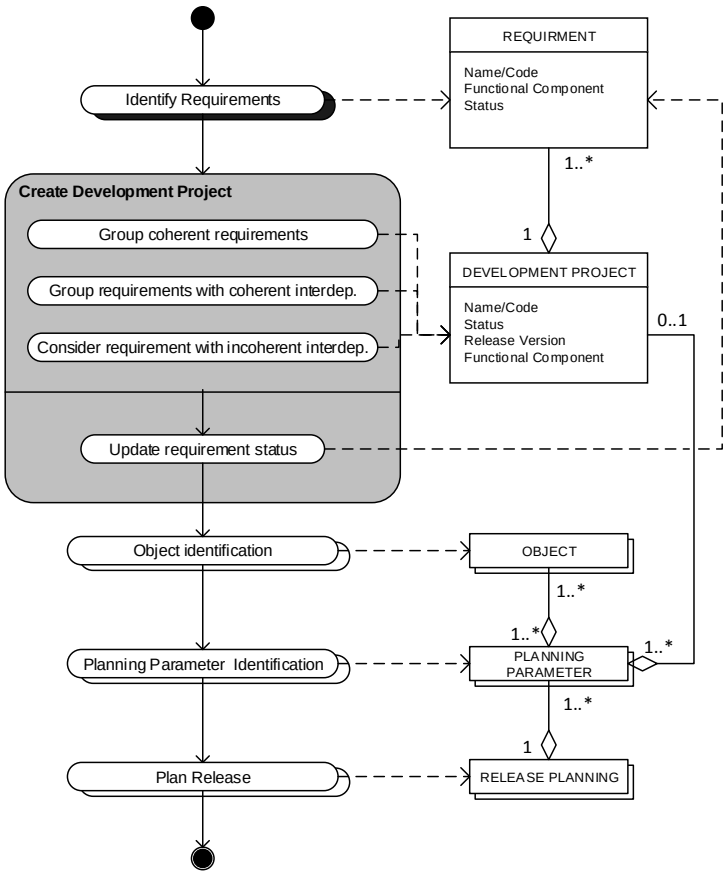


Figure 24: Method fragment for creating development projects

4.2.1.1. ARTIFACT STATE

A representation of the current artifact state is shown in Figure 25. In this figure, an unequal list of requirements is shown, which will result in a smaller list of development projects. This figure will incrementally be extended through chapter 4.2 and 4.3.

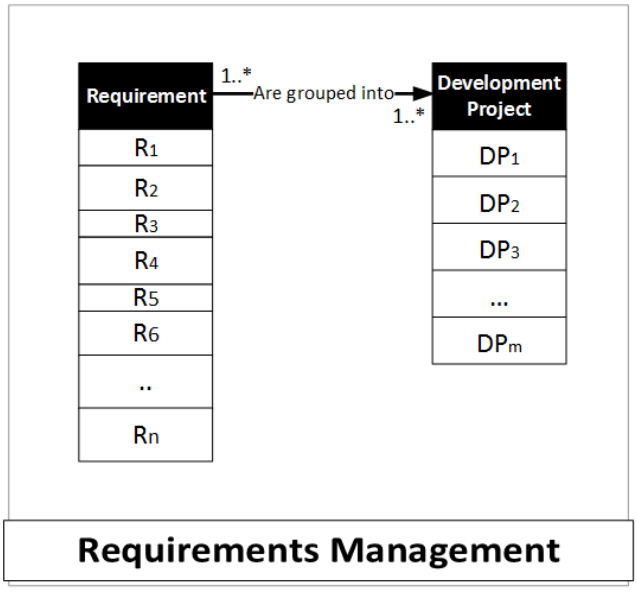


Figure 25: Artifact State 1/5

4.2.2. OBJECT IDENTIFICATION

The object identification phase is for identifying objects to include in the release planning process. Objects in this context are considered entities or variables that remain unchanged over multiple releases. The constructed release planning process distinct four kinds of objects:

- Development Team; which are the different development teams in an organization. Examples of development teams are UI design, architecture, application development, testing, or even documentation writers.
- Release; which is the identified release to include in the release planning.
- Stakeholder; which is the stakeholder to include in the release planning process. Per stakeholder, a relative weight/importance as to be determines, which is a value between 1 (low weight/importance) and 9 (high weight/importance).

The fourth objects is the development project, but this object is already identified in the previous activity. In Figure 26, a PDD is presented of the release planning process with the object identification activity expanded. In Figure 27, the overview from Figure 25 is extended with the objects mentioned in this chapter.

The identification of the objects is necessary to determine the variables that will be used in the release planning. These steps are described in the next chapters.

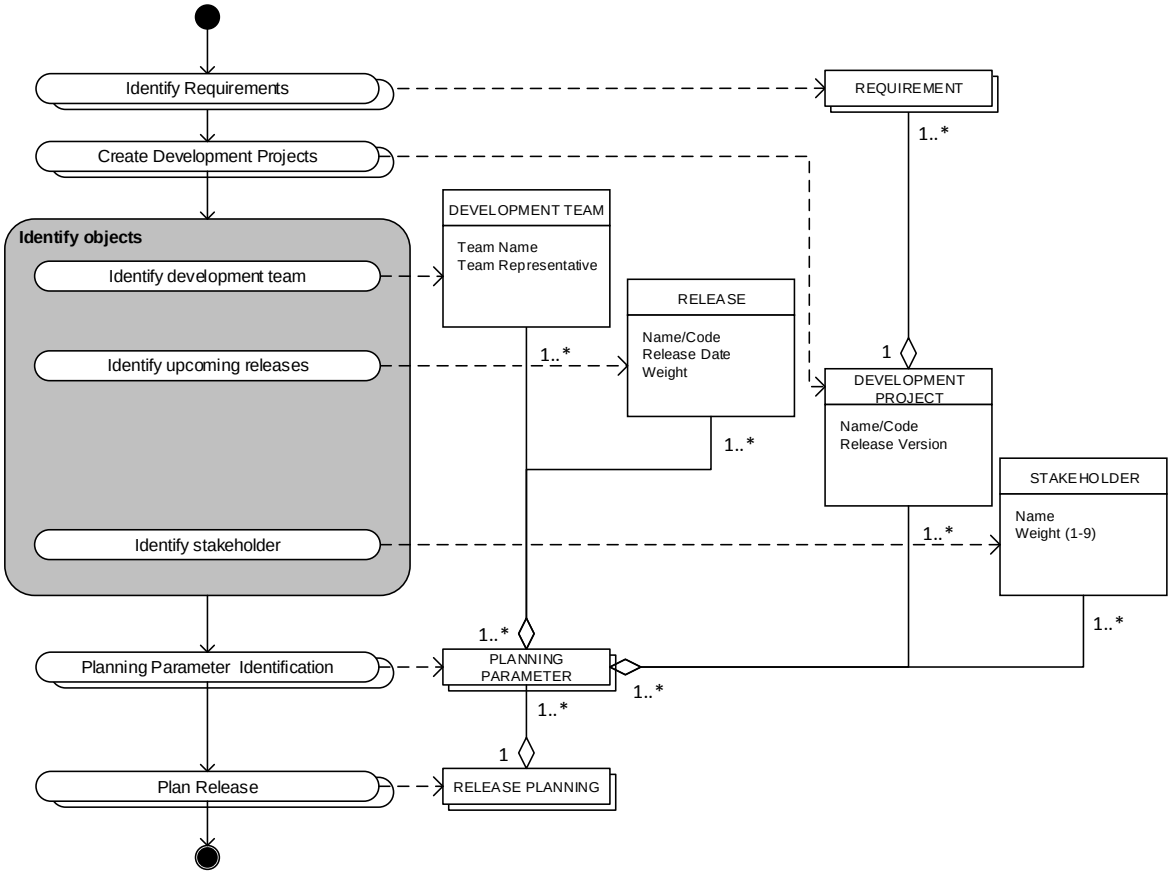


Figure 26: PDD of the constructed release planning method, with object identification expanded

4.2.2.1. ARTIFACT STATE

In Figure 27, the artifact is extended with the object identified in chapter 4.2.2.

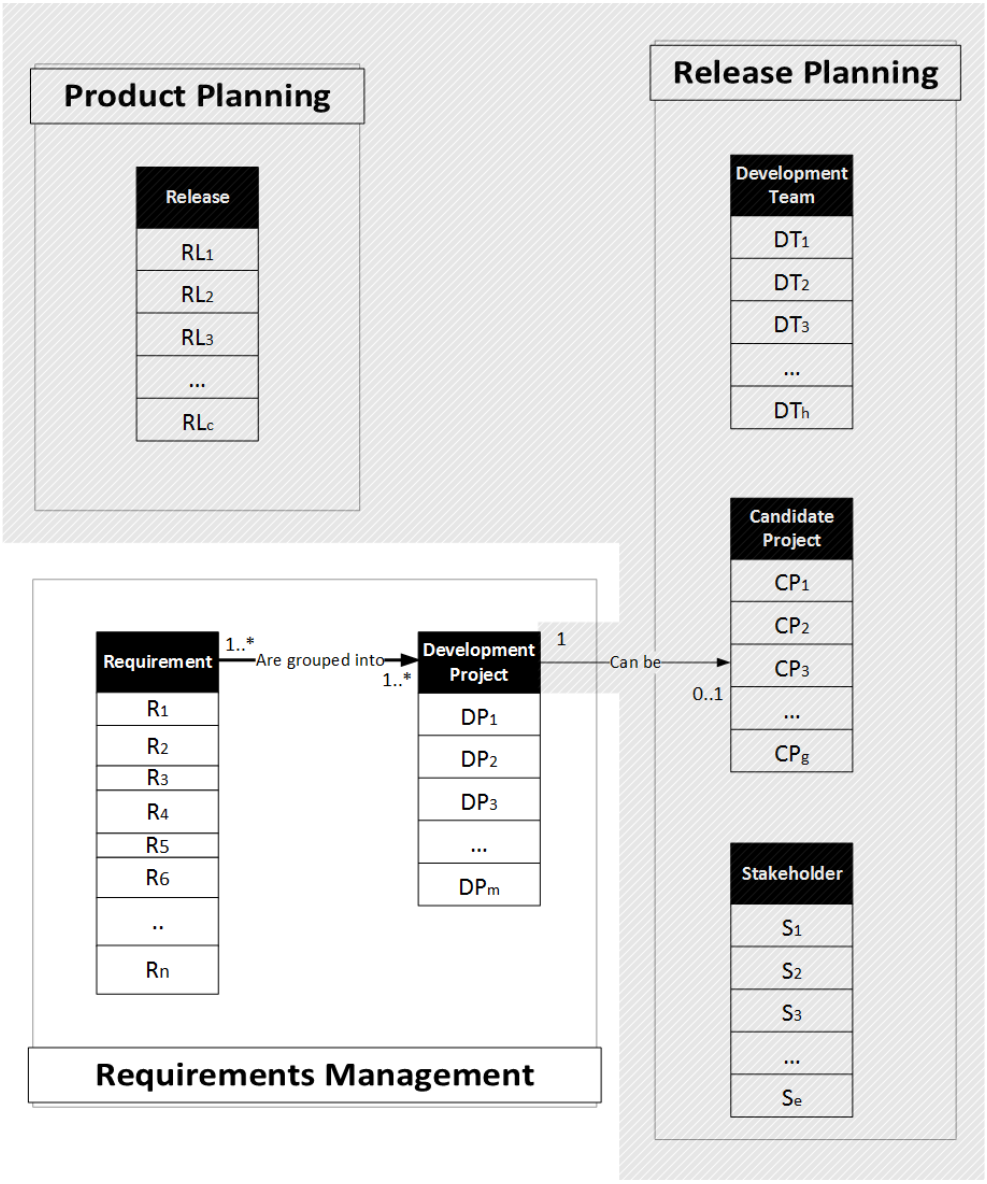


Figure 27: Artifact State 2/5

4.2.3. PLANNING PARAMETER IDENTIFICATION

The main inputs for a release planning are different planning parameters. The planning parameters are based on combinations of objects. In this generic release planning method, there is a distinction between four planning parameters:

- Development Team Capacity; which is a planning parameter that represents the available resources per identified development team for each identified release. So if the release planning

includes for example 3 teams and 2 releases, there are 6 Development Team Capacity planning parameters (one for each team in each release).

- Project Effort; which is the required effort for each development team to develop a certain development project. It is necessary to identify the effort per team for all development projects.
- Development Project Weighted Average Satisfaction (WAS); which is a planning parameter that represents the perceived stakeholder satisfaction for a development project in a certain release. The WAS is calculated from two other variables: stakeholder urgency and stakeholder value.
- Temporal interrelationships; which are projects that should not be in the same release as each other (but can be in any other release)

Figure 28 shows a PDD of the release planning process with the object identification and planning parameter identification activities and concepts expanded. The planning parameters Development Team Capacity, Project Effort, and Development Project are described in the upcoming sub-chapters.

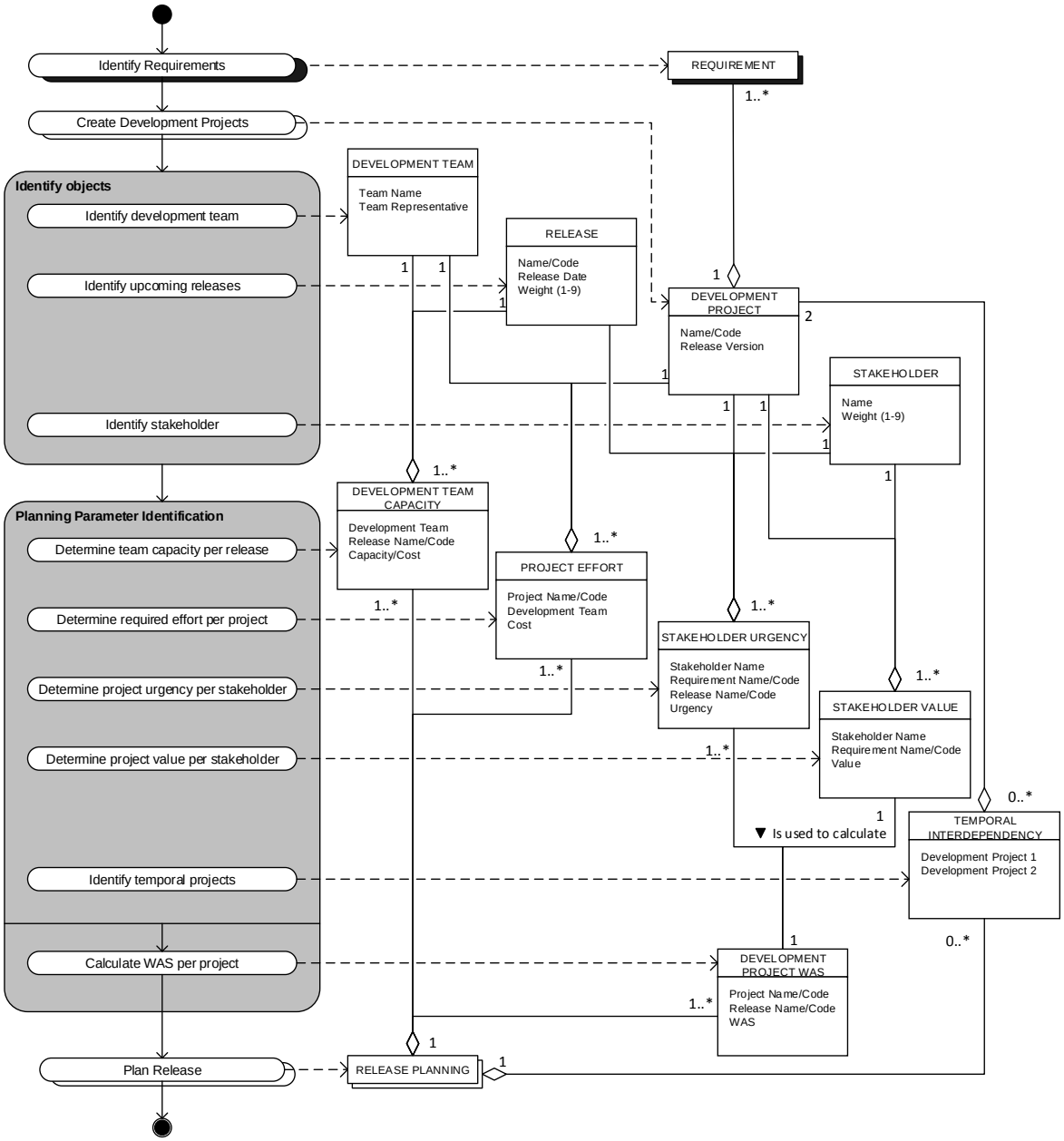


Figure 28: PDD of the release planning process, with the planning parameter identification expanded

4.2.3.1. DEVELOPMENT TEAM CAPACITY

For each release that is included in the release planning, a number of available man-hours per teams must be known. There is not a formal method to derive this number, because it is dependent on many situational factors (e.g. number of FTE, upcoming holidays, new- or leaving employees, etc.). The development team capacity should be determined through a personnel planning, or should be determined in the product roadmap.

The number of development team capacity planning parameters should equal the number of development teams * the number of releases included in the release planning. In Table 15, an example is shown of the collected capacity variables of two teams {A,B} for 3 releases.

Team	**A**	**B**	**A**	**B**	**A**	**B**
Release	1	1	2	2	3	3
Capacity	**40**	**60**	**100**	**10**	**40**	**60**

Table 15: Example of the development team capacity variables

4.2.3.2. DEVELOPMENT PROJECT EFFORT

The project effort is represented as the estimated amount of man-hours required per development team to develop a certain development project. The amount of required man-hours is estimated through an expertise based method called Wideband Delphi. Using this method, each development team expert (with expertise in their own development field), make an estimation of required man-hours per development project for the work that is relevant to the development team.

If all estimations are made, the number of project effort planning parameters should be equal with the number of development teams * the number of development projects. In Table 16, an example is shown of the collected variables of two teams {A,B} and 3 projects.

Team	**A**	**B**	**A**	**B**	**A**	**B**
Project	1	1	2	2	3	3
Effort	**5**	**10**	**7**	**0**	**1**	**14**

Table 16: Example of the development project effort variables

4.2.3.3. DEVELOPMENT PROJECT WEIGHTED AVERAGE SATISFACTION

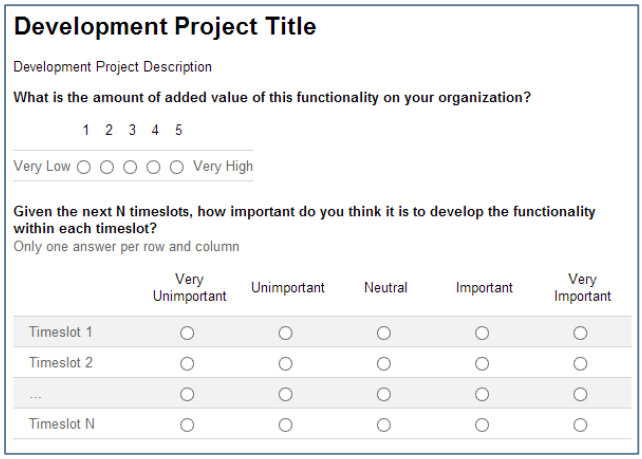
The main principle of the optimal release planning process is that stakeholder can participate in the release planning process. This participation is done in a form of a survey, where stakeholders can assign weights and urgencies to requirements. To include the prioritization method of the EVOLVE method into the optimal release planning method, some adaptions have from the method as described in chapter 2.1.3.1. These adaptions have been made in order to increase the ease of use from the business perspective, and are based on a review of the method increment by two product managers at AFAS.

The goal of the stakeholder participation is to assign one priority value (WAS) to a development project in a certain release. So if the release planning is made for three releases, there will be three values per

development project. The stakeholder population can exist of (but is not limited to) customers, partners, management, developers, or product managers. Each stakeholder receives a questionnaire where the stakeholder must answer two questions for each candidate development project:

- *What is the amount of added value of this functionality on your organization?* In this question, the stakeholder must value each development project on a scale from 1 (not desirable) to 5 (very desirable);
- *Given the next N timeslots, how important do you think it is to develop the functionality within each release?* For this question, the adaption has been made to replace a release with a timeslot (e.g. 2 – 3 months). This adaption is made for two reasons: 1) it is not known if all stakeholders are familiar with the release code. 2) stakeholders might expect that the development project will actually be released in one of the proposed releases (which is not necessarily true). In this question, the stakeholder must indicate, per timeslot, the importance of the development project on a 1 (very unimportant) to 5 grade (very important).

An example of the stakeholder survey is shown in Figure 29.



Development Project Title

Development Project Description

What is the amount of added value of this functionality on your organization?

1 2 3 4 5

Very Low ○ ○ ○ ○ ○ Very High

Given the next N timeslots, how important do you think it is to develop the functionality within each timeslot?

Only one answer per row and column

	Very Unimportant	Unimportant	Neutral	Important	Very Important
Timeslot 1	○	○	○	○	○
Timeslot 2	○	○	○	○	○
...	○	○	○	○	○
Timeslot N	○	○	○	○	○

Figure 29: Example of the stakeholder survey

Based on the results of the respondents, the weighted average satisfaction for each development project in each release will be calculated through the formula described in chapter 2.1.3.1.

WAS in ratio

If the number of stakeholders differ over multiple release planning session, the total WAS value (over all development projects and releases) can fluctuate, which makes it hard to compare the results over different release planning sessions. It is therefore necessary to calculate the ratio WAS per release planning session. To do this, the product manager has to divide each WAS value by the sum of all WAS values of the release planning session, and multiply the result by 100.

Table 17 shown an example of the ratio WAS value. Both tables have 10 development projects (P1, … , P10). The left table represents a prioritization session with only a few stakeholders, resulting in a WAS score between 1 and 20 for each development project in each release (R1 & R2). The right table represents a prioritization session with more stakeholders, resulting in a WAS score between 7 and 140. At first sight, these results cannot be compared. But by calculating the ratio WAS, the results are equalized and comparable. With this adaption, the total WAS value of a release planning is always 100, no matter how many stakeholders are involved in the prioritization. The different WAS score per development project also represent the percentage of the total satisfaction.

But why is this useful, since there is only one release planning session per release? The reason for this is to compare the not selected projects over time. If a project is not planned in a release, stakeholders can vote on the same project in another session. It could be interesting for the product manager to measure the WAS of a project over multiple release planning sessions. The ratio WAS results can also be used to compare the results of the internal stakeholders with the external stakeholders, or other statistical analysis.

P	WAS R1	Ratio	WAS R2	Ratio
1	9	3,90	8	3,46
2	15	6,49	18	7,79
3	11	4,76	14	6,06
4	4	1,73	14	6,06
5	19	8,23	19	8,23
6	15	6,49	1	0,43
7	9	3,90	10	4,33
8	11	4,76	14	6,06
9	20	8,66	1	0,43
10	17	7,36	2	0,87

P	WAS R1	Ratio	WAS R2	Ratio
1	63	3,90	56	3,46
2	105	6,49	126	7,79
3	77	4,76	98	6,06
4	28	1,73	98	6,06
5	133	8,23	133	8,23
6	105	6,49	7	0,43
7	63	3,90	70	4,33
8	77	4,76	98	6,06
9	140	8,66	7	0,43
10	119	7,36	14	0,87

Table 17: Ratio WAS value example

4.2.3.4. ARTIFACT STATE

Taking into account the three described planning parameters (Capacity, Effort, and WAS), the artifact is updated. The current state of the artifact is shown in Figure 30.

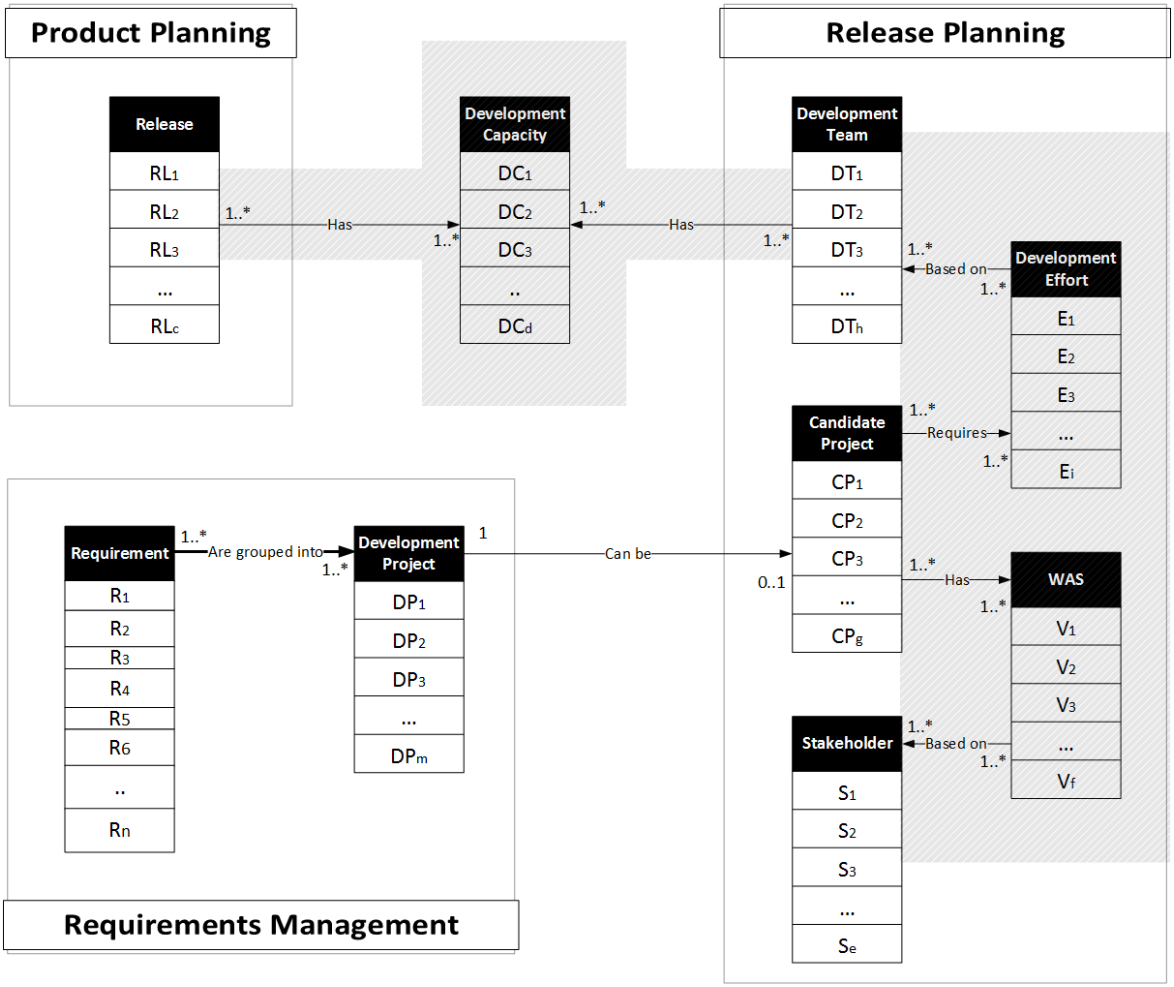


Figure 30: Artifact State 3/5

4.2.4. PLAN RELEASE

If all planning parameters are identified, the release can be planned. In order to find the optimal release planning, a computational solver using Linear Programming will be used in order to find the optimal sub-set of development projects in releases so the highest possible WAS is achieved, while the effort does not exceed the capacity of each development team, in each release.

The PDD to visualize this process is presented in Figure 31, where after the sub activities are briefly described in Table 18.

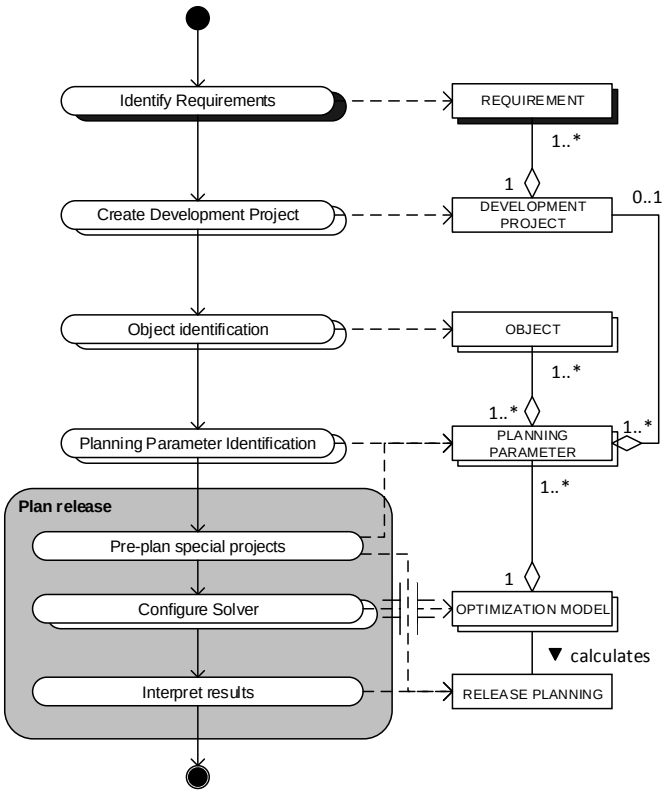


Figure 31: PDD of the release planning method, with the actual release planning activity expanded

Activity	Description
Pre-plan special projects	Special projects is a collective noun for development projects which will be developed in a predefined release, no matter the WAS or effort. Examples of special projects can be a development project that has to be developed before a specific date due to statutory changes, or that the product manager committed to the client to develop the project a particular release. Special projects are always allocated to a release, before the computer model calculates the optimal combination of projects in releases. If a special project is allocated to a project, it is important that the planning parameters are updated. So if a development project is planned in the first coming release, the product manager has to subtract the team effort of the development project from the available team capacity of the release where the project is planned.
Configure solver	In order to make the solver calculate the optimal release planning, it has to be configured with data and constraints. The solver uses the planning parameters to calculate the optimal release planning. More on this subject in chapter 4.2.4.1
Interpret results	Depending on the solver, the results may not be visually attractive and not really easy to interpret for people who don't have in depth knowledge about the process. It is therefore necessary that the product manager interprets the solver results, and make a nice overview of the final results. An example is shown in chapter 0

Table 18: Release planning sub-activity description

In the next two sub-chapters, the solver configuration and the interpretation of the solver results are elaborated.

4.2.4.1. CONFIGURING SOLVER

In chapter 4.1, it was mentioned that Microsoft Solver Foundation (MSF) was used to make a conceptual solution to the release planning problem, using Linear Programming (LP). In this chapter, the configuration of the solver is explained. In this explanation only MSF is shown, although there are other compatible solvers on the market like for example Gurobi, or CPlex from IBM.

Since the MSF is very extensive, the focus is only on the configuration in the release planning context. MSF uses four main principles to develop an optimization model:

- Set
- Parameter
- Goal
- Constraint

The four main principles are describes in Table 19.

Principle	Description
Set	Sets are identical to the objects used in this method. For the solver, 3 different sets are identified: • Projects • Releases • Temporal projects
Parameter	Parameters are identical to the planning parameters used in this method. In the release planning context, there are 4 kinds of parameters: • WAS • Effort (per team) • Capacity (per team) • Temporal interdependency
Goal	The solver can only have on goal. In the release planning context it is to maximize the sum of the WAS values of the selected development projects
Constraint	The constraints represent the boundaries of the solver. In the software release planning context, the next constraints have to be programmed into the solver • A development project can only be planned in one release • The required effort of the selected development projects must not exceed the available capacity per team for the selected release. • Development projects that are temporal interrelated cannot be in the same release

Table 19: MSF principles

In order to present how the solver is actually configured using the principles from Table 19, a detailed example is given. This example includes 20 development projects, 2 releases, 2 development teams {T1, T2}, and 2 temporal interdependencies. The sets that are used in this release planning simulation are Projects (numbers 1 to 20) and Releases (numbers 1 and 2). The parameters identified in this example are:

- 40 WAS parameters. Each value parameter represents the value of one development project in one release. In this simulation, there are 20 projects and 2 releases. So 20 * 2 makes 40 value parameters (Table 20);
- 40 effort value parameters. Each effort parameter represents the required effort of one particular team, in one particular development project. This simulation contains 20 projects and 2 teams, so 20 * 2 makes 40 effort parameters (Table 21);
- 4 capacity parameters that represents the available capacity in man-hours per release. This simulation uses 2 teams and 2 releases, so 2 * 2 = 4 capacity parameters (Table 22);
- 2 temporal parameters (Table 23).

The defined goal of the solver is to maximize the (WAS) value over two releases, while satisfying the next three constraints:

- A project can only be developed in 0 or 1 release;
- The required effort (per team) of a release must not exceed the available capacity (per team) of a release;
- The projects that have a temporal interdependency (Table 23) will not be planned in one release

The goal of the solver can also be formulated as:

Maximize: $\sum V(selected)$

Subject to: $\forall CP \sum_{k=1}^{c} RL_k \leq 1$

And: $\forall RL \; \forall DT \sum E \leq DC$

Project	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Release	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
WAS	**9**	**12**	**6**	**3**	**14**	**11**	**20**	**3**	**3**	**9**	**18**	**6**	**2**	**7**	**19**	**4**	**2**	**17**	**3**	**2**
Project	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Release	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
WAS	**19**	**2**	**6**	**1**	**1**	**8**	**3**	**14**	**11**	**12**	**17**	**2**	**16**	**16**	**10**	**14**	**7**	**12**	**13**	**15**

Table 20: WAS parameters for the release planning simulation

Project	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Effort T1	10	16	17	12	9	15	13	14	7	13	17	14	15	13	16	8	18	8	14	10
Effort T2	5	11	20	8	4	14	13	9	7	10	15	14	19	17	9	16	12	20	2	11

Table 21: Effort parameters for the release planning simulation

Release	1	2
Capacity T1	100	150
Capacity T2	150	100

Table 22: Capacity parameters for the release planning simulation

Project1 (set)	Project2 (parameter)
1	5
1	4

Table 23: TEMPORAL interdependencies

The method to set-up an optimization model using MSF is modeled in a PDD, which is shown in Figure 32.

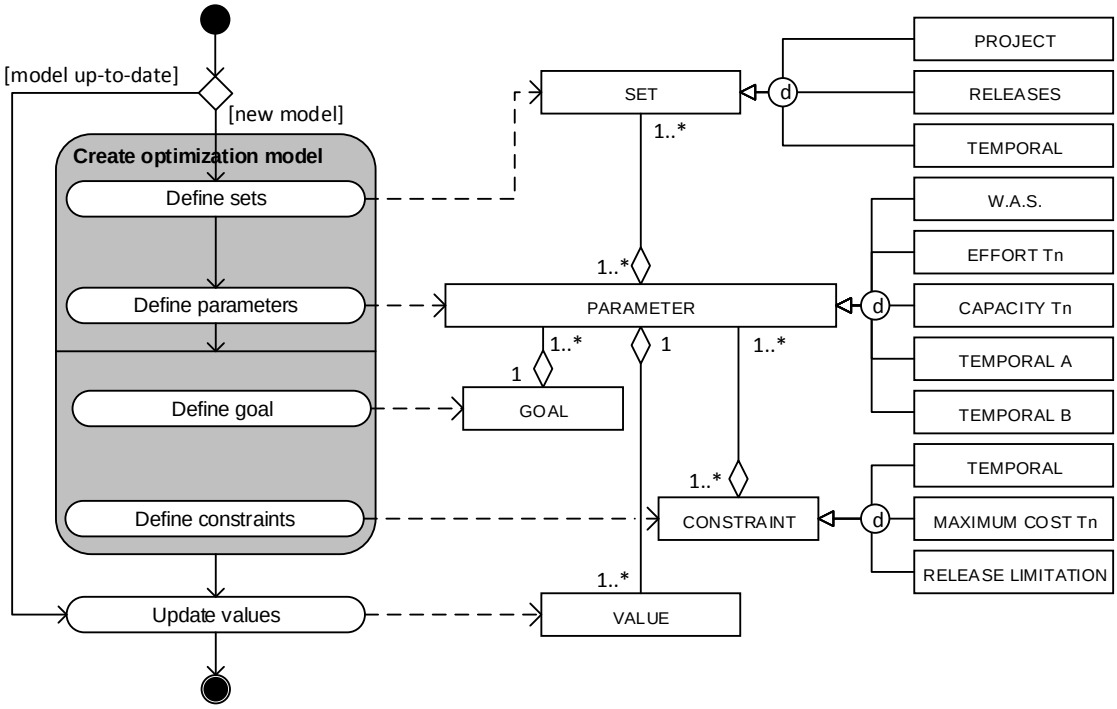


Figure 32: PDD of the activity "configure solver" in the "plan release" activity of the main PDD

4.2.4.2. INTERPRETING COMPUTED RESULTES

When the optimization model is correctly build, and all data is successfully bound to the right sets and parameters, the solver can be initiated. It then calculates each possible combination (within the defined constraints) in order to return the combination(s) which have the highest value.

Name	Value
Solution Type	Optimal
Goal	241
Decision[1, 1]	FALSE
Decision[1, 2]	TRUE
Decision[2, 1]	TRUE
Decision[2, 2]	FALSE
Decision[3, 1]	FALSE
Decision[3, 2]	FALSE
Decision[4, 1]	FALSE
Decision[4, 2]	FALSE
Decision[5, 1]	TRUE
Decision[5, 2]	FALSE
Decision[6, 1]	TRUE
Decision[6, 2]	FALSE

Decision[7, 1]	TRUE
Decision[7, 2]	FALSE
Decision[8, 1]	FALSE
Decision[8, 2]	TRUE
Decision[9, 1]	FALSE
Decision[9, 2]	TRUE
Decision[10, 1]	FALSE
Decision[10, 2]	TRUE
Decision[11, 1]	TRUE
Decision[11, 2]	FALSE
Decision[12, 1]	FALSE
Decision[12, 2]	FALSE
Decision[13, 1]	FALSE
Decision[13, 2]	TRUE

Decision[14, 1]	FALSE
Decision[14, 2]	TRUE
Decision[15, 1]	TRUE
Decision[15, 2]	FALSE
Decision[16, 1]	FALSE
Decision[16, 2]	TRUE
Decision[17, 1]	FALSE
Decision[17, 2]	FALSE
Decision[18, 1]	TRUE
Decision[18, 2]	FALSE
Decision[19, 1]	FALSE
Decision[19, 2]	TRUE
Decision[20, 1]	FALSE
Decision[20, 2]	TRUE

In Table 24, the solver output of the example from chapter 4.2.4.1 is shown. In this example, the solver has made 378.195 calculations to find out that the maximum WAS that could be reached over two releases is 241.

Name	Value
Solution Type	Optimal
Goal	241
Decision[1, 1]	FALSE
Decision[1, 2]	TRUE
Decision[2, 1]	TRUE
Decision[2, 2]	FALSE
Decision[3, 1]	FALSE
Decision[3, 2]	FALSE
Decision[4, 1]	FALSE
Decision[4, 2]	FALSE
Decision[5, 1]	TRUE
Decision[5, 2]	FALSE
Decision[6, 1]	TRUE
Decision[6, 2]	FALSE

Decision[7, 1]	TRUE
Decision[7, 2]	FALSE
Decision[8, 1]	FALSE
Decision[8, 2]	TRUE
Decision[9, 1]	FALSE
Decision[9, 2]	TRUE
Decision[10, 1]	FALSE
Decision[10, 2]	TRUE
Decision[11, 1]	TRUE
Decision[11, 2]	FALSE
Decision[12, 1]	FALSE
Decision[12, 2]	FALSE
Decision[13, 1]	FALSE
Decision[13, 2]	TRUE

Decision[14, 1]	FALSE
Decision[14, 2]	TRUE
Decision[15, 1]	TRUE
Decision[15, 2]	FALSE
Decision[16, 1]	FALSE
Decision[16, 2]	TRUE
Decision[17, 1]	FALSE
Decision[17, 2]	FALSE
Decision[18, 1]	TRUE
Decision[18, 2]	FALSE
Decision[19, 1]	FALSE
Decision[19, 2]	TRUE
Decision[20, 1]	FALSE
Decision[20, 2]	TRUE

Table 24: Release planning proposal of the simulation based on the earlier defined parameters

Now all the mathematical work is completed, the product manager must interpret the results. In the solver output of Table 24, the decisions are shown as: Decision[P,R] = TRUE/FALSE, where P represents the project and R represents the release. To simplify the overview, the product manager could sort the outcomes like in Table 25.

Release 1	Release 2	Not allocated
Project 2	Project 1	Project 3
Project 5	Project 8	Project 4
Project 6	Project 9	Project 12
Project 7	Project 10	Project 17
Project 11	Project 13	
Project 15	Project 14	
Project 18	Project 16	
	Project 19	
	Project 20	

Table 25: An example of a sorted output of the release planning

A final remark on the optimal release planning method is that product software release planning still is a specialist job. The software product manager must not be intimidated into a proposed solution, and should use the output of the presented method only as a guideline. In the end, the selected development projects in a release have to make intuitive sense (Davis, 2003).

4.2.4.3. ARTIFACT STATE

Configuring a solver and interpreting its result is an important part of release planning, and certainly belongs in the artifact that is developed during this research. In Figure 33 the updated artifact is represented, including the use of the solver and its relation with the previous version of the artifact.

Figure 33: Artifact State 4/5

4.3. FUNCTIONAL ARCHITECTURE FRAMEWORK INTEGRATION

Like described in chapter 1.1.3., the current version of the FAF (Salfischberger, van de Weerd, & Brinkkemper, 2011) focuses exclusively on requirements organizing and identification. This is done by linking certain requirements to functional components through coding the requirement database, like described in Figure 4.

In order to answer the main research question of this research, it is key to find out how the optimal release planning method as described in the previous chapter can be integrated with the original FAF process.

When comparing the FAF process (requirement identification) and the release planning process, there are no similar processes found. Both requirement identification and release planning are separate from each other, and both processes even have different triggers (requirement identification is triggered by an incoming requirement, release planning is time triggered). However, the identified functional components from the FAF are considered useful for release planning in two ways:

1. **Development project identification**

The functional components from the FAF can assist the product manager in the identification of development projects. As mentioned earlier in this thesis, development projects contain requirements with similar functionalities (and thus similar functional components). So when creating a development project, candidate requirements for the development project can easily be selected though functional components.

2. **Roadmap theme identification**

Product Road mapping is a focus area in the business function Product Planning of the SPM Competence Model (Bekkers et al., 2010). Product Road mapping involves the short- and long term road mapping of a software product. In these road mappings, future release themes are determined. The release themes are based on for example the market trends, core assets, and future technology. Based on the contents of the release theme, one or more functional components can be attached to the release theme.

If both the development projects and release themes are identified through functional components, the product manager can select the candidate development project by selecting on the functional components attached to the release themes that need to be included in the release.

In Figure 34, an updated version of a part of the optimal release planning method is shown, where the adapted method fragments are shown with grey markings. In this updated version, the FUNCTIONAL COMPONENT now has a broader application in the release planning process. In the DEVELOPMENT PROJECT, the property functional component is adopted. A new concept RELEASE THEME is added which also contains a functional component (the development of RELEASE THEME is not within the scope of this process). An identified release can contain 0 or multiple THEMES, which can be used to select a set of CANDIDATE DEVELOPMENT PROJECTS.

In Figure 35, the final version of the artifact is shown, with as new increments the Functional Architecture Framework, and its relation to the new release planning process.

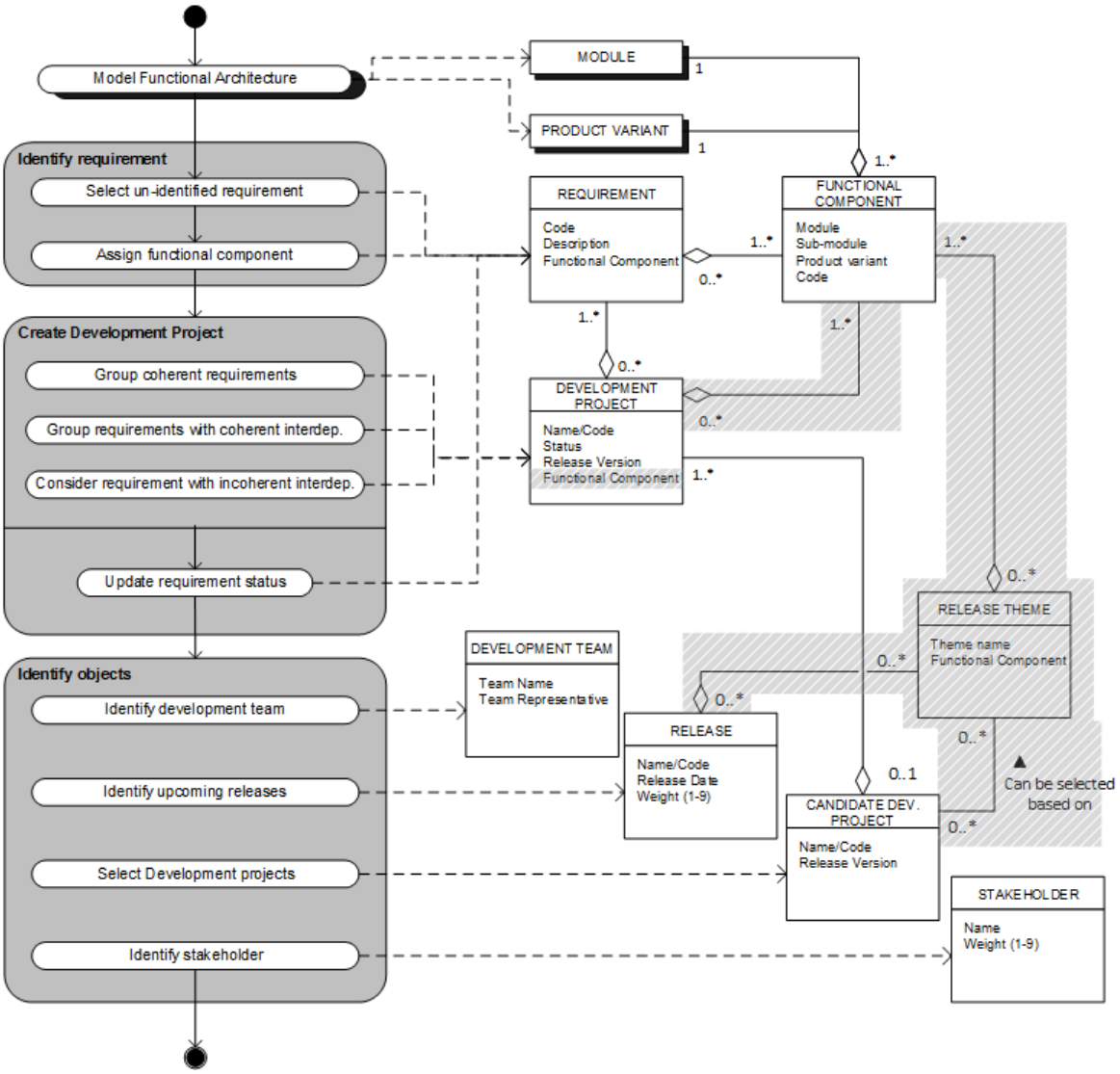


Figure 34: FAF integration in the Release Planning Process

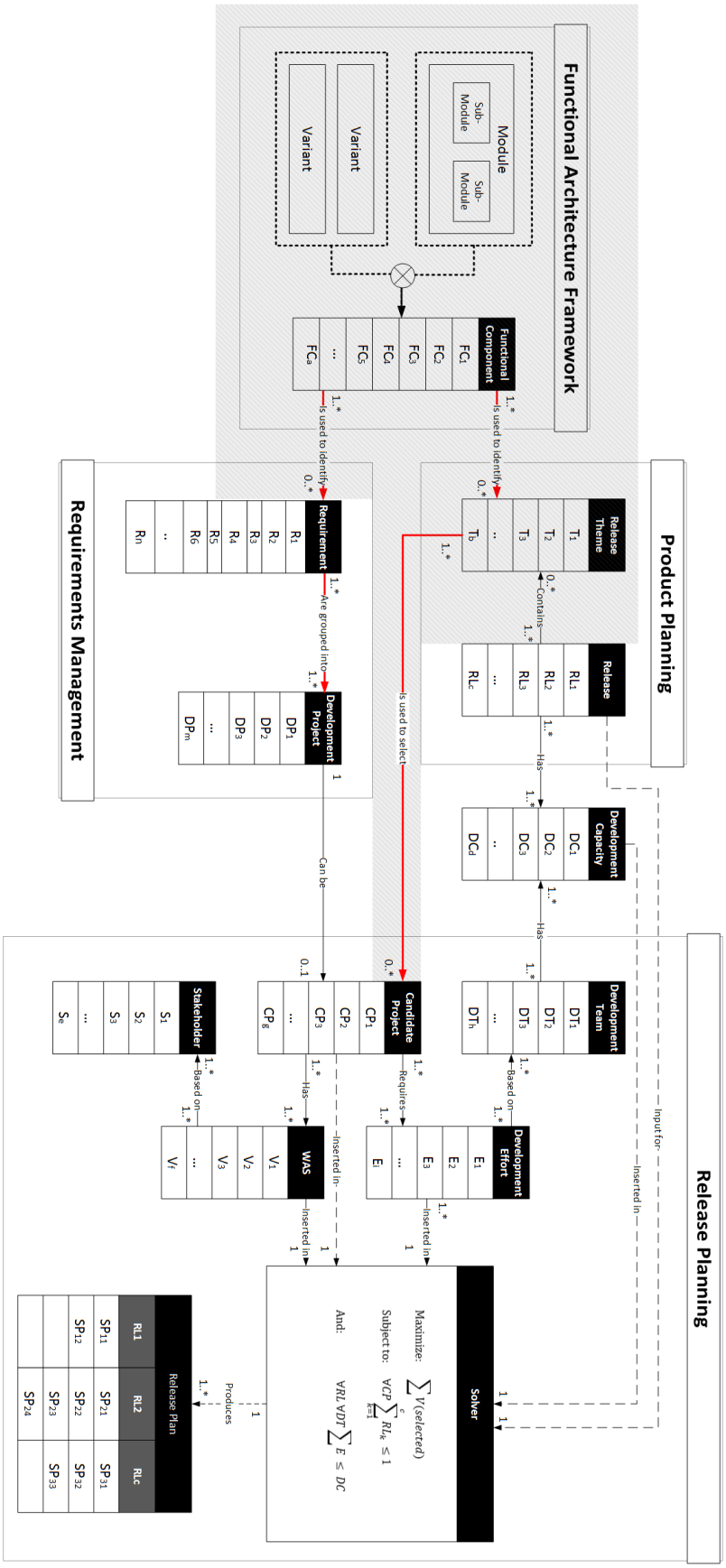


Figure 35: Artifact State 5/5

5. METHOD VALIDATION

Due to confidentiality reasons, the contents of this chapter are hidden in this public version

6. CONCLUSION

In this chapter, the research will be evaluated and the main research question will be answered. The main research question of this research was formulated as follows:

"How can the Functional Architecture Framework assist in a product software release planning?"

In chapter 6.1, the main research question is answered though answering three sub-research questions. In chapter 6.2 the method and the method establishment are reflected, and in chapter 6.3, the limitations of this method and the future work in this domain are described.

6.1. CONTRIBUTIONS AND EVALUATION

In order to answer the main research question, three sub research questions have been formulated. In this chapter each sub question is briefly evaluated and answered.

RQ 1 - What are the aspects of a product software release planning?

For this sub research question, different literature sources regarding requirements prioritization and requirement selection have been studied. Also, the release planning process of a product software company has been analyzed. Based on the literature study and the business environment analysis, a super product software release planning method has been constructed. This super product software release planning provided insight in four important aspects of product software release planning.

The first aspect is the use of development projects (chapter 4.2.1). Development projects consist of one or more (relating) requirements that represent a concrete chunk of functionality that is understandable for each stakeholder. Grouping requirements into development projects has a number of benefits:

1. It reduces the list of candidate requirements, which makes it more likely for stakeholders to participate in the valuation process;
2. The content and the context are understandable for each stakeholder, which helps to better assign a value to a project;
3. In reduces the number of interdependencies that have to be taken into account when planning the release, since many interdependencies are between requirements within a project;
4. It makes the release planning easier, since there are not as many options as when the planning needed to be done on requirement level.

The second aspect of release planning is the object identification (chapter 4.2.2). Objects within the release planning context are entities that are release independent. The objects which need to be identified before the release planning is initiated are:

- The development teams available within an organization;
- The upcoming releases that need to be planned;
- The set of candidate development projects;
- The stakeholders to include in the release planning process.

In order to plan development projects into a release, the planning criteria can't be plucked out of thin air. It is necessary to know certain planning parameters that relate to the development projects and to the release (chapter 4.2.3). The fourth aspect of release planning, are the planning parameters which are currently identified as:

- The value for the joined stakeholder opinion of a development project in each release;
- The cost per development team to develop a development project. The cost can be either in a number of man-hours or in a currency. These numbers need to be as specified as possible, preferably per development team or department involved in the development of the development project;
- The development capacity (in man-hours or money) per development team that is available in the time span of each release to be planned.

The fourth and last aspect of release planning is the automated release planning. When all planning parameters are known, a mathematical optimization algorithm (i.e. linear programming) can be used to calculate the optimal subset of development projects to develop in each release. This optimization algorithm allows through numerous calculations to obtain the highest customer satisfaction, while not exceeding the development capacity of each development team in each release (chapter 4.2.4).

RQ 2 - How do the aspects for product software release planning relate to the Functional Architecture Framework?

The main principle of the FAF is to identify functional components, based on the functional architecture and the variants of the software product. The functional components each represent a functional area within the software product and provide a uniform way to identify and discuss different abstraction levels of a software product.

For this sub research question, the aspects found in RQ 1 were compared to the functional components from the FAF. This comparison resulted in a relation of two product software release planning aspects with the functional components of the FAF (chapter 4.3).

Requirement/Development Project

The FAF was originally designed to identify and organize incoming requirements based on the affecting functional components. These functional components can also be used to identify development projects from the set of identified requirements, since a development project is a group of requirements addressing a similar functionality.

Product roadmap

The product roadmap is a document or a graph that represents the strategic direction of the product. The product roadmap of consist of a number of release themes. It could be useful to link one or more functional components to a release theme, so the candidate development projects can easily be selected by release theme.

RQ 3 - How does the relation between product software release planning and the Functional Architecture Framework (RQ2), contribute to the assistance of the Functional Architecture Framework in product software release planning?

The power of the FAF is the (functional) decomposition of a software product (using Functional Architecture Modeling) into sub-modules, and using these sub-modules together with the available product variants to identify all functional components of a software product (chapter 1.1.3).

These functional components are currently used for requirement identification and requirement organizing. This is done by linking an incoming requirement to a functional component so that requirements involving a similar functionality are easily found. During the development of the main artifact, it became clear that the functional components were also invaluable in the release planning process and

the product road mapping process. The following activities in the optimal release planning processes require the use of functional components (chapter 4.3):

- **Release theme identification**; the content of a release is determined from the product roadmap. During the creation of a product roadmap, a theme is determined per release. A theme can affect one or more functional components of a software product.
- **Development project identification**; In order to reduce the list of requirements and to create concrete pieces of functionality, requirements should be grouped into development projects which affect at least one functional component. In order to select which requirements to include in a development project, the functional component can be used to select the candidate requirements for a development project.
- **Candidate development project identification**; Based on the functional components belonging to a certain release theme, a subset of the available development projects can be selected that will be used in the release planning process.

The functional components which are produced by de Functional Architecture Framework is currently used for requirement management, but can be considered a valuable aspect in the complete domain of software product management.

In Figure 36, the relation of the FAF with the SPM business functions Requirements Management, Release Planning, and Product Planning are visible. Figure 36 shows that a functional component is constructed from the module, sub-module, and the variant of a software product. The functional component is adopted by the release theme and by the requirement (who both can be linked to one or more functional components).

The red line illustrates how the FAF contributes to product software release planning; based on the releases to be planned (determined by the product manager), the release theme(s) is/are selected. Because the release theme(s) have one or more functional components linked to it, the development projects regarding those functional components can be selected as candidate development project. In the situation where an organization does not use release themes, the candidate development project list can also be constructed by the product manager by manually selecting functional components. When the candidate development project list is created, the FAF doesn't have any assistance in the further release planning aspects.

Figure 36 shows that planning parameters the development capacity (per team for each release), candidate project development effort (per team for each candidate development project), candidate project WAS (per candidate development project for each release) are used in a computational solver. The solver calculates the optimal release planning based on the following formula (chapter 4.2.4.1):

Maximize: $\sum V(selected)$

Subject to: $\forall CP \sum_{k=1}^{c} RL_k \leq 1$

And: $\forall RL \; \forall DT \sum E \leq DC$

The goal of the solver is to maximize the sum of the WAS values of the selected candidate development projects. The result of the solver is only valid when:

- A candidate project is only selected for one release ($\forall CP \sum_{k=1}^{c} RL_k \leq 1$)
- The effort per development team in each release, does not exceed the development team capacity for that release ($\forall RL \; \forall DT \sum E \leq DC$)

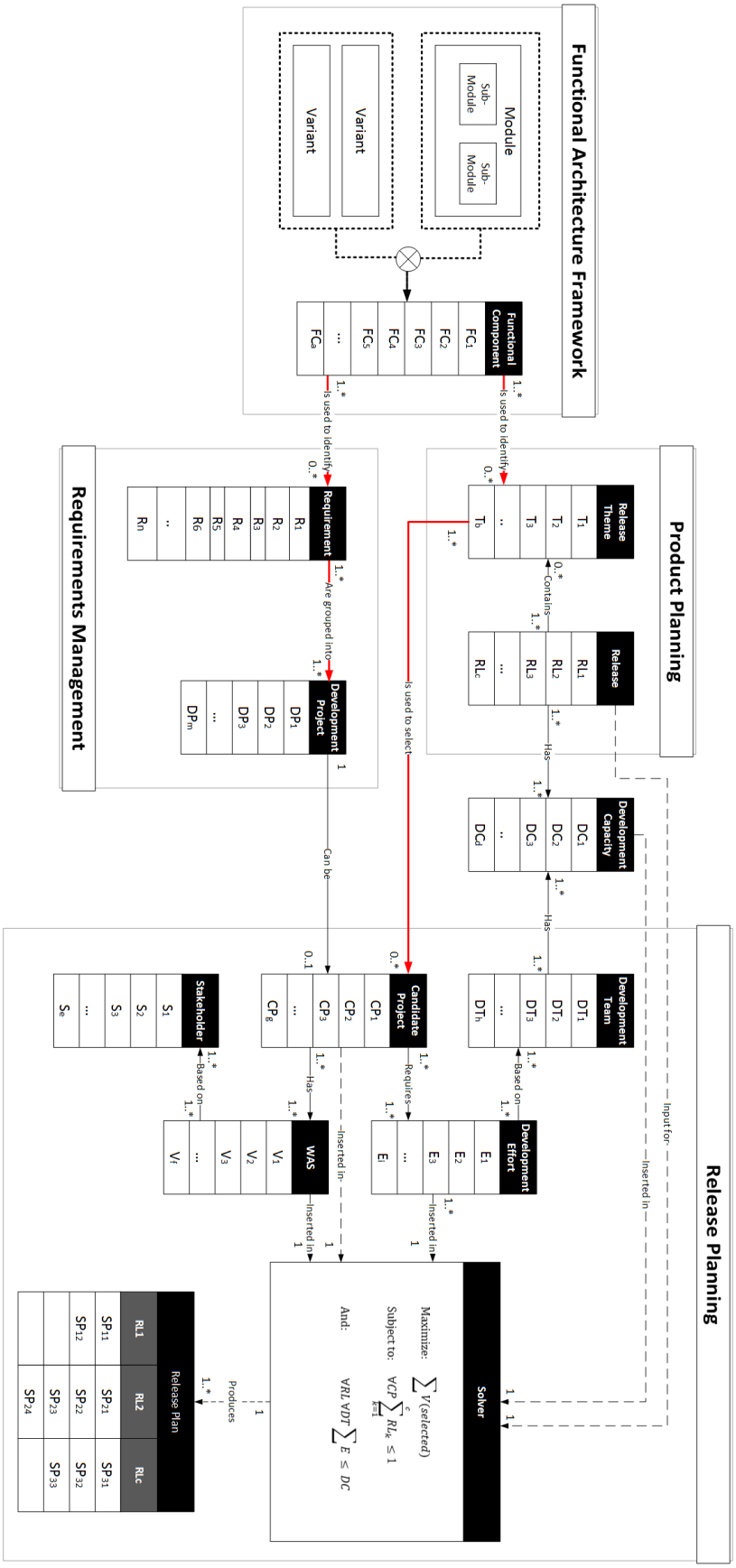


Figure 36: Functional Architecture Framework assistance in product software release planning

The contribution to the business environment is a constructed super product software release planning method. This super method uses the functional components from the Functional Architecture Framework to specify candidate requirements for the upcoming releases. These requirements are grouped into development projects, which get a priority and cost parameter attached. The new method includes a Linear Programming solution in order to find the optimal subset of development projects to develop in the next N releases. The optimal subset consist of the development projects which produce the highest customer value, while not exceeding the development capacity of the development projects of a release.

The scientific contribution of this research is the extension of the Functional Architecture Framework in the Software Product Management domain. This extension is based on the application of the functional components in the business functions Product Planning and Release Planning of the SPM Competence Model. This research also provides more insights on automated software release planning. The insights on automated release planning are:

- What methods are useful to derive the right data for automated software release planning;
- What manual activities are required to prepare the derived data for automated software release planning;

These insights can be used for a baseline to develop a better optimization algorithm for automated release planning, including interdependency support (more on this in chapter 6.3.).

Another scientific contribution is the use of development projects. Using development projects in release planning instead of requirements have a number of advantages. A development project is a group of requirements involving a similar functionality. Grouping these requirements result in a concrete (and preferably independent) chunk of functionality, which is easier for both internal- and external stakeholders to understand and valuate. Using development projects also reduces the list of options to include in a release. This has three advantages:

- The processing time to calculate the optimal release planning is reduced
- It is more likely that stakeholders will participate in the valuation of development projects
- The number of interdependencies will be reduced

In scientific domain, there is only a single paper which shortly mentions the use of development projects, while the use of development projects give a viable structure and overview to the release planning process.

6.2. REFLECTION

During the creation of this method, two somewhat conflicting views were used to balance between theoretical completeness and business acceptance of the method.

In order to strive for the optimal theoretical release planning method, methods that improve validity of the parameters should be adopted in the super method. A method that was left out on purpose was AHP, which improves the quality and validity of the stakeholder's valuation, but greatly increases the number of required questions with the risk that stakeholders end their survey before they are finished.

For the cost estimation methods in chapter 2.2, only three methods were evaluated. This is due to the fact that software cost estimation methods are very extensive, and with the scope of this research, it was considered not relevant to do a complete research in this domain. In order to improve the release planning method created in this research, other development cost estimation methods could be evaluated and assessed in the business domain. A more accurate cost estimation, results in a more accurate release planning.

The optimization model presented in this research must be considered a basic model for software release planning. It still requires future development, since the interdependencies between development projects are not programmed into the solver (more on this in chapter 6.3). This optimization model is also based on Microsoft Solver Foundation. It was not in the scope of this research to evaluate the all the possible solvers available. There could already be solvers available that have a better fit with the goals of software release planning.

6.3. LIMITATIONS AND FUTURE RESEARCH

During the construction of the method, several issues were encountered and decisions were made that have a possible limitation on the new release planning method that was constructed in this research.

Interdependencies

Even though development projects reduce the number of interdependencies, there is always the possibility some interdependencies still remain. The current state of the optimization model is not capable for solving the release planning containing interdependencies (excluding the TEMPORAL interdependency). This is because the use of interdependency makes the model non-linear, which requires a more sophisticated optimization model. Due to the timeframe of this research, it was not possible to develop the optimization model further than the current state.

In order to have the most optimal level of automated software release planning, more research is required on release planning with non-linear variables.

Including quality requirements into development projects

In this research, the focus was on functional requirement, while there are also quality requirements that are focused on the technical aspect of a software product (Nuseibeh & Easterbrook, 2000) (Berander & Andrews, 2005). There are seven categories for quality requirements (Bass, Clements, & Kazman, 2012):

- Availability
- Interoperability
- Modifiability
- Security
- Testability
- Usability

A recent thesis of Seele (2013) has started the research in this domain by successfully mapping the technical architecture of open-source internet browsers to the functional architecture of open-source internet browsers. This mapping will possibly create the opportunity to identify quality requirement and include them in the release planning process described in this thesis.

Validity of the release planning method

Since this release planning method is merely validated in the business environment of AFAS software, the method is not generalizable. Future research in the product software field is needed to demonstrate the validity and generalizability of the method in the product software domain.

Specification level of requirements

During the case study validation there was a constant struggle between having a large set of specific requirement, or having a smaller set with requirements described on a global level. Both approaches have

positive and negative effects. For example specific requirements may result in a better effort estimation from the different development teams, while the magnitude of the stakeholder survey is increased (which may influence the stakeholders' will to participate). In the case study performed for this research, globally described requirements were used in order to increase stakeholder survey participation. Since the requirements were only described globally, it was difficult for the development team to make an effort estimation.

Future research may be required in the domain of requirements management, in order to have the optimal level of requirement specification. The optimal level of requirement specification involves the specification to be:

- Interesting for the external stakeholder to identify with the requirement;
- Understandable for the development team to make an accurate effort estimation;
- Understandable for the product manager so interdependencies can be identified properly.

Value influencing interdependencies

The use of the value influencing interdependencies (ICOST & CVALUE) are vague interdependencies to measure, because:

- It is not possible to say whether for example R1 increases the value of R2 by 30% or decreases the cost by 40%, without a solid argument. A product manager at AFAS also pointed out that they don't use this interdependency at all due to its complexness;
- The order of operations is not defined in any literature; If R1 increases the value of R2 by 15%, and R3 decreases the value of R2 by 20%, in which order are the calculation made? Different order of operations have different results (assuming that these requirements work with ratio values).

Since these interdependencies has the potential to contribute to a more accurate release planning can have, more research is required on the use of these two interdependencies.

REFERENCES

Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice, 3/E.* Pearson Education.

Bebensee, T., van de Weerd, I., & Brinkkemper, S. (2010). Binary Priority List for Prioritizing Software Requirements. In *Requirements Engineering: Foundation for Software Quality* (pp. 67-78). Berlin: Springer.

Bekkers, W., van de Weerd, I., Spruit, M., & Brinkkemper, S. (2010). A Framework for Process Improvement in Software Product Management. In *Systems, Software and Services Process Improvement* (pp. 1 - 12). Berlin: Springer.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative search. *Communications of the ACM, 18*(9), 509-517.

Berander, P., & Andrews, A. (2005). Requirements Prioritization. In *Engineering and Managing Software Requirements* (pp. 69-94). Berlin: Springer Verlag.

Bertsekas, D. P. (1999). *Nonlinear programming.* Athena Scientific.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *IEEE COMPUTER*, 61-72.

Boehm, B., Abts, C., & Chulani, S. (2000). Software development cost estimation approaches—A survey. *Annals of Software Engineering*, 177-205.

Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., & Shelby R. (1995). Cost models for future software life cycle processes: COCOMO 2.0. *Annals of software engineering, 1*(1), 57-94.

Briand, L. C., & Wieczorek, I. (2002). *Resource Estimation in Software Engineering.* New York: John Wiley.

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

Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., & Natt och Dag, J. (2001). An industrial survey of requirements interdependencies in software product release planning. *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on.*

Clark, B., Devnani-Chulani, S., & Boehm, B. (1998). Calibrating the COCOMO II post-architecture model. In *Proceedings of the 20th international conference on Software engineering* (pp. 477-480). IEEE Computer Society.

Davis, A. M. (2003). The Art of Requirements Triage. *Computer, 36*(3), 42-49.

Ebert, C. (2007). The impacts of software product management. *Journal of Systems and Software, 80*(6), 850-861.

Hauser, J. R., & Clausing, D. (1988). *The House of Quality.* Boston, Mass.: Harvard Business School, Pub. Division.

Herrmann, A., & Davena, M. (2008). Requirements prioritization based on benefit and cost prediction: An agenda for future research. In IEEE, *International Requirements Engineering, 2008. RE'08. 16th IEEE* (pp. 125-134).

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS quarterly, 28*(1), 75-105.

Humphrey, W. S. (2000). *The Personal Software Process (PSP).* Carnegie Mellon University: Software Engineering Institute.

IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*.

Jung, H. (1998). Optimizing value and cost in requirements analysis. *Software, IEEE, 15*(4), 74-78.

Karlsson, J., & Ryan, K. (1997). A cost-value approach for prioritizing requirements. *Software, IEEE, 14*(5), 67-74.

Karlsson, J., Wohlin, C., & Regnell, B. (1997). An evaluation of methods for prioritizing software requirements. *Information and Software Technology, 39*, 939-947.

Karlsson, L., Berander, P., Regnell, B., & Wohlin, C. (2004). Requirements Prioritisation: An Experiment on Exhaustive Pair-Wise Comparisons versus Planning Game Partitioning. *Proceedings 8th Conference on Empirical Assessment in Software Engineering.* Edinburgh.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems.* Berlin: Springer.

Leffingwell, D., & Widrig, D. (2000). *Managing software requirements : a unified approach.* Reading: Addison-Wesley.

Li, C., van den Akker, J. M., Brinkkemper, S., & Diepen, G. (2007). *Integrated Requirement Selection and Scheduling for the Release Planning of a Software Product.* Utrecht: Utrecht University.

Morris, J., Lee, G., Parker, K., Bundell, G. A., & Lam, C. (2001). Software component certification. *Computer, 34*(9), 30-36.

Nuseibeh, B., & Easterbrook, S. (2000). Requirements Engineering : A Roadmap. *Proceedings of the Conference on the Future of Software Engineering* (pp. 35-46). ACM.

Pachidi, S., & Brinkkemper, S. (2010). Functional Architecture Modeling for the Software Product Industry. In *Software Architecture* (pp. 198 - 213). Berlin: Springer.

Pressman, R. S., & Ince, D. (1992). *Software engineering: a practitioner's approach.* New York: McGraw-hill.

Racheva, Z., Davena, M., & Buglione, L. (2008). Supporting the dynamic reprioritization of requirements in agile development of software products. *Software Product Management, 2008. IWSPM'08. Second International Workshop on*, 49-58.

Royce, W. W. (1970). Managing the development of large software systems.

Ruhe, G. (2005). Software release planning. *Handbook of software engineering and knowledge engineering, 3*, 365-394.

Ruhe, G., & Saliu, M. O. (2005). The Art and Science of Software Release Planning. *Software, IEEE*, 47-53.

Ruhe, G., Eberlein, A., & Pfahl, D. (2003). Trade-off Analysis for Requirements Selection. *International Journal of Software Engineering and Knowledge Engineering, 13*(4), 345-366.

Russel, S. J., Norvig, P., Canny, J. F., Malik, J. M., & Edwards, D. D. (1995). *Artificial intelligence: a modern approach.* Prentice hall Englewood Cliffs.

Saaty, T. L. (1990). How to make a decision: The analytic hierarchy process. *European Journal of Operational Research, 48*(1), 9-26.

Salfischberger, T., van de Weerd, I., & Brinkkemper, S. (2011). The Functional Architecture Framework for organizing high volume requirements management. *Software Product Management (IWSPM), 2011 Fifth International Workshop on*, (pp. 17 - 25).

Saliu, O., & Ruhe, G. (2005). Supporting Software Release Planning Decisions for Evolving Systems. *Software Engineering Workshop, 2005. 29th Annual IEEE/NASA*, (pp. 14-26).

Seele, W. (2013). *Modeling Functionality in Software Architecture.*

Stellman, A., & Greene, J. (2005). *Applied software project management.* O'Reilly Media, Incorporated.

Ullah, M. I., & Ruhe, G. (2006). Towards Comrehensive Release Planning for Software Product Lines. *Software Product Management, 2006. IWSPM'06. International Workshop on.*

University of Southern California. (2013, 03 28). *Center for Systems and Software Engineering*. Retrieved from COCOMO® II Cost Driver and Scale Driver Help: http://sunset.usc.edu/research/COCOMOII/expert_cocomo/drivers.html

Utrecht University. (2013, 02 01). *ASL Foundations - Scope Change Management*. Retrieved from IT and Software Product Management: http://www.cs.uu.nl/wiki/bin/view/Spm/AslScopeChange

Utrecht University. (2013, 04 11). *Release Planning*. Retrieved from Software Product Management: http://www.cs.uu.nl/docs/vakken/mspm/index.php?id=1&subid=4

Utrecht University. (2013, 03 25). *Software Product Management Competence Model*. Retrieved from IT and Software Product Management: http://www.cs.uu.nl/wiki/bin/view/Spm/RequirementsPrioritization

van de Weerd, I., & Brinkkemper, S. (2008). Meta-modeling for situational analysis and design methods. In M. R. Syed, & S. N. Syed, *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications* (pp. 38-58). Hershey: Idea Group Publishing.

van de Weerd, I., & Brinkkemper, S. (2010, September). Software Product Management Lecture Notes. Utrecht University.

van den Akker, J. M., Brinkkemper, S., Diepen, G., & Versendaal, J. (2005). Determina-tion of the next release of a software prod-uct: an approach using integer linear programming. *Proceeding of the 11th Inter-national Workshop on Requirements Engi-neering: Foundation for Software Quality*, 119-124.

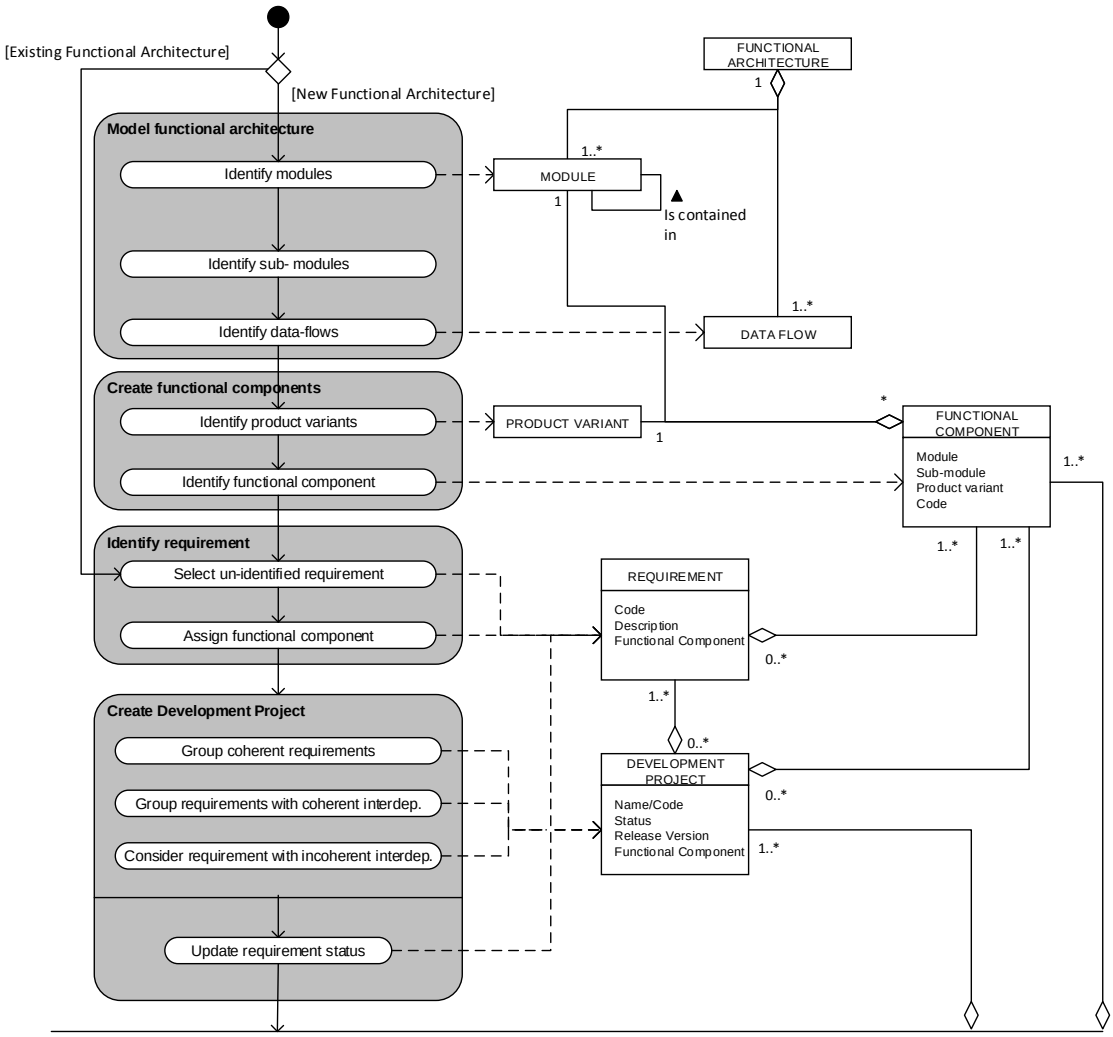
van den Akker, M., Brinkkemper, S., Diepen, G., & Versendaal, J. (2008). Software product release planning through optimization and what-if analysis. *Information and Software Technology, 50*(1-2), 101-111.

Wiegers, K. E. (2000). Stop promising miracles. *Software Development*.

Wiegers, K. E. (2003). *Software requirements : practical techniques for gathering and managing requirements throughout the product development cycle.* Redmond: Microsoft Press.

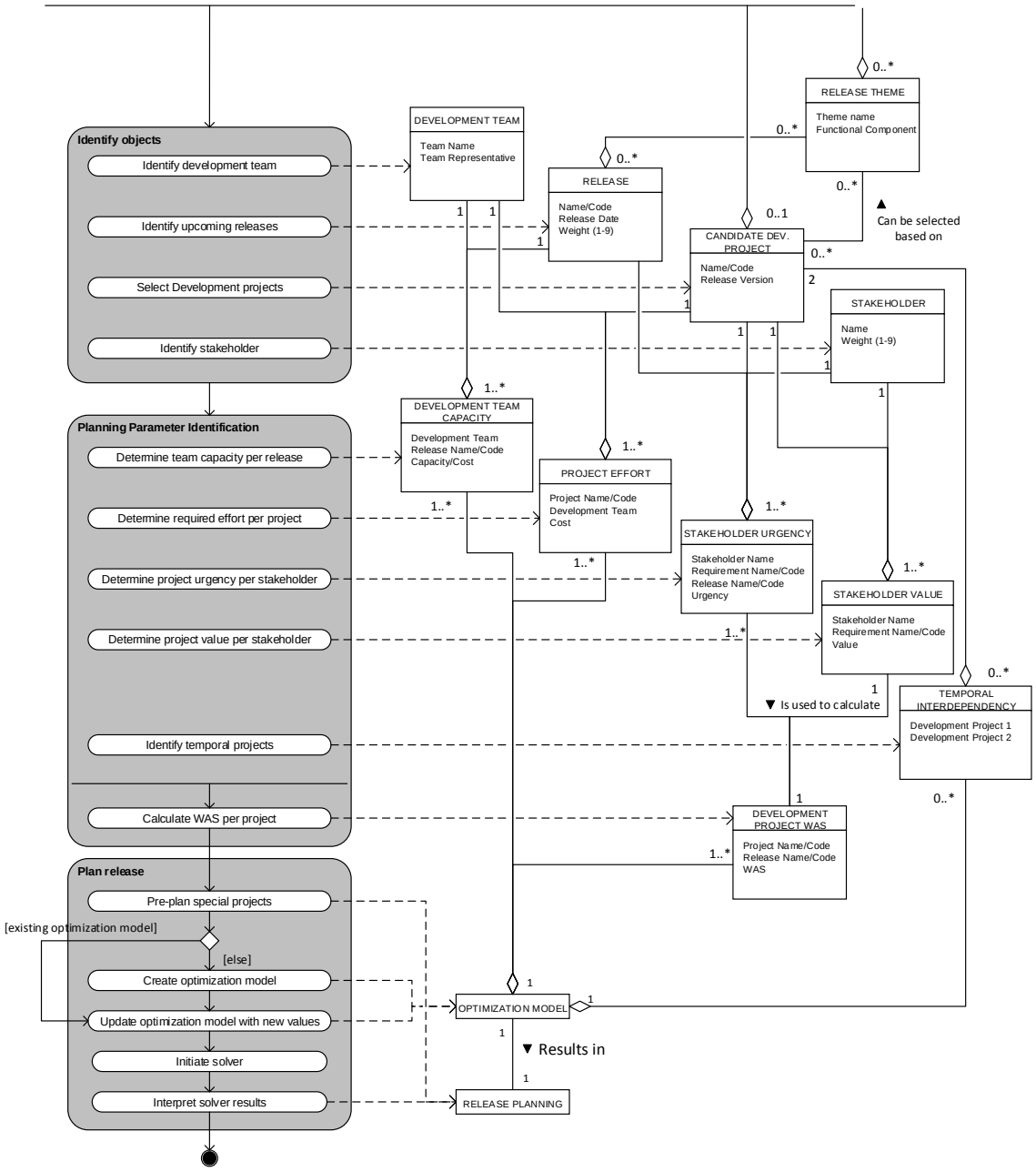
Xu, L., & Brinkkemper, S. (2005). Concepts of product software: Paving the road for urgently needed research.

APPENDIX A (FULL PDD OF THE OPTIMAL RELEASE PLANNING METHOD)



Next page

Previous page



APPENDIX B (QUESTIONNAIRE REQUIREMENT GATHERING)

Introduction

- Project introduction
    - Introduction interviewer
    - Goal: gather requirements for the optimization of the fiscal module of AFAS Profit
    - Scope: the interview only focuses on new requirements in line with a potential process improvement. Requirements involving the current product do not belong in the scope
- Company introduction
    - Function and background interviewee
    - Background company (number of employees, number of clients, etc.)
    - Relationship with AFAS

Main

1. How does the company receives the necessary data from private/business clients?

1.1. How does the company determine what documents the client has to deliver?

2. How does the company process the received data?

2.1. Describe each step in the workflow, with roles and activities

3. Current process improvements that the company has undertaken
4. How does the interviewee thinks the process can be improved?

4.1. [show mockups to improve the creativity of the interviewee]

5. Recap improvements

Ending

- Invite the interviewee to contact interviewer if anything comes to mind
- Invite the interviewee to participate in the questionnaire process for WAS determination

APPENDIX C (WAS COMPARISON)

All stakeholder weighted equally

Project	1	2	3	4	5	6	7	8	9
6 – 12 months	3,66	3,45	2,75	4,20	2,66	2,41	4,37	4,20	2,83
12 – 18 months	3,99	3,41	3,24	4,33	2,95	2,95	4,66	4,37	3,45
18 + months	4,33	3,74	3,54	4,83	3,08	3,45	4,83	4,53	3,79
Average	**3,99**	**3,54**	**3,18**	**4,45**	**2,90**	**2,94**	**4,62**	**4,37**	**3,36**

Only internal stakeholders

Project	1	2	3	4	5	6	7	8	9
6 – 12 months	4,10	3,66	2,46	4,26	2,24	2,67	4,46	4,37	2,86
12 – 18 months	4,46	3,41	2,91	4,21	2,57	3,02	4,66	4,37	3,16
18 + months	5,02	4,06	3,06	4,86	2,57	3,56	4,86	4,57	3,61
Average	**4,53**	**3,71**	**2,81**	**4,44**	**2,46**	**3,08**	**4,66**	**4,44**	**3,21**

Only external stakeholders

Project	1	2	3	4	5	6	7	8	9
6 – 12 months	2,49	2,61	3,33	3,80	2,61	1,54	3,80	4,87	3,33
12 – 18 months	3,33	3,09	4,28	4,28	2,97	2,14	4,75	5,34	4,28
18 + months	3,68	3,09	4,75	4,75	3,33	2,73	4,75	5,34	4,75
Average	**3,17**	**2,93**	**4,12**	**4,28**	**2,97**	**2,14**	**4,43**	**5,19**	**4,12**

APPENDIX D (OML CODE & APPLICATION)

This appendix shows the OML code as it is used by the latest version of the optimization model. This OML code can only be used in Microsoft Solver Foundation.

```
Model[
	Parameters[
		Sets[Integers[0, Infinity]],
		Projects,
		Releases,
		SoftORs
	],

	Parameters[
		Integers[0, Infinity],
		Cost[Projects],
		Value[Projects, Releases],
		MaxCost[Releases],
		Cost2[Projects],
		MaxCost2[Releases],
		SoftOR1[SoftORs],
		SoftOR2[SoftORs]
	],

	Decisions[
		Booleans,
		Decision[Projects, Releases]
	],

	Constraints[
		MaximumOneProject -> Foreach[
			{iter3, Projects},
			Sum[
				{iter4, Releases},
				Decision[iter3, iter4]
			] < 2
		],

		MaximumCost -> Foreach[
			{iter1, Releases},
			Sum[
				{iter2, Projects},
				Decision[iter2, iter1] * Cost[iter2]
			] <  MaxCost[iter1] + 1
		],

		MaximumCost2 -> Foreach[
			{iter1, Releases},
			Sum[
				{iter2, Projects},
			Decision[iter2, iter1] * Cost2[iter2]
			] <  MaxCost2[iter1] + 1
		],

		SoftOR -> Foreach[
			{iter1, Releases},
			{iter2, SoftORs},
			And[
				Decision[SoftOR1[iter2], iter1] == 1,
				Decision[SoftOR2[iter2], iter1] == 1
			]  == 0

		]
```

```
        ],


        Goals[
               Maximize[
               Goal -> Annotation[Sum[
                      Foreach[
                             {iter10, Releases},
                             Foreach[
                                    {iter11, Projects},
                                    Decision[iter11, iter10] * Value[iter11, iter10]
                             ]
                             ]
                      ], "order", 0]
               ]
        ]
]
```

The optimization model code produces:

- A set called Project;
- A set called Releases;
- A parameter called Value;
- A parameter called Cost1 (cost for team 1);
- A parameter called Cost2;
- A parameter called MaxCost1 (capacity team 1);
- A parameter called MaxCost2;
- Two parameter called SoftOR1 & SoftOR2 (for the TEMPORAL interdependencies.

For the implementation of this model, Microsoft Excel was used. By means of screenshots, an explanation is given of how the optimization model is applied in Microsoft Excel.

Although the code can be copy/pasted into the solver, the data range has to be defined manually. This is done though the Binding Editor of the Microsoft Solver Foundation Excel plugin (Figure 37). It is important that the data is stored in columns, with an optional column header like in Table 26.

Set Title	Set Title	Parameter Title
Set Value	Set Value	Parameter Title
Set Value	Set Value	Parameter Title
Set Value	Set Value	Parameter Title
Set Value	Set Value	Parameter Title

Table 26: Data Structure

In the text box Table/Range, the Excel data column range must be defined. Next, the used sets must be defined where after the column for the parameter must be selected.

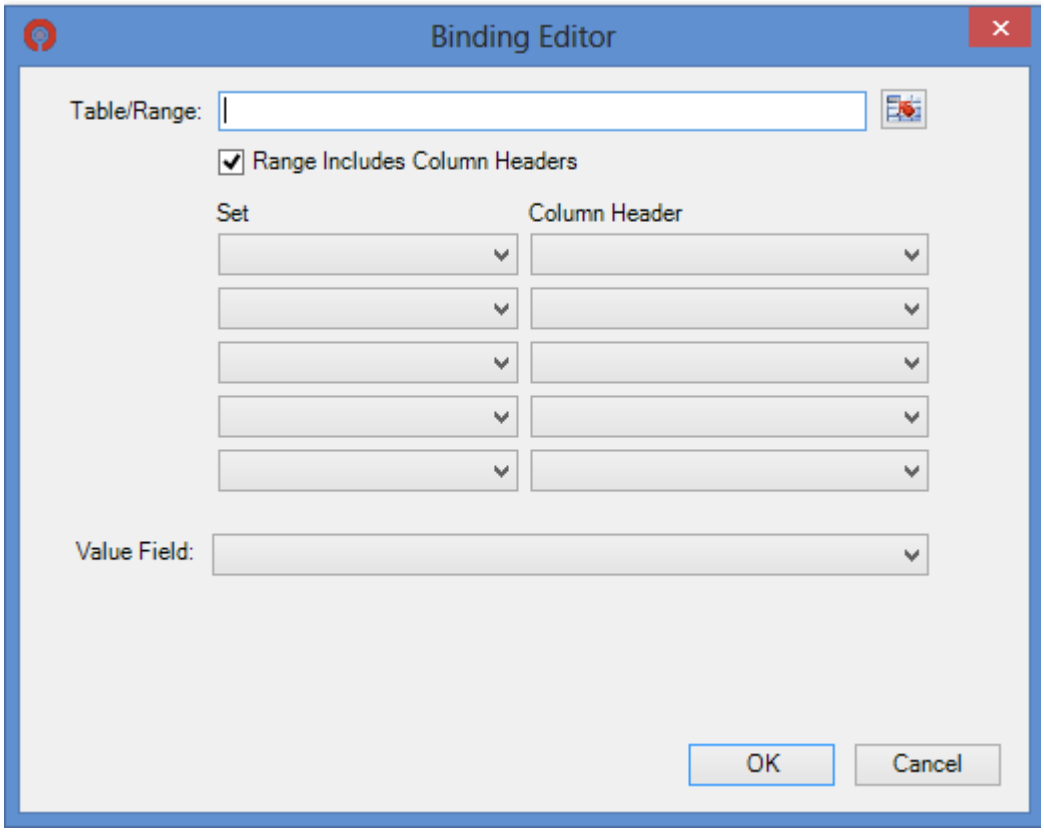


Figure 37: Binding Editor

In Figure 38, Figure 39, Figure 40, and Figure 41, an example is shown of how to use the Binding Editor. In Figure 38, the parameter Value is specified for each combination of a project in a release by identifying the Projects set (red line) and the Releases set (blue line). The actual parameters are stored in the Value column, which is specified in the Binding Editor in the drop down box “Value Field”.

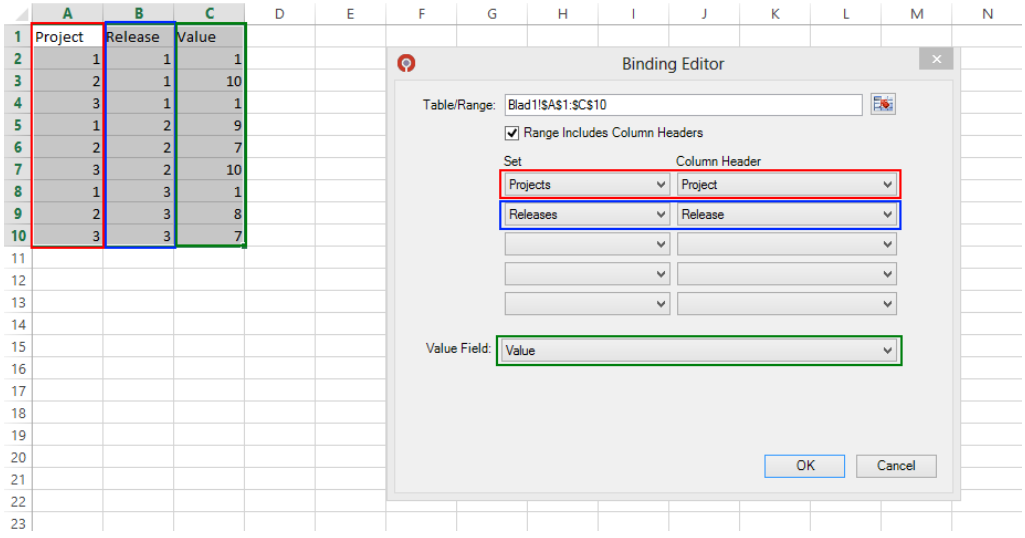


Figure 38: Binding of the Value parameter

In Figure 39 and Figure 40, the effort per project for each team is defined.

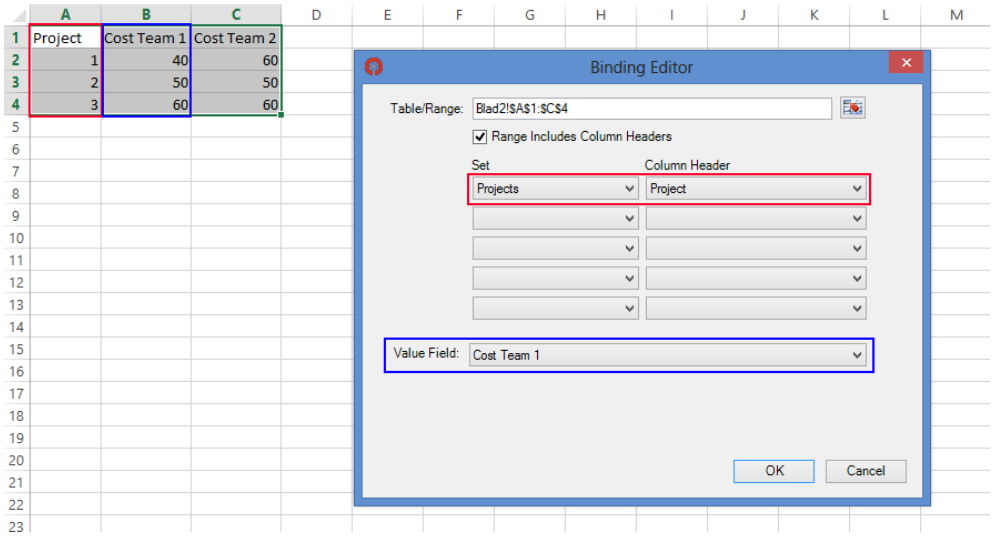


Figure 39: Binding of the Cost1 parameter

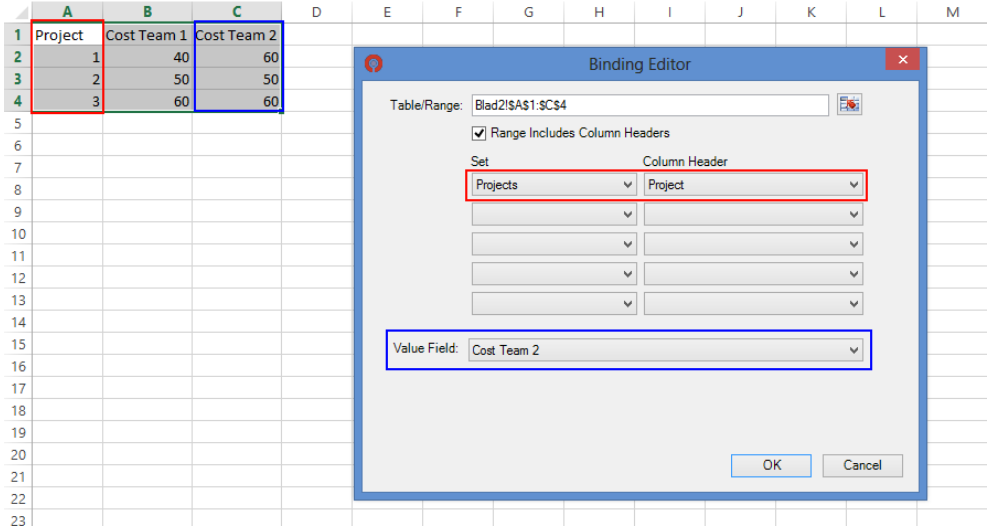


Figure 40: Binding of the Cost2 parameter

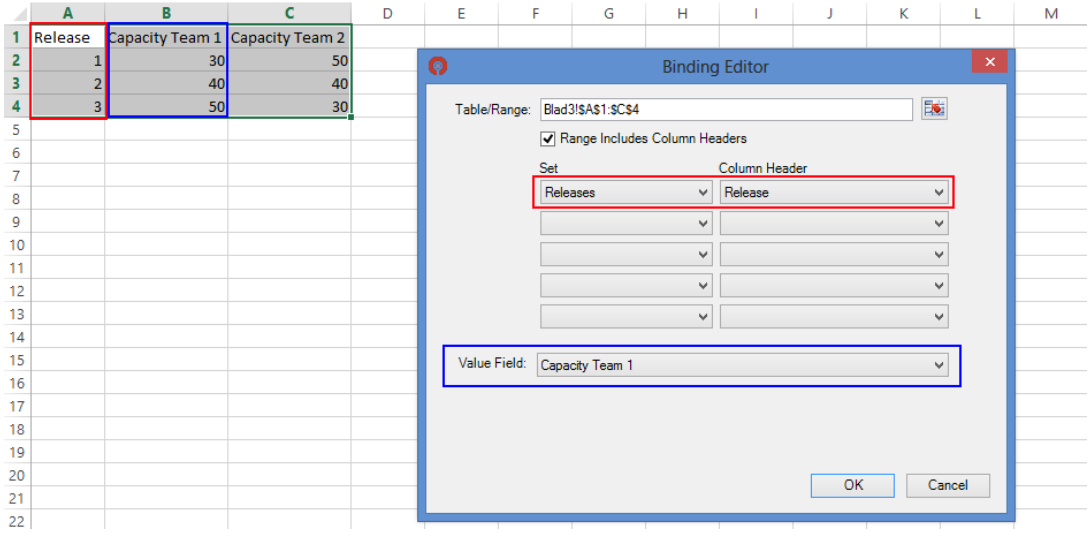


Figure 41: Binding of the Capacity parameter