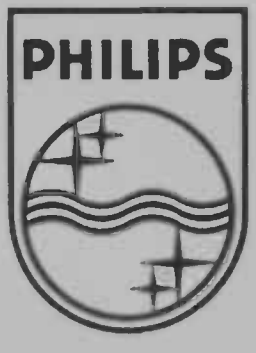
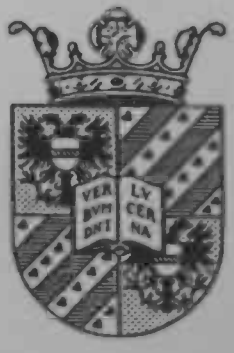
On the funding and economic evaluation of software product line practices

A case study on a software product line initiative

Rodney Heinkens
Philips Medical Systems
University of Groningen
August 19th, 2005





On the funding and economic evaluation of software product line practices

A case study on a software product line initiative

Author
Rodney Heinkens

Supervisors
Prof. Dr. Ir. Jan Bosch
Drs. Albert-Jan Abma
Drs. Stef Zaicsek

Case
Philips Medical Systems

University
University of Groningen
Faculty
Mathematics and Natural Sciences

Deadline
August 12th, 2005

DECLARATION
I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

– Rodney Heinkens –

Best/Groningen, 2005

ABSTRACT

Software product lines

Software product lines provide a means for realizing (1) reduction of development costs, (2) reduction of maintenance costs, (3) shorten time-to-market, and (4) increase the overall quality. Software product lines are defined as "*a set of software-intensive systems sharing a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way*". The key insight is to exploit the commonalities between different products. These commonalities are captured by core assets. Core assets provide a certain degree of variability so that these assets can be used in different contexts. The product line scope defines which products one can derive from the core asset base, i.e. the pool of core assets. It thus specifies implicitly what variability these core assets need to provide. Core assets encompass, among others, a product line architecture, and reusable components. The product line architecture is a reference architecture applicable to all the products included in the product line scope. Each product has its own derivation of the product line architecture, designed for the specific needs of that particular product.

Obviously, the products in the product line scope differ from each other also. In order to deal with variability that cannot be captured in core assets, product-specific assets need to be developed to augment the provided functionality. These product-specific assets need to be designed to fit in the software architecture of the product.

The software product line processes can be categorized in three main activities, i.e. core asset development, product development, and management. Core asset development is concerned with the product line scope and with the development and evolution of the core asset base. Product development is the process of deriving products from the software product line, making use of both core assets and product-specific assets. Finally, management has to orchestrate both the aforementioned processes, including setting the roadmap for future evolution.

Funding software product line activities

When one institutionalizes a software reuse program, the following four cost types play a role.

- C_{org}, the costs that an organization makes to adopt the software product line practices
- C_{cab}, the development and maintenance costs of the core asset base
- C_{unique}, the development costs of the product-specific assets
- C_{reuse}, the costs of integrating a core asset in a product

With respect to the funding model, only C_{org} and C_{cab} will have to be dealt with. The former is the encapsulation of all the costs related to transition the organization from a traditional approach to product development to a software product line approach to product development. The latter is the encapsulation of the costs related to the development, maintenance, and evolution of the core assets. The other two cost types, C_{unique} and C_{reuse}, are both directly related to a particular product, so these costs can be accounted to the subject products.

When institutionalizing a software product line approach, one has to deal with different issues. First, the initial user of a core asset needs to be compensated for the fact that he has to deal with all the childhood illnesses. Second, the business units need some incentive to adopt core assets into their product development. Third, the business units do not only need some incentive to adopt core assets, but they need some incentive to adopt the latest releases of core assets also. These two issues particularly play a role in organizations where the use of the core assets has not been proven beneficial. Finally, when the core asset development is performed solely by one business unit, this business unit forms the bottleneck with respect to development and maintenance of core assets. Therefore, other business units should be involved in the development and maintenance of these assets also. This requires an incentive also, since

the development of a core asset typically requires more effort than the development of a similar product-specific asset.

The funding model proposed in this thesis is referred to as the adjustable incentive-driven funding model, further referred to as the aid model. The aid model was derived from the flat tax model and the reward-based model. The general idea is that all business units initially are accounted for the same amount. However, their behavior with respect to the four identified issues is taken into account also. By doing so, the required amounts per business unit can differ.

Determining the funds all the business units are accounted for is done through six simple steps. The first step requires that the different types of behavior, i.e. behavior related to the aforementioned issues, are provided a certain value. For this purpose, every type of behavior makes use of a behavior constant. The higher value such a constant has, the more impact the subject issue will have on the actual amount that a business unit has to pay. The second step is concerned with quantifying the behavior of all the different business units. Each business unit has a behavior variable, i.e. a variable related to one of the four aforementioned issues. The lower value a business unit scores for these variables, the more beneficial it will be. The third step determines the score of a business unit by summing up the multiplication of all its behavior variables with the corresponding behavior constants. The fourth step then prescribes to calculate the overall score, i.e. the sum of the scores of all the individual business units. The fifth step is used to calculate the cost factor. It is obtained by dividing the projected costs with the overall score. Finally, the cost factor is multiplied by the scores of each of the individual business units to derive the funds that they will have to provide.

Measuring software product line practices

Institutionalizing and sustaining a software product line effort typically is a hard and bumpy road. Strong management commitment is a requirement for doing it successfully. Management has to, among other things, set goals to strive for. Maybe more importantly, the progress towards meeting these goals has to be measured. If the goals are not achieved, or progress towards doing so is stagnating, measuring properties of a software product line effort makes an organization aware that certain aspects of the software product line effort are not covered properly.

With respect to the three fundamental software product line activities, the following business concerns were identified. First, the product line scope needs to be determined adequately. For instance, a too large product line strains assets beyond their ability to accommodate the required variability. Second, the future evolution of core assets needs to be accommodated through for instance documenting how this evolution can be carried through. Third, the use of core assets should be more beneficial than developing a similar product-specific asset from scratch. This obviously is necessary to justify the existence of the software product line. In addition, the use of core assets needs to be stimulated through providing proper production plans and corresponding support. The production plans prescribe how the core assets can be utilized to develop a certain product. By prescribing how this is done, the assets will be utilized in a more structured and more efficient manner. Furthermore, product developers only are concerned with developing and maintaining products as efficient and as fast as possible, as soon as possible, and with a high as possible profit margin. Finally, since management has to orchestrate the other two fundamental activities, management must be equipped with tools to provide the necessary resources, coordinate, and supervise.

In order to address all the above business concerns, three levels of metrics are introduced. These metrics are categorized according to their relevance to the three typical levels of corporate strategy, i.e. operational, business-level, and corporate-level. These strategy levels exhibit a short-term focus, a mid-term focus, and a long-term focus respectively.

With respect to the operational strategies, four different metrics are proposed. The first two metrics are referred to as the absolute reuse level and the relative reuse level. The absolute reuse level states what percentage of an asset results from existing assets. The relative reuse level states what percentage of the available and suitable assets for a given product are utilized for the development of a product. The additional reuse effort reflects how much effort is required to develop a core asset relative to the effort it requires to develop a similar product-specific asset. Additional effort is required for, among other things, incorporate variability. Integrating a core asset does not come free. It has to be tailored according to the requirements of a given product and integrated into that product. These costs, however, typically are lower than developing a product-specific asset from scratch. The integration effort reflects how much effort is required to tailor and integrate a core asset in a product context relative to the effort it requires to develop a similar product-specific asset.

The second level of strategy, the business-level, comprises a cost-benefit analysis. The focus is on the costs and benefits of a particular product. For the costs, the four cost types described above can be utilized. These costs can be extracted from both the amount a business unit is required according to the institutionalized funding model and the project planning. The former provides C_{org} and C_{cab}, and the latter provides C_{unique} and C_{reuse}. Two different types of benefits were identified, i.e. tangible and intangible benefits. The former can easily be calculated. The tangible benefits comprise the benefits regarding the development costs and the maintenance costs. The intangible benefits can only be estimated, since these are dependent on for instance the market in which an organization operates. To deal with intangible benefits, the use of scenarios is proposed. A scenario is composed of a worst-case and a best-case. A scenario thus contains a bandwidth. The worst-case bound represents which benefits certainly are gained and the best-case represents an optimistic, yet realistic view on the benefits. The cost-benefit analysis as a whole also contains a bandwidth of values.

The third and final metric level addresses the corporate-level strategy. It is concerned with the long-term health of the software product line, which is reflected in for instance the product line scope. Two types of scenarios are used, i.e. architectural scenarios and strategic scenarios. The architectural scenario represents what features are enabled by transforming the product line architecture and the strategic scenario represents future evolutions of the market in which the organization operates, the available technologies, and the products. Each architectural scenario has a certain value in each strategic scenario, dependent on how much value the enabled features of that architectural scenario have in a certain strategic scenario. By analyzing and making assumptions on future evolutions, the product line scope can evolve accordingly.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my family, and in specific my lovely fiancée and my beautiful daughters, for their support and motivation during my internship. In addition, I want to thank my colleague Auke Schotanus for his company during the long drives and for motivating me during the internship. I can only hope that I helped you as much as you helped me.

Further, I would like to thank my supervisors for their support and feedback during my internship. Jan Bosch (University of Groningen) has kept my focus on both the academic foundation of the internship as well as on my personal growth as a researcher. I also want to thank Stef Zaicsek (Philips Medical Systems) for his patience with me and for his positive influences in my personal development. The final supervisor I would like to thank is Albert-Jan Abma (Univetsity of Groningen). He provided me with advice on the business aspects of my internship and he helped me prepare for working as an intern, both at the operational level and at the social-interactive level.

With respect to developing my writing skills, I want to thank Henk Obbink (Philips Research), and Marco Sinnema and Sybren Deelstra (University of Groningen). They provided me with feedback on all relevant aspects, which play a role when writing a thesis, among other things, structure and spelling.

Finally, I want to thank all the people who may have not been mentioned above, but did aid me in successfully fulfilling my internship.

ABSTRACT I

ACKNOWLEDGEMENTS IV

PART I INTRODUCTION 1

CHAPTER 1 INTRODUCTION 2

1. INTERNSHIP AND DELIVERABLES 2
2. DOCUMENT OVERVIEW 2

CHAPTER 2 RESEARCH SET-UP 4

1. TRADITIONAL SOFTWARE DEVELOPMENT ISSUES 4
2. SCOPE 4
3. PROBLEM STATEMENT 5
4. CONCEPTUAL MODEL 6
5. METHODOLOGY 7

CHAPTER 3 PHILIPS MEDICAL SYSTEMS 9

1. ORGANIZATIONAL INFORMATION 9
2. MARKET DEMANDS 10
3. ORGANIZATIONAL STRUCTURE 10
4. SUMMARY 11

PART II REUSING WITH PRODUCT LINES 12

CHAPTER 4 SOFTWARE ENGINEERING PRINCIPLES 13

1. SOFTWARE ENGINEERING 13
2. SOFTWARE ARCHITECTURE 14
3. SUMMARY 15

CHAPTER 5 ENGINEERING WITH REUSE 16

1. HISTORY OF SOFTWARE REUSE 16
2. SOFTWARE REUSE 17
3. SOFTWARE REUSE ATTRIBUTES 17
4. REUSE COSTS 18
5. REUSE BENEFITS 18

6. SUMMARY 19

CHAPTER 6 ENGINEERING WITH PRODUCT LINES 20

1. SOFTWARE PRODUCT LINES 20
2. SOFTWARE PRODUCT LINE ACTIVITIES 20
3. PRODUCT LINE COSTS 24
4. PRODUCT LINE BENEFITS 25
5. VARIABILITY 25
6. SUMMARY 28

CHAPTER 7 PHILIPS MEDICAL SYSTEMS REUSE ACTIVITIES - 30 -

1. COMPONENTS & SERVICES - 30 -
2. INCREMENTAL REUSE - 33 -
3. MIP PRODUCT LINE - 34 -
4. ORGANIZATIONAL CHANGES - 35 -
5. OVERVIEW OF CONCERNS - 38 -
6. SUMMARY - 39 -

PART III FUNDING 40

CHAPTER 8 FUNDING FUNDAMENTALS 41

1. COSTS ANALYSIS 41
2. FUNDING APPROACHES 42
3. MANIPULATING BEHAVIOR 43
4. IDENTIFYING ISSUES 43
5. FUNDING MODELS 45

CHAPTER 9 CASE ON FUNDING MODELS 49

1. CONTEXT SKETCH 49
2. OVERVIEW OF MIP FUNDING MODELS 49
3. BRIEF OVERVIEW 51
4. SUMMARY 51

CHAPTER 10 ADJUSTABLE INCENTIVE-DRIVEN MODEL 52

1. RESEARCH QUESTION 52
2. ANALYZING ISSUES 52
3. ADJUSTABLE INCENTIVE-DRIVEN MODEL 54
4. ASSIGNING VALUES 57

5. SUMMARY 59

CHAPTER 11 APPLICABILITY OF THE AID MODEL 60

1. VALIDATION APPROACH 60
2. QUESTIONNAIRE CONCLUSION 62
3. MAP RESULTS ON AID FUNDING MODEL 63
4. PROS AND CONS OF AID MODEL 64
5. THE CONTRIBUTION 65
6. FUTURE WORK 65
7. CONCLUSION 66

PART IV DATA COLLECTION, METRICS AND TRACKING 67

CHAPTER 12 IDENTIFYING BUSINESS CONCERNS 68

1. CORE ASSET DEVELOPMENT 68
2. PRODUCT DEVELOPMENT 69
3. MANAGEMENT 69
4. OVERVIEW OF CONCERNS 70

CHAPTER 13 SOFTWARE PRODUCT LINE METRICS 71

1. DEFINING METRICS 71
2. OPERATIONAL STRATEGY 73
3. BUSINESS-LEVEL STRATEGY 76
4. CORPORATE-LEVEL STRATEGY 78
5. RELATIONS BETWEEN THE METRIC LEVELS 80
6. CONCLUSION 80

CHAPTER 14 VALIDATION OF THE METRICS FRAMEWORK 81

1. VALIDATION APPROACH 81
2. CORE ASSET DEVELOPMENT 81
3. PRODUCT DEVELOPMENT 83
4. MANAGEMENT 84
5. PROS AND CONS 84
6. THE CONTRIBUTION 85
7. FUTURE WORK 85
8. CONCLUSION 85

PART V APPENDICES 86

APPENDIX A REFERENCES 87

APPENDIX B QUESTIONNAIRE ON FUNDING MODELS 89

PART ONE
INTRODUCTION

This part is composed of three chapters. The first chapter provides background information concerning this thesis. Also, an overview of the thesis is presented. The second chapter is concerned with all relevant aspects of the research, among others, a presentation of the research questions and how these research questions are related to each other. The final chapter is used to introduce Philips Medical Systems. The purpose of explicating this organization is twofold. First, it is necessary to illustrate the problem statement and place it in a real-life context. Second, it is required that the context where the observations are made is specified, since these observations need not be applicable to other contexts. This obviously is of most importance with respect to validating the proposed solutions.

CHAPTER 1
INTRODUCTION

This chapter provides general information concerning the internship. It will be concluded with an overview of this thesis.

1. INTERNSHIP AND DELIVERABLES

This thesis forms one of the three deliverables of my internship at Philips Medical Systems. The internship is the obligatory graduation assignment for my study in Computing Science at the University of Groningen. More specific, it forms the graduation assignment for two masters in Computing Science, namely Software and Systems Engineering, and Business and Public Policy. Both masters are aimed at preparing the student to function within an organization. The focus of the former master is on several aspects of software development, e.g. project management and software architectures, and the focus of the latter is on innovative research within business organizations as well as within public policy organizations.

The other two deliverables are (1) an advisory report, which will be written in collaboration with my colleague Auke Schotanus, and (2) a presentation of my findings. In the advisory report, the results from my research will be combined with the results of my colleague's research. His research is concerned with several organizational aspects, e.g. organizational structure, roles and responsibilities. The advice thus takes both the organizational and the economic aspects into account, and it will specifically address the situation at Philips Medical Systems. The other deliverable, the presentation, only addresses the economic aspects.

The research has been conducted at Philips Medical Systems in Best, and the start date and end date are December 13th, 2004 and August 12th, 2005 respectively. Philips Medical Systems will be used to validate the solutions proposed in this thesis. Several organizational aspects of Philips Medical Systems, e.g. organizational structure and competitors, will be discussed briefly in chapter 3.

Much has been written on the subjects software reuse and software product lines and many different terms have been introduced. This thesis provides a unified terminology that is mainly based on the terminology proposed by the Carnegie Mellon Software Engineering Institute [11] [26].

2. DOCUMENT OVERVIEW

This thesis is composed of six parts, each part consisting of multiple chapters. The following figure specifies the structure of the thesis and, in specific, how these parts are related to the research approach.

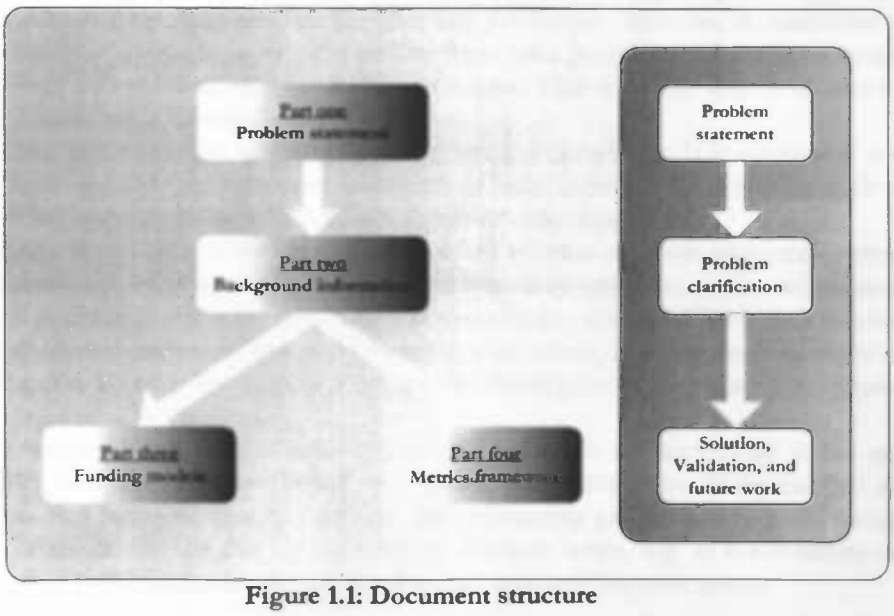


Figure 1.1: Document structure

The first part provides, besides this introductory chapter, the presentation of the problem statement. The problem statement consists of two main research questions, which are decomposed into multiple research questions. The first main research question is concerned with funding models for software reuse programs. The second main research question is concerned with measuring a software reuse program. In the final chapter of this part, Philips Medical Systems is introduced. Philips Medical Systems functions as a case study. The organization will be used for illustrating the problem statement in a real-life context, including a discussion on related issues and the implementation. It will also be used to validate the proposed solutions in the given context.

The second part is concerned with providing relevant background information. First, the notion of software architecture is introduced. Then, software reuse in general is introduced, including a brief history on software reuse. The subsequent chapter discusses software product lines, which can best be summed up as a planned software reuse program taking a top-down approach. The final chapter discusses the software reuse activities of Philips Medical Systems.

The third part deals with the first main research question, i.e. the main research question on funding a software reuse program. First, the theory on funding in general is presented. Then, the experiences of Philips Medical Systems regarding their funding models are presented. Based on the results of a questionnaire and on field observations, a funding model will be proposed in the subsequent chapter. Finally, this funding model will be validated through yet another questionnaire.

The subsequent part is concerned with the other main research question, i.e. the main research question on measuring a software reuse program. First, the business concerns, which need to be addressed by the software metrics, are explicated. The second chapter then presents the theory on software metrics and, based on the aforementioned business concerns, an overview of relevant metrics is proposed. The final chapter discusses the validation of the proposed set of metrics.

CHAPTER 2
RESEARCH SET-UP

This chapter presents the problem statement. First, the background will be discussed briefly and the scope of the research is defined accordingly. Then, the problem statement is explicated. The problem statement is composed of a brief overview of the overall problem and the related issues. The corresponding research objectives and research questions are presented successively. After that, the conceptual model, i.e. a view on how the research questions cohere, is discussed. The chapter will conclude with a discussion of the used research methodologies.

1. TRADITIONAL SOFTWARE DEVELOPMENT ISSUES

The general trend is that software products in general are growing larger and more complex [5], and the weakness of the traditional approach to software development is beginning to show [47]. Since software products are growing larger and more complex, the architectural and detailed design processes require more effort. Also, the implementation process will require more effort, since the productivity of software engineers will decrease rather than increase due to more complex software. This is, among other things, a consequence of the fact that the software products require more extensive testing. Thus, the first issue is that software is becoming more costly to develop and maintain. The next issue is that all of the aforementioned contribute to a lengthened time-to-market also.

Another important issue is the quality of software products. When a software product is larger and more complex, it is likely to contain more errors. This obviously has a negative impact on the quality of the software products. Consequently, more effort will be necessary for the maintenance of these products. The focus of business organizations shifts from innovative development to maintenance, leading to less competitive business organizations [5].

Summarized, the traditional approach to software development in time will lead to higher development and maintenance costs, lengthened time-to-market, reduced quality of software products, and less competitive business organizations. The notion of software reuse is presented as a means to deal with all of these issues [5] [11] [26] [27] [32] [40]. Software reuse can simply be summarized as the use of existing assets. These assets range from software architectures to reusable components to corresponding documentation. Reusable assets will further be referred to as core assets.

2. SCOPE

When institutionalizing software reuse in an organization, several aspects play a role. These aspects can be categorized in the following four dimensions: business, architecture, process, and organization [5] [28] [46]. These dimensions will be discussed briefly. The first dimension, business, is concerned with capital, funding and, of most importance, how to make profits from your products. For instance, institutionalizing a software reuse program requires a large up-front investment. This capital is tied up at least until the first product using the core assets is released.

Architecture, also referred to as engineering, is the second dimension. It is concerned with the technologies to make reuse possible, i.e. the technical means to build software. Examples of such technologies are object-oriented languages, development tools and product line principles.

The third dimension, process, is the encapsulation of all relevant software engineering processes. With respect to institutionalizing a software reuse program, the development processes need to incorporate the additional activities peculiar to software reuse and additional roles, responsibilities, and relationships need to be defined. Examples of such activities are integrating core assets, and defining and evolving the product line scope. Examples of roles and relationships are the developers of core assets, the product developers using these assets, and their relation.

The final dimension is organization. This dimension deals with software reuse at the organizational level and, in specific, how the aforementioned roles and responsibilities should be mapped to the organizational structures. For instance, multiple models for structuring an organization are proposed in the literature and the reuse developers can be allocated at different levels, e.g. at the business unit level. In addition, it needs to be prescribed who should develop and maintain the core assets.

This thesis focuses on two aspects within the business dimension, namely funding, and data collection, metrics, and tracking. Even though this thesis focuses mainly on the business dimension, the other dimensions will have to be addressed also. This is because a change in one dimension will affect all other dimensions as well [46]. The software reuse program is presumed to take a software product line approach, i.e. a planned software reuse program taking a top-down approach.

3. PROBLEM STATEMENT

As mentioned above, this thesis aims at proposing a funding model and a set of related metrics, i.e. a metrics framework. The former is concerned with allocating funds to cover for the costs of certain reuse-related activities. Multiple funding models are proposed in the literature, varying from flat tax models to usage-based models. Regarding software reuse, several business issues, which will have to be taken into account when institutionalizing a funding model, play a role. These business issues will be discussed in the following. First, when one is the first user of a core asset, one will have to deal with the corresponding childhood illnesses. Such childhood illnesses are for instance integration problems and unsatisfactory non-functional properties, e.g. performance and reliability. Since having to deal with such problems implies additional costs and longer lead times, developers often are reluctant to act as first users of a core asset. Second, the usage of the core assets needs to be stimulated. In a context where the use of core assets has not been proven beneficial, an incentive for using these assets seems mandatory. A related issue is that when multiple releases of the same core asset are in existence, the usage of the latest releases of that asset should be stimulated. This is necessary, because when multiple releases of a core asset are in use, it might be the case that the same programming error is reported for different releases. Consequently, the same error needs to be resolved in multiple releases, i.e. multiple releases of the same component need to be maintained concurrently. Finally, other business units should contribute to the pool of core assets also. This is necessary, since the capacity of the reuse developers forms the bottleneck with respect to the development of core assets. The issue with other business units contributing to the asset pool is that the development of core assets typically is more expensive than developing a product-specific asset. This is because a certain level of variability must be incorporated in the reusable asset. This obviously forms a barrier for business units, since they are strongly focused on shortening the time-to-market and reduce the costs.

The latter, the metrics framework, is concerned with measuring a software product line program, and in specific with identifying, analyzing, and quantifying the costs and benefits over time. In order to identify the benefits, the development, maintenance, and integration costs of core assets are often set out against the development and maintenance costs that would have been required to implement similar assets for just one particular product. The development costs of core assets typically are higher than the development costs of product-specific assets. The benefits, however, are gained when a core asset is used multiple times. This is due to the fact that the costs of integrating a core asset typically are lower than the costs of developing a similar asset for a specific product. In addition to the reduced development costs, the maintenance costs can be reduced dramatically also. When applying software reuse, the maintenance for all products making use of a core asset is done by maintaining just that core asset and updating all the products using that asset. Besides these savings on development and maintenance costs, other benefits can be gained also. Two different types of benefits can be gained, namely tactical engineering benefits and strategic business benefits. Besides the savings on the development and maintenance costs, the tactical engineering benefits are increased overall quality of the assets and a shortened time-to-market for new features or new products. The strategic business benefits need to be identified for each organization individually. There, however, have been identified certain typical strategic business benefits, e.g. a strengthened market position due to more robust products and shortened time-to-market, higher customer satisfaction due to shortened time-to-market, and more accurate planning and budgeting of product development. The metrics framework will address all the specific needs of an organization taking a software reuse approach to product development, including providing the necessities for performing and evaluating investments analyses.

3.1 RESEARCH OBJECTIVE

Both funding and data collection, metrics, and tracking have been identified as activities crucial to the success of the institutionalization of a software reuse program [11]. This thesis proposes both a funding model aiding in the success of the institutionalization of a software product line program and a framework for measuring a software product line program over time. The former is necessary, since the funding models currently proposed in the literature do not consider the aforementioned four business issues. It has been identified that rewards help stimulating the adoption of core assets in product releases, but it has not yet been specified what kind of rewards should be provided [8].

The need for the latter, the metrics framework, arose due to the fact that many software reuse metrics exist, but there is no complete, complementing set of metrics as of yet. The metrics framework will encompass, among other things, investment analysis, progress of adopting reuse practices, and a means to perform costs and benefits analyses.

The research objective thus is twofold. First, a funding model incorporating the four identified business issues will be proposed. Second, a metrics framework will be proposed, addressing several relevant aspects regarding the costs and benefits of a software reuse program. The contribution to the field of software engineering is thus providing additional support related to the institutionalization of a software reuse program. It should be kept in mind that the software reuse program is presumed to take a software product line approach.

3.2 RESEARCH QUESTIONS

Based upon the above, the following two main research questions are defined:

A. How can a funding model contribute to the success of a software product line program?
B. How can one determine if a software product line program is successful?

In order to provide an answer to the main research questions, several research questions need to be answered first. For the first main research question, the following research questions are relevant:

A.1 Which funding models are proposed in the literature?
A.2 What characteristics do those funding models exhibit?
A.3 How can behavior, with respect to a software product line program, be altered?
A.4 What types of rewards/punishments are most effective with respect to altering behavior?

For the other main research question, these research questions are relevant:

B.1 What types of costs and benefits typically are related to software product lines?
B.2 Which economic models are proposed in the literature?
B.3 How can costs and benefits related to a software product line program be quantified?

4. CONCEPTUAL MODEL

The first objective is to provide means to choose the most appropriate funding model. In order to do so, it needs to be clear what attributes a funding model should have in order to contribute to the success of a software reuse program. Both the literature study and the case study are used to develop and validate theories concerning the relevant issues.

The other objective is to provide a framework for measuring a software product line program. In order to do so, it needs to be clear what aspects of a software product line program should be analyzed in order to determine the success of that program. Thus, the relevant costs and benefits need to be analyzed and quantified.

The conceptual model provides a means of viewing how the research questions are correlated and how solving these questions leads to the desired results, i.e. how the research objectives can be realized. The conceptual model is presented in the figure below.

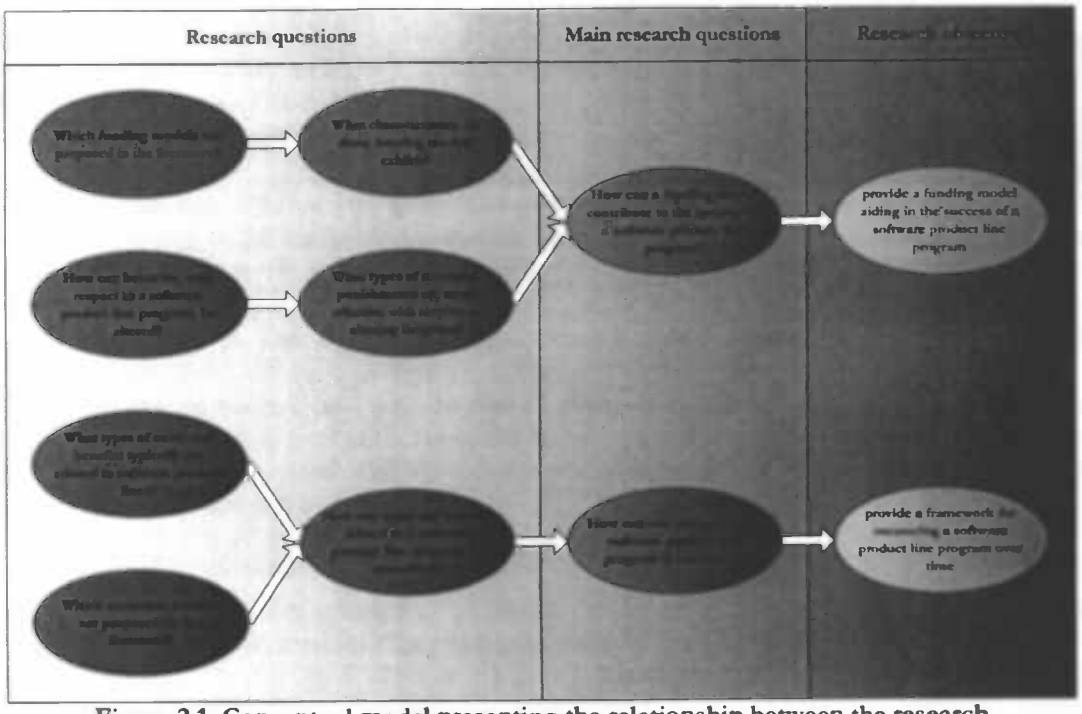


Figure 2.1: Conceptual model presenting the relationship between the research questions and the research objectives

5. METHODOLOGY

During the research, mainly two sources of information were used, i.e. the literature on software product lines and the employees of Philips Medical Systems. In addition, an analytical approach to derive solutions was used. All of the above will be discussed in the following. The discussion on the approaches to validation of the funding model and the metrics framework, however, are postponed to the final chapters of third and fourth part respectively.

5.1 THEORETICAL INPUT

Software reuse is the foundation on which the notion of software product lines is built. Therefore, many books and scientific papers on both software reuse in general and, in specific, on different aspects of software product lines were utilized. The focus mainly was on (1) how reuse activities can be funded and (2) how metrics can be applied to assess a software product line program. As mentioned earlier, the other dimensions of reuse, i.e. architecture, process and organization, needed to be addressed also, since any change in one dimension implies changes in the other dimensions as well.

From a theoretical point of view, employees from Philips Research, the University of Groningen and the Carnegie Mellon Software Engineering Institute were approached also.

5.2 PRACTICAL INPUT

During this research, different levels of management were asked to provide information. One of the success factors of a software reuse program is top management commitment [14] [35] [37] [48]. It therefore seemed imperative that higher management was engaged to provide information also. The employees provided information in the following three manners.

First, the employees of Philips Medical Systems were asked to participate in face-to-face interviews. These interviews provided much information on the subject context and how certain business units regarded the software product line initiative at Philips Medical Systems. Also, based on these interviews, different issues regarding the institutionalizing of a software product line initiative were identified.

A second manner in which they were asked to provide information was through the use of a questionnaire. The questionnaire only addressed the funding model aspects, but the business concerns required for deriving a metrics framework implicitly were provided also.

The third and final manner in which information was obtained from the employees was through their participation at the Philips Medical Systems Software Summit 2005. The participants were asked to give their opinion regarding multiple subjects related to their software product line initiative, among which the funding model.

5.3 ANALYTICAL APPROACH

In order to derive a funding model, an analytical approach was used. The main sources of input were the opinions of experts from the field, i.e. employees from both Philips Medical Systems and Philips Research, professors from the faculty of Management of the University of Groningen and researchers from the Carnegie Mellon Software Engineering Institute.

Field observations regarding business concerns and analyzing how software metrics relate to these observations provided the input for deriving the metrics framework.

CHAPTER 3
PHILIPS MEDICAL SYSTEMS

The medical market is a tough market to operate in. When one releases a product, these typically have to be maintained for approximately a decade. In addition, due to a rapidly growing medical knowledge base and correspondingly growing technical knowledge base, customers demand fast processing of new techniques in existing products. This market, specifically how Philips Medical Systems operates in it, is the topic of this chapter.

This chapter thus provides a brief introduction to Philips Medical Systems. First, general information concerning the organization is provided. Then, the forces from the market working on Philips Medical Systems are discussed briefly. After that, the organizational structure of Philips Medical Systems is discussed. Finally, a conclusion will be provided.

1. ORGANIZATIONAL INFORMATION

Philips Medical Systems, further referred to as Philips Medical Systems, is a key player in the field of medical appliances and related services. They maintain multiple product lines, among which:

- Imaging
- Ultrasound
- Defibrillation
- Cardiac monitoring
- Healthcare IT

An example of a typical system developed by Philips Medical Systems, referred to as a modality, is depicted by the figure below. Typically, such modalities need to be maintained for five to twelve years after the final shipment of a modality.

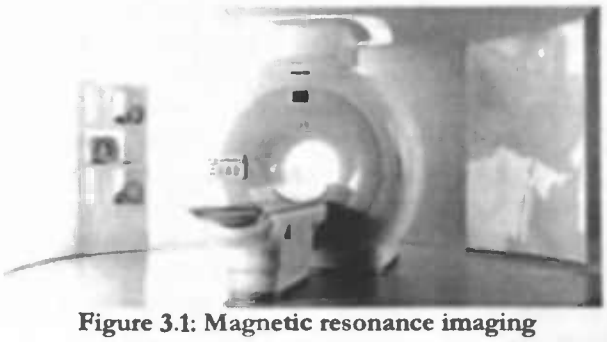


Figure 3.1: Magnetic resonance imaging

As becomes clear from Figure 3.1, these systems consist of large hardware components. The main advantages relative to the competitors, however, are to be gained with specialized software applications. Examples of such applications are two-dimensional and three-dimensional viewing, and multi-modality, i.e. combinations of multiple modalities in order to obtain data with different abstractions. The latter is for instance useful when combining the rough data used for automated detection of anomalies and accurate data for automated analysis of these anomalies. An example of such a multi-modality is the combination of a PET scanner with a CAT scanner, i.e. a position emission tomography scanner and a computerized axial tomography scanner respectively. The former scanner returns rough images of large parts of the body and the latter returns accurate information about specific locations, typically small parts of the body. One of the problems with developing such multi-modalities is that Philips Medical Systems is divided into multiple business groups, each of these groups is specialized only in their own systems, and these groups are used to operate autonomously. In order to develop multi-modalities, however, collaborations between different business groups obviously are a prerequisite.

2. MARKET DEMANDS

As mentioned above, Philips Medical Systems is one of the key players in the field of medical appliances and related services. In order to keep this position, they will have to at least react to the market. The customer demands from the medical market will be discussed briefly in the following.

First, the customers demand one look & feel for all the products. Since Philips Medical Systems has acquired multiple imaging companies in the near past, many different graphical user interfaces reside in the Philips Medical Systems products. The customer wishes that a person familiar with one Philips Medical Systems Modality also could easily start using another modality. A way to achieve this is through unifying the user interfaces of the product base, both for the hardware components and for the software components.

The second issue, which has been addressed earlier also, is the notion of interoperability. Customers demand interoperability between different modalities in order to obtain clinical information with a higher level of detail. The demand for interoperability, however, is not only restricted to interoperability between Philips Medical Systems modalities, the so-called multi-modalities. It also includes interoperability between products from different vendors. In order to achieve such a form of interoperability, industrial standards have been defined, among which DICOM.

The final issue mentioned here is that clients demand that adding features to currently released products should be realized faster. An example of such a feature is for instance 3D viewing.

The above describes market demands that Philips Medical Systems should react to. But, they do not only react to the market, they also set the market. This is for instance the case with extending the idea of the Philips Ambilight televisions. These televisions project the colors from the screen to the wall surrounding it, which creates a more intensive viewing experience. They have extended this idea for use with medical appliances as well. The idea is that a scanner is located in the center of a room and, while the patient is being scanned, soothing images of for instance a fish tank are projected on the walls of that room. This especially seems useful when treating young children.

3. ORGANIZATIONAL STRUCTURE

Philips Medical Systems takes a multi-level structure, which is depicted by the figure below. The CEO of Philips Medical Systems is Jouko Karvinen. He is supported by the Corporate Technology Office. The Corporate Technology Office can best be regarded as the technological conscience of Philips Medical Systems. It is concerned with policies with respect to technology and how these policies can be synchronized with the long-term roadmapping. In addition, it coordinates collaborations with external partners and manages the total research program. The Corporate Technology Office is the owner of the development processes at Philips Medical Systems. Furthermore, it provides the CEO with counsel regarding the development programs. This counsel leans on technological trends, needs, and the capabilities of Philips Medical Systems. Finally, it also looks at clinical trends and domains. The Corporate Technology Office links these trends and domains with the strategy of Philips Medical Systems, and it initiates the corresponding clinical research programs. Based on results gained from these research programs, the potential of new markets are taken into account with these counsels also. The Corporate Technology Office is a firm believer that software reuse in the form of software product lines should be adopted in order to maintain their market position.

As mentioned above, Philips Medical Systems consists of multiple business groups. These business groups are the following: X-Ray & MR, Cardiac & Monitoring Systems, Medical IT, Ultrasound, and Computed Tomography & Nuclear Medicine. These business groups, in turn, are divided into multiple business units. The business group X-Ray & MR consists of the following business units: CardioVascular X-Ray, General X-Ray, Generators, Tubes & Third Party Businesses, Components, and Magnetic Resonance. The business group Cardiac & Monitoring Systems consists of the following business units: Patient Monitoring, Defibrillators, and Stress Testing. Then, the business group Medical IT consists of the following business units: IT Systems and Components & Services. The business group Ultrasound is not divided into multiple business units. Finally, the business units Computed Tomography and Nuclear Medicine are not included in a business group. They both operate at the business unit level. The general structure of Philips Medical Systems is graphically represented in the figure below.

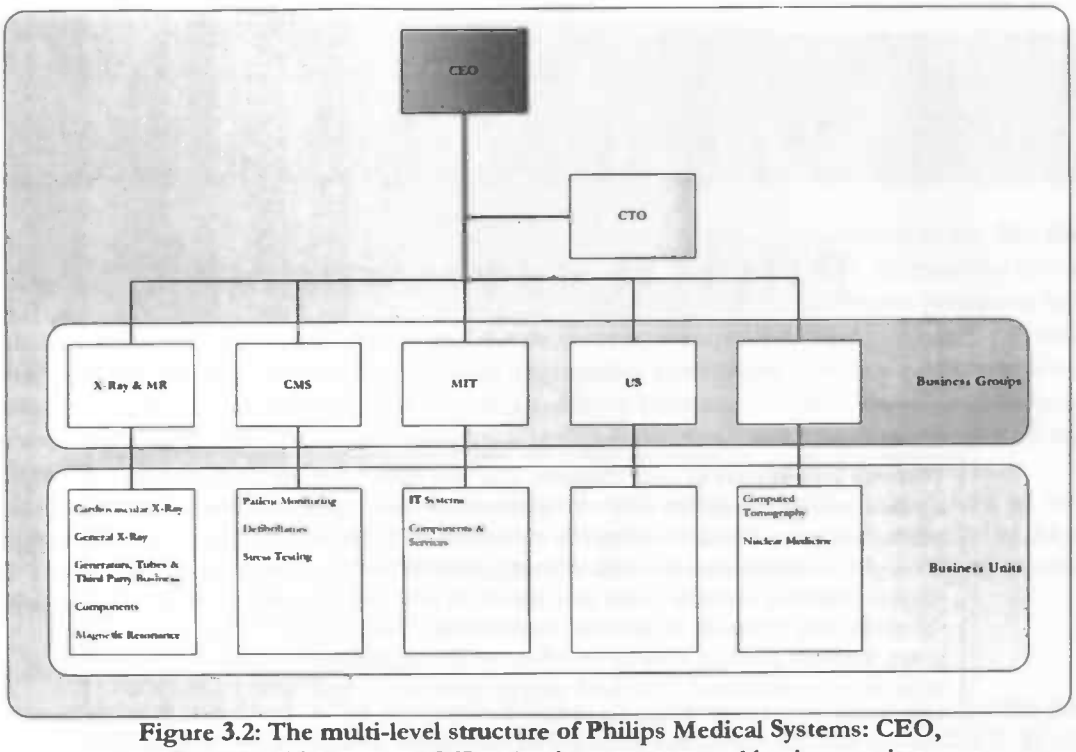


Figure 3.2: The multi-level structure of Philips Medical Systems: CEO, Corporate Technology Office, business groups and business units

The business unit Components & Services deserves special attention. Formally, Components & Services operates and reports to the same manager, as does the business unit IT Systems. They, however, are considered an independent, autonomously operating business unit, meaning that there is no direct relation between Components & Services and IT Systems. Components & Services is the business unit responsible for the development of core assets, also referred to as the reuse producer or the domain engineer. The other business units act as product developers, also referred to as reuse consumers or application engineers, meaning that they make use of these core assets in their product releases.

Having identified Components & Services as the sole core asset developer, the structure of Philips Medical Systems can now be identified as a domain-engineering-unit model [5]. The domain-engineering-unit model imposes a strict separation between core asset development and product development. One of the advantages of this model is that all support related to the core assets is taken care of by one business unit. This obviously simplifies the communication with respect to support between the business units. This, however, also leads to a disadvantage. Since one business unit is responsible for the development and maintenance of all the core assets, the domain engineering unit can easily become a bottleneck for the product developers. This is because they cannot serve all the support requests of the business units at once and a selection will have to be made.

Another advantage of a domain engineering unit is that it is less likely to include too much product-specific information, since they do not develop products themselves. This, however, also leads to another disadvantage. The domain engineering unit typically has little experience with the behavior of their core assets in product-contexts. Consequently, the quality attributes of their core assets may be poor in product contexts, while it performs well in their own testing environment. One way to deal with this is to engage the domain engineering unit to some extent in the product development processes.

4. SUMMARY

Philips Medical Systems is concerned with the development of multiple product lines. The market demands multi-modality systems and Philips Medical Systems will have to respond to that. In addition, the customers have expressed their desire for the following. First, all products from Philips Medical Systems should exhibit one look & feel. Second, all the products, even of different vendors, should be interoperable. Finally, the time-to-market for adding new features needs to be shortened. In order to achieve all of this, a collaboration between multiple business units must be realized. The first step in doing so is through the business unit Components & Services, which develops core assets to be used by all the other business units in product releases.

PART TWO
REUSING WITH PRODUCT LINES

This part is composed of four chapters. The first three chapters are used to introduce concepts from the field of Computing Science, which are used in the subsequent parts. The first chapter is concerned with discussing software engineering as a discipline and with defining software architecture. The second chapter discusses software reuse as a means to deal with the shortcomings of the traditional approach to software engineering. Then, the third chapter proposes software product lines as an effective means of implementing software reuse in an organization. The final chapter provides a discussion on the software reuse program of Philips Medical.

CHAPTER 4
SOFTWARE ENGINEERING PRINCIPLES

Software engineering as a discipline has matured much. This is for instance reflected in the fact that programming languages have evolved from machine languages providing just basic instructions to sophisticated object-oriented programming languages. More importantly, the development processes have become more structured and multiple models to traverse these processes have been proposed, e.g. the waterfall model and the iterative model. The software engineering community also has become aware of the importance of explicating the overall structure of a software product, i.e. the software architecture. How the development of software products is composed of multiple processes and what the role of the software architecture is will be described below.

This chapter first discusses the discipline of software engineering, including a discussion on the traditional approach to software development. The notion of software architecture is introduced in the second section. Three different purposes of software architecture are presented also. The chapter is concluded with a summary.

1. SOFTWARE ENGINEERING

Software engineering is defined as "*an engineering discipline which is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use*" [43]. In the context of Philips Medical Systems, a system comprises both hardware components and software components. However, this thesis only focuses on the software components of such a system. The composition of software components will further be referred to as a product. Within the discipline of software engineering, four fundamental processes can be identified, namely requirements engineering, software development, software validation, and software maintenance.

First, the process of requirements engineering takes place. The output of this process is a specification of the product to build. Two types of requirements have to be specified, i.e. functional requirements and quality requirements. Functional requirements describe the functionality that the product should offer and quality requirements prescribe the properties that the product should exhibit, e.g. a certain level of performance.

The second process is software development, which consists of software design and implementation. Two types of designs can be identified, namely architectural design and detailed design. The architectural design aims at developing a software architecture, which is the design of a product with a high level of abstraction. The detailed design is the translation step between the software architecture and the actual implementation and thus it is a design with a lower level of abstraction.

Software validation forms the third process of software engineering. It aims at testing every aspect of a product, varying from single components to the functionality offered by the product as a whole. Typically, the following four tests are used: unit testing, component testing, integration testing, and acceptance testing.

The final process is software maintenance, which is the process of changing a product after deployment. Maintenance comes in two flavors, namely debugging and adding functionality. The former is concerned with fixing programming errors and the latter with expanding the functionality offered by the product.

1.1 TRADITIONAL SOFTWARE ENGINEERING

Traditional software engineering exhibits three typical characteristics [5]. First, software engineering processes often are organized around projects. The goal of such a project is to deliver one product. All the effort put into developing that one product is likely to be lost, i.e. existing assets cannot be used in future products. Second, there is a strong focus on delivering the product on time. This is for instance reflected in the decision-making processes. To gain time, sub-optimal decisions often are made, for instance during the architectural design process. This relates to the third and final characteristic. Software engineers often only look for short-term sub-optimal solutions and the long-term vision, taking a maintenance and evolutionary perspective, is not considered.

Based on the above, several problems concerning traditional software engineering can be identified. Even though there is such a strong focus on time, only few projects actually succeed in delivering a product on time. In addition, since sub-optimal decisions are made, the overall quality suffers, which especially holds for the long-term quality. Consequently, the long-term maintenance costs form a substantial percentage of the total costs of the products. The final problem mentioned here is the fact that companies are becoming less competitive because their focus shifts from developing products to maintaining products.

2. SOFTWARE ARCHITECTURE

Software architecture is defined as follows "*the software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them*" [3]. So, the software architecture can be seen as the overall design of a product. As mentioned in the above definition, a software architecture consists of multiple software components. Such a component is defined as "*a unit of composition with explicitly specified provided, required and configuration interfaces and quality attributes*" [5]. These components provide certain functionality, which is specified in the provided interface. To provide certain functionality, it might be the case that a component needs to interact with another component, i.e. a component is dependent on the functionality provided by another component. If a component relies on functionality provided by some other component, this is specified in the required interface. The relations between the components, as specified in the software architecture, are explicated through the provided and required interfaces. The configuration interface is used to fine-tune a component to make it fulfill a specific requirement, i.e. alter its behavior. Altering behavior can for instance be useful when regarding software reuse, because components must then be used in different contexts.

A software architecture constrains all quality attributes of a product. Each software architecture imposes theoretical minimum and maximum values on the quality attributes of a product, which is, among others, reflected in the structure of the subject product. It should be noted that the values for the quality attributes are nothing more than indicative values. The actual values of the quality attributes are strongly influenced by the actual implementation. Architectural styles, architectural patterns and design patterns can be used to adapt the structure and the corresponding quality attributes.

2.1 ARCHITECTURAL PURPOSES

The explicit representation of the software architecture can be used for three purposes [5], which will be discussed briefly in the following. First, it can be used to evaluate the potential for the quality attributes offered by the software architecture. When a mismatch between the required and the actual quality attributes is discovered, this obviously will have to be resolved. As shown above, a software product is derived by first explicating the requirements, then deriving a software architecture and detailed design which both incorporate these requirements, and finally implementing the product. When a mismatch is discovered during the architectural design, no effort has been put in the detailed design or implementation of the product. But, if a mismatch is discovered during the implementation of the product, the architectural design, detailed design, and (parts of) the implementation will have to be revisited. Therefore, the software architecture provides means to validate that the quality attributes, as prescribed in the quality requirements, can be realized. More importantly, if these quality attributes cannot be realized, this can be discovered sooner, so no effort will be spent on the subsequent activities.

Second, it can be used to communicate with stakeholders early on in the development process. This obviously is possible due to the high level of abstraction that a software architecture has. Several views on a software architecture can be derived [31], each view providing some specific type of information. It can also be used to discuss trade-offs with customers, e.g. between contradicting requirements.

Third and last, a software architecture can function as a reference architecture for software product lines, also referred to as a product line architecture. This offers the potential to reuse existing assets in the development of new products. The following two chapters will discuss software reuse and software product lines in more detail.

3. SUMMARY

Software engineering is an engineering discipline concerned with all aspects of software production. Four fundamental processes can be identified, namely requirements engineering, software development, software validation, and software maintenance. Within the process of software development, architectural design takes an important position. A software architecture is the global design of a product and it has a high level of abstraction. The subject system is composed of multiple subsystems or components. Components typically have three different types of interfaces. First, components may depend on functionality offered by other components. If so, this is specified in the required interface. Second, components obviously provide certain functionality, which is specified in the provided interface. Finally, the behavior of components can be manipulated, with the purpose of incorporating variability. Also, three different purposes for software architectures have been identified, namely communication with stakeholders, providing a means for evaluating the potential for the quality attributes, and provide the basics for implementing software product lines.

CHAPTER 5
ENGINEERING WITH REUSE

In the previous chapter, it was stated that software engineering as a discipline has matured much. Even so, there still is much room for improvement. Traditionally, all effort spent on the development of a particular product is not utilized for the development of future products. However, since software is intangible, it provides potential to reuse assets practically free. Different benefits can be gained when one applies software reuse successfully ranging from realizing cost reductions to shorten the time-to-market of new products. Reusing software also brings forth additional costs peculiar to reusing assets. The potentials of software reuse have been acknowledged since the beginning of software engineering, but it turned out that reusing assets cannot be performed as straightforward as one initially assumed. This chapter introduces the notion of software reuse as an approach to software development. It constitutes the basics for software product lines, the topic of the next chapter.

This chapter thus addresses the notion of software reuse. The first section presents the history of software reuse. Then, it is specified what is considered to be software reuse. A definition is provided also. The subsequent section discusses multiple attributes of software reuse. Sections 4 and 5 discuss the costs and benefits of software reuse respectively. Finally, the chapter is concluded with a summary.

1. HISTORY OF SOFTWARE REUSE

The first approach to software reuse was to make use of componentization. It was proposed in the late 1960s at the NATO conference on software engineering in Garmish. The first article on componentization was written by Douglas McIlroy and it is called *Mass Produced Software Components*. The idea simply was to develop components that provide certain functionality. Multiple products requiring that functionality can then use those components, which would lead to the following benefits. First, the development costs and maintenance costs of the reusable components can be spread across all the products using these components. This leads to both lower development costs and lower maintenance costs. Then, since the components are used in multiple contexts, the quality of the components is expected to increase also. This is because the components are used more extensively and more programming errors are likely to surface. Finally, the time-to-market for new products can be reduced, since parts of the subject products are obtained through using components.

Examples of successfully reused assets are, among other things, operating systems and compilers. The former, for instance, can be installed on any PC configuration. The installation procedure uses the configuration of the PC as input to tailor the operating system according to the requirements automatically. The latter, the compilers, are developed in such a manner that they comprise roughly three separate parts, namely a front-end, a semantic representation and a back-end [22]. The front-end translates some input language to an intermediate form, i.e. the semantic representation. The back-end then takes the semantic representation as input and translates it to the target language, i.e. the language of the target machine. By using such a semantic representation, the front-end and back-end of the compiler can be unaware of each other. So, when a new programming language is defined, only a new front-end needs to be constructed. The front-end should then translate the input language to the semantic representation. All the back-ends can then be reused for all the target machines. In addition, if a new target machine is introduced, only a new back-end needs to be developed. That back-end should then translate the semantic representation to the target language. This way, all the front-ends can be reused.

The two cases of software reuse mentioned above are not always identified as being software reuse. This is due to the fact that the functionality offered by these types are embedded in the infrastructure. This, however, is related to the typical lifecycle of reusable components. Components initially are part of the application code, i.e. developed for a specific product. A well-known example is Microsoft's web browser Internet Explorer. Internet Explorer initially was an application, which could be installed on a Microsoft Windows operating system. Over time, Internet Explorer was adopted by the underlying operating system; it became a part of the infrastructure.

Despite the many promised benefits, software reuse programs have a history of being applied unsuccessfully in software development organizations. Such organizations typically develop products in a certain

domain. At the level of software architecture, much progress regarding software reuse has been achieved. Within the software engineering community, multiple smaller communities regarding software architecture can be identified. These smaller communities for instance focus on how one can evaluate the potential of a specific quality attribute and which architectures can be applied best for what domain. Reference architectures, e.g. the blackboard architecture, exhibit certain attributes that seem most useful in certain domains. This can thus be seen as the reuse of architectural knowledge. Reusing software components, however, has not yet reached such a level of maturity. The reuse of software components typically does not cross the boundaries of an organization, with the exception of COTS components. Below, a typology of software reuse is presented.

2. SOFTWARE REUSE

To put it simply, software reuse is concerned with using assets, which have been used in earlier products. However, this is just one aspect of software reuse. It encompasses much more. Typically, two ways of categorizing software reuse are used [5] [28]. The first categorization is opportunistic versus planned reuse. With opportunistic reuse, the developers need to search for assets themselves. In addition, these assets are not designed specifically for reuse. On the other hand, planned reuse prescribes that the assets are developed for reuse and are evolved accordingly. Therefore, the design for, and development and evolution of core assets are other aspects of software reuse.

The second categorization of software reuse is bottom-up versus top-down. In the former approach, the core assets are stored in a repository and the engineers have to search for suitable assets themselves. The latter approach dictates the explicit design of a higher-level structure, for instance a product line architecture, and the core assets are designed as parts that fit into this higher-level structure. Thus, yet another aspect of software reuse is designing and evolving a higher-level structure, which can be used to derive products using core assets. It should be noted that this higher-level structure typically is regarded as a core asset itself.

The following definition for software reuse, which is derived from the above, will be used in this thesis. Software reuse is a process, which encompasses the design and evolution of a higher-level structure, the design, development and evolution of core assets that can be used within that higher-level structure and the development of products whose structure is a derivation of that higher-level structure and is composed of core assets and product-specific assets.

In order for a software reuse program to be successful, the program must be planned and take a top-down approach [5] [28]. The definition above reflects these properties also. An example of such a software reuse program is a software product line, which is the topic of the following chapter.

3. SOFTWARE REUSE ATTRIBUTES

This section addresses three attributes of software reuse. First, a typology is provided, i.e. the three levels of software reuse.

3.1 REUSE TYPOLOGY

Three levels of software reuse can be identified [5], namely:

- Reuse of assets over subsequent versions
- Reuse of assets over subsequent versions and various products
- Reuse of assets over subsequent versions, various products, and various organizations

In this document, the focus is on the second level of reuse, i.e. reusing assets over subsequent versions and various products, with the emphasis on the latter.

3.2 REUSE DEVELOPMENT

When regarding software reuse, two types of development processes can be identified [5] [11] [28] [32] [40]. The first type, core asset development, also referred to as domain engineering, is concerned with the development and evolution of the core assets. The second type, referred to as product development or application engineering, is concerned with the development of actual products, using core assets like reference architectures and reusable components.

Core asset development and product development intertwine over time. Often, assets from product development are mined for the development of a core asset variant. Core assets adopt more and more

features from the product-specific assets. This typically is the case when certain business units have achieved a high level of expertise regarding a certain aspect of a domain, which proves of great value for core asset development. The following chapter will elaborate on these two development processes in a software product line context.

3.3 BLACK-BOX VERSUS WHITE-BOX

Two primary approaches to software reuse can be identified [32] [40]. The first approach is white-box reuse. Many organizations utilize this type of software reuse. White-box reuse simply is copying and modifying existing software. This only results in a one-time benefit, which does not extend beyond the software development process. Software validation and software maintenance will both have to be revisited entirely for every single product.

Black-box reuse, which is the focus of this document, is concerned with using core assets without modification at the level of source code. All the behavior modifications are realized through so-called variation points, which will be discussed in the subsequent chapter. Black-box reuse exhibits the possibilities to reduce the development and maintenance costs, shorten the time-to-market, and increase the level of quality of the software.

4. REUSE COSTS

Software reuse programs offer great potential on reducing development costs, reducing maintenance costs, shortening the time-to-market, and increase the overall product quality. This, however, does not come free. Regarding the institutionalization of a software reuse program, one needs to consider the following four cost types:

- C_{org}, the costs that an organization makes to adopt the software reuse practices
- C_{cab}, the development and maintenance costs of the core asset base
- C_{unique}, the development costs of the product-specific assets
- C_{reuse}, the costs of integrating a core asset in a product

The organizational transition costs, C_{org}, are the encapsulation of all costs related to adopting a software reuse approach to product development. Adopting such an approach typically requires a reorganization at different levels. This is due to the need for additional roles, responsibilities, and relations. C_{org} encompasses, among other things, the costs related to train the developers, restructuring the organization and improve the development processes.

The costs related to the core asset base, C_{cab}, are composed of multiple factors. Assuming that the software reuse program is planned and takes a top-down approach, the first step is to identify all the commonalities and variations between the different products. Then, a higher-level structure, which is applicable to all the products, needs to be designed. After having defined the software architecture, the different core assets need to be developed. Finally, the higher-level structure and the core assets can be used to derive products. The described process, however, is an iterative process. The higher-level structure and the corresponding core assets need to be maintained and evolved according to the market demands.

Since core assets only provide common functionality, the product-specific functionality needs to be provided in another manner. This is done by developing product-specific assets. The costs of developing and maintaining these product-specific assets are referred to as C_{unique}.

Reusing a core asset does not come free. Because core assets need to be used in multiple products, they also should provide some level of variability. Tailoring a core asset to make it adhere to the product-specific requirements thus is required. In addition, the asset needs to be integrated and tested in the context of that particular product. The corresponding costs are referred to as C_{reuse}.

5. REUSE BENEFITS

As mentioned above, software reuse promises lowering the development costs and the maintenance costs, shorten the time-to-market and increase the overall quality of software products. This section takes a closer look at these promises, more specifically, how these promises can be fulfilled.

First, regard the reduction of the development costs. Reusing core assets exhibits economies of scale, i.e. the notion that the average costs of core assets decline if they are used in more products [4]. This is because software is intangible and reproduction of software practically is free. Regarding the use of core

assets, the costs of developing the core assets, C_{cab}, and the costs of integrating the core assets, C_{reuse}, need to be taken into account [10]. In a typical software reuse context, the following rules of thumbs exist [28] [40]. First, developing a core asset costs an additional 50 percent of effort relative to the costs of developing a product-specific asset with similar functionality. Second, integrating a core asset in a product costs 25 percent of the effort of what it would have cost when a product-specific asset with similar functionality would have been developed from scratch. One can state that using a core asset in a product context results in a 75 percent savings on the development costs for a particular product. Given these numbers, it can be calculated how many times a core asset needs to be used in different products in order to break-even, i.e. obtain a net profit figure of zero. To do so, the following equation needs to be resolved, where n stands for the number of products using the core asset at hand.

$$C_{cab} + \sum_{i=1}^{n}(C_{reuse}(i)) \leq \sum_{i=1}^{n}(1 - C_{reuse}(i))$$

Equation 5.1: equation to determine the break-even point for core assets

Filling in the values from above, it turns out that a core asset should be used at least three times in multiple products in order to break-even. It should be noted that the net present value, i.e. the effect of time on investments, is not taken into account in the above equation and that the core asset base consists of solely one asset.

Second, the maintenance of core assets replaces the maintenance for all the product-specific assets, which have been substituted by these core assets. The maintenance of the core assets is performed centrally. This way, the maintenance costs for just the core assets can be spread across all products using these core assets. Consequently, the product developers do not have to take the maintenance for those assets for their account, besides the reporting of errors and applying the corresponding patches. This obviously results in lower maintenance costs. Research has shown that up to 80 percent of the total product costs are due to maintenance [5]. A reduction in maintenance costs thus offers great potential in reducing the total costs of a product dramatically.

Third, the overall quality is expected to increase. If a core asset is improved, this means that all the products that use this core asset, can be improved also. Since the core assets are used in multiple contexts, more programming errors are expected to surface.

Finally, the integration of a core asset typically requires less time than developing a similar product-specific asset from scratch. Therefore, using core assets should result in a shortened time-to-market. However, it should be noted that the development of a core asset typically takes more time as does the development of a product-specific asset. As a consequence, the initial product using that core asset will experience a lengthened time-to-market rather that a shortened time-to-market.

6. SUMMARY

Software reuse is a process, which encompasses the design and evolution of a higher-level structure, the design, development, and evolution of core assets to be used within that higher-level structure and the development of products whose structure is a derivation of that higher-level structure and is composed of core assets and product-specific assets. The approach to software reuse with the highest potential is planned reuse of core assets without modification and taking a top-down approach. Core assets must provide variability in order to be usable in differing contexts. Institutionalizing a software reuse program typically brings along the following four types of costs: C_{org}, C_{cab}, C_{unique}, and C_{reuse}. Finally, software reuse promises the following benefits: a reduction in the development costs, a reduction in maintenance costs, a shortened time-to-market for new products, and a higher overall quality of software products.

CHAPTER 6
ENGINEERING WITH PRODUCT LINES

As mentioned, the discipline of software engineering has matured much since its beginning. Software reuse, conversely, has not yet reached a similar degree of maturity. Even though for instance architectural knowledge is shared across organizational boundaries, organizations still struggle to develop reusable components that are only meant for use within the boundaries of that organization, let alone develop components, which can be reused by multiple organizations. The maturity of the software engineering discipline is for instance reflected in the degree of structuring the development processes. A similar maturing of software reuse has occurred, namely the introduction of software product lines. Software product lines provide an implementation of a software reuse program that is planned, takes a top-down approach, and prescribes a structured approach to product derivation.

This chapter elaborates on the notion of software product lines. The first section presents the definition and the subsequent section describes the three fundamental activities concerned with software product line development, i.e. core asset development, product development, and management. The third and fourth sections discuss the costs and benefits of software reuse in a software product line context respectively. The fifth section presents a diversity of variation mechanisms. The chapter will conclude with a summary.

1. SOFTWARE PRODUCT LINES

As mentioned in the previous chapter, a software product line is an implementation of a planned software reuse programs taking a top-down approach. Both being planned and taking a top-down approach are key success factors for any software reuse program. When using software product lines, one is planning, enabling, and enforcing software reuse. In [11], software product lines are defined as "*a set of software-intensive systems sharing a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way*". The key insight regarding software product lines is to exploit the commonalities between different products in a structured manner. Exploiting commonalities is realized through the use of core assets. Such core assets comprise, among others, a product line architecture, reusable components and corresponding documentation. As mentioned earlier, software reuse offers economies of scale. However, besides economies of scale, software product lines offer economies of scope also. Economies of scope exist if savings are achieved when one increases the variety of goods and services that one produces [4]. In fact, one of the most important processes regarding software product lines is scoping, i.e. defining the product line scope [11]. The product line scope determines which products can be derived from the software product line, i.e. it determines the size and the boundaries of the set in the definition of software product lines above. Therefore, it can be stated that the product line scope determines the leverage of the economies of scope. When the scope is defined too large, the needed variability exceeds the variability that the core assets can offer. In addition, when the scope is defined too small, the core assets are too product-specific, restricting future growth. In addition to this, the leverage offered by the economies of scope would then be minimal.

2. SOFTWARE PRODUCT LINE ACTIVITIES

Regarding software product lines, three fundamental activities can be identified, namely core asset development, product development, and management. As mentioned in the previous chapter, core asset development is concerned with the development and evolution of the core asset base. Product development, inversely, is concerned with using these products to derive products. Finally, management is involved with the orchestration of the aforementioned two activities. The following three subsections present these three activities in more detail. The fourth and final subsection will discuss the relations between these processes.

2.1 CORE ASSET DEVELOPMENT

Core asset development, also referred to as domain engineering, is concerned with the establishment of a production capability for the different products in the software product line. The figure below illustrates the process of core asset development, including the inputs and the outputs.

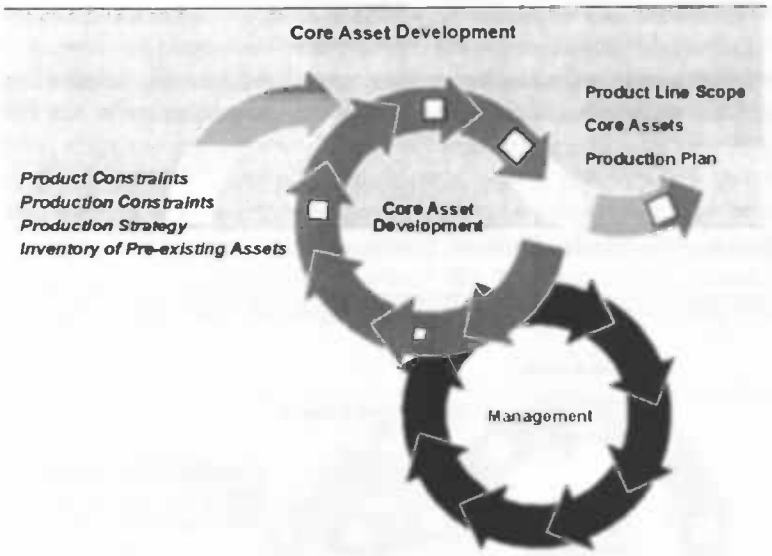


Figure 6.1: Core asset development requires four inputs, product constraints, production constraints, production strategy, and the inventory of preexisting assets, and it returns the product line scope, core assets, and production plans as outputs [26]

Examples of core assets are the product line architecture, reusable components, documentation etc. The product line architecture is defined as "*the common architecture for a set of related products or systems developed by an organization*" [5] and it forms the foundation for all the products in the product line scope. All products derived from the software product line use a derivation of the product line architecture as its software architecture. The use of a derivation of a product line architecture is necessary to be able to predict to some extent how the reusable components behave, since the context is somewhat familiar. In addition, the software architecture dictates the proper level of abstraction for the components. This is due to the fact that the components have been designed for use within a derivation of that architecture.

As shown in Figure 6.1, the core asset development activities result in three outputs, namely (1) the product line scope, (2) core assets, and (3) production plans. These outputs will be discussed below.

(1) As mentioned, the product line scope determines which products can be derived from the software product line. When a product is included in the product line scope, the core assets should support the common parts of the features, which that product must offer, including the necessary variability. Also, a derivation of the product line architecture should be used as the software architecture of the subject product. Due to, among others, changing market conditions and variation in the organization's roadmap, the product line scope evolves.

(2) The core assets are the basis for the production of products. Core assets vary from the product line architecture to documentation, from reusable components to commercial off-the-shelf components. When a derivation of the product line architecture is used, the behavior of reusable components can be predicted more accurately. Such a derivation is obtained by using variation points in the product line architecture, i.e. locations at which variation will occur [28]. By using core assets, it is possible to standardize the product development process. This is because, according to the definition of software product lines, all core assets are to be used in a prescribed way. Every core asset should therefore have a standardized process attached to it. Such an attached process specifies how the core asset can be used in a particular product context. Besides these attached processes, defining how the core asset base has to evolve is an important aspect of creating the core asset base also.

(3) As mentioned, core assets should have standardized processes attached to them, specifying how these core assets can be used within the development of products. For the development of a product, multiple core assets can be used. Consequently, multiple attached processes will have to be consulted also. A production plan for a specific product is a collection of all the relevant attached processes necessary for the development of that product. It specifies step by step how products are produced, and in particular, which core assets are to be used in what manner at what time in the product development process. It can best be regarded as a programmer's guide for all the relevant core assets, including how these core assets can be tailored for use in the specific product. Without production plans, product developers probably do not know the relations between the different core assets and they do not know how the variation points can be used optimally.

The figure below illustrates the relation between core assets, the attached processes, and production plans.

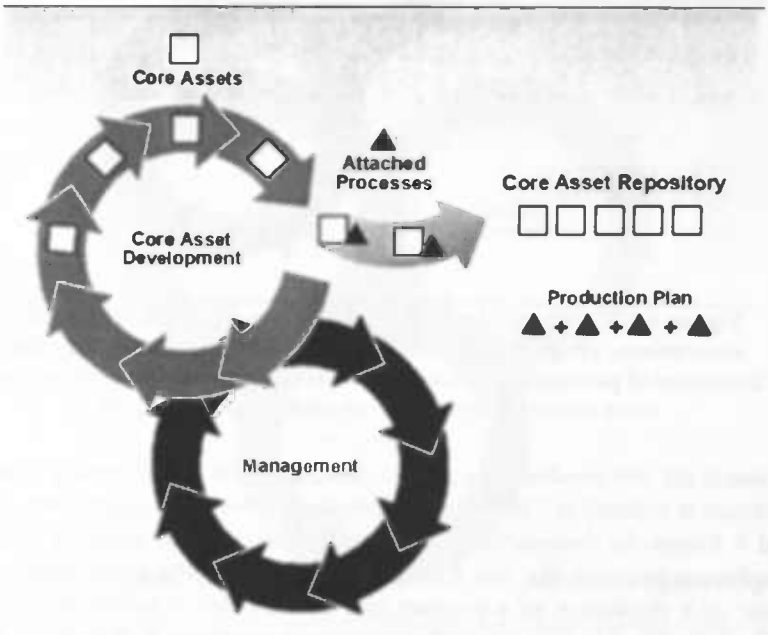


Figure 6.2: The relation between core assets, attached processes, and production plans [26]

In order to achieve all of the above outputs, multiple inputs are necessary. These inputs will be discussed briefly here. First, it should be specified what the commonalities and the variations between the products are, taking both a present perspective and a future perspective. The latter is necessary to also take the evolution of the software product line into account. Second, architectural styles, patterns, and frameworks are imperative for defining the proper product line architecture. When defining the product line architecture, one should take into account both functional requirements and quality requirements. As mentioned in chapter 4, a software architecture imposes theoretical minimum and maximum values on the quality attributes of a given product. Third, the production constraints need to be regarded. Examples of production constraints are time-to-market demands and legacy-system constraints. The overall approach for realization of the core assets is referred to as the production strategy, which is the fourth input. Examples of relevant issues are what funding model to choose or whether or not currently existing products should be mined for core assets. Finally, the inventory of preexisting assets can be used as a basis for core asset development. Existing products can for instance be mined for core assets or the software architectures of such products can be used as a basis for the product line architecture.

2.2 PRODUCT DEVELOPMENT

Product development, also referred to as application engineering, is the process of deriving products from the software product line using core assets and product-specific assets. As mentioned, core assets need to be tailored to provide the necessary functionality for a specific product and this can be realized through using variation points. A derivation of the product line architecture is used as the software architecture for the subject product. In addition, reusable components are used to implement common features. Functionality, which has not been adopted by the core asset base, needs to be developed for each product separately. Such assets will be referred to as product-specific assets, i.e. assets only applicable for one particular product.

The product development activity requires four inputs, namely the product-specific requirements, the product line scope, the relevant core assets and the relevant production plan. The latter three relate to the three outputs of core asset development and were already treated in the previous subsection. The product-specific requirements are all the requirements to which the product should adhere. The output of this activity comprises only products, ready for shipment. The figure below illustrates the activity of product development.

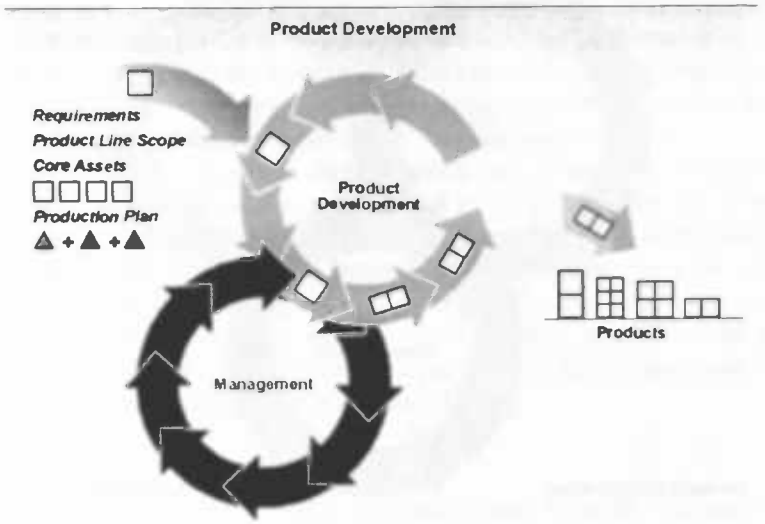


Figure 6.3: Product development requires four inputs, requirements, product line scope, core assets and production plans, and returns actual products as output [26]

2.3 MANAGEMENT

Two levels of management can be identified. i.e. technical management and organizational management. In order for a software product line to be successful, both levels of management should be strongly committed to the software product line effort [11].

Technical management is concerned with the operational activities in the core asset development and product development activities. Examples of technical-management activities are data collection, metrics and tracking, scoping, and configuration management.

The definition of organizational management is "*the authority that is responsible for the ultimate success or failure of the product line effort*" [11]. It is concerned with all organizational aspects regarding software product lines. Examples of organizational-management activities are launching and institutionalizing a software product line, determining a funding model, and defining the organizational structure. In addition to this, organizational management orchestrates core asset development, product development, and technical-management activities.

2.4 RELATIONS BETWEEN THE PROCESSES

All three main activities, core asset development, product development, and management, influence each other. For instance, core asset development and product development can occur in either order. The first scenario is that core assets are used to derive products. This is the standard approach of product development in a software product line context. As mentioned in chapter 5, the typical lifecycle of core assets is that they initially are part of the application code, i.e. product-specific code. Due to e.g. a changing market, that part of the functionality can be identified as a potential core asset. The product containing that code can then be mined for the core asset. An alternative obviously is to develop that core asset from scratch. In addition, new products and markets give rise for the need to evolve the product line scope. The product line scope is an output of the core asset development, but it will be influenced strongly by the strategic roadmap of the management, i.e. defining the direction for an organizations future growth. The relations between the three fundamental processes can be drawn as follows.

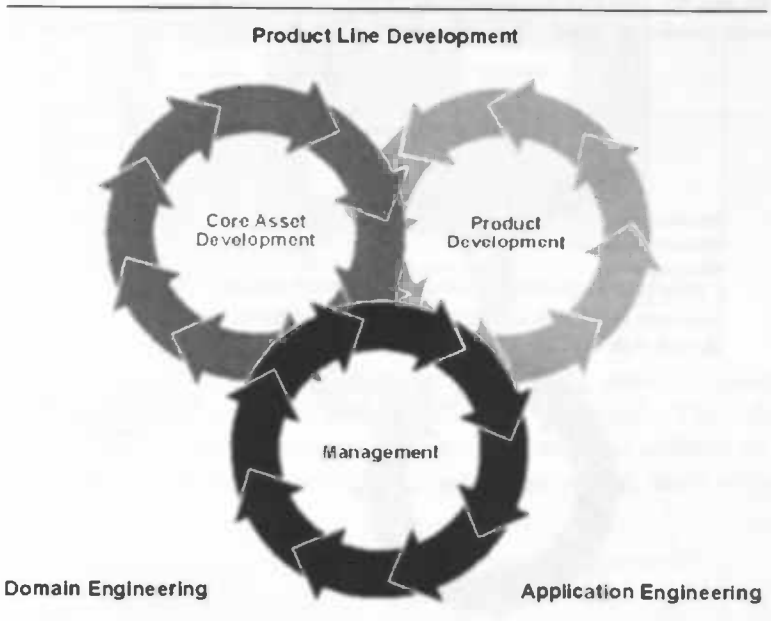


Figure 6.4: The relations between the three fundamental activities regarding software product line engineering [26]

3. PRODUCT LINE COSTS

Software product lines practices has the same four cost types as software reuse practices do:

- C_{org}, the costs that an organization makes to adopt the software reuse practices
- C_{cab}, the development and maintenance costs of the core asset base
- C_{unique}, the development costs of the product-specific assets
- C_{reuse}, the costs of integrating a core asset in a product

There is a difference, however. The previous chapter prescribed a general values for integrating a core asset relative to developing a similar product-specific asset. The provided values was 25 percent, meaning that integrating costs 75 percent less effort as it takes to develop a similar product-specific asset. In a software product line context, a different value holds in general. In [10], five percent was suggested. Reduced

4. PRODUCT LINE BENEFITS

The main benefits gained from adopting a software product line approach [10] [11] [12] [27] [39] can be categorized in two classes, i.e. tactical engineering benefits and strategic business benefits. The former consists of the typical software reuse benefits: the time-to-market of new products can be reduced, the overall quality can be increased, the development costs can be reduced, and the maintenance costs can be reduced. An additional benefit of software product lines is that, due to the structured manner of product development, an increase in the total number of benefits can be managed more effectively.

The latter, the strategic business benefits are derived from the tactical engineering benefits. The shortened time-to-market strengthens ones market position, since one is able to respond to changing market conditions faster. Another benefit is that the profit margins are higher. This obviously is a result of the reduced development and maintenance costs. Because the overall quality of products is higher, for instance reflected by a smaller error density, the company reputation is strengthened. Since the total number of products can be increased and managed more effectively, both the scalability of the business models in terms of products and markets and, the agility to enter new markets increase. Due to, among other things, the structured manner of working, product deployments are less risky. The core assets have been used in multiple contexts and have therefore become more robust and predictable with respect to their behavior. The most uncertain aspects of deriving new products obviously reside in the product-specific assets. Related benefits are for instance the leverage of R&D investments over multiple products, a modular product line architecture facilitates the process of refreshing the software product line, and the domain-expertise can be leveraged across all the products within the product line scope.

Finally, intangible benefits can be gained also. Research has shown that the staff turnover in a software product line context is lower than it is in a traditional software development context. This is, among others, due to the fact that developers have reported satisfaction with the asset-based approach and that they can focus on interesting, more challenging aspects. This is possible, since the standard components of products are captured by the core assets and do not have to be developed for each specific product. Due to the fact that the use of core assets leads to a more accurate analysis of the time-to-market of a product and, therefore, less problems with delivering a product on time, customers have expressed their satisfaction regarding the software product line initiatives also.

5. VARIABILITY

As mentioned earlier, variation points are used to provide certain variability for core assets. These variation points offer possibilities to alter certain parts of the behavior of core assets, so that these assets are applicable to multiple contexts. This section presents a diversity of variability mechanisms [5] [11] [21] [28] [32] [44], which can be used to implement the variation points.

5.1 FAÇADES

In [28], the following definition is provided "*A façade is a packaged subset of components, or references to components, selected from a component system. Each façade provides public access to only those parts of the component system that have been chosen to be available for reuse.*". A façade thus provides a unified interface to a set of interfaces in a subsystem and it can best be regarded as a higher-level interface. The following illustrates the motivation for using façades. Structuring a system into subsystems reduces the complexity of that system. A subsystem typically consists of multiple classes, with each class providing its own interface. When one wants to make use of such a subsystem, all the individual interfaces of the classes will have to be used, as is illustrated by the figure below. The external classes represent classes not included in the subsystem. The black lines indicate which classes communicate with each other through method calls. The dark grey box represents the subsystem.

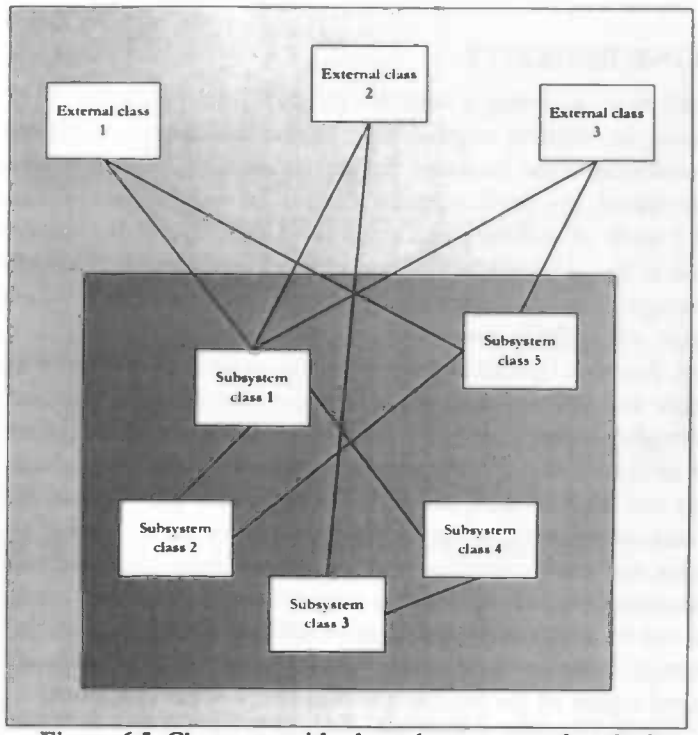


Figure 6.5: Classes outside the subsystem need to deal with interfaces at the level of classes

However, by introducing a façade, the communication between the external classes and the internal classes can be realized through just one higher-level interface, i.e. the façade. The following figure shows the same communication pattern as does the previous figure, but with the addition of a façade. It is clear that the internal structure is not relevant anymore for the external classes, since all public access is provided via the façade.

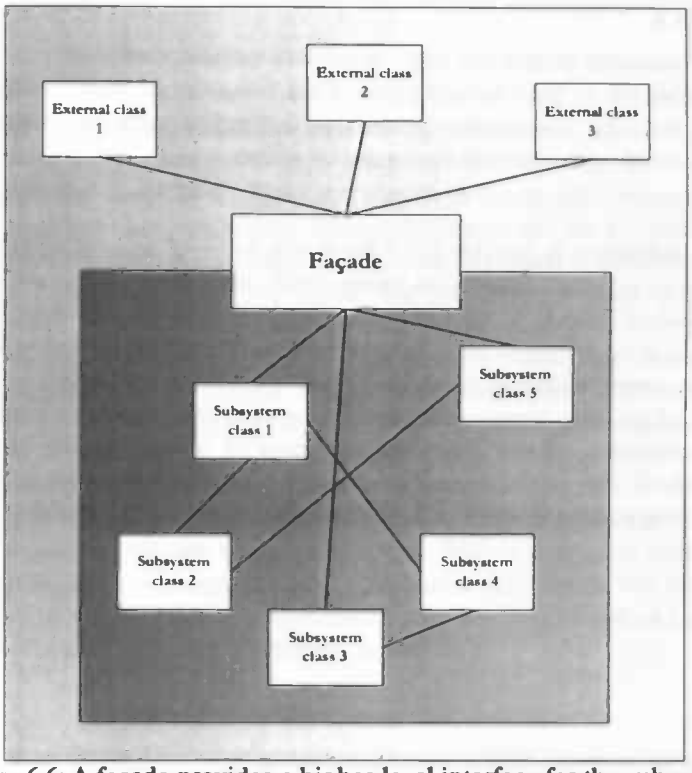


Figure 6.6: A façade provides a higher-level interface for the subsystem and irrelevant parts of interfaces are kept hidden

The following illustrates another type of usage of façades. Subsystems sometimes provide multiple implementations of the same algorithm. This can for instance be the case with sorting algorithms. When a component is used in a real-time context, sorting needs to be carried out as fast as possible. The drawback of fast sorting algorithms is the complexity of these sorting algorithms, e.g. resulting in high memory usage. When using that same component in a strict embedded context, memory usage may form a restriction rather than efficiency. Then, the complex algorithms need to be replaced by less memory-consuming algorithms, which often are slower. To offer both sorting techniques as a means of variability, façades can be used. The subsystem then requires two façades, namely one for each sorting algorithm. This is depicted by the figure below.

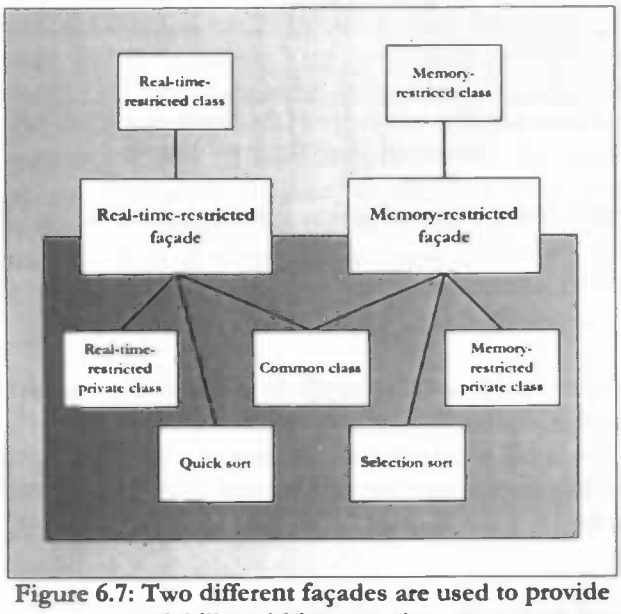


Figure 6.7: Two different façades are used to provide variability within one subsystem

5.2 INHERITANCE

The second method for providing variability is inheritance. The most well known application of inheritance is object-oriented software development. With object-oriented software development, objects are represented as classes. With respect to inheritance, two types of classes can be identified, namely superclasses and subclasses. A subclass inherits all the properties of its superclass. Such properties are its state and its behavior, i.e. all variable declarations and all method declarations respectively. The strength of inheritance is that subclasses not only exhibit the state and the behavior of its superclass, but they can also introduce additional variable declarations and method declarations. If necessary, the subclass can even override one of the inherited methods, i.e. declare its own implementation of that algorithm. The following figure illustrates the usage of inheritance.

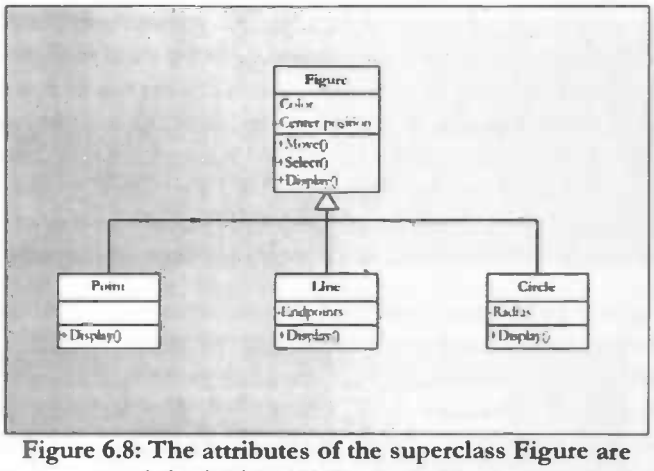


Figure 6.8: The attributes of the superclass Figure are inherited by all three subclasses

The class `Figure` is the superclass and it exhibits the following features:

- State; `Color`, `Center position`
- Behavior; `Move()`, `Select()`, `Display()`

All three subclasses override the method `Display()`. In addition, the subclasses `Line` and `Circle` both introduce an additional variable, `Endpoints` and `Radius` respectively.

The principle above also holds when using inheritance as a variability mechanism in a software product line context. The main difference is that the components in such a context have a much greater granularity. If this would not be the case, the leverage of using the software product line would be diminished.

5.3 EXTENSIONS

The third method discussed here is the use of extensions. Extensions are use case based mechanisms. Extensions are realized through so-called extension points, i.e. variation points where use cases can be extended with additional behavior. One can develop the additional behavior himself or make use of existing variants, if any.

5.4 PARAMETERIZATION

Yet another method for introducing variability is by using parameters. Typically, four different types of parameters are used. First, compilation parameters are parameters that are set at compile time. An example of such a parameter is the usage of `#ifdef` statements in the programming language C. The second type of parameters is configuration parameters. These parameters are set at configuration time. Such parameters are used for instance with installments of operating systems. The third type comprises runtime initialization parameters. These values are set during the initialization phase of a system. Finally, table-driven parameters are used. When using this type of parameterization, the users can configure the system by modifying parameters that are stored in a table. This is comparable with the symbol table that a compiler uses to store the semantic representation of processed source code, i.e. it is represented in an intermediate form. Even though different tables contain the same symbols, their semantics are likely to differ.

5.5 GENERATION

The final method discussed here is source code generation. A component generator generates components and various relationships between these components. Such a generator requires input specified in a specialized specification language. An example of a generator is for instance the model driven architecture. Model-driven-architecture tools require input specified in the Unified Modeling Language. Based on these models, the model-driven-architecture tools generate source code.

6. SUMMARY

A software product line is defined as "*a set of software-intensive systems sharing a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way*". The two key words regarding software product lines are commonality and variability.

The former is concerned with the use of core assets and the latter with variability mechanisms and product-specific assets. Core assets comprise, among others, a product line architecture, reusable components, and production plans.

The cost types of a software product line program are equal to the cost types of a typical software reuse program. The actual values for these cost types of software product line practices often are relatively lower, due to the fact that the entire process is more structured, e.g. because of the use of production plans. The benefits to be gained from software product lines encompass much more than just the standard reuse benefits, the tactical engineering benefits. These additional benefits are referred to as strategic business benefits and are derived from the tactical engineering benefits. These additional benefits hold in a software product line context specifically due to the structured process of deriving products.

Software product line practice consists of the following three fundamental activities: core asset development, product development, and management. Core asset development is concerned with developing the core asset base, the collection of core assets to be used in actual product development. However, different products have different requirements. To deal with this, multiple variation mechanisms can be used to provide the necessary variability. These variation mechanisms are the following: façades, inheritance, extensions, parameterization, and generation. Product development aims at developing products within the product line scope making use of both core assets and product-specific assets. Finally, management is concerned with orchestrating core asset development and product development at both the operational and the organizational level.

CHAPTER 7
PHILIPS MEDICAL SYSTEMS REUSE ACTIVITIES

The previous three chapters introduced the basic of software engineering taking a software product line approach. There are many different ways an organization can implement a software product line effort. All kinds of aspects play a role, e.g. how to structure the organization, how to fund the core asset development, how to assess the initiative, and who should fulfill what role. This chapter describes an overall view of the path that Philips Medical Systems chose to implement their software product line.

This chapter discusses the software reuse program that is being institutionalized at Philips Medical Systems. First, the three aforementioned programs of Components & Services, i.e. Medical Imaging Platform, UI Harmonization and Interoperability, are presented in more detail. The second section then discusses their incremental approach to adopting software product line practices. The subsequent section presents upcoming organizational changes with respect to the software reuse program. Finally, the chapter concludes with a summary.

1. COMPONENTS & SERVICES

In the previous part, the business unit Components & Services was identified as the sole core asset developer. In addition, the market demands driving Philips Medical Systems were discussed briefly. Components & Services currently is running three programs, which are all associated to these market demands. This section provides insight into the association between the market demands and the Components & Services programs. First, an elaboration on these demands is provided. The following subsections illustrate the associations between these demands and the Components & Services programs. The final subsection provides a brief overview.

First, regard the demand for multi-modalities. As mentioned earlier, a multi-modality is a product combining multiple modalities, e.g. a PET scanner and a CAT scanner. This is useful because different scanners return different types of data. In order for modalities to communicate with each other, a communication standard has to be used. Currently, the generally accepted standard is DICOM, which stands for Digital Imaging and Communications in Medicine [23]. Both the Medical Imaging Platform program and the Interoperability program address the issue of interoperability.

Since one vendor typically does not produce all the necessary medical appliances and, in addition, hospitals do not want to be dependent on one vendor, because of possible production faults, different products from different vendors need to be capable of interoperating. Consequently, besides the demand for multi-modalities, customers demand a high degree of interoperability between products from different vendors also. Since DICOM should provide such interoperability, the customers demand that all appliances adhere to this standard. The program Interoperability addresses this.

Customers have expressed their desire for one look and feel for all the products from one vendor. This implies that not only the graphical user interfaces need to be aligned, but the hardware interfaces need to be aligned also. In order to realize a unified user interface, Components & Services is running the program UI Harmonization. The actual unification is realized through the MIP assets and, therefore, the both the program UI Harmonization and the MIP program address the unification of the user interfaces.

The final market demand is that the time-to-market for new features should be shortened. One of the promises of applying a software product line approach is that it should lead to a shortened time-to-market. Consequently, the MIP program addresses this issue.

Besides these demands, Philips Medical Systems also wants to decrease the development costs, decrease the maintenance costs, and increase the quality of their products. The following three subsections describe the three aforementioned Components & Services programs in more detail.

1.1 MEDICAL IMAGING PLATFORM

Philips Medical Systems adopted the software product line approach for the development of their products. This is possible since their products exhibit many commonalities, e.g. database handling, viewing, and printing. They can exploit these commonalities by initiating a software product line effort. Components & Services therefore started the program Medical Imaging Platform, also referred to as MIP, which is the actual implementation of the software product line effort. MIP was initiated in the year 1998 and it has matured much since then. The MIP assets are divided over the following five segments: base, services, connectivity, database, and viewing. During the last years, several MIP assets have been deployed in different products, with varying success. The costs and benefits to these uses often have not been quantified.

Components & Services deploys a MIP release every half year. The other business units can place requirements for the next MIP releases. Based on the demand for certain functionality, Components & Services decides whether or not a requirement is included in the next release. Obviously, the most demanded functionality will be included first.

All the business units using MIP assets have to pay up-front for these assets on a yearly base. The necessary budget for the following year is determined and the participating business units will have to pay their proper share. The current funding model requires a business unit to pay his share for all segments that it is using. Two types of uses are to be identified, namely feasibility testing and deployment in actual products. Given one segment, business units using the core assets for feasibility testing only pay half the amount the business units using these assets in product releases do. To cover for all the costs related to the MIP activities, such calculations are made for all of the five segments.

Currently , solely the business unit Components & Services develops MIP assets. All the other business units only make use of these assets. This is illustrated by the figure below.

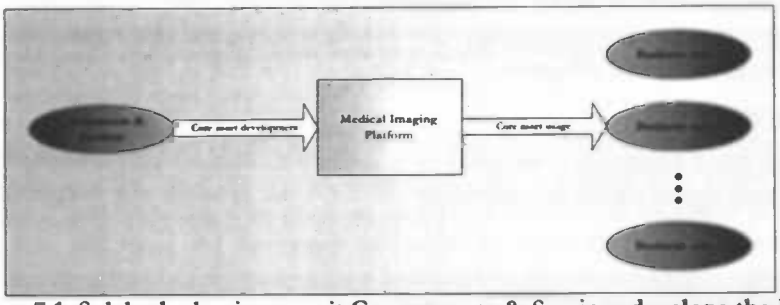


Figure 7.1: Solely the business unit Components & Services develops the MIP assets and all the other business units deploy these assets in their products

1.1.1 RELATION TO THE MARKET DEMANDS

As mentioned earlier, the market demands a faster implementation of new features. One of the four promises of software reuse, and in particular software product line practices, is that the time-to-market of new products is shortened. Besides this demand, Philips Medical Systems is interested in strengthening their strategic position, for instance by shortening the time-to-market and increase the overall product quality. The latter is a typical benefit of software product lines also. Besides strengthening the strategic position, Philips Medical Systems wishes to reduce the development costs and the maintenance costs. Applying a software product line effort enables this. Especially lowering the maintenance costs is of great importance, since modalities have to be maintained for approximately a decade after they are shipped.

By applying the software product line approach, standardization over all the products in the product line scope is enforced. This holds for standardizing the graphical user interface also.

Finally, interoperability and multi-modality become more feasible. This also is a result of the use of common assets across multiple products. The different products each exhibit a similar software architecture, i.e. a derivation of the product line architecture. Therefore, all the assets have a similar level of abstraction, which enables multi-modality. When different levels of abstractions would be used, different levels of functionality would be encapsulated in the assets also.

1.2 UI HARMONIZATION

The program UI harmonization is aimed at providing a unified interface for all the Philips Medical Systems products. This holds for both the hardware components and for the software components of these products. All the scanners exhibit similar shapes, colors and icons and, all the software applications exhibit similar colors, icons and menus. More importantly, users familiar with one Philips Medical Systems product should intuitively know how other Philips Medical Systems products work also. This is for instance reflected in the slogan currently used by Philips: Sense and Simplicity. The UI Harmonization is, among others, realized through the use of MIP assets. The assets provide the GUI and can be tailored according to specific product requirements.

1.2.1 RELATION TO THE MARKET DEMANDS

The program UI harmonization is related solely to the demand for one look & feel. It provides, however, input for the MIP program.

1.3 INTEROPERABILITY

Interoperability is established through the use of DICOM. By adhering to the DICOM standards, the interoperability between different medical appliances, even from different vendors, should be guaranteed. Since all medical appliances should adhere to this standard, it enables multi-modality also. Examples of agreements in the standard are data representations and naming conventions. Because different appliances need to share information, this is particularly useful regarding multi-modalities.

1.3.1 RELATION TO THE MARKET DEMANDS

The program Interoperability should ensure the interoperability between different products, ranging from monitors to printers. In addition, multi-modality could be regarded as the extension of this principle, because multi-modality requires interoperability between different scanners. Again, the interoperability program provides input for the MIP program.

1.4 OVERVIEW

As stated above, all three programs contribute to some extent to meet all the aforementioned market demands. The following figure shows schematically the relations between the programs and the market demands.

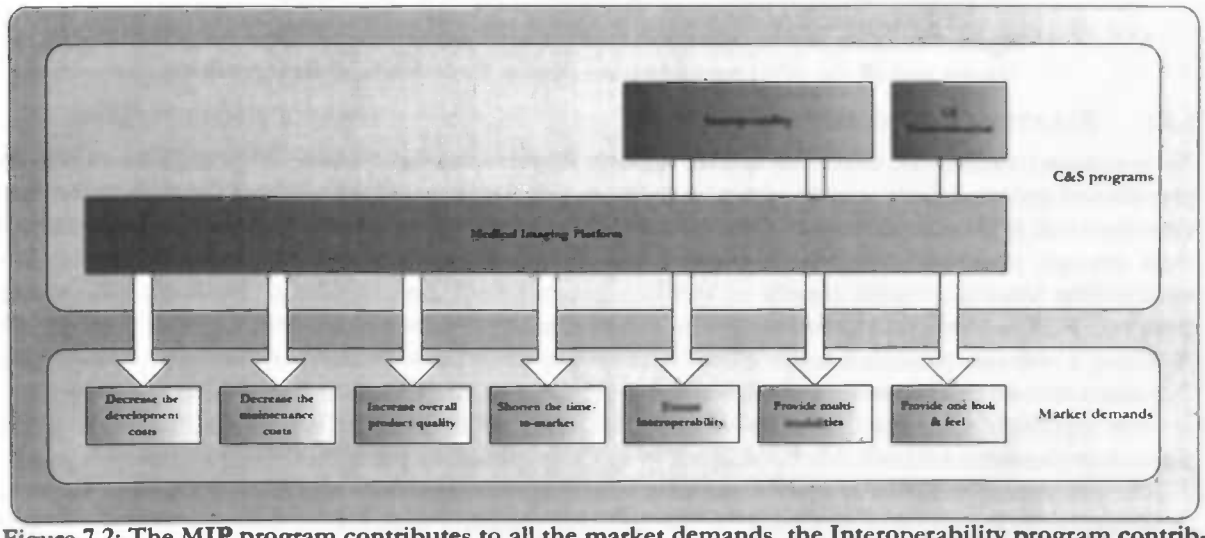


Figure 7.2: The MIP program contributes to all the market demands, the Interoperability program contributes to the interoperability and the multi-modality demands and the UI Harmonization program contributes to the unified look and feel of all the Philips Medical Systems products

2. INCREMENTAL REUSE

The products included in the product line scope are all composed of the following nine segments: Console Workstation Workflow Engine IC, Acquisition Control, Data Handler IC, Workspot Viewer IC, Clinical Applications, Services, Connectivity, Database, and Viewing. The typical infrastructure consists of a PC running the Windows XP operating system and providing the .NET environment. The traditional development process of Philips Medical Systems is that all products are developed singularly, i.e. each business unit was operating autonomously and independent of each other. Consequently, all nine segments were developed and maintained by each business unit separately, as is illustrated by the figure below. In that figure, an orange segment illustrates a segment that is reused. If a segment consists of multiple colors and bos, it is developed by each business unit individually.

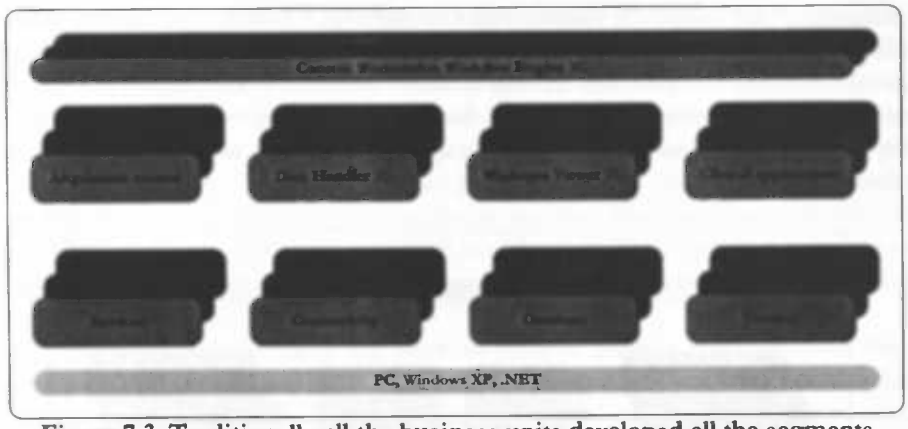


Figure 7.3: Traditionally all the business units developed all the segments individually

As mentioned earlier, all the products included in the product line scope exhibit similar functionality. The MIP program is aimed at exploiting the commonalities in the following five segments: Base, Services, Connectivity, Database, and Viewing. The platform provides core assets to cover the common features across the product line, including the necessary variability to make these assets usable across multiple products. Figure 7.4 illustrates which segments are reused currently, and, more specific, exhibits the growth in reuse. It should be noted, however, that not all business unit have adopted the MIP assets in all the segments yet. For simplicity, this has been omitted in the figure.

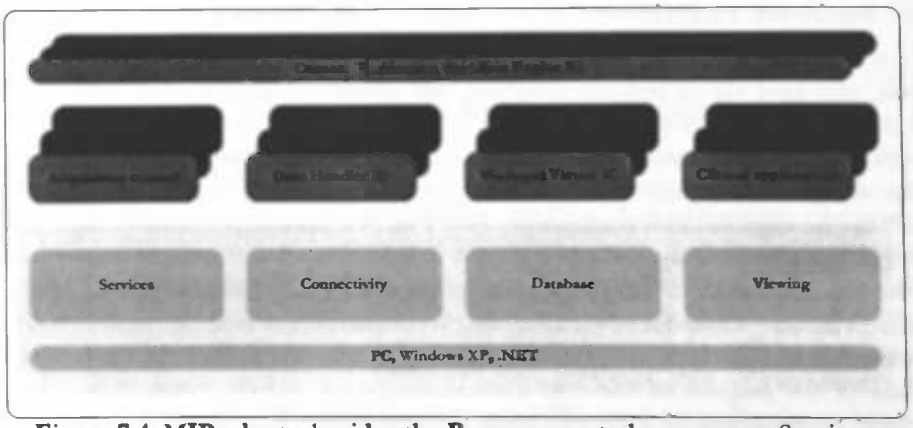


Figure 7.4: MIP adopts, besides the Base segment, the segments Services, Connectivity, Database, and Viewing

Regarding the MIP releases, multiple issues arise. First, since different business units are using different MIP releases, multiple versions of the platform need maintenance. The second issue, strongly related to the previous one, is that the MIP releases do not provide backwards compatibility. Because multiple versions of MIP assets require maintenance, the reduction of the maintenance costs obviously suffers. Also, when MIP releases are not backwards compatible, there is no direct incentive for business units to adopt a

newer version of the MIP release. The final issue mentioned here is that not all business units have adopted the same segments. For instance, one segment might first adopt the Viewing segment, while another business unit first adopts the Connectivity segment. This way, different versions of different segments are in use.

One way to deal with this is to provide a more complete solution, i.e. a semi-finished product. It should be backwards compatible, so business units can easily adopt the latest release. In addition, the maintenance effort can then be fully dedicated to support the latest release(s). Such a semi-finished product is currently in development and it is referred to as the Philips Medical Workspot. There are currently two types of the Philips Medical Workspot in development, namely a standard version and a real-time version.

The Philips Medical Workspot is a solution making use of MIP assets and it provides, besides the MIP segments, the Data Handler IC, Workspot Viewer IC and the Console Workstation Workflow Engine IC. The remaining two segments, Acquisition Control and Clinical applications, will have to be developed and maintained for each product individually. These two segments also are the specialization of the business units, i.e. where the products exhibit mainly variable functionality, so little leverage could be achieved by searching for commonalities. The following figure illustrates the principles of the Philips Medical Workspot from a software product line perspective. Figure 7.5 clarifies that the reuse level over time has increased drastically.

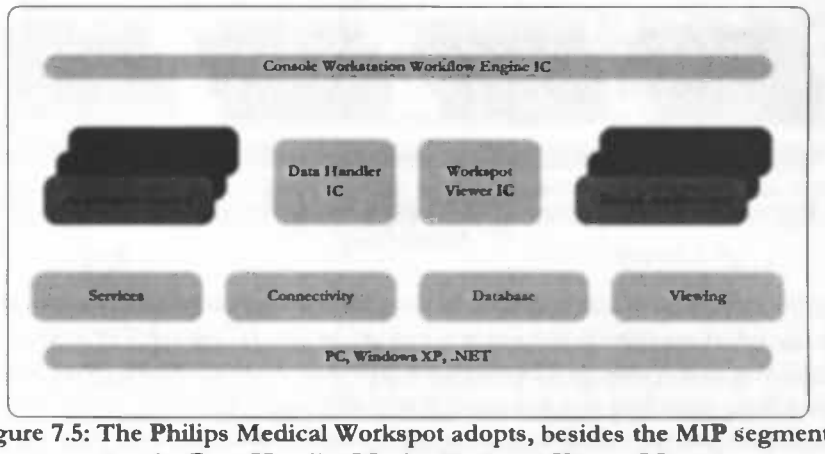


Figure 7.5: The Philips Medical Workspot adopts, besides the MIP segments, also the Data Handler IC, the Workspot Viewer IC, and the Console Workstation Workflow Engine IC

3. MIP PRODUCT LINE

In the previous chapter, software product lines were defined as "a set of software-intensive systems sharing a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way". This section is concerned with mapping the above definition on the approach of Philips Medical Systems.

Given the above product segments, it could be stated that the Philips Medical Systems approach to software reuse can be regarded as a software product line. It indeed consists of multiple software-intensive systems sharing a common managed set of features. In addition, all systems are meant for use in one market, namely the medical market. Finally, the systems are developed from a common set of core assets, i.e. the MIP assets. The only difference regarding the above definition and the practices of Philips Medical Systems, however, is the fact that the development of products should take place in a prescribed way. As mentioned, one of the outputs of the core asset development activity constitutes, among other things, production plans, i.e. programmer guides to using the core assets. Due to the lack of such documentation, the business units do not have enough knowledge about using these MIP assets. The problems resulting from this lack of documentation are threefold and will be discussed in the following.

First, the architectural solution, i.e. the MIP product line architecture, is not documented in a proper manner. For instance, a document providing an architectural overview is missing. Such a document should describe the global design, the design decisions, and the corresponding rationale. The business units therefore have difficulties deriving a software architecture from it. The second problem with the documentation is that it only describes what assets can do, i.e. a functional description of the interfaces, not how to tailor them for use in a particular product context. A manner to deal with this is for instance to provide

example implementations, which function as references for the business units. Finally, the documentation does not address the quality attributes of the MIP assets and the presumptions regarding the context are not explicated, e.g. how much memory or CPU time is required for certain actions. For instance, an MR scanner requires a high throughput of images; it processes approximately thirty thousand images per second. Consequently, an MR scanner has high demands with respect to certain quality attributes, e.g. throughput, performance. It should be clear in advance whether or not the MIP assets are capable of adhering to these requirements. Zooming in on the third problem, Components & Services does not validate the quality attributes of the MIP assets for all possible product contexts. Two next steps for addressing this problem seem apparent. First, it should be specified, for instance through an explicit product line scope, for which major contexts the MIP assets have been designed. The use of MIP assets in these contexts should be validated by incorporating the quality attributes of a number of representative modalities to the test case. This manner, the risks of using the MIP assets outside the intended area are apparent for the business units. If the need arises, the business units contact Components & Services for additional support and advice beforehand. Second, the modalities with high demands can be excluded from the product line scope. An argument for doing so is that might be not economically feasible to support these modalities. If this is the case, this could arise the need for an additional software product line, i.e. a product line specifically supporting the modalities with high demands. If the high demands only hold for certain assets of the software product line, e.g. database handling, setting up a software product line on top of the two product lines might be a solution, i.e. a software product line providing variability to derive either a product line for high demands or the current MIP product line. This, however, requires additional management of commonalities between not only products, but also between a higher abstraction of the products, i.e. between software product lines.

Either way, Components & Services will have to provide additional support for the integration of MIP assets and, by doing so, become more aware of the problems during integration, e.g. due to poor performance. In addition, Components & Services has to collaborate with the business units to define the required and actual values for the quality attributes in a proper manner. Components & Services should thus become more aware of the product contexts and the corresponding presumptions.

Because of the lack of proper documenting the architectural solution, lack of validating the behavior of MIP assets in all product contexts, lack of documenting the quality attributes of the MIP assets, and dependencies on legacy systems, the behavior of MIP assets in certain product contexts is unpredictable. This is enforced by the fact that a number of business units still use a legacy architecture. This has led to numerous integration problems and consequently negative experiences with the MIP assets.

One of the key elements of a software product line effort is to standardize both development processes, i.e. core asset development and product development. For instance, core assets should have attached processes associated with it and it should be dictated how the core asset base should be updated as the product line evolves. Regarding product development, production plans should prescribe how core assets can be utilized to derive products, specifically how to tailor these assets. In order to benefit fully from their software product line approach, it is imperative that Philips Medical Systems standardizes their development processes. Fortunately, Components & Services has acknowledged this.

4. ORGANIZATIONAL CHANGES

Based on the above, multiple goals have been identified. Philips Medical Systems first wants to meet the market demands. But, they also wish to reduce the development costs, reduce the maintenance costs, and increase the overall product quality. In order to do so, multiple organizational changes will have to take place. The business unit Components & Services does not have enough capacity to develop and maintain all segments for all business units themselves. In time, the business units will have to contribute to the core asset base also. This, however, leads to the first problem. The mindset of a core asset developer typically is different from the mindset of a product developer. The core asset developer has a long-term vision, while the product developer is more likely to have a short-term vision. This is best illustrated by the phrase "e pluribus unum", which stands for "out of many, one" [11]. It is stated that the best product line organizations view their business as building one product line. All business units have the long-term health of the product line as their major concern. In contrast, other organizations tend to view their business as turning out products. The main difference between the mindsets of the core asset developer and the product developer is that the product developer will adapt a core asset to his specific needs, if necessary. He is driven by business demands and looming deadlines rather than by the long-term health of the software product line.

In the adoption of a software product line approach at Philips Medical Systems, three different phases can be identified. The first phase is that Components & Services acts as the sole core asset developer. The second phase consists of collaborations between the different business units to develop core assets together. The final phase is that all business units develop core assets. These phases will be referred to as sole asset development, joint asset development and parallel asset development respectively and will be discussed in the following.

4.1 JOINT ASSET DEVELOPMENT

The sole asset development phase is initially characterized by the lack of core assets. During this phase, the focus will be on defining the product-line scope, determine the most appropriate product-line architecture and initiate the development of core assets. Since there will be few to no product releases during this phase, strong management commitment is necessary to sustain the initiative. The funding issue peculiar to this phase is that only costs are apparent, not benefits. The first attempts of deriving products from the software product line should be supported by higher management. The initial product derivations function as pilot projects, e.g. to validate the product-line architecture.

4.1.1 PHILIPS MEDICAL SYSTEMS

The initial practice was that solely the business unit Components & Services developed MIP assets. This led to the following issues. First, because Components & Services was the sole developer of core assets, their production capacity formed a bottleneck. This was enforced by the fact that multiple versions of multiple core assets needed to be maintained. Second, Components & Services validated all their assets in an isolated manner, i.e. they did not regard actual product contexts. Therefore, Components & Services had little experience in using their core assets in actual products. The final issue mentioned here is that Components & Services developed assets without providing the proper documentation. This has led to difficulties regarding tailoring and integrating MIP assets in product contexts, e.g. due to the fact that business units do not understand the architectural solution or which presumptions regarding certain assets are made.

4.2 JOINT ASSET DEVELOPMENT

A growing, maturing core asset base characterizes the joint asset development phase. The product line scope and the product line architecture are defined properly and both can now be used to design and develop additional core assets. These core assets can then be tailored and used for the derivation of actual products. Since the core assets are used for the first time in a product context, several childhood illnesses might have to be dealt with. Examples of childhood illnesses are memory leaks, performance problems, and inferior programmer guides. The first user of a core asset therefore functions sort of as a guinea pig, i.e. all childhood illnesses need to be identified and resolved during their derivation process. Another issue regards the time-to-market. As mentioned, one of the benefits of software reuse is a shortened time-to-market. This, however, does not hold for the initial derived products from the software product line. This is, among other things, because the development of a core asset requires more effort than the development of a product-specific asset requires and that possible childhood illnesses have to be dealt with.

Considering the aforementioned issues, it seems sensible to compensate the first user of a core asset. A funding model can for instance function as a means to provide a monetary compensation. Another compensation method is to provide additional support during the integration of the core assets. This way, the core asset developers experience from first-hand how the core assets behave in a product context and the product developers get affinity with using the core assets in their products more rapidly.

Another issue is that the different product developers need to adopt the core assets in their product development process, i.e. make use of core assets instead of developing similar product-specific assets from scratch. If the use of core assets has not yet proven to be more beneficial than developing each product individually, for instance due to too many childhood illnesses, an artificial incentive seems appropriate to stimulate the adoption of core assets. The adoption of core assets not only is necessary to support the software product line initiative, it also aids in providing the core asset developers with feedback, e.g. feedback on how well the core assets perform in a product context and which problems need resolving.

The characteristic of this phase, which distinguishes this phase most from the sole asset development phase, is the collaboration between the core asset developers and the product developers. In the sole asset development phase, the core asset developers develop and maintain all core assets and, therefore, the capacity of the core asset developer forms a bottleneck. By collaborating with the business units, this bot-

tleneck is less apparent. To initiate such collaborations, an artificial incentive seems in place. This is due to the higher development costs of a core asset compared with the development costs of a similar product-specific asset. One method to compensate is to exchange engineers from the core asset developers to the product developers. In order to stimulate the initiation of these collaborations, artificial incentives seem appropriate also, e.g. monetary compensations. These collaborations have the benefit that the core asset developers experience how their core assets behave in product contexts the hard way and that the product developers not only learn to use core assets, but also learn to develop core assets.

4.2.1 PHILIPS MEDICAL SYSTEMS

Developing MIP assets differs from developing product-specific assets in many ways. An important difference is that the development of MIP assets should be driven by maintaining the long-term health of the MIP product line and developing product-specific assets is driven by business, e.g. time-to-market. The above suggests that for the business units to be capable of developing MIP assets, they would have to adopt the mindset of a core asset developer, i.e. look at the long-term health of the software product line, rather than act solely on business demands. Components & Services is currently collaborating with business units to develop MIP assets together and this has proven to be a fruitful method. Such a form of collaboration namely provides many benefits. First, the bottleneck formed by Components & Services is less apparent, since they now have support from other business units. Second, the business units learn to work with the core assets. This results in more understanding by the business units how these core assets are structured and how they can be used in products. Finally, the business units have more confidence in the quality of the core assets, since they have helped building the assets. The emphasis of this phase is on adopting the core asset developer mindset at the business units.

4.3 PARALLEL ASSET DEVELOPMENT

A mature core asset base characterizes the parallel asset development phase. The core asset base is to some extent stable and only is changed for maintenance or when an alteration of the product-line scope needs to occur, e.g. due to changing market demands. Since the core assets do evolve, new releases of these assets will be released periodically. Typical issues regarding new releases of core assets are the adoption of these assets in the product releases and backwards compatibility. These two obviously are strongly related. Providing backwards compatibility is hard, since this implies that interfaces need to be stable. When using wrappers to introduce new interfaces, this will lead to performance problems. When a core asset does provide backwards compatibility, a business unit is likely to adopt the newer version. However, when backwards compatibility is not provided, the benefits of adopting the newer version of a core asset might not outweigh the costs of adopting the newer version, at least not from a business unit's perspective. Consequently, the core asset developers need to maintain multiple versions of a core asset. This obviously diminishes the leverage of the centralized maintenance. But, it also leads to additional pressure on the core asset developer's product capacity, which formed a bottleneck to begin with. This is due to the focus shift from asset development to asset maintenance.

If all went well, the product developers have learned in the previous phase how they can develop a core asset. The characteristic that distinguishes this phase from the previous phase is that product developers are able to act as core asset developers, i.e. they develop core assets individually. Obviously, since the development of a core asset requires more effort than developing a product-specific asset does, an incentive seems in place. Besides monetary rewards or additional support, one can also propose that when a business unit develops a core asset, the core asset developer takes over the future maintenance of that asset. Typical difficulties are to determine whether or not an asset actually is a core asset and how the tacit knowledge regarding that core asset can be transferred from the business unit to the core asset developer.

4.3.1 PHILIPS MEDICAL SYSTEMS

The final phase is characterized by the fact that all business units contribute to the core asset base. This is referred to as the organizational concept Component Warehouse. By doing so, all business units can specialize in a certain segment of assets. The benefits of joint asset development are enforced by this phase. The emphasis will be on rapid development of high-quality core assets.

4.4 CURRENT STATE OF THE EFFORT

Philips Medical Systems currently is in the transitioning from sole asset development to joint asset development. One of the first projects of combined asset development was with the business unit PET and Components & Services. Both agreed that the project is successful in its intention. The developers from

PET indeed are acquiring feeling for core asset development, i.e. they are adopting the core asset developer mindset and they believe that they can integrate future core assets more efficient due to this experience. In addition, Components & Services obtained more affinity with the behavior of their core assets in actual product contexts, i.e. they experience the benefits and drawbacks of using the MIP assets from first-hand.

To have business units collaborate on core asset development also leads to certain difficulties. First, if a business unit develops a core asset, who will become the owner of the core asset? Will they hand it over to Components & Services or will they remain the owner of the core asset. A second difficulty is how to deal with the maintenance and related costs. A business unit will not likely be willing to maintain a core asset for other business units without some form of compensation. Such compensation could for instance be a monetary reward. If a business unit remains the owner of a core asset, the other business units have to pay that business unit for using the subject asset. This calls for additional cash flows.

The fact that business units develop core assets characterizes the third stage, i.e. parallel asset development. An additional problem arises with this. If for instance core assets are handed over to Components & Services, how should the tacit knowledge regarding these assets be transferred? Engineers that have worked on these assets, will they be transferred also? Does proper documentation suffice? Finally, if a business unit is compensated for developing core assets rather than product-specific assets, who validates that the proclaimed core asset indeed incorporates the required variability?

Philips Medical Systems currently is in the transition from being in the sole asset development phase to being in the joint asset development phase. This is reflected by the fact that the MIP assets have matured, the first collaborations to joint development of MIP assets are taking place, and that Components & Services as a business unit has matured.

5. OVERVIEW OF CONCERNS

Concerning the institutionalizing of the software product line effort, all the employees whom are related to software development, ranging from software engineers to upper management, are affected somehow. The upper management demands for a high level of reuse and that the demand for a unified look and feel for all the Philips Medical Systems products is met. The former is for instance enforced by cutting the budgets of the business units. A business unit must have solid reasons for not using MIP assets and therefore not be cut in their budget.

At the usage level, two types of stakeholders can be identified. The first type of users is the ones who develop the core assets, Components & Services. Their interest in the appliance of software reuse is threefold. First, it is in their best interest that all the business units make use of as much core assets as possible, since this justifies their existence. Second, they also want the business units to use the most recent releases of the core assets, since this would result in less maintenance effort. As mentioned earlier, shifting the focus from development to maintenance leads to less innovating development. Finally, due to the fact that Components & Services currently forms the bottleneck, it is in the best interest that the business units contribute to the core asset base also, the parallel asset development phase. This strengthens the position of Components & Services, since the core asset base grows more mature more rapidly, on the condition that acceptance of core assets to the core asset base is managed correctly and the maintenance and ownership issues are resolved in a satisfactory manner. Upper management supports the activities of Components & Services, which enforces their position.

The second type of users is the business units using the core assets in their product releases. As became clear from the many interviews, the biggest concerns of the business units are the quality attributes of the core assets and the usability of the core assets. This is related to the fact that their main business concerns consist of shortening time-to-market, reduce development costs, and reduce maintenance costs. Typical issues regarding the business units and Components & Services are the lack of trust and the poor performance of assets in actual product contexts. The former is addressed by involving the business units more in the core asset development process, e.g. the joint asset development phase. The latter is addressed in two ways. First, since Components & Services cooperates with business units to develop core assets, they are also more involved in testing the core assets in a product context automatically. Besides this, they also started testing the MIP assets in product contexts on their own.

The above leads to the need for the following changes. First, Components & Services needs to provide proper documentation on the tailoring of MIP assets for product contexts and regarding the quality attributes of the MIP assets. Components & Services also has to shift its focus from developing MIP assets to supporting the development of MIP assets by other business units. This implies that the other

business units have to adopt the core asset developer mindset in order to develop these MIP assets. In short, the usability of the MIP assets needs to be improved and the collaborations between Components & Services and other business units are a manner to identify and resolve the difficulties.

6. SUMMARY

Components & Services currently is running three programs, namely Medical Imaging Platform, Interoperability and UI Harmonization. The Medical Imaging Platform comprises the software product line initiative of Philips Medical Systems, Interoperability is aimed at validating interoperability between different products, and UI Harmonization is concerned with the unification of both the hardware interfaces and the graphical user interfaces. Philips Medical Systems is institutionalizing their software reuse practices in an incremental manner. The key element from their software product line effort that currently is missing is the prescription of how products can be derived from the software product line; it is not specified how a software architecture can be derived from the product line architecture and it is not specified how the core assets should be tailored for use in certain products. Philips Medical Systems currently wants to get all the business units involved in the core asset development process. In order to do so, they will have to go through three overlapping phases, i.e. sole asset development, joint asset development, and parallel asset development. The general idea of these phases is that with each subsequent phase, the interference of the business units increases. Also, their expertise regarding both developing and using core assets increases. The focus of Components & Services will shift from core asset development to supporting core asset development.

PART THREE
FUNDING

This part is composed of four chapters. The first chapter discusses funding from a theoretical approach. The second chapter then discusses the funding models applied by Philips Medical Systems. The subsequent chapter is used to propose a funding model. In order to develop ideas for such a model, many interviews have been held and a questionnaire was sent out. The final chapter presents the validation of the proposed funding model.

CHAPTER 8
FUNDING FUNDAMENTALS

When comparing traditional software development with software development taking a software product line approach, many differences can be identified. One of the major differences is how one can allocate costs to actual products. Traditional software development is organized around projects leading to the development of a single product. Hence, all the development costs can be allocated to that single product. When taking a software product line approach to product development, however, costs cannot be allocated that easily. Why this is so and, more importantly, how to deal with will be discussed below.

This chapter provides a theoretic perspective on the notion of funding in a software product line context. First, the four cost types relevant for a software reuse program are categorized and analyzed. Then, two subjects relevant to funding software reuse will be discussed, i.e. the three fundamental approaches to funding in general and multiple approaches to manipulate behavior. The fourth section is concerned with the issues typically for a software product line context that can be addressed with funding. The subsequent section discusses multiple funding models, as proposed in the literature. The chapter concludes with an overview.

1. COSTS ANALYSIS

Funding in general is concerned with providing funds to cover for the costs of performing certain activities. Typically, two dimensions of costs can be identified, namely variable costs versus fid costs and direct costs versus indirect costs. These two dimensions will be explained in the following.

First, the variable costs and the fid costs dimension will be explained. Variable costs simply are costs which increase when the output increases also. An example of variable costs are the salaries of employees. The longer employees are working at a conveyor belt, the higher the output of that conveyor belt and, correspondingly, the higher the salary costs. Fid costs, however, are costs that are spent, regardless of the output. An example of fid costs are the costs of renting a production facility. The rent needs to be paid, whether or not the production facility is in use.

The other dimension of costs is direct versus indirect costs. Direct costs can be traced directly to the products. An example is an employee working on a product. The salary costs of that employee are directly related to that product, i.e. it influences the development costs of that particular product directly. Indirect costs do not exhibit this property. Therefore, linking indirect costs to actual products is much harder. As an example, regard the administrative support. The administrative support deals with all the paper work surrounding the production of products. However, when multiple products are being developed, it is hard to determine how much of the administrative support costs are related to each specific product. In [4], a more thorough discussion on these two dimensions in general business practices is presented.

1.1 PRODUCT LINE COSTS

As mentioned in chapter 5 and chapter 6, the following four cost types related to software reuse practices can be identified:

- C_{org}, the costs that an organization makes to adopt the software product line practices
- C_{cab}, the development costs of the core asset base
- C_{unique}, the development costs of product-specific assets
- C_{reuse}, the costs of integrating core assets in products

These four costs can be classified according to the aforementioned two costs dimensions. First, consider C_{org}. These costs typically only appear in the beginning of the software product line practices, i.e. in the transitioning phase. The transition from the traditional organization to a reuse organization requires multiple organizational changes. Because a new business unit is needed, i.e. a core asset developer, a reorganization of the structure is for instance required. Also, the core asset developers need to adopt the core asset developer mindset rather than the product developer mindset, which can for instance be realized through training. The main difference between these mindsets is that the former takes the long-term health of the

software product line into account and the latter is more focused on developing products on time. Because there is no direct relation between the products and these costs and, there is no link between these costs and the output of products, these costs can be classified as indirect and fid.

The second cost type is C_{cab}. It constitutes of the costs for developing, maintaining, and evolving the core asset base, which typically occurs during the entire lifecycle of the software product line. The core asset base consists of all the core assets which can be used in all the products within the product line scope. To some extent, there exists a relation between the product line scope and a part of the expenses related to the core asset base, i.e. the part that provides the variability. The more variability is required, the higher the development costs for the corresponding assets. This, however, is a link to the product line scope instead of the actual products. Because there is no direct relation between the products and these costs and, there is no link between these costs and the output of products, these costs can be classified as indirect and fid.

Both C_{unique} and C_{reuse} are related to actual products. The former is concerned with the costs of developing, maintaining, and evolving the product-specific assets, and the latter with tailoring and integrating core assets into actual products. Both have to be performed for each product individually and, therefore, the related costs increase when the output increases also. Because there is a direct relation between the products and these costs and, there is a link between these costs and the output of products, these costs can thus be classified as direct and variable.

Obviously, the two cost types that cannot be allocated directly to products, i.e. C_{org} and C_{cab}, do have to covered for somehow. In order to deal with such indirect costs, many funding models have been proposed. A funding model prescribes how certain costs should be allocated to different business units. Since both C_{unique} and C_{reuse} have a direct relation with product development, these costs can be allocated to the subject products. All the four cost types and the two dimensions are presented in the table below:

Table 8.1: Overview of cost dimensions and the reuse costs

Cost type	Dimensions	
	Variable/Fid	**Direct/Indirect**
C_{org}	Fid	Indirect
C_{cab}	Fid	Indirect
C_{unique}	Variable	Direct
C_{reuse}	Variable	Direct

2. FUNDING APPROACHES

In accounting, typically three different approaches to funding can be identified. These approaches will be discussed next. The first approach is the causal approach. An example of such an approach is the activity-based costing model [1]. The causal approach prescribes that the costs need to be distributed across all the entities responsible for these costs. The aspect peculiar to this approach is that it takes the specific shares for the expenses into account. For instance, if one business unit would request two third of the functionality of the variable part of a core asset and another business unit requests the other one third, the former has to pay two third of the costs and the latter would have to pay one third. This approach to funding is fair in the sense that everybody pays his proper share for certain overhead activities. It, however, does hold back innovative development in an organization, which will be illustrated by the following approach to funding.

The second approach is based upon the principle of fairness. For instance, when developing an innovative product with a large potential, a rising star [24], revenues are expected to show in a later stage. It would be perceived as being unfair to charge the developers of that innovative product as much as the ones whom are producing a product with stable and large revenues, also referred to as a cash cow [24]. When using activity-based costing, this obviously is possible, since the developers of a rising star rely heavily on, for instance, the research-and-development and marketing entities of an organization. As a consequence, the costs of these entities would be strongly related to those developers. This would obstruct the process of innovation within an organization, since it is then less attractive to innovate, if possible at all. It would be perceived more fair if the revenues of the developers of the cash cow are used to sponsor the activities related to innovations of an organization.

The third approach can be regarded as an income tax. The ones with the highest incomes are charged the most. The income tax funding model, which will be discussed in below, is an example of such a model.

3. MANIPULATING BEHAVIOR

In [30], four different approaches regarding the manipulation of behavior are proposed, which will be discussed briefly in this section. Behavior is strengthened when it increases in frequency and behavior is weakened when it occurs less frequent. The first two approaches are aimed at strengthening desired behavior. The other two are aimed at weakening undesired behavior.

3.1 POSITIVE REINFORCEMENT

The first altering method discussed here is positive reinforcement. Positive reinforcement is the process of presenting something pleasing when the behavior aimed at is exhibited. An example of positive reinforcement is to reward the use of core assets in product releases with monetary gifts.

3.2 NEGATIVE REINFORCEMENT

Negative reinforcement is the process of taking something displeasing away when the desired behavior is exhibited. This is illustrated by the following example. The maintenance of core assets is taken care of by the core asset developer and the maintenance of product-specific assets is taken care of by the product developers. However, if a business unit develops a core asset instead of a product-specific asset, they will be rewarded by the core asset developers, since they will take over all future maintenance for that asset.

3.3 PUNISHMENT

Punishment can be regarded as an opposite of both positive and negative reinforcement. Therefore, punishment has two flavors. First, regard the opposite of positive reinforcement, i.e. take something pleasing away, when undesired behavior is exhibited. An example of presenting something displeasing is that when a business unit does not adopt enough core assets in its product development processes, it will have to take over the maintenance for some core asset.

The other flavor of punishment is the opposite of negative reinforcement, i.e. present something displeasing when undesired behavior is exhibited. If a business unit is not using enough core assets in its product releases, upper management can present that business unit a monetary penalty.

3.4 EXTINCTION

Extinction is the process of ignoring undesired behavior. By ignoring the behavior, it might weaken. This approach requires monitoring, since the behavior might not extinct at all. To illustrate the mechanism of extinction, regard the following example. Upper management wants all the business units to adopt the core assets in their product releases as much as possible. The current product development is done in the traditional manner. They could rely on the business units to adopt the core assets. Upper management does not apply any mechanism to influence the adoption of core assets in the product development processes. An argument for applying this method is that the use of core asset should lead to lower development costs and other benefits, which could be regarded as incentives also.

4. IDENTIFYING ISSUES

Regarding the funding model, the following issues need to be addressed. The first users of a core asset deserve compensation for having to deal with possible childhood illnesses. Transitioning from a traditional development process to a software product line should be beneficial for any business unit, e.g. adopting the MIP architecture and the MIP assets should be more beneficial than developing all software components of the modalities from scratch is. This includes both the adoption of core assets and the adoption of new releases of those assets. The final issue that needs to be addressed is the fact that business units have to be compensated for the development of core assets, since this requires more effort than the development of a product-specific asset requires. All these four issues will be related to the three phases in the following.

4.1 COMPENSATE INITIAL USERS

First, regard the compensation of the initial users of a core asset. During the joint asset development phase, the focus should be mainly on compensating initial users of a core asset. By strongly supporting the initial use of core assets, the following benefits, among others, are gained. Initial uses of core assets function as running pilot projects and compensating business units for it shows both support of higher management and provides the core asset developers with feedback regarding their core assets. Specifically the feedback on the usability of the product line architecture is important, since a change to the architecture implies changing all related assets also. In addition, both support from higher management and running pilot projects are success factors for a software reuse program [14] [35] [37] [48].

During the joint asset phase, characterized by a maturing core asset base, the assets have grown more mature and the initial users of new assets or new releases of assets do not need strong compensation anymore.

Especially in the parallel asset development phase, the assets will have matured fully and the initial users of new assets therefore need little compensation. One, however, should be cautious with the initial releases of core assets developed by business units without the interference of the core asset developers. They might lack the experience that the core asset developers have built up during the earlier two phases. If so, the initial users of the corresponding core assets do need to be compensated accordingly.

4.2 ADOPTING CORE ASSETS

The second issue relates to the incentives regarding the adoption of core assets in the product development processes. As mentioned, during the sole asset development phase, few core assets are available for use in product derivation. In addition, the core assets typically are immature in this phase. This immaturity implies that for instance the interfaces of the core assets might change. The impact of having to change interfaces needs to be minimal, so not too much products must be dependent on these interfaces. Therefore, there need not be a strong incentive for adopting core assets during that phase. It seems, however, appropriate to stimulate the use of core assets to some extent, e.g. for initiating pilot projects.

In the joint asset development phase, the assets are growing more mature and it therefore seems sensible to stimulate the adoption of core assets more. Stimulating the adoption of core assets contributes to, among other things, the validation of the use of the core assets in different product contexts. Consequently, the core asset developers receive more feedback on all aspects of the core assets they have developed.

Finally, in the parallel asset development phase, the core asset base has grown mature and stable. It therefore does not seems required that the adoption of using core assets needs to be stimulated. This is reasonable since the use of core assets will have proven itself by then.

4.3 ADOPTING NEW RELEASES

The third issue, stimulating the adoption of new releases of core assets, grows more important when the core assets are used in more products. In the initial phase, typically no new versions of core assets will be released. Therefore, no incentive is required.

During the joint asset development phase, the core asset base is more mature and the core assets are used in different product contexts. As a consequence, the need for newer versions of core assets arises, among others due to changing market demands. The business units need to adopt these newer releases also. This is necessary, because when multiple releases of a core asset are in use, multiple releases of that asset require maintenance also. This leads to a less competitive organization, since the focus shifts from development to maintenance and the leverage concerning the maintenance costs decreases.

During the parallel asset development phase, the development of core assets will speed up and newer versions of core assets are likely to be released also. It thus seems sensible to stimulate the adoption of the latest releases also. It should be noted that when backwards compatibility for core assets is provided, little effort is required to adopt the latest release and mainly benefits are to be gained. The main drawback of providing backwards compatibility is that the core asset developers are restricted with respect to the development of their assets and that quality attributes like performance are likely to suffer. In addition, it introduces many technical difficulties.

4.4 CONTRIBUTE CORE ASSETS

The final issue is related to the contribution of core assets by the business units. Initially, this need not be stimulated. This obviously is due to the fact that the product line architecture and the product line scope are under development, the small base of assets are not yet stable, and the core asset developers first need to standardize their core asset development processes. The core assets need to provide a uniform "user interface", i.e. when one knows how to tailor the first core asset, one approximately knows how to tailor the next core asset. This implies that the type of interfaces and that the provided production plans need to be unified.

However, during the subsequent two phases, the business units need to be involved in the development of core assets. As mentioned, developing a core asset requires more effort than developing a product-specific asset requires. During the joint asset development phase, the business units will be collaborating with the core asset developers. This is necessary, since the business unit need to obtain affinity with the development of core assets and with providing the required variability. Since the core asset developers support the business units, little compensation seems required to stimulate the collaborations.

When the business units are developing core assets on their own, i.e. during the parallel asset development phase, a strong compensation seems appropriate. This obviously is due to the fact that they do not receive any support from the core asset developers. One does need to monitor the core assets developed by the business units. This is because every business unit can claim that an asset is reusable and that the business unit consequently should receive a compensation for it.

4.5 OVERVIEW

The table below provides an overview of the relations between the different development phases and the aforementioned four issues.

Table 8.2: An overview of the funding issues related to the three phases of adopting software product line practices

Development phase	Issue			
	Compensate first user	**Incentives for adopting core assets**	**Incentives for adopting new releases**	**Contribute core asset**
Sole asset	Strong	Weak	None	None
Joint asset	Moderate	Moderate	Weak	Weak
Parallel asset	Weak	Weak	Moderate	Strong

5. FUNDING MODELS

With respect to software product lines, there are many proposed funding models. A variety of these models will be discussed in the following subsections. After each funding model, the type of organization, which best suits the described funding model, is described.

5.1 OVERHEAD/TAX FUNDING MODEL

These funding models are the most simplistic models. The general idea of these models is that a certain variable is multiplied with a certain constant. The difference in these models lies with the variable and constant to be used.

With overhead funding [17] [28], the variable part consists of the budget of a project or business unit and the constant part is a factor. A surcharge for every business unit is calculated by multiplying the projected budget of that business unit with the factor. That factor should be given such a value that all the costs related to the software product line approach can be recovered through the surcharges. The problem with this model is that business units that make use of the core assets extensively, are charged relatively less than business units that do not make use of these assets. The cause of this is that every business unit is accounted for the same percentage and this percentage has no relation with the actual use of the core assets.

The second form of this model is called tax funding [17]. This model uses some variable specific for a business unit and multiplies it with a certain factor. For instance, the income tax model uses the income of a business unit for the variable part. The sum of all incomes and the projected costs of the core asset development can then be used to derive the factor, i.e. the percentage required to break-even with the costs for the core asset development, assuming that these costs have been projected up-front.

Another variant of the tax funding model is the flat tax. With flat tax funding, all business units are required to pay for the same amount, regardless of their turnovers, their incomes or their actual use of core assets.

5.1.1 SUITABILITY

When regarding the overhead funding model, both the constant part and the variable part need to be determined at forehand. When these values are not projected accurately, the business units might be charged too much for the core asset development or they might be charged too less for the core asset development.

A similar reasoning can be given for the income tax model, with the focus being on the incomes rather than on the budgets.

The flat tax model can only be applied in an organization where all the business units have got similar budgets and incomes. This is due to the fact that all business units are charged for the same amount.

All these funding models can only be applied in an organization where all business units make use of the core assets to the same degree. If there exist large deviations with respect to the use of core assets between the business units, the business units relying strongly on the core assets pay relatively less than the business units who do not rely strongly on the assets. One might argue that this is an incentive for business units to adopt core assets in the product development. However, it should be kept in mind that not all business units might be capable of using all the core assets in their products e.g. due to lack of supported functionality or a narrow product line scope.

5.2 PAY-FOR-COMPONENTS MODEL

With respect to investments, two variations of the pay-for-components model [6] [17] [32] can be distinguished. The one model requires an initial investment from higher hand and the other model requires that the investment costs are projected up-front and that these costs are spread across the business units accordingly. Within these two variations, yet another distinction can be made. These four variants of the pay-for-components model will be discussed below.

The first variation of the pay-for-components model requires that the initial investments are up-front covered for by higher hand, i.e. corporate management. This is because the business units are charged for the core assets and corresponding services they make use of and they can thus only be charged after the core assets have been finished and are ready for deployment in a product release. As mentioned, the business units are charged for using core assets. The idea is that by doing so, the costs for future core asset development costs can be covered. Two distinctions with respect to paying for core assets can be distinguished. This distinction is related to how often a core asset may be used, after having paid once for it. The two approaches are that a core asset is bought once and that core asset can be used as often as is desirable without further costs, and that for each time a core asset is used in a product it has to be paid for again. The former obviously requires less overhead administration, but the latter seems more fair in the sense that a business unit making intensive use of a core asset is paying more than a business unit who is using that same asset only once.

The other variation of the funding model prescribes that the investments have to be projected and that the investment costs are spread across all the participating business units. Obviously, not only the investment costs have to be projected, but the usage of the participating business units has to be projected also. This is due to the fact that this is necessary for the distribution of the investment costs. Again, a distinction with respect to the use can be made, i.e. pay once and use as often as desirable or pay once per product.

5.2.1 SUITABILITY

This funding model is most appropriate in organizations where the degree of using core assets deviates strongly between the different business units. This obviously is because the costs of developing certain core assets can be traced directly to the business units making use of these assets. This way, a business unit relying strongly on the core assets is charged more than a business unit relying less on the assets. The problem with this model is that business units have to start paying for a core asset as soon as they start using it. This could function as a financial threshold. A similar line of reasoning holds for newer releases of core assets. This model therefore seems most appropriate in an organization where the use of the available core asset base has proven beneficial.

5.3 ACTIVITY-BASED COSTING MODEL

The activity-based costing model [16] [17] provides a more accurate approach to relate overhead expenses directly to the sold products. As mentioned earlier, the overhead funding model does not distribute the costs of the core asset development well, since both projects with a high level of reuse and projects with a low level of reuse are charged for the same percentage. The overhead costs must be significant and the overhead costs must be poorly accounted by the traditional means, e.g. the overhead funding model. In addition, the overhead costs must consist of repetitive activities. This all is required to be able to perform the required calculations. By analyzing the relations between the products and the indirect costs, the proper shares for these costs are determined and allocated to products accordingly. In [16], a more thorough discussion on how to allocate the costs is provided.

5.3.1 SUITABILITY

This funding model also is most appropriate in organizations where the degree of using core assets deviates strongly between the different business units, for the same reasons as mentioned with the pay-for-components model. An additional requirement is that the core asset development costs need to be traceable to the business units. This typically is not the case. Again, the problem with this model is that business units have to start paying for a core asset as soon as they start using it. This could function as a financial threshold. This model therefore seems most appropriate in an organization where the use of the available core asset base has proven beneficial.

5.4 REWARD-BASED MODEL

The reward-based model [8] is more complicated in comparison with the aforementioned models. It makes use of an additional actor, a reuse manager. The reuse manager is responsible for creating the incentive structure for each reuse transaction. Three reward functions are used, namely:

- Core asset developer reward function, used to compute the reward for the core asset developer each time one of the core assets is used in a product release.
- Product developer reward function, used to compute the reward for the product developer each time a core asset has been integrated into a product.
- Sponsor reward function, used to compute the reward for the sponsor each time one of the core assets, which is sponsored by him, is used.

The reward functions can for instance be linear decreasing. This implies that the first use of a core asset results in higher rewards than the subsequent uses do. The general idea is that the first users are compensated more for the problems that arise with being the first user, e.g. integration and quality-attribute related problems. Also, product developers are rewarded when they adopt core assets in their product development process. The model, however, does not take the evolution of the core assets into account. Neither does it take into account that the core asset development requires a specific mindset, i.e. prioritize the long-term health of the software product line. The traditional mindset is to focus solely on the delivery of products, which for instance could affect the quality of the assets in a negative manner.

5.4.1 SUITABILITY

This funding model is most appropriate in organizations where the development of core assets and the use of these assets need stimulation. There typically is no designated core asset developer. All the business units are expected to contribute core assets to the core asset base. As mentioned above, developing core assets, or maintaining a software product line, requires a certain mindset, i.e. prioritize the long-term health of the software product line above turning out products. This funding model therefore does not seem to be applicable to an organization that is in either the sole asset development phase or the joint asset development phase. This obviously is due to the fact that the mindset needs to be adopted by all the participating business units. The core assets would otherwise be developed too much in a product-specific manner with the key motivator being time-to-market or low development costs rather than quality.

5.5 BARTER MODEL

When using the barter model [6], all the current business units agree on putting effort in the development of reusable components. The amount of effort can for instance be subject to the revenues or profits of the business unit at hand. The main advantage is that there is no additional flow of funds necessary for the development of core assets. However, the different business units need to have a high level of trust into each other, for instance regarding the quality attributes of the core assets. It should be noted that certain core assets, e.g. the product line scope and the product line architecture, need to be maintained centrally. This is because the product line scope and the product line architecture need to be determined and maintained as objective as possible, so there is no preference for certain products.

5.5.1 SUITABILITY

Again, this funding model only seems applicable to an organization in the parallel asset development phase, for the same reasons as with the reward-based model. In addition, the business units in such an organization must have experiences with collaborating, since a high level of trust is imperative.

5.6 OVERVIEW

The table below presents the aforementioned funding models and their relation to the four issues.

Table 8.3: An overview of the funding models and how these models map to the issues

Funding model	Issues			
	Compensate first user	**Incentives for adopting core assets**	**Incentives for adopting new releases**	**Contribute core asset**
Overhead/tax	No	No	No	No
Pay-for-components	No	The contrary	No	No
Activity based costing	No	The contrary	No	No
Reward-based	Yes	Yes	Yes	Yes
Barter	No	No	No	Yes

CHAPTER 9
ENGINEERING WITH REUSE

The previous chapter introduced the notion of funding models. In addition, several funding models were proposed. How several funding models are put into practice is the topic of the following.

This chapter thus discusses the funding models that were used by Philips Medical Systems. The first section provides a context sketch. This seems appropriate, because the experiences of Philips Medical Systems are not typical for all organizations applying a software product line approach. The following section discusses the funding models in a chronological order. A brief discussion on the advantages and disadvantages will be presented accordingly. The third section relates these funding models to the four aforementioned issues regarding funding software reuse. The chapter will be concluded with a summary.

1. CONTEXT SKETCH

Components & Services is developing MIP assets from 1998 up to now. Over time, many problems with respect to the use of these assets have played a large role in the slow adoption of these assets in the product development. A selection of typical problems regarding the use of MIP assets will be presented here briefly. First, the focus of Components & Services has in the beginning been too much on functionality and too little on quality. As a consequence, many problems with respect to quality played a role. For example, the performance of certain MIP assets did not suffice. Also, the memory usage of certain assets was far too high. The fact that the focus has been to much on functionality has been acknowledged by Components & Services and a focus shift towards quality is noticeable.

Another problem regards the documentation. Many business units have complained about the lack of proper documentation. There does circulate documentation describing the functionality of the core assets, but there is no proper guide for using the variability offered by the assets. As mentioned earlier, for instance how to use the architectural solution, how to tailor the MIP assets, and how the MIP assets fit in the architectural solution is not described. Examples of desired documentation comprise reference guides, i.e. examples describing how an example product was derived using a set of core assets, or clear descriptions of how a collection of core assets can be tailored for use in a specific product, i.e. a production plan.

It should be stated that Components & Services and their MIP assets have matured much since 1998. An illustrative example is the following. One of the goals of Philips Medical Systems is, as mentioned earlier, to achieve UI harmonization, i.e. one look and feel for the user interfaces of all their products. The initial assets providing the GUI displayed for instance all buttons and icons for all possible modalities. It was not taken into consideration that certain buttons were specific for a MR scanner and were for instance never used in a PET scanner. The latest versions of the assets providing the GUI could be tailored with respect to the context in which it is used. This way, the redundant buttons and icons were discarded and only the functional buttons and icons were visible.

Even though Components & Services and their MIP assets have matured much since 1998, still the business units are reluctant to adopt a new released core asset in their product development. This is due to, among other things, the problems sketched above.

2. OVERVIEW OF MIP FUNDING MODELS

Philips Medical Systems has initiated the MIP project in 1998 and several funding models were applied. This section presents all these funding models. In addition, the experienced advantages and disadvantages of these funding models will be discussed briefly. The funding models described below were, as stated above, applied in a situation were the business units were somewhat reluctant to adopt the core assets and there existed little trust among the business units towards the quality of the core assets.

2.1 PAY-FOR-COMPONENTS MODEL

The first year, 1998, the development of MIP assets was funded according to a pay-for-components model. The business units X-Ray and MR were the only two users of the MIP assets. They therefore had to cover for all the costs related to the development of the MIP assets. In addition, they had to pay up-front. Since only two business units participated, one of the biggest worries related to the use of this funding model was that the MIP assets would not be built in a generic enough manner, i.e. the product-line scope would narrow down to only incorporate the products developed by MR and X-Ray. This worry was enforced by the fact that the two business units funding the core asset development could use the financial dependency of C&S as a means to influence the decision making process of C&S.

2.2 TOP-DOWN FUNDING

The second and third year, 1999 and 2000, the MIP project was fully funded by corporate management. The greatest benefit of this approach is that there is no financial threshold to adopt MIP assets. Hence, the non-participating business units could start participating free. Another benefit of top-down funding is that there is no need for an additional cash flow. However, due to the immaturity of the MIP assets, the assets were not adopted en masse by the business units. An additional disadvantage of this approach is that it does not relate the costs of developing and maintaining the core assets to the business units and the products they sell. This way, if the development and maintenance costs of core assets far exceed the benefits of the usage of these assets at corporate-level, this would be less noticeable. This is because the relation between investments in core assets and the use of these assets in products is vaguer. Business units could at most compare the integration costs of the MIP assets with the costs it would have required to develop a similar product-specific asset from scratch. If the business units have to pay for the core assets themselves, they will be paying more attention to the return on investments and, consequently, the costs of developing the MIP assets would become more interesting.

The fourth year, MIP was funded entirely by one business unit, namely MIT. In order to compensate MIT for this, the financial targets of MIT have been reduced by the same amount as was required for the funding of MIP. This can thus be seen as a variant on the model used in the second and third year, where the same reasoning concerning the benefits and drawbacks holds.

This type of funding has not been included in the overview of the funding models. This is due to the fact that it is not a real funding model, i.e. all the required funds are directly provided by corporate management and, therefore, there is no prescribing of costs to products or business units.

2.3 PAY-FOR-SUITE MODEL

From then on, a variant on the pay-for-components model was used. This pay-for-components model prescribes up-front payment for using core assets, regardless how often these core assets are used. These up-front payments are paid on a yearly base. The general idea is as follows. First, an estimate of the necessary funds for the following year is made. Then, the related costs are spread across all the participating business units, taking their use of the core assets into account also. In 2002, only X-Ray, MR, MIT, and US were participants. In the past two years, 2003 and 2004, all the business units started adopting core assets and, therefore, the funding was distributed across all of the business units.

There, however, exist differences with the pay-for-components model discussed in the previous chapter. This version makes use of yet another distinction, namely in the type of use. If a business unit plans to make use of MIP assets in a product release, they will be charged relatively more than a business unit who only uses MIP assets in a prototyping or feasibility project.

In addition, instead of paying for components, the business units have to pay for suites of components. The five main segments of MIP assets, i.e. base, connectivity, database, (system) services, and viewing, are used to account the business units for their use. In order to make the distinction between this model and the other pay-for-components model clear, this model will be referred to as the pay-for-suite model.

The main advantage of this model is that it links the core asset development costs to the business units in a transparent manner. This transparency is achieved by using the five segments instead of loose assets. The main drawback of this funding model is that the business units have to start paying for a MIP segment as soon as they want to work with it, i.e. it exhibits a financial threshold. This can thus be regarded as a disincentive to adopt core assets in the product development processes, since not all the segments have matured enough. Given the context sketched above, it is expected that the initial integration costs more than it returns.

3. BRIEF OVERVIEW

This section provides an overview of how the presented funding models relate to the aforementioned four issues that can be addressed with a funding model. This has not yet been addressed in the above. None of these funding models has a mechanism to compensate the first user of a core asset. Also, none of the models has an incentive mechanism regarding the adoption of more and newer core assets, or an incentive to stimulate the development of core assets by business units other than Components & Services. The latter, however, is not yet a real problem for Philips Medical Systems, since the business units have not matured with respect to core asset development, i.e. they have not yet adopted the core asset developer mindset. This is reflected in the fact that Philips Medical Systems is in the transitioning from the exclusive asset development phase to the joint asset development phase.

The most recent projects consist of collaborations between business units and Components & Services and, therefore, the time seems right to extend their funding model with a corresponding incentive. The big drawback of the most recent funding model is that there is a disincentive to adopt core assets in the product release. As soon as a business unit adopts a segment, they will be charged from that point on for using that segment. This can only work when the use of core assets has a reasonable return on investment, which has not always been the case at Philips Medical Systems. The following table presents the relation between the four issues and the used funding models.

Table 9.1: An overview of the funding models and how these models map to the issues

Funding model	Issues			
	Compensate first user	**Incentives for adopting core assets**	**Incentives for adopting new releases**	**Contribute core asset**
Pay-for-components	No	The contrary	No	No
Top-down	No	No	No	No
Pay-for-suite	No	The contrary	No	No

4. SUMMARY

Philips Medical Systems has used multiple funding models in the past, i.e. a pay-for-components funding model, a top-down funding model, and a pay-for-suite funding model. None of these models addresses the four aforementioned issues: compensate first users of core assets, stimulate the adoption of core assets, stimulate the adoption of new releases of core assets, and stimulate the development of core assets by business units.

CHAPTER 10
ADJUSTABLE INCENTIVE-DRIVEN MODEL

The current collection of funding models that are proposed in the literature do not seem to be applicable to software product line practices. Therefore, this chapter proposes a model that takes the four identified issues into account and assumes the existence of a core asset developer. The latter is necessary, since the product line scope and the product line architecture need to be maintained by a centralized organizational entity.

The first section recapitulates the first research question. The second section is concerned with the process of deriving the adjustable incentive-driven model. After that, the solution is presented. The fourth section explains how the adjustable incentive-driven model should be utilized. Finally, the chapter is concluded with a summary.

1. RESEARCH QUESTION

As mentioned in chapter 2, one of the objectives of this thesis is to propose a novel funding model that aids in the success of the institutionalizing of a software product line effort. The first main research question therefore was: How can a funding model contribute to the success of a software product line program?

In chapter 8, multiple funding models were discussed, including the characteristics they exhibit. In addition, four different issues have been identified, i.e. compensate first users, stimulate adoption of core assets, stimulate adoption of new releases of core assets, and stimulate contribution of core assets by product developers. In order to determine how behavior of business units can be altered in order to deal with the four aforementioned issues, different types of rewards have been identified, ranging from monetary rewards to additional support form the core asset developers during product derivation. Regarding the funding model, however, only monetary rewards are taken into consideration. How the above leads to solving the main research question and, consequently, how the research objective is achieved, is depicted by the following figure.

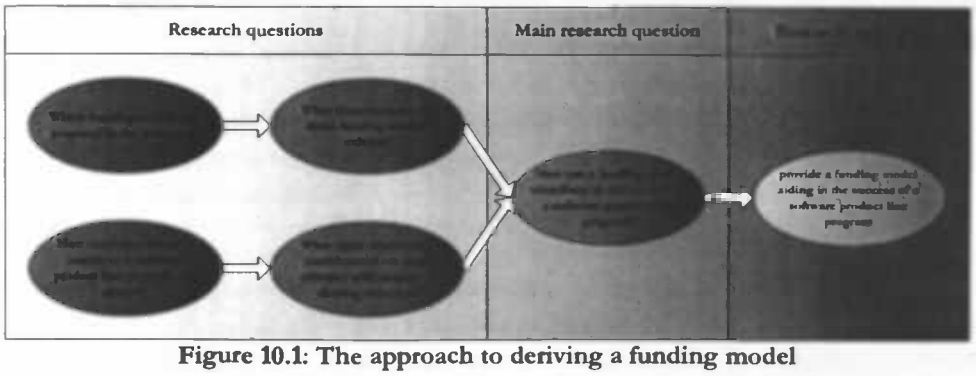


Figure 10.1: The approach to deriving a funding model

2. ANALYZING ISSUES

This section is concerned with the process of deriving a funding model that aids in the success of a software product line program.

2.1 PHILIPS MEDICAL SYSTEMS SOFTWARE SUMMIT 2005

The Philips Medical Systems Software Summit is a yearly event. All business units concerned with software development are invited to send at least one representative. The latest trends on technologies related to software development are topics of discussion. The author of this thesis was invited to hold a presentation on funding models for software product lines from a theoretic perspective. The presentation was purely informative and illustrated the pros and the cons of all the aforementioned funding models. After the presentation, all the participants were invited to fill out a questionnaire. They were asked, among other

things, which funding model they preferred and why this is so. The inputs from these business unit managers aided in the derivation of the funding model.

All the persons involved in filling out the questionnaire had different backgrounds, ranging from business unit managers in Cleveland, U.S. to business unit managers in Best, the Netherlands to business unit managers in Bangalore, India. The results of the questionnaire, however, were less diverse. Two funding models were by far the most popular, namely the tax model and the reward-based model. A brief recapitulation on these funding models is presented below. For a more thorough discussion on these models, refer to chapter 8.

- **Tax model:** This funding model requires a certain percentage of some variable. A popular tax model was the flat tax model. This model prescribes that all business units contribute the same amount.
- **Reward-based model:** This funding model makes use of incentives. The typical rewards stimulate both the development of core assets and the use of these assets in actual products. In addition, a reward is presented to the reuse sponsor, i.e. a person arranging that a product developer is able to use a core asset. The rewards related to a specific asset degrade when it is used more often.

Based on the above, it seems that the reward-based model is an adequate funding model concerning the institutionalization of a software product line effort. However, as mentioned earlier, this funding model does not apply well to organizations that have a designated core asset developer. In such an organization, the other business units typically are only concerned with turning out products and not with developing high-quality core assets. In order to achieve an organization where business units are capable of developing high-quality core assets, a funding model should exhibit yet another property. If a core asset base, including a proper product line scope and product line architecture, have been realized by a central organizational entity, Components & Services in the case of Philips Medical Systems, other business units can start using the architectural solution and the corresponding core assets. In addition, the business units can collaborate with the core asset developer. Such collaborations leads to multiple benefits. First, the core asset developers experience from first-hand how well their core assets behave in actual product contexts. Second, since engineers from other business units are contributing to the development of core assets also, the core asset developers increase their development capacity. Third, due to the collaborations, the business units learn how core assets are constructed and they learn more rapidly how to use future assets. Finally, the business units learn how they can develop core assets individually. All of these benefits were observed at Philips Medical Systems, e.g. with the collaboration between PET and Components & Services. A funding model should therefore stimulate collaborations among core asset developers and product developers also, since this is required for the development of high-quality assets.

At the Software Summit, additional concerns were expressed also. The main problem some business units have with adopting core assets is that the use of these assets does not lead to the typical software reuse promises, i.e. lowered development costs, lowered development costs, shortened time-to-market, and increased quality. As stated earlier, this is due to problems with, among other things, the provided documentation regarding both the explanation and use of the architectural solution and corresponding core assets, and the quality attributes of the core assets, e.g. because the assets are used in other contexts than which they have been validated for. Therefore, they suggested incorporating the quality of all aspects of the core assets into a funding model, varying from performance of the assets to the quality of the provided documentation. A method for doing so is to cut the budgets of the core asset developers when the quality of the assets does not suffice. This, however, could lead to a vicious circle rather than increased quality. As mentioned above, collaborations between the business units and the core asset developer lead to more feedback during core asset development and more understanding of the architectural solution and how the core assets fit in that solution accordingly, higher-quality core assets and improved integration efficiency of these assets respectively. Consequently, stimulating collaborations should enable higher-quality core assets without a further need for rewards or punishments.

2.2 HYBRID MODEL

The funding model should have the following properties. First, it should address the issue that the initial user of a core asset typically has to deal with the childhood illnesses of a core asset. It is expected that a core asset grows more robust and mature over time. It therefore seems sensible to degrade the compensation of using a core asset when it is used more often.

Second, the adoption of core assets needs to be stimulated. Organizations taking on a software product line approach to product development typically are burdened with legacy products, as is the case with Philips Medical Systems. In order to adopt the core assets in the products, an incremental, evolutionary approach proves the most valuable approach. This requires certain additional activities, e.g. transforming the legacy architecture towards a derivation of the software product line architecture, wrap core assets in order to make these assets fit in the legacy architecture, and provide additional support regarding the integration and use of the core assets in other contexts than they have been developed for. The first two activities are accounted to the product developers and providing additional support has to be taken care of by the core asset developers.

Third, when product developers use the core assets, different problems concerning these core assets surface. At Philips Medical Systems, such problems vary from too high memory utilization to poor performance to insufficient usability. In addition, due to market changes and a correspondingly evolving product line scope, new requirements will have to be captured by the core asset base. Consequently, the core asset developer has to develop new releases of the core assets. The business units will then have to adopt these newer releases. If not, the core asset developer has to maintain both the new release and the old release. If multiple releases are out and in use by different business units also, this leads to a focus shift from development towards maintenance due to the fact that all releases require maintenance. The adoption of new releases should therefore be stimulated so that the number of different releases that require maintenance is minimized.

Fourth, the business units need to develop core assets also. As mentioned, when an organization assigns a core asset developer, its capacity forms a bottleneck with respect to core asset development. Their capacity can be augmented by stimulating both collaborations between business units and the core asset developer, and let business units develop core assets individually. However, business units typically are driven by product development and do not consider the long-term health of the software product line. These business units need to be convinced that the long-term health of the software product line is a prerequisite for keeping business healthy on the long-term. By stimulating such collaborations, business units learn to develop core assets on their own and along the way begin to comprehend that the long-term health of the software product line is required to keep business healthy for not only the subject business unit, but for the organization as a whole also.

Finally, the core assets need to adhere to all non-functional requirements of all possible products that are included in the product line scope. At Philips Medical Systems, the core asset developers initially developed and tested all core assets without involving any other business unit. Because of this, the products that have to cope with a legacy architecture, cannot always make use of certain core assets. This, however, was not clear beforehand, since the quality attributes that the core assets exhibit, were not documented.

In conclusion, there exists no funding model that supports both rewarding certain behavior and an incremental, evolutionary approach to adopting a software product line. The funding model should thus provide compensation for users of core assets, but the compensation needs to degrade after each time an asset is used. Second, the adoption of core assets needs to be stimulated. In addition, the adoption of new releases of core assets needs to be stimulated also. Fourth, having business units develop core assets rather than specific assets needs to be stimulated also. It should, however, be kept in mind that the business units first need to adopt the mindset of a core asset developer. As a consequence, the collaborations between business units and the core asset developer need to be stimulated first.

3. ADJUSTABLE INCENTIVE-DRIVEN MODEL

A flat tax model prescribes that all business units are accounted for the same amount. This proves to be a fair starting point for assigning rewards. The general idea of the funding model is as follows. If all business units exhibit the same behavior regarding all four issues, then all are accounted for the same amount, regardless of their usage of assets. This reflects the principles of the flat tax model. But, the rewards start playing a role when business units exhibit different behaviors. In order to effectuate the rewards, the total costs of core asset development are spread across all business units taking the behaviors of all these busi-

ness units concerning each of the four issues into consideration. The following presents the basics of the novel funding model and it is augmented stepwise to incorporate all the above-mentioned requirements.

First, consider the case that an organization consists of two business units, referred to as business unit A and business unit B, and only the issue regarding the adoption of core assets needs to be addressed. If both have adopted a similar number of core assets in their product development processes, both will be accounted for the same amount. However, when business unit A adopts more core assets than business unit B does, business unit A has to be rewarded for that. In order to do so, the funding model requires that the behaviors are quantified, so that the reward can be determined. One could for instance use the number of adopted core assets as the determinant for the reward. Consider that business unit A has adopted ten core assets, business unit B has adopted five core assets, and a total of twelve core assets that are applicable to both business units are available. Business unit A is then awarded a score of twelve minus ten and business unit B is awarded a score of twelve minus five. The total costs of core asset development then is divided by the sum of the two scores. Finally, the result from this division is multiplied with the individual scores of each business unit and the costs for which they are accounted are determined accordingly. By doing so, business unit A obviously is accounted for less than business unit B.

However, the funding model needs to incorporate the other three issues also. If a similar method for quantifying the behavior as above is applied to these other three issues, a business unit is awarded a total of four scores. Then, the sum of the four individual scores is the value that is used for deriving the costs for which that business unit is accounted in a similar fashion as above. So, all issues are then taken into consideration.

As mentioned in chapter 7, a transition of an organization from a traditional approach to a software product line approach can be divided into three overlapping phases, i.e. sole asset development, joint asset development, and parallel asset development. In chapter 8, it was identified that at each of these phases the issues play a role to different extents. For instance, during the sole asset development phase, the focus should be on compensating the initial users of a core asset, while in the parallel asset development phase, the focus should be on stimulating the development of core assets by business units other than the core asset developer. In addition to this, the issues play a role to different extents at different organizations. If, for instance, the core assets of some organization are of high quality to begin with, the initial users of these assets need little to no compensation for being initial users. The fact that they are lowering the development costs, lowering the maintenance costs, shortening the time-to-market, and increasing the overall quality of their products, are incentives on their own. It, therefore, seems imperative to include a certain weight to the scores per issue also. By doing so, an organization can manipulate the scores of business units more accurately. By incorporating the weights regarding the four issues, the funding model now is applicable to all three phases and can be adjusted for the specific context in which it is used.

A structured approach to determining the costs for which business units are accounted, incorporating all the above, is presented in the following subsections. This prescription of determining the costs each business unit is accounted for, is referred to as the Adjustable Incentive-Driven model or simply the aid model.

3.1 STEPWISE DETERMINING

Based on the above, the following six steps, which determine the costs that each business unit is accounted for, are abstracted. All of these steps are discussed in more detail in the following subsections.

Step 1 Assign values to the behavior constants.
Step 2 Assign values to the behavior variables.
Step 3 Calculate the individual score for each business unit.
Step 4 Calculate the overall score.
Step 5 Derive the value for the cost factor.
Step 6 Use the software product line costs to calculate the individual costs.

3.2 STEP 1

First, the different behaviors should be assigned a quantitative value, which is realized through the behavior constants ω_i. A behavior constant is a constant, which indicates to what extent the subject behavior is to be influenced through the funding model. The most important behavior should be awarded the highest value. For instance, in the sole asset development phase, the focus should be on compensating initial users of core assets and, therefore, the corresponding behavior constant should be awarded the highest value.

$$\omega = \begin{bmatrix} \omega_1 = v_1 \\ \omega_2 = v_2 \\ \omega_3 = v_3 \\ \omega_4 = v_4 \end{bmatrix}$$

Equation 10.1: Assign values to behavior constants

3.3 STEP 2

The second step is concerned with quantifying the exhibited behavior of the different business units using the behavior variables. A behavior variable specifies a quantitative value for a particular business unit and its corresponding behavior. The behavior variables will have to be assigned values for each specific business unit. Examples of how the behaviors regarding each of the issues can be quantified are presented in section 4. The lower value one scores for the behavior variable, the more beneficial it is. For instance, in the parallel asset development phase, the focus should be on contributing core assets by business units other than the core asset developer and, therefore, the business unit contributing many core assets should be assigned a low value. Conversely, a business unit that does not contribute core assets should be assigned a high value.

$$i = \begin{bmatrix} i_1 = v_1 \\ i_2 = v_2 \\ i_3 = v_3 \\ i_4 = v_4 \end{bmatrix}$$

Equation 10.2: Assign values to behavior variables

3.4 STEP 3

At this point, both the behavior constants and the behavior variables for all the business units have been evaluated. The scores for each individual business unit can be calculated using the following equation.

$$Score_i = \sum_{j=1}^{n} \omega_j \cdot i_j$$

Equation 10.3: Calculation of the score of a business unit

3.5 STEP 4

This step is concerned with calculating the overall score of the organization, i.e. the sum of the scores of all the business units. It can be derived with the following equation.

$$Overall\ score = \sum_{i=1}^{n} Score_i$$

Equation 10.4: Calculation of the overall score

3.6 STEP 5

The fifth step prescribes the calculation of the cost factor. The cost factor is the value, which is multiplied with an individual score to determine the actual required funds. The following equation prescribes how the cost factor can be calculated.

$$Cost\ factor = \frac{Costs}{Overall\ score}$$

Equation 10.5: The calculation of the cost factor

3.7 STEP 6

The final step comprises calculating the funds that each business unit is required to pay. It uses the total score of a business unit to determine how much a business unit is required to pay. The idea is that if one has a higher score, one has to pay a larger share of the costs also. The following equation determines the required amount of funds.

$$Required\ funds_i = Cost\ factor \cdot Score_i$$

Equation 10.6: The actual calculation of the required funds

4. ASSIGNING VALUES

This section presents how the four issues can be assigned values for the behavior variables. These methods are purely to illustrate that it is possible to quantify all types of behavior. Other methods for quantifying the behaviors might be more appropriate for certain organizations.

4.1 COMPENSATE FIRST USER

The first behavior variable is concerned with assigning values to the behavior related to the initial uses of a core asset. The reward-based model prescribes that each time a core asset is used in a different product context, the corresponding reward decreases. This seems to be a good approach to use for the aid model also, since it is expected that every time a core asset is used in a product context, less integration problems arise due to maturing assets. The simplest approach is to define a reference table. An example of such a model is presented below.

Table 10.1: Example reference table for compensation of first user

How often used before	Awarded behavior value
0	1
1	2
2	4
...	...
n	*max*

Such a table specifies what value is awarded to a certain core asset, depending on how often a core asset has been used already. It should be monitored for every asset how often is has been used already, so it can easily be referenced what the corresponding value is for the behavior variables. If one is the initial user of a core asset, one is awarded the value one. The second user is awarded the behavior value two, the third is awarded the value four, and so forth.

A more complicated variant on this approach is to set up a reference table for each core asset, with values specific for that core asset. By doing so, initial uses of problematic core assets can be compensated more than initial uses of robust core assets.

An additional solution to deal with the initial integration problems is to provide additional support during the initial integration. The core asset developer should then not only function as developer, but also as a supporting member of the product developer.

4.2 ADOPT CORE ASSETS

Quantifying the behavior with respect to the adoption of core assets can be done by taking the reuse level into account. The reuse level specifies how much of a product comes from reuse. For instance, if a product has a reuse level of 40 per cent, this implies that 40 per cent of that product is derived from core assets, e.g. an architectural solution and reusable components. In order to determine quantitative values for the behavior variables, a certain value, referred to as the base value, can be divided by the reuse level. This way, the higher reuse level one has, the lower value one is awarded for the corresponding behavior variable.

4.3 ADOPT NEW RELEASES

The adoption of new releases can be quantified in a similar fashion as quantifying the behavior related to compensating the first user. Again, a reference table can be used. This time, the reference table is not taking specific assets into consideration, but is concerned with assigning values to certain releases. For instance, consider that three core asset bases have been released. The following table then assigns values to all three releases.

Table 10.2: Example reference table for adoption of new releases

Core asset base release	Awarded behavior value
3.0	1
2.0	3
1.0	10

The reference table is used in a similar fashion as the above mentioned reference table. There are additional methods to stimulate the adoption of core assets, besides making the use of old releases more expensive. For instance, when the maintenance for an old release simply is halted, the business units will have to adopt a newer release. This approach, however, brings along many organizational difficulties, e.g. acceptance within organization and problems regarding already released products.

An additional manner is to provide backwards compatibility for the core asset base releases. Even though this implies that the core asset developers are restricted in a technical manner, e.g. due to interfaces restrictions, the core asset developers can declare certain parts of the interfaces as deprecated, but still provide support for those interface for a while. By doing so, the product developers have time to make the transition towards the new release, but still can lean on the implementation at hand. At Philips Medical Systems, for instance, some business units are reluctant to adopt new releases of the core asset base because they expect additional integration problems.

To put it simply, the best method to persuade the product developers to adopt the newest releases is to ensure that the costs of integrating the newest release are lower than the benefits gained from that release. For instance, providing backwards compatibility, additional integration support, and sufficing documentations are manners to do so.

4.4 STIMULATE CONTRIBUTION OF CORE ASSETS

Finally, the contribution of core assets by business units other than the core asset developer needs to be stimulated. As mentioned, developing a core asset requires more effort than developing a product-specific asset does. This is, among others, due to the required variability that a core asset must offer. Two types of contributions can be identified, i.e. collaborations between product developers and core asset developers for the development of core assets, and product developers developing core assets individually.

First, consider the collaborations between the product developers and the core asset developers, i.e. the joint asset development phase. Many benefits from this manner of working are to be gained, e.g. experience in developing core assets, experience in usability of core assets, augmentation of core asset development capacity, and increasing trust in core assets. A method to quantify the behavior for business units that are collaborating in core asset development is the following. The method is similar to using the reuse level. If one divides the number of employees of a business unit that are working on the development of core assets and divide it with the total number of employees of that business unit, the percentage working on core assets per business unit is obtained. This percentage can then be used in a similar fashion as the reuse level. It should, however, be noted that during the joint asset development phase the business units are supported by the core asset developers. It might be necessary to incorporate the support of the core

asset developers also. This is especially necessary when, with respect to the core-asset-developer – product-developer ratios of these collaborations, large deviations exist.

Second, the business units can develop and contribute core assets individually. This way, the core asset development capacity is augmented even more and the community feeling within the organization is enforced. The same method as for the joint asset development can be applied, but there is no need to incorporate the support of the core asset developers, since the business units are developing all core assets on their own.

Other arguments for developing core assets rather than product-specific assets exist also. First, future maintenance for core assets that are developed by product developers is taken care of by the core asset developer. Second, a requirement that the core asset developers might not be able to develop due to lack of capacity, now can be developed because the core asset developers are supported with engineers from other business units.

5. SUMMARY

This chapter introduced the adjustable incentive-driven model. This model assigns both a weight to all relevant types of behavior and assigns an actual value to all the business units for their exhibited behavior. The former are referred to as behavior constants and the latter as behavior variables. Making use of these values, the funds required for each specific business unit can be calculated easily. The issues that are addressed by the aid funding model comprise compensating the initial users of core assets, stimulate the adoption of core assets, stimulate the adoption of new releases of core assets, and stimulate the development of core assets by business units other than core asset developers. In addition, the funding model is applicable to an organization that has an entity that is assigned as a core asset developer.

CHAPTER 11
APPLICABILITY OF THE AID MODEL

As different organizations each exhibit their distinctive characteristics, they typically exhibit commonalities also. The previous chapter introduced the aid model. This model addresses four issues that are typical for any organization that takes a software product line approach to product development. However, it provides a certain degree of variability also. This variability is provides as a means to tailor the model to make it applicable for the subject organization.

This chapter presents the validation of the aid funding model, which was introduced in the previous chapter. First, an introduction to the validation approach is provided. Then, a brief overview of both the used questionnaire and the corresponding conclusions is provided. After that, the conclusions from the questionnaire are mapped on the aid model. The fourth section discusses the pros and cons of the model and the fifth model explicates the contribution. After having provided future directions, a conclusion is provided.

1. VALIDATION APPROACH

The general opinions regarding funding models were initially polled during the Philips Medical Systems Software Summit. Based on the results from that questionnaire and corresponding discussion, it was concluded that a flat tax model combined with a reward-based model seemed most appropriate in that context, i.e. an organization where the usage of core assets has not yet been proven to be beneficial in all product contexts. Based on these observations, a novel funding model was derived.

The validation of the derived model, i.e. the aid funding model, is done through the use of a questionnaire. This questionnaire is provided in appendix B. It should be noted that the purpose of sending out the questionnaire is twofold. As mentioned in chapter 1, this thesis forms one of the deliverables of the graduation assignment. Another deliverable consists of a consultancy report. Several questions from the questionnaire are related to that consultancy report and not to this thesis. The results, however, do provide more insight in the context in which the interviewees operate. Where appropriate, the additional observations will be provided also.

Besides the aforementioned two methods, the opinions of individuals were polled during interviews. The purpose of these interviews varied over time. Initially, the interviews were used to gain more insight in the context, i.e. the organization Philips Medical Systems and how it operates internally. After that, the interviews were used to generate ideas and solutions for the perceived problems. Finally, these ideas and solutions were used as input for the interviews with the purpose of validating the applicability in the subject context and, more importantly, validating that this indeed provides a solution to the perceived problems. The validation approach is graphically depicted by the figure below.

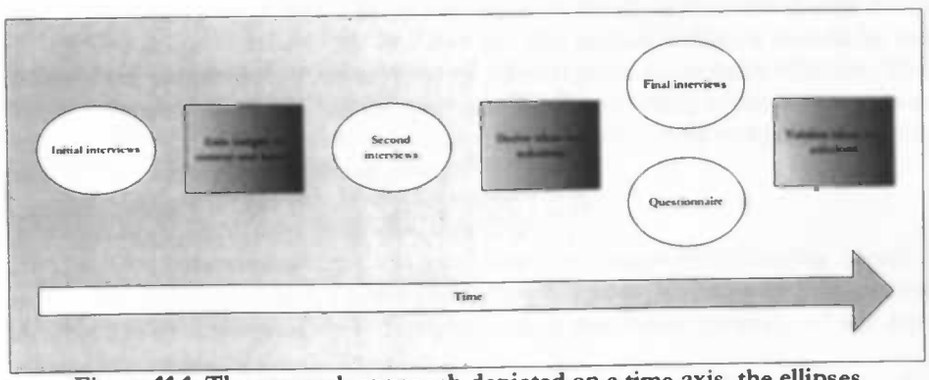


Figure 11.1: The research approach depicted on a time axis, the ellipses represent activities and the rectangles represent results

1.1 POPULATION

The population to whom the questionnaire was sent out consisted mainly of business unit managers from Philips Medical Systems, both operating in core asset development and in product development. These business unit managers are scattered across multiple sites, i.e. sites in the Netherlands, the United States, Germany, and India. In addition, people from the Corporate Technology Office and Philips Research were invited to fill in the questionnaire also. Unfortunately, not all persons responded to the questionnaire.

The interviews were conducted with both core asset developers and product developers. The main part of the interviews were conducted in Best, the Netherlands. Also, telephone conferences were used to interview business unit managers from abroad, i.e. managers from Hamburg, Germany and Cleveland, United States

Finally, experts from the field were contacted also. The Software Engineering Institute provided suggestions via e-mails. In addition, professors from the University of Groningen, faculty Business, provided guidance regarding funding in general, behavior in organizations, and manipulating behavior in organizations.

1.2 CONTEXT

A more detailed discussion on Philips Medical Systems and their approach to software product lines is provided in chapter 7. This subsection merely provides a sketch of properties of Philips Medical Systems relevant for the validation of the solution.

Philips Medical Systems is a large organization with sites across the world. The core asset developer, i.e. Components & Services, is located in both Best, the Netherlands and Bangalore, India. Since many product developers are located elsewhere, communication is an issue. As a result, the lack of proper documentation becomes an even bigger issue. As stated in the interviews and the questionnaire, the key drivers for business units of Philips Medical Systems for adoption of MIP assets is first to shorten the time-to-market and second to reduce the development costs. Momentarily, many business units do not rely on the quality of the MIP assets. The expected benefits, i.e. the typical reuse benefits, are not realized by all the business units and, therefore, the business units are somewhat reluctant to adopt the MIP assets. Besides the lack of trust, other issues play a role at Philips Medical Systems also. For instance, newer releases of core assets are not backwards compatible. This implies that when a business unit has had bad experiences with integrating a core asset in a product, that business unit will obviously be less willing to integrate a newer version of that core asset, since they expect to experience many difficulties regarding the integration again. Another issue regards the interoperability program. As mentioned, the DICOM standard is used to ensure interoperability between different medical appliances, even from different vendors. In practice, it turns out that not all vendors adhere strictly to the standard. The MIP assets, however, for instance do not accept data which does not completely adhere to the DICOM standard. The data is rejected. This has led to many interoperability problems between Philips Medical Systems products and other appliances.

As described in chapter 7, Philips Medical Systems takes an incremental approach regarding the adoption of the software product line initiative. Since many business units have to deal with legacy assets, software architectures among others, this is necessary. A typical modality namely needs to be maintained for about a decade after the final deployment. Due to this, many business units are working with their own legacy software architecture instead of a derivation of the product line architecture. This led for instance to the aforementioned performance issues due to the fact that the actual context differs from the presumed context. An example of a technical problem is the following. The use of interfaces in the MIP assets is somewhat ambiguous and it is not always described how certain calls are handled. This for instance led to problems regarding the use of database queries. For instance, it was not clear whether a certain call to the database returned a processed set of data or just raw data.

Other business units are more enthusiastic about the use of the core assets. Multiple arguments have been given. The use of core assets for instance allowed them to provide functionality in their products which would otherwise not be possible due to lack of capacity. Another experienced benefit regards the UI Harmonization. As mentioned, the program UI Harmonization aims at unifying the user interfaces of all the products from Philips Medical Systems. The use of MIP assets in product development aids in unifying the graphical user interface. The next step, the PMW, provides a ready-to-use workstation which can be augmented with clinical applications and acquisition control, which both are specific per product.

The user interfaces of all PMW implementations therefore will have similar user interfaces. The only difference between these interfaces will be the variability in the offered buttons and icons, which are specific for each modality. Finally, since Components & Services is collaborating more with business units and is experiencing at first hand how the MIP assets behave in a product context, e.g. the collaboration with PET, they have acknowledged that the aforementioned issues indeed need to be resolved. They are currently improving their core asset base accordingly. Another benefit of these collaborations is that the product developers involved in the collaboration gain more insight in the use of the MIP assets also, which was underlined by PET. The involved engineers feel that they now understand the MIP architecture, the construction of MIP assets, and the use of MIP assets better. In addition, they feel they are capable of developing MIP assets without the support of Components & Services.

In conclusion, the MIP assets are growing more mature, due to, among others, the increasing collaborations between Components & Services and the business units. There, however, still does exist a lack of trust regarding the MIP assets and this trust will have to be earned again. Since the typical incentives for using the core assets, i.e. the typical reuse benefits, do not hold for all involved business units, for instance a monetary reward seems in place for dealing with the issues. From many interviews, it became clear that there is a long learning curve regarding the use of MIP assets.

1.3 ABSTRACTION FROM CONTEXT

The research was conducted in a large organization, which is structured as a domain-engineering-unit model. The organization is adopting the software product line approach to product development and it is taking an incremental approach in doing so. The organization can roughly be divided into two parties, i.e. a party that believes the reuse program is successful and a party that believes it is not. With respect to the results from the questionnaire and the interviews, the perspectives of both parties have been taken into account. The core assets have not yet been proven properly applicable to all product contexts within the product line scope. This is due to lack of proper documentation, poor usability, and poor quality attributes among other. It can be stated that simply the availability of the core assets and the expected benefits gained from the usage of these assets do not function as incentives on its own to start adopting the assets. Artificial incentives seem to be appropriate to initiate the adoption of the core assets in the product development processes.

2. QUESTIONNAIRE CONCLUSION

This section is concerned with both the derivation and the presentation of the conclusions from the questionnaire. First, the questionnaire will be discussed briefly. After that, the results from the questionnaire are presented. Finally, a conclusion will be derived from these results.

2.1 SEGMENTATION

The questionnaire can mainly be divided into four segments. Each of these segments addresses a specific subject. The first segment concerns the current MIP funding model. The questions are mainly used to discover which properties of that model were perceived as positive and which were not. Opinions on variations of that model were asked also. The second segment mainly is concerned with different types of behavior. The interviewees were asked whether they felt that certain behavior should be rewarded. The third segment then aims at opinions on what types of rewards seem to be most effective. This segment is used mainly for the consultancy report, but the questions on the monetary rewards or compensations hold for the validation of the aid funding model also. The final segment is only to be used for the consultancy report and it addresses the adoption of newer assets.

2.2 RESULTS

The first segment of the questionnaire is concerned with the current MIP funding model. The general opinion is that the MIP funding model, i.e. the pay-per-suite model, is perceived as a reasonably fair and transparent funding model. Transparency is thought of as a mandatory property of any funding model. Fairness is a desired property also.

Based on the results from the second and third segment, the initial perceived issues regarding funding, i.e. the issues concerning the first user of a core asset, the incentives for adopting more and newer core assets and, the incentive for business units to (co-)develop core assets, indeed require resolution. A funding model can contribute in doing so. There, however, can be identified two parties regarding whether or not a monetary reward will contribute to the desired goals.

The party that believes that a monetary reward will not aid in the success of the software product line initiative believes this for the following reasons. First, they believe that the use of core assets should be beneficial on its own. This implies that the core assets should be easy-to-use and the documentation should be accessible for all business units. Suggestions for improving the core assets and the documentations are for instance to make the core assets less generic and include example implementations in the documentations, i.e. reference production plans. A related issue is that they believe that funding leads to mediocre assets since there is no need for the core asset developers to excel. It is for instance stated that software reuse must be enforced or earned. A manner for it be earned is by developing high quality core assets, which prove to be beneficial enough on their own, and there thus is no need for an artificial reward. The problem with this is that there always has to be one business unit who has to act as a first user in order for the core asset to be proven beneficial, which brings us back to one of the four issues, compensation for the first user. The adoption of more and newer core assets is strongly intertwined with this. As long as there is no trust in the core assets regarding, among other things, the return of these assets, there is no direct incentive to adopt the assets.

However, the larger part of the interviewees feels that monetary rewards will aid in resolving the aforementioned four issues. They acknowledge it solves just one piece of the puzzle, but it will help business units make the first step towards the adoption of the MIP assets. Other issues that need to be resolved are that the use of the MIP assets should be beneficial on its own by providing additional support and proper documentations, e.g. production plans, and the MIP assets should be validated for all product contexts, i.e. the validation set of Components & Services should be augmented to include all products that fall within the product line scope.

The final segment does not contribute to the validation of the aid funding model.

2.3 Conclusion

Based on both the interviews and the questionnaire results, it can be stated that the four identified issues indeed need to be resolved. In addition, it appears that a monetary reward seems to be effective regarding all four issues. Many believe that using core assets should be made more attractive by improving, among others, the usability, documentation, and the quality attributes of these assets. This, however, does require pilot projects and funding, since the core assets can only be optimized for the given contexts when it is clear how these assets behave in those contexts. When the core assets have grown mature enough, i.e. they are easy-to-use and the usage of these assets does realize the typical software reuse benefits, artificial incentives indeed are not required any longer. However, before achieving that level of maturity, the four identified issues do play a role in the progression of the core assets and it seems mandatory to provide some sort of compensation to deal with the issues. Besides monetary rewards, other rewards were expected to be effective also. An example of such a reward is receiving additional support from the core asset developers.

3. Map results on aid funding model

The aid funding model takes four specific issues into account. All these four issues have been identified as being crucial to the success of the software product line initiative in the context sketched above. In addition, it has been identified that monetary rewards seem to be effective to deal with all these issues in that context.

Further, the aid funding model adheres to all the identified properties a funding model should exhibit. First, transparency was identified as a desired property. Since the aid funding model calculates the required contributions for all the business units in six steps, which are composed of simplistic equations, the aid funding model indeed derives the required amounts in a transparent manner.

Second, it was stated earlier that when multiple releases of a core asset are being used, this decreases the benefits regarding the centralization of maintenance. This is because multiple versions of the same assets require maintenance. As stated in chapter 8, one of the approaches to funding was based on the principle of fairness. This was for instance necessary to make innovative development possible. The same reasoning holds for the adoption of newer versions of core assets. Since the lack of adopting the latest release leads to the need for additional maintenance support for older versions of core assets, less capacity can be assigned to innovative development, i.e. the development of new core assets. Therefore, it seems fair to make the corresponding business units pay for the additional maintenance support also.

Since monetary rewards appear to be effective, a funding model is the ideal manner to provide these rewards. The aid model makes use of individual scores for behavior per business unit. This score indicates to what extent a business unit is accounted for the development and the maintenance of the core assets.

3.1 ADDITIONAL ISSUES

The aid funding model has been set up in such a manner that it can easily be extended when other issues, which are peculiar to an organization, have been identified. In addition, if one of the identified issues does not play a role in a specific organization, the effects of that particular issue can be diminished through the use of the behavior constants or the behavior variables. It even is possible to leave these issues out of the equation entirely.

Other issues need to be regarded also. For instance, when adopting a software product line approach in an incremental manner, the core assets obviously are adopted in an incremental manner also. One of the issues with adopting core assets in an incremental manner is the fact that each product at that moment already has a software architecture, which likely differs from the product line architecture. As mentioned in chapter 6, each product should make use of a derivation of the product line architecture. This is necessary since the core assets are developed to fit in that architecture. Aspects like, among other things, the level of abstraction of the subsystems and all the defined interfaces need to be interoperable with the legacy architecture, which typically is not the case. Consequently, core assets will have to be wrapped. When adopting the core assets in the product development processes, a transformation from the legacy architecture towards a derivation of the product line architecture is imperative. For certain products, e.g. an MR scanner, this has proven to be a big issue. A monetary compensation for the additional effort necessary for the architectural transformation might be justifiable also. Further research is necessary to identify more issues.

4. PROS AND CONS OF AID MODEL

This section presents a brief discussion on the pros and cons of the aid model, which was presented in the previous chapter. The aid model is constructed in a simple and easy manner; it requires six steps to derive the values all business units are accounted for. The aforementioned reward-based model appears to be effective with respect to the issues also, but it requires much bookkeeping. The rewards are awarded based upon singular core assets that may be produced by any business unit. In addition, it requires a cash flow for each business unit that contributes a core asset, for each business unit that uses a core asset, and for each reuse sponsor of a core asset. On the other hand, the aid model only requires one cash flow per business unit per period, since all business units contribute a certain amount to the core asset developer. The rewards determine how much each of the business unit has to pay per period.

Furthermore, since the model explicitly addresses the four identified issues, an organization taking a software product line approach will be aware of these issues beforehand. This is due to the fact that the organization has to monitor how the business unit behave with respect to these issues in order to determine the required amounts. For instance, it forces an organization to monitor which business units adopt which core assets. It implicitly is then monitoring the rate of adopting core assets.

The aid model can be applied to all types of organization during all three phases of asset development. This is because the use of the behavior constants influence what issues are considered to be important and, consequently, have much influence on the amount a business unit is accounted for.

An additional benefit of the aid model is that is constructed in such a manner that additional issues can be addressed with it also. The sole constraint is that the behavior with respect to that issue can be quantified.

A difficulty concerning the implementation of the aid model is to determine what values should be awarded to both the behavior constants and the behavior variables. Obviously, business units not scoring well on certain issues will object when these issues are considered more important and are awarded high values for the corresponding behavior constants accordingly. A manner to deal with the acceptance of the funding model is to make sure that the amounts that business units are accounted for differ little when institutionalizing the funding model and that the deviations can increase over time.

In addition, it might be difficult for management to quantify the behavior of business units with respect to the four issues. In the previous chapter, four methods for quantifying the behavior with respect to the issues are provided. In certain cases, other methods may be more desirable.

In conclusion, the aid model addresses the four issues* that typically play a role at an organization taking a software product line approach by using rewards. It forces an organization to monitor the behavior exhibited by all business units with respect to these four issues. Furthermore, because of the provided variability, the aid model can be applied to all organizations during all phases of asset development. Acceptability for values for behavior constants might lead to resistance from business units not scoring well on certain issues. In addition, quantifying the behavior of business units might prove to be difficult also.

5. THE CONTRIBUTION

This thesis provides an overview of several funding models proposed in the literature, including an overview of the pros and cons of these models, and the suitability of the models. The main contribution of this thesis regarding the funding of software reuse practices is the novel funding model, i.e. the aid model. During this research, four issues peculiar to software reuse have been identified. Only one of the proposed funding models takes all these issues into account, namely the reward-based model. This model, however, is not applicable to all software reuse organizations.

The adjustable incentive-driven model is a funding model, which is very flexible regarding rewarding behavior. It can be applied to all contexts of software reuse organizations. If certain issues are peculiar to an organization, the funding model can easily be augmented to incorporate that issue also. Also, the funding model is applicable to all three phases of adopting a software product line approach. This is realized through the use of the issues and corresponding behavior constants. If certain issues are found to be important, the corresponding behavior constants are awarded a relative high value. In the sole asset development for instance, all assets are developed by the core asset developers. The focus is on defining the product line scope and the product line architecture and, develop the initial core assets. Therefore, there is no incentive for business units to develop core assets necessary. However, when the core assets have matured and newer versions are released, an incentive for using the newer version might seem in place, especially when backwards compatibility is not provided, as is the case at Philips Medical Systems.

6. FUTURE WORK

Much still needs to be done with respect to the funding of software product line practices. Three main directions can be identified. First, the four identified issues are not exhaustive and the model needs to be augmented with additional issues. An example of an additional issue is to take into account whether a business unit copes with a legacy architecture, i.e. if there is a need to transform a legacy architecture towards a derivation of the product line architecture, which would influence, among others, the rate of adopting core assets. If so, the business units might need additional compensation, since such a transformation requires additional effort. The aid model was derived in a specific organization, i.e. Philips Medical Systems. As each organization has its own characteristics, e.g. their approach to software product lines and the organizational culture, other organizational contexts are likely to provide additional issues that can be resolved with monetary rewards and, therefore, by a funding model. Second, the research on funding models should thus be performed in other organizational contexts also, especially with respect to the validation of the funding model. Finally, the actual validation of the aid funding model can only be done by applying it at a certain organization. This, unfortunately, resides outside the scope of this thesis. By actually applying the model, it will become clear whether the model indeed resolves the four identified issues and does not have any hidden drawbacks.

Future work thus comprises (1) augment the aid model with additional issues, (2) validate the aid model in different contexts, and (3) validate the model by actually applying it in an organization.

---

* A note on the four issues. An organization should be aware that not all problems related to these four issues are accounted solely to the product developers. If certain core assets are not applicable to all product contexts, behave poorly in certain product contexts, legacy architectures are not considered by the core asset developers, or proper documentation regarding the use of the core assets is not provided, the business units should not be accounted for this. It therefore seems mandatory that the causes for certain behavior of business units are looked at more closely also.

7. CONCLUSION

Based on the results from the questionnaire, the aid model adheres to all the identified requirements for a funding model. First, it is an easy-to-use funding model. Second, the calculations are transparent. Finally, it takes all the identified issues regarding the funding of core asset development into account. It can easily be augmented with additional issues, so it is even wider applicable. By using the behavior variables and the behavior variables, it is not only applicable to different organizations, but it is also applicable to all three asset development phases, i.e. sole asset development, joint asset development, and parallel asset development. The aid model thus qualifies as a simplistic and transparent model that addresses the four identified issues in a fair manner.

PART FOUR
DATA COLLECTION, METRICS AND TRACKING

This part is composed of three chapters. The first chapter first discusses the business concerns related to the three software product line activities, i.e. core asset development, product development, and management. The second chapter then introduces multiple software reuse metrics and transforms these into software product line metrics. The final chapter is concerned with the validation of the framework, i.e. it needs to be validated that the metrics framework indeed does address all the business concerns of the three software product line activities.

CHAPTER 12
IDENTIFYING BUSINESS CONCERNS

Keeping the long-term health of the software product line as the major concern is a sign that an organization is mature with respect to their effort [11]. Obviously, the ultimate goal of any organization is not to keep their software product line as healthy as possible on the long term, but to sell products. It, however, should be acknowledged that the software product line provides an indispensable means of achieving the long-term goals and keep business healthy.

This chapter provides an overview of all business concerns divided over the three software product line activities [11]. The first section addresses the core asset development. Then, the product development is addressed and, finally, management is addressed. The chapter will conclude with a brief overview of the concerns.

1. CORE ASSET DEVELOPMENT

Core asset development is the activity of developing, maintaining, and evolving all the core assets. As mentioned earlier, five different inputs are required to deliver three different outputs. The former consist of the product constraints, styles, patterns and frameworks, production constraints, production strategy, and the inventory of preexisting assets. The latter consists of the product line scope, the core asset base, and the corresponding production plans.

The main concern of this activity thus is managing the five different inputs in order to derive the three outputs. The objective of this activity is threefold. This will be discussed in more detail in the following three subsections.

1.1 PRODUCT LINE SCOPE

First, the product line scope determines which products can be derived from the software product line. The product line scope therefore determines, among other things, the required variability which the core assets need to incorporate. Therefore, it is a strong determinant with respect to the leverage of the use of core assets. For instance, when the scope is too large, too much variability is required to incorporate all the possible requirements. This counteracts the leverage of using the core assets. This is because the core assets need to incorporate so much variability that the development of these assets is relatively high. In addition, the use of the assets in product contexts will require so much tailoring that it might cost more to tailor and integrate it then it would have cost when a similar product-specific asset would have been developed from scratch.

On the other hand, if the product line scope is defined too small, other problems arise. For instance, the core assets are only used in a small amount of products and therefore the returns of these core assets is minimal. In addition, if the scope is small, only little variability is required in the core assets. This obviously hinders the future growth of the software product line, since the core assets are not built in a generic enough fashion.

As mentioned, keeping the long-term health of the software product line is the major concern of an organization that is mature with respect to their software product line effort. The product line scope is an indicator for the future growth and, therefore, an indicator for the long-term health. One of the major business concerns of an organization taking a software product line approach thus is that the product line scope needs to be defined adequately and adjusted, when necessary.

1.2 CORE ASSET BASE

Second, the core asset base is the collection of all core assets, including a product line architecture and reusable components. The use of these core assets forms the essence of a software product line effort. This is due to the fact that all benefits are gained by the use of these assets. The long-term health of the software product line also is reflected by the maturity of these core assets. As mentioned above, one of the key determinants of the long-term health is to what extent the core assets are built in a generic fashion. With this in mind, the first business concern regarding the core asset base can be identified, namely the future evolution of the core assets needs to be accommodated through for instance documenting how this evolution should be carried through.

Obviously, core assets are to be used within multiple products contexts. These core assets should therefore provide a certain amount of variability. How this variability can be utilized to meet the requirements of a specific product, is specified in an attached process. This, again, can be realized through documenting the necessary steps. The second business concern related to the core asset base then is that the core assets should be applicable to all products contexts, which fall within the product line scope. In addition, the use of core assets should be more beneficial than developing a similar product-specific asset from scratch. If this is not the case, the software product line effort obviously is not worth having. Please note that it is beneficial when the benefits are greater than the costs and that not all benefits can be specified in amounts, e.g. customer satisfaction and unified user interfaces.

1.3 PRODUCTION PLANS

Finally, the production plans are strong determinants with respect to the usability of the core assets. Productions plans describe how a certain product can be derived from a collection of core assets. They constitute of all the attached processes of all related core assets required for the development of a specific product. It, among others, specifies how the specific tools are to be applied in order to use, tailor, and evolve the subject core assets.

As mentioned above, one of the major concerns of core asset development is that the core assets should be usable in all products within the product line scope. Also, the use of such core assets needs to be more beneficial than developing a similar product-specific asset from scratch. The production plans address the user side of the core assets, i.e. they specify how a collection of core assets can be utilized in the derivation process of a particular product.

The underlying business concern is that the adoption of core assets is stimulated in the product development processes of the organization without artificial incentives. This is because providing proper production plans improves the benefits gained from the use of the core assets and it stimulates the product developers to adopt the core assets.

2. PRODUCT DEVELOPMENT

Product development is the process of deriving products making use of both core assets and product-specific assets. Production plans prescribe how all the core assets should be utilized in a particular product context in order to optimize the production efficiency. Obviously, the main business concern of this activity is to develop and sell products and produce revenues accordingly. The underlying business concern is that the products need to be developed as efficient as possible, but with preservation of the required quality. Doing so results in better competitive positions and higher profit margins. Also, when using higher-quality assets, the organizations reputations becomes stronger, as does the competitive position. Finally, when the time-to-market becomes shorter and better predictable, the competitive position is strengthened again. As mentioned above, all these requirements for a stronger market position are typical benefits gained from the application of a software product line approach. A related concern is that it should be proven that the use of core assets is beneficial, i.e. the adoption of core assets in the product development processes needs to be stimulated.

3. MANAGEMENT

Management orchestrates the above two activities at two different levels, i.e. technical and organizational. The above two activities must be given resources, coordinated, and supervised at both levels. As mentioned earlier, organizational management is the authority that is responsible for the ultimate success or failure of the product line effort. The main concern of this activity therefore is that the other two activities are performed in an adequate manner. Management must be equipped with tools to provide the necessary resources, and coordinate and supervise the other two activities. Typical business concerns of management are roadmapping and scoping. Management thus has to make sure that the product line scope is aligned with the roadmapping of the entire organization. Furthermore, all the above mentioned concerns have to be taken into account by management also. Management has to intervene for instance when the product line scope and the organizational roadmapping are not aligned or the production plans are of poor quality leading to a stagnating adoption of core assets.

4. OVERVIEW OF CONCERNS

In the above, many different concerns have been identified. The following chapter provides an amount of metrics addressing all the identified business concerns. In the third chapter of this part, the mapping between these business concerns and the metrics is addressed. These business concerns will be listed briefly below.

- The product line scope needs to be determined adequately
- The future evolution of the core assets needs to be accommodated through for instance documenting how this evolution can be carried through
- The use of core assets should be more beneficial than developing a similar product-specific asset from scratch
- Stimulate the use of core assets through providing proper production plans
- Develop and maintain products as efficient as possible, as soon as possible, and with a high as possible profit margin
- Tools to provide the necessary resources, coordinate, and supervise the other two activities must be available

CHAPTER 13
SOFTWARE PRODUCT LINE METRICS

Measuring and monitoring is a key element when launching and sustaining a software product line initiative, since this forces an organization to evaluate their processes and identify the weaknesses. This chapter provides an overview of software reuse metrics. The main purpose of this chapter is to transform these software reuse metrics into software product line metrics.

The first section introduces the definitions of software metrics, software reuse metrics, and software product line metrics. The subsequent sections discuss multiple applications of these metrics, i.e. how they contribute to all three levels of corporate strategy. First, the operational level regarding metrics is discussed. These metrics aim at providing more insight in certain aspects of the development processes. The second level of metrics is concerned with the costs and benefits related to the software product line approach. It provides more insight in the business-level of business concerns. Finally, the corporate-level is addressed. The main purpose of the strategic level is to provide more insight into the long-term vision of the organization. The chapter will then be concluded with an overview of the given metrics.

1. DEFINING METRICS

In [40], software metrics are defined as "*quantifiable attributes of software, either directly measured or computed from values directly measured from the software*". Software reuse metrics are software metrics specifically designed for use in a software reuse context. The definition used in this thesis for software reuse metrics is quantifiable attributes of software either directly measured or computed from values directly measured from the software, with that software being developed by using existing assets also. The metrics of interest, however, are software product line metrics. Software product line metrics are software reuse metrics specifically to be used when one adopts software product lines as a means of implementing a software reuse program. These metrics address specifically the concerns of the three software product line activities, i.e. core asset development, product development, and management.

The purpose of using software product line metrics is to discover which benefits; strategic business benefits as well as tactical engineering benefits, are gained by the use of a product line approach and, maybe more important, at what costs these benefits are realized. As mentioned, the tactical engineering benefits comprise, among others, the typical software reuse benefits. The strategic business benefits are, among other things, a strengthened company reputation and strengthened market positions, due to more robust products.

In literature, there are many proposed software reuse metrics. These metrics often are captured in economic models. This chapter will focus on different software reuse metrics and how they can be transformed into software product line metrics, i.e. how they can be used to address the specific needs of an organization taking a software product line approach. The following chapter applies the derived metrics into a framework.

1.1 UNITS OF MEASURE

Many units of measure can be used, varying from lines of source code to components. There is no single best unit of measure. For instance, when one uses only a small part of the functionality provided by a component, how much should count as being reused? When measured in components, the entire component would count as being reused, even though only a small part of that object is actually being reused. When counted in LOC, how do you determine how much LOC is actually reused? This seems to be too much work to determine accurately. The best unit of measure depends on multiple factors. For instance, when tools for deriving the actual used source code of a component are available, it makes sense to make use of LOC. However, when the components are all equally of size, it can be argued that components provide a good measure of unit also. Also, effort estimations seem appropriate also.

1.2 CORPORATE STRATEGY

Three different levels of corporate strategy can be identified, i.e. operational, business-level and corporate-level. Operational strategies are concerned with how the component parts of an organization deliver effectively the corporate- and business-level strategies in terms of resources, processes and peoples [29]. The general idea regarding the metrics is that certain aspects of the development processes are analyzed. By doing so, the weaknesses of these processes can be identified more effectively.

The business-level strategy is about how to compete successfully in particular markets [29]. The focus thus is on the overall development process of products rather than on aspects of these development processes. The main purpose is to identify the costs and benefits of developing a product in a software product line context and compare it with developing that product in a traditional manner. By doing so, it can be analyzed how effective product development is performed.

Corporate-level strategy is concerned with the overall purpose and scope of an organization, and how value will be added to the different parts of the organization [29]. In the case of an organization taking a software product line approach, for instance the notion of scoping becomes important. This obviously is due to the fact that future products need to be taken into account also. An example of a possible issue is that current and future products might have conflicting requirements.

1.3 RELATING METRICS TO STRATEGY LEVELS

With respect to the three levels of strategy, many different metrics can be used. In order to support all three levels of strategy levels, we introduce three levels of software metrics also. As mentioned, the lowest level of strategy, operational strategy, is concerned with aspects of the development processes. The metrics related to this level of strategy should therefore measure certain aspects of the development processes.

The second level of strategy, the business-level strategy, is focused on the product development as a whole rather than on aspects of product development. The metrics related to the operational strategy ideally should support the metrics related to this level of strategy. By doing so, the effort required to perform the analysis for this strategy level is minimal.

The highest level of strategy, corporate-level strategy, is concerned with the long-term roadmapping and scoping of the organization. With respect to a software product line effort, the notion of scoping is of most importance. The metrics supporting this level of strategy should therefore support the process of scoping, i.e. determine which products are incorporated in the software product lines and which are not. The cost-benefit analysis from the business-level strategy can be used to determine whether or not certain products have proven to be beneficial to be incorporated into the product line scope. By using such information, the earlier investment analyses can be evaluated and, where necessary, adjusted. This obviously is necessary to be capable of performing more accurate future investment analyses.

The relations between the three levels of strategy and the corresponding metrics are depicted by the figure below.

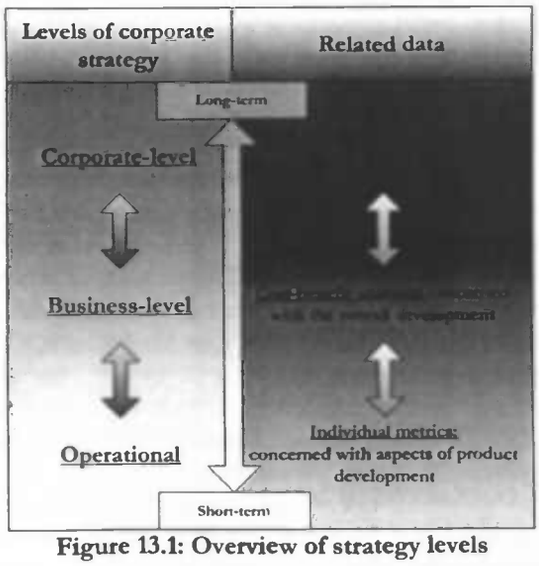


Figure 13.1: Overview of strategy levels

2. OPERATIONAL STRATEGY

The following subsections describe a selection of metrics for the operational level of corporate strategy [7] [13] [18] [19] [20] [42]. They specifically address certain aspects of core asset development and product development. These metrics form the input for the cost-benefit analysis provided in the following section. The focus of this level is on aspects of products, e.g. core assets.

2.1 REUSE LEVELS

A reuse level specifies the percentage of source code of a product resulting from software reuse. It, however, does not include all the core assets, for instance the product line architecture or documentation. It is an easy-to-calculate metric, giving some insight solely into the actual reused source code. In literature, only one reuse level is proposed. When regarding a software product line initiative, this metric does not suffice. Why this is so, will be discussed below. To deal with software product lines also, an additional reuse level is introduced here. The traditional reuse level will be referred to as the absolute reuse level and the additional reuse level will be referred to as the relative reuse level.

The absolute reuse level requires two parameters, namely the amount of reused source code and the total amount of source code of the product at hand, as specified by the following equation.

$$Absolute\ reuse\ level = \frac{reused\ source\ code}{total\ source\ code} \cdot 100\%$$

Equation 13.1: Absolute reuse level

In order to make use of the absolute reuse level to measure progress in a software product line context, caution must be taken. Consider for instance the case that a product-specific feature is added to a product. Even though there is no decrease in the actual reused source code, the reuse level will be lower. A lower reuse level intuitively indicates that the developers of the subject product make less use of core assets, even though they have no choice in the matter. In addition to this, incorporating the product-specific feature into the core asset base, thus extending the variability offered by the core assets, could lead to both less effective core asset development and less effective product development. The former is due to more complex core assets and the latter due to more complex configuration processes and integration processes of these core assets. It can thus be argued that higher absolute reuse levels do not strictly imply more effective development.

So, specifically when dealing with software product lines, an additional metric is necessary, namely one that specifies how much of the core assets available for a product are actually used in that product. To provide such information, this thesis introduces the relative reuse level. This metric is particularly useful to measure the progress of an incremental approach to institutionalizing a software reuse program, since this metric should then increase over time. Only when core assets are developed faster than they are adopted in a product, the relative reuse level would decrease rather than increase. It is calculated equally as the absolute relative level, with the difference being that the denominator is the total available reusable code rather than the total amount of source code. The total available reusable code reflects the size of the code of all the core assets which could theoretically be used in a given product. The relative reuse level of a product thus equals 100 percent when all the available core assets, for that specific product, are used, even though the product might make use of product-specific assets also. The relative reuse level is calculated as follows.

$$Relative\ reuse\ level = \frac{reused\ source\ code}{total\ available\ reusable\ code} \cdot 100\%$$

Equation 13.2: Relative reuse level

2.1.1 APPLICABILITY TO SOFTWARE PRODUCT LINE PRACTICES

This subsection addresses the applicability of the both the absolute reuse level and the relative reuse level with respect to the three fundamental software product line activities.

- **Core asset development:** Both the absolute reuse level and the relative reuse level indicate to what extent the core assets are actually used in products. If some core asset structurally is not used, this could for instance be an indicator that the product developers have better solutions for the functionality offered by that core asset. It also might indicate that the corresponding production plan or the quality attributes the core asset exhibits do not suffice. Shortly, it are indicators for the core asset developers whether or not improvement of the core assets is necessary. It should be noted that the two reuse levels do not indicate specifically per core asset how often it is used, but it can be used as a starting point to look why the adoption of core assets is for instance stagnating.
- **Product development:** The product developers can make use of both reuse levels to measure the progress of adopting the use of core assets in their processes.
- **Management:** Management can measure the progress of both the development of core assets and the adoption of these core assets in product development. If the absolute reuse level differs too much from the relative reuse level, implying that the core asset base is relatively small, it is an indicator that one of the following might be occurring: (1) the core asset development cannot keep up, e.g. with evolving market demands or (2) the product line scope is defined too large.

2.2 ADDITIONAL REUSE EFFORT

The Relative Cost of Writing for Reuse indicates how much more effort is required to develop a core asset in comparison with the development of a product-specific asset. Since the necessary variability needs to be incorporated also, the development of core assets typically requires more effort than the development of product-specific assets. In most software reuse contexts, the development of a core asset requires about 1.5 times the effort the development of a product-specific asset requires [40]. It should be noted that this metric is concerned only with writing source code, i.e. the development of reusable components. This metric is calculated as follows.

$$Relative\ Cost\ of\ Writing\ for\ Reuse = \frac{Cost\ of\ code\ written\ for\ reuse}{Cost\ of\ new\ code}$$

Equation 13.3: Additional reuse effort

A shortcoming of the metric above is that it does not take for instance the development of the product line architecture into account. Therefore, a transformation of this metric is necessary. To specify the distinction between these metrics, this thesis introduces the metric additional reuse effort. The additional reuse effort specifies the relation between the development of all core assets and the development of similar product-specific assets. Two inputs are required, namely the cost developing a core asset, C_{cab}, and the costs of developing a similar asset for a specific product, $cost\ of\ asset_{product\text{-}specific}$. To specify that a particular asset is being addressed and not the entire asset base, a_i is provided as a parameter, which obviously represents the subject asset. The additional reuse effort is then calculated as follows.

$$Additional\ reuse\ effort = \frac{C_{cab}(a_i)}{Cost\ of\ asset_{product-specific}(a_i)}$$

Equation 13.4: Additional reuse effort

2.2.1 APPLICABILITY TO SOFTWARE PRODUCT LINE PRACTICES

This subsection addresses the applicability of the additional reuse effort with respect to the three fundamental software product line activities.

- **Core asset development:** The additional reuse effort indicates how effective core asset development relative to product-specific development is performed. If there exist significant differences between the effort needed for the core asset development and the product-specific development, one might need to revise the product line scope. An example of a cause for relatively too high costs for core asset development is for instance that the scope is defined too large and, therefore, too much variability has to be incorporated in the core assets. On the other hand, if the core asset development costs are too low, this might indicate problems also. For instance, the level of abstraction of the components might be too low, which diminishes the leverage of the

usage of these components. Another reason might be that the product line scope is defined too small. As mentioned earlier, a too small product line scope restricts future growth of the entire software product line. This is, among others, due to the fact that with a small product line scope, the core assets typically are not developed in a generic enough fashion. This is reflected in the fact that there is not much additional effort required to incorporate the necessary variability.

- **Product development:** Regarding the additional reuse effort, there is no direct relevance for product development. Product developers need to get involved, though. This is due to the need for the estimation for the costs of developing a similar product-specific asset, which can only be done accurately by persons having experience with building such assets for specific products.
- **Management:** Since management is concerned with, among others, providing resources for core asset development, it is important that the core asset development is performed effectively. If the core asset development efforts stay too high or too low, management needs to intervene.

2.3 INTEGRATION EFFORT

The Relative Cost of Reuse is a software reuse metric indicating how much effort is required to integrate a reusable component into a product and compare it with the effort it requires to develop a similar product-specific asset from scratch. Again, this metric is aimed at reusing components only. The Relative Cost of Reuse is calculated as follows.

$$Relative\ Cost\ of\ Reuse = \frac{Cost\ of\ tailoring\ and\ integrating\ component\ for\ reuse}{Cost\ of\ developing\ similar\ product-specific\ component}$$

Equation 13.5: Relative Cost of Reuse

In order to indicate that the reuse of other assets is incorporated also, e.g. a product line architecture, the software product line metric, referred to as the integration effort, is defined as follows*.

$$Integration\ effort = \frac{Cost\ of\ tailoring\ and\ integrating\ core\ asset\ for\ reuse}{Cost\ of\ developing\ similar\ product-specific\ asset}$$

Equation 13.6: Integration effort

2.3.1 APPLICABILITY TO SOFTWARE PRODUCT LINE PRACTICES

This subsection discusses the applicability of the integration effort with respect to the three fundamental software product line activities.

- **Core asset development:** This metric is a very important indicator for the successfulness of the core asset development. The aim of integrating a core asset into a specific product should be beneficial. This metric is an indicator specifically for the development costs. If these costs are not much lower than the costs of developing a similar product-specific asset, business units will likely not adopt the core assets. However, it can be argued that other benefits might outweigh high integration costs, e.g. a shortened time-to-market or a unified user interface. But, when the integration costs are relatively high, it is likely that the other benefits will be less noticeable also.
- **Product development:** This metric also is very important for the product developers. This is due to the fact that the usage of the core assets again should be more beneficial than developing the assets from scratch. When the integration effort becomes too high, the product developers will not be willing to adopt the core assets in their product development. The main reasons for adopting the core assets obviously are that the development costs and the maintenance costs can be reduced, the time-to-market can be shortened and the overall quality can be increased. All of these reasons are reflected in this metric.

* Similar metrics can be used for the maintenance costs, the quality of assets, and the time-to-market. These additional metrics are useful when regarding an assessment for an entire product, which is the topic of the subsequent section.

- **Management:** Obviously, since management is concerned with the orchestration of the core asset development and the product development, this metric is very important for them also. Management has to monitor whether or not the estimated expectations regarding the investments and the related return-on-investment are realized.

3. BUSINESS-LEVEL STRATEGY

This section is concerned with the business-level of corporate strategy. The focus is on identifying and quantifying certain costs and benefits of a software product line initiative, and, in specific, on products as a whole. This is done by analyzing the costs and benefits per product. The metrics of this level function as input for the metrics of the corporate-level strategy.

3.1 COST-BENEFIT ANALYSIS

Regarding the cost-benefit analysis, many different models have been developed [7] [10] [12] [34] [36] [39]. A cost-benefit analysis is nothing more than the weighing of the costs versus the benefits. If all benefits are referred to as B_i and all costs as C_i, then the core of a cost-benefit analysis is presented by the following equation*.

$$CBA = \sum_{i=1}^{n} B_i - \sum_{j=1}^{m} C_i$$

Equation 13.7: Initial cost-benefit analysis

These models have in common that the focus is mainly on both the development costs and the maintenance costs. Obviously, it is hard to estimate for instance how much of the increased revenues are due to higher-quality products or a shortened time-to-market, since so many factors play a role. The cost-benefit analysis presented in the following identifies, besides the standard cost types, two different types of benefits, i.e. quantitative benefits and qualitative benefits. The main difference between these two benefits is that the former is tangible and the latter intangible. Obviously, both have to be taken into account in order to get an overall picture.

3.2 COSTS

As mentioned earlier, four different software product line cost types were identified, i.e. C_{org}, C_{cab}, C_{unique}, and C_{reuse}. With respect to the development of products, all these costs will have to be taken into account. This includes the costs for changing the organization, and developing, maintaining, and evolving the core asset base. Even though the costs are not directly related to that product, these costs will have to be allocated to products somehow. The easiest way to allocate the proper share of C_{org} and C_{cab} is to make use of the funding amount of each business unit distributed over their products in an appropriate manner. An example of such a manner is to allocate more costs to certain products when these products are depending more extensively on the core assets. If a business unit is concerned with the development of just one product, the entire amount can be allocated to that product.

The other costs can be derived from the product development process. All the costs related to integrating core assets in a product and the costs related to developing product-specific assets, i.e. C_{reuse} and C_{unique} respectively, are values that can be abstracted from project management processes.

Table 13.1: Cost derivation

Cost type	How to derive
C_{org}	These values are represented by the required amount for funding
C_{cab}	
C_{unique}	These costs can be derived from project management processes, e.g. from planning and budgeting
C_{reuse}	

* In order for a cost-benefit analysis to be accurate, the net present value of the costs and benefits should be taken. However, since the cost-benefit analyses should take place on a regular base and are only meant to reflect on a given period, this has been omitted. For details on the net present value, refer to subsection 4.1 of this chapter.

3.3 TANGIBLE BENEFITS

The tangible benefits are benefits that can be measured directly. The main benefits that can be measured are the benefits regarding both the development costs and the maintenance costs. Both will be discussed in the following subsections. The general idea behind these two benefits is that applying a software product line approach leads to savings concerning both the development and maintenance costs, e.g. by using core assets rather than develop product-specific assets from scratch. The metrics from the previous section can be used to estimate the benefits.

3.3.1 DEVELOPMENT BENEFITS

The first tangible benefit concerns the reduced development costs. By using the core assets rather than developing product-specific assets from scratch, the development costs of products can be decreased drastically. This benefit of the software product line expresses the savings on the development costs, i.e. the costs that have not been spent due to the software product line approach. In the previous section, it was prescribed that the use of core assets should be compared with developing product-specific assets from scratch. All core assets used in a certain product therefore have led to a decrease in the development costs at an asset level. However, the cost-benefit analysis addresses the benefits at the product level. By adding all the benefits of the individual assets, the reduced development costs due to the use of core assets are obtained.

3.3.2 MAINTENANCE BENEFITS

The second tangible benefit concerns the reduced maintenance costs. The core asset developer maintains the core assets and provides patches or bug fixes accordingly. From a business unit point-of-view, the maintenance costs for a given asset consist solely of applying the patches or bug fixes provided by the core asset developer. The costs for the maintenance of the core asset have already been captured in C_{cab}. The benefits concerning maintenance gained from the software product line effort therefore are equal to the expected costs of having to resolve certain assets themselves minus the costs of applying the patches or bug fixes provided by the core asset developer. The expected costs are likely not to differ too much from the costs the core asset developer has made.

3.4 INTANGIBLE BENEFITS

Intangible benefits are benefits that cannot be measured directly. Examples of such benefits are increased sales due to higher-quality products or less turnover due to more interesting and challenging work because of the software product line approach. For a thorough overview of intangible benefits, refer to chapter 6. In order to deal with these benefits, the use of scenarios is proposed here. It should be noted that these benefits should only be taken into account when the estimates exhibit an acceptable degree of certainty. This is especially true when the worst case and the best case of a given scenario differ much. The following subsection presents how scenarios can be used to quantify intangible benefits. Finally, it is presented how the intangible benefits can be incorporated in the overall cost-benefit analysis.

3.4.1 SCENARIOS

A scenario can be sketched for all the identified benefits. For a given organization, specifically the intangible benefits appear to vary. For instance, Philips Medical Systems' products have to exhibit a similar user interface. If they realize this, the customers will be more satisfied with their products. This could lead to an increase of sales. A scenario can be sketched in which the unified user interface is related to the increased sales. A bandwidth for the worst case and the best case of the scenario needs to be determined. The worst-case scenario, i.e. the lower bound of the bandwidth, should prescribe how sales are affected anyway, whether it is in a negative or in a positive manner. The best-case scenario should provide an optimistic, yet realistic view on how the intangible benefit affects sales.

It seems imperative that the marketing and sales entities of a business unit are involved in sketching these scenarios. They are in the best position for doing so, since they are strongly interacting with the customers and are likely to have an idea of how certain factors can influence sales.

3.4.2 ADDING UP BENEFITS

All the quantified scenarios should be taken into account when performing the cost-benefit analysis. The intangible benefits, however, have a higher degree of uncertainty than the other benefits have. This is due to the fact that all kinds of factors are of influence on the intangible benefits. As mentioned above, the scenarios are composed of bandwidths, with the borders being the worst-case scenario and the best-case scenario. When adding all the intangible benefits, two values are of interest. First, consider the worst-case scenarios. When adding all the worst-case scenario values, the total value of all scenarios is obtained. Since the worst-case scenario should only contain guaranteed benefits, one can assume that the summed worst-case scenarios will be realized with a certain degree of likeliness. In addition, the same can be done for the best-case scenarios. The total quantified benefits gained from the intangible are likely to reside within the bandwidth of the total of the worst-case scenarios and the total of the best-case scenarios.

3.5 PERFORM COST-BENEFIT ANALYSIS

In the previous subsections, all the costs and benefits were identified. Even though the equation above prescribes how to handle the costs and benefits, that equation needs to be augmented with an additional factor, namely the bandwidth of the total intangible benefits. Consequently, the cost-benefit analysis will have a bandwidth also. The bandwidth of the cost-benefit-analysis is calculated as follows.

$$\left(\sum_{i=1}^{m} B_{tan_i} + \sum_{j=1}^{n} B_{intan_j} - \sum_{k=1}^{o} C_k\right)_{worst-case} \leq CBA \leq \left(\sum_{i=1}^{m} B_{tan_i} + \sum_{j=1}^{n} B_{intan_j} - \sum_{k=1}^{o} C_k\right)_{best-case}$$

Equation 13.8: Cost-benefit analysis

In the above calculation, the lower bound of the bandwidth is determined by the worst-case scenarios of the intangible benefits and the upper bound by the best-case scenarios. As mentioned, the cost-benefit analysis only considers one product and every product thus requires the same estimations and calculations. As commonality is a key component in software product lines, the same holds for the cost-benefit analysis. Since all products included in a product line scope exhibit commonalities and are aimed at a single market, the expected costs, tangible benefits, and intangible benefits of a given product can easily be translated to a similar product. Especially the savings concerning both the development efforts and the maintenance efforts are likely to be transferable across multiple products, since any core asset will be used in multiple products by definition.

4. CORPORATE-LEVEL STRATEGY

The corporate-level of strategy addresses the long-term vision of an organization. The focus of this level is on the entire product base and the corresponding product line scope. The related metrics are concerned with return-on-investment and scoping. This section provides a manner to look at investment analyses, both from a retrospective as a prospective point of view. Many options for doing so are proposed in the literature [10] [11] [12] [15] [39] [41] [49]. This section proposes the most suitable option. The main reasons why the chosen solution is most appropriate is that even though it takes into account scenarios, time, and uncertainty, it still is not complex. Other methods, for instance, do not discount the returns or take uncertainty into account. A typical approach to discount money in business is referred to as the net present value. The net present value prescribes that returns now are worth more than equal returns later on. The general idea behind discounting returns is that an organization always has the option to invest their capital in the capital market with the corresponding interests and returns.

4.1 NET PRESENT VALUE

The net present value [9] is a method to incorporate the factor time with respect to the value of money. The net present value is calculated as follows.

$$NPV = \frac{cash\ flow}{(1 + discount\ rate)^{time}}$$

Equation 13.9: Calculating the net present value

The present value of a cash flow is thus determined by discounting it for *time* periods using the value of *discount rate* per period Typical values for these variables are entire years and ten per cent respectively. For instance, if one expects that an investment returns € 1,000,000 over two years, the net present value of that return is calculated as follows.

$$NPV = \frac{1{,}000{,}000}{(1+0.10)^2} = 826{,}446$$

Equation 13.10: Calculation example

When comparing investments, the return of € 1,000,000 over two years equals as a present return of € 826,446. If another investment returns € 925,000, but these returns are expected to occur after one year, the net present value of that investment is € 840,909. By utilizing the net present value, it is concluded that, even though the returns are lower, the latter investment is preferable. This is because this investment exhibits a higher discounted return. Since the net present value prescribes that future returns are discounted, it provides a means to evaluate investments with returns that vary with respect to the time in which these returns are expected.

4.2 UNCERTAINTY AND SCENARIOS

The other factor influencing investment analysis is uncertainty. When making investments, the actual returns and the time in which these returns occur typically are unknown. A structured approach to consider the investments and corresponding returns is by using architectural scenarios and strategic scenarios [1] [49]. The former constitutes a structured evaluation of all investments and returns, discounted with respect to the time factor. Architectural scenarios can be considered as a sequence of events, which are either investments or returns. Investments are represented by negative cash flows and returns are represented by positive cash flows. The net present value of an architectural scenario is calculated by adding all the net present values of each of the individual events.

The latter, strategic scenarios, encapsulate assumptions about the future: market developments, application developments, and technology developments. An architectural scenario enables certain features for a product by performing an architectural transformation. If a strategic scenario predicts a high business for an enabled feature, the architectural scenario has a high value in that strategic scenario. Conversely, if an enabled feature has no business value in a given strategic scenario, the value of the architectural scenario is low.

Obviously, uncertainty plays a large role in evaluating the architectural scenarios. It therefore seems appropriate to speak of expected net present value. The following subsection describes how the above can be used for assessing the product line scope and the evolution of it.

4.3 MODELING THE VALUE OF INVESTMENTS

An investment is of high value when the expected benefits from that investment are very probable and the expected time between making the investment and getting the return on investment is short. The model to evaluate an architectural scenario comprises the following four steps [49].

Step 1 Sketch the architectural scenario
Step 2 Sketch the strategic scenarios that are of most importance
Step 3 Estimate the cash flows for the architectural scenarios and the strategic scenarios; estimate both the required investments and the corresponding returns
Step 4 Calculate the expected NPV with the following equation.

$$NPV_{expected}(ArchScenario, StratScenario[1..n]) =$$

$$\sum NPV(ArchScenario, StratScenario[i] * probability(StratScenario[i])$$

Equation 13.11: Deriving the expected net present value

The method of architectural evaluation merely provides a means to support decision making regarding the long-term. For instance, the product line scope is reflected in the architectural scenario and by analyzing strategic scenarios, the direction in which to evolve the product line scope can be chosen more accurately.

5. RELATIONS BETWEEN THE METRIC LEVELS

Above, three levels of metrics have been presented. First, the lowest level of metrics supports the identification of weaknesses in the development processes at the asset level. These metrics each address certain aspects of core asset development and product development. The second level of metrics provides a means to perform cost-benefit analyses at the product level. Finally, the third level of metrics is concerned with the notion of scoping, i.e. investment analyses. Based on expected market evolutions, the most valuable architectural scenario is chosen. The relations among the collected data, the three levels of metrics, and the corresponding outputs are presented in the figure below.

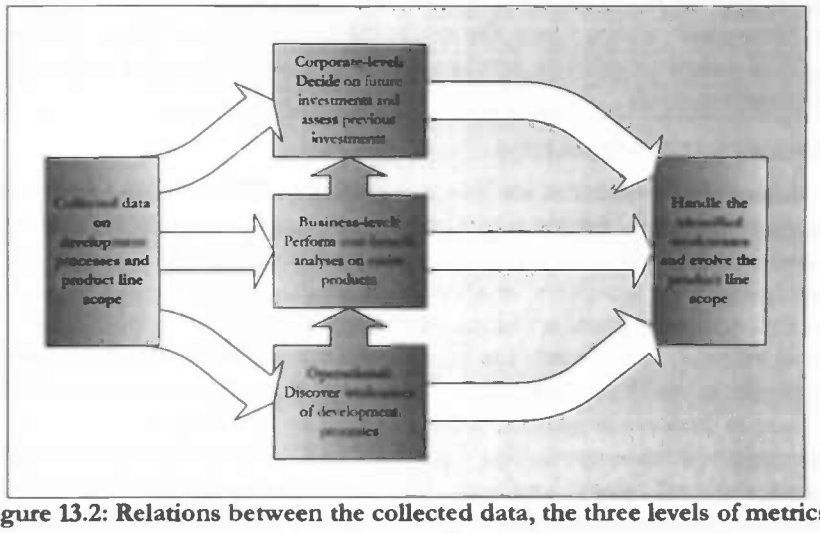


Figure 13.2: Relations between the collected data, the three levels of metrics, and the corresponding output.

6. CONCLUSION

Software reuse metrics are defined as quantifiable attributes of software either directly measured or computed from values directly measured from the software, with that software being developed by using existing assets also. Software product line metrics are referred to as software reuse metrics that address the specific needs of an organization taking a software product line approach to product development. Three levels of metrics have been proposed, each metric addressing a particular strategy level. These strategy levels comprise operational, business-level, and corporate-level. The lowest level of metrics is concerned with aspects of core asset development and product development. The second level of metrics consists of a cost-benefit analysis at the product level. The third and final level is concerned with investment analysis and focuses on the entire product base.

CHAPTER 14
ADDRESSING BUSINESS CONCERNS

In chapter 12, the business concerns of the three software product line activities were identified. The previous chapter introduced three levels of metrics, which addressed three levels of strategy. The focus of this chapter is to validate that the metrics indeed address all identified business concerns.

The first section presents the approach to validation. The subsequent three sections each address a specific software product line activity and the corresponding business concerns. The fifth section provides a discussion on the pros and cons of the metrics framework. Next, the contribution of this thesis regarding metrics is explicated. After that, future directions for the metrics framework are presented and a conclusion is provided.

1. VALIDATION APPROACH

The other objective of this thesis is to provide a framework for measuring a software product line program over time. The obvious main research question is: How can one determine if a software product line program is successful? In order to determine this, all the related costs and benefits first needed to be identified. Then, economic models needed to be considered also, e.g. how to identify and quantify certain benefits. This brings forth the need to obtain methods to quantify the identified costs and benefits. All this has led to the development of a metrics framework, which was the topic of the previous chapter.

The validation of the metrics framework takes an informal approach. The general idea of the metrics framework is that all main concerns of the three software product line activities are addressed. These concerns will be discussed in the following three subsections. In chapter 12, the typical business concerns related to a software product line effort were identified. The previous chapter proposed a structured method to analyze the progress of the software product line effort, the related costs and benefits, and the correctness of the product line scope. In addition, it provided a means to help identify costs and benefits in general*, and quantify intangible benefits.

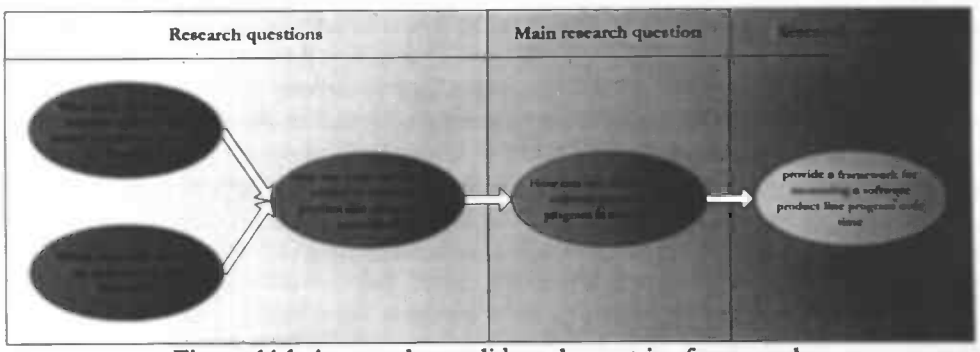


Figure 14.1: Approach to validate the metrics framework

2. CORE ASSET DEVELOPMENT

Four business concerns related to core asset development have been identified. Each of these business concerns will be addressed in the following subsections. The goal of these subsections is to present why and how the metrics framework supports that business concern.

2.1 PRODUCT LINE SCOPE

The first major business concern is that the product line scope needs to be determined adequately. The product line scope is a strong indicator for the future of the software product line, since it determines both the current product base and accommodates the growth of the software product line with respect to the future product base. The correctness of the product line scope is addressed at multiple metric levels.

* An elaboration on all related costs and benefits is provided in chapters 5 and 6.

The operational level is concerned with identifying weaknesses of both core asset development and product development. Certain weaknesses in these processes might indicate that the product line scope is defined too large or too small. For instance, when the development of a core asset requires multiple times the effort it requires to develop a similar product-specific asset, this might be due to the fact that the core assets have to incorporate too much variability. Inversely, if the core assets need to incorporate little variability, this will be reflected by little differences between the effort required to develop core assets and the effort required to develop a similar product-specific asset. The latter might be an indicator that the product line scope is defined too small.

The second level of metrics is concerned with products as a whole. All the costs and benefits regarding the utilization of the software product line approach for a given product are taken into consideration. As with the operational level, certain inefficiencies regarding aspects of product development might indicate that the product line scope is defined too large or too small.

Finally, the corporate-level metrics address the product line scope in specific. This is done by providing a means of evaluating the business value of the product line scope in a given future context. The product line scope is for instance reflected in the product line architecture. This is because the product line architecture needs to provide both the potential to implement certain functionality and impose values for the quality attributes of the products so that these adhere to the nonfunctional requirements for those products.

2.1.1 CONCLUSION

Since the metrics provide a means to analyze the product line scope at multiple levels, including future evolution and the corresponding business value for the scope, it is concluded that the metrics support determining whether the product line scope suffices.

2.2 EVOLVING THE CORE ASSETS

Due to, among others, changing markets and corresponding customer demands, the core assets will have to evolve accordingly. An organization needs to anticipate the direction in which to evolve the core assets. A manner to do this is by analyzing future directions of the market in which an organization operates, future directions of the technology that organizations can use, and future directions of the products. Consequently, by doing so, it can be determined how the product line architecture should evolve in order that it best suits those directions. The former is referred to as sketching strategic scenarios and the latter as sketching architectural scenarios. This is addressed by the metrics that support the corporate-level strategy. In addition, the lowest-level metrics are concerned with the development and maintenance of core assets. If significant differences between the effort required to develop core assets and the effort required to develop similar product-specific assets exists, this might be due to the fact that future evolution of the core assets is not considered (properly).

Since the future evolution of the core assets is anticipated, the documentation of the core assets can be augmented with how the evolution of the core assets can be realized. Examples of how efficient evolving core assets is performed, is represented by the lowest-level metrics.

2.2.1 CONCLUSION

By providing a structured analysis of future evolution, an organization can anticipate the future requirements. Since guesses can be made concerning future evolution, the core asset documentation can be augmented with a guideline to incorporate future requirements.

2.3 BENEFITS FROM USING THE CORE ASSETS

The use of core assets should be more beneficial than developing a similar product-specific asset from scratch. Obviously, if this were not the case, one would be better of not using the software product line as it is. Explicit effort therefore is required to determine whether the software product line approach pays. For instance, the lowest level of metrics addresses the benefits of individual core assets. Both the development of core assets and the use of core assets are addressed. The former is done by comparing the development of core assets with the development of similar assets for specific products, i.e. these assets do not provide variability. The latter is addressed by comparing the tailoring and the integration of a core asset into a product context with the development of a similar product-specific asset from scratch. The second level of metrics provide an integral view on the benefits of individual products. The benefits of all core assets used in a particular product are added and the result of this is compared with all the related costs of using the core assets.

The metrics framework explicates all the costs and benefits over a given period. In addition, it provides a means to quantify intangible benefits by applying scenarios. Typically, intangible benefits are not considered explicitly in a cost-benefit analysis, but merely can be used to persuade an organization for whom the costs and tangible benefits differ little.

2.3.1 CONCLUSION

The operational strategies and the business-level strategy both address the benefits gained from using core assets and compare these with the costs required for both core asset development and product development.

2.4 PROVIDE PRODUCTION PLANS

The definition of software product line states that products are developed in a prescribed way. This is realized by providing the proper production plans. Through these production plans, the product developers are instructed how to tailor and integrate all relevant core assets for their specific product. By doing so, the product developers will experience less trouble using the core assets and the use of core assets likely will be performed more efficiently. The efficiency of using core assets is reflected by the metrics that support both the operational strategies and the business-level of strategy. The same reasoning as in the previous subsection holds. The operational strategies address the benefits and, therefore, the efficiency at the asset level and the business-level strategy addresses the benefits and efficiency at the product level.

2.4.1 CONCLUSION

The availability of proper production plans is not addressed explicitly in the metrics framework. However, if for instance the use of a core asset requires relatively much effort to integrate it into a product context, this might be a sign that the documentation does not suffice. This was for instance observed at Philips Medical Systems.

3. PRODUCT DEVELOPMENT

One business concern related to product development has been identified. This business concern will be addressed in the following subsections. The goal of this subsection is to present why and how the metrics framework supports that business concern.

3.1 REALIZING SOFTWARE REUSE BENEFITS

The main drivers for product developers to adopt core assets are that products need to be developed and maintained as efficient as possible, deployed as soon as possible, and with a high as possible profit margin. As mentioned above, the benefits of using the core assets are explicated. The operational strategies address the benefits of using core assets at the asset level and the business-level addresses the benefits of using core assets at the product level. In addition, since the corporate-level metrics force an organization to explicate assumptions on the future and anticipate regarding future evolutions of the market, the available technologies, and the products, the architecture will evolve accordingly. The product line scope and the product line architecture evolve so that these enable the implementation of requirements that will likely hold given the assumption about the strategic scenarios. Since the product line architecture enables certain features, these features will likely be implemented more efficiently than when an entire architectural transformation would have to take place afterwards.

3.1.1 CONCLUSION

Besides explicating the benefits of using core assets, the metrics framework forces to anticipate future requirements and evolve the product line scope and the corresponding product line architecture accordingly. By doing so, future evolution of the core asset base and the corresponding product base is likely to be performed more efficient.

4. MANAGEMENT

One business concern related to product development has been identified. This business concern will be addressed in the following subsections. The goal of this subsection is to present why and how the metrics framework supports that business concern.

4.1 TOOLS FOR RESOURCES ALLOCATION, COORDINATION, AND SUPERVISION

Management has to be equipped with tools to provide the necessary resources, and coordinate and supervise both core asset development and product development. In addition, management has to set goals and progress towards meeting these goals has to be measured.

By providing overviews of how much effort is required to develop core assets, use core assets, develop product-specific assets, and develop products as a whole, management can assign development capacity to the business units. The metrics altogether provide a means to coordinate and supervise core asset development and product development. If certain processes require suspicious efforts for instance, management knows that they should intervene.

Furthermore, management has to set goals. The metrics should support measuring progress towards meeting the goals. These goals should encompass not only short-term planning, but long-term investment analyses also. In addition, the long-term investment analyses need to be assessed, for instance by using cost-benefit analyses of previous products. By doing so, lessons can be learned from past investments analyses and future investment analyses can be performed more accurately.

4.1.1 CONCLUSION

All levels of metrics provide insight in both core asset development and product development. In addition, management is not only involved in the current practices, but by explicating the architectural scenarios and the strategic scenarios, management is involved in investment analyses also.

5. PROS AND CONS

The entire framework of metrics provides a means to measure multiple relevant aspects of a software product line initiative. In chapter 12, different business concerns for the three software product line activities were identified. Chapter 13, in turn, introduced three levels of corporate strategy and provided three levels of metrics that each support a strategy level. In the previous three sections, it was argued how the three levels of metrics address those business concerns.

The metrics framework addresses three levels of strategy. It, however, does not require that all levels of metrics are collected individually. For instance, the second level of metrics, the business-level strategy, leans extensively on the first level of metrics. For instance, when the savings on the development costs resulting from the use of core assets need to be derived, the metrics addressing the operational strategies provide all the required data at the asset level, e.g. the metrics integration effort.

In [11], it was stated that an organization that is mature with respect to their software product line considers the long-term health of the software product line as their major concern. The long-term health of a software product line is for instance reflected in the product line scope. If the scope is defined adequately, the development of core assets is performed effectively, as is the use of the core assets. The third strategy level, the corporate-level, is concerned with the future evolution of the software product line and, therefore, with the evolution of the product line scope also. Based on expectations regarding future evolutions of the market, technologies, and products, the scope is evolved accordingly. By anticipating on future requirements, the product line architecture enables certain features to be incorporated in the core assets.

A drawback of the metrics framework or, even stronger, a drawback of any set of metrics is that people tend to manipulate the results [38]. In order to deal with this, it should be explicated what the purpose of the metrics framework is. A manner to avoid this is to introduce the use of metrics in three steps [30]. First, set goals to strive for. Second, commit the developers to these goals. Examples to do so are to provide monetary rewards or other rewards when goals are met, or even when progress towards meeting these goals is achieved. Finally, the developers should be supported by higher management, for instance by providing support. In addition, the obtained data should be provided as feedback to both product developers and core asset developers. This way, they are committed when things are progressing or are alarmed when progress is stagnating. The most important of all is that the metrics are in no way used to

punish the developers. If the metrics lead to inaccurate conclusions, there is no purpose in using the metrics at all.

Finally, since the metrics should not require too much effort, these are set up in a generic manner. Therefore, the expertise of the developers and management is required to discover which problems are actually occurring. For instance, are the relatively high development costs of core assets due to a too high level of (architectural) abstraction and encompass these assets too much functionality, or are the high development costs due to a too large defined product line scope.

6. THE CONTRIBUTION

Many of the software product line metrics found in the literature were aimed at software reuse in general. In addition, there was no unified terminology regarding these metrics. This thesis proposes an overview of metrics, from short-term to long-term visions, making use of the terminology provided by the Software Engineering Institute. Since a software product line approach provides a means of institutionalizing software reuse, it seemed sensible to apply, and where necessary, adjust the software product line metrics for use in a software product line context rather than just a software reuse context. Also, the applicability of the metrics to the three main software product line activities has been made explicit.

The first adjustment necessary for the transformation from software product line metrics into software product line metrics was the augmentation of the reuse level. A distinction was made, which seemed appropriate for a software product line approach. Traditionally, only the software reuse level was suggested. When doing the research for this thesis, it seemed appropriate to make a distinction in reuse levels. First, the traditional reuse level, i.e. the part of a product coming from reuse divided by the total size of the product, is an indicator of how much of the core assets is used in a particular product. However, in a software product line context, it might be the case that a certain product cannot make use of core assets extensively, since the product requirements do not allow for this. Therefore, this thesis introduced the relative reuse level. The relative reuse level states what percentage of the core assets applicable for a product is actually being used for the development of that product.

The main contribution of this thesis is that a framework of metrics is provided. That framework is composed of three levels that each addresses a strategy level of corporate strategy. The metrics framework was constructed in such a manner that a metrics level provides input to the next metric level. For instance, the cost-benefit analyses can be used to assess previous investment analyses.

7. FUTURE WORK

This chapter merely provides an informal approach to validating that the metrics framework indeed addresses all business concerns. Formal validation therefore is required to ensure that all the needs of all three activities are taken into consideration.

The metrics framework needs to be augmented with a mechanism to include other assets besides source code also, e.g. architectural solutions. The assets can be taken into account, but no method for quantifying the resulting benefits is provided. Above, it has been suggested to use the notion of effort for all assets, but more complex methods are likely to lead to more accurate results.

As mentioned above, it is not yet clear if the chosen generic nature of the metrics framework is too generic. To validate this, the metrics framework should be applied in practice and the corresponding problems should be discovered through the framework. In addition, a reference list for the metrics could be initiated. For instance, the average values for the integration effort can be gathered by multiple case studies and, from these values, a reference value for that metric can be obtained. That way, an organization knows when the integration effort is relatively high.

8. CONCLUSION

Based on the above, the proposed metrics framework indeed addresses all the identified business concerns of software product line activities. Furthermore, the metrics require little effort and, therefore, can easily be incorporated in the typical software product line processes. The main purpose of the metrics framework is to maintain the long-term health of the software product line. It therefore is concerned both with determining the product line scope and the product line architecture accordingly in order to incorporate future requirements, and identify weaknesses of any of the development processes. The main drawback regarding the latter is that identifying these weaknesses requires the expertise and experience of those involved also.

Part five
Appendices

This part is composed of the appendices. The first appendix contains the reference list. The second appendix consists of the glossary. The third appendix contains the index. The final appendix provides the second questionnaire.

APPENDIX A
REFERENCES

[1] America, P., Hammer, D., Ionita, M.T., Obbink, H., Rommes, E., "Scenario-Based Decision Making for Architectural Variability in Product Families", Proceedings of the 3rd International Conference on Software Product Lines, Springer.

[2] Atkinson, A., Banker, R., Kaplan, R., Young, S.M., "Management Accounting 3rd Edition", Pearson Education.

[3] Bass, L., Clements, P. and Kazmar, R.: "Software Architecture in Practice", Addison-Wesley.

[4] Besanko, D., Dranove, D., Shanley, M.: "Economies of Strategy 2nd Edition", John Wiley & Sons, Inc.

[5] Bosch, J., "Design & Use of Software Architectures: Adopting and evolving a product line approach", Addison-Wesley.

[6] Bosch, J., "On the development of Software Product-Family Components", Proceedings of the 3rd International Conference on Software Product Lines.

[7] Böckle, G., Clements, P., McGregor, J.D., Muthig, D., Schmid, K., "Calculating ROI for Software Product Lines", IEEE Software, Vol. 21, Issue 3.

[8] Carballo, J.-A., Belluominu, W., Montoye, R., Cohn, D., "A reward-based economic model for component reuse", International Workshop on Reuse Economics.

[9] Chambers, D.R., Lacey, N.J., "Modern Corporate Finance: Theory & Practice 4th Edition", Hayden McNeil Publishing.

[10] Clements, P.C., McGregor, J.D., Cohen, S.G., "The Structured Intuitive Model for Product Line Economics (SIMPLE)", http://www.sei.cmu.edu/publications/documents/05.reports-/05tr003.html.

[11] Clements, P.C., Northrop, L., "Software Product Lines: Practices and Patterns", Addison-Wesley.

[12] Cohen, S.G., "Predicting When Product Line Investment Pays", http://www.sei.cmu.edu/-publications/documents/03.reports/03tn017.html.

[13] Devanbu, P., Karstu, S., Melo, W., Thomas, W., "Analytical and Empirical Evaluation of Software Reuse Metrics", Proceedings of 18th International Conference on Software Engineering.

[14] Fafchamps, D., Organizational Factors and Reuse, IEEE Software, Vol. 11, Issue 5.

[15] Favaro, J., "A Comparison of Approaches to Reuse Investment Analysis", Proceedings of the 4th International Conference on Software Reusability.

[16] Fichman, R.G., Kemerer, C.F., "Activity Based Costing for Component-Based Software Development", Information Technology and Management, Vol. 3, Issue 1-2.

[17] Fichman, R.G., Kemerer, F., "Incentive compatibility and systematic software reuse", Journal of Systems and Software, Vol. 57, Issue 1.

[18] Frakes, W., Terry, C., "Software Reuse: Metrics and Models", ACM Computing Surveys, Vol. 28, Issue 2.

[19] Frakes, W.B., Succi, G., "An industrial study of reuse, quality, and productivity", The Journal of Systems and Software, Vol. 57, Issue 2.

[20] Gaffney Jr., J.E., Cruickshank, R.D., "A General Economics Model of Software Reuse", Proceedings of the 14th International Conference on Software Engineering.

[21] Gomaa, H., "Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures", Addison-Wesley Professional.

[22] Grune, D., Bal, H.E., Jacobs, C.J.H., Langendoen, K.G., "Modern Compiler Design", Wiley.

[23] http://medical.nema.org/

[24] http://www.bcg.com/

[25] http://www.medical.philips.com/

[26] http://www.sei.cmu.edu/productlines/framework.html

[27] http://www.softwareproductlines.com/benefits/benefits.html

[28] Jacobson, I., Griss, M., Jonsson, P., "Software Reuse: Architecture, Process and Organization for Business Success", Addison-Wesley.

[29] Johnson, G., Scholes, K., Whittington, R., "Exploring Corporate Strategy: Text and Cases 7th Edition", Prentice Hall.

[30] Kreitner, R., Kinicki, A., "Organizational behaviour 2nd European Edition", McGraw-Hill Education.

[31] Kruchten, P., "Architectural Blueprints—The "4+1" View Model of Software Architecture", IEEE Software, Vol. 12, Issue 6.

[32] Lim, W.C., "Managing Software Reuse: A comprehensive Guide to Strategically Reengineering the Organization for Reusable Components", Prentice Hall.

[33] Malan, R., Wentzel, K., "Economics of Software Reuse Revisited", Proceedings of the 3rd Irvine Software Composium.

[34] Margono, J., Rhoads, T.E., "Software reuse economics: cost-benefit analysis on a large-scale Ada project", Proceedings of the 14th international conference on Software engineering.

[35] Menzies, T., Di Stefano, J.S., More Success and Failure Factors in Software Reuse, IEEE Trans-actions on Software Engineering, Vol. 29, No. 5.

[36] Mili, A., Fowler Chmiel, S., Gottumukkala, R., Zhang, L., "An Integrated Cost Model for Software Reuse", Proceedings of the 22nd international conference on Software engineering.

[37] Morisio, M., Ezran, M., Tully, C., "Success and Failure Factors in Software Reuse", IEEE Transactions, Vol. 28, Issue 4.

[38] Nicholas, J.M., "Competitive Manufacturing Management: Continuous Improvement, Lean Production, Customer-Focused Quality", Irwin McGraw-Hill.

[39] Peterson, D., "Economics of Software Product Lines", Proceedings of the 5th International Workshop on Product Family Engineering, Springer.

[40] Poulin, J.S., "Measuring Software Reuse: Principles, Practices and Economic Models", Addison-Wesley.

[41] Schmid, K., "Reuse Economics from a Product Line Point of View", International Workshop on Reuse Economics.

[42] Schmietendorf, A., Dimitrov, E., Dumke, R., Foltin, E., Wipprecht, M., "Conception and Experience of Metrics-Based Software Reuse in Practice", Proceedings of the 14th International Workshop on Statistical Modelling.

[43] Sommerville, I.: "Software Engineering 6th Edition", Addison-Wesley.

[44] Svahnberg, M., van Gurp, J., Bosch, J., "A Taxonomy of Variability Realization Techniques", accepted for publication in Software: Practice and Experience.

[45] Tomer, A., Goldin, L., Kuflik, T., Kimchi, E., Schach, S.R., "Evaluating Software Reuse Alternatives: A Model and Its Application to an Industrial Case Study", IEEE Transactions on Software Engineering, Vol. 30, Issue 9.

[46] van der Linden, F., Bosch, J., Kamsties, E., Känsälä, K., Obbink, H., "Software Product Family Evaluation", Proceedings of the 3rd International Conference on Software Product Lines.

[47] van Gurp, J., Bosch, J., "Design Erosion: Problems & Causes", Journal of Systems & Software, Vol. 61, Issue 2.

[48] Wappler, T., Remember the Basics: Key Success Factors for Launching and Institutionalizing a Software Product Line, Software Product Lines: Proceedings of the 1st Software Product Line Conference.

[49] Wesselius, J.H., "Modeling Architectural Value: Cash Flow, Time and Uncertainty", accepted for the 9th International Software Product Line conference.

APPENDIX B
QUESTIONNAIRE ON FUNDING MODEL

This appendix presents the questionnaire, which was used to validate the ideas concerning the most appropriate funding model.

Introduction

Funding is a key issue when institutionalizing a software reuse program in an organization. A funding model allocates resources for the development of reusable assets, e.g. reusable components. To illustrate some aspects of a funding model, the Philips Medical Systems funding model will be discussed briefly.

The funding model used by Philips Medical Systems has two typical characteristics. The first characteristic is that business units have to pay up-front on a yearly base. The second characteristic is that the Philips Medical Systems funding model makes use of two different types of users, namely users that use the reusable assets in product releases and users that use the reusable assets for prototyping and/or feasibility study. The former type pays double the amount as does the latter type. The yearly budgets are spread across all the business units with respect to their expected usage of reusable assets.

This part of the questionnaire works as follows. In the following table, you are asked to provide an answer between 1 and 5. These values represent (1) absolutely disagree, (2) disagree, (3) neutral, (4) agree and (5) absolutely agree.

1) The current MIP funding model suffices.
2) The current MIP funding model allocates costs in a transparent manner.
3) In order for a funding model to be effective, it should be transparent.
4) The current MIP funding model is fair in the sense that everyone pays a proper share of the costs.
5) The business units with the highest profits should pay the highest contributions to the funding of MIP, regardless their usage of the MIP assets.
6) A business unit should pay every time for reusable assets when they sell a product that uses these reusable assets.
7) A business unit should pay once (per year) for a reusable asset, a license for that asset is then bought and it may be used in any product of that business line.
8) Up-front payment for predicted use of reusable assets, as is done with the MIP funding model, is fair.
9) The more reusable assets one uses, the more beneficial it should be. For rewards, see questions 12 – 18.
10) The newer versions of reusable assets one adopts, e.g. newer MIP releases, the more beneficial it should be. For rewards, see questions 12 – 18.
11) Act as a first user of a reusable asset should be made more beneficial, since the first user has to deal with the childhood illnesses. For rewards, see questions 12 – 20.
12) Receiving discount on the funding of reusable assets is an effective reward regarding stimulating software reuse.
13) Receiving additional integration support is an effective reward regarding stimulating software reuse.

14) Providing higher ranking to your PRQ's, thus more likely that your requirement is included in the next release, is an effective reward regarding stimulating software reuse.
15) Receiving financial rewards stimulates a business unit to develop reusable assets rather than develop product-specific assets or reengineer a product-specific asset into a reusable asset.
16) Receiving additional developers from the software reuse team stimulates a business unit to develop reusable assets rather than develop product-specific assets or reengineer a product-specific asset into a reusable asset.
17) Receiving additional integration support from the reuse team stimulates a business unit to develop reusable assets rather than develop product-specific assets or reengineer a product-specific asset into a reusable asset.
18) Receiving maintenance support sooner, e.g. bug-fixing, stimulates a business unit to develop reusable assets rather than develop product-specific assets or reengineer a product-specific asset into a reusable asset.
19) Knowing that future maintenance of a reusable asset developed by a business unit, will be taken care of by the reuse group stimulates a business unit to develop reusable assets rather than develop product-specific assets or reengineer a product-specific asset into a reusable asset.
20) Having PRQ's ranked higher stimulates a business unit to develop reusable assets rather than develop product-specific assets or reengineer a product-specific asset into a reusable asset.
21) If new features are only provided for new releases of reusable assets, a business unit adopts the newer versions of these assets sooner.

What characteristic, in your opinion, a funding model definitely should have in order to stimulate the usage of reusable assets, whether it be one of the aforementioned ones or not?

What kind of reward would stimulate business units to develop reusable assets rather than product-specific assets, whether it be one of the aforementioned ones or not? Please fill in the grey boxes below.