Petri I. Salonen

EVALUATION OF A PRODUCT PLATFORM STRATEGY FOR ANALYTICAL APPLICATION SOFTWARE

HELSINKI SCHOOL OF ECONOMICS

ACTA UNIVERSITATIS OECONOMICAE HELSINGIENSIS

A-239

Petri I. Salonen

EVALUATION OF A PRODUCT PLATFORM STRATEGY FOR ANALYTICAL APPLICATION SOFTWARE

HELSINKI SCHOOL OF ECONOMICS

ACTA UNIVERSITATIS OECONOMICAE HELSINGIENSIS

A-239

ISSN 1237-556X
ISBN 951-791-867-4
ISBN 951-791-868-2 (Electronic dissertation)

Helsinki School of Economics -
HeSE print 2004

To Rita and her unconditional love.

Never, give in! Never give in! Never, never, never, never…. In nothing great or small, large or petty, never give in except to convictions or honor and good sense!

------ Sir Winston Churchill

Abstract

The challenge for software business executives is to bring new software products to the market rapidly, using both a software development approach that enables effective delivery of products and a robust software product architecture that addresses the needs of selected market segments. This study addresses the development of analytical application software solutions that are the foundation for a decision support solution, providing to end user organization executive management the needed tools to track critical key performance indicators using technologies such as digital dashboard software.

Unfortunately, executive management has to make several decisions long before the actual software product is delivered, and some of these decisions can have a long-lasting impact on both future software development and the market segmentation of the software product. To alleviate the challenge of long-term strategic software product development, researchers have applied the concept of the product platform from mechanical engineering to software development, enabling the utilization of a common core product platform that becomes the foundation for derivative product development within a product family. The product platform concept has been demonstrated within several industries, for example, the automobile industry (Ford automobiles) and the electronics industry (Hewlett Packard printers). Existing software related product platform literature does not address the practical implications of building software products using a product platform approach.

This study introduces software product line engineering as a viable alternative foundation for software product family development using the product platform approach. Its aim is to identify an optimal analytical application software architecture that becomes the foundation for long-term derivative software development using the same common core (the product platform) across different derivative products for given market segments. The product platform development approach has the aim, contrary to that of the traditional software development approach, of maximizing the revenue (and not of minimizing the cost) that can be leveraged from a product platform using a product architecture that is specifically designed to be common to all selected market segments. The study also introduces six different alignment perspectives that demonstrate the relationships between the selected product architecture, market segment, and technology dimensions. Each alignment perspective has characteristics which depend on the emphasis given any of the three dimensions. When setting long-term product development strategy, each of these dimensions must be carefully evaluated against the others before management makes a decision on any of the dimensions. Negligence in this evaluation could result in a disconnect between the dimensions, with long-lasting impact. From our analysis, it is evident that each alignment perspective can be specific to each software vendor due to the characteristics of the vendor, such as its core competence in technology, its software application domain, and its selected market segment.

The results of this longitudinal (ten year) single-case study demonstrate the use of the product platform concepts and alignment perspectives introduced herein. These alignment perspectives help to show how changes in product architecture, market segmentation, or technology can impact a software vendor's product development effectiveness, and how executive management can assess the impact and reasons for these changes. The empirical evidence reinforces the researcher's view that a software vendor can achieve significant benefits using the product platform concept in its software development. The study also demonstrates how technology selections can impact future market segmentation strategies for a software vendor and how these selections can impact software development.

KEYWORDS: DATA WAREHOUSING, ANALYTICAL APPLICATION SOFTWARE, PRODUCT PLATFORM, SOFTWARE APPLICATION FRAME STRATEGY, SOFTWARE PRODUCT LINE ENGINEERING, SOFTWARE BUSINESS

Acknowledgements

It all started in the most eastern Swedish-speaking elementary school in the world. Our school was very small and competitive, and it became evident that nothing comes for free in this world and the winners are those who fight for their future. My parents, Leena-Ilkka Salonen and Seppo Olavi Salonen, gave me an understanding of why one has to go to school, and I am very grateful that they gave me the chance to do what I did. I knew already in 1989, when getting my master's degree in Accounting from the Helsinki Swedish School of Economics and Administration, that I would return to school once I had accumulated some practical experience in the software industry. Mr. Kyösti Anttonen believed in me and hired me into Unic Oy, and I am still on that road. Thank you, Kyösti, for the opportunity you gave me! With the support of Jorma Hänninen (Chief Executive Officer at the time for Unic Oy), I continued with my studies at the Helsinki School of Economics. I am greatly indebted to the vision Jorma had, that he felt the importance of continued education. I wish all CEO's could be as forward looking as Jorma was in the early 1990s. I promise to continue in the tradition he established.

I am forever indebted to my mentor and always-helpful Professor, Markku Sääksjärvi, from the Helsinki School of Economics, for providing me with the keys to the academic world. Professor Sääksjärvi did not give up on me, even though my dissertation work took quite a while due to my other responsibilities as Chief Executive Officer for an international software company. To make things even more challenging for my family, we decided to immigrate to United States in the late 1990s. Even then, Professor Sääksjärvi did not give up on me. Only Professor Sääksjärvi and I know what took place during those years. It was certainly not easy for anybody.

I am also very grateful to all of the employees of the case study organization. I did not have any understanding of IBM midrange environments, but Mr. Jorma Vesterinen, Mr. Markku Riekkinen, and others have convinced me that IBM has created a truly great piece of technology. I would also like to thank Mr. Mika Rihtilä and Ms. Hannele Piri for the numerous hours they spent collecting financial information that did not previously exist. My longtime colleague Mr. Olli-Pekka Siikarla gave me the foundation for understanding the importance of software architecture during the long nights and weekends that we spent together. This time has had a tremendous impact on my life. Timo Sinisalmi showed me what it takes to become successful in product development by keeping things simple.

At the worst time in my research, Dr. Liisa Savunen of the Academy of Finland led me back to the road and gave me several pieces of practical advice that I will be sure to pass on if I ever coach other Ph.D. students. Also, the feedback from Dr. Veikko Seppänen from Oulu University and Dr. Jyrki Kontio of the Helsinki University of Technology provided me with valuable insight during the process of finalizing my work. I am also indebted to Dr. Alan Chmura of the University of Wyoming, who was willing to take on the huge task of proofreading my work. This type of friendship is hard to find anywhere!

Without the understanding of my dear wife Rita Salonen and my children, Sebastian and Daniela Salonen, this work would never have been completed, period and end of discussion. The ups and downs of doctoral work have taken their toll on our lives, but my wife kept me "hanging in there," even when I felt that the work would never be finished. It is unbelievable how much resilience my family has shown during the years when "papsu" had to work on this dissertation in his Trophy Club home office and Kalevankatu 52 apartment. Even our family dog, Karva, knew enough to stay away and keep quiet when I was working. Rita, one of these days we will travel around the world as we envisioned while we were still in Finland – that is a promise!

Finally, please forgive me for not mentioning the hundreds of different people I have met during the years who have influenced my way of thinking. You all deserve to be mentioned, but the limited space does not allow me to do that.

Petri I. Salonen
Dallas, Texas and Helsinki, Finland
May 2004

Abbreviations

AFE	Application Family Engineering
API	Application Program Interface
APPC	Advanced Program-to-Program Communications
AS/400	Application System/400
ASE	Application System Engineering
CBSE	Component-Based Software Engineering
CEO	Chief Executive Officer
CFO	Chief Financial Officer
CIF	Corporate Information Factory
CMM	Capability Maturity Model
COM	Component Object Model
CoPAM	Component-oriented Platform Architecting Method
CORBA	Common Object Request Broker Architecture
COTS	Commercial-Off-The-Shelf Software
CSE	Component System Engineering
CSF	Critical Success Factor
CSV	Core Strategic Vision Framework

DCOM	Distributed Component Object Model
DLL	Dynamically Linked Library
DOLAP	Desktop On-Line Analytical Processing
DSS	Decision Support System
EIS	Executive Information System
ERP	Enterprise Resource Planning
ETL	Extraction, Transformation and Load
FeatuRSEB	Feature Reuse-driven Software Engineering
FODA	Feature-Oriented Domain Analysis
FORM	Feature-Oriented Reuse Method
HOLAP	Hybrid On-Line Analytical Processing
IC	Integrated Circuit
IDC	International Data Corporation
IDL	Interface Definition Language
ISV	Independent Software Vendor
IT	Information Technology
KLOC	Thousand (kilo-) Lines of code
KobrA	Komponentenbasierte Anwendungsentwicklung (from German language)
KPI	Key Performance Indicator
MOLAP	Multidimensional On-Line Analytical Processing
MVC	Model-View-Controller
ODBC	Open DataBase Connectivity
ODS	Operational Data Store
OLAP	On-Line Analytical Processing
OLE DB for OLAP	Microsoft OLE DB for Analytical Processing (OLAP) – set of objects and interfaces that extends the ability of OLE DB to provide access to multidimensional data stores (OLAP applications)
OLTP	On-Line Transaction Processing
OOA&D	Object-Oriented Analysis and Design
PDA	Personal Digital Assistant
R&D	Research and Development
RAD	Rapid Application Development
ROLAP	Relational On-Line Analytical Processing
RPG	Report Program Generator (iSeries/400 based programming language)
SEI	Software Engineering Institute
SNA	Systems Network Architecture (IBM technology)
SQL	Structured Query Language
TDWI	Data Warehousing Institute
UML	Unified Modeling Language

Figures and Tables

1. INTRODUCTION

Software development has become a major part of our lives, and computer based systems pervade today's society (Miller and Ebert, 2002). Hardware and software systems have become increasingly integrated with each other, and hardware applications that were formerly not controlled with software are more or less driven by software in today's world. These changes have increased the pressure for software organizations to find more effective ways to build and maintain software. The software engineering world has evolved during the last twenty years from traditional waterfall development methodologies to highly componentized software development methodologies. Software organizations are studying the manufacture of physical products to learn how software development can achieve better leverage. McGregor et al. (2002) give examples of Eli Whitney and Henry Ford and their approach of interchangeable parts. Another view of this is the concept of modularity, wherein products are assembled from a set of modules. Each module can be complex internally, but with clearly defined interfaces, each module can be linked with other modules in different combinations. An organization "can use modularity to produce a wide variety of products in extremely short lead times" (O'Grady, 1999). Modular products require an adaptable software architecture that forms the foundation and structure for a product family. According to Cusumano (2004), modularity can be defined as "a subset of functionality that is smaller than the whole product and that the designers can isolate from other small chunks of functionality and, to some degree, test as a separate unit."

Modularity can also be tied to a promising product development approach – product platform theory – which has been identified within mechanical engineering. Product platform theory emerged in the early 90s, introduced by Wheelwright and Clark (1992) and Meyer and Utterback (1993), with several other follow-up articles (Meyer and Lopez, 1995; Meyer and Zack, 1996; Meyer et al., 1997; Meyer and Seliger, 1998) and books (Meyer and Lehnerd, 1997; McGrath, 1995; 2001). This theory has its origin in mechanical engineering. According to Sääksjärvi (2002) the theory proposed product architecture as the foundation for effective leverage of implemented underlying technology in a series of products in a product family. Sääksjärvi (1998; 2002) reviewed product platform theory from the software development perspective, resulting in a framework (Software Application Frame Strategy) which is specifically adapted to the software products domain. This framework considers other implications or dimensions in software development that a software vendor must consider, such as technology selection and market segmentation. The Software Application Frame Strategy framework from Sääksjärvi (2002) is not bound to any specific software domain.

Our aim in this study is to research practical implications with rich insight via a longitudinal single-case study company into how the analytical application software domain could utilize this type of software development approach. Analytical application solutions were first introduced in 1997 by International Data Corporation in an article describing analytical application solution functionality. A follow-up article to this article was published in the Journal of Data Warehousing in the fall of 1998, where Morris (1998) described the purpose of an analytical application and the intended use and required functionality. Gleason (1998) took the next step to discuss whether an analytical application should be bought or built and what type of functional components can be found in an analytical application. Surgan (2000) discussed and analyzed in a follow-up article the different time eras of decision support. Her claim in the article is that the data warehousing market has matured and that the next logical step in the decision support domain would be to use and support solutions that have an end-to-end integrated data warehouse solution with pre-built functionality for given vertical markets. The Data Warehousing Institute (TDWI) has increased the number of studies of the use of analytical application software solutions from the end user organization perspective. A recent broad definition of analytical application software was released by Eckerson (2002) as follows:

> "An analytical application is a domain-specific solution that enables all types of business users to access, analyze, and act on information in the context of the business processes and tasks they manage. The solution leverages data warehouses and analytical tools and integrates with operational systems."

These analytical applications include predefined business metrics for a selected vertical market segment and cover typically 80% of the business analytics and processes that a given vertical industry segment requires (Morris, 1998). This type of transition from custom-built data warehousing environments to prepackaged analytical applications changes the focus from technology development to contents-driven development, where given vertical market segments with corresponding business processes and key metrics must be implemented into the analytical application software solution. This change requires an adaptable product architecture that enables a software vendor to address given market segments using a common architectural core that is optimized to cover these market segments. We will limit the discussion of software architecture and definitions to issues and concepts relating to product platform theory.

We have selected Ulrich and Eppinger's (1995) definition of product architecture, which relates to the concept of product platforms. According to Ulrich and Eppinger (ibid), an architecture of a product is "the scheme by which the functional elements of the product are arranged into physical chunks and by which the chunks interact." In a manner similar to O'Grady (1999) and Cusumano (2004), Ulrich and Eppinger (ibid) emphasize the importance of modularity in an architecture and the importance of how each module within the architecture interacts according to predefined and well defined interfaces. We recognize the importance of software architecture when using the product platform approach in software development and therefore we will explore different domain-specific architectural models for analytical application software solutions and different implementation architectural styles that can be found in software architecture literature. We will exclude discussion of software architecture with respect to its different definitions that can be found in the literature. According to Malveau and Mowbray (2004) there are several different schools of thought for software architecture, and these are outside the bounds of our research. The Software Engineering Institute (SEI) has conducted considerable of research into software architecture (Clements and Northrop, 2002; Clements et al., 2002; Bass et al., 2003). Their web site (SEI, 2004) has a broad collection of software architecture related definitions and literature with case studies.

The findings of existing software related product platform literature do not include discussion of the practical implementation of software products using the product platform approach. Sääkjärvi (1998, 2002) introduced his Software Application Frame Strategy framework without practical implications for any specific domain. The aim of this study is to include software engineering related issues when implementing software products using the product platform approach. A closely related software engineering approach with aims similar to those of the software product platform approach is that of large-scale reuse using software product line engineering as its implementation approach in derivative software development.

1.1 Research Problem

An emerging concept from mechanical engineering, introduced in the early 1990's, has recently been reviewed for its applicability for use in the development of software products (Meyer and Seliger, 1998; Sääksjärvi, 1998; 2002). The framework – the Software Application Frame Strategy – proposed by Sääksjärvi (2002) presents without empirical consideration three generic alignment perspectives that a software vendor can utilize in development of software products. Our aim in this study is to evaluate this framework in the light of analytical application software solutions, with the intent of describing the practical implications of how this type of framework can be used in analytical application solution development. Existing software-related literature does not specifically address the complexities and approaches of software development using the product platform concept. Our aim is to describe closely related software engineering concepts that can be linked to the product platform concept.

Part of software development, specifically derivative software development, is to define an adaptive domain-specific software architecture that is the foundation for implementation of an optimal architectural construct which is in turn the foundation for an analytical application solution. This architectural construct – later defined as an analytical application frame – must be adaptable to changes in market conditions (market segmentation) and technological changes that could take place either in end user Information Technology (IT) infrastructure environments or underlying technologies that have to be selected as the foundation for any software domain (such as the operating system, the hardware environment, the database management system, etc.).

The aim of a product platform is to enable implementation of reuse in large using technologies like component-based software engineering. Reuse as defined in the software engineering literature is not sufficient to implement software related product platforms. The main aim of software product platform related theory is to maximize revenues by reusing a large common architecture core within different market segments, while the aim of reuse, according to the literature, is typically to minimize costs when implementing software solutions. Therefore, the aim of product platform development is different and the strategic choices must be made at the executive management level and not within individual product development, marketing, or technology teams within a software organization. There is plenty of research in software reuse, component-based software development, and software product line development, but none of these corresponds directly to the concept of product platform theory other than the approach of having a modular architecture with modules with well defined interfaces. We will exclude from this study discussion of reuse within software development and associated economic metrics, as there is plenty of research within these domains (Mili et al., 2002).

Analytical application solutions can be traced back to the decision support domain with data warehousing, On-Line Analytical Processing (OLAP), and Executive Information Systems (EIS) architectures. The evolution of each of these has set the boundaries for what current analytical application software solutions can provide to end user organizations. For a software vendor to be able to recognize the limitations, it has to build an optimal common architectural construct that can be the basis for derivative software products. This entity can not be too large, as it would lose its flexibility. Selected technologies in the software solution must be restricted and not cover all the different possible combinations that a software vendor is trying to reach from market segmentation perspective. Another perspective to evaluate the success of software development using the product platform approach is to measure economical metrics that provide to software organizations their effectiveness and efficiency measures ex-post. We will evaluate and apply these metrics in this study and provide discussion of the relevance and use of these specifically using a longitudinal single-case study vendor within the analytical application software solution domain.

Existing analytical application software literature is more or less based on the perspective of end user organizations (traditional IS research). Our approach will be from the analytical application software vendor's perspective. Our study will include three interrelated domains, with emphasis on the analytical application software domain. The first domain is a literature search of product platform related literature with corresponding and related software engineering theories. The second domain is an analysis of analytical application software with respect to related technologies and domain-specific architectural models (data warehouse architectures). The third domain is more software engineering related, with discussion of how software product line/family development can be used to implement common software assets using different derivative techniques. Based on the assumptions presented above, our main research objectives are as follows:

RQ 1: How can a software vendor apply the product platform approach to its software business and development of software products?
RQ 2: What types of generic software application frame architectures can be identified for analytical application software solutions?
RQ 3: How can software vendors balance their software product architectures when changes take place in marketing and/or technology selections?
RQ 4: How can effectiveness criteria change when applying the product platform approach in analytical application software development?

The aim of the first research objective is to explore existing product platform theory related literature specifically within software development, and to compare this with the development of physical products using a product platform concept. This literature study will enable us to have a better understanding of following topics:

- What is known in existing product platform literature, both in physical and in software product development, and what types of differences can be identified between these two different product types (physical vs. software products)?
- How do available software related platform literatures explain the role of IT infrastructure and possible dependences between runtime- and development-time environments?
- How does the existing software related platform literature define the process of building a platform in the software products domain?
- How does the existing software development community recognize product platform development and the respective product line development of software products?

Our aim in the second research objective is to explore different data warehousing architectural models that are potential models of the foundation for an analytical application solution and what type of architectural model supports an optimal architectural construct that can be reused across different derivative products within an analytical application solution.

The third research question explores how different alignment perspectives between product architecture, technology, and market segmentation in the implementation will impact the development of analytical application software solutions. We will discuss how product platform theory can be beneficial in analytical application software development and how success can be measured using different metrics defined within the product platform literature.

The final research question aims to explain how software development using the product platform approach could change based on the effectiveness and efficiency criteria that are measured ex-post. These measurements are demonstrated via a longitudinal interpretative single-case study. We will also discuss how product platform approach changes traditional process-oriented software product development and how analytical application software vendors can build a product platform for their derivative products.

1.2 Contributions

Our study was divided into four different research questions to be responded to. Our first research question explores how software vendors can apply the software related product platform approach in the development of software products. Prior work is mostly based on generic software products, using the product platform approach without addressing any software application domain specifically. Existing software-related product platform literature introduced the product platform concept to software development without expressing constructive advice for the implementation approaches a software vendor can take when building software products. This study introduces software engineering related concepts such as software product line engineering with corresponding domain and application engineering and component-based engineering as possible implementation approaches using the software related product platform approach.

Secondly, our study provides valuable information showing how analytical application software has evolved during the years from the architectural, technological, and functional perspectives. This evolution includes a comparison of different data warehousing and OLAP architectural models with corresponding information distribution technologies such as EIS. Our discussion of different architectural models provides the foundation for building an analytical application architecture with a corresponding optimal architectural construct (analytical application frame) that can be reused across different vertical and horizontal market segments. This discussion is needed to understand how each technological choice, and each market segmentation selection, will impact product architecture. Without this type of alignment perspective, a software vendor could potentially run into future market segmentation difficulties, as some of these selections could be in conflict with the overall strategy for the software vendor. Part of defining an analytical application frame is to define the needed functionality that is going to be the same for any selected market segment. We need to analyze the functionality that is expected from an analytical application software solution.

Thirdly, our study introduces a strategic framework – Analytical Application Frame Strategy – that is a refined framework of the Software Application Frame Strategy introduced by Sääksjärvi (2002). Our framework reflects the requirements of analytical application software solutions with six corresponding alignment perspectives that a software vendor can take when implementing its software development strategy in conjunction with business strategy.

Fourthly, we will demonstrate how product line engineering can be one good alternative approach to building analytical application software solutions with derivative products that create a product family. We will also discuss different techniques that an analytical application software vendor can utilize when creating derivative software solutions using the product platform approach. We will demonstrate different implementation mechanisms for application frames using software product line engineering with corresponding software assets. This type of discussion has not been published before and will therefore provide a foundation when defining a strategy for derivative software product development.

Finally, the findings from the existing software related platform literature, analytical application software literature, and software engineering literature are used to obtain ideas, characterize the construct, and provide feedback for improvement in a longitudinal interpretative single-case study of a decision support organization undertaking analytical application solution development. The benefit of this case study is that it demonstrates the use of the Analytical Application Frame Strategy framework and its corresponding alignment perspectives. We also analyze the impact the use of the Analytical Application Frame Strategy could have on future development of the software products within the case study company.

1.3 Structure of the Dissertation

To support the objectives for this study, we have divided our research into eight consecutive chapters, where by one chapter builds the foundation for the following one:

Chapter 1: Introduction
Chapter 2: Product Platform Theory
Chapter 3: Analytical Application Frame Architecture Development
Chapter 4: Defining a Balance Between Sub-Strategies in Analytical Application Software
Chapter 5: Implementation Approaches for a Software Aapplication Frame
Chapter 6: Empirical Research Design
Chapter 7: Analytical Application Development in a Case Study Company
Chapter 8: Key Findings and Summary

The first chapter sets the environment for our research with an explanation of the research objectives and research methodology. We summarize our findings and note the contribution of the study.

Chapter 2 is a literature study of the existing product platform literature with corresponding references to software engineering related technologies that can be used for building software using the product platform approach.

Chapter 3 is a literature study comparing analytical application software solutions with corresponding technologies such as data warehouses, data marts, and executive information systems. The chapter includes also the functional requirements for an analytical application software solution and a comparison of different data warehouse architectural models having potential for an analytical application software solution.

Chapter 4 includes discussion of the Analytical Application Frame Strategy framework with different alignment perspectives that a software vendor can utilize when implementing analytical application software solutions. We will also compare different alignment perspectives with each other and discuss their impact on software development and future market segmentation strategies.

Chapter 5 includes discussion of different software development approaches that an analytical application software vendor can take when utilizing the software related product platform approach. We will also explore different variation techniques that are available to an analytical application software vendor. This analysis is implemented in the light of three different sub-strategies defined within an Analytical Application Frame Strategy framework.

Chapter 6 discusses our empirical research design, including our research methodology, our research strategy, and the framework of analysis. We will also discuss data collection methods used in the study and which types of analysis tools were used when analyzing the data. We will also discuss the reliability and validity of the study.

Chapter 7 is a longitudinal interpretative single-case case study analysis of a software vendor. The aim of the study is to obtain ideas and provide rich insight when using an Analytical Application Frame Strategy framework in analytical application software development. This study provides a foundation for understanding the complexities that are involved in development of analytical application software.

Chapter 8 summarizes the contribution of this study, with discussion of the generality of the study. We selected the interpretative case study as our research methodology using the hermeneutic circle from Klein and Myers (1999). We will therefore summarize our case study results in the light of their seven principles. We will give recommendations for future research and discuss the limitations of this study.

2. PRODUCT PLATFORM THEORY

The aim of this chapter is to explore product platform theory specifically from the software product development perspective. We will initiate our discussion by reflecting on the changes that are taking place within software development and the types of implementation approaches introduced for software development during the last few years. This discussion sets the stage for the following aim, which is to explore existing product platform literature specifically from the high technology and software development perspectives. We will also introduce product platform related performance metrics that have been introduced in the existing product platform literature. These metrics will be used when measuring effectiveness and efficiency of product development using the product platform approach. Finally, we will explore closely related software engineering technologies for derivative software development and the potential of these for use in software product line engineering.

2.1 A Reflection of the Software Industry

The aim of this chapter is to shed some light on the changes that are taking place within the software industry. Software vendors must continuously improve their software development processes to be able to compete in their respective software product domains. We will discuss current software research and implementation approaches that software organizations are using to increase productivity and decrease time-to-market.

2.1.1 Changes in the Software Industry

Software development is still often practiced as a craft - by intuition and experience (Qunitas, 1991). Software development is still a relatively young industry, and has therefore not reached the maturity typically found in more traditional branches of industry (Jacobson, 1992; Herzum and Sims, 2000). According to Biberstein (1997), if the software engineering profession is to become truly an engineering discipline, organizations should find “the industrial, or factory, solution for building software.” Herzum and Sims (2000) characterize a mature industry by four different criteria: built-to-order, third-party parts market, maintenance by substitution, and finally supply chain. First of all, build-to-order is a manufacturing process where predefined parts (components) are used to assemble customized products according to customer wishes. Secondly, in a mature industry, there is a third-party market that builds components according to specifications and standards. Thirdly, products should be built of components, and each component can be replaced by new components without disturbing the whole. Finally, the costs of the development should be a small part of the overall supply chain.

Several different approaches to software engineering have been discussed in the literature, such as Hofman and Rockart’s (1994) discussion of application templates as a future method of software implementation to provide an alternative to the traditional buy or build approach. Card (1995) points out in his article "The Rad Fad: Is Timing Really Everything?" that during the 1970's productivity was the fashionable concern, and that during the 1980's quality took the center stage of software development. The theme of the 1990's was time-to-market, rapid development, and related themes. All these different approaches, whether consisting of improvements in object-oriented systems engineering or other software development methods, have a common aim: to get software engineering and software development closer to the manufacturing of physical products (or at least to achieve similar productivity results). Manufacturing organizations have introduced the concept of Mass Customization as a way of modular development (Feitzinger and Lee, 1997) that provides the flexibility to customize a product quickly and inexpensively. Gilmore and Pine II (1997) described the four cases of customization to provide better service to customers.

There are dozens of different ways of crafting software. Many are traditional, with waterfall software engineering models and other iterative prototyping models. Each of these models has to be adjusted to each software company. As Brooks (1987, 1995) has stated in his famous book, *The Mythical Man-Month: Essays on Software Engineering*, there are no silver bullets in software engineering. Many misconceptions have arisen when software companies have assumed that object-oriented software development will be a silver bullet for success. This has turned out to be a misconception of high degree. According to our own experience, object orientation takes time to adopt, and the development organization typically has to be restructured to better meet the objectives of new software engineering processes. Johnson (2000) discusses the ups and downs of object-oriented systems development. This has been frequently discussed in other articles as well (Briand et al. 1999; Pancake, 1995; Fichman and Kemerer, 1993).

According to Hoch et al. (2000), product development cycles are getting shorter and this has changed the competitive landscape for software vendors. According to the study of Hoch et al. (ibid), 80 percent of new software has been on the market for less than two years. Companies competing in Internet time must constantly change their ideas, experiment, and plan complex new products and technologies to be competitive. Their competitive advantage could appear or disappear overnight, which makes the competition extremely harsh. Cusumano and Yoffie (1998) found that some of the pre-Internet world strategic precepts, such as vision, leadership, innovation, quality, barriers to entry, customer lock-in, switching costs, and partner relationships still hold true.

According to Biberstein (1997), the software engineering world has for some time "conceived that applications systems could be built from prefabricated parts such as integrated circuits (ICs), mounted together in accordance with plans, an architecture, and design drawings produced by the system analyst." This type of software development approach is also closely related to component-based software engineering that "has emerged as a key element in the development of complex software systems" (Hopkins, 2000). A similar idea is presented by O'Grady's (1999) discussion of developing software products using a modular approach.

Software IT infrastructure could be very different in different end user organizations, and this could in some cases require cross-platform development. This type of cross-platform development is not without challenges, as have been reported by organizations such as Netscape (Cusumano and Yoffie, 1999). Brereton et al. (1999) argue that software evolution will become a problem in the future due to evolution and advances in technology. The problem could become worse when different components from different sources need to be integrated. This could lead to obsolescence and a lack of confidence in the software industry (ibid).

2.1.2 New Ways of Implementing Software

Due to the profound changes in the software development environment, the software industry and academics are all searching for new ways of designing and implementing software. According to Brereton et al. (1999), software will be "increasingly component-based, that is, components will be customizable and flexible, rather than rigid." The authors (ibid) conclude that software component technology must achieve or create a rigid "glue" technology that enables software components to be assembled and disassembled effectively. Based on our literature search on current focus areas within software engineering, it was very evident to us that component-based software engineering (CBSE) is one of the most studied topics in current software engineering research (Jacobsen et al, 1997; Szyperski, 1997, 2003; Sametinger, 1997; Herzum and Sims, 2000). Component-based software engineering has been compared with traditional software development methods, and experiences with CBSE have already been reported (Sparling, 2000).

Another research domain close to component-based software engineering is product line software development (McGregor et al., 2002; Northrop, 2002), which aims to achieve larger scale reuse among software development organizations. One current discussion is whether software organizations have aimed to created software components that are too small, while the latest literature suggests that component-based development should be more coarse-grained to be able to achieve more effective reuse of software assets among different products in a product line. Iansiti and MacCormack (1997) discuss a software development process that is specifically geared to software development in Internet time. This software development process is a flexible approach that "allows companies to respond to changes in markets and technologies during the development cycle." The increased pressure to increase the time-to-market has forced software vendors to change their software development practices. Carmel (1995) discusses cycle-time reduction as a fundamental competitive product strategy which "reduces the time from conception to delivery of the product to market."

Booch (1996) predicts that future software development will be more complex due to distributed computing and greater user expectations for better visualization and access to information. Complexity will not only be seen in traditional software packages, but also in embedded software systems such as household devices, etc. Consumers drive software development efforts to be more complex because consumers learn from one product release to the next how the product could be improved (Jacobsen et al., 1999). Welke (1994) suggested in his article "The Shifting Software Development Paradigm," that software development will shift towards production of "commercially available object components for general and niche applications." His view of software development was very accurate, and with the current knowledge and advancement in software engineering, his prediction is closer to reality.

A group of researchers decided in 1996 to research current practices of software vendors around the world (Hoch et al., 2000). This decision was inspired by the research conducted by Cusumano and Selby (1995) into Microsoft and its software practices as key factors for its continuing success. The research groups wanted to know whether the lessons learned from Microsoft were applicable to the overall software industry. Therefore, this group made a survey of a large sample of software companies around the world to find the secrets to their software success and failure stories. This study is the first of its kind to be executed on a large scale and on a global basis. Over 450 software executives from over 100 software companies were interviewed, as were over 50 industry experts. The survey was based on three different segments: professional services, enterprise solutions, and mass-market products.

Iansiti (1998) performed four empirical studies in which each comprised field investigations of competitors in a focused industry segment. These industry segments were semiconductors, mainframe and supercomputer subsystems, workstations and servers, and software. The aim of the study was to investigate technology integration and its effects on product development. Good technology selections were found to be a critical factor in successful Research and Development (R&D), and a good match between technology and product architecture was critical to the products' competitiveness (Iansiti, 1998). Iansiti (ibid) researched Microsoft, Netscape, Yahoo, and NetDynamics and found that technology integration capability and its importance in the software industry is similar to that of the semiconductor and other hardware industries.

Yoffie and Cusumano (1999) introduced a concept called "Judo Strategy," which describes how companies should compete in Internet time. Their findings are based on research they did on two companies, namely Microsoft and Netscape Communications, with three recommendations for software organizations when setting their strategy. The first recommendation suggests that software vendors should move rapidly to uncontested ground to avoid head-to-head competition. The second recommendation is to be flexible when a superior force attacks directly. The third and final recommendation is to exploit leverage that uses the weight and strategy of opponents against them (Yoffie and Cusumano, 1999).

Changes in the software development community due to technological changes (such as the Internet and wireless technology) and "time-to-market" pressures require software vendors to review their existing software development processes to reflect this new change. Software vendors are now facing a new situation in which project specifications are changing during the course of the project. This makes the old traditional software processes, with "concept and specification freeze" phases, obsolete. Organizations such as Microsoft have introduced a new software development approach in which customer feedback is taken into consideration before the product is shipped (see Fig. 1).

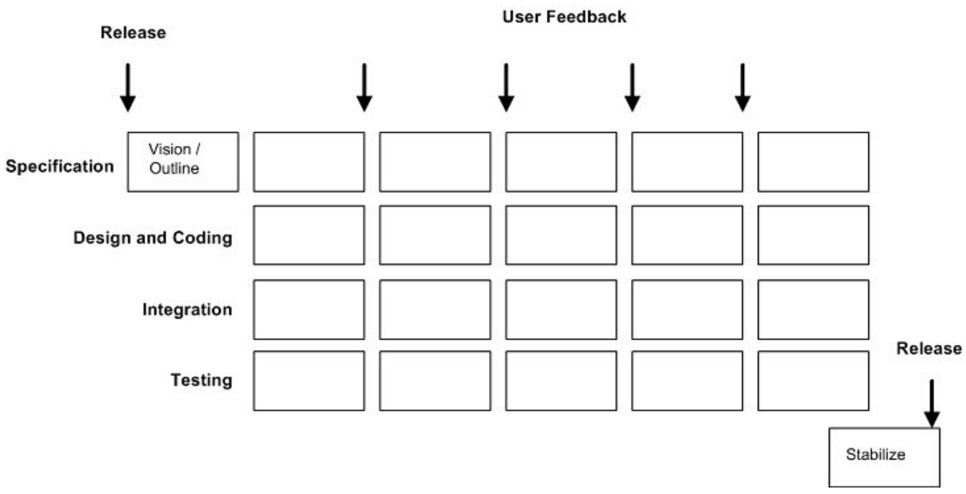


Fig. 1. A Flexible Approach to Software Development (Iansiti, 1998).

Iansiti (ibid) also suggests changes to the traditional concept of concept development and implementation of the product. The author defines total lead-time as a total of concept lead-time and development lead-time. Lead-time is the time during which new requirements can be introduced into the project, whereas development lead time is the time when the product's architecture is frozen and can therefore not be changed. The author (ibid) suggests that concept and development lead-time should to some extent be executed simultaneously. He emphasizes that this model is not the same as concurrent engineering, because concurrent engineering does not normally imply simultaneous execution of conceptualization and implementation.

Similarly, Cusumano and Selby (1995) explain a software development process called "Sync-and-stabilize," used by Microsoft and, during the last few years, also by other software organizations around the world (Cusumano, 2004). This software development process is based on the idea that programmers are encouraged to "innovate and experiment but frequently synchronize their designs with other team members by creating software builds of the product as often as possible" (Cusumano, ibid). This type of development approach enables software organizations to incorporate new ideas and concepts along the way. This is different from the more traditional waterfall approach, where all of the functionality must be decided upfront.

In summary, changes in technology and the requirement to introduce new products to markets quickly require software vendors to innovate and test new ways of implementing software. Another question that each software vendor faces is the question of whether to become a software products company, a services company, or a combination of both. According to Cusumano (ibid), many companies change their business model from a software products company to a service company due to decreased software product sales. According to Cusumano (ibid), hybrid software companies can achieve the best of two worlds. When these companies apply the software product approach, they can achieve rapid growth and large profit margins. When software sales are down, these companies can substitute software product revenue with service revenue.

2.2 The Product Platform

The aim of this chapter is to explore the product platform related literature in mechanical engineering and high-technology companies. The product platform concept was initially introduced in mechanical engineering, but it has lately also been discussed within the software engineering domain. This will be the aim of our next chapter.

2.2.1 The Product Platform in Mechanical Engineering

Several research papers on the product platform theory within mechanical engineering were published during the 1990s. A common approach in these papers is to make product development more effective by building a large common product platform that will be reused across a set of products in a product family. The product platform is seen as a long term investment wherein the cost of the product platform is considerably higher than the cost of an individual product. Sanderson and Uzumeri (1995) discuss the impact of product variety on a company, and Henderson and Clark (1990) discuss the potential financial impact of choosing an appropriate architecture for a set of products. Papers have also been published that address the issue of planning commonality within a set of products (Martin and Ishii, 1997; Wheelwright and Clark, 1992; Meyer and Lehnerd, 1997; Robertson and Ulrich, 1998).

Wheelwright and Clark (1992) concentrate on describing how an underlying product architecture can be used in successive derivative product development. According to the authors, the initial platform of a product family consists of "the subsystems and subsystem interfaces of the basic product design." These subsystems "can be incrementally reengineered or redefined to generate specific product offerings." A platform extension occurs when a subsystem within an existing product platform changes substantially or new subsystems are added to the design. The requirements for these platform extensions are that they should not interfere with existing primary subsystems. The authors also conclude that product platform renewal occurs when "product design is rearchitected to incorporate major new subsystems and new subsystem interfaces."

Meyer and Utterback (1993) added core capability as a key driver for successful product platform development. They claim that core capabilities cannot be separated from the products that the company produces. The authors also emphasize that a robust product platform is the "heart of a successful product family" where "generic core capabilities in any product family exist in product technology, market understanding, and so on." They define a product family as products that share a common platform, but have specific features and functionality. This approach enables companies to create products for different market segments by using a common product platform. The authors define general-purpose applications that serve different customer groups and industries as horizontal market applications. The authors claim that successful product family development requires a solid understanding of customer requirements and of their technical infrastructure. Management must identify the importance of product platform development and its impact on long-term product development productivity, as product platform development has a longer lifecycle than individual products. The platform enables companies to create product variations more effectively than creating each product from scratch.

Typically, software vendors create new products based on a market-pull situation. This situation is characterized by a market opportunity with specific customer needs. Ulrich and Eppinger (1995) classified products into four different categories: technology push products, platform products, process-intensive products, and customized products. Typically, a technology push product is based on proprietary technology invented by the software vendor. The vendor identifies a market to which to apply this technology. This approach has obvious risk if the market is not accepting this technology. A platform product is a product which is built on top of pre-existing technological sub-system. According to Ulrich and Eppinger (ibid), products such as Sony's Walkman and Apple's Macintosh are typical platform products – both of these have been the basis for derivative product development. Process-intensive products are products where the product design and the process cannot be separated from each other. Typical examples of these products are foods and chemicals. Finally, customized products are products that are variations from a standard configuration, such as switches, motors and batteries.

Ulrich and Eppinger (1995) argue that products that are built using a technology platform are simpler to develop than if they were developed from scratch. Product architecture is part of the technology platform. Ulrich and Eppinger (ibid) define products in two different ways: functionally and physically. Functional elements describe the operations and transformations that contribute to the overall performance of the product, while physical elements describe the parts, components, and subassemblies of the product. They define the product architecture as "the scheme by which the functional elements of the product are arranged into physical chunks and by which the chunks interact." Modular product architecture must have well defined interactions between the modules, and every module needs to implement one of a few functional elements of the architecture. Based on these original product platform publications, some authors have also calculated both product platform efficiency and effectiveness, measuring how well a product platform has performed in a given environment (Meyer et al., 1997). According to Jacobsen et al. (1997), software reuse has been recognized since McIlroy (1969) introduced libraries of shared components. The aim of software reuse is different from that of product platform development. Software reuse focuses on cost reduction and overall cost savings when using common software assets, while the product platform strategy focuses on maximizing software revenue by using an effective product platform. Effectiveness and efficiency are measured using software revenue as one of the key drivers.

Product platform theory has been widely researched within mechanical engineering, and several studies have shown a positive impact of using a product platform in developing products such as automobiles or other physical artifacts (Gonzales-Zugasti and Otto, 2000; Dahmus et al., 2000; Meyer and Lehnerd, 1997). Dahmus et al. (2000) discuss both Volkswagen's and Ford Motor Company's product platform development for different car brands with the use of common product components by sharing these components in different brands. Product platform development within mechanical engineering is very different from software engineering and it is therefore important to identify these differences and research whether software product platforms can be used in developing software products.

Based on these findings from physical product manufacturing using a product platform with a corresponding product architecture, our next aim is to research existing knowledge in product platform theory within high technology companies and specifically within software products. Software product line product development addresses the questions and issues arising from the contention that software organizations no longer can afford to develop multiple software products one product at a time (Bosch, 2000). There could be several reasons for this, such as the pressure to improve time to market or pressure to maintain market share. We suggest that software development organizations have to constantly look for more effective ways of implementing products in a manner similar to that used in the world of mechanical engineering.

2.2.2 Product Platforms for High Technology Companies

McGrath (1995, 2001) argues that product strategy work is more difficult in high-technology companies compared with other more stable industries. McGrath published his initial work on product strategies in 1995, specifically addressing product development strategies for high-technology companies. His second edition of the same publication was released in 2001. The aim of this release was to give an updated version of the work that had taken place since the original publication was released. McGrath's initial work explains in detail the elements that are required to build a product strategy for high technology companies, and his second edition of the same work extends the original work with a new framework that he names his "Core Strategic Vision Framework (CSV)." This framework is composed of several sub-strategies (core competence, competitive strategy, technology strategy, product strategy, business charter, financial plan) that have to be aligned with the Core Strategic Vision to achieve an optimal product development strategy. Another new framework found in his new work is the Market Platform Plan (MPP), which integrates knowledge about the market and knowledge about the product and its defining technology.

According to McGrath (2001), a product platform is primarily a planning construct which "sets the architectural rules and technology elements that enable multiple product offerings." Architectural rules portray how technology elements are integrated and how these elements interface with each other. McGrath also lists other technologies as important parts of the platform. These other technologies enable the vendor to develop a complete product offering. The author also defines a product platform not as a product, but "a collection of the common elements, particularly the underlying defining technology, implemented across a range of products." McGrath (ibid) defines a product platform to be "the lowest common denominator of relevant technology in a set of products or a product line." In a software application product, McGrath defines the product platform as consisting of the architecture (such as mainframe, client/server, desktop, or Web-based), input/output, and application functionality.

McGrath also emphasizes that a product platform should not be as seen static, but as a construction that will evolve along with time and customer requirements. A product platform should not include functionality requirements for a specific customer, but it should portray customer needs from all relevant markets that the vendor is going to address. According to McGrath, a product platform should evolve according to desired vector of differentiation. This vector is based on the defining technology of the platform. McGrath defines the defining technology as that platform element that clearly distinguishes it from other platform elements. An example of defining technology in an Apple Macintosh computer is its easy-to-use graphical user interface. This defining technology will typically differentiate it from other products in the same domain and will therefore provide its competitiveness. McGrath divides platform technology elements into three categories: defining, supporting, and segmenting. Defining elements gives the vector of differentiation for the products, while supporting elements support or enhance the defining technology. The segmenting technology elements address the specific customer value proposition for the given segments.

According to McGrath, the key benefits of implementing a product platform as part of the overall strategy in the organization are numerous. First of all, product platforms focus management on key decisions at the right time: product will be deployed more rapidly and consistently, the platform approach encourages a longer-term view on product strategy, a platform strategy can leverage operational efficiencies, and product platform principles help management anticipate replacement of a major platform. The main emphasis in product platform development is therefore an emphasis on a very strategic level, while product managers and developers can manage development of individual products without having to involve senior management.

McGrath was able to bring additional dimensions into strategic product development, such as marketing strategies, product differentiation strategies, and several other strategies that will affect the company when implementing products using a product platform strategy. McGrath did not specifically address the needs of a software vendor, and his view of a product platform in the software domain is very naïve and leaves a lot for software developers to desire. Issues such as software architecture, software development, and IT infrastructure elements are completely ignored in McGrath's work, which does not discuss how a software product platform should be defined, designed, or even implemented.

2.2.3 Other Known Product Platform Publications and Sources

Meyer and Lehnerd (1997) published a book *The Power of Product Platforms – Building Value and Cost Leadership* that describes the product platform and its theories with a new concept they call "Power Tower." According to this construct, organizations are able to plan their products with respective product families using vertical leverage within a price/performance tier and/or according to a market segmentation using horizontal platform leverage. This was the initial publication that introduced market segmentation strategy into product platform theory. The authors emphasize the importance of internal software architecture and effective software interface management in product platform development. According to the authors, manufacturing of the products should be a part of the platform, and they go so far as to conclude that even the evolution of manufacturing processes could become an opportunity for innovation. The authors do not specifically discuss software manufacturing, but refer to more generic terms of manufacturing.

The authors portray the evolution of a product family, product platform renewal, and new product creation in the following way (see Fig. 2).

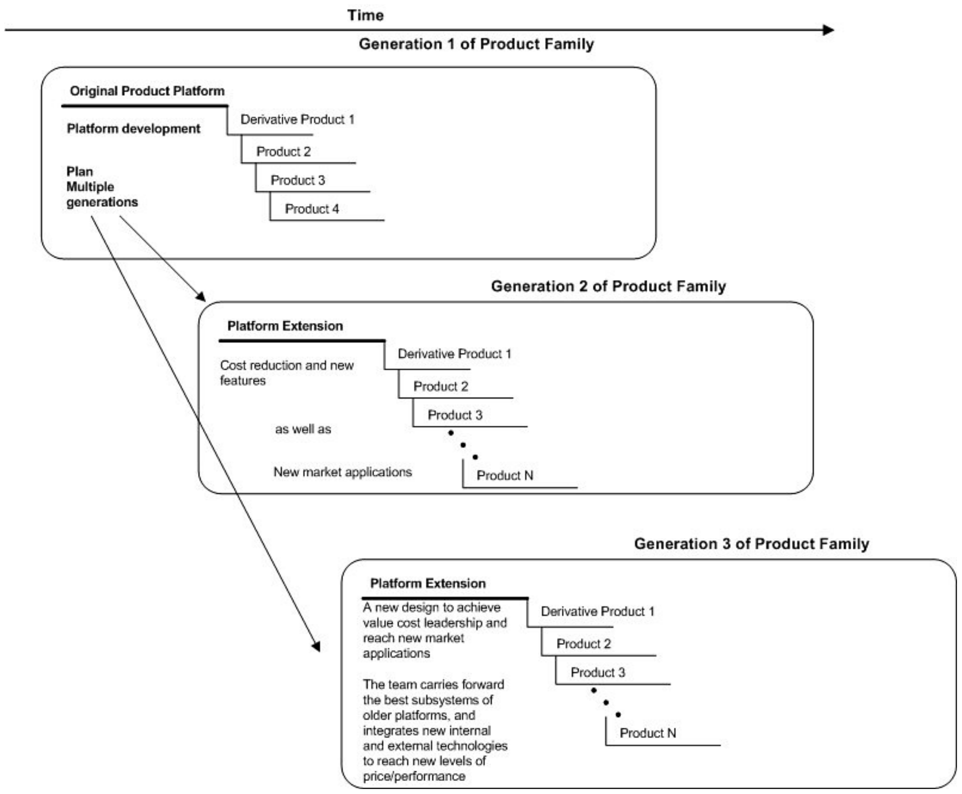

Time
Generation 1 of Product Family
Original Product Platform
Platform development
Plan
Multiple
generations
Derivative Product 1
Product 2
Product 3
Product 4
Generation 2 of Product Family
Platform Extension
Cost reduction and new features
as well as
New market applications
Derivative Product 1
Product 2
Product 3
Product N
Generation 3 of Product Family
Platform Extension
A new design to achieve value cost leadership and reach new market applications
The team carries forward the best subsystems of older platforms, and integrates new internal and external technologies to reach new levels of price/performance
Derivative Product 1
Product 2
Product 3
Product N

Fig. 2. Product Platform Evolution (Meyer and Lehnerd, 1997).

Figure 2 portrays three product generations: the initial product platform with its derivative products and two new product family generations with corresponding derivative products. The authors' view of modularity is closely related to physical product development without referencing component-based software engineering. Meyer and Lehnerd (ibid) conclude that one of the common building blocks for the product platform is a manufacturing process and that "best-in-class processes have to be integrated with platform design" that could obviously be such as a selected software engineering methodology or approach. The authors do not discuss the granularity of the software modules or subsystems other than via composite design, used to identify the commonalities across different vertical market segments. The authors (ibid) conclude that some of the key building blocks for the platform might come from other companies via subcontracting, but the discussion does not explain the risks of subcontracting core competitive and defining technologies outside the internal development team. The authors also discuss a great deal of both perceived and latent customer needs in product development, and the importance of this in achieving a competitive edge in the selected product domain.

Another recent product platform research project was completed by Gawer (2000) and Gawer and Cusumano (2002). This work concentrated on analyzing how some organizations such as Intel, Cisco, Microsoft, and many others have been able to achieve product platform leadership in their domain. According to Gawer and Cusumano (2000), modular design of products has a powerful impact on innovation, whereby "innovation can happen on modules of the product without having to impact (and threaten the integrity of) the overall system." According to the authors, product platform leadership refers to a common objective sought by the companies to drive innovation in their industry. The authors implemented a framework – Four Levers of Platform Leadership – that can be used by organizations that have product platform leadership or want to achieve product platform leadership to "design and test the validity of their strategy, given the circumstances of their industry and the competences of their corporation." These four levers are as follows:

- Scope of the firm – defines what the company should do internally and what kind of work should be encouraged by external vendors.
- Product technology (architecture, interfaces, intellectual property) – defines the product modularity, interfaces, and how open these interfaces will be to complementors.
- Relationships with external complementors – defines how collaborative versus competitive the platform leader should be with the complementors.
- Internal organization – defines how the product platform leader manages external and internal conflicts and conflicts of interest when working together with complementors.

The authors conclude that to be able to sustain product platform leadership, all these four levers have to be managed effectively and be kept part of the corporate strategy. According to the authors, organizations such as Microsoft have controlled their product platforms by controlling application programming interfaces. These interfaces have not been submitted to any standards body, but have been controlled by Microsoft. Microsoft is a good example of a software organization building a product platform such as an operating system environment and then building software complements by competing with other software organizations that use the operating system platform as the basis for development.

2.3 The Product Platform in Software Development

The aim of this chapter is to introduce software related product platform literature published during the 1990s and early 2000s. First of all, we will introduce Meyer and Lopez's (1995) discussion of technology development within a technology company with an operating system software product. Secondly, Meyer and Zack (1996) introduced the product platform theory into information products, with examples of two electronic publishing companies. Thirdly, Meyer and Seliger (1998) introduced the theory of platform strategy for software products and, finally, Sääksjärvi (1998, 2002) redefined these into the software application and frame strategy framework.

We will compare these literature sources based on criteria set in Appendix 1 to help us to find the criteria that are important for software development organizations in evaluating the use of the product platform concept. The evaluation criteria are set into different categories, such as the definition of a product platform (both traditional and software related), manufacturing or design of the product platform, IT infrastructure and architecture related questions, product architecture related questions, technology related questions, market segmentation/leverage related questions, economical measures such as platform efficiency and effectiveness, and finally core competence/capability characteristics for the software vendor (see also Appendix 1).

2.3.1 The Software Platform as Part of Core Technology Capability

Meyer and Lopez (1995) released one of the first references to product platform related software development. The aim of their study was to map the evolution of a software company's product family development and to identify core capabilities in product families to be able to understand the relationships between these two. The study introduced a software company that provides UNIX operating system adaptation for PC platforms and real-time capability to the UNIX platform. This study consisted of a detailed analysis of the company's technology strategy and product platform and its product family development strategy. One of the key elements in the analysis was to find and analyze the importance of core capability in product family development and how a software vendor's technology strategy affects software product development. Meyer and Lopez (ibid) emphasized that investments in a company's core capabilities and underlying technologies have to be continuous to ensure successful platform renewal or platform extension.

According to the authors, a product platform design is "comprised of subsystems or modules and the interfaces between these modules." These subsystems or modules are typically based on software components with predefined and standardized interfaces and with component part numbers. Software interfaces can be categorized into internal and external interfaces. An example of internal interfaces within the UNIX environment is "those protocols, rules, and mechanisms by which data are exchanged between UNIX kernel, systems administration, and utilities modules." Meyer and Lopez identify a layered architectural model for the system architecture of a UNIX operating system environment. This layered architecture consists of three main layers, namely the hardware, operating system, and application layers.

Meyer and Lopez define a product family as "a set of products that share core technology and address a related set of market applications." According to the authors, the technological foundation of the product family is the product platform, and a platform "is the physical implementation of a technical design that serves as the base architecture for a series of derivative products." The authors consider product platform renewal the basis for continued successful derivative product family development. A part of this renewal is the need for core competence (Prahalad and Hamel, 1990) and the capability of selected underlying technologies. The authors identified two main core competencies in the company, namely the ability to make UNIX work in PC environments and the ability to add real-time capability to UNIX. Integration of external technologies to the core product platform influenced the evolution of the case study company's product family. The management of the company had to nurture relationships with the UNIX providers and also make sure they knew what these vendors were up to in their development of subsequent releases, as the case study company had based its whole business on adding value to the basic core UNIX.

The paper from Meyer and Lopez was the first product platform paper to focus on technology driven software development in a software development organization. Meyer and Lopez identified the importance of a balanced technology strategy to enable the organization to achieve greater market leverage for its UNIX operating system adaptations to PC platforms. The authors did not explain process platform development from a software development organization perspective, but rather emphasized the importance of a balanced technology strategy in product platform development. The authors viewed software development from a very technical perspective without explaining in detail how interface management is linked to software development practices or development methodologies.

Meyer and Lopez also discussed the impact of the integration of external technology into product platform development. One application of this approach is composite design, which enables the software vendor to identify the elements or subsystems in the product platform that will be either developed internally or purchased or sub-contracted from third-party vendors. The authors failed to discuss the dependencies of IT infrastructure on the software development environment (the development time environment) and the end user organization IT infrastructure (the runtime environment). Within software development, software organizations have to balance between the software development time and runtime environments. Emphasis in one could impact the other either adversely or favorably. The authors emphasized two areas in product platform development: solid technology strategy that has to be in balance with the product platform and architecture. The case study company that the authors analyzed had changed and added to its development strategy a new application area, progressing from pure UNIX platform adaptation to PC hardware platforms to embedded real-time capability to UNIX platforms. This change can be regarded as a challenge for any software vendor, as the underlying IT infrastructure technology changed from one hardware environment to another.

2.3.2 Product Platform Theory in Information Products

Meyer and Zack (1996) published an article within product platform theory about the development of product platforms for information products. They authors conclude that within the domain of information products, the concept of sub-systems should be replaced with information units, a repository instead of a product platform, and a refinery process instead of production or manufacturing process. The discussion of the process platform and the development of information products was new to existing product platform theory, and the authors even argue that the process platform is a tool to enable greater product variation. Several other authors (Meyer and Lopez, 1995; Meyer and Lehnerd, 1997; Meyer and Seliger, 1998) refer to product variation as a means to leverage a product platform. The authors (ibid) do not specifically discuss how product variation can be implemented in practical terms. Their approach is very abstract, leaving the reader to identify the means of using a product platform in its product development. According to the authors, the process platform "is composed of the technologies, facilities, and processes for manufacturing a firm's products." Technological leverage is "the extent to which investments in basic product and process platforms serve as a foundation for efficiently developing derivative products." These statements include a strong emphasis on the process platform as the engine for derivative products within the information products domain.

Meyer and Zack (1996) define the repository of a database services company as the product platform and the information content as the substance for the product platform. The refinery is the process platform, where the database company manufactures information based on five information processing stages; acquisition, refinement, storage/retrieval, distribution, and presentation or use. These stages resemble a typical decision support application and the processes that must be undertaken when the decision support solution is delivered to the end user organization. According to the authors (ibid), the combination of the two platform elements (product platform and process platform), the repository, and the refinery is the foundation for derivative product development. The technical leverage is measured when the database company is able to efficiently and effectively manufacture new information products for different market segments using the same underlying repository and process platform. These different derivative products can be different from many different perspectives – for example, from that of packaging, formatting, and even distribution media.

The analysis from Meyer and Zack (ibid) is more or less driven from the information content provider perspective, including only a few references to software product development and how software development could affect the development of information products. A few similarities exist in software development between regular database driven products and information products. The flexibility of information products (such as traditional database client/server applications) is also dependent on software related designs such as the database structure, the flexibility of user interfaces to accommodate new vertical market sectors, and the flexibility of the distribution architecture, as information content has to be delivered to different devices or software applications such as email, Lotus Notes etc. From the software development perspective, architectural decisions for information products must support different configurations of the driving software application and, therefore, the database structure or persistent data store has to be abstracted from user interfaces and information collection mechanisms to enable maximal variety for different vertical markets. Without this type of abstraction, each derivative product might require changes in its user interface to reflect the new contents of the database. This type of software development could become a nightmare for a software vendor, as several releases might require simultaneous maintenance and support.

2.3.3 Platform Strategy for Software Products

Meyer and Seliger (1998) introduced a new product platform architecture geared toward software development in their article "Product Platforms in Software Development." The authors define a product platform as "a set of subsystems and interfaces that form a common structure from which a stream of derivative products can be efficiently developed and produced." Meyer and Seliger (ibid) base their research on two hypotheses of product platform development advantages. Their first hypothesis is that a well-designed software platform architecture can provide a substantial R&D advantage for software development organizations. They conclude that efficiency that can be achieved by using a product platform approach measured both by accumulated costs in the platform itself and how quickly a software vendor can generate new products from a common product platform. The second hypothesis is that platform architecture provides a software company a firm business model that enables it to build market share and revenue.

Meyer and Seliger (ibid) propose an architectural model that they call their "Platform Strategy for Software Products." This architectural model is based on three separate architectural layers. It is the first attempt from the product platform research community to help software development organizations to build product platforms for software products (see Fig. 3).

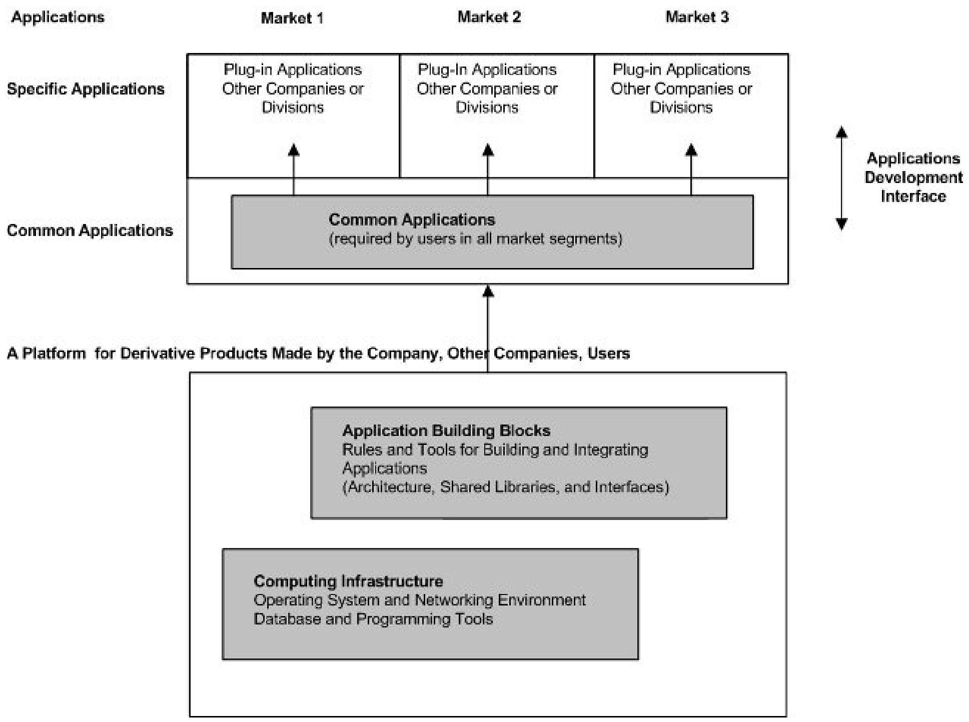

Applications
Market 1
Market 2
Market 3
Specific Applications
Plug-in Applications Other Companies or Divisions
Plug-In Applications Other Companies or Divisions
Plug-in Applications Other Companies or Divisions
Applications Development Interface
Common Applications
Common Applications (required by users in all market segments)
A Platform for Derivative Products Made by the Company, Other Companies, Users
Application Building Blocks
Rules and Tools for Building and Integrating Applications
(Architecture, Shared Libraries, and Interfaces)
Computing Infrastructure
Operating System and Networking Environment
Database and Programming Tools

Fig. 3. Platform Strategy for Software Products (Meyer and Seliger, 1998).

Meyer and Seliger (ibid) depict the first architectural layer as the supporting or enabling technologies layer. These are the elements for core technologies that are the basis for a particular generation of a product family created from a common product platform. The authors include the operating system environment and the hardware and networking environment as part of the computing infrastructure environment for a software developer. The authors conclude that the platform "is composed of the developer's design strategies and specific implementation procedures and protocols for the products within the entire family." The rules and tools of a software development organization could become standards and protocols for the software organization and they could be regarded as part of the product platform for the software organization. Their article includes a vague reference to the software development processes that a software organization must manage when building software solutions for a selected market domain.

The second architectural layer consists of applications or solutions "that are composed of specific modules of software that plug into and work seamlessly with the underlying software." This description explains how each product platform requires a software layer that includes all common elements for each and every planned vertical market segment. The third and final layer is needed to provide software extensions to each specific vertical market that requires either specific functionality or pricing that cannot be achieved by providing all of the functionality that a high-end or low-end end user organization might expect (price/ performance segmentation).

Meyer and Seliger (ibid) emphasize the importance of application programming interfaces that provide software organizations with strong product platform leverage by providing a third-party development community with a published application programming interface. The authors give some examples of successful software development using a product platform theory by describing the success of Visio Corporation within the graphics software industry. Visio Corporation (now part of Microsoft Corporation) was able to create a third-party development community by providing a generic charting engine (the product platform) with an application programming interface (API). The result of this development was that the software engine became the industry standard within the technical graphics industry. Another example that the authors portray is Lotus Corporation and their attempt to apply component-based development to the office application suite domain. Lotus decided to create an application suite that shares some common subsystems between different applications in a way similar to that used by Microsoft with Microsoft Office.

Meyer and Seliger (ibid) propose that component-based software engineering could have a fundamental impact on software development. This approach enables concurrent engineering and incremental development of software applications as long as these components are well defined, are loosely coupled, and have well-defined component interfaces. By using a modular approach, software vendors can externalize some of software development to accelerate time to market for their products. According to the authors, the platform "is a collection of subsystems, themselves composed of modules or components, any number of which may solve a particular application problem or requirement." The authors also include the characteristics of a "complete" platform: that it should "include a linkage to all the building blocks that the application developer requires to satisfy the user at a reasonable cost and time."

Finally, the Meyer and Seliger (ibid) discuss the timing and renewal of the product platform and the importance of having the commitment of the executive management of the software vendor. According to the authors, platform development requires a holistic approach to the development of software applications, and this will require resources that management has to commit to core software platform development. Each company also needs to view platform development as a vehicle to get into new market segments via effective and quick release of derivative products. The aim is to be able to reuse the applications across different market segments and therefore avoid redevelopment of functions and features that can be reused in each market segment. According to the authors, development of a software product platform is "both architecture and an implementation of architecture that comprises core subsystems that propel a family of software products or internal corporate applications."

Even though the authors conclude that software systems typically include many more subsystems when compared with traditional physical products, they argue that the traditional physical product platform development theory can be applied to the development of a product platform for software products. The authors assume that component-based software engineering might be the best approach for building software product platforms. To support this argument, they built the layered platform architectural model described earlier. The authors fail to discuss the dependencies between the layers or how software development is still coupled with the IT infrastructure that is supported by the end user organization. Nor do they discuss the needed software development environment that could be different from that of the environment in which the software is run.

Meyer and Seliger (ibid) do not specifically address the complexities of a given software domain. The authors' architectural model is generic, but we argue that these types of generalizations could be too strong, as each software domain might include its own specific peculiarity on each and every level of the proposed architectural model. Meyer and Seliger (ibid) use a generalized definition of the software product platform and argue that this generalized definition serves well also for software. We argue that this definition is vague, as it does not imply the dependencies of software development, and the authors fail to discuss in detail the component granularity levels that a software product platform should consist of and how a platform should be composed. If the underlying IT infrastructure is selected carelessly, it could restrict the market segments and run-time hardware environments both for the end user and the software vendor organization.

Meyer and Seliger (ibid) exclude completely all the complexities of the software development process: they include the developer's design strategies and specific implementation procedures and protocols in the core platform without discussing how they relate to overall product platform development. They have not specifically separated the process platform from the product platform as Meyer and Zack (1996) did in their article "Platform Development of Information Products." A question arises as to whether the process platform introduced in the Meyer and Zack article is unnecessary or if Meyer and Seliger presume it to be irrelevant in the development of software products.

The authors also discuss the importance of identifying the commonalities between different market segments to provide leverage to different market segments. Meyer and Seliger conclude that the product platform is the base software engine and that the derivative products are the add-in modules that "can be seamlessly plugged into the base engine." We argue that this analogy is oversimplified, as the authors have failed to include other key components into the software platform development, such as software development environment with included elements. The authors also introduce a new component-based software product platform architecture for the first time in the product platform literature. This architecture is divided into three main layers.

2.3.4 The Software Application Frame and Frame Strategy

Sääksjärvi (1998) studied existing product platform theory both in the physical and the software related domains in a research project. The results of this research were published in the fall of 1998. To augment this literature research, the study also included four software companies as case studies to test whether these companies had recognized or deliberately used a product platform as a basis for their product development strategy. Based on the results from PLAT98 research (Sääksjärvi, 1998; Sääksjärvi and Salonen, 1998), the product platform theory from mechanical engineering must be amended to be used as a basis for software product development. Sääksjärvi (ibid) introduced a new concept of a frame and frame strategy with three different sub-strategies, namely the frame architecture, frame leverage, and frame technology sub-strategies (see Fig. 4).

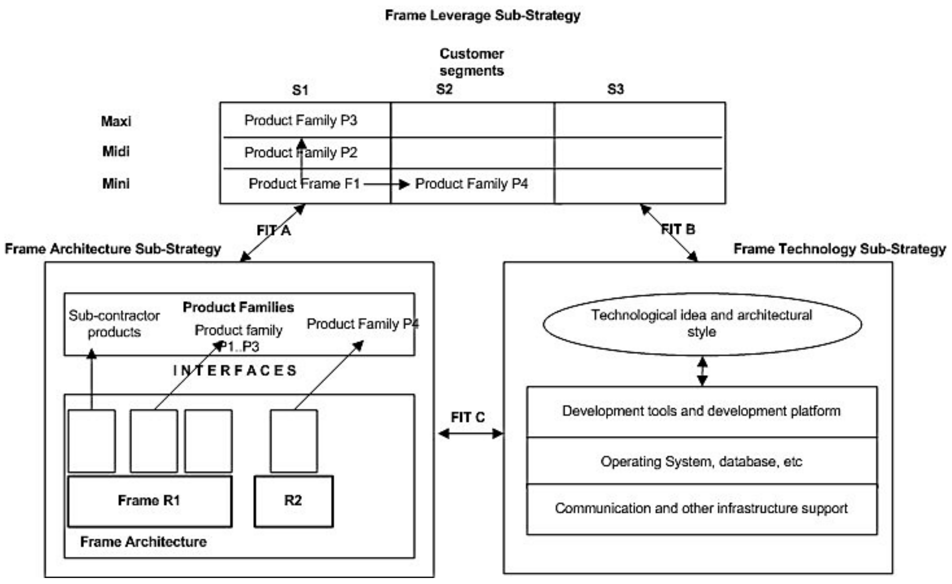

Frame Leverage Sub-Strategy
Customer segments
S1
S2
S3
Maxi
Midi
Mini
Product Family P3
Product Family P2
Product Frame F1
Product Family P4
FIT A
FIT B
Frame Architecture Sub-Strategy
Frame Technology Sub-Strategy
Sub-contractor products
Product Families
Product family P1..P3
Product Family P4
INTERFACES
Frame R1
R2
Frame Architecture
FIT C
Technological idea and architectural style
Development tools and development platform
Operating System, database, etc
Communication and other infrastructure support

Fig. 4. Frame Elements and their Strategic Fit (Sääksjärvi, 1998).

Sääksjärvi (ibid) concludes in his research that the use of product platform theory in the development of software products requires a strategic alignment between the frame architecture, the frame leverage, and the frame technology sub-strategies. If and when a software vendor analyzes strategic fit between these three sub-strategies, the analysis should help the software vendor to solve the following questions (Sääksjärvi, 1998):

- What is the product idea of the company, its technological solution and core competence (frame strategy)?

- What are the vendor's customer segments and product families, and which new product family is the application frame going to help to build (frame leverage)?
- What are the needed application frame components for all of the products in the product family (frame architecture)?
- What is the common technology for all of the products in the product family and what are the supported infrastructure selections (frame technology)?

The author argues that these three sub-strategies "should be tightly interconnected and therefore, they should be aligned into an integrated software application frame strategy." This type of discussion is not discussed or introduced in earlier product platform related literature. According to Sääksjärvi's framework, application frame strategy consists of a frame architecture, frame extensions to enable leverage, interfaces, and development partners. Thus, the core of every product needs to be based on a solid and flexible architecture, supporting several different market segments and changes within these. Frame extensions can be based either on vertical or horizontal extensions. Core product architecture is also directly linked to the functional requirements of the software package.

The framework in Figure 4 portrays three different sub-strategies and their interrelationships. Frame architecture sub-strategy includes the application frame(s) and their corresponding interfaces and these will be the basis for derivative software development using the product family/product line approach. According to Figure 4, the more flexible the frame architecture is, the more flexibility the software vendor achieves when planning its future software releases. The second sub-strategy within the framework is frame technology sub-strategy which is divided into three sub-components:

1. Architectural style (and technology recipe)
2. Software development and implementation technologies and tools (process technologies)
3. IT infrastructure technologies (both internal and external).

The first sub-component sets the boundaries for the software from an architectural perspective and describes the main idea of the solution. The second sub-component within the frame technology sub-strategy describes the implementation technologies and the development tool strategy. The final sub-component within the frame technology sub-strategy is the set of decisions that need to be made on both internal and external IT infrastructure technologies. This was something that had not been discussed in prior product platform literature. Sääksjärvi concludes that the main difference between component-based software development and application frame development is that the latter is used to implement very similar products with the same underlying technology, while component-based software development could in some cases address the needs for several different types of products and even underlying technologies.

The frame leverage sub-strategy sets the directions and strategies for extending the application frame into new market segments. This sub-strategy will be crucial for software vendors, as they can leverage software assets within the software application frame to derive new products for new market segments. To enable effective frame leverage to different market segments, the frame architecture itself must be implemented to be flexible for different market segment implementations. Sääksjärvi (1998) has included as a part of his software frame architecture vertical connectors (V1...V4) that provide the ability to implement effective frame leverage for selected market segments. A sign of effective product leverage is when the software vendor does not have to make any changes (or to make only minor changes) in the frame architecture when extending to new market segments. It can be said that an innovative and flexible frame architecture is a good basis for effective platform leverage.

Sääksjärvi (ibid) studied four software companies, all having a very strong and pioneering technology strategy and technology recipe. Two of the four case study companies had very close ties to the end user organization IT infrastructure, while two of them had effective interfaces that isolated the hardware dependent differences. Sääksjärvi (ibid) concludes that a good software application frame will also include the accumulation of core knowledge into the application frame, and that this will help software vendors to expand their business to the next level. According to the study, the selected case study companies all had a core defining technology and core competence in a given domain. This helped these companies to grow, not only domestically, but also internationally. The author concluded in his study that the design of a core application frame is an innovative process and that the process might be different from case to case. This process can be iterative, and the innovation can be based on an existing product, the technology selection, or a specific need in the market.

A follow-up article from Sääksjärvi (2002) introduces a new "integrative concept of software application frame" that "conserves the original objective of the product platform." An application frame is a large reusable core used as base for generation of derivative products that can be installed in the customer environment or be offered as part of a vendor's service to customers (ibid). The software application frame is "an implemented and real construct" to fulfill the criteria of a software application frame (ibid). Sääksjärvi (2002) proposes that a software frame follows four main principles that have been presented in the product platform literature (Wheelwright and Clark, 1992; Meyer and Utterback, 1993; McGrath, 1995, 2001). These four principles are as follows (Sääksjärvi, 2002):

1. The aim of the software application frame is to improve the business effectiveness of application development, applying large-scale reuse of a common application core in several successive products.

2. The application frame is an implementation of the selected architectural style using a set of underlying technologies that will be conserved in all products generated from the frame. The initial frame consists of subsystems and subsystem interfaces that will be incrementally complemented to customer applications.
3. A software application frame extension occurs if particular subsystems within the existing frame will be substantially changed and/or new subsystems are added to existing frame without disturbing the primary subsystems and interfaces.
4. A software application frame renewal occurs when the application is redesigned to incorporate new major subsystems and interfaces.

These four principles define a software application frame, but according to Sääksjärvi (ibid), the complexities in software development due to frame architecture, frame extensions, implementation and process technology require a software vendor to view these simultaneously. Sääksjärvi (ibid) proposes an integrated business strategy whereby application frame architecture, frame leverage, and frame technology sub-strategy should be aligned (integrated) with each other (see Fig. 5).

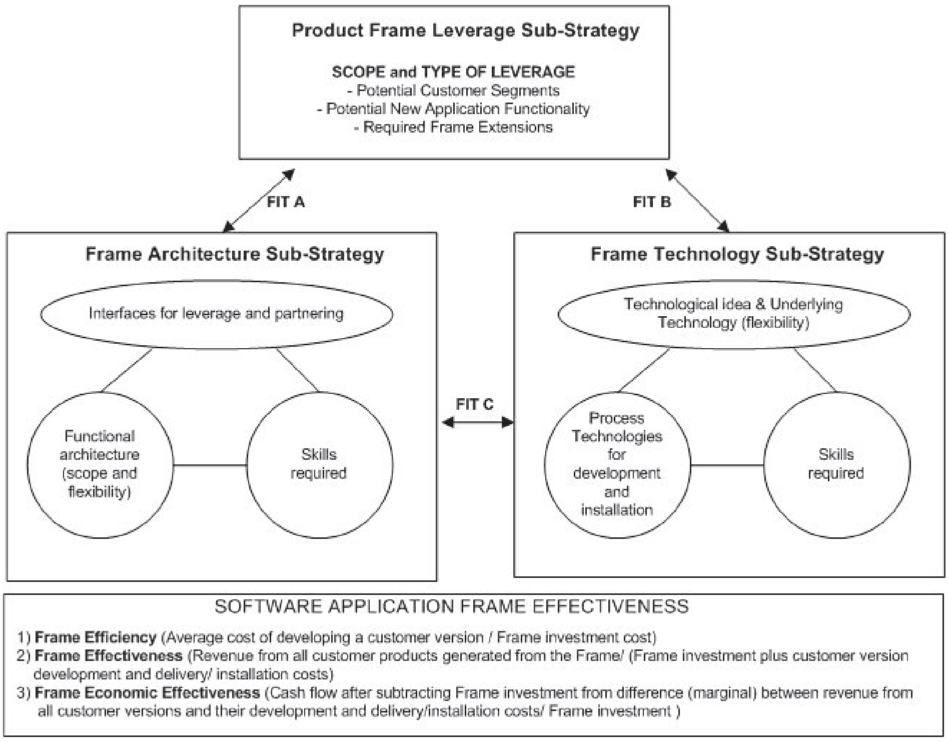

Product Frame Leverage Sub-Strategy
SCOPE and TYPE OF LEVERAGE
- Potential Customer Segments
- Potential New Application Functionality
- Required Frame Extensions
FIT A
FIT B
Frame Architecture Sub-Strategy
Interfaces for leverage and partnering
Functional architecture (scope and flexibility)
Skills required
FIT C
Frame Technology Sub-Strategy
Technological idea & Underlying Technology (flexibility)
Process Technologies for development and installation
Skills required
SOFTWARE APPLICATION FRAME EFFECTIVENESS
1) Frame Efficiency (Average cost of developing a customer version / Frame investment cost)
2) Frame Effectiveness (Revenue from all customer products generated from the Frame/ (Frame investment plus customer version development and delivery/ installation costs)
3) Frame Economic Effectiveness (Cash flow after subtracting Frame investment from difference (marginal) between revenue from all customer versions and their development and delivery/installation costs/ Frame investment)

Fig. 5. Software Application Frame Strategy (Sääksjärvi, 2002).

Sääksjärvi (2002) proposes in his work that there are three different fits to be considered and improved to achieve an integrated software application frame strategy. He proposed that the fit between frame architecture and frame technology sub-strategy be called "Cohesion of frame architecture and technology," the fit between frame leverage and frame architecture be called "Responsiveness of the frame architecture to leverage" and the fit between frame technology and leverage be called "Flexibility of technology for leverage." Sääksjärvi (ibid) presented without any empirical consideration three generic alignment perspectives (processes) in his Software Frame Strategy framework, wherein each alignment perspective portrays specific characteristics of the alignment process. The first alignment perspective – "Implementation of leverage potential" – responds to the typical customer requirement based-design and the implementation of frame architecture. The second alignment perspective – "Technology transformation" – is used in new market segments, and the third alignment perspective – "Frame architecture renewal" includes new market segments.

Sääksjärvi includes discussion of the continuous need for re-alignment of the three sub-strategies within the software application frame strategy as customer requirements and technologies change. Sääksjärvi concludes that the traditional software process approach follows "different effectiveness logic than the structural (architectural) product frame approach offered by the original product platform theory." For this purpose he proposed that the product platform-based engineering oriented effectiveness norms (platform efficiency and effectiveness) be expanded to more carefully calculate economic effectiveness.

2.4 Findings and Comparison of Product Platform Related Theories

To facilitate comparison as part of our research into existing software related product platform theory, we created a comparison matrix with comparison criteria, including characteristics specific to software development (Appendix 1). A common theme for all existing research was the lack of discussion of the technical implementation of software products when using product platform approach. Our research in the existing software related literature showed two sources that were specifically written from a software development perspective – those of Meyer and Seliger (1998) and Sääksjärvi (1998, 2002). Both papers discussed issues specific to software development, including software architecture, market segmentation, etc. Sääksjärvi also included discussion of the complexities associated with underlying IT infrastructure technology in software development and the importance of aligning technology sub-strategy with a frame architecture sub-strategy and a frame leverage sub-strategy, as each of these strategies will impact the overall product development strategy.

A study of the existing software related product platform literature showed that the definition of a product platform varied greatly among the different authors. Meyer and Lopez (1995) did not clearly define a process platform – they conclude that "the design concepts comprising a product's architecture are physically implemented as product and process platforms." Meyer and Seliger (1998) dropped completely the concept of process platform in their paper, even though that was very strongly emphasized in the paper of Meyer and Zack (1996). Sääksjärvi (2002) recognized the importance of separating the frame architecture from the software development process, as both of these have completely different requirements – these two must be able to be aligned with each in an integrated (balanced) way to provide maximal frame leverage to different market segments.

Another interesting finding is in the paper of Meyer and Seliger (1998). The authors did not include common applications as part of the product platform. We argue that this could be a mistake by Meyer and Seliger (ibid), as common applications that are shared by all market segments should be included as part of the software product platform and will be used across different derivative products within a product line. All of the existing literature sources discussed platform extensions and platform renewal, but this discussion was not specifically geared toward software development and the complexities of software variation techniques within software product line development.

Meyer and Seliger (ibid) include discussion of software component development as a basis for building a product platform. This discussion does not relate to any specific technology, nor do the authors discuss how this type of software development could be implemented using software product line development.

The complexity of incremental software development for a product platform with known complexities such as dependency of software components in a multi-layered architectural model is not discussed in any of the existing product platform articles. Software products specifically within the distributed computing environment could have a complex architecture. Each layer of software components could impact the overall functionality of the software product platform and the decision of when to renew a product platform or when to extend it to avoid breakages in the overall software solution.

None of the existing software related product platform papers discussed the granularity of a product platform and its application frame(s). This type of discussion is part of the process of software asset development that is discussed in the software product line literature. Part of designing a product platform is to define the commonalities across different software products. These commonalities are then implemented as part of the software assets used across different product lines. We will discuss software related product line development as one possible implementation approach for software application frames in Chapter 5.

Several software product platform papers discuss the use of external third-party development organizations, and some authors even conclude that platform leadership can only be achieved by having a strong product platform that will be extended by third-party innovation (Gawer, 2000; Gawer and Cusumano, 2002). Software organizations must decide whether software product platforms are to be used as internal productivity accelerators or if software organizations are going to use product platforms both to accelerate external innovation and achieve lock-in of software architecture with respect to the general market. Examples of this type of product platform leadership and development can be found from several organizations, such as the software vendors behind Palm software and Visio. These software solutions provide an underlying product platform that external software development organizations use as the foundation for their own complementary software development or add-in development, as Meyer and Seliger (1998) noted in their paper. According to Sääksjärvi (1998), software development of the product platform itself must be controlled by the software product platform provider, as the evolution of the platform has to be controlled carefully. This type of software product platform control was also identified in the work of Gawer and Cusumano (2002) when analyzing software product platform development at Microsoft.

If the software development organization decides to externalize any development that is linked to its defining technology, the vector of differentiation could have a negative impact on the software organization, as part of the core competence will be externalized. Usage of third-party development organizations requires a strong partnering strategy and a well-defined software architecture with a strong software development strategy. A software product platform that is exposed to third-party developers requires effective application programming interfaces or connection points (Pronk, 2000). The externalization decision is also very important for a software vendor. As long as the application programming interfaces are unpublished, the software vendor can make radical changes to the underlying software architecture without having to worry about breaking external software applications. Another important factor for software organizations is to accumulate domain knowledge in the selected software domain, as this will become the key competitive differentiator for the software organization. It is also recommended that the core software platform development be kept internal to the software vendor, specifically if parts of the software platform are developed externally using third-party development organizations.

In the next chapter we will discuss closely related software engineering theories that can be used when implementing software application frames with corresponding software assets. These theories will be closely related to software product line engineering with domain and application engineering (the foundation for building a product line) and component-based software engineering that enables software organizations to build software assets of different granularity levels. The chapter is an introduction to the topics surrounding the implementation techniques of an application frame. Chapter 5 will explore application frame variation in more detail, using both software engineering and application solution variation techniques with associated software asset development.

2.5 Platform Metrics in Product Platform Development

Several authors have discussed product platform effectiveness (Baker and Freeland, 1975; Brown and Svenson, 1975; Cordero, 1990). Meyer et al. (1997) propose two metrics for measuring the performance of research and development, namely platform efficiency and platform effectiveness. They argue that existing knowledge in R&D performance measurement does not help management to understand "the longer term dynamics of evolving product lines, the renewal of their underlying architectures, and the leverage that these architectures provide in derivative products." The authors (ibid) define the effectiveness of a company's new product development activity as:

- Its ability to create a continuous stream of successful new products over an extended period to time
- The attractiveness of these products to the firm's chosen markets.

According to the Myer et al. (ibid), platform efficiency measures "the degree [to which] a platform allows economical generation of derivative products." Platform effectiveness measures "the degree to which the products based on product platform produce revenue for the company relative to the cost of developing those products." The authors rightfully argue that measurement of platform development could result to inconsistent results if the measurements are based on profit and not sales, as the authors suggest. "Profit" is measured differently in different organizations, which could lead to inconsistencies if these organizations are compared. The authors conclude that platform measurements can be applied at three levels of the product family:

- Individual products within a product platform version of a product family
- At an aggregated level for the product family as a whole for successive platform versions
- Comparatively across different platform versions for different product families.

Platform Efficiency. Meyer et al. (ibid) define overall platform efficiency as R&D costs for derivative products divided by R&D costs for the platform version:

$$\textit{Platform Efficiency} = \frac{\text{R \& D costs for a derivative product}}{\text{R \& D costs for platform version}}$$

This formula measures how much the derivative product cost in relation to the cost of the platform itself. According to the authors, different industries will have different results. In their case study company, the authors measured an efficiency value of 0.10 or less, which indicated that it took less than 10 percent of overall product platform costs to create a derivative product. Needless to say, if the cost of a derivative product is higher than or similar to the cost of the product platform, the platform is poorly designed and the organization should consider replacing it. This platform efficiency ratio can also be used to measure the weaknesses of the underlying product architecture.

Platform Effectiveness. Meyer et al. (1997) measure platform effectiveness as the accumulated sales of products or groups of products divided by development costs:

$$\textit{Platform Effectiveness} = \frac{\text{Total sales of all products derived from the product platform}}{\text{Development costs of platform, platform extensions and its derivate products}}$$

The authors discuss signs of declining product platform effectiveness and its linkage to platform efficiency. Declining platform efficiency might well be a sign of a need to renew the underlying product platform. External technologies might threaten the vendor and its competitiveness, as competitors are using new technology and achieving high platform effectiveness, while the subject company is using old technology in its development.

Meyer et al. (ibid) conclude in their article that they did not provide any definite answers as to when management should decide to create a new product platform, but the concepts and metrics in the article help managers to determine when product platforms should be made obsolete and how to calculate the efficiency and effectiveness of the platform. They also conclude that companies do not typically gather basic data on their product development efforts. They found it difficult to collect data on direct product development related costs, as these are not collected in most companies. The study by Meyer et al. did not cover specific needs and measurements of software vendors. Our claim is that additional dimensions on the effectiveness and efficiency measures need to be implemented for software companies. These measures will be closely related to software engineering and component-based development. The framework proposed by Meyer et al. (1997) gives a basis from which software companies can evaluate the success of their product platform development.

Meyer and Lehnerd (1997) suggest that companies should measure the profitability of derivative products. They have defined a cost price ratio (CPR) that is the quotient of costs of goods sold and net sales. Obviously, each company will define its own way of calculating the cost of goods sold and net sales, and this is why Meyer et al. (1997) decided to use sales as the basis for their product platform efficiency calculations. The authors conclude that a basic problem in calculating product platform and product profitability is the fact that companies do not track expenses and sales in a similar manner and continuously across different years.

2.6 Other Useful Metrics Applied to Software Application Frame Development

According to Herzum and Sims (2000), software development is still viewed as a craft in many software organizations: a programmer is still an artist. This type of software development is inevitably going to change as software organizations aim to increase their productivity with corresponding decreased time-to-market for their software solutions. Another challenge for software organizations is that of implementing accurate project tracking applications which enable software organizations to collect and measure every activity that relates to a given product so as to be able to calculate product profitability. Unfortunately, according to our experience, tracking projects with their associated software development phases is viewed as being too burdensome amongst software engineers, resulting in inadequate measurements.

According to Kaplan and Cooper (1998) some companies have cost systems that are inadequate even for financial reporting purposes: transactions are "either not recorded or are recorded incorrectly." Meyer and Lehnerd (1997) conclude that organizations typically maintain profitability numbers for their business units, but only a few organizations can demonstrate profitability on individual products. According to Meyer and Lehnerd (ibid), gross margin is typically a feasible measure and starting point for calculation of product profitability.

This chapter is divided into two sub-chapters. The first sub-chapter discusses the challenges of traditional product profitability measures, while the second sub-chapter relates to metrics derived from product platform theory.

2.6.1 Economic Metrics for Software Businesses

Economic metrics for software vendors can in some cases be considerably different from those of manufacturing organizations. A good example of this is gross margin percentage. For software companies, this measure could be as high as 99%, while manufacturing organizations have to include the direct costs of materials, which vary with sales. Software organizations can have different business models: some organizations concentrate on software product sales, while others might have service-oriented models or even a combination of these (hybrid). According to Cusumano (2004), software service organizations will have much lower gross profit margins on sales due to the costs associated with sales (direct labor etc.). Gross margin percentages as low as 30% are not unusual, and therefore Cusumano (ibid) suggests that software products sales are "generally much more profitable than services and maintenance revenues and easier to grow without adding head count." In a similar manner Cusumano (ibid) concludes that gross profit margin does not provide enough information about the economic state of a software company given other considerable cost items such as research and development costs, sales and marketing costs, and general administration costs. According to Cusumano (ibid), sales and marketing costs should be between 20-30 percent of total revenues, research and development costs about 10 to 15 percent, and general administration costs about 5 percent. It is very important to understand that different software business models have different foundations for evaluating the metrics, as service-oriented organizations are very different from software product companies.

Calculation of product profitability using traditional cost accounting practices emphasizes the collection of all direct costs related to the development of the product. Within software organizations, the largest separate cost item is the labor cost (both internal and external). In some cases, major parts of the labor costs could be based on software outsourcing contracts or other similar arrangements that the software vendor has in development of the products. Once all direct costs are allocated to the individual products, software vendors have to allocate indirect costs such as administration costs, marketing, software support, and other overhead costs to the software product. These allocation rules can vary between software organizations, as each of these has its own characteristics. A typical allocation basis is to use sales revenue as the driver, as it is impossible to know exactly how these general costs can be allocated to each individual product. Several different cost accounting practices have been introduced during the last few years, including activity-based management (ABM), activity-based costing (ABC), and target costing (Kaplan and Cooper, 1998). Each practice aims to help organizations to set appropriate allocation rules for cost allocation.

The character of a software organization compared to a manufacturing organization with physical products is very different, as manufacturing organizations have to deal with raw materials and other variable costs that are very dependent on overall revenue. Because of this, economic measures such as a gross margin of 30% could be typical for a manufacturing company, while a software company gross margin might be 95% (Bragg, 2002). According to Cusumano (2004), the cost of software license fees (software license revenue) includes mainly items such as "compact discs, printed manuals, packaging, freight, inventory, third-party royalties, and amortization expense related to capitalized software development costs" (ibid).

In the case of software organizations, revenue numbers do not drive the direct costs, and therefore a software vendor must control the amount of investment that is put into any specific product or product family. A traditional measurement for organizations to use is gross margin, reflecting the margin when the direct cost of sales is subtracted from total revenue. This gross margin must be high enough to cover all of the overhead costs, such as administration and marketing, to enable a software vendor to contribute a healthy operating margin.

Gross margin measurement is a textbook example of a profitability calculation in any organization regardless of business domain, and therefore we find it interesting to complement this measurement with the measurements that have been introduced by product platform theory. Both of these measurement types (traditional margin metrics and product platform metrics) are highly dependent on how well the software organization has been able to record internal transactions, with correct allocation to each and every product within a product family. Gross margin can be calculated in a given currency or as percentage of total revenue, and in some cases it might be of interest for the software vendor to calculate the share percentage of the overall gross margin that a given product has been able to contribute. This could in some cases reveal that some products that the company is heavily invested in might not contribute enough to the bottom line for the software vendor.

Gross margin percentage enables executive management to compare different products within a product family or compare different product families with each other. These measurements do not give the management any idea of long-term profitability and possible leveraging of the product platform, and this is why additional product platform specific measurements have been introduced by Meyer et al. (1997) to demonstrate long-term survivability of a product family. According to Uusi-Rauva (1989), the role of product calculation is to allocate direct costs to comparable sales revenue so as to be able to decide upon product assortment and market segmentation.

Regardless of the selected economic metrics, a software vendor must have the ability to trace product related historical costs and revenues. Software vendors need a solid project tracking application that will not only track time usage per function, but the time must be allotted separately to products or product families. According to Meyer and Lehnerd (1997), gathering this type of information could be very laborious, and in some cases could take months to accomplish, with the risk of considerable inaccuracy. If the company does not have records or other means for cost allocation, the management of the software organization must use its subjective assessment of whether the investment in the product has been reasonable. Meyer and Lehnerd (ibid) suggest that organizations maintain a simple spreadsheet or database to keep this data with corresponding line items for each product platform version with reference to each corresponding product and product family. Each line item should additionally include all costs drivers that have been included in the calculations (engineering, development, sales etc.). If the software vendor uses external third-party software contractors in product development, these costs have to be allocated to each function and product. Meyer et al. (ibid) concluded that profitability calculations are hard to measure because organizations do not track sales per product or might not have enough cost information on which to base their calculations.

2.6.2 Extended Product Platform Metrics for Software Products

Traditional product platform related metrics were introduced in prior chapters with both platform efficiency and effectiveness ratios. To be able to calculate these ratios, a software vendor should collect all costs related to the development of the software application frame (the product platform) and total revenue generated from the products that are derived from the software application frame. The aim of an analytical application software solution is to provide prepackaged analytics and business processes that a given vertical market segment can utilize "out-of-the-box." The reality for a software vendor is typically that 80% of the requirements are built into the software solution while the remaining 20% have to be manually built, either by the end user organization or alternatively by the software organization, yielding installation revenue. We have defined this type of revenue as service revenue. Where a software company realizes both software product revenue and service revenue, the management should measure these two types of revenues over time. If the service revenue goes up over time, it could be due to the increased complexity of the software application and/or a decrease in software license sales. Due to the specifics of analytical application software (which could be any software domain with installation activity) we decided to include effectiveness ratios without and with service revenue. Depending on the software solution, service revenue could be considerable when compared to the total revenue of software sales. This could be a sign of weakness in the software application frame.

The aim for a software vendor is to retain its customer base and realize considerable annual maintenance and support payments. An important measurement for a software vendor is to accumulate the number of customers the vendor has been able to acquire during the lifetime of the software application frame. This measurement enables the calculation of software application frame margin (Total revenue from software sales plus service revenue subtracted from total cost of software application frame development). When the software application frame margin is divided by the number of customers, the software vendor is able to calculate the software application frame's margin per customer. This number should be tracked and measured across a longer period of time. The following measurements will be used in Chapter 7 when analyzing our case study software vendor:

- Total direct costs allocated to a software application frame release
- Total revenue for software products for a given software application frame
- Service revenue associated with the software application frame
- Software application frame margin (total revenue from software sales plus service revenue minus total direct costs of development of the software application frame
- Number of customers
- Software application frame margin per customer (software application frame margin divided by number of customers for the products associated with a given software application frame).

The first measurement portrays the overall costs in the software application frame. This number is also used as the denominator in calculating software application frame efficiency. This number will include all the measurable costs of developing the software application frame, such as personnel costs, costs of outsourcing, etc.

The second measurement is the total revenue for all of the products that have been derived from the software application frame. This number gives us the ability to calculate software application frame effectiveness.

The third measurement is the overall service revenue that has been generated when installing the software solution in the end user organization IT infrastructure. For some software product domains, this type of revenue might not exist at all. This measurement is needed to calculate the relationship between service revenue and software license revenue. Seasoned software executives might discover that the prior revenue type could easily hide problems associated with the software itself, and this could become a problem for the software vendor in the future. Our recommendation is to calculate software application frame effectiveness numbers with and without service revenue. It could be deceiving to assume that a software application frame is effective by reviewing only software license revenue numbers. The service revenue could be extensive, which typically means that the software is not functionally flexible to install or does not have the functionality that is expected from the software solution. Therefore, the less time the software vendor has to spend tailoring or modifying the software on site, the more efficient the software application frame could become from a leverage perspective. If the service revenue makes up most of the overall revenue, the effectiveness numbers could become meaningless. This type of discussion was not part of the existing product platform and software related application frame literature. Service-related strategy is discussed in Cusumano's (2003) article, where the author explores the differences between a pure software products company and a software company that sells both products and services. According to the author (ibid), service related organizations will require more employees, and if the software company undertakes customization of the software, the more service-oriented it becomes. We argue that this type of software vendor and its business model is very different from a completely product-oriented company, and should therefore be measured and treated more such as a service organization with associated metrics.

The fourth measurement (software application frame margin) is used to evaluate whether the software application frame investment has been paid off or whether it is still "under water," i.e., the investment has not yet been recouped.

The two final (fifth and sixth) measurements give an overall picture of how broadly the application software application frame has been used and what software application frame margin per customer the software vendor has been able to achieve.

2.7 Related Theories in Software Development with Large-Scale Reuse

The aim of this chapter is to discuss what type of issues a software vendor could face when seeking and selecting a development approach for a software application frame and its corresponding software assets. We will assume that development of a software application frame must be based on large-scale reuse as defined in principal one (Sääksjärvi, 2002) for a software application frame. We will initiate our discussion by reflecting on the business case that a software vendor must set to achieve economic benefits in its derivative software development. Secondly, as part of building software assets for an application frame with the product line/product family engineering approach, a software vendor has to identify the commonalities and variabilities for a product line, and domain engineering with application engineering is typically applied to achieve this. A solid domain model with corresponding architecture and software assets would be useless without an understanding of technical means to implement a software application frame and its software assets. Therefore, we will introduce component-based software engineering (CBSE) as a vehicle to achieve effective software engineering results.

2.7.1 Building a Business Case for Large-Scale Reuse

Clements and Northrop (2002) discuss the importance of building a business case for building software products, and specifically for large-scale reuse. Schmidt (2003) concludes that any business case must have critical success factors and contingencies to enable predictable results. According to Clements and Northrop (ibid), a business plan should address following key questions:

- What are the specific changes that must occur?
- What are the benefits of making the change?
- What are the costs and risks?
- How do we measure success?

Schmidt (ibid) argues that a business case should include several summarized, well-defined financial metrics such as net cash flow, discounted cash flow, internal rate of return, and payback period. Weiss and Lai (1999) illustrate the cost model for product lines versus single-systems. With this model, the cumulative cost without a product line rises steadily from zero, while with the product line approach the cumulative cost begins with the initial cost of building the core assets and then climbs at a shallower rate as new products are introduced. This type of discussion, together with business case and other related documentation, is important when deciding on investing in product line development using a software application frame.

Based on the aim of effective software application frame leverage using different variation techniques, a software vendor must identify technologies and methodologies that will support effective implementation of software application frames. Sääkjärvi (1998, 2002) collected several principles from existing product platform related literature and used this to address software-related application frame construction. According to Sääksjärvi (ibid), the aim of a software application frame is to apply "large-scale reuse of a common application core in several successive products." Based on this requirement of large-scale reuse, we need to identify practical means of implementing an application frame using software engineering related technologies. There is a lot of research within software reuse. Van der Linden (2002a) shows that "an increasing number of assets other than software – for example, designs, patterns, requirements, test specifications, and test results – become reusable." According to Mili et al. (2002), Henderson-Sellers and Edwards (1990) argue that the traditional waterfall lifecycle has met criticism, as "each lifecycles stage is influenced mainly by the previous stages (top-down), while existence of reusable components requires some sort of look-ahead procedure to identify opportunities for reuse and take advantage of them." For Mili et al. (ibid), reuse has traditionally meant reuse of small code fragments due to inadequate documentation, but with effective use of component libraries, organizations could achieve reuse on a larger scale. According to Henderson and Edwards (ibid), object-oriented software development requires a mix of top-down and bottom-up approaches. Van der Linden portrays the historical relationship between the level of reuse and the development paradigm (see Fig. 6).

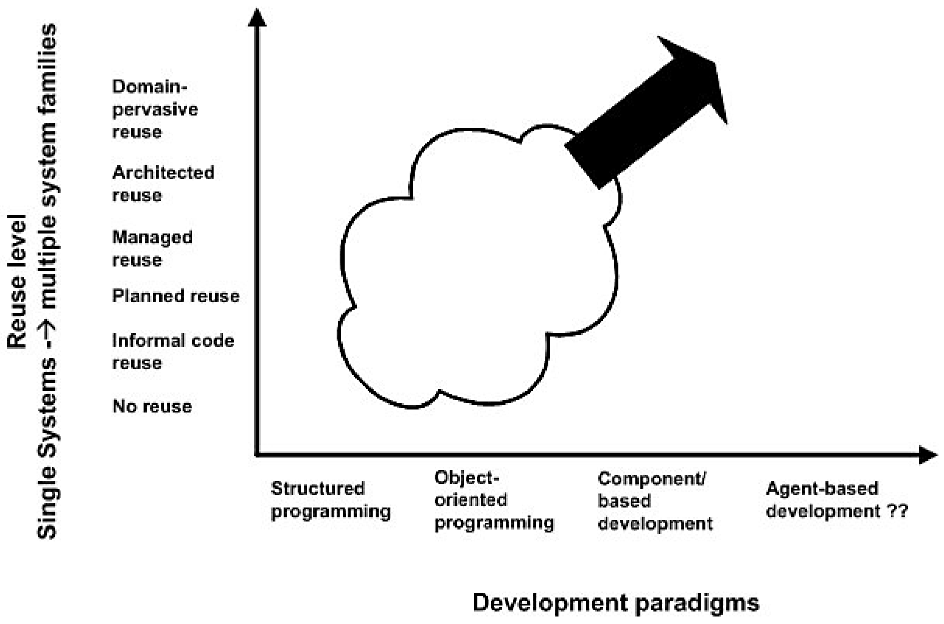


Fig. 6. Moving from Single Systems to Multiple System Families (van der Linden, 2002a).

Figure 6 also portrays the evolution of development paradigms on the x-axis, where the software engineering has evolved from structured programming and moved towards larger granular reuse using the component-based software development paradigm. We will initiate our discussion by exploring and defining software product line (product family) engineering, which we argue is a solid proponent and foundation in the technical implementation approach to a software application frame. This discussion is then expanded into domain engineering and application engineering, which are needed in product line engineering efforts. Finally, we will include discussion of component-based software engineering as a solid proponent for technical implementation of software assets within a software application frame.

2.7.2 Product Line Engineering Using Product Families

Jacobsen et al. (1997) introduced the concept of Application Family Engineering, which he defines as "a process that determines how to decompose the overall set of applications into a suite of application systems and supporting component systems." Jacobsen et al. (ibid) explain that Application Family Engineering has been inspired by Kruchten (1995) and his famous article "The 4+1 view model of architecture." A key for Application Family engineering is to "identify the architecture and a set of components systems that will support the suite of applications to be built." To accomplish the aims of Application Family Engineering, Jacobsen et al. (ibid) define two other engineering disciplines as Application System Engineering (ASE) and Component System Engineering (CSE). According to Wijnstra (2002), Application Family Engineering is responsible for product family architecture and reuse, while Component System Engineering "focuses on the development, maintenance and administration of reusable assets within the platform." The role of Application System Engineering is to focus on the development of the end products using the two former engineering practices.

According to Thiel and Peruzzi (2000), their research on product line engineering was inspired both by the work of Jacobson et al. (1997) on Reuse-Based Software Engineering and the work of Kang and colleagues on Feature-Oriented Domain Analysis (Kang et al, 1990). This collaborative work is very closely related to engineering practice product line engineering. Product line engineering has become an increasingly important research topic during the 1990s. The Software Engineering Institute (SEI) at Carnegie Mellon University has released several reports on product line software development and related topics such as software architecture, domain engineering etc. (Bass et al, 1997; 1998a; 1999). This research initiative from SEI was launched in January 1997 to help to "facilitate and accelerate the transition to sound software engineering practices using a product line approach" (Bass et al., 1999). The aim of this program is to help organizations to produce similar systems of predictable quality at lower cost. Bass et al. (ibid) conclude that the increasing competition among software vendors has forced these vendors to introduce new products and functionality at an increasing speed, and product line development helps these organizations to reach this goal. Bass et al. (1997, 1999) concluded that an increasing number of organizations cannot afford to develop multiple software products one product at a time, and therefore many organizations have introduced the concept of product line development. Clements and Northrop (2002) define a product line as follows:

> "A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market or mission and that are developed from a common set of core assets in a prescribed way."

It is clear from our literature research that existing product line engineering literature defines product line engineering and product family engineering differently. To Clements and Northrop (ibid) a product family is "a set of systems built from a common set of software assets." Geyer and Becker (2002) conclude that a product family is based on the same reuse infrastructure. According to van der Linden (2002b) the term product family and product line are used differently in some European companies. Van der Linden (ibid) defines product line and product family concepts in a manner similar to that of Geyer and Becker (ibid) by stating that a product line indicates "a set of related, commercial products that appear similar to users but often are built using different technologies," while a product family describes "a collection of products that are based on the same technology." Similar comments were presented by Northrop (2002), where different practitioners use different terms but the aim is the same (ibid):

> "Regardless of the terminology, software product line practice involves strategic, large-grained reuse, which means that software product lines are as much about business practices as they are about technical practices."

Given the different product line and product family engineering definitions, we will take the same approach as Northrop (2002) did when comparing product line and product family definitions. The aim of product line or product family engineering should be implementation of a software application frame involving large-scale reuse. We will use product line and product family interchangeably when analyzing derivative product development of analytical application software from a common analytical application frame. When we compare the definitions of product line and product family using the Geyer and Becker (2002) and van der Linden (2002b) approaches, we conclude that the product family is closer to the concept of software application frame implementation. This is based on the fact that derivative products from a software application frame are based on same underlying technology as defined in the product family and not only on same type of functionality, as was the case with product line engineering.

Bosch (2000) introduces two different ways of initiating a product line development effort. The first approach is to evolve the existing set of products into a product line. In this process, components are identified that fulfill the requirements of more than one product. Once these components have been identified, they will be adapted into existing products using these shared components. The more radical approach is more revolutionary, where the existing set of products is replaced by a product line architecture and set of components. This approach is more risky, as the initial investment is typically large and as requirements might have changed during the development time. This revolutionary approach is also more economical, as the evolutionary model typically requires longer development time and larger investment. This type of evolutionary development can also be linked to discussion of software application frame renewal, as software organizations might in some cases be better off by completely replacing an existing software application frame: the existing software application frame might be too expensive to maintain and might not allow future feature development.

Several research results have been published about the experiences of product line development (Macala et al., 1996; Dikel et al., 1997; Bosch, 1999a; 1999b; Bosch and Högström, 2000). The results show clearly that several important core competences are needed when implementing product lines, such as domain-specific expertise, strong architectural knowledge, management of software requirements, and configuration management. Key factors for successful product line development are (Macala et al. 1996, Dickel et al. 1997, Bosch 2000, Bass et al. 1999):

- Domain-Specific Expertise
- Architectures
- Configuration Management
- Business Models
- Scoping the Domain
- Avoid the "Least Common Dominator" Concept
- Managing Requirements in the Product-Line Context
- A Separate Domain Engineering Unit

According to several publications, the results of product line development have increased the productivity of product development. Thiel and Hein (2002) discuss the pervasiveness of variability when operating a product line. Jaaksi (2002) discusses via a case study how a mobile phone manufacturer has achieved product line development in mobile browsers. Other software product line related articles have been published, such as discussion of industry-specific software product line architectures (Bosch, 1999a), product line software components (Bosch, 1999b) and evolution of product lines (Svahnberg and Bosch, 1999). Lopez-Herrejon and Batory (2001) propose a standard problem to evaluate product line methodologies and conclude that several methodologies have been implemented to create product line architectures. The authors (ibid) also conclude that the "state-of-the-art" is still immature and requires further development.

According to Mili et al. (2002), the lifecycle of a product family has different characteristics when compared with traditional software lifecycles. A product family lifecycle is:

- Architecture-Based
- Economically Driven
- Reuse-Driven
- Domain-Specific
- Process-Driven
- A Result of a Producer-Consumer Relationship

According to Mili et al. (ibid), the product line development lifecycle has two main phases. The first phase is domain engineering, whereby the software organizations produce common artifacts. The second phase is application engineering, which consumes the artifacts from domain engineering (Arango, 1994) with the intention of building a product family. The aim of the next sub-chapter is to explain how domain engineering and application engineering are used within product line engineering.

2.7.3 Domain Engineering with Application Engineering

As the software application frame approach implies reuse of large software entities in the form of software application frame and corresponding frame components, a method is needed to extract commonality amongst existing or planned products within the product line. Several practitioners and software engineering sources (Arango, 1994; Griss, 1996; 2000a; Griss et al., 1998; Pressman, 2000; Svahnberg and Bengtsson, 2000) consider domain engineering a technique "to systematically extract features from existing or planned members of a product line." Within domain engineering, software designers "use clusters of sets of features to shape the design of a set of components that cover the product line" (Griss, 2000a). According to Shlaer and Mellor (1988), there are four different types of domains within systems:

- Application domains, dealing exclusively with business logic
- Service domains, which embody things such as user interfaces, communications, event messaging, and other general utilities
- Architectural domains, which embody architectural choices in terms of specific meta level artifacts (templates, patterns, etc.) and guidelines
- Implementation domains, which include programming languages (and language constructs), networks, operating systems, and common class libraries.

Several domain analysis methods have been introduced to the software engineering community, including Feature-Oriented Domain Analysis FODA (Kang et al, 1990), Feature Reuse-driven Software Engineering FeatuRSEB (Griss et al., 1998), Feature-Oriented Reuse Method FORM (Kang et al, 1998) and several others (Griss, 2000a). Mili et al. (2002) recognize Reuse Business Methodology from Jacobsen et al. (1997) as a method with a set of guidelines and models to help large-scale reuse. According to the authors (ibid), the Reuse Business Methodology does not include domain engineering as such, but "distributes the main domain engineering process between application family engineering and component system engineering."

Clements and Northrop (1999) define domain analysis as "the process of identifying, collecting, organizing, and representing relevant information in a domain for the purpose of making that information reusable in a variety of contexts." The output of a domain analysis is a domain model, which defines the functions, objects, data, and relationships in a domain. The authors argue that a domain model will help organizations to capture the commonalities and variability of related software systems. Development of a product requires knowledge from several different domains. The software vendor must be able to merge the skills of each domain to be able to produce successful software products. Domain analysis is dependent on two factors, namely the depth of the organization's domain experience and the amount of resources that the organization can devote to the analysis.

Sametinger (1997) concludes that domain engineering can be seen as "a continuing process of creating and maintaining the reuse infrastructure in a certain domain." He classifies domains into two different categories: vertical domains and horizontal domains. Vertical domains typically address a specific class of systems, while horizontal domains are based on generic software components used across different vertical domains. Sametinger (ibid) concludes that common objects (or components) will occur in multiple applications, and that domain analysis will basically identify the commonalities. He wants to separate domain engineering from application engineering. Domain engineering is the process whereby the software engineer identifies the commonalities between similar systems within a problem domain, while application engineering takes the results of domain engineering into the software development process. The domain engineering process will identify the reuse opportunities in the design of the applications themselves. Similarly, Mili et al. (2002) conclude that the technical aspects of product line engineering can be classified into domain engineering and application engineering (see Fig. 7).

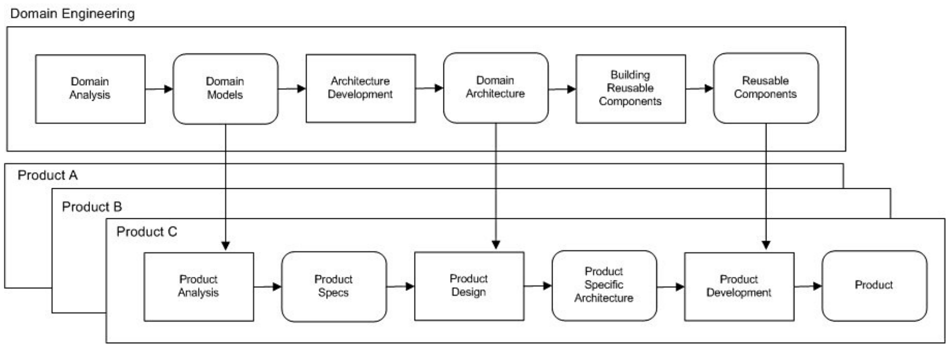


Fig. 7. The PLE Lifecycle as Domain and Application Engineering Phases (Mili et al, 2002).

According to the authors (ibid), domain engineering "includes activities such as domain analysis and definition, development of domain (reference architecture), and development of (core) assets (components)." Furthermore (ibid), "application engineering includes application requirements analysis, instantiation of the reference architecture, instantiation and customization of domain components, and component-based development." One of the key elements in domain engineering is the development and definition of reference architecture that will be the basis for instantiation of products within a product line. Without a solid reference architecture, the basis for the products will be weak and cause difficulties. As product line engineering aims to achieve reuse in large-scale with derivative product development, the next chapter discusses the possible building components and techniques for building software assets for a software application frame in a product line or product family.

2.7.4 Component-Based Software Engineering (CBSE)

Several books on large scale component-based software development have been released, advocating either large-scale component-based development, a software factory approach, or product line development to manage software development on larger scale (compared with the traditional object-oriented development approach) (D'Souza and Wills, 1999; Jaaksi et al, 1999; Brown, 2000; Herzum and Sims, 2000; Clements and Northrop, 2002; Atkinson et al., 2002). According to Bosch and Högström (2000), during the second half of the 1980's, the software development community identified the need for software components to be of larger granularity levels than functions and classes, and therefore new concepts such as object-oriented frameworks (Johnson and Foote, 1988; Fayad et al. 1999) were introduced to improve reuse of common software assets. According to Mili et al. (2002), an application framework "may be roughly defined as a set of interacting objects, together, realizing a set of functions." The authors (ibid) concluded that a set of functions defines "the area of expertise or competence of the framework." Frameworks can be divided into business domains or computing domains, where the latter domains can be represented by frameworks such as the Model-View-Controller (MVC) framework.

Component-based development has been studied and discussed in several books (Szyperski, 1997, 1998, 2003; Sametinger, 1997; Pfister and Szyperski, 1998; Hopkins, 2000; Vayda, 2000; Broy et al, 1998). The common thread in these studies is the attempt to find ways to increase software development productivity for software development organizations. The software development community has not reached a true maturity level in a manner similar to that of the manufacturers of physical products. A sign of a mature software industry can be measured in several different ways: some organizations might measure how well software can substitute for components within the software product without breaking linkages to other software components. Other software organizations might act as brokers for commercial software components in fashion similar to that of car part dealers, who compete with each other on the OEM and after sale markets for car parts. According to Pressman (2000), component-based software engineering enables organizations to increase their software development productivity and increase the quality of software, but that the industry still has to overcome roadblocks before CBSE will be widely used in the industry. According to the Pressman (ibid), component-based software engineering can be divided into two separate processes, namely domain engineering and component-based development. Each of these literature sources emphasizes the importance of creating loosely coupled software components to enable software component substitution in cases where one software component reaches the end of its lifecycle.

Even if software development has increased its maturity, Vayda (2000) argues that the software industry still has a way to go to achieve manufacturing processes similar to those of other industries today. Vayda concludes that the software industry is "the only industry that builds the factory and manufacturing processes at the same time it builds the actual product." Another view is introduced by Wallnau et al. (2002), in which software systems become more dominated by commercial components. This will take some of the control of software architectures from software vendors, as the market will decide which components will succeed and how these components interacts with each other. This change towards component-based software development using commercial software components (COTS commercial off-the-shelf products) will require new approaches from software architects, as some of the control will be outside the development team. This trend will also require increased emphasis on software application programming interfaces and encapsulation of functionality into components to be able to replace software components that are no longer supported and need to be replaced with new components according to the model suggested by Herzum and Sims (2000). According to Clements (1996), components have increased in their size and complexity, and the interaction between components has become better in both communication and interaction.

Software components are categorized differently based on the purpose, functionality, and characteristics that they provide in the software product into which they are implemented. Sametinger (1997) has identified three different component types as a result of a domain analysis: general-purpose components, domain-specific components, and product-specific components. This categorization is according to vertical and horizontal application frame leverage, as general-purpose components can be used in different applications in different domains and domain-specific components can be used within one domain. Additionally, product-specific components might be specific to the application and not reused in any other application. Another categorization is to classify software components according to a selected architectural software model. According to Clements (1996) and Szyperski (2003), component-based development should be based on a layered architectural style. In this layered architecture, the IT infrastructure is the lowest layer, including infrastructure that is needed in building the software solution (such as the operating system and network environment). In the layered architectural model, each layer is allowed to use the components at the same or of the next lower level. This approach basically isolates different layers from each other and enables software organizations to replace components in distant layers more easily.

We will explore the use of component-based software development in more detail in Chapter 5, with additional discussion of product line engineering and component-based software engineering, implementation of software assets, and component granularity and its impact of building application frames. The aim of next chapter is to explore the impact of software metrics for software development specifically in reference to implementation of a software application frame with the use of product line engineering and component-based software engineering.

2.8 Definitions of Concepts and Assumptions in the Study

Part of the core aim of the software application frame is "to improve the effectiveness of application development by applying large-scale reuse of a common application core in several successive products." Within our study, we will be using the term "software application frame" as the concept of product platform that is more commonly used in manufacturing of physical products. Using the same analogy, a software application frame for analytical application software solutions will be named the "analytical application frame," specifically geared to development of analytical application software solutions.

Based on this definition, our aim is to identify an optimal common architectural construct for an analytical application that can be reused across different vertical domains. To be able to define an analytical application frame for an analytical application solution, we must have a thorough understanding of the complexities of these types of solutions. Our definition of an analytical application frame is as follows:

> "An analytical application frame is an optimal architectural construct that is common to all derivative products within an analytical application. This architectural construct is common to all vertical market segments and horizontal business processes that a market segment requires."

Sääksjärvi (2002) defined four principles for a software frame. These principles are more or less generic and do not specifically address analytical application software domain characteristics. As these principles from Sääksjärvi (ibid) are presented without any empirical consideration, we expect to identify characteristics that are specific to analytical application software, its corresponding architectural model, and functionality. Therefore, our aim is to revisit these four principles with respect to the analytical application frame later in this study. These will then be applied in our case study research.

Analytical application software solutions are intended for decision support. Each will have its own specific characteristics and product architecture, which can potentially be very different from other software application domains. The original product platform theory emphasizes the use of product architecture in the development of a product platform. According to Ulrich and Eppinger (1995), product architecture is characterized as follows: "the architecture of a product is the scheme by which the functional elements of the product are arranged into physical chunks (building blocks) and by which the chunks interact." According to Ulrich and Eppinger (ibid) a modular architecture has two main properties: "chunks implement one or a few functional elements in their entirety" and "the interactions between chunks are well defined and are generally fundamental to the primary functions of the product."

Within our study, we will conform to Ulrich and Eppinger's (ibid) definition of product architecture when analyzing possible architectural constructs for analytical application software. Our reasoning for selecting the definition from Ulrich and Eppinger (ibid) goes back to the theory of the product platform, where a "product platform is the physical implementation of a technical design that serves as the base architecture for a series of derivative products" (Meyer and Lopez, 1995) and "a product platform is comprised of subsystems or modules and the interfaces between these modules" (ibid). In the same vein, our aim within this study is to portray a modular architecture with functional components for an analytical application software solution and to demonstrate different interfaces between these modules in the form of application programming interfaces. O'Grady (1999) concluded that an architecture specifies the main modules and their interfaces and that functions within a product can be classified into three different groups, namely core functions, overlapping functions, and individual functions. Modularity can also be reviewed from the granularity perspective, where product architectures with smaller modules can cause software vendors problems due to the larger number of combinations that these modules or subsystems can cause. From the product platform perspective, we will concentrate on and limit our research to explain the role of software product architecture from the granularity perspective using the definition from Ulrich and Eppinger (1995).

There is a great deal of research in software architecture (Clements and Northrop, 2002) specifically by the Software Engineering Institute (SEI). We will exclude discussion of different attributes for software architectures other than those specifically related to analytical application software development. We will discuss the importance of the domain-specific architecture model versus technological architectural style, as these could represent different requirements. We will explain the importance of modular software design (Cusumano, 2004) and the layered software architectural model (Jacobsen et al., 1997). According to Cusumano (ibid) there is a "sliding scale of modularity for almost any complex product," whether it is an automobile or software system. According to Cusumano (ibid), a software module consists of a set of functions and can be separately tested. Within a modular architecture, components are loosely coupled, while in an integral architecture components are tightly coupled and interdependent with each other (Cusumano, ibid).

These conclusions of modular architecture can be related back to software related product platform development, in which a highly granular software architecture with a software application frame must be well tested and used across different market segments. If a software application frame were a "collection of software components," the concept of the product platform would be lost, as the combination of small-grained components would cause a software vendor problems in testing: the possible combinations/configurations between software components grow exponentially. The traditional component-based software development approach has different aims compared with the software application frame approach. The former represents purely a software development approach while the latter is a strategic approach that the software development organization and its management decide upon.

The effectiveness and efficiency of the product platform is dependent on how well derivative products can be built from the product platform in the form of a product family. According to Meyer et al. (1997) a product family can be defined as "a set of products that share common technology and address a related set of market applications." We have to separate defining technology that provides the "vector of differentiation" (McGrath, 2001) for a software vendor from underlying technology that represents the technological choices that an analytical application software vendor makes when selecting its IT infrastructure, such as its database management system, its operating system, and its distributed technology.

Another key question that software vendors might ask themselves is: what type of implementation approach should be used in implementation of a software application frame? Our understanding of an analytical application frame does not improve our understanding of how this can be implemented and deployed to end user organizations. We will explore what types of software assets and development means can be used when building analytical application software solutions using a software application frame as their basis.

Interest in component-based software development has increased, with a corresponding increase in the literature (Larsen, 1999; Stafford and Wolf, 2001; Mili et al., 2002). One of the key aims of this type of development approach is to concentrate on domain-specific functionality with corresponding software assets. Part of this trend is the use of commercial infrastructure software assets that are maintained by organizations that have this as their business model. Other software vendors concentrate on building value-added software as defined by Ryans et al. (2000).

Part of a successful software application frame strategy is to provide a good foundation for derivative software development. According to McGrath (2001), development of a product platform without a solid product line strategy is useless. Therefore, our aim was to explore product line literature on a general level. In Chapter 5, we will explain in more detail how software assets can be built using product line engineering with associated domain engineering and application engineering. Both of these latter engineering technologies are used to define, design, implement, and test software assets within a product line.

2.9 Chapter Summary

In our research into product platform related literature, we were able to re-establish a link between the original product platform theory defined by Wheelwright and Clark (1992) with the current product platform theory emphasizing software development. Wheelwright and Clark have a strong emphasis on product architecture and development of physical products, while Meyer and Utterback (1993) added core capability to the concept of product platform development and development of product families. Ulrich and Eppinger (1995) added the importance of the technology platform and modular development of products. Finally, McGrath (1995, 2001) added the concept of product platform strategy into the theory of product platforms. McGrath (2001) also added additional corporate strategies, including product line, product differentiation, and pricing into the overall concept of the development of high technology products.

However, none of these theories was specifically geared toward software development and the specifics of software products. Therefore, our analysis was specifically aimed at the product platform theory of software products. Meyer and Lopez (1995) wrote the first product platform paper introduced within the domain of software development. This paper introduced technology driven development of UNIX operating systems. This paper was the first attempt to discuss the peculiarities of software development, but the authors did not include any discussion of either process or software development (within the integrated development environment) of the products and how both the internal (development) environment and the external (end user organization) environment might impact the development of a product platform. Meyer and Zack (1996) released a follow-up paper about development of information products, but this paper was more or less geared towards implementation of information contents and not specifically software products. The third and final paper that Marc H. Meyer released with Seliger was released in 1998, with a specific emphasis on product platforms in software development (Meyer and Seliger, 1998). This paper also introduced several other concepts that were not discussed in prior papers, specifically layered software architecture, object-oriented design philosophies, and component-based development.

Based on these existing product platform theories, Sääksjärvi (1998, 2002) concluded that existing product platform theory and product platform theory within software development were not sufficient due to the greater complexities of software development. Sääksjärvi (ibid) concludes that several factors, such as dependence on the underlying IT infrastructure, complexity of software architectures, and other factors within software development are difficult to manage in an overall software development process. Proper management will require consideration of additional dimensions, such as frame technology, frame architecture, and frame leverage sub-strategy. Sääksjärvi (ibid) extended product platform theory by concluding that pure product architecture design with selected technology is not enough in software development: three separate sub-strategies should be carefully aligned (frame architecture, frame technology, and frame leverage sub-strategy).

We also concluded that development of software products is more complicated when compared with manufacturing of physical products for several reasons, such as dependencies in the underlying information technology infrastructure, the software development environment, and software. Some of the existing software related product platform papers identified component-based software engineering as a good basis for building a product platform or a software application frame (Meyer and Seliger, 1998; Sääksjärvi, 2002). Each of these generalizations is far from the complexities that can be found within software product development. Software vendors must select technologies such as the underlying operating and hardware system environment and the software execution environment required when running and using software solutions. Another factor in existing software related product platform literature is the lack of discussion as to whether the software or application domain will impact both the selected software architecture and the product architecture. Existing literature generalizes software product platform development for all software domains, while we argue that each software application domain could have specific characteristics, such as selection of technology, product architecture, and market segmentation.

We also discussed existing product platform related metrics, concluding that these measurements are not specifically geared to development of software products. These measurements could be amended with measurements and research found in software product line engineering with associated component-based software engineering and reuse literature as such. We extended and created general business measurements that will be used in our case study analysis to demonstrate metrics such as gross margin and software application frame margin, and other metrics that will be used in the case study.

One of our findings in the existing product platform literature was the lack of discussion concerning the difference between a software application frame release, an actual product release and their metrics. This discussion is important for analyzing the effectiveness and efficiency of a software application frame. The difficulty of comparing and measuring effectiveness and efficiency measures in different software companies could be challenging, as each software company is different and has different internal product development processes, levels of tracking costs, and time spent in each development phase. We have excluded topics around software reuse in this study, as much research has been done in measuring software asset reuse (Karlsson, 1995; Mili et al., 2002). Unfortunately, this research has not led to any significant results for several different reasons, all well documented.

Based on these findings within the product platform related theory within the software application domain, we have managed to respond to the first research question - How can a software vendor apply the product platform approach to its software business and development of software products? - by providing to the reader a theoretical review of existing software related product platform literature with suggestions for related software engineering approaches that can be used when implementing a software application frame. This chapter provided us with a generic framework for software application development using product platform theory. The next objective is to review analytical application software solutions from an evolutional perspective by analyzing how these types of application solutions have developed from well known decision support architectures such as data warehousing, On-Line Analytical Processing solutions, and traditional **E**xecutive **I**nformation **S**ystems.

The next research question is directed towards our software application research domain – prepackaged analytical application software solutions – and this requires us to define an appropriate architectural model with the expected functionality that these solutions provide. Our following research question is as follows. It will be reviewed in the next chapter:

RQ 2: What types of generic software application frame architectures can be identified for analytical application software solutions?

This research question enables us to achieve a better understanding of our software domain – analytical application solutions – and what type of functionality and architectural models these types of solutions can be built on.

3. ANALYTICAL APPLICATION FRAME ARCHITECTURE DEVELOPMENT

The aim of this chapter is to introduce the software application domain that will be the focus of this study. This domain is analytical application software solutions. We have divided the chapter into four main sections. The first section lays the foundation for analytical application software and technologies that are needed to implement these types of solutions. In the second section we will discuss possible architectural models and implementation approaches that an analytical application software vendor can utilize when implementing analytical application software solutions. The third section defines the use and functionality requirements for an analytical application software solution, and the fourth section defines an Analytical Application Functional Architecture that portrays both functional components within an analytical application solution and how information flow impacts each and every one of these functional components. The first three sections are based on the literature study and have a strong theoretical emphasis, while the last section on functional architecture provides managerial implications with contributions to the existing analytical application software domain. We will explain the complexities of information flow in an analytical application and we will also provide an end user organization perspective in the use of these types of applications.

3.1 Introduction

Decision Support Systems (DSS) have evolved from traditional Executive Information Systems into today's business intelligences solutions, which are both web-enabled and leverage different analytical solutions for different vertical markets. The role of DSS solutions (Scott, 1971) was to help management to make choices based on the information that was delivered typically in the form of Executive Information Systems. Rockard and Delong (1988) defined EIS as the solution for upper management to find problems, while a DSS would provide staff people a vehicle to study these problems and find and offer alternatives for problem solving.

To enable deeper analysis of corporate information, organizations have complemented these Executive Information Systems with implementation of data warehouses that are targeted to decision support and "provide historical, summarized and consolidated data," not detailed records such as typical On-Line Transaction Processing (OLTP) applications (Chaudhuri and Dayal, 1997). According to Inmon (1992), a data warehouse is a "subject-oriented, integrated, time-varying, non-volatile collection of data that is used primarily in organizational decision making." Chaudhuri and Dayal (1997) conclude that "data warehousing is a collection of decision support technology, aimed at enabling the knowledge worker (executive, manager, and analyst) to make better and faster decisions." According to Zachman (1997), the lack of an enterprise wide data architecture resulted in the implementation of a data warehouse solution because of the discontinuity in the existing legacy operational systems. Paller (1996) argues that another reason for data warehousing is to resolve data access difficulties in organizations (Paller, ibid). Data warehousing allows organizations to gather vital corporate information from different operational systems and to provide this information to management. Zachman (1997) asserts that one of the biggest efforts in data warehouse projects is to reverse engineer the often polluted operational data and give this data meaning.

The decision support (and specifically the data warehousing) market has evolved and matured during the last few years. A comparison between the trends in data warehousing from 1997 to 1999 (Gray and Watson, 1997; Watson and Gray, 1998; Gray, 1999; Watson et al., 2000) shows that data warehousing is moving toward the analytical application market space. The market is experiencing a trend similar to that which took place in the late 1980s and early 1990s with Enterprise Resource Planning Applications (ERP). It was very typical for end user organizations to build their own ERP packages, but as this approach was very costly, the market gradually accepted the concept of standardized ERP packages that covered 80 percent of the needs and requirements of an organization. A similar trend is taking place for data warehouse vendors: the market expects delivery of prepackaged analytical applications that cover most of the needs for a given vertical domain (Surgan, 2000). According to Eckerson (1998b), data warehousing has "crossed the chasm:" a late majority (Moore, 1991) of end user organizations contemplate having a data warehouse solution implemented.

These data warehousing technologies are the foundation for analytical application solutions (Gleason, 1998), as software vendors have to select either a data warehouse or a related data mart implementation approach as the foundation for these analytical solutions. Whether an end user organization decides to use data mart technology or data warehouse technology to feed the business analyst requirements depends on end user organization IT infrastructure selections and the organization's information systems strategy. Implementation of an enterprise data warehouse is a "long and complex process, requiring extensive business modeling, and may take many years to succeed" (Chaudhuri and Dayal, 1997). Several organizations have taken another implementation approach by building data marts that departmental subsets focused on selected subjects. According to Chaudhuri and Dayal (1997), implementation of data marts is faster and does not require enterprise-wide consensus. Jarke et al. (2000) conclude that "data marts are small data warehouses, which contain only a subset of the enterprise-wide data warehouse."

According to Watson and Gray (1998), the decision support industry is driving toward data mart technologies with a pre-packaged approach. Even if tools and technologies are improving, end user organizations should avoid building independent data marts, as these could become "the legacy systems of the future" (Watson and Gray, 1998). Gray (1999) concludes that if data mart technology is used, large data warehouse environments should be created "bottom-up." This approach enables end user organizations to create a virtual warehouse with several integrated subject-specific data marts as a basis.

The aim of the next chapter is to portray and discuss decision support technologies that are part of an analytical application software solution. This enables us to achieve a perspective on how analytical application software solutions have evolved during the years and what types of characteristics, such as architectural models, implementation models, and software technologies can be identified when building analytical applications.

3.2 Technologies Supporting Analytical Application Software

The analysis of analytical application solutions in this study is based on the perspective of a software vendor. We will exclude an analysis of implementation methodologies that portray implementation of an analytical application solution in an end user organization. A typical analytical application solution is based on an architectural model where data is collected from different operational applications into a centralized database. The collection process is controlled and managed by functional software components that make the Extraction, Transformation, and Load (ETL) of the decision support database automatic according to a schedule of the end user organization.

An analytical application solution architecture consists of several different technologies. Each and every one of these technologies has a predefined role in the overall architecture, providing the value proposition that is created when an end user analyst uses the information from a common information repository. According to Sprague (1980), a decision support (DSS) solution is composed of three major components: database, model base, and user interface. A centralized information repository is known as a data warehouse and is a core component in an analytical application solution. A data warehouse as such does not provide any value for an end user, but must be supported by technologies such as OLAP with corresponding Executive Information System user interfaces that help business analysts to utilize information from the data warehouse repository (see Fig. 8).

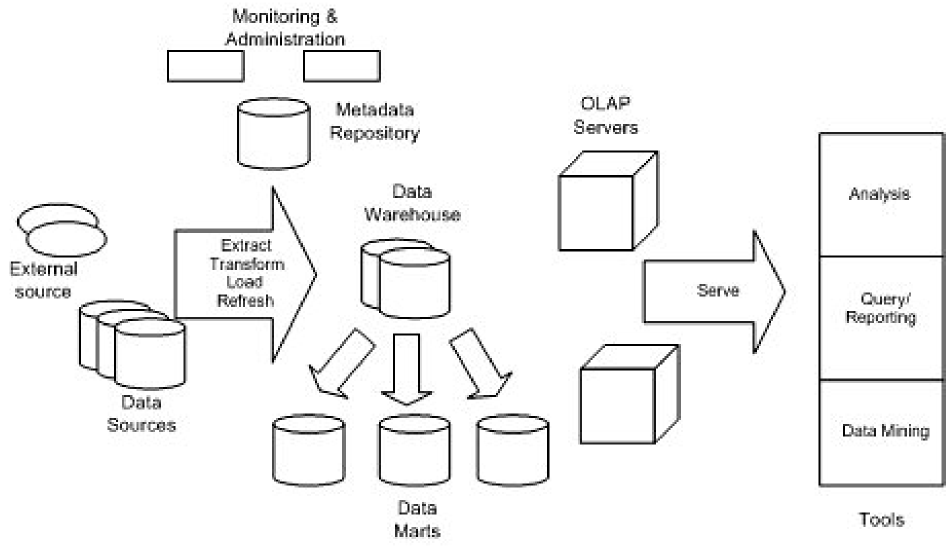


Fig. 8. Data Warehousing Architecture (Chaudhuri and Dayal, 1997).

Applications such as the balanced scorecard (Kaplan and Norton, 1996), together with corresponding dashboard technologies, enable end user organizations to present critical key performance indicators and success factors (CSF) that each business segment in the organization requires. The balanced scorecard idea is several years old, and analytical application solutions are finally deploying the collection and display of these kinds of applications on broader basis (Gray, 1999). According to Abatangelo (2001), traditional business intelligence or decision support applications have to integrate to three categories of data: operational data, business meta data, and external data. The use of internal data combined with external information (such as demographics) could provide an end user organization with a profound understanding of the data provided internally.

The question that an analytical application software vendor could potentially face is: how to support several different vertical market segments without having to have several different versions of the same software package. This type of challenge can be translated to a need to be able to build a common software application frame that can be the foundation for derivative products within a product family. Each vertical market segment has different business analytics with corresponding meta data, and the challenge for a software vendor is to be able to maintain these analytical solutions: to be able to update them with new versions without breaking existing solutions. The “glue” between these decision support components - as defined by Sprague (1980) - is an effective and centralized meta data repository (Hero, 2001; Soschin, 2001). Soschin (ibid) describes meta data as “the information that describes each component of an enterprise information system.” The traditional definition of meta data is known to be “data about data,” but this definition is according to Soschin too limited and does not portray the importance of meta data in implementation and management of information systems. Meta data is also important for end user analysts, as each measurement (key performance indicator) and originating data element has a universal and standardized description. According to Soschin (ibid), meta data management within an organization should be standardized, and this process should reflect data modeling, hardware and software configuration, data consistency, data conventions, external standards, modeling standards, and exchange standards.

Our emphasis in this study is to analyze development of prepackaged analytical applications. Our aim is to research data warehouse architectures and technologies, as a data warehouse is the underlying technology infrastructure for storage of information for the analytical application such as key business metrics and business processes. A data warehouse is just one functional element in an overall analytical application needed when building software for the analytical application software market. Other components, such as OLAP software with corresponding dashboard technologies, are needed to be able to display the contents of the database and provide much needed analytical information for end users using an Executive Information Systems user interface. We will first describe data warehousing and data mart technologies, then work our way to associated OLAP and EIS technologies as portrayed in Figure 8.

3.2.1 Data Warehouse Technologies

The main idea behind data warehousing was and still is the need to separate the operational data environment from the decision support environment, as operational applications were never planned or intended to be used as basis for decision support. Dr. E.F Codd introduced a model in the 1970s in which traditional entity relationship modeling (E-R modeling) breaks entities (or tables in a physical database) into “small sets of distinct attributes (or columns), joining them on unique identifiers (or keys)” (Van Dyk, 2002). This process of normalization is not well suited for querying and reporting, as reports and queries must include multiple table joins and this is very time consuming: reporting will become extremely slow when the database tables are large. E-R modeling is widely used amongst software vendors to provide the foundation for database models with intensive transaction processing requirements and a need for continuous record insertions, updates, or retrievals. According to Van Dyk (ibid), Inmon amended the model of highly normalized databases by adding subject area report tables that made reporting and querying more intuitive and effective. Van Dyk (ibid) argues that this approach “allowed for little or no reporting performance improvements since much of the design emphasis remained on the store of data in the normalized model.” The original definition of a data warehouse from Inmon (1993) relates back to the ability to store a historical record of data for the all transactions. This typically resulted in extremely large data warehouse installations that were very common in early 90s. These large enterprise wide data warehouse implementations could take up to two years to implement, requiring tens of millions of dollars to accomplish. This resulted in several failures, as the organizations had been changed several times during project implementation, resulting in an obsolete and inconsistent data model.

To avoid the bottlenecks and performance problems, Kimball (1996) defined a new model that was based on the concept of not having multiple normalized versions of the descriptive table but simply a single intuitive denormalized version of the tables – a star schema. This approach – dimensional modeling – gives end user analysts a simple database model that is based on business needs and different subject areas. According to Van Dyk (2002), this dimensional model is not without problems, mostly due to denormalization of the database tables and problems associated with this process. A dimensional star schema consists of a fact table with surrounding dimension tables. A row in a fact table corresponds to a measurement. Fact tables express the many-to-many relationships between dimensions in dimensional models (Kimball and Ross, 2002). Dimension tables are entry points into the fact table, providing "slice and dice" capability to business analysts.

Numerous books have been written about different data warehousing topics (Baquin and Edelstein, 1997; Humphries et al, 1999; Agosta, 2000; Kelly, 1996, 1997: Singh, 1998, 1999; Kimball, 1996; Kimball et al, 1998; Giovinazzo, 2000; Jarke et al, 2000; Gray and Watson, 1998; Moeller, 2001; and Debevoise, 1999). The common theme for all these books is more or less to provide advice for end user organizations in the data warehouse implementation process. A typical architectural model for data warehousing as defined by data warehousing literature is a process model in which data is extracted, transformed and loaded into a data warehouse database and furthermore distributed to end users for data analysis. This process-oriented architectural model describes the typical phases that a data warehouse implementation must undergo and it also portrays the functional elements to be included in a data warehouse implementation (see Fig. 9).

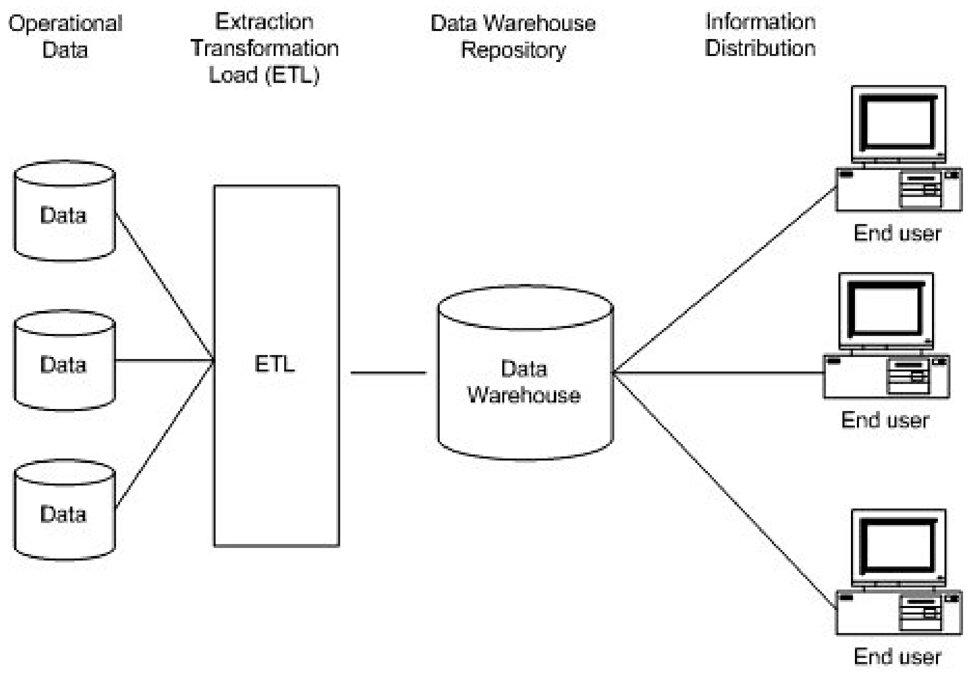


Fig. 9. Traditional Data Warehouse Architecture.

Some authors (Agosta 2000; Baumöl et al, 2000; Moeller, 2001) add an iterative nature to the data warehousing process, where the overall solution is build incrementally and some phases undergo several iterations. Agosta (2000) describes a data warehousing process as synonymous with the information supply chain, where "elementary operational events – dollar sales, unit item, quantity sold, and inventory on hand – get molded into business events." Jarke (2000) brings reengineering of legacy systems to the data-warehousing domain. According to the authors, data warehousing can be viewed as a mean of reengineering legacy systems, as a data warehouse brings heterogeneous data together "under a common conceptual and technical umbrella." Therefore, they argue that a data warehouse is "a buffer between transaction processing and analytic processing," as operational databases are not well suited for decision support.

Moss and Adelman (2000) suggest prototyping using a rapid application development approach (RAD) as a preferred development approach to data warehousing. The authors suggest that a prototype should not be a throw-away proof of concept, but "a real system without having to be completely rewritten" (Moss and Adelman, 2000). Several articles have been written evaluating different data warehousing methodologies (Thomann and Wells, 1998; 1999; 2000), but all of these methodologies analyze data warehouse implementation from an end user organization perspective and do not therefore provide any guidance to software vendors in building software solutions other than providing an understanding of how these organizations expect the implementation process to be managed. These methodologies are not specifically defined for implementation around prepackaged analytical applications: they are more or less defined to manage a traditional business intelligence or data warehouse implementation process.

According Moeller (2001), development of a data warehouse requires two separate architectures: a technical architecture and an information architecture. The technical architecture is important from the software development perspective, as it lays the foundation of all required technologies for the solution, including relational database technology, distributed computing technology, and numerous other technologies. Information architecture describes the behavior of the included data elements, the data warehouse content, and how data interacts with business rules that are built into the decision support application. According to Moeller, business objects within the data warehouse architecture and defined by the information architecture use the services of the components that are defined by the technical architecture. These two separate architectures bring software vendors information on two intersecting architectures. Both of them will require profound planning by the software vendor.

To facilitate and make enterprise data warehouse development more easy and controlled, Inmon et al. (2001) have also introduced the concept of Corporate Information Factory (CIF) which includes Operation Data Stores (ODS) as a key element in the overall architecture. An operational data store is an intermediate layer that is in some cases built between the operational sources and the data warehouse environment (Inmon, 1999). According to Jarke (2000), an ODS contains "subject-oriented, collectively integrated, volatile, current valued, and detailed data." The main difference between a data warehouse environment and an operational data store is that the latter is subject to change much more frequently. An article by Van Dyk (2002) includes discussion of maintaining a unique normalized staging area/ operational data store as a single source for the data warehouse environment. This eliminates the need to maintain several ETL processes within the overall solution. Werner et al. (2002) explain that a Corporate Information Factory can play a significant role in collecting Web data, enabling decision makers to analyze click-stream data and therefore have a better understanding of their e-business activities. The authors found that an additional ETL layer between the data warehouse and the data mart reduced significantly the amount of Web data for analysis.

3.2.2 Data Mart Technologies

Another approach to building analytical application solutions is to use data mart model as basis for solution development. Kimball et al. (1998) describe a data mart as a logical subset of a data warehouse. A data mart is typically restricted to a single business process or to a group of related business processes. Watson and Gray (1998) conclude in their article of new developments in data warehousing through 1998 that the industry has accepted the data mart approach for specific functional domains such as finance, insurance, etc. Watson and Gray (ibid) state that many organizations have started to build distributed data marts rather than building an enterprise-wide data warehouse. The article suggests that organizations should avoid independent data marts, as they could lead to long-term data warehouse problems. Eckerson (1998b) argues that one of the reasons for data warehousing "crossing the chasm" was the popular and risk free data mart approach, which led companies to successful implementations. Another significant contribution is a bottom-up methodology presented by strong industry professionals such as Dr. Kimball.

Griffin (1998) discusses whether organizations need a single, enterprise wide data warehouse or whether departmental data marts are sufficient as basis for decision support. According to the author, smaller data marts are easier to build, but at the same time these can cause problems for the IT department from a support and cost perspective, as each data mart can be based on a different database technology, a different operating system environment, and a variety of different OLAP reporting tools. The author suggests that the key to building separate data marts is to build a scalable technical architecture that can be a basis for enterprise-wide warehouse development. The author points out that the choice of whether to build a data mart or an enterprise-wide data warehouse is often political. Cranford (1998) defends the data mart approach in building financial data marts against enterprise-wide data warehouse development. According to the author, these data marts "require smaller amounts of source data, fewer data elements to define, and fewer business rules to develop." Therefore, these data marts are much quicker to develop and will not cost as much as typical enterprise-wide solutions.

Kimball et al. (1998) conclude that a data mart has to be "represented by a dimensional model within a single data warehouse and must consist of conformed dimensions and facts." According to Kimball (ibid), a data warehouse is "nothing more than the union of all the constituent data marts." This is a major deviation from Inmon's original definition of a data warehouse. These two different beliefs have formed two different data warehousing "camps." One of these beliefs emphasizes that the only way to build data warehouses is by using an enterprise data warehouse approach, while Kimball and his followers believe that a data warehouse can be built using subject-oriented data marts using conformed dimensions (common dimensions across different data marts), these data marts being linked together to an enterprise data warehouse.

Giovinazzo (2000) warns that a common misconception is that a data mart is a small data warehouse. The author explains that the difference in these is the scope, as a data mart typically focuses on an individual subject area within an organization, while a data warehouse represents the entire organization. Van Dyk (2002) explains that the data model and design method introduced by Kimball "allowed for intelligently and selectively denormalizing the data model while still retaining the measurements inherent in Inmon's approach." The main reason for this design method was to eliminate the performance bottlenecks that were inherent from highly normalized data models with corresponding table joins in reporting. According to Van Dyk (ibid), in the star schema data model introduced by Kimball, there where "no longer multiple normalized versions of the descriptive table but simply a single intuitive denormalized version of the tables."

Data marts can also be divided into different categories based on the aims of the solution. Moeller (2001) divides them into dependent data marts, independent data marts, integrated data marts, and point solution data marts. Dependent data marts typically coexist with centralized data warehouse. This model is a tiered solution, and the data marts are basically subsets of the enterprise data warehouse. Independent data marts are architected with departmental needs as basis. Pure data warehousing theorists dislike such kind of development, as it can lead to data anarchy in the organization. Moeller explains that integrated data marts can be compared to a shopping mall, where the data marts have a "unifying theme and well defined standards rules for participants." Therefore, every data mart must comply with the rules before it is accepted to the mall. Point solution data marts are needed when "each data mart is built to specific needs of a particular group without adhering to any integrative architecture, standard, or other conformance with any other existing data mart" (Moeller, ibid).

According to Imhoff (1999), size is not the deciding factor in determining a data mart. A common misconception is that data marts must be separated from the data warehouse environment. The author concludes that the data warehouse environment can include a set of star schemas that fulfill the business purpose and can be regarded as a data mart. A data mart need not be based on relational technology: it can as well be based on flat file technology, multidimensional OLAP technology, or any other technology that support the chosen business purpose. From a software development perspective, both a detail level data warehouse and a data mart implementation are similar, as the technology used is typically based on relational technology.

3.2.3 OLAP and EIS Technologies

An analytical application software solution must include tools for end user analysts to analyze information stored in either data mart or data warehouse database structures. Some software development organizations have concentrated their efforts on providing server software solutions for analytical application solutions, while some software vendors concentrate on building end user query and reporting tools that enable end users to explore analytical information in the solution databases. Balanced Scorecard solutions with respective Key Performance Indicators (KPIs) and Critical Success Factors (CSFs) can be part of an overall analytical application software solution. These measurements have to be both stored in a central repository and displayed to an end user using some kind of display technology, such as a personal computer or cellular phone. Regardless of the software solution emphasis of the software vendor, each software application has to provide a value proposition that provides the required return on investment numbers for the end user organization.

According to the architectural model in Figure 8, a data warehouse repository will be the storage mechanism and repository for business critical information. This information is moved into different analytical applications that are driven by technologies such as multidimensional On-Line Analytical Processing engines (OLAP) and operational reports that will be the basis for decision making. The purpose of these solutions is to provide the presentation layer for business users. Depending on the decision support tools category, some presentation tools are more passive, while typical OLAP desktop tools (DOLAP) enable multi-dimensional analysis of data residing in the decision support server. Orfali et al. (1999) conclude that EIS and DSS applications "provide the human interface to data warehouses." This statement is the key of understanding the role of data warehousing. A data warehouse is the repository for information collection and storage and does not as such provide any tools for information querying or reporting. These additional Executive Information Systems and decision support tools (DSS) tools provide flexible analysis capabilities for end-user organizations.

From a technical perspective, there are several different implementation technologies behind OLAP solutions. The most traditional implementation is a multi-dimensional database that has been tuned for Multidimensional On-Line Analytical Processing (MOLAP), while Relational On-Line Analytical Processing (ROLAP) software products are based on relational technology. These relational OLAP products typically implement a star-schema database design pattern that resembles a star with a fact table in the center of the star and a dimension table for each dimension that can be found in the fact table. If a software vendor decides to implement the underlying storage database model based on a star schema, the software vendor must build easy-to-use tools to view and analyze the underlying relational database structure. A star schema as such does not lend itself to any easier usage of information in the database, as end users must understand Structured Query Language (SQL) syntax to be able to utilize the contents of the database. It is fair to say that different desktop OLAP tools permit users to create multi-dimensional views on top of ordinary two-dimensional relational databases. A variation of these traditional OLAP tools includes other categories with slightly different technological foundations, such as Hybrid On-Line Analytical Processing (HOLAP), which provides a solution with both relational and multidimensional features.

There are several categories of OLAP tools, everything ranging from desktop OLAP tools to server-based proprietary multidimensional OLAP tools. The purpose of these tools is the same: to enable multidimensional analysis of data that resides in data warehouse or data mart structures. All of these different OLAP tool categories refer to a decision support database or engine in one way or the other, and each represents a different underlying technological architecture. Haisten (1996) identified 16 different tool categories that can be used in decision support analysis. Eckerson (1998a) has developed a model that portrays two different decision support domains: reporting and analysis. This model helps end user organizations and software vendors to categorize and understand the breadth of different analysis tools that can be used in conjunction with decision support software solutions. According to Eckerson (ibid), there is a disconnect between software vendors and end user organizations: software vendors oversell their software packages with functionality that end users are not going to use or which is too difficult to manage.

3.3 Analytical Application Architectural Models

The aim of this chapter is to explore different alternative architectural models that a software vendor can select when implementing analytical application software solutions. The first sub-chapter discusses the main differences between a data warehouse and a data mart implementation approach. The second sub-chapter explains the most common data warehousing architectural models. The final sub-chapter will explain the pros and cons of each architectural model and how these could impact development of an analytical application software solution.

3.3.1 Data Warehouse vs. Data Mart Implementation

One of the main new themes in the data warehousing domain is the argument whether data warehousing should be implemented from an enterprise view or by implementing subject specific data marts one at a time to satisfy the needs of corporate reporting (Tipnis and Epifano, 1998; Griffin, 1998; Russell, 2000; Inmon, 1998; Hackney, 1998; Tanrikorur, 1998; White, 2000b). The discussion of these two models will bring additional topics to be decided by a data warehouse software vendor. If both centralized data warehousing and subject-specific data marts are implemented, software vendors have to decide on the data distribution model and IT infrastructure strategies. Several different architectures can be applied; including centralized, federated, and distributed architectures. These architectural models will also influence software application frame development, as the selected architecture can impact software application frame technology sub-strategy.

Of these two different approaches to data warehousing, one emphasizes an enterprise view of building a data warehouse and the other is based on subject specific departmental data marts. Griffin (1998) points out that small-scale data marts are easier to implement than the enterprise-wide data warehouse, and the payback is typically short. Griffin emphasizes that if a data mart approach is taken, the organization must centralize meta data management to smooth the transition from the data marts to data warehouse. Tanrikorur (1998) adds that corporate wide data warehouses require an extensive commitment of the company and its management. Corporate data warehouses typically have high volumes of multiple data sources, and sorting data out becomes a major undertaking. Singh (1999) states that the quickest way to implement a data warehouse is to use a bottom-up approach, where the user organization builds departmental data marts with forethought of an overall corporate information strategy.

Several authors discuss the pros and cons of both bottom-up and top-down data warehouse implementation models. Inmon (1998) argues strongly that bottom-up data warehouse development, implementing separate data marts, will lead into data redundancy and simply can not be implemented. His view is that each organization has to build an enterprise wide data warehouse without using departmental data marts. Departmental data marts are typically seen as an easy way to implement a decision support system for departmental use, but Inmon's argument is that this approach will lead to duplication of data integration efforts and, in the worst cases, to different operational system environments. According to Inmon (ibid), independent data marts could lead to departmental selection of hardware and software, increasing the overall cost of using the data mart implementation approach.

Hackney (1998) lists the pros and cons of using a bottom-up or top-down strategy in building enterprise wide data warehouses. His main arguments for a top-down approach are that the data marts are architected from the enterprise wide data warehouse and therefore the data marts "inherit" the architecture from the parent data warehouse. This approach will have an enterprise view of the data and one single and central meta data repository, making the maintenance of the system less complex. The cons of having an enterprise view according to Hackney (ibid) are several, one of them being the long implementation time. When an enterprise data warehouse is built, the subject specific data marts will typically be in production after 15 months, which also increases the risks of the implementation, as the organization might have changed during the implementation. Hackney (ibid) states that one of the greatest dangers in the data warehousing world is the creation of non-architected data marts. He argues that these data marts become LegaMarts. These LegaMarts face challenges in later integration work with other data marts or the enterprise data warehouse.

Tanrikorur (1998) suggest a hybrid approach in her article "Enterprise DSS Architecture – A Hybrid Approach." Tanrikorur states that the implementation of an enterprise wide data warehouse does not become a reality without commitment, and that to implement an enterprise wide data warehouse is neither easy nor cheap. Tanrikorur (ibid) suggests that organizations combine the top-down and bottom-up approaches in what she calls a "hybrid approach." Gill and Rao (1996) suggested this approach back in 1996 when they called for a "combination approach." Their hybrid approach emphasizes that the data marts should be designed with the enterprise model in mind, but it is not necessary to build the enterprise wide data model at beginning. The only requirement is to "identify all major problem areas that will eventually be included in the total enterprise solution" (Tanrikorur, ibid). The key people in the organization should agree upon the enterprise data model before the subject-specific data marts are built. This hybrid approach favors iterative development of a data warehouse solution, which also supports Inmon's original idea of iterative data warehouse development (Inmon, 1996).

White (2000b) also suggests a hybrid model – The Federated Data Warehouse – which resembles the model introduced by Tanrikorur (1998). White claims that there is a trend toward the use of dependent data marts, and that these data marts are typically also the basis for turnkey analytic application packages. According to this model, the "design and development of independent data marts must be managed and must be based on a shared business model of an organization's business intelligence requirements" (White, 2000b). This architectural model includes a common data staging area that includes all the needed elements for independent data marts. The shared business model ensures consistency in the use of data names and business definitions and allows a centralized storage of all key business measures and processes that can be shared across different data marts in the organization. Adamson and Venerable (1998) also follow and suggest a hybrid approach for building a data warehouse. The pros of a "bottom-up" strategy are the opposite: implementation of data marts is fast, typically taking between six and nine months for a focused data mart. Return on investment in a data mart is quick, as the customer sees the business value much more quickly.

Regardless of the selected architectural model, a software vendor has to adjust the selected model with consideration of several other factors, such as the end user organization IT infrastructure, available application development tools, and the selected database solution. The bottom line for an analytical application software vendor is to select its architectural model according to the requirements of solution functionality (product architecture with required functionality), and the selected market segment (and this market segment could have its own restrictions with respect to the operating system environment and other IT infrastructure elements that the solution must support). Without considering these additional boundaries, whether technical or functional, an analytical application software vendor could dramatically restrict its future market segmentation strategies.

3.3.2 Different Data Warehouse Architectural Models

Data warehousing literature suggests several different architectural models that end user organizations can select when building their decision support solutions. This literature is specifically written and analyzed from the end user organization perspective and does not therefore reflect the requirements that analytical application software vendors might have when developing packaged analytical application solutions. Our analysis of different architectural models within data warehousing should consider not only the software development perspective, but also what architectural model provides the best foundation for development of analytical applications for different vertical markets. We assume that there will not be a single "truth" when selecting architectures, as each selected solution could be a combination of different strategic elements.

Regardless of whether a software vendor selects an enterprise-wide or a data mart implementation approach, the overall aim is the same: to provide analytical application information to end users. The three main data warehouse architectural models are centralized, federated and multi-tiered/distributed (Jarke et al., 2000). Another architectural model that is based on a hub-and-spoke architecture is positioned as an alternative in distributed environments with the need to incorporate local meta data and business metrics (Hero, 2001).

The centralized approach includes one data warehouse (or data mart). All information is stored in a centralized database. In a federated architectural model, data is logically consolidated but stored in separate physical databases or even at different physical sites. A tiered architecture includes a centralized data warehouse with one or several data marts on different tiers that include summary of data from previous tiers.

Each architectural model has its strengths and weaknesses, and these must be evaluated together with information technology (IT) infrastructure constraints and the selected software architectural model that the software vendor sets according to its technology strategy. Noaman and Barker (1997) assert that each organization must decide whether a data warehouse should be distributed or reside on an integrated centralized or single machine. Russell (2000) discusses building an adaptable decision support architecture wherein each organization must balance between the centralized model and distributed and propagated architectures, normalized and denormalized table architectures, and database architectures (including parallel technology and database engine architecture).

A centralized data warehouse model is traditionally depicted with an information flow model in which data sources are portrayed leftmost, the data warehouse database is in the center, and data distribution and reporting tools are rightmost (Inmon, 2002; Watson and Haley, 1997; Tipnis and Epifano, 1998). This view of a data warehouse, where data flows from operational systems into an integrated data warehouse from which users can create queries or reports for decision support, is described by several different data warehousing sources (Watson and Haley, 1997; Inmon, 1992, 1996, 1997; Noaman and Barker, 1997; Singh, 1998; Orfali et al., 1999; Moeller, 2001).

The centralized architecture is similar to that of the traditional architecture that Bill Inmon describes (1992). This model suggests that there is only one enterprise data warehouse with included subject-specific areas. According to Inmon (2002), it makes sense to maintain a single centralized data warehouse if the environment supports this approach. In some cases a distributed data warehouse environment makes more sense, such as an environment where most of the businesses are distributed geographically. Another typical case where data distribution makes sense is if the amount of data becomes very large. In that case, it makes sense to distribute the data warehouse technologically to several processors while still keeping a logical view of the overall data warehouse. Some organizations have not built their data warehouse environment in a coherent manner, and such organizations might have several independent data warehouses in different geographical locations.

According to Özsu and Valduriez (1996) a fully replicated data warehouse is similar to a centralized data warehouse architecture. In this model, the centralized master data warehouse database is fully replicated to each remote site (local replica data warehouse databases).

Several authors (White, 2000b; Hackney, 2000a; Hackney, 2000b; Jarke et al. 2000) suggest a federated data warehouse implementation model. Jarke et al. (2000) portray a federated data warehouse architecture where data is logically consolidated, but stored physically in separate databases. In this approach, local data marts store only the information that is relevant to the department or subject, and the logical data warehouse is virtual, while in the tiered architecture the logical data warehouse is physical and each tier can have its own data marts that contain summarized information and do not include detailed data as in the federated approach. According to Hackney (2000b), federated data warehouse architecture is "an overall system architecture that accommodates multiple data warehouse/data mart systems, operational data stores, amorphous reporting systems, analytical applications etc." The federated data warehouse architecture adds a technical detail in the design of a data warehouse, namely a data staging area that is the main source of information for the data warehouse integration.

White (2000b) describes this federated data warehouse approach in an architectural model which combines two concepts, namely data flow from operational data stores to the data staging area and the development process from initial business requirements analysis to implementation of the data warehouse models. White includes the implementation of the data models for both the data staging area and the data warehouses. He also includes data reengineering tools and data profiling tools in the model. This architectural model differs from traditional data warehouse models by adding the data staging phase in the implementation of the data warehouse. The main idea behind White's federated data warehouse model is to enable organizations to have a centralized data warehouse with independent data marts. The consistency of these data marts is controlled via a common business area model and data staging area for the entire enterprise (see Fig. 10).

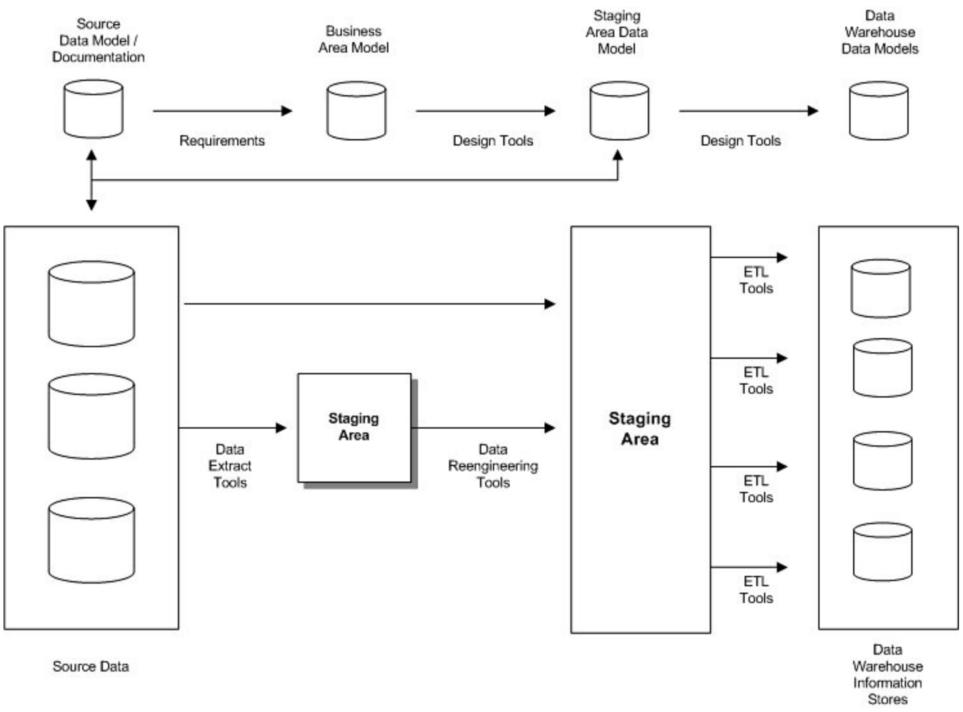


Fig. 10. Federated Data Warehouse Architecture (White, 2000b).

According to Noaman and Barker (1997), the disadvantages of a centralized data warehouse architecture can be overcome by implementing a distributed data warehouse architecture that "mirrors more accurately the physical distribution of the actual organization." This approach requires that local data warehouses be implemented by extracting information from local operational systems, resulting in a local data warehouse in each branch. Each local data warehouse is a subset of an overall view of the corporate-wide data warehouse. Noaman and Barker (1997) describe two main schools of thought relative to distributed data warehouses: those of Inmon (1993) and White (1995). Inmon's approach emphasizes the existence of both local and global data warehouses, where local data warehouses are fed by local operational systems and these data warehouses can be mutually exclusive. White's (ibid) approach, also known as "two-tier data warehouse," is a combination of a centralized data warehouse and decentralized data marts. These decentralized data marts are updated and fed from the centralized data warehouse environment.

Noaman and Barker (1997) suggest an alternative model for distributed data warehouses. In this model, decision makers are divided into two different categories; those that make decisions for the entire corporation and those that make decisions at local branches. The proposed architecture mirrors these two different categories with a Distributed Detailed Data

Warehouse (DDDW) and a Centralized Summarized Data Warehouse (CSDW). This model provides distributed transparency, where local decision makers use local data warehouses while decision makers for the entire organization can use these local data warehouses for detail analysis and also a summarized data warehouse that is centralized and gives users at headquarters an overall view of the business.

Another architectural model based on hub-and-spoke architecture with a centralized meta data repository is proposed by Hero (2001). According to Hero, the advantage of having hub-and-spoke architecture with a middleware platform enables applications to have better scalability and flexibility. Applications are integrated, and a hub-and-spoke design “connects source and target systems to a central hub.” The main idea behind this architectural model is that each data source represents a spoke and an “information broker” serves as the hub. A key element in this architecture is to have a strong centralized meta data repository that is, according to Hero, “a central storage area for source and target meta data, source and target interface mappings, business rules, transformation rules, data validation rules, scheduling, and other information about the data exchange process.” These spokes can be different for midrange systems, mainframes, Windows NT based servers, or even proprietary file systems, according to Hero (ibid).

3.3.3 Pros and Cons of Different Data Warehouse Architectural Models

There are two considerations when deciding upon an architectural model for building a decision support solution. First of all, the software vendor has to decide whether the solution is going to be implemented based on a data mart or a data warehouse implementation approach. Second, the software vendor has to select an architectural model for the solution. According to our previous chapter, the main models are centralized, federated, or multi-tiered/distributed or a combination of these (Russell, 2000).

Several authors list pros and cons of the centralized data warehouse model (Noaman and Barker, 1997; Russell, 2000; Moeller, 2001). A centralized model offers a high level of security and ease of management. A centralized approach is convenient for organizations whose operational applications are already centralized. According to Noaman and Barker (1997), a centralized model suits organizations with a mainframe strategy: all OLAP tools can run against a centralized data warehouse with centralized processing from a single location. But the disadvantages of a centralized model are numerous, such as lost performance compared with distributed or multi-tiered environments, where some of the processing can be decentralized. Other disadvantages are expensive: expandability of the data warehouse, and reliability, as a centralized data warehouse is the single point of failure. The cost of implementation and vendor dependency can be a disadvantage of building a centralized data warehouse.

A traditional centralized model would lead to what Hero calls "Point-to-Point Chaos," where far too many points of data exchange exist with the IT infrastructure. This model could lead to repetitive ETL processes across the organization. Changes in either the data mappings or business rules would lead to chaos, and would be extremely hard to maintain. A fully distributed model with several architectural tiers requires selection of distributed computing technology and will therefore require understanding of complex programming issues such as load balancing, message marshaling, and several others typical for software engineering in a distributed environment.

The federated approach is similar to the centralized concept, but it adds a data staging area or operational data store that will reduce the volatility between the data source and data warehouse database. This architectural model also includes a centralized meta data repository and a business area model (White, 2000b) that will be available for all dependent or independent data marts that are integrated together to achieve an overall view of the enterprise. According to White (2000b), the federated approach will accept independent extraction, transformation, and load processes in a manner similar to the hub-and-spoke model proposed by Hero. The main concept behind a centralized meta data database is to keep all meta data with corresponding ETL rules (as well as other rules) centralized, even the rules for ETL process that are executed in remote locations. The main advantage that Hero's (2001) hub-and-spoke and White's (2000b) federated data warehouse models provide to software development organizations is an easier model for creating analytical solutions, as all meta data with respective business rules, transformation rules, etc. is centrally stored, and this helps the maintenance and upgrades of new releases to end user organizations. Hackney (2000a) concludes that a federated data warehouse model shares key metrics, measures, and dimensions across the entire range of business intelligence systems in an organization. In similar manner, Hackney (2000b) concludes that if "the Internet is a network of networks, a federated data warehouse architecture is an architecture of architectures."

The distributed/multi-tiered architectural model requires software vendors to select a distributed computing model and decide how to manage the distribution of both data and meta data. The distributed data warehouse model combined with a layered approach can be analyzed from two different perspectives. First of all, distribution does not necessarily mean multi-tier, as some IT infrastructure environments distribute information to remote locations and use this distributed information as if it were local. The other alternative is to have a true multi-tier architecture, where information flows between different architectural layers and this information flow is managed by distributed components such as Distributed Component Object Model (DCOM) or Common Object Request Broker Architecture (CORBA). In distributed environments where information is replicated from one location to the other, some application logic could be achieved by using the native functionality of the selected database management system.

Russell (2000) asserts that "decision alternatives for the distribution and propagation of data can be viewed across a continuum." On one end, organizations can build highly distributed architectures with independent data marts, and on the other end, organizations can build centralized enterprise data warehouse architectures with low propagation of data. A decision that is in between these two continuums is the hub and spoke approach with high propagation of data. According to Russell, there are several advantages to building distributed environments, such as performance improvement and less complex and smaller data models. The hub and spoke approach has disadvantages, including a higher cost of propagation, increased time to build compared with a centralized or data mart approach, and distribution of the business analytics calculations needed for cross subject area analysis. This can lead to a credibility crisis, as these spoke data marts may yield different results when compared with results from the hub environment. In the analysis of these continuums, Russell concludes that "once an architectural decision is made, the continuums become unidirectional and future architectural decisions can be made only one direction – outward." This means in reality that if an organization decides to take a distributed data warehouse approach, the organization cannot move back to a centralized architecture. Russell (ibid) suggests that organizations build their data warehouse foundation architecture with a centralized data warehouse approach. This architecture can be the basis for subject specific data mart analysis.

3.3.4 Conclusions – Findings of Architectural Models

The architectural models identified in the data warehousing literature are portrayed at a very high abstraction level – such as architectural patterns or designs - and do not necessarily provide a clear view of all the functional components that are required within an analytical application solution. These architectural models are geared more toward demonstrating information flow within the solution than showing how each functional component interacts with other functional components. From the software development perspective, the architectural models that can be found in the data warehousing literature do not provide enough detail level information to be useful in software development. We argue that there is a gap in information between the software vendor's perspectives. Each vendor has to explore these architectural models and build more detailed plans that consider both the complexities and the interactions of functional components.

A traditional centralized data warehouse model is known to have one centralized enterprise repository, while a multi-tiered/distributed architectural model can potentially have data on several architectural tiers. This increases the complexity of software development, as the solution has to provide tight information integrity across each architectural layer. From the software development perspective, the traditional centralized data warehouse model is arguably the easiest architectural model but also the most inefficient environment in end user organizations with heterogeneous data sources and hardware and software environments. The multi-tiered/

distributed architectural model provides end users with local data mart repositories. If the hub-and-spoke architectural model is deployed, the software vendor will be challenged to keep each spoke updated with current business analytics and metrics as described by Hero (2001). This model could potentially lead to integrity problems, as each spoke can have a different set of business rules and measures, localized based on the requirements of the remote organization (that portrays the spoke). In a federated data warehouse architectural model, database repositories can be physically distributed in the organization, but the end user is provided with a "virtual view" of the enterprise data.

Analytical application software vendors have to decide on an implementation approach for their analytical applications and selection of an architectural model for the solution. We have discussed the pros and cons of each implementation approach, and we believe that a hybrid approach (such as the federated data warehouse environment) is more efficient for end user organizations and for the software vendor. The hybrid model brings the advantages of having a centralized database repository and independent data marts using a shared business model for all of the end user organization's analytics requirements. We also believe that a common staging area is beneficial for an analytical application solution as, it shields the solution from underlying operational system database volatility. The federated data warehouse environment supports the concept of common meta data and business analytics repositories that are key elements in building analytical application solutions for different vertical market segments.

3.4 Analytical Application Software Solutions

The move from custom-made tailored analytical applications to prepackaged applications is gaining foothold amongst end user organizations (Morris, 1998; Eckerson, 2002). The aim of prepackaging is to provide to end user organizations predefined analytical solutions with predefined business metrics and business processes that are typical for the selected vertical market segment. Gleason (1998) divides business solutions into two categories: vertical solutions and horizontal solutions. Vertical solutions are defined for specific industries such as insurance, retail, distribution, and banking, while horizontal solutions focus on corporate functions across different departments, business units, or even industries. From a technical implementation perspective, data warehouse components will be needed to provide the ETL functionality to collect information from different operational applications. We see data warehousing with its included decision support applications (such as OLAP) as elementary components when building analytical prepackaged solutions. Morris (1998) concludes that existing data warehouse solutions must "incorporate data from both financial and non-financial information sources that are required by new analytic methodologies, such as the balanced scorecard."

3.4.1 Definition of an Analytical Application

The history of analytical solution development dates back into the early 1990s, when E. F. Codd presented the original description of analytical applications in a ComputerWorld article in September 1993. Several other authors (Morris, 1998; Gleason, 1998; McGuff and Kador, 1999; Surgan, 2000) also raised the question and discussed whether organizations should build or buy analytical applications and how an analytical application should be positioned within the decision support market.

One of the most influential articles within the field of analytical solution development was released by International Data Corporation (IDC, 1997). This article gave the first comprehensive definition of analytical solution development, according to Morris (1998). Analytical solutions represent a convergence of several major forces in the software market (Morris, ibid). First of all, analytic solutions provide and extend the existing business process methodologies with new elements such as the balanced scorecard, which was originally defined by Kaplan and Norton (1996). Secondly, analytical solutions drive end user organizations to buy solutions instead of trying to build them from scratch, which is typically a lengthy process and prone to failure. Morris (1998) emphasizes that this approach also supports the notion of traditional decision support definition, where these solutions are defined and designed to support decision support business requirements and not transactional systems and reporting that is typically driven from these systems. Morris (ibid) argues that analytical applications are extensions of data warehousing, where end user organizations are looking for new ways of analyzing information, such as balanced scorecard and data mining applications. It is fair to conclude that analytic applications require a data warehousing or data mart implementation model. Our aim is to explore how these implementation models can be used as bases for analytical application solution development from the software vendor's perspective.

According to Morris (ibid), analytical applications have to support three main characteristics. First of all, packaged analytical solutions have to support different business processes such as automation of "groups of tasks pertaining to the review and optimization of business operations." These business processes reflect the specified market segment. Associated business processes that must be "packaged" in the solution include "business rules, procedures, and techniques with an accompanying methodology that explains how these elements are to be used together to successfully complete a set of activities." Secondly, analytical solutions have to be able to operate separately from the organization's transactional systems while being dependent on the data provided by the transactional systems. Morris (ibid) concludes that analytical solutions have to support business analysts, sending data back to the analytical application. This type of closed-loop analysis will enable business analysts to react more quickly to the information that requires immediate action. The third and final definition of an analytical solution according to Morris is that it must include time-oriented and integrated data. This requirement is supported by data warehouse functional components such as ETL tools, which “extract, transform, and integrate data from multiple sources (both internal and/or external to the business).” The second element in the final requirement was to provide time-based analysis. This is a perfect match for any decision support application that provides time-series analysis and storage mechanisms for historical data.

Gleason (1998) discusses the relationship between tools and applications to the data warehouse. He sees generic tools that can be used to build many applications, while data warehouse applications, packaged applications, and business applications are built and designed for “a specific industry, business function, or type of company.” Gleason (ibid) defines a business application as “a value-added decision support system with front-end that has already figured out what answers business users need and how they want to access it.” This move from custom-built applications towards pre-built solutions helps end-user organizations to implement their analytical solutions without having to start from scratch. Gleason (ibid) also concludes that these packaged solutions do not always fulfill all of the requirements that an organization might have for the application, but the reduced risk and faster implementation will usually compensate for lost functionality. According to Gleason (ibid), a key advantage for end user organizations using vertical packaged solutions is that these packages typically already include best practices, key measures, standards and knowledge of the selected vertical industry. These solutions include data models that reflect typical business processes for the selected vertical industry. Finally, these packaged applications will bring a better return on investment (ROI) for user organizations, as they provide the opportunity for more rapid implementation, letting end user organizations concentrate on their own core businesses and competences and not to building these packaged vertical applications with a customized approach.

Eckerson (2002) categorizes analytical applications into different categories: packaged analytic applications, custom analytical applications, analytic development platforms, and business analytic tools (or analytic tools). According to Eckerson (ibid), an analytical application runs “using data warehousing technology, embeds analytic tools, and employs business process logic.” This statement reinforces our conclusion that an analytical application software solution is based on a selected data warehousing architectural model, and that this model is selected using different criteria in different software vendor organizations.

3.4.2 Analytical Application Functionality

The change from customized data warehouse applications toward packaged data warehouse applications is discussed by Gleason (1998) as being needed to enable provision of industry specific business solutions quickly, including most of the known business metrics and processes. According to Gleason (ibid), a business application is a “value-added decision support system with a front-end that has already figured out what answers business users need and how they want access it." Gleason (ibid) concludes that an analytical business application consists of two main components, a data model and the decision support application itself. The latter component typically includes data analysis and front-end access for a specific industry. The business application typically covers common needs for an industry and will therefore not cover all the needs for every company. According to Gleason (ibid), the data model is based on an industry-specific template that has the needed design (measures and facts) for the given industry. This template can be regarded as intellectual capital for the software vendor. Gleason (ibid) does not discuss what kinds of difficulties the implementation of these templates will bring to a decision support vendor or whether the software vendor should aim to separate the development of the software application for a given industry from development of these industry specific templates.

Gleason (ibid) proposes a high-level architecture model for packaged analytical applications with three distinctive architectural tiers, namely a presentation tier for presenting the information, a business logic tier for the business logic itself, and finally the data tier that embeds the persistent data that is collected from different operational applications (see Fig. 11).

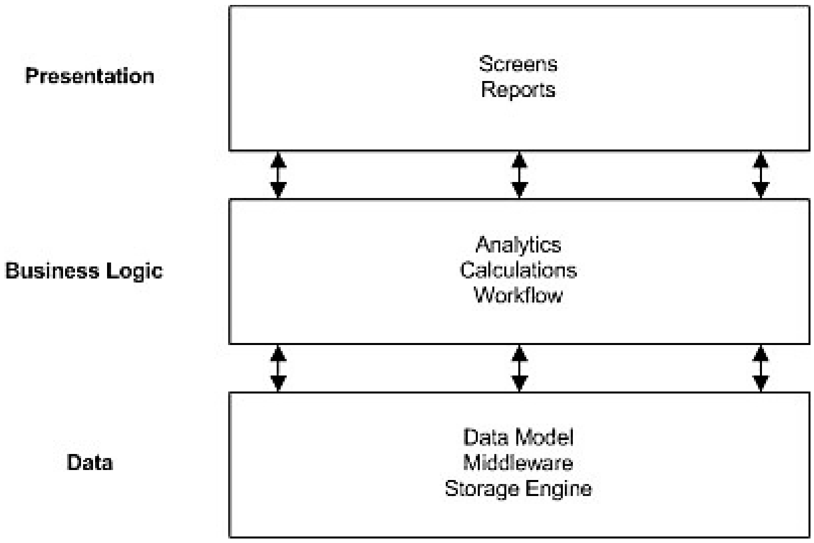


Fig. 11. Three-tiered Packaged Application Architecture (Gleason, 1998; Original source: Julie Hahnke, 1997).

The top tier, the presentation layer, typically consists of predefined reports for the industry, either in graphical or numerical format. These reports can be viewed either online or in printed format. The business logic layer can consist of industry specific or business specific analytics and calculation rules and sometimes even workflow capabilities. According to Gleason (ibid), the data tier typically has the greatest variation between different packaged applications, as some packages provide sophisticated data support. This layer consists of three main elements: the data model, the middleware, and the storage engine. The storage engine can be based either on relational or multidimensional technology. From a software development perspective, Gleason does not provide any discussion of the underlying technological selections that a decision support vendor has to take when building this three-tier architecture, nor does he discuss the granularity of the decision support components needed to build a decision support application frame.

White (1999) argues that organizations are seeking faster ways to deploy decision-processing applications, and that the marketplace is moving rapidly toward analytical applications, whereas data warehousing vendors are moving from technology driven, build-your-own solutions towards supplying business-driven, packaged applications. White divides packaged decision processing solutions into three flavors:

- Information templates are predefined and customizable applications for building a data warehouse or data mart to support a specific business function or purpose (sales analysis, for example).
- Reporting/analysis templates supply predefined and customizable queries and analyses for a specific decision support tool.
- Analytical application packages offer a complete business solution for analyzing and reporting on information in a specific business area.

White (ibid) distinguishes between three levels of packaged solutions. The last category, with all needed key metrics for the selected market segment, will be the best basis for a decision support vendor from a software product platform perspective. White describes information templates as including three different inner templates that are typically a part of ETL components and the base for building a business specific database model for the decision support application. He defines the extract and transformation template as the predefined rules to extract information from selected operational applications and to transform it into the format of the analytical solutions, while his business area template defines the application itself, including dimensions, data types, and predefined reports that belong to the solution.

White (2000a) also discusses the differences between traditional data warehousing and analytical solution development. According to White, data warehousing projects have been able to help many organizations to reduce costs and increase revenues. With analytical applications, organizations can potentially achieve several aims. First of all, these solutions enable organizations to reduce the cost of implementation of the decision support application. Secondly, they will ease the complexity of decision support tools and data access from operational sources. Finally, they improve the quality of analytical information. His argument is that traditional data warehousing installations with many disparate data sources lead to complex and costly implementations. Therefore, organizations are looking into off-the-shelf packaged solutions: solutions that provide most common functionality for a given vertical market segment or horizontal business function. White (ibid) also discusses the pros and cons of buying or building an analytical solution. One of the cons of buying a solution is the restriction that a packaged application might have if it does not fit into "shared decision-processing information architecture." Therefore, the marketplace will include a combination of both built and bought analytical solutions, and these solutions should be able to interact with each other. Organizations could be reluctant to buy pre-packaged analytical solutions, as they presume that their application requirements are unique and are difficult to integrate into an overall decision support architecture.

Surgan (2000) analyzed the data warehousing market and classified its evolution into three main phases: the custom phase, the component phase, and the packaged phase. The custom phase took place in the late 1980s and early 1990s, when early adopters began building enterprise data warehouses under the leadership of Bill Inmon. The second phase evolved in the mid-1990s, when organizations began looking for data warehouse solutions, and some organizations began developing data mart solutions to minimize the risk of not completing their decision support projects. This also resulted in a "methodology" war between some practitioners, whereby some claimed that the data mart approach leads to islands of disparate data marts, while an enterprise data warehouse leads to better results. Surgan (ibid) stated that, as we moved into year 2000, organizations were more ready to buy prepackaged solutions that are both easy to install and easy to support. Surgan (ibid) also stated that these prepackaged solutions must "encapsulate the experience and expertise gained in data warehouses that have already been implemented and operate successfully." According to Surgan, a packaged solution should include the following functional components:

- ETL
- Data warehouse design
- Data warehouse management
- Meta data repository
- Scheduler
- Integrated query tool for data analysis

The analysis of different architectural data warehouse models showed three major architectural styles that a software development organization can choose from when implementing analytical applications. First of all, the traditional centralized data warehouse model suggests that all functional frame components be centralized and accessed from a centralized location. The second approach is the federated architectural model, providing user organizations with a "virtual" view of enterprise information in the form of one or several data marts with the ability of cross subject-specific analysis between different data marts. The third alternative is to build a multi-tiered/distributed architectural model, where analytical application functionality could be spread across multiple tiers. This approach poses several software engineering challenges for software vendors. Our discussion of fully centralized vs. fully distributed vs. federated data warehouse model resulted in a conclusion that the architectural model should not restrict the functional requirements that end user organizations expect from these kinds of solutions. The requirements for distributed information and offline usage of analytical information will require additional functionality of analytical solutions, such as support for local meta data, together with corresponding calculation and business process logic that provides analysis results to the end user using an end user tool. The purpose of next chapter is to discuss the selection process of architectural model for an analytical application solution. This discussion includes consideration of a large common core that can be reused across different vertical and horizontal market segments.

3.4.3 Selection of an Architectural Model for an Analytical Application

Development of an analytical application using the product platform approach requires software vendors to evaluate the selected underlying architectural model by using criteria from product platform theory. Chapter 2 included a description of the software application frame (Sääksjärvi, 2002) which, according to Sääksjärvi, represents "a large common core" that is used across different products within a product family. This common core should be optimized to include functional components that are common to all the software products within a product family. Variations are implemented by frame extensions that provide needed functionality to each selected market segment. Within our software solution domain, this common core is defined as the "analytical application frame," common across any selected vertical market segment that the software vendor decides to address. This analytical application frame will be defined in more detail in Chapter 4, together with its corresponding characteristics.

To be able to evaluate the feasibility of each architectural model within analytical application frame development, we listed the pros and cons of each architectural model specifically from an analytical application frame development perspective. Schuff and Louis (2001) conclude that "information technology departments have historically cycled between centralized and decentralized application software distribution, although modular program design and enterprise management software may break the cycle." Schuff and Louis (ibid) also argue that even if the centralized architectural model could be proven best for IT management, a centralized scheme might not be appropriate to all organizations. This type of discussion is also prevalent in analytical application software, as some of the centralized functionality is required at remote locations which will require local meta data and a local business analytics repository. When the solution is centralized, the application software will reside centrally. In a decentralized scheme, application software typically resides on each of the client PCs or, alternatively, at remote server locations. This type of decision-making by IT organizations in end user organizations will reflect the selection of the domain-specific architectural model for software vendors as well. At this boundary, IS research and software engineering research will meet. The problem of changing from one scheme to another could become a major obstacle for any software vendor. This was seen clearly in the movement from the traditional client-server model to the thin-client model during the late 1990s and early 2000s.

Another viewpoint the software organizations must reflect on is discussed by Hasselbring (2000), who asserts that the information system architecture is composed of three layers on top of each other. The initial layer, technology architecture, defines the information and communication infrastructure. This corresponds to the selection of the operating system environment, the database management system, and other IT infrastructure elements for an analytical application software vendor. The second layer, application architecture, defines the actual implementation of the business concepts. In this layer, the "central goal is to provide the 'glue' between the application domain described in the business architecture." The final layer – business architecture – defines the organizational structure and the workflows for business rules and processes. This layer corresponds to the functionality and measurements to be found in the business analytics repository in an analytical application software solution.

Our definition of the analytical application frame seeks to find "an optimal architectural construct" that is common to all derivative products. This definition is measured against different IT architectural models that we have selected for review, namely the centralized, federated, and distributed/multi-tier architectural models. We consciously excluded the hub-and-spoke model introduced by Hero (2001) from our comparison table, as this model has its disadvantages, such as not being able to perform cross subject area analysis, specifically if most of the calculation rules exist in the spokes (data marts). Russell (2000) concludes that even if the hub were available for centralized reporting, most of the interesting calculations exist in the spokes, as 80 percent of the contents of a data mart are calculated results, not derived directly from source data. Hero (2001) concludes that once an organization keeps all calculation rules in a centralized meta data repository, any changes in the rules will be broadcast to all remote spokes. This approach is close to the federated data warehouse model introduced by White (2000b). From a software application development perspective, this type of hub-and-spoke approach is very difficult. From an analytical application frame perspective, the distribution of application logic is wide, requiring tight control of the replication/transfer of business rules across different spokes.

Our definition of architecture in this study relates back to product platform theory and the definition of Ulrich and Eppinger (1995): "the architecture of a product is the scheme by which the functional elements of the product are arranged into physical chunks and by which the chunks interact." To be able to compare our three architectural models from a software development perspective, we defined factors that would have an impact on the development of an analytical application frame and its corresponding frame components. One of the factors that makes software development more difficult when compared with development of physical products is the fact that the software development environment (the development environment) can differ from the environment in which the software solution is run (the runtime environment). This is something that software development organizations must consider carefully. We selected four major factors in our comparison table (see Table 1):

- Characteristics of the analytical application frame architecture
  - Type of architectural model
  - Requirements of the underlying IT infrastructure
  - Requirements from an end user organization that could reflect on the selected architectural model
- Functional flexibility of the analytical application frame architecture
  - Modularity of the application frame
  - Data distribution requirements in end user organizations
  - Selection of IT infrastructure elements such as DCOM, CORBA etc.
- Complexity of software development
  - Complexity of the runtime and development environments
  - Complexity of the architectural model
- Core competence requirements
  - Core competence of software vendor
  - Core competence of third-party vendors
  - Selection of technology, market segment, and product functionality based on the software vendor's core competence

Comparison Criteria	Centralized Data Warehouse Model	Federated Data Warehouse Model	Distributed/multi-tiered Data Warehouse Model
Characteristics of analytical application frame architecture • Type of architectural model • Requirements of underlying IT infrastructure • Requirements from end user organization that could reflect on selected architectural model	• Client/server environment with centralized data warehouse database server and typically also one hardware environment	• Requires more from frame architecture as independent or dependent data marts require a holistic or virtual view of all data • Might also require more of underlying IT infrastructure as due to virtual data warehouse functionality	• Product architecture is typically multi-tier with different flavors of data distribution (independent data marts and dependent data marts) • Requires typically more complex hardware, software architecture and is more costly to maintain
Functional flexibility of analytical application frame architecture • Modularity of the application frame • Data distribution requirements in end user organizations • Selection of IT infrastructure elements such as DCOM, CORBA etc.	• All functional components are within the same solution architecture (data extraction, database model, data distribution) • Data distribution to other remote locations could be a challenge using centralized model – remote users might not have access	• Data warehouse consists of several data marts that will be viewed in a holistic way, new additional data marts are added via a common data staging area and business area model is updated at the same time. • End user organizations have the flexibility to view subject specific data marts and link these into an overall data warehouse view of the organization.	• Selected technological environment very challenging for a software vendor and but a distributed frame architecture (DCOM or CORBA) enables support of remote locations if the distributed model is built into the frame • Flexibility is tied to selected distributed model. Dependent data marts are controlled by enterprise DW, while independent data marts can cause isolated data silos.
Complexity of software development • Complexity of runtime and development environment • Complexity of architectural model	• All functional software components within one centralized location, could lead to sloppy internal frame architecture	• Resembles centralized data warehouse model, but implementation of logical data warehouse model requires technical skills from the software development organization	• Distributed computing is very challenging for software developers as data marts (both dependent and independent) could reside in different geographical locations
Core competence requirements • Core competence of software vendor • Core competence of third-party vendors • Selection of technology, market segment and product functionality based on software vendor's core competence	• Traditional client/ server programming model • Architectural model that is typically known also by traditional programming software engineers • Does not necessarily require competency in distributed environments	• Client/server programming with application programming interfaces as dependent or independent data marts must conform to a logical view to the end user using the system • An understanding of the overall architectural model with data staging areas etc. and how these can be easily used from an end user application	• Requires selection of distributed computing model and this additional infrastructure selection requires skills from the software development organization • An understanding of distributed environments with its complexities • An understanding of distributed runtime- and development time environments such as CORBA and DCOM

Table 1. Comparison of Data Warehouse Architectural Models.

3.4.3.1 Characteristics of an analytical application frame architecture.

The aim of the first criterion is to portray the IT infrastructure model that the selected architectural model represents and what type of requirements can be foreseen when selecting the model. The main difference between the three architectural models when comparing analytical application frame architecture is the fact that both the federated and distributed/multi-tiered architectural models require more from the analytical application frame architecture, both from the IT infrastructure a well as the software development perspective when compared to the traditional centralized architectural model. Secondly, both the federated and the distributed/multi-tiered model can include both the data mart and the data warehouse architectural models as their underlying foundation, while the centralized model uses typically one or the other but not both. The centralized architectural model is less demanding for end user organizations from an IT infrastructure perspective, as these organizations have to maintain only one hardware/software environment for the analytical application environment, while the two other models require more from the IT infrastructure.

3.4.3.2 Functional flexibility of an analytical application frame architecture

The second criterion portrays the functional flexibility of an analytical application frame architecture. In the centralized model, all functional analytical application frame components are within the same hardware/software environment, while in the federated architectural model, additional data marts can be added via a common data staging area. The common business area model is updated at the same time. From an IT infrastructure perspective, the federated architectural model could be very close to the centralized model if the independent data marts reside in the same hardware/software environment. The federated data warehouse model also provides the opportunity to have several different data marts across different hardware and software environments. The software solution itself will cause the end user experience to be "virtual": the underlying architectural construct with either data warehouse or subject-specific data mart models can be seamlessly viewed using an end user tool. From an end user organization perspective, the federated architectural model enables organizations to build an overall view of the business in a manner similar to that which the traditional enterprise data warehouse architecture was intended to provide. The adaptability of an analytical application frame architecture in the distributed/multi-tiered model is tied to the selected distributed model, as some hardware/software selections could restrict the usage of the solution in some runtime environments. If the surrounding data marts are maintained by a centralized data warehouse environment, the consistency and reliability of the surrounding data marts could be better than in an architectural model where each data mart is treated as an isolated subject-specific data mart without a centralized business area model controlling the business rules with corresponding key metrics.

3.4.3.3 Complexity of software development

The third comparison criterion is viewed from a software development perspective. This comparison criterion is critical for a software vendor. First of all, if the software vendor selects a centralized model as the model for development, all functional analytical application frame components will typically reside in the same hardware/software environment. In the federated architectural model, the software development team must be able to provide a holistic view of the business even if the model includes one or more independent data marts. This requires both technical skills and development of a logical software application layer that provides business views to a disparate physical implementation. From an IT infrastructure perspective, a federated architectural model can be either centralized into one hardware/software environment or it can be physically distributed into several hardware/software environments.

3.4.3.4 Core competence requirements

The distributed/multi-tiered environment is difficult from a software implementation perspective, as the analytical application frame components must be implemented using a distributed technology such as DCOM or CORBA. Regardless of the selected architectural model, it will require more core competence from the software development and end user organizations due to its more complex IT infrastructure environment. This is the final and fourth comparison criterion. According to our comparison table, both the federated environment and the distributed/multi-tier environment require more core competence from both software development organizations and the organizations maintaining the solutions.

3.4.3.5 Conclusion

When we review these different data warehouse architectural models, the question remains: which of these models is most optimized for analytical application frame development, providing "an optimal architectural construct that is common to all derivative products?" The response to this question can not be viewed solely from an analytical application architectural perspective for several reasons. First of all, the solution and the flexibility of the solution is dependent of many other factors, such as the underlying IT infrastructure and the selection of a software development approach. Secondly, if the software vendor selects a development environment that does not provide the needed requirements for the selected architectural model, the software development environment becomes a restriction on the selected architectural model. Thirdly, selection of architectural model could result in lock-in to the selected architectural model, as transition from one architectural model to another might be technologically the same as rewriting the analytical application frame. Changes from one data warehouse architectural model to another were discussed thoroughly by Russell (2000) and additional insight was provided by Hero (2001), who discussed the hub-and-spoke architectural model and what it entails in distributed environments.

Our comparison of different architectural models from the software vendor's perspective shows clearly that the question of architectural model is not only dependent on functional/application specific criteria, but that it has to be evaluated against a selected technology. An analytical application frame component can be implemented as a local frame component without any distributed functionality. This will obviously prevent the software vendor from extending its solutions to market segments that require distributed characteristics. We can conclude that the dependence of analytical application frame technology on analytical application frame architecture is critical for future market segmentation attempts, as one or the other could impact how easily the software vendor can derive products for new market segments.

From a functional perspective, a centralized architectural model does not provide the flexibility an information user might desire when compared to highly distributed solution architectures. The federated architectural model provides the necessary centralized business area models and data staging areas to make ETL procedures more effective when requirements change frequently. A centralized model enables organizations to have all ETL, business rules, and other corresponding meta data in one centralized location due to its centralized character. This type of architectural model does not support the advantages of hub-and-spoke architecture, where extraction, transformation, and load can happen either in the spokes (data sources) or alternatively in the hub itself (typically the data warehouse server). The centralized concept fits where everything is centralized and all information for the data processing must exist in the central location. The distributed/multi-tiered model is effective model for end user organizations, where for example sales representatives need their local copy of data to be able to run the application in offline mode. A distributed architectural model will require more from a software vendor, but this model will give the software vendor more flexibility to define an analytical application frame that can be configured in different ways, such as placing functionality in different architectural tiers. This type of distribution is not possible in traditional centralized solutions: the underlying application frame does not have the ability to distribute functionality across different architectural tiers.

Based on our comparison of different data warehouse architectural models, we propose a hybrid model that includes characteristics from both the federated and the multi-tiered/distributed architectural models. A key element of these environments is a common business area model that enables end user organizations to share key business metrics across all dependent or independent subject-specific data marts. The federated data warehouse model includes a data staging area concept that enables software vendors to minimize the volatility of operational data sources. A data staging area helps software vendors to create prepackaged ETL templates between the data staging area and the data warehouse structures, while the ETL logic is typically adjusted in each and every end user environment due to differences in operational applications. We do not believe these types of adjustments will go away any time soon. A multi-tiered/distributed environment gives software organizations the flexibility to create software solutions that can take advantage of distributed environments, with characteristics such as better recovery and scalability and access to local meta data in remote locations if needed.

3.5 Selected Analytical Application Functional Architecture

An aim of this chapter is to provide a constructive analysis of an analytical application from both the functional and the information flow perspective. We identified the needed functional components that an analytical application software solution requires in our prior analysis. Another aim of this chapter is to demonstrate and build an architectural model – an Analytical Application Functional Architecture - that demonstrates how each functional component is connected to each other and how these functional components cooperate and behave when data is loaded into the solution and end users utilize the information in their analysis. This model is based on the recommendation we made in our comparisons of different data warehouse architectural models. We concluded that a hybrid approach with characteristics from both the federated and the multi-tiered/distributed architectural models is the most suitable solution for an analytical application software solution.

In the first sub-chapter, we will introduce an Analytical Application Functional Architecture with all related functional components. In the second sub-chapter, we will discuss information flow, with the corresponding functional components. In the third and final sub-chapter, we will discuss the requirements that a software vendor must satisfy when distributing analytical information to end user devices.

3.5.1 An Analytical Application Functional Architecture

Regardless of the data warehouse architectural model selected, an analytical application requires a set of functional components to fulfill the functional criteria set by end user organizations. We compared three different architectural models (centralized, federated, and multi-tiered/distributed) in a previous chapter with respect to their pros and cons, concluding that the centralized data warehousing architectural model includes more cons than pros when implementing adaptive data warehousing architectures. A hybrid data warehouse architecture with distributed characteristics as a foundation for an analytical application software solution provides a compelling foundation for an analytical application architecture. The data staging area reduces the volatility of operational data sources. Centralized meta data and business area models standardize business metrics across all dependent and independent data marts, and centralized ETL rules function for the overall analytical application functional architecture.

Each functional component/module in an analytical application should interact in a predefined and controlled way. All functional components within an analytical application solution are controlled via effective application programming interfaces that enable external organizations to utilize the functionality of the analytical application frame without having to worry about internal details. This type of modularity was discussed in Chapter 2. The requirement for these functional software components/modules is that they be generic, because they are part of an analytical application frame and will be used across different vertical market segments.

Based on our comparison of different data warehouse architectural models, we concluded that a hybrid data warehouse model provides the needed characteristics for an analytical application software vendor. First of all, a traditional centralized data warehouse model does not provide the distributed characteristics that end user organizations require if and when these organizations are highly distributed. Secondly, implementing a solution based on centralized approach does not necessarily lead to a technical implementation that lets the software vendor change from one model to the other as was described by Russell (2000). Thirdly, the federated data warehouse model provides two key concepts for an analytical application software vendor. These are a centralized common business area model and a data staging area that shields the software solution from volatility that the operational application might cause.

By combining the architectural functional elements from federated and multi-tiered/distributed architectural models and three-tiered packaged application architecture as shown in Figure 11, we can derive the needed functional components and the architectural tiers that are needed within an analytical application software solution (see Fig. 12).

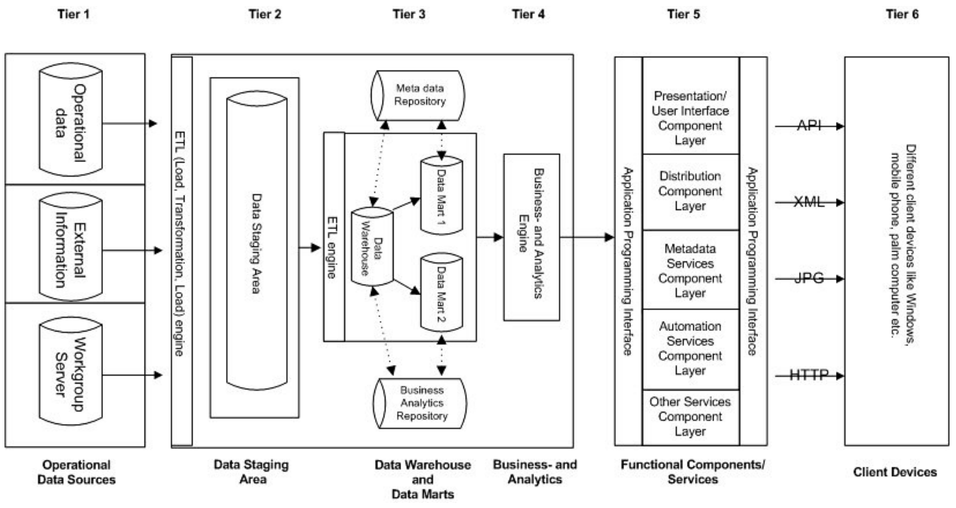

Tier 1
Tier 2
Tier 3
Tier 4
Tier 5
Tier 6
Operational data
External Information
Workgroup Server
ETL (Load, Transformation, Load) engine
Data Staging Area
ETL engine
Meta data Repository
Data Warehouse
Data Mart 1
Data Mart 2
Business Analytics Repository
Business- and Analytics Engine
Application Programming Interface
Presentation/ User Interface Component Layer
Distribution Component Layer
Metadata Services Component Layer
Automation Services Component Layer
Other Services Component Layer
Application Programming Interface
API
XML
JPG
HTTP
Different client devices like Windows, mobile phone, palm computer etc.
Operational Data Sources
Data Staging Area
Data Warehouse and Data Marts
Business- and Analytics
Functional Components/ Services
Client Devices

Fig. 12. Analytical Application Functional Architecture.

We have divided our architectural model with corresponding functional components into six separate tiers. Each tier has is predefined role in the overall solution. First of all, tier one represents the operational data layer: Enterprise Resource Planning software, payroll software applications, or any other software application that represents and operational application with aims completely different from those of an analytical/decision support software solution. According to Inmon (2002), "the operational level of data holds application oriented primitive data only and primarily serves the high-performance transaction-processing community," while the data warehouse level of data "holds integrated, historical primitive data that cannot be updated." The role of each of these environments is specific to their aim and must be clearly separated to avoid any conflicts, such as performance problems.

Tier two represents the layer between the data warehouse model and the operational data. The role of this layer is to isolate numerous changes that could take place in the operational data (White, 2000b; Jarke et al. 2000). Some of these changes could be related to upgrades of operational applications, with corresponding changes in database structures. In some cases, an end user organization might completely replace an operational application to another. The role of the data staging area is to isolate these types of changes and to enable analytical application software vendors to build predefined ETL rules between the data staging tier and the data warehouse tier (tier three).

Tier three represents a combination of the centralized data warehouse model and subject-specific data marts controlled using the centralized business analytics and meta data repository. The role of these repositories, according to the federated data warehouse model, is to provide consistent measures across any organizational unit using the analytical application, and also to ensure controlled and centralized meta data with corresponding ETL rules across the solution. In some cases, a software vendor might decide not to implement a data warehouse architectural model in its analytical application solution, but to base the overall solution on either independent or dependent subject-specific data marts. In these cases, the solution includes one to many data marts without a centralized data warehouse. Regardless of the architectural model (with or without a data warehouse structure), the analytical application solution integrates and has dependencies to meta data and business analytics repositories that provides the needed centralized metrics to the solution and integration logic to the data staging layer. These dependencies are portrayed as dotted lines in Figure 12.

Tier four represents the multidimensional/analytics engine, which interfaces the data warehouse/data mart tier with the corresponding meta data and business analytics repository. In some cases, this tier is built on OLAP architectures and functionality, while in other cases this architectural layer could consist of a software engine that provides multidimensional functionality by reading relational data and converting it into a multidimensional view.

Tier five consists of several functional services/engines that enable end users or other applications to utilize an analytical application solution. These functional components are separated from tier four using effective application programming interfaces. Tier five includes meta data and automation services that can be used by other software applications to request meta data or execute other automation tasks, such as report distribution.

Finally, tier six represents different user interfaces that end users might use while analyzing corporate data within the analytical application solution. The main idea for an analytical application is to use standardized interfaces such as OLE DB for OLAP, an industry standard for accessing multidimensional data from an OLAP database.

3.5.2 Information Flow in an Analytical Application

An analytical application frame can be said to be effective if it can support implementation of different vertical solutions using the same underlying software package. The underlying premise of analytical applications is that they will include predefined business processes, predefined measurements, and mostly also predefined extraction, transformation, and load definitions to selected operational applications. According to our architectural model above, we have defined an operational data store or alternatively a data staging area that acts as the interface between the operational application and data warehouse or data mart database solutions. Van Dyk (2002) recommends that organizations “maintain a unique normalized staging area/operational data store as a single source.”

The value of having a data staging area is to be able to provide a centralized location for integration of different operational data sources into one centralized data store with source and target interface mappings that can be reused when new data sources are added into the data staging area. This architectural model protects the analytical application from unnecessary volatility that could happen if the extraction, transformation, and load rules and processes were implemented between the data staging area and the analytical application and not between the data staging area and the operational data source. Another advantage of having a centralized data staging area is to be able to combine internal and external data. This could provide valuable information to end user organizations when combined and analyzed. If the data staging area were completely left out of the overall architectural model, each separate data mart would have to have its own data extraction algorithms.

The data staging area basically acts as interface file layer for the data warehouse layer, making it easier to manage the changes that occur in the core operational applications and reducing the volatility between the operational applications and data warehouse/data mart architectural tier. All extraction logic, together with source and target meta data, and source and target meta interface mappings with transformation rules, must be maintained in one centralized location, which in our case would be a central meta data repository. Once the sources have been identified for the selected vertical market segment, the corresponding data model for the data warehouse or subject-specific data mart is created, and this definition is stored in the common business analytics repository. This repository also includes all the needed information about the business processes for the solution and is therefore the intellectual knowledge database for the solution itself.

We have separated the business analytics repository from the common meta data repository, as the former is specifically implemented to enable vertical solution leverage for an analytical application vendor from an application frame perspective. A meta data repository will include all of the needed elements that are not specific for a vertical industry, while the business analytics repository includes all industry specific information. When the packaged analytical solution is installed at an end user organization, the installation process will create all physical database models with included business processes from the business analytics repository. There are plenty of discussions and differences of opinion of how a physical database model should be laid out in an analytical solution. Ralph Kimball (1996, 1998; Kimball and Ross, 2002) and several other authors suggest dimensional modeling with star schemas for the proposed database model while others such as Imhoff (2000) argue that a star schema does not comply well with data mining activities or statistical analysis. If a user wants to perform exploratory analysis, there should "be no hint of bias or arbitrary establishment of data relationships" (Imhoff, 2000). This requirement is obviously contrary to the star schema, as it typically has predetermined and physicalized relationships that enable multidimensional analysis. We argue that installation of the analytical application should include an automated process of database creation according to the business metrics that are stored in the business analytics repository. The same requirement applies to the staging area model, as this must be implemented for the integration of operational data sources. All end user organizations might not accept implementation of a data staging area, and this needs to be an optional feature in the overall analytical application, enabling ETL linkages directly to the data warehouse or data mart tier.

As the physical database model is such as a skeleton without data, the next logical step is to populate these database models with data from operational data sources. Operational databases in end user organizations are typically dispersed in different operational applications, and these applications can be distributed across the organization in different operating system and hardware environments. The aim of an analytical application software vendor should be to enable the extraction of information by using a standard protocol such as XML to transfer data from different operational sources. This transfer can be based on predefined schedules. It should include basic agent technology, where the source application notifies the transfer mechanism that transfer should take place. In the analytical application architectural model, the first integration level is from the operational data source to the data staging area. This integration is performed using extraction, transformation, and load functional components. This integration operation typically requires three separate elements, the ETL process descriptions (in a physical table), an ETL process source map, and ETL process statistics. The first element describes all of the extraction processes from operational sources, the second element shows how operational source fields are mapped into the data staging area, and the ETL process statistics show statistics such as time schedules for the process. This work requires upfront definition and is typically implemented one operational data source at a time.

3.5.3 Information Distribution in an Analytical Application

The results from an analytical solution are distributed to end users using several layers of functional components before they are displayed on the user interface of choice. Information in either data warehouse or data mart databases is kept in physical relational tables. When the end user wants to analyze this information, a multidimensional engine provides the needed "slice-and-dice" functionality that is expected in analytical solutions. There are two main ways of providing multidimensional information: either by storing and using a multidimensional model or by using a multidimensional engine that reads relational tables in a data warehouse or a data mart solution and does the calculation "on-the-fly." Both of these approaches have pros and cons. For example, the latter approach will require a powerful server hardware environment to perform the calculations.

The main purpose of the multidimensional calculation engine is to fulfill the role of information broker between the end user's analysis and actual data in the data warehouse. The calculation engine implements the needed calculations that the business analytics repository transmits. The calculation engine logic is closely related to report templates or predefined reports that align the given vertical market segment with predefined business processes and key performance indicators. One could obviously argue that even if the analytical application vendors are able to create a generic analytical application model for a specified vertical market, the end user organizations will want to change it to reflect their own environment. Changes that end user organizations want must be implemented in either the centralized meta data repository or the business analytics repository. This provides the ability for the software vendor to maintain the solutions and provide new versions of the solution without breaking existing solutions that end user organizations have amended.

Typically, a multidimensional calculation engine has to interface in two different directions; we will call these directions the input and output interfaces. First of all, the engine needs to "understand" the underlying logical data model (that links to the physical data model), and secondly it needs to provide analysis capabilities for the end users via a proprietary or open API. This output interface provides data for analysis (OLAP) tools with predefined reports. Alternatively, end user organizations can build applications using the published API and leverage the data warehouse calculation engine as a "black box" for their information requests. If the application programming interface is proprietary, the software vendor must provide its own front-end reporting tools, but if the API is based on an industry standard API, any reporting tool that complies with this standard can be used.

The challenge for an analytical application software vendor is to provide an intelligent wrapper layer between the multidimensional engine and the database models (both logical and physical) that is exactly the same for any given vertical market segment. The emphasis will be on the software component interface management that basically provides this abstraction layer. The second layer that needs to exist is the layer between the calculation engine and predefined reports. This "output" interface knows how to provide predefined reports with corresponding calculation results. This is implemented using an effective application programming interface and by using the information stored in the business analytics repository. These predefined reports are typically saved in physical relational tables with predefined attributes and rules that the calculation engine is able to utilize. The same calculation engine will typically also provide a data stream to a communication module that manages both traditional client/server communication and new distributed computing models. This distributed nature will require more from the software application frame technology sub-strategy, as each functional component in the analytical application frame can reside in different architectural layers on different physical or logical server environments. From an implementation perspective, the programming logic should flow equally whether the data warehouse model is centralized or distributed.

The last layer in an analytical application architectural model is the layer responsible for displaying information. The underlying data warehouse architecture has to support different user interfaces such as web-based access, traditional PC-based access, and different mobile devices. Some of these end user tools have strong emphasis on being able to navigate in different relational and multidimensional databases using open standards for access (such as ODBC). Other analytical application software vendors might build their own proprietary user interfaces that are tightly integrated into their own data models and structures. Standardization within the analytical application software space could potentially increase the competition of software vendors, as end user organizations could select end user access tools based on functional criteria rather than on what database might be supported.

According to Gleason (1998), user access tools are selected based on the user community. Some organizations will be happy with basic query and reporting tools, while some organizations require more complex OLAP tools. To satisfy different needs, software vendors should select their application development environment and corresponding software assets with the aim of providing maximal reuse of existing software assets across different end user access environments. Software development tools with corresponding application frameworks (such as Microsoft .NET) should support automated deployment of different end user devices such as Personal Digital Assistants (PDA's), Web browsers, and more traditional personal computers with Windows operating systems. This application development approach enables software development organizations to concentrate on building application or domain specific logic and leave application infrastructure support to vendors such as Microsoft.

3.5.4 Conclusion

The existing analytical application and data warehousing literature does not explain in detail how each and every functional component within an analytical application software solution interacts with every other component. Our architectural construct – Analytical Application Functional Architecture – portrays how information flow is organized within an analytical application solution. This architectural model is a combination of the multi-tiered/distributed model and the federated architectural model, as it includes functional components - business analytics, meta data repository, and data staging area - which are characteristic of the federated data warehouse model.

The Analytical Application Functional Architecture shows the importance of modularity and the layered architectural model with effective application programming interfaces between the layers. This provides software vendors with the ability to replace modules or enable third-party vendors to utilize the software solution using published application programming interfaces. The client device tier (tier six) demonstrates how end users do not have to worry about underlying data structures, as the user interface should provide a "virtual view" of underlying business analytics. All services in each layer should be transparent to the end user. The underlying architectural model can be either multi-tiered/distributed, where each layer includes summarized information from the previous architectural layer, or alternatively the view could be logical, where databases are located in different physical locations across the network. Regardless of the architectural model selected as the foundation for an analytical application solution, the software vendor could potentially face difficulties when changing from one architectural model to the other, as explained by Russell (2000).

Information distribution within an analytical application could pose challenges for software vendors in the future. According to Gleason (1998), end user organizations have to evaluate analytical application solutions based on "how information is delivered to different types of users." Gleason (ibid) provides three factors that end user organizations should consider: user classification, user mobility, and scalability. The first factor portrays the need for different types of end user access tools, where business analysts might require heavy-duty analytical power while executive users are typically happy with executive dashboards or other types of easy-to-use viewers. Due to these different requirements, an analytical application frame and its adaptability can be controlled using effective application programming interfaces. This is a core requirement within product platform theory.

The second factor is user mobility, which will have an impact on the application frame architectural model from an information distribution perspective. Organizations that are geographically dispersed and highly distributed with mobile users could potentially require a combination of multi-tiered/distributed functionality with local business analytics and meta data, while other users could be satisfied with internet-enabled access in a centralized environment.

The third and final factor is application scalability. According to Gleason (ibid), scalability in the case of an analytical application is “the property that provides support for additional users, larger databases, and higher performance by adding more computer resources.” This requirement translates back to the selected architectural model that is the foundation for the analytical application solution, and whether the analytical application frame is flexible from a scalability perspective. In our comparison of different data warehouse architectural models, we concluded that the traditional centralized data warehousing model could potentially have scalability problems when compared with the federated and multi-tiered/distributed architectural model. An analytical application solution based on the distributed and federated architectural model could potentially have several database/hardware environments to provide the scalability for the solution, with the environment viewed as one from the end user perspective (the virtual view).

3.6 Chapter Summary

Our comparison of different data warehouse models showed clear advantages for the federated data warehouse model with a centralized meta data repository providing common dimensions and measures for both dependent and as independent data marts. This architectural model could be called the “architecture of all architectures” as it provides a “virtual view” of the whole enterprise. This approach is also based on the “bottom-up” implementation approach, wherein the enterprise-wide analytical solutions are built from dependent data marts with an enterprise view in mind. Another advantage of the federated data warehouse model is the added data staging area, which enables easier implementation of analytical solutions as a staging area hides the complexities of operational data sources. We recommend selection of some of the characteristics of a federated data warehouse model combined with the multi-tiered/distributed data warehouse architectural model as the best foundation for analytical application software development.

We also concluded that, regardless of which architectural model is selected as the foundation for an analytical application solution, existing data warehousing literature portrays these architectural models on an abstraction level which is too high, with an emphasis on information flow within the solution. Because of this, these models will not be helpful to analytical application software vendors as the foundation for software development. Software vendors require more detailed architectural descriptions of how these functional components within an analytical application interact with each other, and what type of complexities a software vendor could run into during software development. More detail-level research is needed in the analytical application software domain; research specifically geared towards software development organizations. This type of literature is non-existent and therefore this study will add practical and tangible advice for software vendors within the analytical application solution domain.

Thanks to our literature search of functional requirements for an analytical application combined with analysis of different data warehouse architectural models, we were able to construct an architectural model which we named the "Analytical Application Functional Architecture." This architectural model demonstrates the complexities of information flow and the dependence of different functional components within an analytical application solution. The challenge for an analytical application software vendor is to provide an architectural model that will provide an optimal architectural construct that can become the basis for an analytical application frame. The challenge in the implementation is the heterogeneity amongst end user organizations, as each organization, even in the same vertical domain, might have different requirements for business metrics, associated reports and charts, ETL integration, and more. Management of database models and prepackaged solutions is by no means easy to implement and maintain, and these types of solutions should have the ability to be maintained and updated by end user organizations while still having version compatibility with new releases from the software vendor.

In our analysis of different data warehouse implementation models and data warehouse architectural models, it became clear that selection of a product development strategy requires additional dimensions for analysis, such as market segmentation and technology selections. Selection of the optimal analytical application frame architecture can not be achieved without aligning the selection to underlying and defining technology. Both of these technologies could impact the adaptability of an analytical application frame architecture, and selection of one will impact the selection of the other. Because of this, to be able to resolve software related implementation issues with respect to analytical application software, we need to evaluate the impact of technological choices on both the analytical application frame architecture and current and future market segmentation.

The main aim of the next chapter is to refine the framework from Sääksjärvi (2002) to reflect the analytical application software domain and analyze what type of alignment perspectives can be identified when implementing analytical application software solutions. Each alignment perspective could emphasize each sub-strategy differently, causing a software vendor to experience different types of disconnects if and when one sub-strategy receives less emphasis than the two other sub-strategies. These strategic alignment perspectives have not been analyzed in any previous decision support literature, either from a software development perspective or even an end user organization perspective. Based on our findings so far, the next research question is as follows:

RQ 3: How can software vendors balance their software product architectures when changes take place in marketing and/or technology selections?

The analysis of research question three will be the foundation for our fourth research question and the foundation for analyzing how analytical application frames can be implemented from a software engineering perspective.

4. DEFINING A BALANCE BETWEEN SUB-STRATEGIES IN ANALYTICAL APPLICATION SOFTWARE

Our analysis of analytical application software characteristics and functionality in the previous chapter gave us an understanding of the complexities that software vendors could face when developing analytical application software solutions. It was also evident to us that an analytical application is more complex that many other software solutions, as it has to include the ability to provide predefined business metrics with corresponding analytics applicable to end user organizations in specific vertical market segments. This generalization of software solutions requires adaptability of the software application frame, which can be defined as "the ability to utilize a common architectural construct across selected vertical market segments." It is important to realize that this architectural construct could be different in different software companies, as the selection of which vertical market segment to serve can be very different between software companies.

Our aim in this chapter is to identify an optimal common architectural construct that can be reused across all selected vertical market segments. We will define required functional components for an analytical application frame based on our recommended hybrid architectural model, described in Chapter 3. We concluded that usage of product platform theory within the software product domain requires continuous analysis of three sub-strategies (technology, application frame architecture, and application frame leverage), and that this analysis will result in different alignment perspectives that a software vendor can utilize in setting its business strategy. We will demonstrate each and every sub-strategy separately in this chapter together with the combined Analytical Application Frame Strategy framework that forms the foundation for building analytical application software solutions using product platform theory.

4.1 Defining an Analytical Application Frame Architecture Sub-Strategy

The primary aim of this chapter is to define an analytical application frame architecture sub-strategy with its corresponding components. The secondary aim of this chapter is to compare different analytical application architectural models from two different perspectives: those of the end user organization and the software vendor. This comparison demonstrates a traditional intersection of information systems science and software engineering/computer science. This type of analysis is useful for software development organizations, as end user organization IT infrastructure could become an obstacle to a software vendor's market segmentation strategies.

4.1.1 Analytical Application Frame Architecture

The challenge of defining an analytical application frame and its architecture can be viewed from several different perspectives. First of all, the software vendor should maximize the granularity of the software application frame to enable maximal reuse of existing software assets.

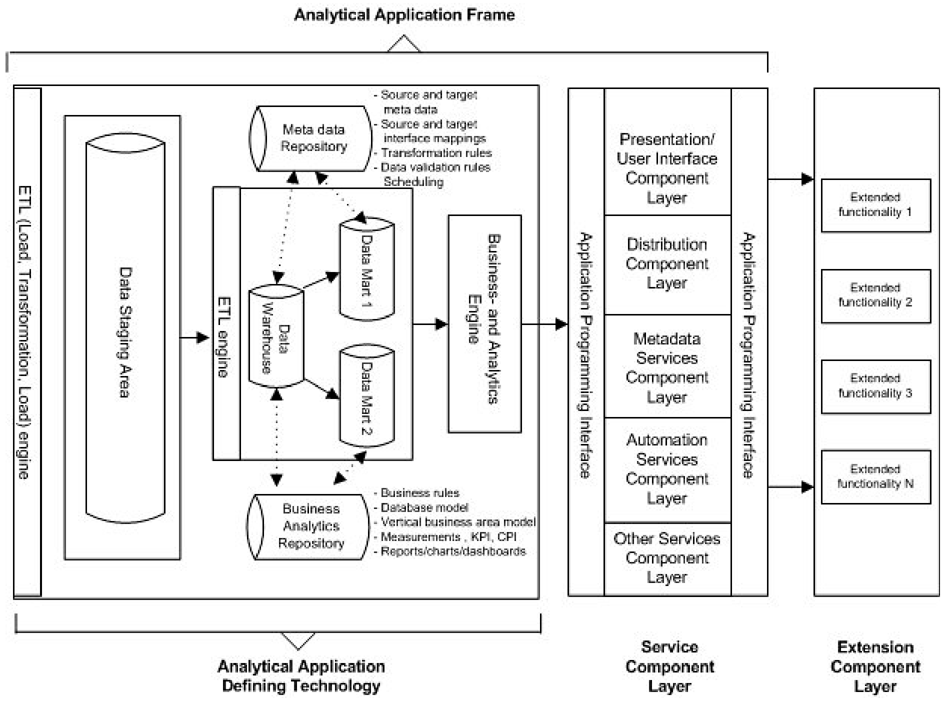
Secondly, a software vendor must have a thorough understanding of an analytical application software domain and what functionality is expected in an analytical application. We gave a definition for an analytical application software solution in Chapter 1. A further definition is needed to describe the functional components within an analytical application solution, specifically concerning an analytical application frame. These functional elements – analytical application frame components – are constructs that compose an analytical application frame. Therefore, we define an analytical application frame component as follows:

> "An analytical application frame component is a coarse-grained functional entity that is part of an analytical application frame. One or more of these frame components compose an analytical application frame architecture. A frame component can be anything from a granular software component to a database repository."

Our analysis of existing data warehousing (Sach, 1997; Watson and Haley, 1997; Gray and Watson, 1998) and analytical application software (Morris, 1998; Gleason, 1998; Surgan, 2000) gave us direction as to what type of functionality must be found in an analytical application. We also concluded in prior chapters that an analytical application will typically include either data warehousing or data mart technology as its underlying foundation, and therefore the solution must include related functionality, such as ETL technology, database administration tools, a database management solution (a data warehouse storage engine), and some kind of data distribution functionality, perhaps OLAP or traditional EIS interface technology. As our research concentrates on analytical applications, selection of a data warehouse architectural model with its corresponding characteristics is of importance for an analytical application software vendor.

We concluded in our comparison of data warehousing and data mart architectures that a hybrid data warehouse model with characteristics from both federated and multi-tiered/distributed functionality provides a foundation for software organizations to build an analytical application frame that can be the foundation for derivative products within a product family. The value of the federated data warehouse model is the existence of a common business area model and a meta data repository that can be used across the data warehouse and data mart models. This ensures integrity in business metrics across the organization. The data staging area reduces the volatility that operational data sources could cause for software and end user organizations. We excluded the traditional centralized data warehouse model as a foundation for an analytical application solution, as it does not correspond to the requirements of highly distributed end user environments, where end users have access to local meta data and business analytics without having to have access to a centralized data warehouse/data mart architecture.

From a software development perspective, a traditional centralized model would have been arguably the simplest model, as all functional components reside in the same hardware environment, whereas in highly distributed environments, software organizations must build software infrastructure to manage data distribution and data integrity to be able to support these types of solutions. Another valuable characteristic of multi-tiered/distributed environments is that they allow end user organizations to install the solution either in a centralized or distributed fashion, as all of the functional components can reside in the same hardware/software environment. We defined an Analytical Application Functional Architecture in previous chapter that included all of the functional components needed for an analytical application software solution. To define an optimal architectural construct (analytical application frame), we have to identify all of the functional components that are needed across all vertical and/ or horizontal market segments. This will become the analytical application frame. This type of reasoning combined, with our suggested hybrid data warehouse model, leads us to the following architectural model for analytical application software solutions (see Fig. 13):



Analytical Application Frame
ETL (Load, Transformation, Load) engine
Data Staging Area
ETL engine
Meta data Repository
- Source and target meta data
- Source and target interface mappings
- Transformation rules
- Data validation rules
Scheduling
Data Warehouse
Data Mart 1
Data Mart 2
Business Analytics Repository
- Business rules
- Database model
- Vertical business area model
- Measurements , KPI, CPI
- Reports/charts/dashboards
Business- and Analytics Engine
Application Programming Interface
Presentation/ User Interface Component Layer
Distribution Component Layer
Metadata Services Component Layer
Automation Services Component Layer
Other Services Component Layer
Application Programming Interface
Extended functionality 1
Extended functionality 2
Extended functionality 3
Extended functionality N
Analytical Application Defining Technology
Service Component Layer
Extension Component Layer

Fig. 13. Analytical Application Frame Architecture Sub-Strategy for an Analytical Application.

An analytical application frame establishes the foundation for all derivative vertical solutions that a software vendor introduces to the marketplace. Thus, we have divided an analytical application frame into two layers: the "Analytical Application Defining Technology" includes functionality that gives the software vendor a vector of differentiation, while the "Service Component Layer" includes functionality that can be built on top of defining technology. The third architectural layer portrays the “Extension Component Layer,” which enables software organizations to extend the software application frame with extensions that provide the means to fulfill the required vertical market requirements without having to rewrite the entire software application frame.

We asserted in previous chapters that the aim of a software vendor is to identify an optimal software application frame to maximize the leverage of software assets across selected market segments. Identifying an optimal architectural construct can be very different for different software vendors, as the selection of technologies, vertical market segments, and product architecture can be very different from one software vendor to another. A software vendor can only measure software application frame effectiveness and efficiency ex-post, where considerable investment might already have been made in the software application frame before the measurement results are available to executive management. Another key measurement for a software vendor is to maximize the lifetime of the software application frame, as each derivative product cycle within an effective software application frame provides leverage to the investment that the software vendor has in the software application frame. In some cases, an optimal analytical application frame could be inefficient (suboptimal) for one software organization and optimal for another. Our reasoning for selecting specific functional components as part of the analytical application frame is based on the analysis in Chapter 3, in which we identified all of the functional components that were needed within an analytical application software solution. The software vendor needs to identify all of the required components to be able to have an analytical application solution. These components are portrayed in Figure 13. From a semantics perspective, the arrow from the data staging area to the data warehouse portrays the data flow from the data staging area to the data warehouse via ETL rules. The dependency between the meta data and the business analytics repositories is portrayed with bilateral arrows, as a change in any of these will have an impact on the solution. The data flow from the data warehouse structures to the data mart environment is portrayed with unilateral arrows, as each data mart is loaded from a centralized data warehouse environment. In a similar manner, the information flow to the business and analytics engine and corresponding overlaying architectural layers is unilateral: information is read from the lower levels and displayed to the end user via different user interface devices.

4.1.1.1 Analytical application defining technology

The two key repositories in an analytical application frame are the meta data repository and the business analytics repository, providing the needed physical database structure to store information for any given vertical market segment. The software vendor must separate the physical database structures from the actual solution content that will give selected vertical market segments the needed business metrics and integration with given operational applications, such as ERP software packages. One of the main components in an analytical application is the ETL component, which enables end user organizations to load information from different operational sources. The ETL engine utilizes at the time of data load rules defined in the meta data repository that include information about source and target meta data, source and target interface mappings, different transformation rules, data validation rules, and scheduling functionality.

A meta data repository, together with the business analytics repository, is a key component which defines an analytical application defining technology. The business analytics repository consists of business rules for the given vertical market segments, the database model itself, key metrics for the given vertical market segment, and reports (charts) and dashboards for the given market segment. The importance of having a centralized business analytics repository is evident when an end user organization wants to create both dependent and independent data marts that will be sharing common business analytics rules with the business analytics repository.

Besides these repositories, we have included data staging database structures, data warehouse, and data mart database structures as key components in an analytical application software solution. These structures, together with extraction, transformation, and load procedures, are required regardless of the selected vertical market segment. These ETL procedures are controlled by meta data and business analytics repositories. Meyer and Zack (1996) defined a similar model in their analysis of information products in which the contents of the information products are generated using what they defined as their "process platform." A key element of defining technology is the "business and analytics engine," needed to leverage the contents of the solution and provide analytics results to the service component layer.

4.1.1.2 Service component layer

An analytical application software solution requires a multitude of different services that we have not listed as defining technologies. Some of these services are optional and some software vendors might not have all of the listed functionality in their solution. These services can be called externally from any other software application as long as the software vendor has published its application programming interfaces. Based on the analysis of analytical application software solutions in Chapter 3, we found that at least the following services are needed in these types of solutions:

- Presentation and user interface component layer
- Distribution component layer
- Meta data services component layer
- Automation services component layer
- Other services component layer

The presentation and user interface component layer provides the needed functionality for the software vendor to support a variety of different end user interfaces in end user data analysis. A key requirement in analytical application solutions is to be able to provide the same chart/report to different client technologies such as Windows, an Internet browser, a Palm device, or any user interface (UI) that the vendor might support, without having to maintain different versions of the same chart/report. Several software development environments have automatic support for different end user (UI) devices, enabling software vendors to concentrate on building the business logic, rather than building infrastructure software components. The concept of separating the presentation layer from the business layer is known as the model-view-controller architectural model (Krasner and Pope, 1988). Similar experiences were reported by Sharp (2000) in the development of avionics software using a layered architectural model and a medium-grained architectural model. The use of patterns enabled the case study company (The Boeing Company) to spread expertise across the developers, enhancing reuse in development of the software solution.

The distribution component layer provides the needed functionality to distribute reports/charts to different types of user devices. This component layer includes the ability to publish and subscribe to different reports/charts based on given intervals or threshold values that end users provide for each report/chart. A typical threshold value could be a budget variance number that would trigger an event if the value exceeded or were under a specified value.

The meta data services component layer is used in conjunction with the other services, such as the presentation/user interface component layer, as this service provides end users with the needed meta data about selected reports/charts and the source of the information. In some cases, the meta data services component layer provides meta data information to other software packages via information exchange using published application programming interfaces. This layer includes information about the existing data warehouse and/or subject-specific data marts and provides to end users a "virtual view" of the overall solution.

The automation service component layer provides end user organizations with the ability to automate the overall solution, from data extraction to report/chart distribution. This type of service is important in environments where the solution is embedded with other software applications, as one software application might execute actions within the analytical application software solution by calling these automation services.

The other service component layer represents all other possible services that an analytical application solution might include. This functionality varies from one software vendor to another.

4.1.1.3 Extension component layer

The combination of analytical application defining technology with its corresponding functionality and the service component layer is defined as an analytical application frame.

These layers are required in any analytical application software application regardless of vertical or horizontal market segment. Some software vendors within the analytical application software domain might have slightly different software application frames because of their market segmentation or core competence. The final layer in the analytical application frame architecture is the extension component layer, used to implement specific functionality required in selected market segments or horizontal solutions. These extensions can be implemented by the analytical application software vendors themselves or alternatively by third-party development organizations that provide add-on solutions that can be used with the overall analytical solution.

The boundaries between basic functionality and extended functionality could be very narrow, and in some cases a software vendor might decide to include functionality in the analytical application frame to avoid software related versioning and configuration overhead. Some of this extended functionality could be managed by software configuration, using different software engineering techniques such as parameterization, inheritance, and other methods. The use of an extension component layer will always require application of some type of software engineering variation technique. These techniques will be explored in more detail in Chapter 5.

4.1.2 The Analytical Application Frame Architecture From Two Perspectives

We compared different data warehouse architectural models in Chapter 3 purely from a software vendor's perspective. Another perspective is to compare analytical application software development from both the software vendor's perspective and that of the end user organization. The former perspective reflects the typical requirements of software engineering and computer science, while the latter perspective reflects end user IT infrastructure research and usage, that is, a traditional information systems science perspective. Both perspectives are important, but the difference in these viewpoints is considerable. According to Schuff and Louis (2001), IT departments have traditionally "cycled between centralized and decentralized application software distribution, although modular program design and enterprise management software may break that cycle." According to the authors, IT departments in end users organizations are achieving major cost savings by centralizing and standardizing software using thin-client environments. The authors recognize that centralized software architectures might not be appropriate in all organizations, and therefore these organizations prefer distributed schemes. Our selection of a hybrid data warehouse foundation (federated with distributed characteristics) for a software vendor was based on increased functionality requirements from end user organizations, where remote end user locations have a need to analyze information using local meta data and data repositories. This type of environment can be supported only by using distributed technology.

Schuff and Louis (ibid) define a framework with three evaluations criteria when seeking balance between control, reliability, and speed in an architectural model. These three factors are application modularity, bandwidth, and feasibility of a uniform configuration. When evaluating these from two different perspectives, we defined three main comparison criteria (see Table 2):

- Selection of an analytical application frame architectural model
- Functional adaptability of the analytical application frame architecture
- Underlying information technology infrastructure

Criteria	Software vendor's perspective	End user organization perspective
Analytical application frame architectural model • From IT infrastructure perspective • From development environment perspective	• Selection of analytical application frame architectural model will impact several things such as selection of development tools, selection of IT infrastructure, and market segmentation in some cases. • Changes from one architectural model to another might be impossible in some cases, for example, the change from a distributed architectural model back to a centralized architectural model.	• Existing IT infrastructure might not support some architectural models – some models (such as middle-tier solutions) might require additional IT infrastructure investments. End user organizations are typically not that interested in how the internal frame architecture is implemented. • End user organization are more interested in external IT infrastructure requirements such as whether the solution is client/server, mainframe or fully distributed with middle-tier technologies.
Functional adaptability of analytical application frame architecture • Business model with business processes • Key performance indicators, critical success factors • Software application frame modularity • Flexibility to change and update the solution • Flexibility to create derivative products to different market segments	• Each frame component must be flexible and support several different vertical markets without having to change the underlying frame components. • Vertical market solutions must be managed via business analytics and the meta data repository • All frame components must support ongoing maintenance and additions to the solution that will be updated to end user organizations. • Application modularity enables software vendors to create different variations of the software more easily than if the application were monolithic.	• End user organization must be able to amend the solution delivered by the software vendor. They also have the ability to update the solution in parallel with own internal solution development without jeopardizing the integrity of the solution. • Software application frame architecture must provide the flexibility to change the underlying IT infrastructure without breaking the overall solution.
Underlying information technology (IT) infrastructure • Change of IT infrastructure from one to another • Impact on future market segmentation	• The software vendor must select its underlying IT infrastructure environment in concert with its development tools and core competencies. Wrong infrastructure selection can also restrict market segmentation, as some organizations might not have the required IT infrastructure. • If the application frame is closely tied to the IT infrastructure, the software vendor might not be able to amend its solution to other market segments in the future.	• Selection of analytical application solution must reflect the underlying IT infrastructure environment that exists in the organization. Some analytical solutions could be suitable for the organization, but the solution from the software vendor does not support appropriate IT infrastructure environment. • Changes in IT infrastructure can create integration problems within an end user organization.

Table 2. Architectural Comparison Criteria for a Software Vendor and an End User Organization.

4.1.2.1 Selection of an analytical application frame architectural model

Another critical area of consideration is selection of the software architectural model, both from the end user organization IT infrastructure perspective and the software application frame perspective. Some software organizations have achieved lock-in in their application frame architecture. This lock-in will impede end user organizations planning to move to other solutions. Morris and Ferguson (1993) discuss proprietary architectures and how these architectures can provide a technological edge for software organizations with the results of lock-in strategy. A good example of this type of software organization is Microsoft, which has its own proprietary operating system environment with its corresponding software development environment, software application tools, and database management system. In similar way, IBM has achieved lock-in with the IBM midrange computer hardware environment, as the software implemented in those environments using proprietary development tools will only work in those environments. During the last few years, IBM has opened the iSeries/400 environment to cross-platform development tools such as Java and C++. Even if the IT industry discusses open environments, the unfortunate reality is that each "standard" has different implementations. Examples include UNIX and CORBA and associated technologies.

Gawer (2000) and Gawer and Cusumano (2002) describe how Microsoft has achieved its software platform leadership and how it has even moved to what is called supporting application software development, competing with other software vendors that provide solutions for Microsoft operating system environments. Other vendors, notably Intel, carefully consider any development in support product areas due to the competitive reasons and the fact that they might not get large product platform leverage without these third-party development organizations.

Selection of an architectural model for an analytical application could be very different. Selection criteria for a software vendor and an end user organization may vary considerably. First of all, software vendors have to consider the underlying infrastructure environment that a selected market segment broadly supports. Secondly, the selected architectural model must reflect back to the core competence of the software vendor, as a highly centralized architecture is very different from a distributed or federated architectural model. Thirdly, the selected runtime and development environments have to support the domain-specific architectural model, which in this case is the domain of analytical applications. Some development environments are proprietary to specific hardware environments, and this could restrict the selection of the hardware and/or operating system environment.

A software vendor should select a software development approach that enables the vendor to move from centralized to distributed data warehouse architectures without having to rewrite the defining technologies (and corresponding frame components). According to Russell (2000), a move from the distributed to the centralized data warehouse model could be extremely difficult compared with a move from the centralized model to the distributed model, but if the software architectural model and the development approach are modular, movement in either direction should be possible for the software vendor. Our conclusion is that it is impossible to select an optimal analytical application frame architecture without first analyzing data warehouse architectural models together with the corresponding analytical application frame technology sub-strategy and the analytical application frame leverage sub-strategy. This analysis is important, as the dependencies in the underlying and defining technologies will impact the adaptability of the analytical application frame architecture to different market segments.

4.1.2.2 Functional adaptability of frame architecture

Adaptability of an application frame architecture could mean completely different things to a software vendor and an end user organization. A software vendor aims to identify "the most optimal architectural construct" that can be reused across different market segments, while an end user organization measures architectural adaptability in terms of its ability to distribute information across the organization and change and maintain the analytical application without breaking the solution that was delivered by the software vendor. Functional adaptability also means that end user organizations are able to create incremental updates on the solution itself concurrently with the software organization. New updates of the analytical solution can be inserted on top of end user organization modifications. Another sign of adaptability to an end user organization is the ability to change the underlying IT infrastructure without having to discontinue the use of the solution or change the solution provider. An example of this would be iSeries/400 solutions that can not be run in any other hardware or operating system environments. To a software vendor, functional adaptability means that the solution can be used as the foundation for several derivative products in a product line/family and that the software application frame is sufficiently modular to be changed to reflect new IT infrastructure requirements. Application and software application frame modularity give software development the flexibility to drop and add new software modules on a required basis. This type of component-based development is not possible in old legacy applications without strong application wrappers.

4.1.2.3 Underlying information technology infrastructure

The third comparison criterion is to compare IT infrastructure requirements between the software vendor and end user organization. This comparison reveals that end user organizations have typically already selected the underlying IT infrastructure to be supported, while software organizations usually aim to support as many IT infrastructure environments as possible. This mismatch between supported IT infrastructure environments (the software vendor) and required infrastructure environments (the end user organization) could be considerable, limiting future market segmentation for the software vendor. Another dilemma that an end user organization can run into is that of integration problems: when the application software solution breaks due to IT infrastructure issues. An example of this could be when an end user organization updates any element in the underlying technology (operating system environment, database management system etc.) and this causes the software solution to break. These types of problems could cause a nightmare to software vendors (as well as end user organizations), as these problems are unexpected and typically very difficult to identify.

4.1.3 Summary

We based our analytical application frame architecture model on our analysis of analytical application solutions in Chapter 3. Selection of the data warehouse architectural model was based on architecture comparison, also portrayed in Chapter 3. We also defined an analytical application frame architecture based on our functionality analysis. One of the key tasks that a software vendor must accomplish is to identify an optimal architectural construct (software application frame) that can be used across a given vertical market segment. The key is to understand that the aim for a software vendor is not to try to support all of the possible market segments, but to predict as accurately as possible the segments that the company is probably going to address with the selected software application frame. It is not accurate to suggest that a software application frame is exactly the same for two different software vendors, but it is accurate to claim that each software domain must include a minimum set of functional components included in the solution. We have identified these minimum requirements that any analytical application software vendor must support. These functional components are either part of the defining technology or part of the service component layer. We also concluded that each software vendor might have a slightly different service component layer, as each software vendor could address different market segments and have a different product development strategy. Another key element is to understand that defining technology. Its embedded functional components can provide a key competitive advantage for a software vendor. This key competitiveness can be achieved by intelligent implementation of the required functionality. This will give the software vendor a "vector of differentiation" as was described earlier in this chapter.

We also emphasized the importance of addressing the end user organization and the software vendor perspectives when implementing and selecting an architectural model with corresponding IT infrastructure for an analytical application software solution. Software vendors have to consider the underlying IT infrastructure to be supported, as end user organizations could have a variety of different underlying IT infrastructure technologies. This finding will further emphasize the lack of discussion in existing traditional product platform theory of the complexities that software vendors face when selecting a product platform approach in developing software. It is justified to claim that a software vendor has to analyze a combination of three dimensions (selected product architecture, technology, and market segment) before defining an optimal software application frame for a given software domain. The first dimension is the analytical application frame with included components (defining technology and the other layers within an analytical application frame architecture), the second dimension is the underlying technology, and the third dimension is the selected market segment. We can conclude that it is impossible to define one or the other without including discussion of the remaining two. It is important to be able to define the defining technologies as flexibly as possible so that the software organization can select new technologies to support new IT infrastructures when a selected market segment so requires. Incorrect selection of an IT infrastructure could lock in the software vendor for years and prevent the vendor from moving into new lucrative market segments.

4.2 Defining an Analytical Application Frame Technology Sub-Strategy

Part of the overall business strategy for a software vendor is to define a technology strategy that enables the software vendor to achieve the implementation of a flexible application frame architecture that gives the software vendor the ability reflect and adjust to the requirements that are set by a specific market segment or even the software functionality that is needed to fulfill the expectations of the end users. We have divided this chapter into two sub-chapters with an initial analysis of technology strategy and implications of selecting technologies. We also discuss and explore technology selections specifically for an analytical application software vendor.

4.2.1 Technology Selections for an Analytical Application Software Vendor

Software technologies are evolving at an increasing pace. This could lead to technology selections that become obsolete from both the software architecture and the market segmentation perspective (Clements and Northrop, 2002). According to McGrath (2001), a vendor has to identify and understand "the future roadmap of key technologies, emerging technologies that could affect the vision in the future and unrelated technologies that could possibly create substitute products." Selection of an underlying technology could potentially impact long-term software development strategy for a software vendor, specifically if technologies are selected inadequately and without reflecting the market segmentation strategy of the software products. Cowan et al. (2002) conclude in their article "Software Engineering Technology Watch" that software vendors are facing tremendous difficulties in trying to predict what technologies will bring onboard in the future. According to the authors (ibid), "the evolution of software technology is fast paced and determined by many factors. Most cannot be identified, let alone predicted, with any significant advance notice." Even if a software organization were able to predict technology trends, it must be able to adapt to them without having to rewrite the entire application suite that the vendor might have implemented for a market segment.

According to Clements and Northrop (2002), technology forecasting is used within product family software development to ensure the survivability of any technology selection. From a software application frame (product platform) perspective, each selection of technology can have an impact on the lifecycle of the derivative products and thereby also on the efficiency and effectiveness of the software application frame. Technology forecasting helps in identifying technology trends and predicting relevant markets and what standards will prevail in the future. Clements and Northrop (ibid) divide technology forecasting into two different areas, namely internal development and customer solutions. Internal development includes "selections of tools, processes, and methods for producing the software that will end up in the software," while customer solutions are decisions that reflect the end user organizations and what features and capabilities are embedded into the products. Technology forecasting is important within product line/product family development, as each technology selection could impact the future derivative product development efforts both in a positive and a negative way. If technology selections are made poorly, the software organization might face a situation in which the overall solution must be replaced due to architectural inflexibility or obsolete technology.

A classical perspective of software architecture is provided by Jacobsen et al. (1997), in which IT infrastructure components (system software components) are the foundation and lowest level in a layered software architectural model. The following layer consists of middleware components that enable software organizations to build distributed solutions. These middleware components can include both domain-specific and generic middleware components. The next layer before the application systems layer contains business-specific components that portray the selected software domain and its associated software components. Jacobsen et al.'s (ibid) architectural model is a generic model which does not specifically consider other aspects of software technology selections.

Bosch (2000) concludes that standardized infrastructure consists of "the operating system and the typical commercial components on top of it, such as the database management system and a graphical user interface." The following two layers within a layered architectural model are business-specific and application systems. The former layer is domain-specific and the latter layer portrays complete applications systems. (Jacobsen et al., 1997). Pronk (2000) concludes in his case study within the medical imaging software domain that an architectural model for a product line should have one main objective. This objective is to avoid "a monolithic design by extreme de-coupling of components and localization of functionality so that every component can be replaced or upgraded in isolation." Pronk's (ibid) statement reflects back to our definition of an analytical application frame and its characteristics. Pronk (ibid) portrays an architectural model for the imaging software domain by having three horizontal architectural layers (a technical layer, an application layer, and a user interface and display) and a separate vertical infrastructure layer that "shields the general purpose hardware and the operation system calls, and offers basic support classes for licensing, logging, and other infrastructure services to be used by all software."

We concluded earlier in this chapter that we have separated the domain-specific product architecture (federated, centralized etc.) from the selected technological style (client/server, distributed etc.), as these can be very different from each other. Domain-specific architecture describes the domain-specific software characteristics, such as in the case of a federated data warehouse model that provides a "virtual view" of the data warehouse/data marts database across the end user organization IT environment. The domain-specific architectural model could in some cases be using a different technical implementation architectural style. An example of this would be when the domain-specific solution supports the centralized data warehouse model, but the implementation architectural style is distributed. This type of combination gives the software vendor the flexibility to support two different types of domain-specific architectural models. Because of this, we have separated these two architectures by defining a domain-specific architectural style as part of analytical application frame architecture and a technological architectural style as part of technology strategy.

Our findings in technology permutation (Table 3) and discussion of the dependences between the domain-specific architectural model and selected technologies demonstrate the impact of underlying infrastructure selections on software application frame development. These selections are to some extent tied to the selected software domain. In the case of an analytical application software solution, the software vendor must select the operating system environment, the database management system, the distributed component technology (such as DCOM and CORBA), the architectural style, and non-domain specific application building blocks. These selections are tightly integrated to the selected integrated development environment. Some IT infrastructure selections could limit the selection of integrated development environment and vice versa. The software vendor must select an appropriate development approach for its software development. During the years, several approaches have been introduced to the marketplace, but none of these has provided the "silver bullet" defined by Brooks (1987). A solid proponent within the product platform development approach is software product line engineering. The general approach is the same, whereby a software vendor aims to create common software assets that can be utilized in derivative software development. The key therefore is to provide a modular software architecture with effective interfaces to enable the software vendor to create an optimal software application frame that can be reused over a longer period of time. Another key component in software product line engineering is component-based software engineering (Bass et al. 1998a, 1999; Clements and Northrop, 2002; Bosch, 2000). By summarizing the technology selections that a software vendor must make, we can identify following model (see Fig. 14):

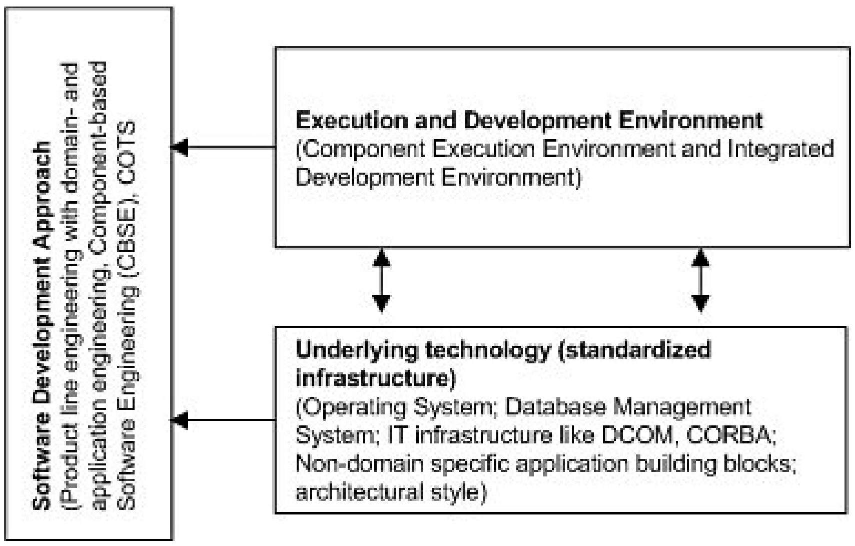

Software Development Approach
(Product line engineering with domain- and application engineering, Component-based Software Engineering (CBSE), COTS
Execution and Development Environment
(Component Execution Environment and Integrated Development Environment)
Underlying technology (standardized infrastructure)
(Operating System; Database Management System; IT infrastructure like DCOM, CORBA; Non-domain specific application building blocks; architectural style)

Fig. 14. Analytical Application Frame Technology Sub-Strategy for an Analytical Application.

Figure 14 demonstrates the relationship between the underlying technology, execution and development environment and the selected software development approach. It is imperative to conclude that technological selections as such do not necessarily impact the selected software development approach. This selection describes only what type of methodology is used when designing and implementing the software solution.

4.2.1.1 Underlying technology (standardized infrastructure)

Selection of an underlying technology, such as an operating system and database management system, can have long-lived impact on a software vendor, as some of these technologies are proprietary, supporting different combinations of hardware and software environments. A good example of this is the iSeries/400 environment, which did not initially support development environments other than its proprietary development language or any database environment other than its own. Software vendors who initially selected these environments have been forced to invest in new environments during the years, as some of the required functionality can not be built using pure iSeries/400 technology. Another approach that some software vendors have tried during the years is to support cross-platform development using specific application development tools and languages. These attempts have been costly to some companies, as in the case described by Cusumano and Yoffie (1999). Cusumano and Yoffie (ibid) explain how a selected technology strategy can lead to unexpected costs. Their example is from Netscape Corporation's selection of a cross-platform development approach for an Internet browser. Cross-platform development could potentially become a burden for a software vendor due to different technologies and possible conflicts in each selection. The following table shows an example of how an analytical application software vendor could face difficulty if the vendor decides to implement a solution for multiple hardware and operating system environments. The permutations of operating system environments, database management system environments, and distributed technologies can be vast. Adding one new technology to the table will increase the number of permutations exponentially (see Table 3).

Criteria	Microsoft Technology	UNIX or LINUX Technology	AS/400 Technology
Operating System			
• Microsoft specific OS	• Only Windows	• No support	• No support
• Generic OS	• No support	• Support for several OS	• No support for other OS
• OS/400	• No support	• No support	• Native support
Database environment			
• Microsoft specific DB	• Works in Windows	• No support	• No support
• Generic DB	• Works in Windows	• Support for several DB	• Support for several DB
• OS/400	• No support	• No support	• Native support
Development tools			
• Microsoft specific	• Works in Windows	• No support	• No support
• Generic	• Works in Windows	• Support	• Support
• OS/400 tools (ILE RPG)	• No support	• No support	• Native support
Distributed technology			
• Microsoft specific	• DCOM	• No support	• No support
• Generic	• CORBA	• Support	• Support

Table 3. Technology Permutation Matrix for a Software Vendor.

The aim of the permutation matrix above is not to simulate all the different technology permutations, but to show the complexity that an analytical application vendor might face in development of its software solution. The matrix shows clearly that selection of an operating system and hardware environment could influence all other underlying technology elements, and also all other execution and development environment selections.

Besides selection of an operating system environment and database management system, an analytical application software vendor must select an architectural style (client/server, pipe-and-filter, etc.) that could be different from the domain-specific architectural model specific to analytical application software solutions. We described three architectural models (centralized, multi-tiered/distributed, and federated) in Chapter 3 that can be the foundation for an analytical application vendor. These architectural models are specific to the software domain. The combination of these selections with the implementation architectural style will define how complex and adaptable the analytical application frame architecture will become and what type of granularity the software application frame will represent. When an analytical application software vendor selects its application and domain-specific architectural model and implementation architectural style, it has to align these selections to additional technologies such as distributed computing technology (DCOM, CORBA) and other IT infrastructure elements that are part of the underlying technology. Some of these architectural dependences can be isolated using a layered software architecture as proposed by Jacobsen et al. (1997) This could reduce some of the dependences that could be caused by technological selections.

4.2.1.2 Execution and development environment

The software tools that an analytical application software vendor uses as part of development of analytical application solutions must support software development from a whole lifecycle perspective. According to Jarig and Bosch (2002), existing software tools are too bound to specific development phases. According to Stuart et al. (2000), to achieve the full benefit of product line development, "processes and methods of the development effort must be supported by tools." According to Stuart et al. (ibid), an industrial consortium has been founded (Computer Technology Corporation) with the aim to developing tool support "needed to make product line development a reality."

Depending on our analytical application technology strategy, underlying technology will have an impact on the execution and development environment, as some integrated development environments are integrated into the underlying technology, such as the iSeries/400 RPG programming language. The executables from the RPG (Report Program Generator) development environment will run only in the iSeries/400 hardware environment. Some integrated development environments will support cross-platform development with different underlying IT infrastructures. This approach could help a software vendor to alleviate some of the risks associated with the selection of technology strategy. These cross-platform development environments have identical runtime and software development environments. This enables a software vendor to create software solutions for different IT hardware and software underlying technologies without having to provide different releases of the software package. Another scenario for a software vendor is to consciously select specific underlying technology for a selected market segment with the aim of achieving a competitive edge. If these selections are proprietary and might become legacy technology in the long run, this market segmentation could become a burden for the software vendor. Changes in the underlying technology for the software application frame could be outside the boundaries of the core competence areas that the software vendor represents. This type of evolution could become costly for the software vendor, as new personnel with associated technologies must be acquired in one way or the other.

Herzum and Sims (2000) introduced the concept of the "software component factory," emphasizing "reuse on a large scale" by using very large coarse-grained software components.

The authors (ibid) suggest that the technical architecture "is the set of architectural principles, models, and design patterns that defines and environment in which technical complexities at all phases of development are hidden from the functional developer." Basically, the developer should not have to worry about underlying technologies, but to concentrate on building the domain-specific features (that have been defined in analytical application frame architecture sub-strategy) and functions that end-user organizations expect to find in the software product under construction. This hiding is what the authors refer as a "business component virtual machine." The overall technical architecture is shielded from the developer, implementing all the IT infrastructure components that are required in a software development environment, such as the technical infrastructure with its component execution environment and an integrated development environment that together form the business component virtual machine.

Cheeseman and Daniels (2001) explain the importance of target technology, specifically in distributed computing environments when assembling components into applications and application systems. According to the authors, software component environments have to conform to existing component standards that provide basic infrastructure services such as transaction support, security, and concurrency for the application component to rely on. Sametinger (1997) explains the dependences between the selected hardware, the operating system, and other systems on the run-time environment. According to the author, components and component platforms are more reusable the fewer platforms and underlying infrastructure they depend on. Sametinger also distinguishes between the execution platform and the composition platform in a way similar to Jaaksi et al. (1999). The execution platform is the platform where a component can be executed. This is typically dependent on the underlying IT infrastructure environment. The composition platform (or the development platform) is the platform that is used when components are assembled, integrated, and compiled into a binary executable. The aim of any software development team should be to limit the dependences both in the run-time and the development environments to optimize the product functionality for the possible technological combinations. Efficient usage of application programming interfaces is a key in limiting the dependences and cohesion amongst software components in the architecture. D'Souza and Wills (1999) separate the design and composing activity of components into two different activities. The authors suggest delaying "the bindings made when components are composed so that the composition can be done as late as possible."

4.2.1.3 Software development approach

The software development community has transitioned from the traditional Waterfall and Spiral software development process models to iterative and incremental models. A software process is described by several different software engineering authorities (Royce, 1998; Kruchten, 1999; Cheesman and Daniels, 2001). Most of them describe it similarly to Ambler (2001) as "a set of project phases, stages, methods, techniques, and practices that people employ to develop and maintain software and its associated artifacts (plans, documents, models, code, test cases, manuals, and so forth)." There are numerous articles about current development strategies and processes (Brown and Wallnau, 1996; Pfister and Szyperski, 1998; High, 1998; Cusumano and Yoffie, 1999; Sparling (2000). A new industry standard modeling language has emerged, namely Unified Modeling Language (UML). This new modeling standard has been applied to different software development techniques, such as component-based engineering (Kobryn, 1999; 2000).

Several different software development approaches have been introduced on top of this new modeling language, one of the major approaches being the Unified Software Development Process from Jacobsen et al. (1999). Some of these new development process models favor the Unified Software Development Process (Fowler and Scott, 1997; Jacobson, 1999; Kruchten, 1999), but some argue that the proposed process model has deficiencies and must include new additions to be useful (Rosenberg and Scott, 1999; Ambler, 2000). One of these proposed additions is the ICONIX Unified Object Modeling approach from Rosenberg and Scott (1999). Another addition is the Object-Oriented Software Process (OOSP) from Ambler (1998; 1999), which extends the Unified Software Development Process with a production phase in the software process model. Two component-based models are also introduced, Catalysis from D'Souza and Wills (1999) and another from Herzum and Sims (2000). This approach is named the Business Component Factory. All of these latter software development methodologies are based on UML in one way or the other. Fowler and Scott (1997) point out that UML is a successor to all the different Object-oriented Analysis and Design (OOA&D) methods and that UML is a modeling language and not a method. According to the authors, a method will include both a modeling language and a process.

Software product line development has been introduced as a development approach with the aim of reuse in large-scale systems (Jaaksi et al., 1999; Brown, 2000). According to Atkinson et al. (2000), systematic product line development remains "the exception rather than the rule, and its potential remain largely unfulfilled." Atkinson et al. (ibid) continue, explaining that traditional software implementation technologies do not support the rapid and cost effective "adaptation of implemented code in a way required by [a] genuine product line approach." Because of this, Atkinson et al. (ibid) proposes a software development method KobrA (Komponentenbasieerte Anwendungsentwicklung) that cleanly integrates product line development and component-based software development into a "systematic unified approach to software development and maintenance." The authors (ibid) explain that product line development is typically "reuse in the large" while software development using a component-based development paradigm is "reuse in the small." The KobrA approach utilizes the strengths of both product line engineering and component-based software engineering.

Figure 14 portrays a software development approach with corresponding domain and application engineering practices and component-based development as the foundation for a software application frame implementation approach. Software product line development includes characteristics similar to software application frame development. First of all, both approaches have the aim of creating software assets that are common to all derivative products. Secondly, both approaches have the aim of reuse on a large scale, with a modular architecture and well-defined interfaces. Part of a software development approach is also the selection of a development notation such as UML. This selection depends on several criteria, such as the selected product line engineering approach, the tool selection, and core competence of the development personnel. We will explore in more detail in Chapter 5 how each selection impacts development from a practical perspective and how each selection is backed up with recent product line literature.

4.2.2 Summary

In our analysis, it became evident that selections for the underlying technology can have an impact on an analytical application frame architecture and its defining technologies. As an example, the selection of weak database management technology would adversely impact the robustness of several functional components within an analytical application employing the technology for the data staging area, the data warehouse, and the data mart implementation. If we draw a parallel between IT infrastructure selection and defining technology, the overlying layers within the analytical application frame architecture will be impacted as well. Selection of underlying technologies will have an immediate impact on the execution and development environment, as some development environments do not support all IT infrastructure environments and vice versa.

Our analysis also demonstrated the need to and importance of separating the IT architectural style from the domain-specific architectural model, as the former is part of underlying technology selection while the latter is defined in the analytical application frame architecture. Each of these selections is closely related but with different requirements, and each must be selected with different criteria. Software developers who concentrate mainly on the software application domain expect the IT infrastructure environment to provide services at runtime, such as replication functionality in the selected database management system.

All of the underlying technology selections will be the basis for the software development approach and the software development methodology with its included development processes. Our definition of a software development approach consists of all software engineering approaches, such as software product line engineering and its associated domain and application engineering approaches. The software development approach is complemented with development methodologies (such as the Rational Unified Process) with corresponding processes and component-based development. Part of software development for a specific software domain is the definition of a product line development approach with corresponding architectural models that will be shared by all derivative products. If the software organization does not include these in the process, analytical application frame development might be effective, but there might be additional factors that relate specifically to software product line development that must be considered to achieve maximum leverage for derivative product development. Product line development as such is the method of delivering derivative products using a software application frame. We will include discussion of different implementation variation techniques within the analytical application software domain in Chapter 5.

4.3 Defining an Analytical Application Frame Leverage Sub-Strategy

We concluded in our analytical application frame technology sub-strategy analysis that product line engineering contributes to the implementation of a product family in analytical application software solutions. There are many practical examples of the use of software product line development as a basis for derivative product development (Thiel and Peruzzi, 2000; Jaaksi, 2002). One of the key factors for successful derivative product development within a software company is dependent on how well the development organization understands the software domain (Kang et al., 2002). According to Schmid and Verlage (2002), domain engineering approaches have typically failed and run into schedule and cost overruns if the domains are not properly scoped. In similar way, the scope for a software application frame and the size and complexity of the frame are dependent on the specific market segments that the software vendor intends to support. Domain knowledge comes into play when the scope is defined. If the domain is unknown and new, the probability of encountering challenges while an optimal software application frame increases as the unknown becomes larger. A similar discussion is led by Bosch (2002), where he notes that organizations that "are more mature in terms of domain understanding" are more versed to concentrate on domain engineering. We will explore these topics in Chapter 5 when discussing different variation techniques that an analytical application software vendor can undertake while creating its strategy for derivative product development.

4.3.1 Scoping the Market Segment for an Analytical Application Software Solution

According to Cusumano (2004), software organizations have to decide whether they should have a horizontal or vertical market segmentation strategy. According to Cusumano (ibid), a horizontal market segmentation strategy might in some cases lead to "the lure of the horizontal," whereby organizations "overestimate the potential of horizontal markets and end up with products that have too few customers or that are too weak to combat the competition" (ibid). If we inject this statement into the overall trend in the analytical application solution market with the aim of creating prepackaged vertical market solutions (Eckerson, 2002), we can argue that software vendors in the generic infrastructure/tools market will face increased competition as end user organizations have the desire to buy and not build their software solutions, as argued by Gleason (1998) and Morris (1998).

Based on our analysis of product platform related software development, we have identified the importance of a solid product architecture and the importance of selection of technology in implementation of an application frame for analytical application software. The third remaining criterion for successful product family development is the selection of the appropriate scope for the products with respect to market segments and what software applications/solutions are going to be supported by the software vendor. According to Thiel and Peruzzi (2000) in their research on software development within the automotive software system domain, the case study company used product line scoping (PLS) to "identify the functional, technical, political, legal and business constraints that are characteristics to the product line." This activity included three main practices:

- Business investigation, to interview the business executives and to create the business case.
- Product space examination, to identify the characteristics of legacy and competitor products and future extensions to it.
- Standards and technology investigation, to identify the standards and technology drivers that typically comply with the products under consideration for the product line.

If we translate these activities into analytical application software solutions, we do not see anything that would be different when setting strategies for analytical application software solutions. Specifically, the third activity, standards and technology investigation, is important for an analytical application software vendor.

A software vendor faces a dilemma when having to build a software application frame without knowing in advance the possible future market segments and application areas that must be supported. The aim of the analytical application frame and its development is not to try to cover all of the possible variations, but to provide a solid combination of software application frame and software application frame technology that provide economical value in the form of software application frame effectiveness and efficiency. The economic results will be dependent on analytical application frame leverage and how easily a software vendor can provide for new market segments and applications. Part of the decision to be made when deciding on a product family is to set the scope that is economically feasible.

The basis for any software domain is to provide a robust analytical application software frame that the software vendor can use within different market segments without having to make profound changes into the application frame itself. We defined the needed analytical application frame components in this chapter, together with several architectural layers of domain-specific application functionality needed within any market segment. The difference between these analytical application solutions from one market segment to the other is the contents of the business analytics and meta data repository and possible extensions within the extension component layer. The variation within the solution content will be managed by a process in which intellectual property for the selected market segment will be included at the time of delivery using an installation process in which the business analytics and meta data repositories will be populated with market segment specific information. Analytical application frame extensions must be included to the binary executable that is delivered. This type of variation technique will be discussed more thoroughly in the Chapter 5.

If the analytical application frame does not enable further variation, the software vendor must renew the analytical application frame by redesigning the application frame or by creating additional extensions by building additional subsystems that will not disturb the foundation of the analytical application frame itself. Because of this, the challenge will be for an analytical application software vendor to define a solid and optimal architectural application frame that can be reused in all of the selected market segments without having to rewrite any functionality in the analytical application frame itself. This type of extension model was described by Meyer and Seliger (1998), where the final market segmentation is implemented using specific software plug-ins that can be maintained and developed by third-party software organizations. Within the analytical application frame architecture, third-party or vertical market specific software extensions can be implemented in the extension component layer. This layer will then be integrated into the actual application frame using different variation techniques.

According to Meyer and Seliger (1998) and Sääksjärvi (1998, 2002), market segmentation strategy can be implemented both horizontally and vertically using different price/performance tiers. Vertical market segmentation emphasizes the ability to build different software solutions for different market segments (such as manufacturing, insurance, distribution etc.), while horizontal market segmentation can be controlled by implementing software products for different performance/price tiers (see Fig.15).

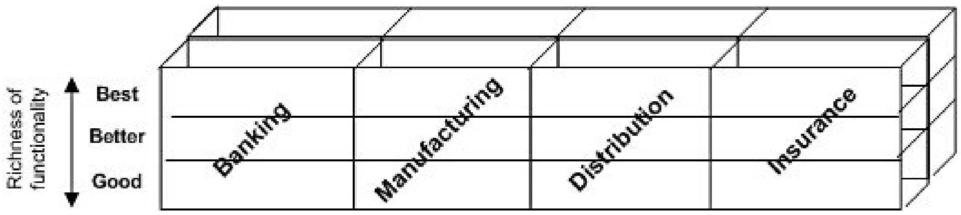


Fig. 15. Analytical Application Frame Leverage Sub-Strategy for an Analytical Application.

In reality, the software vendor can control each price/performance tier in different ways. One way is to scale on the price/performance scale by limiting the functionality of the vertical extensions or limiting the functionality by using control numbers within the application. An analytical application software vendor could, for example, limit the number of applications within a financial solution – the lowest entry level could have only loan analysis, whilst the highest level might include all modules and functionality found in the vertical solution. Another way of controlling functionality is to limit supported business processes or business metrics (for example key performance indicators). According to our analysis of an analytical application software solution, all business metrics and business processes are stored in a common business analytics repository. This repository is centralized, and specific software features can be controlled by software control numbers even if the whole analytical application is delivered to the end user organization.

4.3.2 Different Variation Options For An Analytical Application Solution

An analytical application solution can be varied by several different techniques, some of them being solution variation and some being technical variation. The remaining variation could be a movement on the functional richness axis of the analytical application frame leverage sub-strategy matrix. We concluded previously in this chapter that part of a robust analytical application is the implementation of a domain-specific architecture that provides the ability to build a robust underlying business analytics and meta data repository. The business analytics repository is specifically the foundation for collecting intellectual property from a specified vertical market segment. This knowledge is not related to technology, but to deep business understanding of the selected vertical market segment. The meta data repository describes "data about data," and includes all relevant information about the included source and target meta data and interface mappings and more. An analytical application software vendor must decide the boundary of how far "an optimal architectural entity" or software application frame can become the basis for any vertical market segment, as each modification in the extension component layers will increase the complexity of software development.

Software vendors must distinguish between the technical feasibility of maintaining different vertical solutions and the intellectual skills that somebody within the organization must provide by building predefined business processes and key performance indicators with their corresponding critical success factors. These business processes will have be linked to predefined charts and reports that are typical for the business domain. These charts will be delivered to end user organizations together with the software solution itself. When we analyze our analytical application frame from a market segmentation perspective, the analytical application's underlying technology is completely separated from the analytical solution that will eventually be delivered to the end user organization. Part of the delivery will be to populate the solution with corresponding vertical market intellectual knowledge. This process can be mastered at the time of packaging the deliverable for the end user organization.

In some cases, the software deliverable must include specific functionality to be implemented in the software package itself. This type of functionality will be included in the extension component layer as portrayed in our analytical application frame architecture sub-strategy. Therefore, in some specific cases, extended functionality frame components could even include software modifications or functionality that is specifically optimized for the selected vertical market. This extended functionality requires additional software versioning. It will be included and packaged after physical compilation of the software to binary code or (if the extension is a self-contained software component in binary format), it can be delivered as a physical extension to the analytical application frame. The analytical application frame as such is not a product that can be installed, but will be the basis for packaging and acting as the core for deriving products for different vertical domains by using extension frame components in the packaging. The challenge for any analytical application software vendor is to define the least common denominator functionality in the underlying technology and service component layer that can be the base for derivative analytical solution development.

One of the decision criteria for packaging could be market segmentation based on price/performance segmentation of the software vendor. One product family could be based on building solutions for the financial industry while the other product family could be addressing the insurance industry. Each industry has its own specific needs and needs to be perceived differently. The product lines or families could also represent different ways of analyzing information, such as via the Internet, or alternatively, using a native Windows client. This kind of derivative product development represents technology driven product segmentation, where the software application frame is used to derive new products for new IT infrastructure environments and is therefore not based on vertical market segmentation.

Another view of analytical application frame development is to view it as a base for different horizontal applications, such as customer relationship applications, data mining, or even corporate budgeting and forecasting. Each of these solutions can utilize the analytical application frame, but each solution also requires considerable additional software development. Some of this development could require exceptional functionality that cannot be seen as part of the analytical application frame and needs to be developed outside this application frame. This horizontal development takes more development resources and might lead to its own application frame development.

4.3.3 Summary

We analyzed the required elements for defining analytical application frame leverage using an analytical application frame. The breadth of the analytical application frame is dependent on the scope of the market segmentation of the software vendor. The more vertical market segments to be covered, the more challenging it becomes to define the analytical application frame architecture with associated technological choices. We also concluded the importance of solid domain knowledge as the foundation for defining both the functionality of the solution and the breadth of the market segments.

We identified three different possible variation techniques that an analytical application software vendor can utilize: each of these will be explained in more detail in Chapter 5. Solution variation is based on content variation, wherein the software vendor includes intellectual knowledge of given vertical market segments as key performance indicators, predefined reports/graphs, and business processes. The second variation technique is to use software application frame extensions (extended functionality) that provide the needed additional functionality that is not part of the functionality that is needed by every market segment. A third variation technique is to limit the functionality of the solution by using software control numbers where either the solution content or the functionality is limited by the given control number.

The main challenge for any software vendor is to identify the optimal software application frame entity that covers all the current market segments that the software vendor is going to cover and also possible future market segments. Another challenge that the software vendor is going to face is the selection of technologies (both defining and underlying) that could in some cases restrict future market segmentation if the selections are made carelessly.

4.4 Analytical Application Frame Strategy Framework

The Software Application Frame Strategy framework from Sääksjärvi (1998, 2002) was a generic framework with no specific emphasis on a given software application domain or any empirical consideration. For us to be able to analyze the impact of this framework on analytical application software solutions, we had to construct an analytical application frame architecture that demonstrated all needed architectural layers with corresponding functional components that are needed in these types of solutions. We concluded that software vendors need to separate the domain-specific architectural model from the software architectural style, as these two architectural models could have two different aims.

We also separated defining technology from underlying technology: the former defines "the vector of differentiation" for the software vendor while the latter consists of IT infrastructure technology that a software vendor selects as the foundation for an analytical application solution. The third major consideration for a software vendor is the selection the of market segment that the solution is going to address. The challenge for the software vendor is to define and construct an optimal architectural construct (analytical application frame) that satisfies both the existing and the future market segmentation that it intends to support. As a part of the process of selecting market segment, a software vendor must pose several important questions, as follows:

1. What customer groups or segment(s) are we going to serve?

2. What customer application areas are these customer groups or segments going to need to achieve the value-add from the software solution?
3. What is going to be the technological foundation to build these customer application areas that are needed in the specified customer group/segment?

The challenge for a software vendor is to find an optimal analytical application frame that implements the needed functionality for the selected and future market segments. The aim of the product platform concept is to maximize the economical measures for product platform effectiveness and efficiency. It is important to understand that the software vendor should not try to cover all the different defining and underlying technologies that might possibly be needed in the future. This type of software development would result in a complex analytical application frame that would be too expensive to maintain. The idea of economical performance with effectiveness and efficiency measurements would be negatively impacted.

When analyzing the relationships between each sub-strategy, we have expressed the relationship between each sub-strategy as "fit" and the relationship between the three sub-strategies as "alignment perspective" (process). The following figure portrays an Analytical Application Frame Strategy framework (see Fig. 16):

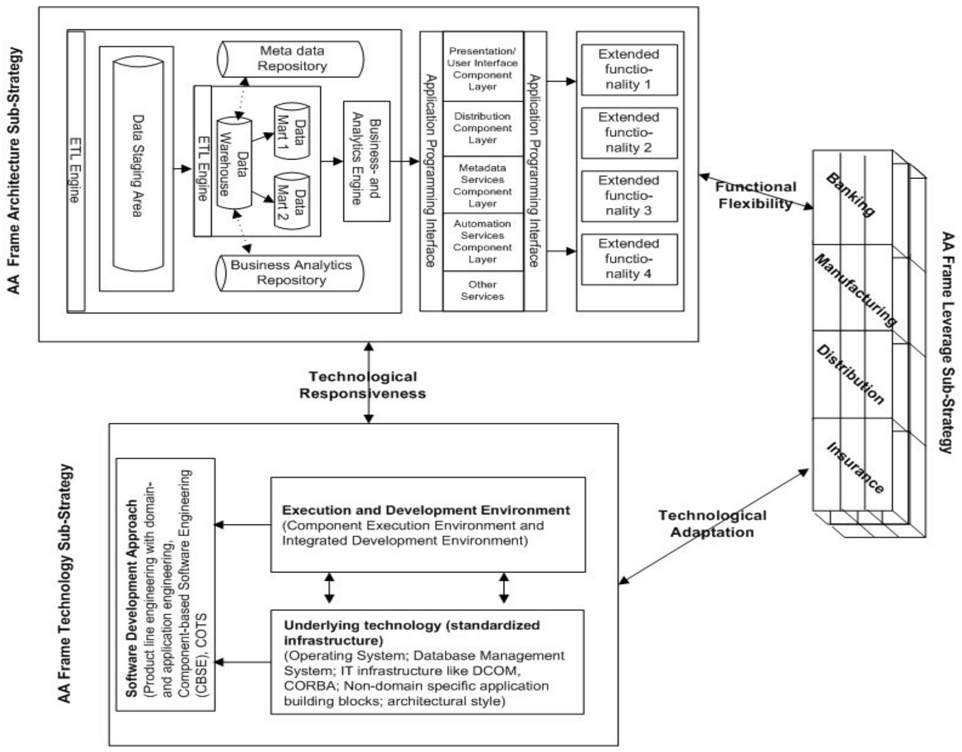

AA Frame Architecture Sub-Strategy
ETL Engine
Data Staging Area
ETL Engine
Meta data Repository
Data Warehouse
Data Mart 1
Data Mart 2
Business Analytics Repository
Business- and Analytics Engine
Application Programming Interface
Presentation/ User Interface Component Layer
Distribution Component Layer
Metadata Services Component Layer
Automation Services Component Layer
Other Services
Application Programming Interface
Extended functio-nality 1
Extended functio-nality 2
Extended functio-nality 3
Extended functio-nality 4
Functional Flexibility
Banking
Manufacturing
Distribution
Insurance
AA Frame Leverage Sub-Strategy
Technological Responsiveness
Technological Adaptation
AA Frame Technology Sub-Strategy
Software Development Approach (Product line engineering with domain- and application engineering, Component-based Software Engineering (CBSE), COTS
Execution and Development Environment (Component Execution Environment and Integrated Development Environment)
Underlying technology (standardized infrastructure) (Operating System; Database Management System; IT infrastructure like DCOM, CORBA; Non-domain specific application building blocks; architectural style)

Fig. 16. Three Different Fits Resulting from the Alignment of Three Sub-Strategies.

An Analytical Application Frame Strategy framework consists of three separate sub-strategies, namely the Analytical Application Frame Architecture Sub-Strategy (“application frame architecture sub-strategy”), the Analytical Application Frame Technology Sub-Strategy (“application frame technology sub-strategy”) and the Analytical Application Frame Leverage Sub-Strategy (“application frame leverage sub-strategy”). The relationship between each sub-strategy portrays the fit between the sub-strategies, each with its own characteristics. We have defined three fits in our framework:

- **Functional Flexibility** describes the relationship between the analytical application frame architecture and analytical application frame leverage sub-strategy
- **Technological Adaptation** describes the relationship between the analytical application frame leverage and analytical application frame technology substrategy
- **Technological Responsiveness** describes the relationship between he analytical application frame technology and analytical application frame architecture sub-strategy

Dependencies within each sub-strategy and its corresponding elements are discussed generally in this chapter to portray possible dependencies that can have an impact on the fit between two sub-strategies and/or alignment perspectives. These dependencies can in some cases be controlled by using any of the sub-strategies to reduce the impact of these dependencies. If, for example, a software vendor wants to reduce the impact of selecting underlying technology, the impact of the selection can be reduced by careful selection of market segment (Technological Adaptation). We argue that careful alignment perspective or fit can reduce the impact of dependencies between the elements in each sub-strategy. Our emphases are in the analysis of alignment perspectives, discussing more generally how the elements and groups of elements within each sub-strategy could potentially impact alignment perspectives or the fit between two sub-strategies.

To achieve a better understanding of different strategic alternatives in the alignment perspectives that a software vendor can take, we have used an analysis approach similar to that introduced by Henderson and Venkatraman (1993). Our analysis will consist of different alignment perspective variations and fit analysis to identify how emphasis on one specific alignment perspective or fit analysis will impact others. Finally, we will compare these alignment perspectives with each other and conclude whether one model versus the other could be an optimal alignment perspective for an analytical application software vendor.

From a semantics perspective, an arrow between each sub-strategy portrays a fit. Both sub-strategies could have an impact on each other (the arrow points in both directions). In a similar manner, within each sub-strategy, arrows between each element demonstrate a relationship either unidirectionally or bidirectionally. Within the analytical application technology sub-strategy, the underlying technology and the execution and development environment are bidirectionally dependent on each other. The selected software development approach does not necessarily impact the underlying technology or the execution and development environment, and therefore we have portrayed this with unidirectional arrows. Within an analytical application frame architecture sub-strategy, each arrow represents the relationship from an information flow perspective, portrayed with unidirectional arrows. Bidirectional arrows are used when a change in any structure could impact both sides of the arrow, such as between data warehouse structures, the meta data, and the business analytics repository.

4.4.1 Analysis of Fit Within the Analytical Application Strategy Framework

Both the analytical application frame architecture sub-strategy and the analytical application frame technology sub-strategy will have an impact on the market leverage of the analytical application. If the application frame architecture is the driving force, the fit between these two sub-strategies is called "*Functional Flexibility.*" This fit portrays the ability of the software vendor to adjust its solution to different market segments from a functional perspective. We emphasized previously that the aim of an analytical application software vendor should not be to cover all possible market segments that it could possibly cover in the future, but the selection should be based on factors such as economic feasibility (application frame effectiveness and efficiency) and other factors that are related to analytic application frame architecture and its selection (such as the architectural model, functional flexibility etc.).

The relationship between analytical application frame technology and analytical application frame leverage sub-strategy is called "*Technological Adaptation.*" This relationship defines how well the software vendor is able to adapt its technological selections to reflect current and future market segmentation alternatives. In some cases, a software vendor might select underlying technology that is restricted (such as the iSeries/400 hardware and operating system environment) but could potentially offer competitive advantage for a software vendor due to strict focus on selected technology and possible innovations around that technology.

The third and final fit demonstrates the software vendor's ability to change either the frame technology sub-strategy or the frame architecture sub-strategy without breaking the overall solution. We have named this relationship or fit "*Technological Responsiveness.*" The fit between an analytical application frame architecture and an analytical application frame technology sub-strategy can not be undervalued, as this fit will impact the future of the company. The result of this fit portrays how well the frame architecture can adapt to new technological changes and whether the analytical application frame architecture is amendable to new market segments. It can therefore be said that, however good the software company's market segmentation, failed technological responsiveness fit can result in a doomed future.

4.4.1.1 Technological responsiveness

"Technological Responsiveness" characterizes the type of impact the selection of underlying technology has on an analytical application architecture or how the analytical application frame architecture can be changed, reflecting possible new underlying technologies. When the software vendor is successful in selecting underlying technology that can be changed or amended during the lifetime of the analytical application frame architecture, we can conclude that technological responsiveness has been good. In a similar way, weak selection of underlying technology could have an adverse impact how well future changes in technology can be made in respect to the analytical application frame architecture. This type of relationship demonstrates the fit between two sub-strategies while changes occur in either sub-strategy. Its corresponding elements might not always impact the fit, and can therefore be managed internally within the sub-strategy itself. If, for example, the software vendor selects execution and development environments that support only specific underlying technologies, the software vendor could potentially select new execution and development environments that support broader selections of underlying technology.

Restricted technology selections, such as selection of a proprietary operating system environment or database management system, can restrict future frame architecture extensions due to technology. In a similar manner, a software development effort could fail in implementing an adaptable analytical application architecture using the best possible technologies. This type of failure could be compared with an analogy in cooking: even the best ingredients can result in a poor meal. Unsuccessful fit between these two sub-strategies can also be a result of a conflict between the domain architects developing the analytical application software solution and the technologists responsible for the underlying technology and execution and development environment.

Another factor that could potentially impact an analytical application software vendor is possible change in the software domain. Changes in software requirements from a functional perspective or changes in general IT infrastructure requirements could cause challenges for the software vendor. This is where technological responsiveness is measured. If the underlying technology needs to be changed or if the underlying technology does not support required software functionality, the software vendor might have to renew its analytical application frame to reflect these new requirements. Another challenge which could become a factor in development of an analytical application software solution occurs when the software domain enters completely new software application areas. An example of this issue within an analytical application software solution is the need to add agent technology with data mining features to the analytical application frame. This type of functionality is not defined as part of analytical application software functionality, and it would therefore challenge an analytical application software vendor to identify the means for implementing this type of functionality. This would challenge the technological responsiveness whereby the fit between these two sub-strategies would be measured. If the software vendor decides to expand its solution base to possible new technologies and/or functional areas (such as agent technology), the complexity of software development will increase and the corresponding analytical application frame architecture could become fragile due to the changes.

4.4.1.2 Functional flexiblity

Changes in market segmentation require flexibility from the analytical application frame architecture to reflect possible new changes in software functionality. This flexibility is defined as "Functional Flexibility." It portrays changes in either an analytical application frame leverage sub-strategy and/or an analytical application frame architecture sub-strategy. Domain-specific software architecture is dependent on the software application domain. In our case, analytical application software architecture requires specific functionality as portrayed in Chapter 3. This functionality is also tied to the current and possible future market segments. The challenge for the software vendor is to balance between the size of the analytical application frame that fulfills each of these market segments and possibly also the analytical application frame extensions (with included vertical market segment functionality) that have been created to support market segments that the analytical application frame does not have.

The success of the development of an analytical application solution for a specific market segment/ group is highly dependent on the software architects within the software vendor firm. Any disconnect between the marketing people defining the market segments the software company is going to address and the software architects implementing the architecture could lead to poor implementation of the application software. This type of miscommunication can also lead to future market segmentation difficulties, as the robustness of the domain-specific architecture is weak and can not be amended with analytical application frame extensions. This type of difficulty is a reflection of weak functional flexibility, which could potentially result in weak analytical application frame efficiency and effectiveness.

The implementation of a selected data warehouse architectural model as the foundation for an analytical application software can have an impact on current and future market segmentation, as some architectural models will not be supported by all market segments. Some analytical application frame architectures will not provide the functionality requirements of the market segment. If the software vendor fails to fulfill these requirements using application frame architecture variation techniques such as modification of business analytics and/or the meta data repository, the solution and the functional flexibility fit might not satisfy the required aims of the software vendor.

Too much emphasis or too broad customer segmentation will cause fragmentation of software development and corresponding problems in the analytical application frame architecture. If the software vendor emphasizes and creates a broad selection of application functionality or application segments, the software development process could become cumbersome due to the combination of customer segments and included technologies.

4.4.1.3 Technological adaptation

The relationship between the analytical application frame technology sub-strategy and the analytical application frame leverage sub-strategy is named "Technology Adaptation," as the selected analytical application frame technology sub-strategy must reflect new possible market segments and/or changes in IT infrastructure technologies in end user organizations. The implication of selecting a wrong technology sub-strategy could lead to a situation in which some market segments can or will not use the analytical application solution due to different IT infrastructure requirements in a selected vertical market segment. This type of conflict is a classic example of a software engineering domain clashing with an information system science domain, as the aims of these two viewpoints are different. Software technologies and software development environments could be very different in their runtime and development time requirements. A good example of a development environment is RPG within the OS/400 operating system environment. The executable programming code for RPG is compatible only with OS/400 environments and can therefore not be run in any hardware environments other than the iSeries/400 environment. If a software company decides to implement solutions using Java, the selection of possible runtime environments is much broader, as the Java runtime environment supports multiple different hardware and operating system environments. Another scenario for conflict between an analytical application frame technology and an analytical application frame leverage sub-strategy could be caused by a software organization that innovates new disruptive technology requiring specific characteristics from the selected market segment.

The balance between selection of an underlying technology and the market segments that the software vendor is going to support must be carefully considered, as too broad a selection in any of these two might cause deterioration of software product development. Too much emphasis on a broad selection of technologies within the analytical application frame using underlying technology could increase the complexity of software development. This could increase the maintenance costs of the analytical application frame(s) and its/their corresponding analytical application frame architecture(s).

4.5 Alignment Perspectives Within an Analytical Application Frame Strategy Framework

While the previous chapter reflected the relationships between two sub-strategies, the relationship between three sub-strategies is regarded as an alignment perspective (see also Venkatraman, 1993). In a manner similar to our analysis of different fits, we will also explore what type of impact any changes in any of the three sub-strategies will have on software development of analytical application software solutions. It is evident from our fit analysis in previous chapters that emphasis on any two sub-strategies within an Analytical Application Frame Strategy framework could potentially lead to weakness in the sub-strategy that has been ignored.

To identify different possible alignment perspectives, we combined each sub-strategy with every other sub-strategy to have a better understanding of how a fit between two sub-strategies could potentially interact with each other. Once this was accomplished, we combined each sub-strategy with a third sub-strategy to see what type of alignment perspective could be identified. Three groupings were identified by taking one sub-strategy at a time for further analysis to see what type of alignment perspective could be possible when the selected sub-strategy was chosen as starting point. Using this inductive approach, we were able to identify two alignment perspectives per selected sub-strategy.

A software vendor that has a strong foothold in technology is more likely to select an alignment perspective in which technology sub-strategy is the driver for software development. In this type of alignment perspective, we were able to identify two main alignment perspectives as follows:

- Implementation of technology in the analytical application frame
- Commercialization of technology innovation

The main difference in these two alignment perspectives is that the first alignment perspective emphasizes the implementation of a strong and adaptable analytical application frame using technology. This type of alignment perspective enables the software organization to use technology as the enabler for effective market segmentation. The second alignment perspective portrays a model in which a software organization uses technology sub-strategy as the driver for market segmentation. This type of market segmentation is a potential approach for innovative start-up companies that have a product idea that they want to bring to the market. The weakness in this alignment perspective is the negligence of analytical application frame architecture and its functionality. This could lead to a software offering that does not address any market segment properly.

If a software vendor has strong domain-specific skills in the solution area, the most likely emphasis for the vendor is an analytical application frame architecture sub-strategy. This type of emphasis can result in two different alignment perspectives, as follows:

- Leverage of an analytical application frame into different market segments
- Implementation of analytical application frame technology into different market segments

In the first alignment perspective, the emphasis is completely on the vertical or horizontal market segmentation. The vendor ignores the importance of technological choices. This could potentially impact future market segmentation and lead to a weak analytical application frame architecture.

In the second alignment perspective, the software vendor aims to build technology within the analytical application software domain. This technology can then be further marketed to specified market segments, such as other Independent Software Vendors (ISV) or end user organizations that have decided to buy analytical application frame components from other software vendors and build the solution themselves. Another option in this type of alignment perspective occurs when the software vendor has decided to renew the application frame architecture with new technology that will address the market segment more efficiently.

The final group has a focus on market segmentation using a top-down approach in which the management of the company selects either an analytical application frame architecture or a technology sub-strategy as a driver. Both of these possible alignment perspectives emphasize selection of a market segment by harnessing a specified market potential:

- Harnessing market potential using an analytical application frame
- Harnessing market potential using technology

When a software vendor leverages its market potential via analytical application frame architecture, the software vendor aims to implement a software solution that reflects the requirements of the selected market segment. In this alignment perspective, the technology is in a weaker position. This type of alignment perspective has potential for organizations that have a strong background in a selected vertical market segment where the management decides to build a solution for that market. The second possible alignment perspective emphasizes a strong technology and is a potential choice when the management of the software vendor has a strong technological background and knowledge in a selected vertical market segment. This alignment perspective gives less emphasis to the analytical application frame architecture and its functionality and includes a risk of creating a solution that is useless for the selected market segment.

In the remaining part of this chapter, we will analyze in more detail characteristics of each possible alignment perspective and the impact of each of these for a software vendor. We argued in prior chapters that emphasis on two sub-strategies can lead to an unbalanced overall strategy with weakness in the sub-strategy that has been ignored. This type of unbalance creates a disconnect which could occur for several different reasons. One reason could be a result of poor communication between different organization groups within the software vendor. Another reason could be the lack of core competence in any of the three sub-strategies. Regardless of the reason for a disconnect, a software vendor must recognize the existence of the disconnect to limit the harm that it can cause.

Each alignment perspective is portrayed using dotted lines between the sub-strategies, demonstrating the direction of the dependency and its characteristics. These dotted lines show unidirectional dependency between three sub-strategies within an Analytical Application Frame Strategy framework. We have named each alignment perspective based on the characteristics of the alignment perspective and the direction of the arrows. The sub-strategy that gets less attention is portrayed with a line pointing toward the sub-strategy that gets less attention within the alignment perspective.

4.5.1 Analytical Application Frame Technology as a Strategic Driver

When an analytical application software vendor selects technology as driver for its development, the software vendor has two alternative alignment perspectives to choose from. The first alignment perspective emphasizes the implementation of a strong analytical application frame architecture that can be easily leveraged to different market segments. The second alternative is to select a market segment with a new innovation that is based on strong technology implementation (see Fig. 17).

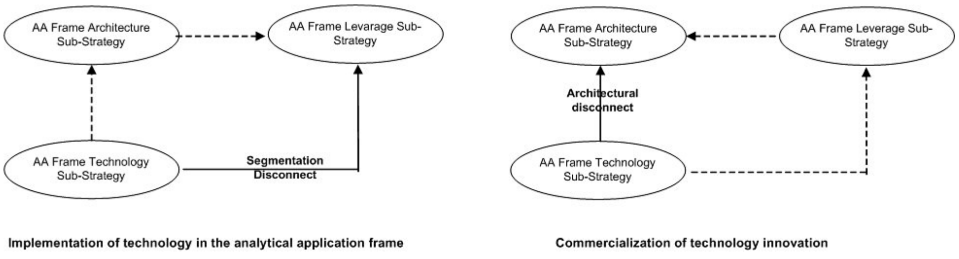

AA Frame Architecture Sub-Strategy
AA Frame Levarage Sub-Strategy
AA Frame Technology Sub-Strategy
Segmentation Disconnect
Implementation of technology in the analytical application frame
AA Frame Architecture Sub-Strategy
AA Frame Leverage Sub-Strategy
Architectural disconnect
AA Frame Technology Sub-Strategy
Commercialization of technology innovation

Fig. 17. Analytical Application Technology Sub-Strategy as a Strategic Driver.

The emphasis on technology reflects each and every layer in an analytical application software vendor's analytical application frame technology sub-strategy with all corresponding elements (underlying technology, execution and development environment, software development environment). Selection of an underlying technology will have an immediate impact on the analytical application frame architecture sub-strategy and its corresponding layers, as any weakness in the underlying technology will impact the adaptability and responsiveness of the analytical application frame architecture and its ability to meet the requirements of other possible new market segments. Besides selecting underlying technologies that are specific for an analytical application software vendor, the software vendor either restricts or enables different choices of the execution and integrated development environment. Some underlying technologies, such as proprietary databases and operating system environments, will not support all execution and development environments. This could have an impact on both the application domain and future segmentation strategies for the software vendor by impeding future market segmentation.

The most profound difference in these two alignment perspectives is the focus on the market segmentation approach, as one alignment perspective is based on building a solution for a specified market segment while the other alignment perspective does not have a true market segmentation strategy: the focus is more or less on technology innovation. By choosing an alignment perspective in which the analytical application frame architecture is the focus of development, the software vendor aims to build a strong analytical application frame architecture using specific technology, with the result being a robust analytical application frame that can be leveraged to different market segments. The other alternative alignment perspective has technology as the foundation for innovation that the software vendor has decided to address. This type of alignment perspective is potentially useful for entrepreneurial organizations with a strong background in selected technologies. These technologies become the foundation for innovation in different market segments. Unfortunately, this type of alignment perspective could result in products that are too technical, with features and functionality that are not needed by end user organizations.

We demonstrated in Table 3 how different technological selections in underlying technology can complicate software development due to the different combinations of underlying technology available in the market place. The software vendor might select a narrow horizontal or vertical market that is know to support specific underlying technology, such as the iSeries/400 hardware and operating system environment. This type of selection can be defensible if the software vendor achieves a competitive edge by becoming expert in the underlying technology or selected market segment. Software vendors should avoid selecting execution and development environments that are not supported by several different underlying technologies, such as operating system and hardware environments.

4.5.1.1 Implementation of technology in the analytical application frame

Selection of underlying technology as the foundation for an analytical application frame architecture must reflect the requirements set by the selected domain-specific architectural model. These requirements, such as the selection of distributed technology, must reflect how end user organizations and selected market segments expect to utilize the solution and what type of functionality is expected from the solution. If the software vendor fails in its selection of underlying technology, resulting in a weak analytical application frame architecture, future market segmentation strategies could be jeopardized. Therefore, selection of both underlying and defining technologies can impact the versatility and adaptability of the analytical application frame architecture. This is measured by how easily the software vendor can move its solutions to new market segments.

If, for example, the software vendor selects a software development language (part of the execution and development environment) which does not support the use of distributed technology, the software vendor would exclude itself from data warehouse models requiring distributed technology. Therefore, technology sub-strategy will clearly limit the choices of architectural models (centralized, distributed, federated) in the analytical application solution domain itself. If the analytical application frame architecture sub-strategy is effective and the integrated application development environment accommodates the selection of different underlying technologies such as operating systems, database management systems, etc., the probability of having less dependence and coupling of analytical application frame components is greater in the selected architectural model.

Different database management systems have their own key innovations and tight coupling to the underlying the hardware and/or operating system environment. Due to these dependencies, selection of some of these underlying technologies could impact the implementation of the analytical application frame components and market segmentation sub-strategy. Some database selections could restrict the software vendor to accessing specific market segments if the underlying database management system technology is known not to be supported in a given market segment. From the software vendor's perspective, database management systems are relatively standardized, but each database environment has its own API, which makes it difficult for software vendors to support several different technologies and combination of technologies. A database management system is just one underlying technology amongst others that must be selected for an analytical application software solution. By using a layered software architectural model, a software development organization can lessen the impact of technological choices if these technologies have effective application programming interfaces based on common standards. These standards can in some cases help software vendors switch between core IT infrastructure technologies and avoid considerable rewriting of the software solution.

The emphasis on this alignment perspective is on technology and application functionality and less on market segmentation. From the software development perspective, this type of alignment perspective could potentially lead to a situation in which the selection of underlying technology is too broad and/or application functionality is too complex. Both of these situations could lead to a weak analytical application frame architecture and a software development environment that is expensive to maintain. Because of this, the definition of the analytical application frame technology and the analytical application frame architecture sub-strategy must be implemented in parallel with selected market segments. This type of alignment perspective with less emphasis on analytical application frame leverage sub-strategy could potentially cause the software vendor "Segmentation disconnect," whereby market segmentation gets the least emphasis of the three sub-strategies within an Analytical Application Frame Strategy framework.

4.5.1.2 Commercialization of technology innovation

The second technology alignment perspective concentrates on commercializing technology within a selected analytical application software domain. In this alignment perspective, the analytical application frame architecture with required functionality will get less attention from the software vendor. This type of alignment perspective could potentially be seen in an entrepreneurial organization with innovative and technology based development within a selected software application or problem domain. This approach is typical bottom-up kind of product innovation, where the software product is innovated and then the software organization selects the market segment when it has the understanding of what the technology innovation can do for the market segment. The software vendor should avoid too broad market segmentation as this could increase the risk of fragmentation of the analytical application frame architecture. We believe that this approach could work if the innovation is either disruptive (Christensen, 1997) or has the capabilities of becoming a "killer application" (Downes and Mui, 1998). The chances for success in this kind of technology innovation is not very high and therefore we would suggest that both established software vendors and newly founded entrepreneurial organizations should carefully consider this alignment perspective due to the high risks. The risks are typically involved with implementing products into the market and running into a situation where the product is right, but market segment or software application domain does not require this kind of solution or even that the product be right, but the pricing (in the price/performance matrix) can not be adjusted according to the expectations of the market (either too low or too high pricing).

This alignment perspective could also serve organizations that have the aim to create and/or invent technology within the analytical application solution domain that other software vendors can use as the foundation (as software assets) in their own software development. These types of inventions could take the software application domain to the next level, something that nobody else has been able to address before. Within analytical application software domain, research organization and market leaders are investing in intelligent agents or mechanisms that enable end users to see relationships in the data and push the results to different end user devices such as PDA's or Internet browsers. These intelligent agents could provide automated "under-the-cover" activity based on business algorithms that are specific to the selected market segment. The commercialization of these types of technologies could reach the early adopters (see also Moore, 1991) in the next few years and reach the majority much later development stages. We believe that once these agents can be prepackaged without having to be customized separately for each organization, they might become more appealing for other analytical application vendors to explore. Similar research is already taking place in predictive modeling of e-commerce sites where organizations want to analyze buying patterns to be able to meet better customer expectations.

Finally, this alignment model emphasizes technology and customer segmentation, but underemphasizes application functionality with respective functional software end user requirements. This type of application solution development is potential for software vendors that are technologists/innovators of technology and can lead to "Architectural disconnect," where the analytical application frame architecture becomes weak due to emphasis on either technology and/or new market segments. These types of software organizations might use radical new underlying technology in their implementation of defining technology, with the risk of not realizing acceptance from IT organizations for the selected market segment. These types of problems can be identified with breakthrough technologies that never reach sufficient maturity or market share for the software vendor to become profitable.

4.5.1.3 Comparison of alignments

We argue that the first alignment perspective with emphasis on implementation of technology in analytical application frame architecture, is more appealing for a software vendor, as this emphasizes the implementation of an adaptable analytical application frame architecture that can be used in different market segments. The other alternative, with emphasis on innovative product development, has more risks associated, such as end user organizations not accepting the new technological innovation that the market does not find useful or practical, or the vertical market segment does not support the selected underlying technology that is the foundation for the software innovation.

Technology as a selling factor might be something that only larger organizations can afford, and therefore the size of the software vendor is also a deciding factor when selecting an alignment perspective with a technological emphasis. Thus, larger organizations could select a business strategy where the aim is to implement software technology for other software vendors within the selected vertical market segment to be used in their own software development. Therefore, we can conclude that organizations using commercialization of technology innovation as their alignment perspective have a business model for providing underlying technology to other software vendors. Obviously, this type of business model is not suitable for smaller organizations with limited cash resources.

Basing the business strategy on commercialization of a technology innovation alignment perspective has associated risks if the software vendor uses radical new technology that is not widely supported by end user organizations or selected market segments. The objective of supporting different underlying technologies could become a major obstacle for a software vendor, specifically if a cross-platform development approach is selected, as software development becomes cumbersome and requires deep core competence from the development organization. Due to these IT infrastructure requirements, development of an analytical application frame and its associated components for other software development organizations could become a challenge for a software vendor, as these analytical application frame components have to support several underlying technologies such as operating system and hardware environments, database management systems, etc. A good example of this would be innovation in analytical application software development using a specific proprietary hardware technology as a foundation. If this approach is selected, the software vendor might be restricted to being able to market its solution to current or future market segments that are known for not supporting these IT infrastructure environments. Therefore, decisions to build an entire product family based on specific technology must be carefully considered and the software vendor should also have alternative strategies if the selected development or market segmentation strategy fails.

We also identified two different disconnects with these two alignment perspectives. Both of these could have an adverse impact on the software vendor. We argue that segmentation disconnect will cause less harm than architectural disconnect, as the previous alignment perspective has the aim of an adaptable frame architecture using a strong technology strategy. Architectural disconnect can lead to a weak analytical application frame resulting in increased software maintenance and development costs.

4.5.2 Analytical Application Frame Architecture as a Strategic Driver

When a software vendor selects application frame architecture as its strategic driver for software product development, the selected analytical application frame architecture can be implemented either by leveraging the analytical application frame architecture into different market segments or by implementing analytical application frame technology within a selected software application domain (see Fig. 18).

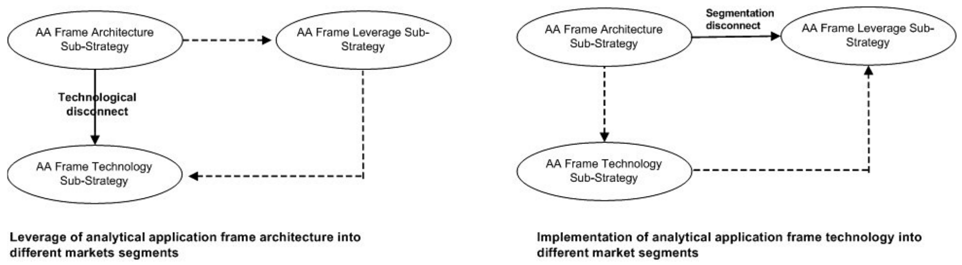

AA Frame Architecture Sub-Strategy
AA Frame Leverage Sub-Strategy
Technological disconnect
AA Frame Technology Sub-Strategy
Leverage of analytical application frame architecture into different markets segments
AA Frame Architecture Sub-Strategy
Segmentation disconnect
AA Frame Leverage Sub-Strategy
AA Frame Technology Sub-Strategy
Implementation of analytical application frame technology into different market segments

Fig. 18. Analytical Application Frame Architecture Sub-Strategy as a Strategic driver.

Both of these models represent a business model in which an analytical application frame architecture is the foundation for solution development. The main difference in these two alignment perspectives is that one perspective is heavily technology oriented while the other is more or less driven by market segmentation. The analytical application software domain with the aim of identifying an optimal analytical application frame construct poses considerable challenges for a software vendor. First of all, an analytical application solution includes a variety of different functional components that must be aligned with selected underlying technology and a selected data warehouse architectural model. Secondly, an analytical application frame architecture with its functional components must be well aligned with a selected domain-specific architectural model and technical architectural style (client/server, distributed, pipe-and-filter etc.). It is imperative for a software vendor to separate these two different architectural models from each other, as each of them poses different requirements. The analytical application frame architecture will include requirements in the software application domain, while the technological architectural style will must consider other factors such as end user organization IT infrastructure and the execution and development environment for the solution itself.

4.5.2.1 Leverage of an analytical application frame into different market segments

Selection of the model in which an analytical application frame architecture is mainly built for selected market segments could restrict the company's expansion into future possible market segments. A software vendor might run the risk of building an analytical application frame that does not allow any future market segmentations, as the technological selections were wrong in the first place, for example, selecting a technology that never takes off in the general information technology market. Because of this, analytical application software vendors should review selected technology in concert with the analytical application frame architecture to make sure that selected domain-specific software architecture is amendable and adaptable to future market segmentation strategies. Underlying technology selections have to support the software vendor in defining robust technology components as part of the analytical application frame architecture (meta data repository, business analytics repository, etc.). In each alignment perspective alternative, the third sub-strategy must be reviewed together with the alignment perspective itself, as it could have a profound impact on future market segmentation strategies. One alignment perspective could work better in one organization than another due to different core competences and backgrounds in software development.

Analytical application software functionality with corresponding vertical market segment specific key metrics and business processes is the main driver in this alignment perspective. Because of this, business analytics and meta data repositories must be adaptable to amendment by both software vendors and end user organizations. This adaptability is realized by effective selection of the underlying technology as a basis for defining technology and its corresponding elements. If the technological foundation is weak, the result is most probably a weak foundation for the analytical application frame architecture. This leads to ineffective future market segmentation possibilities. Key performance indicators and related business processes must be included in the solution, with the ability to change these once the solution has been installed in the end user organization IT infrastructure. These changes are enabled if the solution supports parallel development of the analytical application solution whereby the software vendor can deliver new releases of the solution without overwriting the possible changes that the end user organization has implemented in the solution.

The adaptability and ability to amend the analytical application frame architecture will be based on how well the underlying technology and its analytical application frame components are assembled and designed to accommodate new vertical or functional market requirements. The success of an analytical application frame architecture and its ability to serve different market segments can be measured by economic metrics as described in Chapter 2. A successful analytical application frame is the foundation for future derivative products that the software vendor might not know in advance. The effectiveness and efficiency can only be measured ex-post after the product line/family has been implemented.

Our analytical application frame architecture showed the importance of application programming interfaces that might also give additional leverage to an analytical application software vendor, as these interfaces can be amended for new functionality or even with new analytical application frame components that provide additional functionality for the selected market segments. The analytical application frame architecture is the basis for the ease with which the software vendor can control the price/performance tiers and whether or not the software vendor must replace or amend some analytical application frame components to be able to reach the intended functionality of a selected market segment as portrayed in the analytical application frame architecture model in this chapter.

Finally, this type of alignment perspective could lead to fragmented product development if the software vendor selects too many customer segments to be supported. Each of these vertical market segments could potentially have very different functional requirements (both software functionality and solution content). This could lead to a weakened analytical application frame architecture and increased difficulty in software development. The software vendor might in some situations run into "Technological disconnect," which describes potential weakness in analytical application frame technology sub-strategy.

4.5.2.2 Implementation of analytical application frame technology into market segments

The second possible alignment perspective when using analytical application frame architecture as a basis is when an analytical application vendor emphasizes the alignment perspective between the analytical application frame architecture sub-strategy and the application frame technology sub-strategy. This was defined as "Implementation of application frame technology into different market segments." This type of alignment perspective is less likely to occur for most software vendors, as the emphasis in this alignment perspective is to either create an analytical application frame that becomes part of the technology sub-strategy or the analytical application frame(s) are used to create other derivative products for different market segments. The other alternative is that the software vendor builds analytical application frame components that will be resold to other Independent Software Vendors (ISVs). In both of these alternatives, market segmentation is de-emphasized. This could become a major problem in future market segmentation strategies.

If the software vendor selects this type of alignment perspective and decides to become a software component provider to other software vendors, several other new challenges will became apparent for the software vendor. Configuration management for development, sublicensing policies, and upgrade automation of new releases are major tasks for any software vendor. The quality of these processes must be high, as other vendors might use these as part of their own software deliverables. Another factor that the software vendor must consider when using this alignment perspective is that of ensuring proper selection of underlying technologies, as too narrow selection could restrict the vendor to a narrow market segment which does not provide the needed leverage for the software vendor. With broad technological selections and broad customer market segments, the software vendor could run into a nightmare of compatibility problems in different releases of analytical application frame components with corresponding technologies. This analysis is unfortunately something that software vendors could easily forget, as the dependency in market segmentation is closely tied with technology selections and the corresponding IT infrastructure that end user organizations support.

If the software vendor's business model is to deliver analytical application frame components to other independent software vendors, these frame components become part of other third-party solutions. We strongly argue that this kind of alignment perspective is not suitable for smaller companies with limited personnel and financial resources. We also argue that this kind of alignment perspective is obviously a real threat to all independent analytical application software vendors, as by competing within their selected application domain with aggressive pricing, larger software vendors could impact and destroy the markets for these smaller vendors. This type of alignment perspective is more suited to larger software organizations with development teams that have software development practices that support development of large scale and high quality software. Some database management vendors have included analytical application frame components in their database management software offering, and therefore these components and their classification changes from defining technology to underlying technology that is part of analytical application frame technology sub-strategy. The emphasis on technology without clear market segmentation could potentially lead to "Segmentation disconnect," as the software vendor is highly technology driven, with the emphasis on moving defining technology to become part of underlying technology for other software vendors. This could potentially cause problems in market segmentation, as the requirements of the software assets could change between different market segments and different application areas.

The aim of a software vendor is to optimize the usage of an analytical application frame in different market segments, but this could lead to too broad a selection of underlying technologies to support different end user IT infrastructure environments. With broad technological selections and broad customer market segments, the software vendor could run into a nightmare of compatibility problems in different releases of analytical application frames and technological choices. This analysis is unfortunately something that software vendors could easily forget, as the dependency in market segmentation is closely tied with technology selections and the corresponding IT infrastructure that end user organizations support. Therefore, a software vendor must be aware of any changes in underlying or defining technologies when selecting possible new market segments or application areas.

In some cases, implementation of analytical application frame technology can be used to extend the original analytical application frame with new architectural designs that give the software vendor the possibility of moving on the price/performance matrix using an analytical application frame leverage sub-strategy. An example of this could be when a software vendor extends the existing analytical application frame with technology that extends the solution to new market price/performance tiers. If, for example, an analytical application vendor requires analytical application information to be distributed to any information device, it could build a delivery mechanism that does not exist anywhere else. That system will become a part of the technology sub-strategy.

4.5.2.3 Comparison of alignments

When comparing these two different alignment perspectives with each other, we can conclude that the first perspective underestimates the importance of technology selections while the second uses technology as major part of its development. The second approach differs greatly from the first one by having a completely different approach, as technology becomes a part of the overall development strategy for the software development organization. The weakness of this approach is that the software solution with its analytical application frame and corresponding analytical application frame components might not reflect the requirements of the selected market segment because the technology might have become the focus and major driver for the whole solution. This occurrence could potentially occur within smaller- and mid-sized entrepreneurial software organizations that have management with a strong technological background, aiming to create technology for the selected software application domain, or where the software vendor's business model is to provide software development services for another software vendor in the analytical applications software market. Unfortunately, these software organizations might end up building software for early innovators that are only a small group of the overall opportunity pool.

When an analytical application software vendor emphasizes analytical application frame functionality above technology or market leverage, the vendor either does not have the skills to address any specific vertical market or does not command any specific technological core competence that could drive the product development. Software vendors emphasizing product functionality in selected application areas such as analytical application development might emphasize functionality above technology innovation, and therefore technology could play a secondary role in these software organizations. When software vendors develop analytical application frame components, design and implementation plans should take into account possible future vertical segmentation strategies. Careless planning can cause problems in future integration efforts between analytical application frame and analytical application frame components for extended functionality. Product development with strong emphasis on an analytical application frame sub-strategy is also typically driven by functionality requirements on the analytical application frame components and not by technological selections. Obviously, the functionality decision has to be in sync with the technological selections if the company wants to build a successful analytical application frame sub-strategy that can be extended to different vertical markets.

In these two different alignment perspectives, the perspective with emphasis on technology is less risky, including technological choices that will not carry the solution to new market segments, compared with the other alignment, where market leverage is given more focus than the technology itself. A pure comparison between these two alignment perspectives leaves us to believe that the alignment perspective with emphasis on leveraging an analytical application frame to different markets is more favorable when compared with the alignment perspective having a technology focus and development of analytical application frame components for selected market segments. The reason for this is that we believe the latter perspective is highly unlikely for typical analytical application software vendors and more likely for large software organizations with ambitions plans to implement technologies that can be embedded in other software solutions.

Finally, both alignment perspectives could run into disconnects when applying a specific alignment perspective. Both disconnects are serious if a software vendor does not take them to consideration, but the disconnect in the alignment perspective “Leverage of analytical application frame into different market segments” could become a major issues if technological selections are weak, with a result of a weak and fragile analytical application frame architecture.

4.5.3 Analytical Application Frame Leverage as a Strategic Driver

When an analytical application software vendor selects application frame leverage as the driver for its development, the selected application frame leverage can be achieved either by harnessing market potential using a specific technology strategy or alternatively harnessing market potential by building a strong application frame (see Fig. 19).

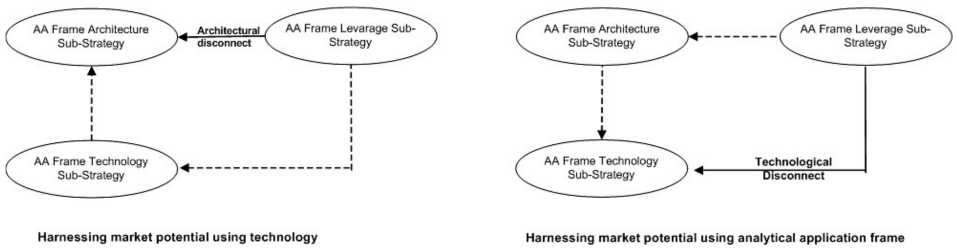

AA Frame Architecture Sub-Strategy
Architectural disconnect
AA Frame Levarage Sub-Strategy
AA Frame Technology Sub-Strategy
Harnessing market potential using technology
AA Frame Architecture Sub-Strategy
AA Frame Leverage Sub-Strategy
AA Frame Technology Sub-Strategy
Technological Disconnect
Harnessing market potential using analytical application frame

Fig. 19. Analytical Application Frame Leverage Sub-Strategy as a Strategic Driver.

Regardless of the selected strategy, the underlying analytical application frame architecture must be adaptable and amendable to enable effective analytical application frame leverage to new market segments. Part of the defining technology for an analytical application software vendor is its meta data and business analytics repository, the foundation for storing key business metrics and meta data information about the corresponding ETL processes. We also concluded that an analytical application software vendor should aim to create "an optimal architectural construct" that can be reused across different market segments. Therefore, the challenge for a software vendor is to create an analytical application frame that is as generic as possible. Only the contents of the analytical application frame repositories will be different at the time of delivery of the final product. This will require stability in the database structures of the application. The differences in each market segment are satisfied by loading vertical market metrics on delivery to the end user organization.

The aim of the software vendor is to identify the functionality of selected analytical application frame components that fulfill most of the requirements for each vertical market, to add market segment specific functionality and also to identify business metrics and processes as part of the packaging of the solution or part of the extended functionality that can be integrated into the analytical application frame at runtime. Sometimes it could be most effective to include functionality in the analytical application frame that some of the vertical market segments do not require. This is typically more effective due to decreased software configuration and version management activity, limiting the number of analytical application frames to the minimum. We will further discuss technical implementation and variation of analytical applications in Chapter 5.

4.5.3.1 Harnessing market potential using technology strategy

The alignment perspective wherein the software vendor selects a market segmentation sub-strategy based on selected technology is based on a top-down planning activity, where the software vendor selects a set of underlying technologies for analytical application software development that will impact not only the market leverage, but also analytical application frame architecture sub-strategy. Our discussion of the analytical application frame technology sub-strategy demonstrated the complexities and interdependencies between different technological selections in solution development and how these selections could become the key element of the success or failure of the software company. Technology based strategies will support the implementation of a strong analytical application frame architecture for the software solution from a technology perspective, but they do not necessarily provide the keys for successful analytical application frame architecture functionality.

Selections in analytical application frame technology include also selection of both the end user IT infrastructure support (runtime environment) and the development environment (underlying technology). Each selection will be a combination of different factors that will impact the overall adaptability of the software solution, the analytical application frame, and its frame architecture. These two types of environments could be different when developing software for end user organizations, but by using effective layered software architecture and by selecting an effective integrated development environment, a software vendor can minimize these dependencies. Integrated development environments are typically based on application generator technologies, with predefined class libraries and/or component frameworks that can be reused in software development. These development environments in some cases have their own proprietary application development language that is compiled to the appropriate target hardware and operating system environment at deployment time. Therefore, we can easily conclude that technological selections specifically within software development will impact every alignment perspective and will therefore become almost a centrepiece of any software development initiative. Emphasis on both market segmentation and application frame technology could potentially lead to "Architectural disconnect," as the software vendor might undervalue the need to build strong architectural bases for the analytical application frame.

Finally, selection of the analytical application frame technology strategy must be evaluated from the perspective of the end user organization and the software development organization. If the selections are performed carelessly, future market segmentation could become very difficult and in some cases impossible. Some end user organizations within a selected market segment could have an IT infrastructure that is very different from the environment that the software organization can support, and this is something that is and has been very difficult for software organizations. Even today, the software development community is divided between the Microsoft camp with its .NET initiative and more open environments, where software can be run on multiple hardware and operating system environments. This type of development enables software organizations to build solutions that can be adjusted to each market situation more easily than building the overall software solution on one technology that is very hard to move to any other environments. An example of a limited technology selection is the selection of the iSeries/400 customer segment, with its corresponding underlying technology selections. These selections could potentially lead to an analytical application frame architecture that will not reflect the requirements of other market segments in the future.

4.5.3.2 Harnessing market potential using an analytical application frame

The alternative to a technology driven alignment perspective is to emphasize the strength of an analytical application frame architecture by selecting a market segment that the software organization will focus on. This approach has potential for organizations with management that has either prior skills or knowledge within a chosen vertical market segment or where the management of the organization has identified a market niche that they want to address with an analytical solution and solid value proposition. In this alignment perspective the technology strategy becomes secondary. This could lead to weak underlying technology and an analytical application architecture that is neither easy to change nor easy to amend with new features due to an inflexible analytical application frame and its components. Therefore, the software could potentially run into "Technological disconnect," as technology does not play strongly in the alignment perspective and the management and development team sets application functionality for a selected market segment as a top priority.

Another consideration when selecting analytical application frame architecture as a driver in development is to evaluate and pay close attention to the selected market segment and what hardware/software environments this market segment is known to support. A pricing structure with corresponding price/performance drivers is important to consider when development a balanced development strategy for analytical application solutions. The approach in this alignment perspective is to specify the functional application areas that the software vendor is going to play in and define the contents of both the meta data and business analytics repositories with all associated data integration rules for different operational applications. The software vendor might need to create additional add-in modules within the extension component layer that are integrated into the analytical application frame at the time of delivery. This type of additional functionality is needed when the functionality of one market segment differs considerably from that of another vertical market segment. Optimization is achieved by separation from the core analytical application frame architecture.

4.5.3.3 Comparison of alignments

Both of these perspectives are initiated from selected market segment(s). The first perspective has technology as its foundation, while the other uses application frame architecture as its driving factor. The technology driven alignment perspective could result in a product suite that is either too complex to use or is technically too advanced for regular end user organizations to maintain and manage. Software organizations with a clear vision of a specified market segment and a strong belief in a selected underlying technology could lead into a solution that becomes either obsolete or is well ahead of its time. There are several examples of this type of technological experimentation in the past few years. For example, IBM OS/2 was selected by several vendors as their main client environment, but the market decided that OS/2 was not the one that would win. Software organizations with OS/2-based products had to retreat and either convert or perish from the marketplace.

In the other alignment perspective where software application functionality is the driving force, software organizations might forget to align the selected defining technology to be in sync with the underlying technology that is part of the analytical application frame technology sub-strategy. Software vendors that have strong domain knowledge in the selected market segment and software application domain could be tempted to not address the importance of a solid analytical application frame technology. This could lead to a weak analytical application frame architecture sub-strategy and make future market segmentation difficult. An example of this could be selection of an operating system and a database management system (part of the underlying technology) that will not scale or support specific market segments or is strictly bound to a selected IT hardware and operating system environment (such as the iSeries/400). These types of selections are very difficult to change later in the development cycle, as they employ proprietary technologies that are not supported by other IT environments.

We argue that both alignment perspectives could be effective when used carefully. The most effective perspective would be where both an analytical application frame architecture sub-strategy and an application frame technology sub-strategy are integrated and in balance with each other when selecting the market segmentation sub-strategy. Both alignment perspectives foster different disconnects. Based on our prior analysis, architectural disconnect is more serious, as the foundation for the analytical application frame could become weak.

4.5.4 Disconnects in Alignment Perspectives

Each alignment perspective includes a possibility for a disconnect that could develop if two sub-strategies within an Analytical Application Frame Strategy framework are emphasized and the third sub-strategy receives less attention. The resulting weak alignment perspective will also result in a weak fit. The question remains whether the weakness is between two sub-strategies or all three selected sub-strategies. We identified six different alignment perspectives and three different types of disconnects that could potentially impair the software vendor engaged in building software solutions. According to our analysis, a disconnect could develop in a software organization if two sub-strategies get more emphasis. The third could either be neglected or possibly impact future product development activities because the software organization did not realize the impact of the decision. We also concluded earlier in this chapter that an optimal alignment perspective could be different for different software vendors based on their background, such as core competence in the software application domain and technology. The idea behind exploring a disconnect is to make software vendors aware of them so as to be able to recognize the possibility of a disconnect when setting their strategies. Based on the six alignment perspectives that were identified in this chapter, we were able to identify three potential disconnects that could develop with unbalanced alignment perspectives:

- Architectural disconnect
- Segmentation disconnect
- Technological disconnect

The common denominator for each disconnect is that they represent weakness in one of the sub-strategies due to less emphasis when compared with the other two sub-strategies. Figure 20 portrays the six different alignment models grouped by each of the three disconnects (see Fig. 20).

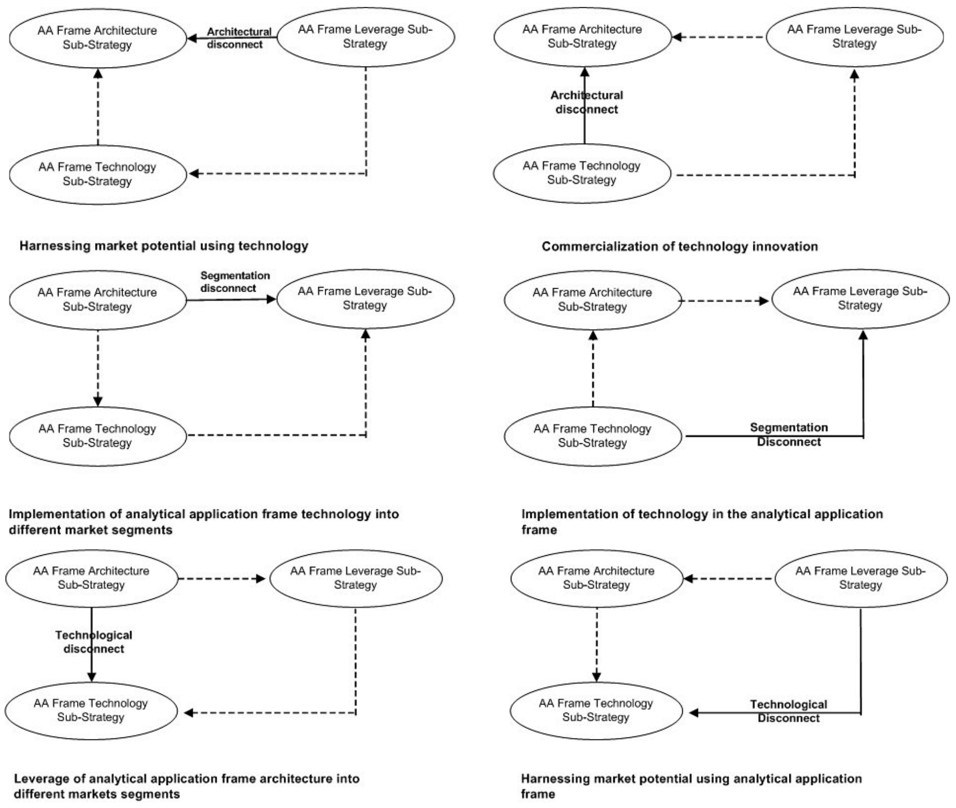

AA Frame Architecture Sub-Strategy
Architectural disconnect
AA Frame Leverage Sub-Strategy
AA Frame Technology Sub-Strategy
Harnessing market potential using technology
AA Frame Architecture Sub-Strategy
AA Frame Leverage Sub-Strategy
Architectural disconnect
AA Frame Technology Sub-Strategy
Commercialization of technology innovation
AA Frame Architecture Sub-Strategy
Segmentation disconnect
AA Frame Leverage Sub-Strategy
AA Frame Technology Sub-Strategy
Implementation of analytical application frame technology into different market segments
AA Frame Architecture Sub-Strategy
AA Frame Leverage Sub-Strategy
AA Frame Technology Sub-Strategy
Segmentation Disconnect
Implementation of technology in the analytical application frame
AA Frame Architecture Sub-Strategy
AA Frame Leverage Sub-Strategy
Technological disconnect
AA Frame Technology Sub-Strategy
Leverage of analytical application frame architecture into different markets segments
AA Frame Architecture Sub-Strategy
AA Frame Leverage Sub-Strategy
AA Frame Technology Sub-Strategy
Technological Disconnect
Harnessing market potential using analytical application frame

Fig. 20. Alignment Perspectives grouped by Different Disconnects.

The two uppermost alignment perspectives portray a situation in which a software vendor de-emphasizes the analytical application frame architecture and emphasizes either the analytical application frame technology sub-strategy and/or the analytical application frame leverage sub-strategy. The two alignment perspectives in the middle section de-emphasize the analytical application frame leverage sub-strategy and emphasize either the analytical application frame technology sub-strategy and/or the analytical application frame architecture sub-strategy. The bottom alignment perspectives in Figure 20 put the analytical application frame technology sub-strategy into an inferior role, while the analytical application frame leverage and/or application frame architecture sub-strategy receives more attention.

4.5.4.1 Architectural disconnect

The first pair of alignment models is represented by "Harnessing market potential using technology" and "Commercialization of technology innovation." Both of the alignment perspectives have the possibility of "Architectural disconnect." The main difference in these two perspectives is that the direction of the first perspective is from an analytical application frame leverage sub-strategy toward an analytical application frame technology sub-strategy. This could potentially lead to a weakened analytical application frame architecture. The main reason for a weakening architecture could be both the selection of too many market segments that the analytical application frame must support or that the software vendor selects too many underlying technologies to build the analytical application frame architecture.

The second alignment has a strong emphasis on a selected market segment using technology. Less emphasis is given to an analytical application frame architecture. This disconnect is specifically risky, as selection of an underlying technology might lead to an analytical application frame that is either too complex to maintain or whose technology has not reached the maturity that most of the end user organizations will be able to accept as part of their solutions.

4.5.4.2 Segmentation disconnect

The second pair of alignment models is represented by "Implementation of analytical application frame technology into different market segments" and "Implementation of technology in the analytical application frame." Both of these alignment perspectives could potentially lead to "Segmentation disconnect," as the analytical application frame leverage sub-strategy is de-emphasized when building an Analytical Application Frame Strategy framework for the software vendor. The main difference in these two alignment perspectives is that in the first perspective, the direction is from the analytical application frame architecture sub-strategy to the analytical application frame technology sub-strategy, while in the second perspective, the direction is from the analytical application frame technology sub-strategy toward the analytical application frame architecture sub-strategy. Both of these perspectives de-emphasize market segmentation of the solution and could potentially lead to analytical application frame and technological selections that do not support future market segmentations, as too much emphasis is put on the other two sub-strategies. The software vendor could potentially select underlying technologies that are known by the development team (part of core competence), but these might not be technologies that are widely accepted by the end user organizations. Therefore the software vendor could face a conflict between the supported IS infrastructure in end user organizations and the infrastructure that is supported by the software solution.

4.5.4.3 Technological disconnect

The last group of alignment perspectives is "Leverage of the analytical application frame architecture into different market segments" and "Harnessing market potential using an analytical application frame." Both of these alignments could possible lead to "Technological disconnect" because of less emphasis on an analytical application frame technology sub-strategy. The difference between these two alignment perspectives is mainly the direction of alignment perspective. The first perspective takes an analytical application frame architecture sub-strategy as the basis when leveraging the analytical application frame to different market segments, while the second alignment perspective takes an analytical application frame leverage sub-strategy as the basis of an analytical application frame architecture sub-strategy. Both of these could potentially lead to a situation in which a future market segmentation strategy could become impaired, either due to the weak technological foundation of the analytical application frame (because of too broad market segmentation) or because the selection of underlying technology is weak, providing a weak foundation for the analytical application frame architecture.

4.6 Chapter Summary

Our analysis of a layered architectural model with emphasis on information distribution and functionality showed how each functional analytical application frame component interacts with each other functional analytical application frame component in an analytical solution. We compared three different data warehouse architectural models and concluded that a hybrid data warehouse model with federated and multi-tier/distributed characteristics provides the most potential adaptability for an analytical application software vendor to accommodate different vertical market segments in its software development. Based on this analysis, we decided to build an analytical application frame architecture that portrays different architectural layers and analytical application frame components.

Our aim was to define an optimal architectural construct for an analytical application solution that could be used across different vertical market segments. To be able to achieve this goal, we identified all of the needed functionality for an analytical application solution via a literature study and placed these functional components into the selected architectural model portrayed in this chapter (the hybrid model). The undermost layer consists of data staging area with inclusive extraction, transformation, and load mechanisms to operational applications.

The second layer in the architectural construct consist of meta data and business analytics repository and data warehouse and data mart database structures that are the foundation for an analytical application solution. Another importance part of an analytical application frame is a generic business and analytics engine that provides the logic for the service component layer and acts as the interface for underlying database structures. This generic business and analytics engine can be used across any vertical or horizontal market segment. The service component layer provides different types of services for the overall analytical application solution such as presentation and distribution of the analytical information for end users. This layer provides the foundation for future devices that might be supported, such as PDAs, cellular phones, etc.

The final layer – the extension component layer – provides the ability for a software vendor or third-party software vendor to create add-ons to the analytical application frame so that the software organization does not need to change the functionality of the frame itself, but concentrate on building additional functionality to the extensions.

We could also see similarities between information products from Meyer and Zack (1996) and analytical applications, as both include characteristics of a process platform where the contents of the database will distinguish one vertical market segment from the other. The idea behind an analytical application solution for a vertical market segment is to have key metrics and predefined reports that are specific to the market segment and cover 80% of the requirements that end user organizations have for the market segment. Myer and Zack (ibid) defined a process platform as the refinery that enabled a data service company to provide different contents to different customers. This process would need to be managed by the analytical application software vendor using a domain specialist who understands the domain and applies this knowledge to the meta data and business analytics repository. This type of derivative product development using business analytics and a meta data repository will be explained in more detail in Chapter 5.

An analytical application software vendor has basically three different ways of creating derivative products within analytical application software domain. The first is to provide database contents for different vertical market segments using the analytical application frame. The second and more difficult method is to use software engineering related technologies that are described within the software product line engineering literature. The third and final variation technique relates back to market segmentation strategy, where a software company uses price/performance as its market segmentation strategy.

To be able to build an analytical application frame architecture sub-strategy, we needed further analysis of an analytical application frame's underlying technology. This enables an analytical application software vendor to be able to get an understanding of how each technology selection might impact the overall software solution. An analytical application frame technology sub-strategy consists of underlying technology that provides core technological elements for an analytical application software vendor, such as an operating system environment, a database management system, and other IT infrastructure elements that the software development organization expects to have access to in its development. We also identified two other important factors belonging to an analytical application frame technology sub-strategy. First of all, selection of an execution and development environment must be in concert with underlying technology, whereby any selection of underlying technology must be supported by the integrated development environment. Secondly, the software vendor must select a software development approach. One potential development approach for an analytical application software vendor is software product line engineering using the component-based development approach. It was evident that software vendors have to separate underlying technology from defining technology and that these two technologies must be well balanced to achieve a flexible application frame that can be used in different market segments.

Based on the findings from chapters two and three, we concluded that existing software product platform related literature needs to be amended to reflect the complexity of software development and also to reflect the needs of an analytical application software vendor. This analysis led to an Analytical Application Frame Strategy framework that reflects the requirements of an analytical application software vendor and was built using our analysis of the analytical application frame architecture, the analytical application frame leverage, and the analytical application frame technology sub-strategies explored in this chapter. We also identified six different possible alignment perspectives that a software vendor can utilize. We also analyzed the relationship between each and every sub-strategy, and we named each of the resulting fits with corresponding names (Functional Flexibility, Technological Responsiveness, and Technological Adaptation). We also showed that each alignment perspective could potentially lead to three different disconnects (architectural, segmentation or technological disconnect). These disconnects might not be avoided in real-life settings, but the awareness of these could help a software vendor to minimize their potential negative impacts.

In our analysis of different possible alignment perspectives, it was evident that technology is a dominant driver in any alignment perspective, as both underlying technology within the analytical application frame technology sub-strategy and defining technology within the analytical application frame architecture sub-strategy will impact current and future market segmentation strategies. Therefore, we argue that software development is still very dependent on successful technology selections, and that this is something that software vendors must recognize continually. Software development is closely tied with underlying IT infrastructure, both from the end user organizations and the development perspectives, and these dependences make software development very dependent on different technologies. We also argue that continuous re-adaptation of alignment perspectives will occur in software organizations depending on the current market situation, core competence, and the selected software market segment. It is also obvious that each software domain must be analyzed separately, as the presumption of an underlying IT infrastructure environment that must be supported is very different between software vendors. An example of this occurs when comparing mass-market software products with word processing and business intelligence or analytical applications that can be very complex and include several interweaved technologies.

Based on our analysis in this chapter, we can conclude that one alignment perspective compared with another could work differently with different software vendors, and therefore it is highly unlikely that an optimized alignment perspective would work exactly the same for any selected analytical application software vendor. The aim for an analytical application software vendor should therefore be to be aware of these different alignment perspectives and the impact of each perspective on its business. Each alignment perspective with unilateral emphasis on any of the three sub-strategies might lead to an imbalanced product development strategy and inhibit future market segmentation strategies for the software vendor. Therefore we suggest that any analytical application software vendor should aim to achieve a balanced and/or integrated strategy between the selected analytical application frame architecture, the analytical application frame leverage, and the analytical application frame technology sub-strategy by evaluating each alignment perspective with its pros and cons. In a balanced/ integrated strategy, the overall corporate strategy is taken into consideration, as executive management sets the foundation for market segmentation and derivative product development using an analytical application frame.

Finally, the next step in our analysis is to include discussion of software implementation approaches for an analytical application frame within the software development community. So far, our analysis has been more or less theoretical. The aim of next chapter is to concretize how an analytical application frame can be implemented and what known implementation techniques are already available in the software development literature to support analytical application frame development. There is extensive research within product line engineering and component-based software development, and the aim of the next chapter is to explore how these research results could be used to implement software products within the analytical application software domain. It is very important to understand that while underlying implementation technologies come and go, Analytical Application Frame Strategy theory will still remain the same; the only difference might be that software application frames can be reused more easily and therefore provide greater platform leverage. We have now reviewed and answered our first three research questions. The last and remaining research question is as follows:

RQ 4: How can effectiveness criteria change when applying the product platform approach in analytical application software development?

The fourth research question requires us to explore different implementation approaches for analytical application software solutions with corresponding variation techniques. Existing product platform related literature does not provide guidance as to how and what type of software engineering approach could be taken when designing and implementing software application frames. Our analysis will reflect theories that are close to the concept of the product platform with the aim of large-scale reuse. Using selected effectiveness and efficiency metrics, software vendors are able to measure whether software application frame development has been accomplished according to the aims set for the product development. This will be discussed in more detail in the case study example introduced in Chapter 7.

5. IMPLEMENTATION APPROACHES FOR A SOFTWARE APPLICATION FRAME

The previous chapter demonstrated several different strategic alignment perspectives that a software vendor could potentially face when implementing analytical application software solutions. Each alternative could be different for each software vendor, and therefore it is important to understand that none of these alternatives provides the "ultimate truth," as each vendor has a different background with different core competences. The aim of these strategic alignment perspectives is to provide an overview of the impact for an analytical application software vendor when balancing between market segmentation, software product architecture, and technological selections. Each alignment perspective is different with a different outcome. Each software vendor has to find an optimal solution based on the specific characteristics of the software vendor.

The aim of this chapter is to provide operational and practical information for analytical application software vendors in the implementation of an analytical application frame with its corresponding analytical application frame software components. Each sub-chapter reflects the Analytical Application Frame Strategy framework from Chapter 4. Thereby we will initiate our discussion with software architectural related issues such as domain-specific software architecture and the importance of creating a modular and layered architectural model for the analytical application software solution. The second major topic in this chapter is to portray in more detail technological selections that a software vendor must undertake when selecting an underlying technology, execution, and development environment. Part of an analytical application technology sub-strategy is selection of a software development approach. In our discussion in Chapter 2, we concluded that software product line engineering with corresponding component-based software engineering is a viable contributor to development of analytical application frames and corresponding analytical application frame components.

We will use the terms "software application frame" and "analytical application frame" interchangeably depending on the instance. The former definition is a generic term for any software application domain, while the latter definition conforms to an analytical application software domain.

Finally, as part of derivative software development, an analytical application frame and its success is measured using product platform effectiveness and efficiency measures defined by Meyer et al. (1997). These measures describe how well a software vendor has been able to create derivative products from an analytical application frame using different variation techniques. We will discuss three possible variation techniques that a software vendor can utilize in its derivative software development. These variation techniques are divided by the three sub-strategies defined within an Analytical Application Frame Strategy framework.

5.1 Introduction

Prior chapters focus more or less on strategic alternatives that an analytical application software vendor can undertake in development of analytical application software solutions, while the aim of this chapter is to provide operational directions and ideas that a software vendor can take when building an analytical application frame with its corresponding product family. Existing software related product platform literature concluded the possibility of component-based software engineering as the foundation for product platform development, but these references did not provide operational information as to how and what type of questions a software vendor might face when implementing analytical application software solutions. A common theme closely related to software related product platforms is the software product line engineering approach, which has been introduced as a viable approach for implementing software products within a product line/family. This approach supports the aim of leveraging an analytical application frame in the form of derivative software products from a common analytical application frame. We defined analytical application frame components as the functional elements that are needed to build an analytical application frame. Each of these functional components consists of a set of software assets that are built internally by the software vendor, outsourced to third-party software development organizations, or purchased in the form of COTS software components.

To shed some light in how market segmentation strategy could be implemented using an Analytical Application Frame Strategy framework incorporating different software engineering related technologies, we decided to research current product line engineering literature to identify technical means for derivative software development. This discussion is extended with the variation options that an analytical application software vendor has when using the analytical application frame itself. We will also discuss the importance of software architectures and corresponding product line architectures when implementing application frame(s) and what type of impact they have on the flexibility of different variation techniques – both solution variation and technological variation.

The software vendor has several different avenues for derivative product development. Some of these are based on analytical application solution variation using application frame architecture as the basis for the variation. Another variation technique is based on software engineering related variation. This is described in both the software engineering and the software product line literature. The third possible variation technique is based on analytical application frame leverage strategy, whereby a software vendor can use pricing and functionality thresholds to market its solution. This can be controlled by control numbers within the solution itself.

Because of these different variation techniques, an analytical application software vendor must distinguish between software asset variation (the application frame itself using software engineering) and solution variation that might not require any customization of the software to reflect the requirements of a given vertical market segment. If the analytical application frame architecture is weak, a variation technique based on solution variation could be a challenge for a software vendor, as the defining technology layer within the analytical application frame architecture (business analytics and meta data repositories) might be weak and not support solution variation and maintenance of these solutions.

Identification of an application frame and its components requires skills from the software vendor to envision future products within a product line, as an application frame measures how effectively an application frame can be used for derivative product development. This efficiency and effectiveness can be measured using the metrics described in Chapter 2. Based on our analysis in the previous chapter, the challenge for an analytical application software vendor is to identify either the market segment(s) or technological innovation(s) that it aims to address. Competition in technology typically provides less value proposition for the majority of end user organizations. Competing in pure technology could lead to a fatal spiral, specifically for smaller software vendors with smaller research and development budgets.

5.2 Implementation of an Analytical Application Frame Architecture

Software products in different application domains share a common characteristic: they require a solid architectural foundation to be able to address current and future market requirements. Software solutions are becoming increasingly complex from a technological perspective. Some of this complexity can be managed by intelligent architectural layering (Jacobsen et al., 1997), with solid application programming interfaces and modular software structure. Software architecture plays a significant role in anticipating future changes to the software solution. Therefore, a weak underlying software architecture is typically a common reason for the high maintenance costs of software products regardless of the software domain.

To be able to provide an adaptable software application frame for derivative software development, an analytical application software vendor must distinguish between two different architectural models – the analytical application frame architecture, selected according to the software domain and its required functionality and the software architectural model that includes elements such as architectural style with design patterns. Software architecture literature includes several different types of architectural styles, such as n-tier client server, layered, pipe-and-filter, or data repository centric. Software implementation of product lines/families also requires product line architecture as a foundation for software implementation. These product lines and product families have different quality attributes and other characteristics when compared to software architectures where the aim is to create a single product.

We have divided this chapter into two sub-chapters. In the first sub-chapter we will explore the importance of software product architectures specifically in derivative software development using a software product line/family approach. In the second sub-chapter, we will discuss the importance of modularity and layered architectures within software development.

5.2.1 Software Architecture in Analytical Applications

According to Messerschmitt and Szyperski (2003), the primary role of software architecture is to "address systemwide properties by providing an overall design framework for a family of software systems." We concluded earlier in Chapter 2 that we will use the definition product architecture defined by Ulrich and Eppinger (1995). This definition emphasizes functional elements which are "arranged into physical chunks and by which the chunks interact." In a similar manner, Messerschmitt and Szyperski (2003) explain that "architecture decomposes system into well-identified modules, describes their mutual dependences and interactions, and specifies the parameters that determine the architecture's degrees of configurability." The most common definition of software architecture is given by Bass et al. (1998a) in following way:

> "The software architecture of a program or computing system is the structure or structures of the system, which comprise components, the externally visible properties of those components and the relationships among them."

The discussion of modularity and interaction between the modules using well-defined interfaces can be linked to analytical application frame architecture and its corresponding analytical application frame components. Each of these frame components interact with each other in a predefined manner via application programming interfaces. Each of these analytical application frame components has a specific role in the overall analytical application frame architecture. An example of this is the business and analytics engine, which provides an interface to the service component layer on one hand. On the other hand, it interfaces to data warehouse and data mart database structures with corresponding business analytics and meta data repositories.

We discussed and compared different data warehouse architectural models in previous chapters and concluded that each architectural model can be viewed from different stakeholders' perspective. A software vendor should aim to define a software architecture that has a broad degree of configurability, as was discussed by Messerschmitt and Szyperski (2003). This configurability reflects back to the ability to utilize an analytical application frame in different vertical market segments without having to modify or adjust the analytical application frame to each market segment. Functional requirements that are specific to a given vertical market segment can be implemented in an extension component layer within an analytical application frame architecture.

Several other authors also explain the importance of software architecture (Jacobsen et al., 1997; Shaw and Garlan, 1996; Kruchten, 1995; Royce, 1998; Bass et al., 1998b; Brown, 2000) in software development. Stafford and Wolf (2001) explain that a system's architecture is "the arrangement of its components into one of more structures defined by the functional role played by each component and the interaction relationships exhibited by the components." According to the authors, there are several different common architectures in place, from general-purpose architectures such as client/server and pipe & filter to domain-specific architectures, such as flight dynamics.

Bengtsson et al. (2000) list three main reasons why software architecture is important in software development. First of all, software architecture sets the constraints for quality requirements (see also Bass, 2001). Secondly, software architecture facilitates communication between different stakeholders (see also Katzman et al., 1994) early in the development process. Thirdly, software architecture facilitates discussion and communication between software architects and software engineers. Kruchten (1995) also explains the importance of different stakeholder perspectives in his landmark article "The 4+1 View Model of Software Architecture." According to this article, the ability to change architecture and use it in different variations (variability) and accommodate it to different new existing and future software requirements is largely defined by the robustness of the underlying software architecture. This type of variability, enabling accommodation of future software requirements, fits well with the requirements of an adaptable analytical application frame that should be built using an architectural model that accommodates future software requirements. This type of adaptability increases the effectiveness and efficiency of an analytical application frame.

In a landmark article, "Architectural Mismatch: Why Reuse Is So Hard," Garlan et al. (1995) introduced the concept of architectural mismatch which occurs when "assumptions of the parts about their intended environment are implicit and either do not match the actual environment or conflict with those of other parts." They name several reasons for this architectural mismatch, such as programming languages, operating platforms, or database schemas in different combinations as a source of complexity. They identified three other assumptions that caused architectural mismatch: infrastructure assumptions for the components, assumptions of what part of the software should hold the main thread of control, and assumptions about the underlying data model. This type of mismatch is evident when an analytical application software vendor fails to align selected analytical application frame technology sub-strategy with selected analytical application frame architecture sub-strategy. This type of mismatch would be a result of failed selection of underlying technology adversely impacting the defining technology selection, resulting in weak market segmentation. Therefore we can argue that the alignment between these sub-strategies will have an immediate impact on current and future market segmentation.

5.2.2 Importance of Software Modularity and Layered Software Architecture

Several authors (Jacobsen et al., 1997; Herzum and Sims, 2000; Bosch, 2000; Griss, 2001; Cheesman and Daniels, 2001; Latchem, 2001) describe component-based software development using a modular and layered architecture. An advantage of a layered approach is to hide the complexities of underlying IT infrastructures from functional developers and therefore differentiate software development of application domain-specific components from component development, what some software theorists call horizontal component application development. The latter components are typically close to the hardware and operating system level and deal with IT infrastructure specific issues. Each component layer or tier is shielded from lower levels using appropriate application programming interfaces that can be called by higher-level components for the functionality that is needed to be able to perform a function. Jacobsen et al. (1997) depict an example of layered software architecture. This architectural model is divided into system specific components, including middleware components and business-specific components that are the basis for applications and application systems (see Fig. 21).

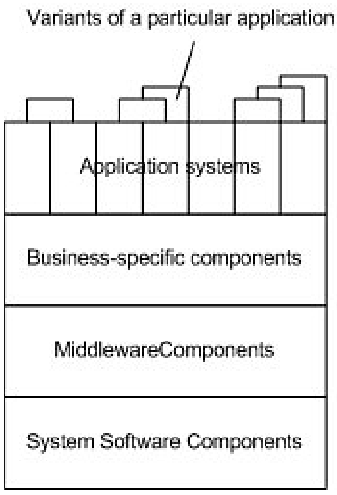

Variants of a particular application system
Application systems
Business-specific components
MiddlewareComponents
System Software Components

Fig. 21. Layered Software Architecture (Jacobsen et al., 1997).

This layered architectural model resembles closely the layered model in Chapter 3 (Figure 11), where the data layer is separated from the business logic and the presentation layer. The only difference between these models is that the architectural model in Figure 11 is specifically geared to analytical application development, while the architectural model in Figure 21 is more or less a generic description of a layered software architectural model. Sharp (2000) concluded that using a layered architectural model and medium-grained architecture enabled the case study company (The Boeing Company) to achieve "the original goals of encapsulating change and maximizing reuse." Sharp (ibid) found out that the layered architectural model enable the case study company to create an application that was more independent of the underlying hardware and operating system environment.

Cheesman and Daniels (2001) view a system product as the two lowest level layers (Business Services and System Services). Adding a user interface (UI) will form an application that end users can access. The importance of separating the user interface from system logic is obvious, as different user interfaces may be connected to the same system services. According to the authors, a component architecture is "a set of application level software components, their structural relationships, and their behavioural relationships." In a similar manner to Jacobsen et al. (1997), Williams (2001) has defined components into three different categories, namely GUI components, service components, and domain components. If we relate this layering or categorization to our analytical application frame architecture, we can identify similar layers of functionality in service component layer within our analytical application frame architecture sub-strategy. These layers are separated by effective application programming interfaces that enable the software development vendor to replace old software components with new components whenever this is required.

In a similar manner, Bosch (2000) concludes that the layered architectural style "decomposes a system into a set of horizontal layers where each layer provides an additional level of abstraction over its lower layers and provides an interface for using the abstraction it represents to a higher-level layer." Bosch (ibid) also discusses different layered architectural models, where some models allow each layer to call only its immediate subordinate layer, while in some cases a layer can invoke calls to any subordinate layer, which obviously increases the coupling between the layers. According to Bosch (ibid), the process of defining a layered architecture includes several steps, such as identification of a number of abstraction levels, representing them as layers, then assigning components to these layers, and possibly "re-modularization of components that contain functionality belonging on different levels." This latter step is very demanding for software vendors, as some of the required functionality must exist in different architectural layers. Therefore, some of this functionality has to be duplicated on several layers to be able to fulfil the end user organization requirements of the software functionality. A good example of this is business and analytics engine and meta data management functionality within a hub-and-spoke architecture, where spokes will perform independent calculations and the hub exists in its own in a highly distributed environment.

5.2.3 Software Architectures in Software Product Lines

Software architectures have a specific role within software product line development. This is discussed by several authors (Bosch and Bengtsson, 2001; Bass et al, 1997; Macala et al., 1996). Bosch and Bengtsson (2001) emphasize the importance of software architecture within product lines to provide a means of identifying commonalities between different software products and building a product line architecture to support maximal reuse of software assets (Bosch, 1999b). Selection of a software architecture will impact the success of reuse across products within a product line. Several studies have been reported of product line architectures (Bass et al, 1997; Macala et al., 1996; Dikel et al., 1997). According to Bosch (2000), software architecture defines the shared components in a software product line. Batory et al. (2002) conclude that product-line architectures are “designs for families of related applications; application construction is accomplished by composing reusable components.”

According to Bosch (2000), a product line architecture describes the common architecture for a set of related products. This type of architecture development must take into account variability and differences between various products in the product line. Wijnstra (2002) differentiates between product family architecture and platform architecture, where the latter architecture “is used to build the reusable assets within a platform” while the former “is the shared architecture with which the family members must comply.” Jacobsen et al. (1997) define this type of architecture and software development as application family engineering. Bosch (2000) divides product line architecture design in six separate steps: business case analysis, scoping, feature and product planning, product line architectural design, component requirement specification, and verification. According to the author (ibid), the selected product line architecture is in some cases used across all different products, while in some cases separate products have their own component implementation. According to the experience of Bosch (ibid), products within a product line typically use the same archetypes, with minor product specific modifications. In a similar way, Clements and Northrop (2002) conclude that there is an architecture for the product line as a whole, but also for each and every individual product. According to the authors, individual product architecture is derived from product line architecture by “exercising built-in variation mechanisms to achieve instances” (ibid). The authors also refer to unknown hardware or other performance-affecting factors at the time of designing the product line architecture. As software organizations do not know future technological requirements, some of the architectural decisions that are made early in development could lead to problems in future derivative product development.

Software architecture with well-composed component distribution across different architectural layers is extremely important for a software vendor, as maintainability, performance, reliability, safety, and security could otherwise be compromised. Specifically, if software vendors have to implement new functionality into the software, a poorly designed layered architectural model could impact the maintenance of the system, as one layer will impact the other. With poor interface management, several layers and dependent components must be changed due to these new functional amendments. The less a software vendor has to change the existing software component structure, the better the software architecture is. Therefore, the software architecture is the core and most important factor in any software development due to the dependences on different elements such as the underlying IT infrastructure described in Chapter 4.

The importance of the software product line architecture within analytical application software solutions impacts mainly how well the analytical application frame architecture has been defined to accommodate different vertical market segments. Each variation technique that an analytical application software vendor can utilize in derivative software development reflects back to the adaptability of the software architecture that has been defined for the analytical application software solution. According to Clements and Northrop (2002), software product line architecture provides the foundation for a software product architecture that is derived from common software assets. In a similar vein, an analytical application software vendor should treat the domain-specific architectural model as the foundation for derivative products that accommodate a given vertical market segment.

5.3 Selection of Analytical Application Frame Technology

Chapter 4 included a description of the analytical application frame technology sub-strategy and how this sub-strategy relates to the two remaining sub-strategies within an Analytical Application Frame Strategy framework. We concluded that underlying technology, together with a corresponding execution and development environment, can impact both current and future market segmentation strategies via different alignment perspectives as identified in Chapter 4. Weak underlying technology could have an adverse impact on the strength of the analytical application frame architecture. We concluded prior to this chapter that the degree of configurability of the software architecture can be linked to the modularity of the software system. We also concluded that the selected software development approach is part of an analytical application frame technology sub-strategy and that the selected approach must support derivative software development. Because of this, we concluded that software product line development is a solid component of analytical application frame development. Software product line engineering has aims similar to those of large-scale reuse and software application frame development (Sääksjärvi, 2002).

Product line engineering is complemented with domain and application engineering. According to several articles (Griss, 2001; Pronk, 2000; Atkinson et al., 2000; America et al., 2000), component-based software engineering is a good complement to these other three engineering approaches. We have also included commercial-off-the-shelf solutions, as these can be deployed as part of the reference or domain architecture as described by Dikel et al. (1997). This architectural layer will also have an impact on the two other layers, but the impact will be in both directions. The execution and development environment layer includes the component execution environment, the integrated development environment, and other tools and methods to enable effective product line engineering. The lower architectural layer (underlying technology: standardized infrastructure) represents technologies that a software vendor includes within the software package/solution. These elements can be purchased and do not require the use of domain-engineering or software development related variation techniques. It is evident that selected underlying technologies could have an impact on all the architectural layers listed above.

The aim of this chapter is to discuss in more detail how each technological selection within an analytical application technology sub-strategy could impact software development of analytical application software solutions. We will first explore how the selection of underlying technology can impact execution and development environment and what type of issues and questions a software vendor might run into in the selection process. Secondly, we will explore the importance of architectural style and patterns and how these could impact the development of an analytical application solution. Thirdly, we will explore the impact of the selection of execution and development environments, as this could become a major factor in market segmentation if a multiplatform software development approach is selected as the foundation for software development. Finally, we will discuss how the selected software development approach could impact analytical application software development. This discussion includes software product line development and development of software assets using component-based software engineering.

5.3.1 Selection of Underlying Technology (Standardized Infrastructure)

We listed the elements belonging to underlying technology in Chapter 4: the database management system, the operating system environment, and distributed technologies such as DCOM and CORBA. Each of these technologies will have an impact on the domain-specific architectural model, which in our case is the analytical application frame architecture defined in Chapter 4. Distributed technology enables an analytical application software vendor to implement distributed features that are also required in the hybrid data warehouse model that we proposed in Chapter 3. This architectural model includes the characteristics of remote data warehouse and data mart database structures. From the end user perspective, these remote databases are viewed as if they were local via a “virtual view.” This type of architectural model can be implemented using distributed technologies. Doerr and Sharp (2000) explain that object request brokers (part of the CORBA standard) provide "a basic communication mechanism that preserves distribution transparency (and other physical architecture dependencies), while preserving a method-based interface between clients and servers." If an analytical application software vendor decides to build a fully distributed solution environment, selection of a distributed infrastructure will become part of the selection process for the infrastructure. These selections will also impact the execution and development environment, as they must support development of distributed environments.

Selection of an underlying operating system and hardware environment could have a profound impact on several other dimensions, such as the execution and development environment, but also on the analytical application frame architecture and its adaptability. In some cases, a software vendor could cause internal lock-in to a specific operating system and hardware environment. This architectural lock-in is described by Morris and Ferguson (1993) specifically from an end user organization perspective, but in this case the lock-in would be for the software vendor, as the end user organization could in some cases move to other analytical application software solutions that support multiplatform environments. A software vendor that selects proprietary architectural models and/or operating system or hardware environments could face difficulties in future market segmentation strategies if the underlying technology selection for some reason should fade or be replaced by new technology. In some cases, disruptive technologies (Christensen, 1997) could change the overall IT market, making the software offering from a software vendor obsolete. An example of a proprietary operating system and hardware environment is described by Schleicher and Taylor (1989), highlighting a development effort using Application System/400 (known today as OS/400 for iSeries/400 hardware). If a software vendor implements application software for this environment with the included development tools (such as RPG), both the software vendor and the end user organization are locked in to the application vendor and hardware environment.

Pronk (2000) concludes that the software industry used to build proprietary solutions, but with increased market pressure and end user organizations demanding non-proprietary and open standards solutions, software vendors have been compelled to initiate the use of COTS components and other available commercial large-scale reuse components. Pronk (ibid) explains that within the medical software domain, the typical hardware environment consists of standard PC architectures, use of standard middleware technology such as DCOM for all inter-process communication, and use of standard software packages such as database management systems, license management, and network software. In their case study, Pronk (ibid) selected COM as a middleware architecture. According to Pronk, "all available interfaces between units in the platform and between the platform and its plug-ins are exported in IDL (Interface Definition Language)." All of the communications between components are based on standard interfaces, and no classes or major structures are exposed over component boundaries. The experiences from this platform development, according to Pronk, have demonstrated that "a dedicated monolithic platform incorporating a lot of application functionality can be well used to obtain a high level of reuse for a product line with very similar products."

Morisawa (2000) explains the difficulties of having different architectural configurations that have to be tested for software applications. Morisawa (ibid) includes discussion of the difficulties in writing solutions in distributed computing environments, as "several architectural questions arise, like where computing resources are distributed, and how the communication among computing resources are implemented." To overcome some of these problems, Morisawa (ibid) introduces a distributed computing model that "classifies products lines for distributed processing systems into seven categories based on the location of data storage and the style of processing between client and server." This example highlights our prior discussion of the complexities of highly distributed environments. When these IT infrastructure selections are combined with domain-specific architectural models such as the federated data warehouse model, the analytical application software vendor will have several challenges to overcome.

Doerr and Sharp (2000) illustrate an architectural model within the avionics software domain where the aim is to shield the underlying infrastructure environment from the application software layer. Doerr and Sharp (ibid) conclude in their case study report that "the success of an avionics product line application rests in large part of the developer's ability to remove platform-specific hardware dependencies and inter-component dependences, as both of these factors induce variability." In this case, Doerr and Sharp do not refer to platforms as defined by other product line definitions (Jaaksi et al., 1999; Pronk, 2000; America et al., 2000; Bosch, 2002), but to hardware platforms and how to shield the applications from the underlying hardware and operating system environment. Bosch (2002) also concludes that standardized infrastructure does not include any domain-specific functionality and does not therefore require variability management or domain engineering.

Where the database management system is the foundation for an analytical application solution, the selection of this technology could have an impact on the robustness and adaptability of any defining technology element. First of all, an analytical application software vendor and its development organization rely on the functionality of the database management system and its application programming interfaces. Any weakness in the underlying technology could translate into a weak analytical application frame architecture. This could have a snowball impact on any alignment perspective portrayed in Chapter 4. Therefore, the competitiveness of defining technology for an analytical application software vendor can be measured by how well the fit between the analytical application frame technology sub-strategy and the analytical application frame architecture sub-strategy is implemented (Technological Responsiveness). In a similar manner, weak underlying technology will have an impact on "Technological

Adaptation," which is the relationship between analytical application frame technology sub-strategy and analytical application frame leverage sub-strategy.

An analytical application software solution is dependent on the performance of the database management system: the ETL engine and its performance is bound to the performance of the database management system. The completeness of application programming interfaces for a database management system is dependent on the implementation of the database management system. Unfortunately there are differences between different database management systems which prevents and analytical application software vendor from having one software release that would implement all these different application programming interfaces.

In summary, selection of underlying technology can impact the development of an analytical application software solution from several different fronts. Selection of a weak and proprietary database management system and operating system and hardware environment could cause considerable harm to a software vendor in the long term both in terms of increased software maintenance and also in weakened market segmentation possibilities.

5.3.2 Architectural Styles and Patterns

Architectural styles are categorized by the Software Engineering Institute into five different categories; the independent components family, the data flow family, the data-centered family, the virtual machines family, and the call and return family. According to Clements and Northrop (2002), architectures are seldom built from scratch and, therefore architectural styles "represent a current approach to reusing architectural design solutions." According to the authors (ibid), a variation of architectural styles is design patterns that "occupy the same role at a finer granularity of design." The authors conclude that, "whereas architectures prescribe how large-grained components (subsystems) interact with each other, patterns usually suggest ways to implement individual (or groups of finer-grained) components." Shaw (1995) compares different architectural styles and argues that different architectural styles "lead not simply to different designs, but to designs with significantly different properties."

Stafford and Wolf (2001) conclude that architectural styles enable stakeholders to communicate about the high-level structure of a software system. These architectural styles also enable software designers to utilize predefined patterns that describe interactions among components. Because of this, these styles help software vendors to depict a high-level view of the component infrastructures. These architectural styles or patterns include the needed functionality to describe the interaction functionality but not application functionality (Bass, 2001). According to several authors (Garlan et al, 1995; Perry and Wolf, 1992; Shaw and Clements, 1997), architectural styles describe the components, their interaction, and possible constraints on the interaction. These constraints could be anything from the underlying technology selection to the corresponding execution and development environment.

Monroe et al. (1997) argue that “architectural styles, object-oriented design, and design patterns all hold promise as approaches that simplify software design and reuse by capturing and exploiting system design knowledge.” According to the authors (ibid), an architectural style “characterizes a family of systems that are related by shared structural and semantic properties” (Abowd et al, 1993). Mili et al. (2002) define architectural styles within software development as “classes of similarly patterned software architectures.” The authors (ibid) conclude that an architectural style is a class of architectures that is characterized by

- Component types
- Communication patterns between the components
- Semantic constraints
- A set of connectors

An analytical application software vendor must distinguish between IT infrastructure environments that are part of the underlying technology and the domain-specific architectural model that belongs to analytical application frame architecture sub-strategy. These two will have a direct dependency as, for example, the distributed data warehouse environment will require specific distributed IT infrastructure technology that will be part of underlying technology. Some IT infrastructure environments could prevent the software vendor from expanding its solutions to new market segments due to weak selection of both the underlying and the defining technologies.

5.3.3 Selection of an Execution and Development Environment

Selection of an execution and development environment is driven by the requirement of whether a software organization must support several different underlying technologies, such as operating system, database management system, and distributed technologies. The selection of these technologies is highly dependent on the analytical application frame leverage sub-strategy and the analytical application frame architecture sub-strategy, as discussed in Chapter 4. If the software organization decides to support a cross-platform development environment, the choice might cause different complications, as described in the case study of Netscape Corporation (Cusumano and Yoffie, 1999). If the selection of the execution and development environment is restricted to one IT infrastructure environment such as the Microsoft Windows software, the software organization could run into other problems, for example restricting its selection of future market segmentation strategies or analytical application frame architecture amendments that are needed to satisfy the needs of end user organizations. According to America et al. (2000), the following development tools are some of the standard tools that software organizations use to support product family development:

- Word processors
- Visual modeling tools

- Code generators
- (Cross-) compilers, linkers, debuggers
- Configuration management tools

Atkinson et al. (2000) assert that current development tools and technologies are not well versed in supporting true software development of product families, as they are more geared towards traditional staged development processes, and therefore "existing product line approaches have been forced to concentrate on the earlier activities in the software life cycle and thus often appear to developers to be somewhat divorced from the real business of coding." This statement demonstrates the dependency between the selected execution and development environment and the selected software development approach. Atkinson et al. (ibid) believe that future execution and development environments have to provide better support for software product family development. The dependency of underlying technology on both the execution and development environment and the software development approach is discussed by Doerr and Sharp (2000). Doerr and Sharp (ibid) conclude in their case study article (The Boeing Company) that "one essential attribute of product line architecture is that it effectively isolate the logical, or static, aspects of the application from any product-specific variations in the physical architecture, or execution environment." This statement goes back to the selection of the underlying technology (standardized infrastructure) and how it impacts all other dimensions within analytical application software development.

It is also important to understand that the selected software development approach will have an impact on how a software vendor selects its underlying technology – in the case of using software product line engineering, it is important to have a good understanding of what is known in the existing software product line literature in the form of case studies and other relevant information sources. Doerr and Sharp (ibid) conclude that the aim of physical and logical architecture is maximize reuse. Physical architecture "supports containing change by isolating application components from hardware and COTS software volatility." Interestingly, this type of volatility in this case relates back to the volatility of physical hardware or even database models. The data staging area within a federated architectural model shields volatility from underlying operational data sources as described in Chapters 3 and 4.

In summary, software development tools supporting serious product line engineering efforts with corresponding development methods such as Component-Oriented Platform Architecting Method Family for Product Family Engineering (CoPAM) (America et al., 2000) and KobrA (Atkinson et al., 2000) are still limited and bound to more traditional development approaches. If a software vendor wants to be serious about supporting true software product line engineering with corresponding domain and application engineering (with component-based development), the selected tools must include all of the life-cycle management areas in the development of common software assets within an analytical application frame.

5.3.4 Selection of a Software Development Approach

The aim of this chapter is to discuss software development approaches that a software vendor can utilize when building analytical application software solutions using a software related product platform approach. We will initiate our discussion of software product line engineering and how it can be linked to derivative software development using a product line/family approach. This discussion is then expanded to show how the literature explains software asset development with different granularity levels in the software assets as the foundation for an analytical application frame. Product line engineering expects domain and application engineering to be part of product line engineering, and these engineering methodologies are used to build the business case for a software organization to utilize derivative software product development. Each software organization needs a practical implementation approach for common software assets within an analytical application frame. We argue that component-based software engineering is a solid supporting engineering approach to produce line development.

5.3.4.1 Software product line engineering

The manufacturing of hardware products has several advantages compared with development of software, such as economies of scale. The experience and knowledge of hardware product lines have provided necessary cost reductions (Svahnberg and Bengtsson, 2000). The same approach is gaining interest within software development. According to Yacoub et al. (2000), product line engineering "is a specialized form of software reuse that promises productivity, quality, and shorter time-to-market in developing similar products in the same domain." The underlying premise of this approach is to provide to software developers – in our case analytical application software vendors – the ability to build a common analytical application frame with corresponding software assets that can be used across different products in a product family.

Wijnstra (2002) concludes that introducing a new product development approach such as product family development – a platform of reusable components – is not enough, as it will impact "a company's business, processes and organization." Wijnstra (ibid) gives an example of this by explaining how the marketing department must work together with product management and project management of the software solution. This type of conclusion reflects back to our Analytical Application Frame Strategy framework, where each alignment perspective must be in balance. If there is imbalance between product management and marketing management, the fit between the analytical application frame architecture sub-strategy and the analytical application frame leverage sub-strategy could become a problem. Therefore, the fit's "Functional Flexibility" could potentially create a problem for the software vendor. Wijnstra (ibid) define the term product family as "a set of related products that are realized on the basis of a shared architecture, using assets from a platform."

Wijnstra (ibid) categorizes product line scope as being focused on external markets, while family and platform scopes should be based both on internal and external considerations. Another interesting question that the author (ibid) raises is whether all of the assets (including product-specific ones) within a platform must be included or only the ones that have been used by several family members. This kind of discussion is relevant in the case of implementation and planning of a software application frame, as some of the needed functionality might be part of the extension component layer and some part of the software application frame itself. Wijnstra even suggests that a product family could be based on several platforms that are built on top of each other. This type of discussion needs to reflect back to a software application frame and its four principles (Sääksjärvi, 2002), defined in Chapter 2. We argue that the notion of having several platforms to support derivative products is against the four principles of software application frames. If additional platforms are needed (such as in the form of software application frame extension), this extension should not have an impact on existing modules in the originating software application frame. Therefore, dependencies between two platforms are not allowed according to the original definitions of a software application frame.

Bosch (2002) introduces different maturity levels for product lines and concludes that product line development in an organization "evolves through a number of maturity levels." These maturity levels include standardized infrastructure, platform, software product line, configurable product base, program of product lines, and product populations (see Fig. 22).

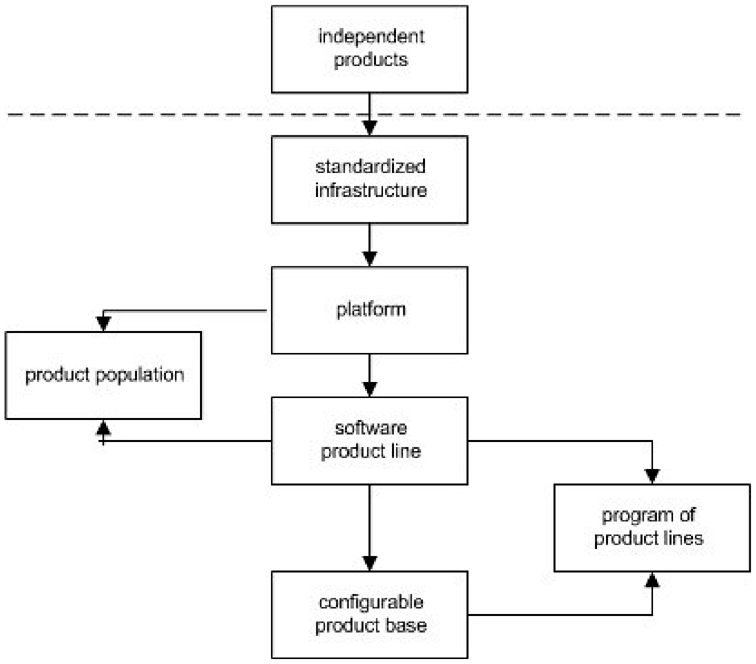


Fig. 22. Maturity Levels for Software Product Lines (Bosch, 2002).

According to Bosch (ibid), a platform precedes a software product line and a standardized infrastructure. It is typically created by a dedicated domain engineering team. Bosch explains that a platform includes a standardized infrastructure. It also "captures all functionality that is common to all products or applications." The common functionality that is not part of the infrastructure must be implemented by the software organization itself. According to Bosch (ibid), the first step in defining commonality in software products within a product line is to standardize the infrastructure for the products under development. This corresponds well with the selection of an underlying technology within an analytical application frame technology sub-strategy. A standardized infrastructure includes elements such as the operation system environment, the database management system, and other infrastructure related components that a software vendor should not implement internally (underlying technology). According to Bosch, the infrastructure layer will not include domain engineering efforts. Software organizations do not include variability management, as it usually does not include any domain-specific functionality. The platform itself requires some level of domain engineering and variability management even if most of the common functionality for all of the derivative products is captured in the platform itself. According to Bosch, most of the engineering in platform development is more or less focused on application engineering efforts.

In summary, the platform layer includes all the standard infrastructures from the previous level plus all functionality that is common to all products and applications. The following level represents the actual software product line that exploits all the communality from the platform, where "consequent development may be undertaken to increase the amount of functionality in the platform to the level where functionality common to several but not all products becomes part of the shared artefacts." This notion relates to our discussion of the analytical application frame and whether some of the functionality should be implemented into the analytical application frame that is common to all derivative products or whether it should be part of extended functionality in our analytical application frame architecture. We will have further discussion about this subject in the section on analytical application solution variation techniques later in this chapter.

Pronk (2000) discusses how the software and academic communities have aimed to achieve higher levels of productivity in software development. Considerable research has been done into organizational issues (Capabilities Maturity Model), development methodologies (object-oriented methodologies, component-based development methodologies), technology (architecture, reuse), but none of these alone has been the silver bullet (Brooks, 1987) for development. According to Pronk (ibid), product line architecture will help organizations to reuse "an entire class of products with only minimal variations to support the diversity of individual product family members." His work introduces a model in the medical imaging domain, where product line architecture is based on a platform that can be varied only through well-defined interfaces. Pronk concludes that a platform consists of "documentation, object models, support libraries, templates and examples and a large set of binaries." Pronk (ibid) bases his product line development approach on Application Family Engineering (AFE), defined and introduced by Jacobsen et al. (1997). According to Pronk (2000), the maturity of the given domain regulates the variations that are needed in the platform.

A common approach in platform development is to build the platform to be adaptable to possible changes in market conditions, whereby the software vendor can make changes in its segmentation strategy. The challenge for any software vendor is to enable the software application frame to be amended with future feature requirements – requirements that are not known at the time of planning the initial software application frame. If the chosen software application domain is mature and the requirements of selected customer segments are well known, most of the functionality can be built into the software application frame, and market segment specific features can be built into the software application frame extensions. Pronk (ibid) concludes that about 80 percent of the required functionality in the case study organization has been derived from the platform, which can be regarded as aggressive reuse. Pronk also explains the complexities of underlying technology (IT infrastructure) in the implementation of the product family due to maintenance of different technological environments and combinations of technologies. This discussion is relevant also for the analytical application software domain, as was seen in Chapter 4 when discussing and exploring selection of an underlying technology with its corresponding execution and development environment.

5.3.4.2 Software assets within a software application frame and component granularity

Recent software product line literature (Jaaksi et al., 1999; Pronk, 2000; America et al, 2000; Wijnstra, 2002, Brown et al., 2002, Bosch, 2002) has defined the concept of a platform as the foundation for software asset development. Bass et al. (1999) explain that several organizations refer to the core asset base as the platform, and that therefore "core software assets" and "platform" might be used interchangeably in some software organizations. Bass et al. (ibid) also conclude that the product line approach involves strategic, large-grain reuse of software assets as opposed to developing products from scratch. Clements and Northrop (2002) include in the core assets the architecture, reusable software components, requirements statements, documentation and specifications, performance model, schedules, budgets, and test plans and cases. They emphasize that the architecture is one of the key core assets in software development. Several organizations refer to asset development as equivalent to product platform development.

Clements and Northrop (1999) define how different organizations obtain software. Any organization has three different ways to obtain software: by developing it itself, by purchasing off the shelf, and by commissioning it to a third party developer. According to the Clements and Northrop (ibid), core asset development has been traditionally called domain engineering. They argue that core asset development or acquisition and product development using core assets can be run in parallel. This basically means that organizations should have both base assets development and product development running at the same time. They also emphasize that development products might sometimes turn into core generic assets for further reuse. The value of core assets is realized when new products are derived from them. The authors also point out that core assets should be made generic to enable them to be the basis for several different products.

According to Bosch and Högström (2000) a software product line "exploits the commonalities within a family of products by developing and maintaining a set of core assets, i.e. a product line architecture and a set of components implementing the architecture." Another definition of software assets is given by Withey (1996), where a software asset is "a description of a partial solution (such as a component of a design document) or knowledge (such as a requirements database or test procedures) that engineers use to build or modify software products." Jaaksi et al. (1999) defines reusable assets as a "collection of reusable components grouped into service blocks, design guidelines, and policies for using them." According to Svahnberg and Bengtsson (2000) the concepts within a product line domain are called artifacts, and these artifacts can be "designed, modeled, analyzed, and instantiated into products of the product line or family."

Among the decisions to be made when deciding on a product line is to set the scope that is economically feasible. According to Clements and Northrop (2002), if the scope is set too large, the associated core assets will be hopelessly too generic and if the scope is too narrow, the market demand for the product suite will be too narrow. According to the authors (ibid), "if the scope bounds the wrong products, the product line will not find a market." This discussion relates back to different alignment perspectives within the Analytical Application Frame Strategy framework. If the analytical application software vendor selects too narrow a market segmentation or an underlying technology that is not widely supported by the general IT market, the software vendor could run into a situation where the solution offering is not paying off for the software vendor.

The question of granularity levels for software assets used in software product line development is addressed by Bosch (1999b). According to Bosch (ibid), some organizations have "moved towards product-line architecture based software development, especially through the use of object-oriented frameworks as reusable assets." These object-oriented frameworks provide larger reusable entities. According to Bosch and Högström (2000), these entities are larger than 100 Thousand (kilo-) Lines of code (KLOC). According to Batory et al. (2000), frameworks often arise in product line implementations. Batory et al. (ibid) argue that frameworks are "appropriate for reusing software parts and specializing them in multiple ways for distinct applications." According to Johnson and Foote (1988), a framework is "a collection of abstract classes that encapsulate common algorithms of a family of applications." Sparks et al. (1996) provide tips for buying, building, and using object-oriented frameworks and explain the difference between traditional reuse of library-based artifacts versus reuse of frameworks. Pronk (2000) concludes that a platform may be considered a component framework as defined by Szyperski (1997). Variation of the platform can be achieve in two ways, both by configuration and via well-defined interfaces (Pronk, ibid).

According to Sharp (2000), patterns are used to capture industry best practices to answer the frequently asked question "why software was designed a certain way." Sharp (ibid) concludes that Gamma et al. (1995) divide their design patterns into three main categories:

- Creational: patterns used in flexibly constructing objects
- Structural: patterns outlining class or object composition
- Behavioural: patterns describing how objects interact and distribute responsibility

One known design pattern is the Model-View-Controller (MVC) pattern, originally developed in the Smalltalk community for development of graphical user interface systems. The main idea behind the MVC architectural pattern is "the separation of the domain model from the user interface that presents domain information to users and allows them to manipulate it" (Sharp, ibid).

Yacoub et al. (2000) argues that product line engineering with commercial-off-the-shelf (COTS) based development has a great deal of potential in practice, but that "acquiring commercial components for a product line carries a lot of risks." Yacoub et al. (ibid) also concludes that the architecture of a product line is known as reference or domain architecture (Dickel et al., 1997) because it is "instantiated in applications that belong to the product line domain." Components within this domain architecture are "deployed either as in-house off-the-shelf (OTS) components or commercial-off-the-shelf (COTS) components" (Yacoub et al., ibid). The characteristic of a COTS component is, according to Yacoub et al., as follows:

- It is sold or licensed to the general public
- Customer have no access to the source code
- The component can only be used as a black box
- The COTS component has been built by a commercial software vendor who also provides maintenance and upgrades

As stated before, the risks associated with using COTS components are considerable, specifically if they are included in implementation of a software application frame defining technology. External risks such as the COTS software vendor going out of business or not providing the needed support and functionality, could be a major issue for a software vendor in future development. We argue that these types of COTS components should be carefully considered and should not be part of any defining technology that can not be quickly replaced by another vendor's technology. A good example of COTS usage within the analytical application software domain is the use of external graphics libraries. The core competence of an analytical application software vendor is to provide analytical information to end users via different devices (Internet browser, Windows application, etc.). These graphics libraries provide the means to display this analytical information in different graphical formats. The problem of using external libraries without having the source code could become a major issue for a software vendor, as it would be tied to the performance of the external software provider in technologies that we define as defining technology. Therefore, the usage of COTS or any third-party technology as part of defining technology should be carefully considered and measured from different alignment perspectives.

5.3.4.3 Component-based software engineering (CBSE) with software assets

Several authors conclude that component-based software engineering (CSBE) can be used as a basis for implementation of software product lines (Bass et al., 1998a, 1999b; Bosch, 1999b; Bosch, 2001; Bosch and Högström, 2000; Batory et al., 2002; Clements and Northrop, 2002), as software components enable software organizations to increase the reuse of software assets. This will reduce the costs of software development and maintenance for software vendors (Svahnberg and Bosch, 1999). The aim of software product line development is to build common core software assets that can be reused across different products belonging to the same product line. This approach is very close to large-scale reuse of software components. Existing literature discusses different granularity levels for component-based development. This is one of the key issues not clearly explained in existing software related product platform literature.

Atkinson et al. (2000) explains the difference between component-based software engineering and product line engineering in following way:

> "Component technologies provide the flexible and rapid configurability needed for genuine product development, while the disciplines of the product line approach provide the methodological foundation needed for the development and deployment of sound component-based frameworks."

This statement portrays our selection of component-based software engineering as a key element in building derivative software products from an analytical application frame. Griss (2001) links product line development with component-based software engineering (CSBE) as a means to achieve reuse on a large scale (see also Jaaksi et al., 1999; Brown, 2000). According to Griss (ibid), there is a compelling business reason to "invest in building and managing a set of products as a family, sharing engineering effort and reusable assets." From the management perspective, product line development is more strategic than development of individual products. According to Griss (ibid), product line development becomes strategically important for the executive management when the software development organization can produce results in terms of reduced overall costs of product development, enhanced competitiveness due to decreased time-to-market of the products, and improved product compatibility.

Pronk (2000) expresses his disbelief in the common presumption that "when defining a platform for a product line architecture one simply thinks of implementing it as a number of completely separated components that can be combined in arbitrary configurations by system groups." In a similar way, Pronk (ibid) does not believe in a model whereby a component-creation group will build simple and generic software components without having a strong linkage to end user application product management. According to Pronk, a new "modern Lego approach" is needed, whereby the platform must be "a confined environment where all of the customers are known and the architecture is controlled." Pronk (ibid) concludes that this type of platform development does not require generic components but rather more or less dedicated building-blocks that help in building the application frame.

America et al. (2000) introduce another family of methods that enable organizations to develop product family architecture. CoPAM is a specific family engineering method that can be applied to the development of product families. America et al. (ibid) divide the overall CoPAM processes into three main sub-processes:

- Platform engineering process: develops the platform, which consists of reusable components
- Product engineering process: develops products using these platform components and adding new components when necessary
- Family engineering process: provides feedback to platform and product engineering processes

America et al. (ibid) recognize that the family engineering process includes several steps that have been identified by Jacobsen et al. (1999) in their Unified Software Development Process method.

Van Ommering and Bosch (2002) conclude that software organizations can build software product lines by not using components at all, but by "using a framework with component plug-ins, where the plug-in components implement diversity rather than commonality." Van Ommering and Bosch (ibid) also discuss whether they believe in the third-party development market. The answer is very interesting, as it reflects also the analytical application solution marketplace. According to van Ommering and Bosch (ibid), the third-party market tends to implement generic software components of different granularity levels. This does not help Philips (their case study company) to implement the needed domain-specific functionality. This is a common problem in the specialized software development domains that build its own defining technologies requiring specific technology, with specialized software assets that can not be purchased as COTS components.

Mili et al. (2002) emphasized several factors that are important to reusable software assets. These criteria can be linked to both the analytical application frame technology sub-strategy and selection of the execution and development environment. Software assets must be programming-language independent to be useful for software organizations, and these software assets must include effective component interfaces. It is of relevance to be able to compose software applications from different software assets, and each component must be properly contained, with effective application programming interfaces.

In summary, component-based software development is a valid software development approach in software product line engineering. It has the potential to help analytical application software vendors to achieve large-scale reuse of common software assets in the form of an analytical application frame. Component-based software engineering emphasizes modularity and effective interface management, which enables analytical application software vendors to create a software application frame that is adaptable for changes and can be used in future market segmentation. If the modularity and interface management is poorly defined and implemented, the software application frame could become useless from the product platform effectiveness and efficiency perspective.

5.4 Leverage of the Analytical Application Frame

As discussed earlier in this chapter, software product line development is one way of carrying out derivative product development for common software assets. Based on our research, it is one of the strongest components that could be linked to development of software products using a software related product platform approach. Existing software related product platform literature does not give practical advice about the methods of implementing variation in software development and how product line and product families can be implemented from a common set of software assets.

In the next three sub-chapters, we will explain the three variation techniques available for an analytical application software vendor. The first variation technique is based on solution variation within the analytical application frame architecture. The second variation technique is based on technical variation that can be found in software engineering related literature when building software product lines or product families. The third and final variation technique is based on market segmentation, without changing the product itself. This type of variation could be most effective for a software vendor, as the same software package is delivered to end user organizations without any modifications or customizations. We will initiate our discussion by exploring what type of changes are taking place within the analytical application software market and what types of market segmentation issues an analytical application software vendor could face.

5.4.1 The Software Business Model and Market Segmentation Approach

A change from customized analytical application development towards packaged solutions is discussed in several articles. The main theme in these is whether end user organizations should buy or build these types of solutions (Morris, 1998; Gleason, 1998; White, 1999; Surgan, 2000). We did not find any literature discussing software product line or product platform development of analytical applications and what types of market segmentation and implementation techniques can be applied when building analytical application software solutions.

This new prepackaged analytical solution development is very different from data warehouse implementation of early 1990s, where data warehouse vendors were still doing customized development for end user organizations. According to Surgan (2000), end user organizations are looking for new ways of building analytical applications. One way of doing this is to purchase and use prepackaged analytical applications having built-in vertical or horizontal functionality that the analytical application software solution vendors have accumulated into their solution. This type of solution development enables rapid and cost effective implementation for end user organizations, but it challenges analytical solution vendors to build robust analytical application architectures that enable effective variation to different market segments or price/performance tiers. Cusumano (2004) divides vertical market segments into several different categories. First of all, a software solution could be based on a given industry, whereby the solution is tailored to a given domain such as heavy-equipment rental markets or the distribution industry. The second category represents a "technical specialty," such as computer-aided design programs. If these programs are tailored to a given vertical market, the category is defined as a "technical specialty for a particular industry." Finally, a software solution could address a "platform-specific market," which represents software solutions that run only on a given operating system or hardware environment or a combination of these.

From the software vendor's perspective, the aim of an analytical application vendor should be to have everything integrated, packaged, and easily implemented and delivered to any end user organization in the selected market segment. From the Analytical Application Frame Strategy perspective, an effective analytical application frame should satisfy the following requirements:

- A software application frame as such is not a deliverable product, but must be packaged with other components that conform to the end product: the analytical software solution itself.
- The software application frame should enable analytical application software vendors to utilize the core software application frame when creating additional frame extensions for different vertical market segments.
- A software application frame must be adaptable to be able to support current and

future market segmentations and architectural models without having to rewrite the core software application frame.

Each of these three requirements can be analyzed by different criteria. These criteria will show whether an analytical application is suitable for analytical application frame development and what kind of restrictions/requirements are needed in the software application frame itself.

America et al. (2000) conclude in their review of the Component-oriented Platform Architecting Method (CoPAM) that any development method should achieve the best possible fit between the following interrelated aspects:

- Business: requirements of the market about the products and how the company intends to respond to it.
- Organization: refers to the structure of the developing organization and their core competence.
- Process: portrays the different development steps and activities and their mutual relationships.
- Architecture: refers to the structure of the products themselves.

This discussion resembles the alignment perspective and fit discussion that we explored in Chapter 4. Business requirements reflect the market segmentation and business model that the software vendor is going to address. Organization, process, and architecture relate one way or the other to both the analytical application frame architecture and analytical application technology within an Analytical Application Frame Strategy framework.

5.4.2 Variation Using an Analytical Application Frame Architecture Sub-Strategy

Meyer and Zack (1996) introduced a process platform for information products to provide content variation using a database repository. This reflects closely the variation techniques which an analytical application software vendor must employ when using a business analytics repository as the foundation. The parallels to the Meyer and Zack information products and analytical application software development are similar. The biggest difference is that in the case of analytical application software, the end user organization must have the ability to modify and extend the solution themselves, while the idea behind information products was more or less that the solution vendor provided the whole solution, and that no customization was needed in the end user organization.

We suggested in Chapter 4 that other analytical frame components and market segment specific variations should be implemented and stored in the meta data and business analytics repository. This variation is implemented in the form of predefined or customized extraction rules for different operational data sources, customized database models with corresponding business processes and business rules and measures, and a selection of predefined reports that the solution will provide “out-of-the-box.” The idea behind these solutions is to provide

80 percent of the functionality that is required in the selected market segment. The remaining 20 percent the end user organization either ignores or satisfies using the repositories to add the required features. A profound requirement from the analytical application frame is the ability to maintain several versions of the solution and parallel development of additional rules without breaking the solution when the solution vendor updates the solution with a new release. This type of maintenance requires strict rules from both end user organizations and software vendors. The included documentation of the solution must provide rules and variation points that are available for extending the contents of the solution.

As each vertical market solution will have a different data model due to specific needs, the installation of the solution must generate the required physical data models, business processes, and measurements for the selected domain. The difference between data models in different market segments could be due to different analysis requirements (Imhoff, 2000; Russell, 2000), the depth of dimensions and hierarchies, or myriad other reasons that require a customized vertical data model. A further breakdown into physical implementation of the database data model in an analytical application implies that the database model must be very flexible when changes are required in the common business area model that is stored in the business analytics repository. Some of these changes can be accommodated using a logical representation of the database model. New changes into the model would be applied in the business analytics repository, automatically generating a new physical representation of the database model. Burwen (2000) has described this approach well, where the physical and logical models are generated according to the specified business process requirement.

Another challenge for a software vendor is to identify functionality for the analytical application frame that will be common for any vertical market segment with analytical application defining technology that will enable collection of different vertical market solutions without having to adjust functionality to any specific market segment. If this kind of adjustment must be done, it will be implemented as extended functionality as defined in the analytical application frame architecture in Chapter 4.

To summarize, the elements that are part of a vertical solution that will be implemented to a meta data and business analytics repository is as follows:

- Customized extraction, transformation, and load (ETL) routines for selected source applications (such as Enterprise Resource Planning applications, etc.)
- Customized data models for each selected vertical domain (both the physical and the logical data model)
- Business processes and calculation rules for the selected vertical domain
- Key Performance Indicator (KPI) and Critical Success Factor (CPI) management for each vertical domain

- Customized reports, charts, dashboards, and templates for each selected vertical domain.

It is also evident that predefined integration logic for operational data sources will require customization for each end user organization, as every organization, even in the same vertical domain, could have different business rules and operational applications. These customizations are controlled via a centralized meta data and business analytics repository as portrayed in an analytical application frame architecture. Adaptations to predefined source and target interface mappings (ETL rules) are also tracked within a meta data repository. This enables solution vendors to perform regular updates of the solution without breaking existing changes that might have been implemented by the end user organization. A centralized business analytics and meta data repository enables software solution vendors and end user organizations to use these repositories in the creation of new additional subject-specific data marts by "inheriting" business metrics from the centralized repository with corresponding business processes. This ensures convergent metrics across all data marts within the end user organization. This is part of a bottom-up data mart implementation approach with an enterprise data warehouse implementation approach. The role of the data staging area is to minimize some of the volatility between the operational application and the analytical application. This architectural solution will help software vendors to concentrate more on the analytical application solution development and let other organizations take care of the data integration issues. Therefore, the data staging area can be viewed as the interface between the analytical application solution vendor and the outside world with the operational data sources.

Analytical application software solutions typically provide a set of reports or charts that include a set of specific analytic measurement for the selected market segment. These reports can also be attached to different data models by using a logical abstraction between the physical and logical representations of the database model with corresponding business processes and measurements. Each measurement should be defined and described by an end user organization. These measures can then be used across any data mart implementation throughout the organization. A common dilemma with independent data marts lacking conformed dimensions and measurements is that each data mart becomes a silo of information and does not relate to the enterprise view of measurements defined by the corporate. This is part of the reason why a federated data warehouse model (and our hybrid data warehouse model defined in Chapter 3) is suitable for enterprise analytical application business requirements, as it includes a centralized business analytics and meta data repository.

Finally, an analytical application software vendor can provide solution variation by using a specific license key that will either enable or disable the breadth of the solution that is installed in the end user organization environment. Jaring and Bosch (2002) conclude that variability can be identified before and after product instantiation. Variation points can be identified and introduced by design time and bound at runtime. License keys are typically used and built by Dynamic Linked Libraries (DLLs), whereby solution functionality is activated at runtime based on the validation of the license key. This type of solution is common amongst analytical application solution vendors. It also makes software deliveries easier, as the software solution is delivered and installed in end user environment and the license key defines what functionality the customer is allowed to utilize. From an analytical application software perspective, the software vendor has two alternative ways of using this type of license key management. The first alternative is to install all meta data and business analytics definitions with the installation of the software, but restrict the use of some parts of the solution by using a software license key. The other way of implementing variation is to provide a menu at the time of installation which will install only the parts of the solution that the software license key recognizes. In either alternative, the software vendor must maintain one and only software solution. Software license key management will deliver the needed variation mechanisms.

5.4.3 Variation Using an Analytical Application Frame Technology Sub-Strategy

A key aim in our research into the variability of software is to identify how existing software product line literature explains software variability with respect to different viewpoints that this type of variation can have on software product line development. Variation in analytical application software must be analyzed from two different perspectives. The first and more common variation technique is to define variation within the analytical solution itself, using business analytics and the meta data repository as the variation point. The second variation technique is to use software related variation techniques that have been defined in component-based software engineering and software product line engineering literature. Bosch (2002) explains that organizations should adopt software product line engineering based on maturity levels that he defines in his work. These maturity levels were discussed earlier in this chapter. According to Bosch (2002), "a software organization typically evolves through a number of maturity levels." This also translates to the conclusion whereby some organization are better off concentrating on domain engineering if the product development organization is well versed in the software domain and less well versed in application engineering. Bosch also concludes that volatile domains could be problematic, even for experienced and mature software organizations. We argue that this has been the case within analytical application software development for years, as the software domain has not had a broad development community producing COTS components. This has also caused implementation of proprietary solutions. End user organizations are requesting standardized interfaces so as to be able to combine "best-of-breed" functional software components from different vendors.

Software adaptability, specifically with respect to domain-specific software architecture, is of importance when deploying variation to enable effective derivative software and solution development. The variability must be based on market analysis that determines the product line scope and the required software assets that are needed to build the artifacts to support the selected market segments. Thiel and Hein (2002) explain the importance of variability within automotive systems. That variability must be systematically considered throughout the development process. These types of systems "have typically thousands of requirements, but some of them are extremely important" and some are based on real-time systems (ibid). According to Thiel and Hein (ibid), the automotive industry has initiated the development of multipurpose platforms that "replace mechanical and electronic components with intelligent software solutions." According to the authors, "variability affects all product line artifacts, from requirements to code."

Leveraging the analytical application frame does not necessary have to be implemented via solution variation as portrayed in the prior chapter. The software vendor can also utilize technical variation using strong technology within a software application frame to derive additional new products. An example of this occurs when a software vendor uses the same software application frame to derive user interfaces for different devices such as traditional Windows, an Internet browser, or even a PDA with a tiny screen. This derivative development requires a strong software application frame for all common elements whereby the variation points are implemented to fulfil different user interface technologies and their requirements. This kind of derivative software development emphasizes the use of technology in derivative software development.

According to Griss (2000b), a product line is "a set of products that share a common set of requirements, but also exhibit significant variability in requirements." According to software engineering literature, implementation of variation can be based on two granularity levels, one that is implemented on the actual components and the other that is implemented in the software product lines. Griss (2000a) proposes domain analysis (Arango, 1994; Griss, 1996; Griss et al., 1998) as the technique to extract features from existing or planned members of a product line. According to Griss (2001), domain analysis affects "the design and implementation of the product line architecture and reusable components, infrastructure, and tools that will be used to construct the product line." Domain analysis will be used to identify the needs for variability within the components in a product line. This variability can be implemented by "using a combination of inheritance and templates or a preprocessor or generator." Griss emphasizes the role of the software architect and required skills in the selected domain to be able to identify the commonalities and differences that need to be defined and implemented in software components. Griss (2000a) defines the complexities when facing "crosscutting" features, where software designers have difficulties in separating concerns into separate components. D'Souza and Wills (1999) emphasize that software developers typically rely on an underlying set of infrastructure services. According to the authors, the role of these services is not to customize the behavior of the system, but to "simply provide an implementation of a common virtual machine for use by all components." This type of infrastructure support is part of underlying technology within the analytical application technology sub-strategy.

Bosch (2000) discusses of the applicability of object-oriented frameworks as components in a product-line architecture seen specifically from the software product line variation perspective. Bosch argues that object-oriented frameworks are a "much more accurate model for reusable components in a product line than the traditional component model." The author identified four different framework component models that can be used within software product line development, namely the product-specific extension model, the standard-specific extension model, the fine-grained extension model, and the generator-based model. Software frameworks can be found in different types of vertical domains such as user interfaces, process control systems within a specific application domain, and financial systems. Niemelä (1999) introduces a component framework of a distributed control system with two dimensions: tiers and elements. According the author (ibid), the tiers of the component framework "define the subsystems in the first tier, the integration platform in the second tier, and the product family in the third tier." According to the author, these tiers explain the domain, technology, and business viewpoints of the framework. Part of the work of Niemelä (ibid) explores the mapping of variability to the architecture and components and discusses the adaptability of a component framework.

According to Svahnberg et al. (2001) software variability is "the ability to change or customize a system." According to the authors, these types of changes can be anticipated by constructing a system architecture that will provide this type of flexibility. Jacobsen et al. (1997) include discussion of several different variation techniques available for software development organizations in product line software development: inheritance, extensions, parameterization, configuration, and generation. According to Svahnberg and Bosch (2000), inheritance can be used if a method needs to be implemented differently in a product within a product line. The second variability approach is to define extensions and extension points to the component in question. This will result in different behavior for the component. The third approach is parameterization, achieved by having a placeholder and defined by build time. Both macros and templates are typically used in parameterization. The remaining three variability mechanisms are configuration and module interconnection languages, generation using high-level language, and finally compile-time selection of different implementations. All of these different variation mechanisms can be used to derive and implement different functionality by using common software assets. The result is a new product with new features within a product line. Wijnstra (2000) discusses using service component frameworks in product variation within medical imaging product line software. Wijnstra (ibid) also explains that a framework is a "skeleton of an application that can be customized to yield a product."

America et al. (2000) distinguish between platform time, product time, and installation time development decisions. When a platform time decision is made, a collection of plug-in components is built (software assets) and these components define the platform. When the development team makes a selection of which components will be used in a product, a product time decision is made. Finally, America et al. (ibid) conclude that the final product can be configured at installation time by using parameter lists and defining actual values for these. This approach enables interesting opportunities in the reuse of common software assets in a platform. First of all, the same platform can be reused in different products belonging to a product family. Secondly, some development decisions can be postponed to later stages in the development process by using parameterized components that will get their values either at product time (when components are selected) or even at installation time. Therefore America et al. (ibid) propose that some software development decisions could be made at installation time. Using parameters in installation basically creates different software configurations. This approach is new to software related product platform literature. Pronk (2000) explains that variation of a platform can be achieved in two different ways, either using configuration or by well-defined interfaces.

Regardless of the selected variation technique, the main aim of these different variation techniques is to provide to an analytical application software vendor the ability to maximize the reuse from an analytical application frame that consists of common software assets. The variation techniques presented in this sub-chapter are more technical in nature and require solid software engineering skills from the development organization. Therefore, the two other variation techniques (solution and market segmentation variation) are according to our view more manageable for analytical application software vendors. In the next chapter, we will introduce our third variation technique, more or less based on market segmentation practice using a price/ performance matrix as defined within an Analytical Application Frame Strategy framework.

5.4.4 Variation Using an Analytical Application Frame Leverage Sub-Strategy

The decision support market has evolved during the last ten years from providing highly customized solutions to providing prepackaged analytical applications (Surgan, 2000). These types of changes will most likely force analytical application software vendors to intensify market analysis to be able to stay competitive. Market analysis will also lessen the risks that software organizations will build products that become obsolete or reflect requirements that are not appealing to a selected market segment. Market analysis will include activities such as gathering competitive studies, customer plans, and strategies. These documents will be the basis for a "cohesive business strategy and plan" (Clements and Northrop, 2002). We identified a similar approach in the studies of Meyer and Seliger (1998) and Sääksjärvi (1998), where market segmentation could be implemented via different price/performance tiers using product family development.

We portrayed both solution variation (part of an analytical application frame architecture) and technical variation (part of an analytical application frame technology) and the remaining sub-strategy within the Analytical Application Frame Strategy framework as an analytical application frame leverage sub-strategy. The aim of this sub-strategy is to define the market segments and application areas that the software vendor is going to address. The software vendor has to be extremely careful of not running into problems with any of the defined disconnects in the alignment models that we portrayed in Chapter 4.

The analytical application software vendor has a few alternative ways of variation using an analytical application frame leverage sub-strategy. Our analysis of solution variation described an approach where each vertical market segment has the same software package, but different vertical market business analytics. Another approach is to accommodate additional requirements by changing the actual software solution, using the approaches that were defined in software variation techniques within software product line development. This approach takes the existing common software application frame as its basis. Any additional requirements are implemented as part of extended functionality, defined in our analytical application frame architecture in Chapter 4. Variation can be achieved by implementing plug-ins to the common software application frame. This method applies also to third-party vendors wanting to innovate additional functionality that can be integrated into the core analytical application frame. Using a third-party development organization to provide additional solutions on top of an analytical application frame could give tremendous leverage for a software vendor, as the application frame with its corresponding components would become part of a solution from other software organizations (Gawer, 2000; Gawer and Cusumano, 2002). This variation technique complies well with an analytical application frame leverage sub-strategy, as it does not typically require any modifications or customizations from the software vendor itself.

Another opportunity for market segmentation and variation is to implement integration to other horizontal solutions such as budgeting and forecasting, customer relationship management, or any other software solution that requires analytics from the analytical application solution. This type of solution integration requires documentation of application programming interfaces for the software application frame, as these will be the variation points that can be used to implement extended functionality or horizontal solutions. It is a challenge for any analytical application solution vendor to define a common software application frame that would be applicable to any vertical or horizontal market without having to add software extensions to the core software application frame. This type of variation is more "marketing-oriented variation," as the variation could be implemented by a third-party solution without having to build anything in the analytical application frame architecture itself.

An analytical application software vendor has two main alternatives to utilize the price/performance matrix defined in an Analytical Application Frame Strategy framework. The first option is to append new segments by building new functions/applications internally. The second option is to use third-party technology. Use of third-party technology requires careful consideration when deciding whether theses areas should be internally managed and become part of the core competence for the software vendor. This type of amendment to new vertical market functions/applications is a big decision for a software vendor and should not be done lightly. The decision to move into a new area just because competition is doing it could become "the decision of all times leading to corporate failure."

One of the most important aims in software application frame development is to identify the least common denominator for the analytical application frame functionality that is going to be used across any vertical market segment. In some cases, the software vendor might decide to include features/functionality in the analytical application frame that are not common to all vertical domains, but which will reduce the maintenance of software assets. Another challenge that a software vendor faces is to design the software application frame to be adaptive for future features and functionality that might be required of the software solution. It is important to note that an analytical application software vendor might have to amend its existing core software application frame with additional functionality to be able to build new vertical extensions and thereby meet the expectations of these additional market segments. This is very important distinction that must be made in software application frame development, as additional software solution requirements are sometimes met by amending the core software application frame and not by creating frame extensions. The decision on these amendments needs to be made on a case by case basis, as each software application domain and intended additional functionality will be different, and some of the functionality might not be used across all market segments. We emphasized in prior chapters the importance of having strong domain knowledge within selected market segments when building software solutions. This knowledge will become invaluable when planning the variability of the core software application frame, when reviewing what parts of common software assets should become a part of the common software application frame, and when determining what pieces should be moved to frame extensions for specific market segments.

Finally, in the same way we already described in the solution variation of an analytical application, the software vendor could utilize license key configuration as means of solution variation. In this variation approach, the functionality of the software will be dependent on the license key that the end user organization receives. This type of market segmentation reflects analytical application frame leverage sub-strategy, where the software license key will control the price/performance level of the software functionality. This type of variation is cost efficient for a software vendor, as it can provide one software configuration to all customers, the license key controlling what features of the software will be activated. Once the end user organization is ready to amend its solution, the only thing that has to happen is the receipt of a new license key that will activate new features in the software solution. This type of variation does not require maintenance of several different versions of the software product, but it can be used for market segmentation purposes on the price/performance matrix that was defined in the Analytical Application Frame Strategy framework in Chapter 4.

5.5 Chapter Summary

The aim of this chapter was to introduce in more detail how an analytical application software vendor can use existing software engineering technologies effectively when implementing an analytical application frame involving large-scale reuse as defined in principle one by Sääksjärvi (2002). We were also able to conclude that references to software engineering approaches in existing software related platform literature discussed component-based software development. These references did not demonstrate or discuss in more detail practicalities when implementing software solutions. Another aim of this chapter was to explore how large-scale reuse can be implemented in the case of analytical application software solutions. We identified software product line engineering as the closest viable option for an analytical application software vendor to create derivative products from an analytical application frame.

The selection of software product line engineering as a viable software development approach prompted us to research product line software engineering in more detail, and to list the most important factors that an analytical application software vendor needs to know when initiating derivative software development. Part of our analysis was to explore what types of methods have been introduced into the product family engineering domain. Due to the newness or immaturity of software development tools to support true product line engineering, methods such as KobrA and CoPAM were introduced in this chapter. Both of these methods support the notion of component-based software engineering as the foundation for derivative products from a common software application frame.

We also explored the use of the platform concept within product line engineering, as this concept has increased in popularity since early 2000. It seems to be a frequently used concept in recent software product line engineering literature. The most common definition of a platform was "a platform is a construct that consists of software assets (or software components) that will be reused across different products." The role of component-based development was also evident in our literature research. We argue that it will increase in the future. It is also very important to realize that component-based development as such does not guarantee successful software application frame development. We believe that development must be a combination of engineering approaches, such as software product line engineering used together with component-based software development. Part of software application frame development is a change in mental attitudes and procedures, together with their associated development processes. This was clearly articulated in the work of Atkinson et al. (2000) and America et al. (2000).

A common thread in software component development with software reuse and software application frame development using common software assets is the aim to reuse underlying artifacts (or software assets) that are part of the a platform. Jacobsen et al. (1997) noted that software reuse can be divided into different abstraction levels by defining concepts, Application Family Engineering (AFE) and Application System Engineering (ASE), which provide the means to plan derivative products with common functionality, enabling software organizations to reuse software assets across product families and product lines. Several authors within the software product line software domain conclude that Jacobsen's work is the prelude to current software product line development and the basis for large-scale reuse within software development. We introduced domain engineering as the approach to identifying commonalties across different products within a product line/product family.

Part of our aim in this chapter was to identify different granularity levels for software components within a software application frame. The article from Bosch (2002) showed maturity levels for software product lines and how independent products with corresponding infrastructure are linked with the platform concept, the software product line, and the configurable product base. This article shows different granularity levels for software product line development. A platform is the layer preceding software product lines. Therefore, software product line development is the means of achieving effective leverage from a common platform with common assets. It was also evident in our literature research that larger granularity levels in software components will enable more effective reuse. This was discussed in several books and published articles. According to several case studies, object-oriented application frameworks have been widely used as underlying software assets in software product line development.

It also became evident in our literature research that the role of software architecture is important when building software architecture for current and future product requirements and features in a product line. It was also clear to us that the implementation of a software application frame must be based on large-scale reuse with coarse-grained software assets (or components). This very same aim has been defined by the software product line engineering community. There was limited discussion of how a software vendor can implement a product line using a price/performance matrix as presented by Meyer and Seliger (1998) and Sääksjärvi (1998).

Existing software related product platform literature includes discussions about product line or family development, but not specifically about different variation techniques. An exception is Meyer and Zack (1996) in their article of information products. We concluded in our analysis of analytical application solution variation that these types of solutions can have variation in three different ways. First of all, analytical applications can be varied purely based on the contents of the analytical application defining technology (meta data and business analytics repository), whereby the software application itself (binary code) will remain the same in each end user organization: only the contents will differ from one vertical market segment to the other. The second type of variation relates to variation that was described in existing product line software literature. This type of variation requires more from the analytical application software vendor. The third variation technique is more or less based on market segmentation, whereby the same software solution is offered to different market segments without really addressing specifically any market segment. This type of market segmentation can be implemented using a price/performance matrix together with license key management. Features and functionality can be controlled, allowing the vendor to enable or disable functionality using license keys.

In summary, we were able to provide a contribution to the exiting analytical application software literature by combining new knowledge of software product line development with the result of three main variation techniques: technological variation found in the software product line engineering literature, solution variation using analytical application frame architecture, and market segmentation with corresponding variation techniques using different variation mechanisms.

6. EMPIRICAL RESEARCH DESIGN

The aim of this chapter is to describe how the empirical research design was constructed and carried out in the analytical interpretative single-case study software organization ("Company"). Selected data collection methods are described, including a corresponding reliability and validity discussion.

According to Myers and Avison (2002), various philosophical perspectives can inform qualitative research, and various qualitative research methods can be used with each philosophical perspective. According to the authors (ibid), the selection of research method will influence the way the researcher collects the data.

We have divided this chapter into four different sub-chapters. First of all, we will discuss our selected research methodology. Secondly, we will explain our selected research framework, which includes an explanation of the selected viewpoints for analysis and the selected time periods. Thirdly, we will explain our selected research strategy with its corresponding implications. Finally, we will explain the reliability and validity of the case study research.

6.1 Research Methodology

To be able to test and evaluate our framework, we decided to base our research on an analytical interpretative single-case study (Klein and Myers, 1999). Our research aim was to draw specific implications from the use of a software application frame strategy framework within analytical application software, and to provide a contribution with rich insight (Walsham, 1995a). Our case study research is supported by quantitative analysis using economic metrics from product platform theory. This type of combined research methodology provides a stronger foundation for analyzing the results of our case study research. Interest in interpretive research has increased in recent years within information systems (Walsham, 1995b), and this type of research methodology enables researchers to "better understand human thought and action in social and organizational contexts" (Klein and Myers, 1999). Interpretative research does not predefine dependent or independent variables, but focuses on the full complexity of human sense making as the situation emerges (Kaplan and Maxwell, 1994).

According to Walsham (1995a), an interpretative case study "is often the in-depth case study, where research involves frequent visits to the field site over an extended period of time." Our analysis took place over several years and could not be performed with any other company in a similar software domain due to competitive factors. Orlikowski and Baroudi (1991) state that "instead of the researcher coming to the field with a well-defined set of constructs and instruments with which to measure the social reality, the interpretative researcher attempts to derive his or her constructs from the field by in-depth examination of and exposure to the phenomenon of interest." This characterizes the interpretative study and information systems as constantly changing, where "the organizations are not static and that the relationships between people, organizations, and technology are not fixed but constantly changing" (Klein and Myers, 1999).

According to Yin (2003), a case study is an "empirical inquiry that investigates a contemporary phenomenon within its real-life context: boundaries between the phenomenon and context are not clearly defined and multiple sources of evidence are used." According to Yin (ibid), single-case studies are appropriate when it is a revelatory case where the situation has been previously inaccessible to scientific investigation. Another case is if the study is a critical case for testing a well-formulated theory, and finally if the case is extreme or unique. The researcher has been involved with the case study company for years, and therefore the case can be regarded as unique and extreme, enabling the researcher to explore the Company in depth for years. Due to competitiveness between similar companies, the researchers would not have gained access to other similar companies to perform a multiple-case study. According to Benbasat et al. (1987), single-case study projects are most useful "at the outset of theory generation and late in theory testing." According to Benbasat et al. (ibid), case study strategy is "well-suited to capturing the knowledge of practitioners and developing theories from it," and most of the case studies are exploratory in their nature.

According to Walsham (1995a), interpretative researchers "are not saying to the reader that they are reporting facts; instead, they are reporting their interpretations of other people's interpretations." Researchers can either be outside observers or involved researchers when conducting interpretative case-study research. As this study has continued for several years with the first research report published in 1998 (Sääksjärvi and Salonen, 1998), the researchers of this study have been both outside observers and involved researchers throughout the analysis period. Between 1995 and 1998, the researcher was product development director for the Company. From late 1998 to 2000, he served as an external observer without interaction with the unit of analysis. In early 2000, the researcher became a part of the organization with access to documentation and other materials that were inaccessible before. The authors (ibid) conclude that neither role should be reviewed as that of an objective reporter, and both of these roles have pros and cons. One could argue that this study would not have achieved the depth that it did due to the involvement of the researcher, as was argued by Nandhakumar (1993) when conducting development of an executive information system as a participant observer.

The researcher has to pose the question in his case study analysis of whether theory formulation before data collection will impact research questions posed in the research of the case study organization. According to grounded theory of Glauser and Strauss (1967), theory should be driven directly from field data, while Walsham (1995a) argues that "it is possible to access existing knowledge of theory in a particular subject domain without being trapped in the view that it represents final truth in that area." According to Walsham (1995a) the "key question for a researcher in any tradition, regardless of philosophical stance, concerns the role of theory in their research." Eisenhardt (1989) identifies three distinct uses of theory:

- As an initial guide to design data collection
- As part of an iterative process of data collection and analysis
- As a final product of the research.

According to Walsham (1995a), the use of theory in the initial stages of interpretative case study research is to create "a theoretical framework which takes account of previous knowledge, and which creates a sensible theoretical basis to inform the topics and approach of the early empirical work." As our research is based on a single case study company, selection of an appropriate qualitative research method to correspond and measure the external and internal validity and reliability of our framework and the generality of our findings is of importance. Qualitative research methods are designed to help researchers to understand people and the social and cultural contexts within which they live. Qualitative research exists in three main forms, positive, interpretive, or critical. Our selected method is interpretive, which is "aimed at producing an understanding of the context of the information system, and the process whereby the information system influences and is influenced by the context" (Walsham, 1993).

Klein and Myers (1999) have defined seven different principles for evaluating interpretive research. According to the authors, these criteria were defined to helps academics to evaluate information systems case study research when interpretive field research is used as a research methodology. The main and the first principle in interpretive research is the hermeneutic circle, which suggests that "we come to understand a complex whole from preconceptions about the meanings of its parts and their interrelationships" (Klein and Myers, 1999). The authors conclude that "the whole story resulting from the application of the individual principles is greater than the sum of the parts, i.e. the separate application of each principle." These seven principles are divided in the following way:

1. The Fundamental Principle of the Hermeneutic Circle
2. The Principle of Contextualization
3. The Principle of Interaction Between the Researchers and the Subjects
4. The Principle of Abstraction and Generalization
5. The Principle of Dialogical Reasoning
6. The Principle of Multiple Interpretations
7. The Principle of Suspicion.

The second principle reflects the social and historical background of the research setting, helping the audience to see and understand the situation under analysis. Therefore, the researcher should aim to explain and ensure that the intended audience of the work can see how the current situation under investigation emerged.

The third principle explores how the research materials were socially constructed though the interaction between the researchers and participants. During the study of the subject, the researcher should recognize that the participants in the study can be seen both as interpreters and participants, and that this could impact the way the results are interpreted.

The fourth principle of abstraction and generalization portrays the process of generalizing concepts, generating theory, and drawing specific implications from the case presented (Walsham, 1995a). According to Klein and Myers (1999), the aim of interpretive researchers is not to specifically falsify theory, as in using theory more or less as a "sensitizing device."

The fifth principle of dialogical reasoning requires the researcher to "confront his or her preconceptions (prejudices) that guided the original research design with the data that emerge though the research process." This principle emphasizes the need to apply the same principle several times during the research process, as each stage of the research process will provide new information and understanding and therefore the previous stage becomes the prejudice for the next.

The sixth principle of multiple interpretations emphasizes the possibility of different interpretations amongst different participants. This requires extreme sensitivity on the part of the researcher. An example of this would be a case in which two people in our case study would interpret a historical event in two completely different ways. Therefore, the researcher must document all of these different viewpoints in the case study research.

The seventh and final principle – the principle of suspicion – emphasizes that the researcher has to have sensitivity to possible biases and systematic distortions in the research. According to the authors, this type of thinking and principle is the least developed in the IS research literature. The authors conclude that none of these principles can be reviewed and used "a la carte," but must be adjusted and evaluated in any particular situation.

Finally, once the research for the case study company has been accomplished, the results of the analysis should be reviewed in the light of these seven principles. The aim of this review is to validate our research strategy, to validate the results of our research, and to ensure that none of these seven principles was ignored during the study.

6.2 Framework of Analysis

The theoretical foundation for our research is based on software related product platform theory, specifically within the analytical application software domain. We discussed in Chapter 2 the limitations of product platform theory as borrowed from mechanical engineering. Due to these limitations, Sääksjärvi (2002) defined a Software Application Frame Strategy framework that addresses the complexity of software development without empirical consideration. The aim of our research is to adapt this generic framework to analytical application software solutions by using the three sub-strategies defined in Chapter 4. We also concluded that an Analytical Application Frame Strategy framework has six different alignment perspectives that a software vendor can utilize, and that each of these perspectives has different characteristics and outcomes within software development. To maximize analytical application frame leverage using the product family solution development approach, a software vendor must carefully weigh the advantages and disadvantages of each alignment perspective in achieving its business strategy aims. Different variation techniques were presented in Chapter 5. Some of these are specific to an analytical application software vendor and and some are generic to any software domain (such as software engineering related variation techniques).

We have analyzed our Analytical Application Frame Strategy framework throughout this study from three different perspectives that corresponds to the three sub-strategies within our framework in the following way (see Table 4):

	Sub-Strategies for an Analytical Application Frame Strategy framework				
Software Application Frame Release	**Analytical Application Frame Technology Sub-Strategy**	**Analytical Application Frame Architecture Sub-Strategy**	**Analytical Application Frame Leverage Sub-Strategy**	**Dominant Alignment Perspective**	**Evidence of Alignment Perspective Disconnect**

Table 4. Framework of Analysis.

The first column lists all the different analytical application frame releases that could be identified in the case study company based on the four principles set in Chapter 2. The next three columns portray the three sub-strategies identified within an Analytical Application Frame Strategy framework in following way:

Analytical Application Frame Architecture Sub-Strategy: Portrays how the case study company defined its analytical application frame architecture.

Analytical Application Frame Technology Sub-Strategy: Portrays how the case study company defined its technological selections with corresponding elements such as underlying technology, execution and development environment, and software development approach.

Analytical Application Frame Leverage Sub-Strategy: Portrays how the company selected its market segments and how the two other sub-strategies supported selected market segmentation.

The last two columns in Table 4 express how the case study analysis was able to demonstrate a dominant alignment perspective and whether there were signs or evidence of alignment perspective disconnect. We have defined a disconnect as a weakness in alignment perspectives with respect to one sub-strategy that has been neglected or received less attention. The analysis of different alignment perspectives with corresponding fits is implemented by carefully examining each analytical application frame release and what type of changes occurred to each of the three sub-strategies during the analysis period.

Once all three sub-strategies are analyzed and reported, the next step is to explore whether any of the six identified alignment perspectives has been used by the case study company and what type of results these alignment perspectives have had both for short- and long-term product development strategies. Based on this analysis, together with financial data and project related data, we will use quantitative research methods to calculate longitudinal analytical application frame effectiveness and efficiency. These numbers will demonstrate whether the case study company has been successful in its product development in the light of analytical application frame development. We are interested to see whether the case study company would have benefited by having strong competence in defining an analytical application frame and its frame components and what kind of results could have been expected with effective use of the framework.

We have divided our case study research into two main analysis periods. The first analysis period is from the foundation to 1998 and the other analysis period is from 1998 to 2002. The reason for dividing the analysis period into two periods is twofold. First of all, the researcher left the Company to work in a United States subsidiary, causing the connection between the participants and the researcher to change from a social perspective. Secondly, during 1998 a research report was published (Sääksjärvi, 1998) whereby the case study company became aware of the software application frame concept. Our aim is to explore whether this had any impact on the development of software releases after 1998. We can conclude that after 1998 the case study company became aware of the possibilities that a software vendor can achieve when using the software application frame development approach.

The theoretical research that has been published about product platform related software development is limited; the case study sheds some light in the difficulties that software engineering will pose for a software vendor and how different development of software products is compared with development of physical artifacts in a manufacturing environment. The software solution from the case study company is based on a client/server architecture using iSeries/400 hardware environment with a corresponding Windows software client. Our analysis will include discussion of different software releases and corresponding analytical application frame releases that can be identified during the evolution of the software products within the case study company.

The quantitative research method is based on efficiency and effectiveness measures defined by Myers et al. (1997). The materials for these measurements were collected using historical records of worldwide product specific sales a well as cost information provided by the financial department of the case study company.

6.3 Selected Research Strategy and its Implementation

The selected research strategy is built to reflect the principles of Klein and Myers (1999) with the idea of the hermeneutic circle, where "the complex whole arises from preconceptions about the meanings of its parts and their interrelationships." The idea of the hermeneutic circle was implemented by studying each sub-strategy within an Analytical Application Frame Strategy framework separately, which gave a better understanding of the whole by interviewing each organization group separately during several sessions. Each sub-strategy within an Analytical Application Frame Strategy framework represents different types of interest groups with their respective aims. By studying and interviewing each group separately and combining this information, we achieved a better understanding of the whole in the form of interaction between these three different alignment perspectives. An example of this is the product development group, with the responsibility to create a domain-specific architectural model, and the technologists group, whose aim is to define the underlying technology for the analytical application software solution. If the cooperation between these two groups is uncoordinated, the "Technological Responsiveness" fit could be adversely impacted. In a similar manner, personnel in marketing must have a coordinated discussion with both software technologists and software architects to avoid either "Functional Flexibility" or "Technological Adaptation" imbalances. In summary, once we achieved better understanding of the parts (sub-strategies), they themselves helped "to codetermine the meaning of the whole" (Gadamer, 1976).

According to Klein and Myers (1999), the principle of contextualization requires that "the subject matter be set in its social and historical context so that the intended audience can see how the current situation under investigation emerged." The principle of contextualization and interpretive research emphasizes that "people, organizations, and technology are not fixed but constantly changing." We could clearly see this in our longitudinal case study research during the last 15 years. The case study company underwent several profound technological changes during the evolution of the software offering. There were several organizational changes during the years, and each and every change impacted the historical evolution of the case study company. Positivist research presumes that historical patterns are repeated in the future, while interpretive research perceives historical events to be historical. Future events are a combination of circumstances that will drive future changes, and historical events are not always repeated. In retrospect, when viewing the historical evolution of the case study company, it is clear that the historical evolution has been a combination of technological changes in the IT market, changes in the end user perception of software functionality, and careful planning from the management of the company as to which market segments/ application areas needs to be addressed. This reflects back to the hermeneutical principle, where cycles of interviews of each sub-strategy gave us a better understanding of the overall software business model of the company.

The principle of interaction between the researcher(s) and the subjects is very challenging for any researcher, as interpretative researchers “must recognize that the participants, just as much as the researcher, can be seen as interpreters and analysts” (Klein and Myers, ibid). The researcher has been with the organization for close to 10 years with the exception of being located in Dallas, Texas, away from the parent company, since 1998. This research project was initiated back in 1997. The Company was led by the founding CEO until 2000, when there was a change in management. The researcher was operationally responsible for the product development organization until 1998 and as Chief Technology Officer from 2000 to 2002. The researcher has relied on historical secondary data from the time period of the foundation of the company until 1995. From that point going forward, the researcher was active as a participant in the organization. As Klein and Myers (1999) conclude, even if the researcher is dependent on secondary historical data, “the researcher’s preconceptions about the participants still affect the construction, documentation, and organization of the material.” Most of the interaction between the researcher and participants was through interviews (physical presence) where the questions were predefined and the researcher took notes during the interview. Some of the questions were posed as email messages (during the period 1998 to 2002), due to physical distance of the researcher and the participants. Additional interviews were performed as telephone interviews to verify (triangulate) the results that were given as part of an email message. Finally, one of the founding developers immigrated to United States in 2002, and the researcher was able to have additional interview sessions with the developer and confirm some of the results that were concluded from the case study research.

The principle of abstraction and generalization emphasized the importance that “theoretical abstractions and generalizations should be carefully related to the field study details as they were experienced and/or collected by the researcher” (Klein and Myers, 1999). Our theoretical framework of Analytical Application Frame Strategy includes presumptions of the impact to analytical application software development when different alignment perspectives are used. These presumptions are tested against a single-case case study company to highlight the use of the framework and each of the included sub-strategies. We concluded earlier in this chapter four different types of generalizations of theory (Walsham, 1995a). Out of these four types, this research concentrates on two types of generalization. The first is based on drawing specific implications of the use of an Analytical Application Frame Strategy framework and its different alignment perspectives. The second type is based on providing rich insight into the use of the framework, specifically in the case of analytical application software solutions. Our analysis of the Software Application Frame Strategy framework construct defined by Sääksjärvi (2002) is retrofitted to include elements of analytical application software solutions, and our research analysis is purely from an analytical application software perspective. We argue that the Software Application Frame Strategy framework has to be adjusted to reflect the given software domain, as each software domain is different as to its characteristics. The reason for this is the difference between each sub-strategy and elements within these sub-strategies. These elements could be very different between different software domains, such as the selection of underlying and defining technology.

The principle of dialogical reasoning is very hard to verify, as "the principle requires the researcher to confront his or her preconceptions (prejudices) that guided the original research design with the data that emerge through the research process" (Klein and Myers, 1999). This principle requires the reader to achieve an understanding of the researcher's own historicity and how each prejudice could become the source for understanding the next stage of the research. This principle is challenging to implement, as the researcher must separate his/her own historical experience, preconceptions, and intellectual background. Each of these will have an impact on how the researcher approaches the research objects, the participants, and how research questions are laid out. Analysis of the findings of an interview will be impacted by the background of the researcher. The researcher has been actively working within the decision support software domain for the last fifteen years, and this has had an impact on the overall research setting and understanding of the problems in software development of analytical applications. The researcher has been actively working within the different sub-strategy domains within the Analytical Application Frame Strategy framework, and this helps to understand the interrelationships between marketing (analytical application frame leverage), the domain-specific product architecture (analytical application frame architecture), and technological choices (analytical application frame technology). Without this understanding, it could be very difficult to set relevant research questions on the one hand and on the other hand draw conclusions from the case study research and the implications between different stakeholders and their social setting.

The principal of multiple interpretations could be identified in the research due to different stakeholders and their interests. Some of the historical evolution was explained differently by the management and the developers. The reason for this could be either that the stakeholder wanted to see the event in a specific light or that the participants had forgotten the factors leading to a specific decision. Most of the findings during our research were well aligned with all of the participants. In cases where there was a difference, we used different types of documentation to verify what really happened. If the questions were the same for two different groups of stakeholders, we compared the answers for each group. If there were any conflicts in the answers, additional questions were posed.

The principle of suspicion requires the researcher to consciously suspect systematic distortions or biases in the analytic process. According to Klein and Myers (1999), this principle appears to be the most underdeveloped principle within the IS research literature. The researcher should critically view any of the statements or findings given by the participants, as various stakeholders could have different aims and interests in the research outcome. According to Klein and Myers (ibid), in some cases strong management could impact the statements and information given by other participants in the study due to different reasons such as fear, conscious political acts, or other psychological reasons. Within the case study company, the founding CEO is known for a strong personal drive and sometimes intimidating personality. This type of knowledge was taken into consideration in the case study research, as some of the findings might have been neutralized by some of the interviewees. As Myer and Klein (ibid) argue, the social and political relationships between different actors within the case study research can impact the outcome of the results. Even if the researcher has been actively involved within the organization for close to ten years, some bias could be present during the "learned" predefined conceptions that the case study company has maintained during the ears. In some cases, some public statements could become "the truth" for the participants because they want to believe them to be true even if the case were far from the reality.

Finally, the interdependence of each of the seven principles will impact the overall results of the research, as described by Klein and Myers (ibid). Each case study is different in its characteristics, and therefore each of the seven principles can have a different impact on the outcome of the study. Klein and Myer (ibid) emphasize that the researcher should not arbitrarily leave any of the principles out without explaining the reason for it. In some cases even Klein and Myers (ibid) approve of a principle being left out, such as in the case of the principle of suspicion. This principle goes beyond the understanding of case study data to the understanding of the social world, where the actors have their social preconception of events and historical happenings for the case study company.

All revenue and cost information in the following charts and tables is reported in the unit of currency used in the case study company's headquarters country. To protect the proprietary information of the case study company (which is privately held), we will report all currency amounts without identifying the name of the currency unit itself. All of the numbers reported represent correct proportions: all analytical application frame efficiency and effectiveness calculations are reported accurately.

6.4 Data Collection Methods

According to Orlikowski and Baroudi (1991), interpretative research is not aiming to generalize from the setting, but the intent is rather to "understand the deeper structure of a phenomenon, which it is believed can then be used to inform other settings" (ibid). We have based our research strategy both on qualitative and some quantitative research methods to provide triangulation for the results found in the study. Some researchers (Gabe, 1994; Kaplan and Ducheon, 1988) argue that a combination of research methods within one study enables stronger triangulation than using only one research method. Jick (1979) explains several important opportunities that triangulation can provide a researcher, such as allowing the researcher to be more confident in the research results. It can stimulate the creation of inventive methods, etc. Therefore, quantitative methods were added to the research by applying Myers et al. (1997) product platform effectiveness and efficiency calculations to the case study research. Both research methods together can provide an enriched explanation of the research problem at hand. The study also employed ethnographic techniques such as observation of participants, researcher interaction with the study objects, documentation review, social contact, unstructured interviews, and structured interviews. The study was executed over several years as portrayed by Orlikowski and Baroudi (1991).

According to Yin (2003), evidence for case studies may come from six different sources: documents, archival records, interviews, direct observation, participant observation, and physical artifacts. The Company has historically externalized most of the client software development to an external software development organization, and therefore the collection of financial records is more accurate when comparing the calculations of internal records of costing information. These external development organizations have been tightly integrated into the core development organization, and all the numbers presented in the calculations include the use of external development organizations, both for server and client software development. Walsham (1995a) concludes that within interpretative case studies, it is particularly important to pay attention to how fieldwork is reported and show how the author has arrived at the results of the research. Walsham (1995a) includes discussion of the minimum requirements of explaining how the researcher performed in following way:

> "As a minimum, reporting on the collection of field data should include details of the research sites chosen, the reason for this choice, the number of people who were interviewed, what hierarchical or professional positions they occupied, what other data sources were used, and over what period the research was conducted."

According to Eisenhardt (1989), overlapping data analysis with data collection provides a number of advantages, giving the researcher a head start in analysis. The analysis of the case study company took place over several years, and several iterations of case writing were made. The skills of the researcher were improved along the way as the understanding of the different implications of different alignment perspectives became clearer. Additional literature analysis of product platform related theory and software product line/family development literature gave the researcher a better understanding of the complexities involved in software development, adding to the researcher's long background in managing software development teams using a component-based software development approach. Our case study material included several different types of documentation sources that were the foundation for our case study analysis. We have divided the documentation into six different categories (see Table 5):

Documentation	**Description**
1. Financial records of the company	Financial statements included annual reports from the foundation of the company until end of 2002: • Income statements • Balance sheets
2. Product development documentation	Project reports and access to a Lotus Domino-based database that enabled us to view historical information by • Project, by developer, by time. Each of these views had associated costs included
3. Internal documentation	Product strategy documents, internal memos, internal Microsoft PowerPoint presentations from different stakeholders
4. External documentation	• Annual user group meeting documentation (slide shows, handouts etc.) • User manuals (both designer's and end users' manuals) with "What's new section" • Installation documentation
5. Internal software tracking application	The Company provided access to all internal tracking applications that are used by the Board of Directors, executive management, and development and sales management. Analytical applications (based on the Company's own applications) • Invoicing database • Contracts database • Financial statements database • Software license database
6. Documentation created during the research	• Microsoft Visio documents (figures, tables) • Microsoft Excel documents (collection of product related financial information)

Table 5. Case Study Documentation.

We had full access to financial documentation from the foundation of the Company until mid-2003. This documentation included all financial annual reports, monthly and quarterly reports, and associated reports. The second category represents product development documentation that includes internal design documents, internal memos of product plans, component model, and other associated documentation. The third category represents internal documents such as product development strategy documents, internal documents that have been used in training business partners and internal personnel. The fourth category represents external documentation such as user manuals, user group documentation, and other documents that represent the company and its products to the market. The fifth category represents all software applications that the company uses internally to track its product development, financial performance, and sales around the world by business-partner.

Due to geographical distance, some of the interviews were conducted using conference calls and emails that were collected in a centralized database. All financial data were recorded in Microsoft Excel spreadsheets. Additional questions about the data resulted in several iterations of these results. The researcher's understanding of the complexity developed during this iterative process as portrayed by Klein and Myers (1999). All interview questions were written before the interview, and the researcher took notes during the interview process. All of our email correspondence related to the data with questions were collected and saved for later reviews.

Even though the researcher had extensive background in the case study company both as an outside observer and as an involved researcher, the in-depth analysis of the development organization during the last 15 years resulted in research findings that were a big surprise even to the founders and key developers of the Company. The case study company had not previously measured or calculated gross margins on any of the products that have been developed during the years. Only through extensive research by the chief financial officer and controller via multiple interviews and passes of the results were gross margin numbers generated and revealed. Another factor that changed the overall landscape of data collection was the departure of the founding Chief Executive Officer of the Company. Social and political issues prohibited the researcher from gaining access to all detail information, as some information was treated as confidential or sensitive (see also Walsham, 1995a).

The case study analysis also consisted of numerous in-depth interviews between 1996 and 2002 (see Appendix 3) with the management and development organization of the Company. Several interviews were performed on-site with the original developers and founders of the Company. Additional questions to confirm results of the interviews were asked and answered either by emails or by additional phone interviews. To achieve triangulation of the results, comparisons were made to old documentation such as product manuals, user conference notes, and other documents. These interviews, combined with product documentation, enabled to us to identify all major software releases that the Company had released during its lifetime. The aim of the interviews was also to explore and identify how the product development organization had defined application specific architectures and how technology selections had been made with respect to the Company's market segmentation strategy (if any). The aim of the literature research, using old documentation, was to identify the key marketing and selling points that the Company had at each product release. From this documentation and product release specific aims, we could identify major product releases and whether these had been major or minor upgrades (see also Appendices 2 and 4). All of the analysis was tabularized in Microsoft Excel spreadsheets, Microsoft Vision graphics images, or Microsoft Word documents. The study also has a database of the historical evolution of the case study analysis since the first report was released in 1998 (Sääksjärvi and Salonen, 1998).

To be able to perform needed effectiveness and efficiency calculations as defined in Chapter 2, the Chief Financial Officer of the case study company provided extensive sales information and product development costing information for the analysis from the foundation of the Company until end of 2002. These numbers included information about how the numbers have been derived and what kind of development cost averages had been used when evaluating the R&D costs for the years that did not have any mechanisms for tracking product development costs. During the study, the researcher discovered the difficulties in collecting financial records of product development costs, as the Company had weak internal processes to collect project and financial data in the early years of software development. To be able to measure software development costs, the researcher had to make approximations that could in some cases be inaccurate and lead to false conclusions. First of all, historical sales revenue per product was allocated with a 70/30 rule, whereby 70 percent of the sales were allocated to the server software product and 30 percent to the client software product. For the historical cost calculations, we used personnel costs as a basis, as well as external invoicing from third-party development organizations. To be able to calculate software application frame effectiveness and efficiency, we analyzed each software application frame separately (but combined frame effectiveness and efficiency numbers are also presented in the analysis).

According to the Chief Financial Officer (CFO), even if the ten years accounting rule for keeping records of old accounting vouchers is already passed, he had access to old financial records that were presented to the shareholders on an annual basis. We also had access to old product documentation that was kept by one of the founding members of the case study company. This provided us with detailed information about how each software product release had evolved during the years. Old presentation materials from user group meetings were accessible in the analysis of both server and client frame releases.

Triangulation of the collected data (financial and project data) and initial analysis results were used to verify in separate interviews with the development managers of the company that the estimates proposed by the CFO of the case study company were in line with reality. To be able to measure the success of any software product, the accounting organization must match the accrued sales revenues with the costs accrued in the products and respective software application frames. Without this matching, calculations of software application frame effectiveness and efficiency are very error prone and leave too much room for incorrect results. We decided to include all known R&D costs and associated product sales costs when calculating software application frame effectiveness. We were able to analyze all product related sales information by using an internal sales tracking application that is used in Board of Directors reporting. This sales profitability application is now used on a regular basis within the Company, and all product related data is automatically collected on monthly basis. The results of the case study caused the management of the Company to initiate the collection of information on a more regular basis.

6.5 Reliability and Validity of the Study

The researcher gained additional understanding of the research domain during the iterative data collection and analysis of the data. According to Walsham (1995a), interviews are "the primary data source since it is through this method that the researcher can best access the interpretations that participants have regarding the actions and events which have or are taking place, and the views and aspirations of themselves and other participants." As our research method is based on a qualitative interpretative case study combined with quantitative analysis, the aim of this chapter is to evaluate each of Klein and Myers' (1999) seven principles when conducting field studies in information systems.

According to Klein and Myers, without these seven principles, each interpretative researcher would have to "spend considerable time deriving the theoretical foundations for their research from diverse literature sources." These principles enable researchers to defend their work, as these principles are firmly grounded in a major direction of interpretative research, and the authors do not have to rely on and use inappropriate positivist criteria. These principles enable external reviewers of the research to verify that none of the seven principles are left out arbitrarily and give validity for the overall research. The main idea behind the hermeneutic circle is to provide to researchers an overall value that is larger than the sum of the parts. Each part will enable the researcher to understand the research results in a different light, rather than just studying the research object as a whole. The conclusions are derived by an iterative process, and as the case study research has lasted for several years (the first research report was published in 1998), the researcher has written and analyzed the case study analysis multiple times. Each time, he has gained more understanding in the product platform related theory, specifically with respect to software development.

All six other principles besides the fundamental principle of the hermeneutic circle are the foundation for evaluating interpretive field studies in information systems. The fundamental principle of the hermeneutic circle was applied in an iterative manner with multiple interviews as well as documentation reviews with corresponding financial and project data. To enable a better understanding of the software development process, we conducted several interviews with both executive management and senior development managers to ensure the validity of our findings. We were also able to break down the data collected into product specific information which had not existed before. This enabled the management and the board to review the profitability and effectiveness and efficiency of the analytical application frame development within the case study company. It was also evident that this breakdown of information into smaller pieces enabled us to achieve a better understanding of the whole and the complexities surrounding the analytical application frame software development.

The analysis of each and every principle leads to a more complete understanding of how an analytical application frame can be applied to analytical application software and how this has been applied in the case study company. The case study research showed also the importance of critical reflection into how software vendors should evaluate a product release from an analytical application frame perspective. We identified difficulties in deciding how each new product release should be evaluated and whether two or more software releases belonged to the same analytical application frame release. We had the same difficulty when we had to evaluate whether an analytical application frame was renewed or merely extended. The case study has enabled us to achieve a better understanding of product platform related analytical application development with its software related complexities. This type of research did not exist before, specifically with respect to analytical application software.

The applicability of the second principle – the principle of contextualization – requires "that the subject matter is set in its social and historical context so that the intended audience can see how the current situation under investigation emerged" (Klein and Myers, 1999). This principle is problematic for several reasons, including changes in the management of the Company and other relationships that the researcher has been able to experience during the last ten years. The case study company was founded by a very strong entrepreneur who led the Company over the years, and several of the decisions were made by the founder. By interviewing other participants in the decision making, the researcher was able to reflect the impact of the social and historical environment at the time.

As for the third principle – the principle of interaction between the researchers and the subjects: the research was a combination of in-depth interviews concerning the subject and the collection of old product documentation and strategy documents presented in the Company's user group meetings. The researcher has been actively involved with both the product development organization and the management of the organization. The object of the most critical part of our study – renewal of the client frame – was implemented without the participation of the researcher, and therefore the research could be viewed both critically and objectively. All research materials were collected, and conflicting documentation or interview results where compared with additional materials or interviews.

The research results from prior product platform theory did not include deep discussion of software related issues around product platform development. Issues such as defining a software application frame for a software product, granularity of a software application frame, dependability of selected technology, and several other issues were completely left out of the discussion in existing literature. Our analysis of the analytical application software domain does not necessarily apply to any software application domain, as some products have extreme characteristics (embedded software in mobile phones, microwave ovens etc.)

Because of this analysis, we argue that we have been able to include necessary elements to comply with the principle of abstraction and generalization.

The main idea behind the principle of dialogical reasoning is that "the researcher should make the historical intellectual basis of the research as transparent as possible to the reader" (Klein and Myers, 1999). This principle is problematic, as the researcher has been involved with the Company and has 15 years of experience in the application domain. Even if a strong attempt was made during the analysis of not letting preconceived experiences reflect the research, an attempt was made to separate the researcher's own ideas from the actual happenings via additional documentation and multiple interviews.

As our research was based on single case interpretative case study, we emphasized strongly the principle of multiple interpretations. First of all, the case study company has evolved in several different stages during its lifetime – from a highly entrepreneurial company to a professionally run international organization with an aggressive focus on third-party development partners. This evolution has enabled a review of different perspectives of the various stakeholders in the study. Conflicting interpretations were verified by additional interviews, and in some cases against additional documentation. Especially during the initial years with strong leadership from the CEO, it was important to get the viewpoints of other founding partners, as the agenda for the CEO could have been different from that of the other stakeholders. For the time period with aggressive growth in personnel (from 1995 to 1998), when the researcher was part of the development organization, the research documentation and decisions where confirmed by both the management team and the development organization. The final analysis period results were more or less based on several interviews and additional documentation produced by both the case study participants and a third-party development organization.

Due to the strong leadership of the founding CEO of the company, the final principle – the principle of suspicion – is of importance to the research evaluation. First of all, all product releases with their corresponding software application frames where confirmed by several founding partners. These interviews where triangulated using supporting documentation. Secondly, several interviews revealed that several decisions regarding both technology and market segmentation were influenced by the CEO of the company without consulting or reflecting on ideas and arguments from other people in the organization. This type of social interaction can be typical for entrepreneurial organizations. This puts additional emphasis on viewing the actions and statements of the actors within the case study company.

7. ANALYTICAL APPLICATION DEVELOPMENT IN A CASE STUDY COMPANY

The aim of this chapter is to perform a longitudinal single-case analysis of a European analytical application software vendor ("Company") in the light of our Analytical Application Frame Strategy framework, defined in Chapter 4. We have divided our analysis into several different sub-chapters. The overall research strategy is based on qualitative interpretative research, using Klein and Myers' (1999) set of principles for conducting and evaluating interpretative field studies in information systems as described in the previous chapter.

We have divided our case analysis into two analysis periods, where the first period ranges from the foundation of the Company to 1998, while the second analysis period ranges from 1998 to 2002. We will analyze each sub-strategy within the Analytical Application Frame Strategy framework separately for both analysis periods to achieve an understanding of how the analytical application frame architecture was built and what type of technological and market segmentation strategies were selected. This analysis will be the basis for our quantitative analysis of software application frame efficiency and effectiveness performance for each analysis period.

We will initiate our discussion by defining some key concepts in our analysis, as well as our analysis approach in this chapter. Once this has been accomplished, we will analyze the case study company using the Analytical Application Frame Strategy framework as the foundation. Finally, we will explain our findings in light of our research questions, suggesting recommendations for software vendors within the analytical application software domain.

7.1 Definitions and Analysis Approach

We noted in previous chapters that we use "software application frame" when referring to software application frame development in generic terms without reflecting any given software domain. In our case study analysis, we have divided our software application frame analysis into two software application frames. The case study company has two main software modules, one that is installed on the iSeries/400 environment and the other installed in client workstations. The reason for this separation is the different technological foundation for each of the software modules. The server software is implemented using iSeries/400 technology, while the client software module is a Windows application with a corresponding execution and integrated development environment. From a commercial perspective, both software solutions are sold together. Therefore, we will include software application frame efficiency and effectiveness calculations for each software application frame (client and server) individually and together. We have defined the software application frame for the server solution as the "server frame" and the corresponding software application frame for the client solution as the "client frame".

Our analysis is divided according to the three separate sub-strategies within an Analytical Application Frame Strategy framework. Each sub-strategy is analyzed according to the discussion introduced in Chapter 4. This discussion will include analysis of whether any alignment perspective has been dominant within the selected analysis period and what implications this has had for the case study company. We divided our case study analysis into two separate analysis periods. The first period represents a time period in which the case study company was more or less unaware of the possibilities of the software application frame concept. The second analysis period represents a time period in which the development organization actively implemented derivative products using a client frame. Another reason for the time period selection is the clear renewal of both server and client frame versions with server frame V2, released during 1997-1998. A new client frame was implemented during 1998-1999. The initial server and client frames had scalability problems, and the client frame had architectural deficiencies that could not be fixed by amending the existing client frame. The development team concluded that the only way to continue adding features and functionality to the client frame was to renew the whole client frame.

The CEO at the time explained in our interview in early 1998 the following about the client and server frames and their architectural structure and performance:

> "It seems to have become a cyclical problem for us with these two different application frames. Once we are able to remove the bottlenecks from the server frame, we seem to run into the same problem in the client frame and vice versa."

Interestingly, the renewal of the server frame was initiated roughly one year prior to the renewal of the client frame, and this was exactly what the CEO referred to in the quotation above. The new server frame was improved to enable larger databases and perform data loads more efficiently. Soon thereafter, the development team initiated the renewal of the client frame.

7.2 Background of the Case Study Organization

Our case study company was founded in 1987 by an entrepreneur who had the vision to create a management software solution for IBM midrange end user organizations. The Company is privately held, specializing in horizontal data warehouse and analytical application solution development for organizations with IBM iSeries/400 technology. Over the years, hundreds of organizations in different vertical market segments, such as manufacturing, insurance, finance, distribution, etc. have chosen to run their management reporting using the software solutions from the case study company.

The Company is headquartered in Helsinki, Finland, with direct operations in the US and Sweden. The Company has a strong business-partner strategy (channel strategy), with a partner network both in the US and Europe. It has recently (2002) opened its South American marketplace. The Company has based its product development and product offerings on the award winning iSeries/400 hardware platform (previously known as AS/400), with an original market segment of small-and medium sized organizations. The performance improvements in Series/400 server technology, together with new market segmentation strategies, have enabled the Company to offer its solutions to larger organizations as well. IBM Corporation rates the case study company as a key player within the iSeries/400 business intelligence market, and the Company has received several awards due to its product development efforts. The Company was also included in the famous IBM Red Book series in the mid 1990s (Chilanti, 1997) that rated and compared different serious business intelligence vendors. Part of the overall strategy for the company has been to form relationships with universities to perform basic research into the development of new technologies that can be applied within the decision support domain.

The case study company has used outsourcing throughout its entire history, both in server and client software development. The initial server software solution was implemented jointly with a third-party organization. The result was not impressive: it had to be redeveloped by the case study company. The first client frame was implemented by a small external software organization according to the specifications of the Company. Later, the main developer joined the case study company. The renewal of the client frame was implemented by another third-party software organization. This organization has worked jointly with the Company since 1997. We can conclude that the Company has accrued valuable information about external outsourcing for more than ten years and is in good position to continue with this model in the future as well.

7.2.1 Business Model for the Case Study Company

The Company's business model throughout the years has been to be a software products company. Leverage is achieved by selling the same software package to any firm in any vertical industry without any tailor-made modifications. This packaged approach has enabled a relatively small development organization to implement a product that has been sold to hundreds of end-user organizations around the world, each of these organizations having the same core software solution. The Company is facing a tough decision as the software analytics marketplace matures, with larger software organizations providing similar solutions at a fraction of the traditional price. The question will become whether some of the lost software license revenue will have to be replaced with additional service revenue or whether the Company must transform itself to a hybrid, where the solution is packaged together with a set of solutions that are maintained by the Company. This type of combination could result in lock-in for the end user organization, as it would require continuous support from the software vendor.

The financial performance of the Company has been reasonable during the years, with growth in total revenues and personnel. The early decision by the founders of the Company to become a software products company directed all activities into developing a generic software solution that could be easily implemented in any end user organization. The share of license revenue as a percent of total revenue increased steadily during the years until 1997 (65%). This share has decreased since then, reaching 33% in 2002 (see Fig. 23).

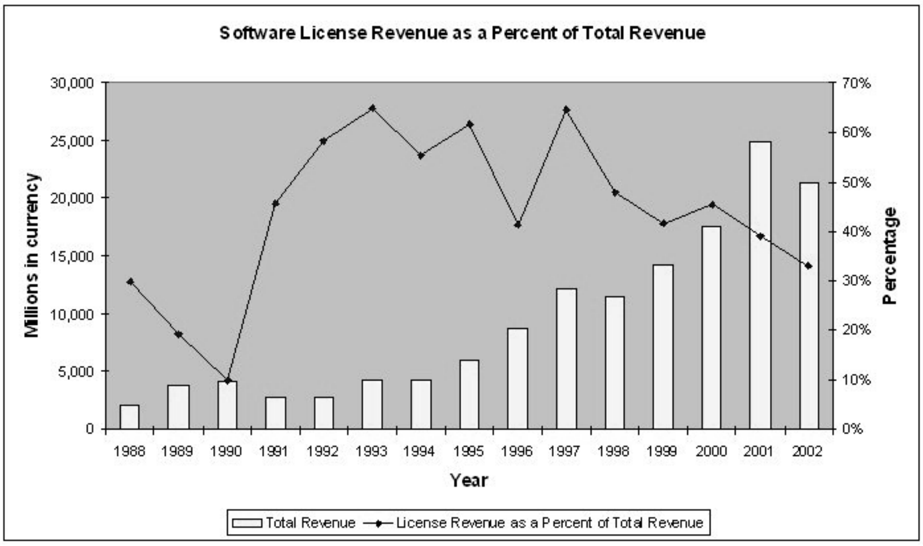


Fig. 23. Total Revenue and Software License Revenue as a Percent of Total Revenue.

The Company has invested aggressively in product development during the years, with two considerable peaks in research and development. The first peak was in 1991 (37%), when the Company invested heavily in redevelopment of the server software solution. The second clear peak was in 1998 (40%), when the client frame was renewed (V2) to reflect the new IT infrastructure requirements (see Fig. 24).

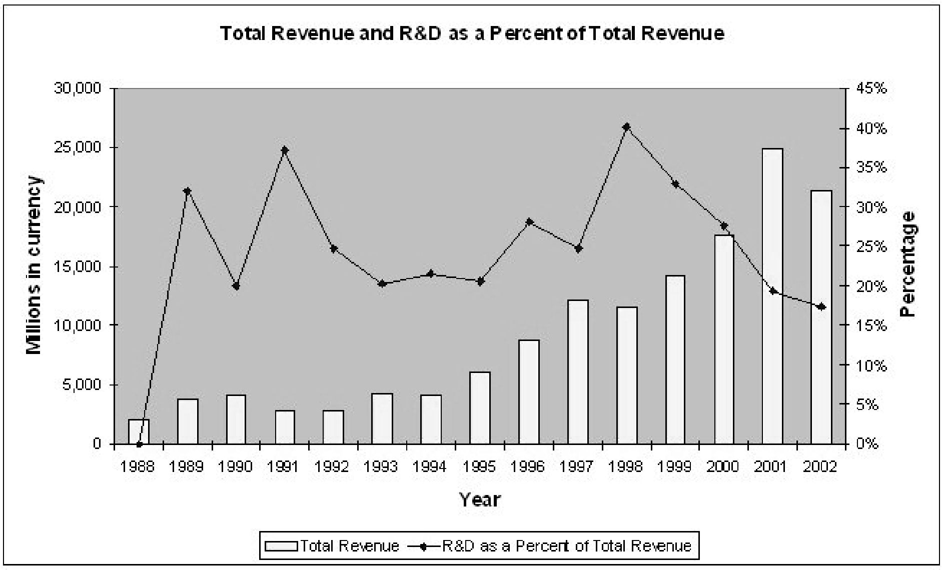


Fig. 24. Total Revenue and R&D as a Percent of Total Revenue.

The challenging economic environment on a global basis has also required the Company to decrease its expenditure on product development in 2002 to meet the decreased global software license sales. Even though the share of research and development costs relative to total revenue decreased, the case study company kept its development budget on higher levels going forward. According to Cusumano (2004), software organizations typically have research and development costs that are between 15 to 20 percent of total revenue. Even in 2002, the Company invested 17 percent of its total revenue in research and development. Overall, the Company has invested heavily in product development over the years (between 14% and 40%), with strong emphasis on technology development.

We concluded earlier in this chapter that the Company wanted to be a software products company, and that software license revenue should be the driving factor when doing business. Initially, the Company undertook a considerable amount of implementation consulting for customers that had bought the initial software release (see Fig. 25).

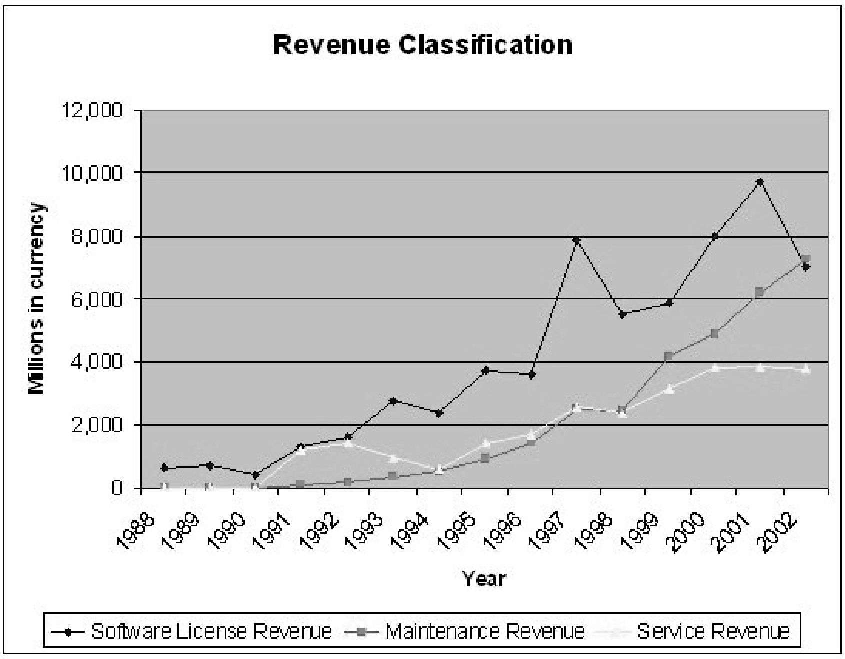


Fig. 25. Revenue Classification.

Initially, the Company had mostly service revenue, but very early (1990), software products revenue became a larger contributor to total revenue. During 1993, there was a clear increase in software license revenue, mainly due to the opening of international markets for the Company. The share of service revenue relative to software license revenue has dropped during the years, mainly due to a larger share of the software license revenue coming from international markets and business partners. The Company does not have knowledge of how much service revenue is generated by international business partners, or of how this will lead to a reduced service revenue share relative to total software revenue. The relationship between maintenance and software license revenue has been stable during the years, with a clear drop in software license revenue in 2002, when these two revenue types were almost equal. Typically, the Company has charged 15% to 20% of the initial software license revenue as maintenance. This maintenance revenue covers the cost of support and continued development of new software product releases.

7.2.2 Software Product Release Analysis Overview

A study of all the releases of both server and client products of the case study company was performed based on product documentation, user group meeting materials, and several repetitive interviews (see also Appendix 4). As the Company was founded in late 1980s, many of the details had been lost in the "bigger picture," but we were able to reconstruct the past using several interviews. The Development Manager for the server (OLAP Server) product went so far as to investigate the source code to see his textual comments in the past releases. Early manuals were also found in private homes, even though the Company had moved its office location several times. All of the software releases (both client and server software releases) are listed in Appendix 2. Figure 26 shows a longitudinal overview of the software product release evolution since 1989 (see Fig. 26).

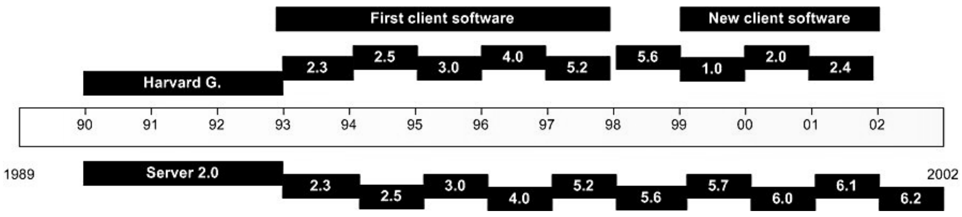


Fig. 26. Server and Client Product Release Milestones.

The initial software product architecture (not shown in Figure 26) consisted of a server software product (1.0) and client with presentation capabilities (Harvard Graphics). The initial software product offering was completely replaced by subsequent releases (server software 2.0 and beyond). Development after 1.0 did not use any of the earlier technology (other than the RPG implementation language). The second server release (Server 2.0) became the foundation for all subsequent server frame releases and was based on iSeries/400 technology, with Harvard Graphics as the front-end software to display information. With integrated software release 2.3, the case study company replaced Harvard Graphics with its own software client that was more integrated with the server software module. Version 2.3 of both the server and the client software became the basis for subsequent releases of the software. Our interview with the server Development Manager revealed that "version 2.3 of the server and client software was strong enough for distribution to the international marketplace." The integrated software solution was initially based on traditional client/server architecture, with two software modules (see Fig. 27).

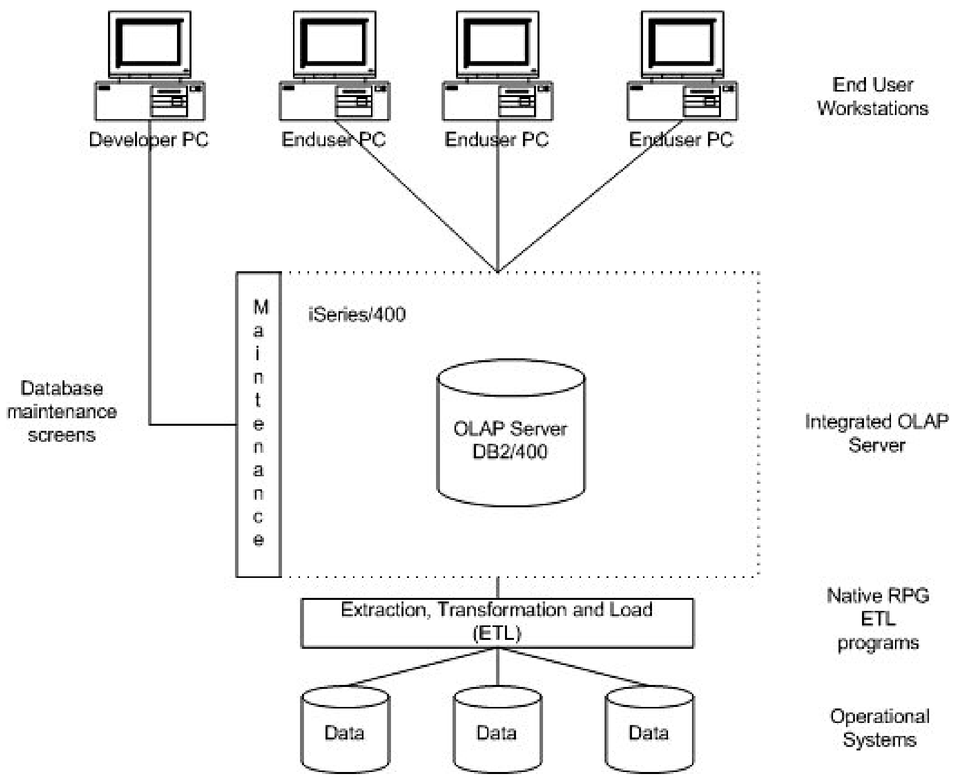


Fig. 27. Original Software Application Architecture.

Figure 27 demonstrates the overall solution offering, with its integrated multidimensional OLAP Server that includes ETL functionality and client software that provides the end user experience. The first integrated release with both the server and the client software module was introduced with version 2.3. The two subsequent integrated software releases (2.5 and 3.0) did not include any specific new functionality. With integrated software release 4.0, several major changes took place. First of all, the Company introduced new software modules for budgeting and planning solutions. Secondly, the client frame communication module was converted from a 16-bit environment to a 32-bit environment. This became a requirement from the IT infrastructure perspective. Due to additional technologies in development, the Company added new complexity into its software development (see Fig. 28).

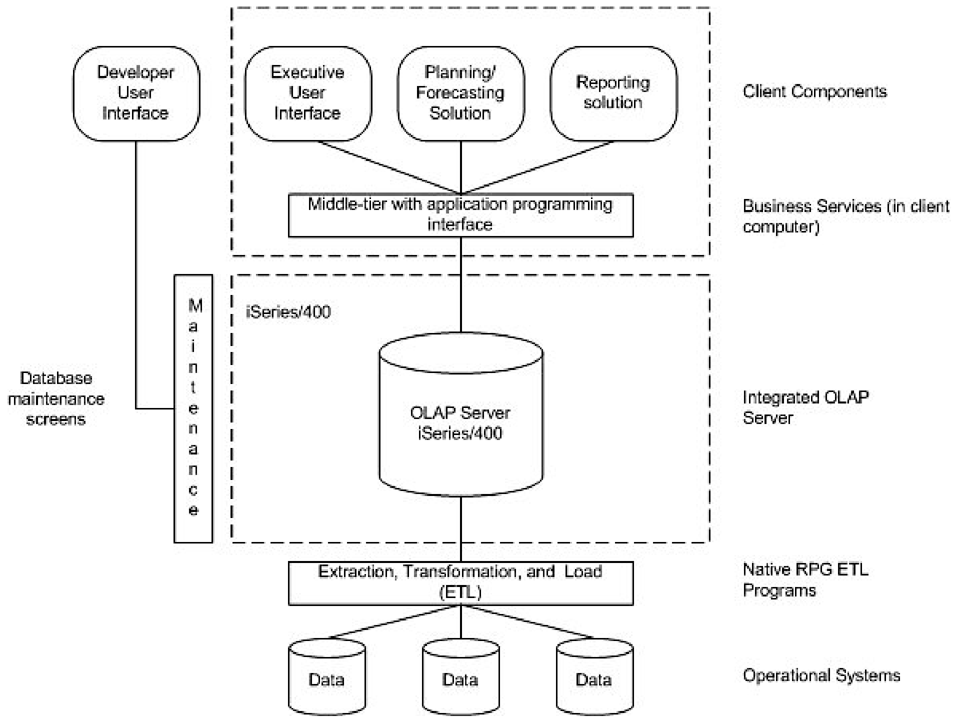


Fig. 28. Extended Solution Offering.

With integrated software releases 5.2 and 5.6, the Company introduced major changes in the server software by improving the performance of database load and data retrieval (software release 5.2) and adding new business calculation rules to the server (software release 5.6). With integrated software release 5.6, the case study company converted the internal 16-bit client software to 32-bit technology. This undertaking took several months to accomplish. This exercise, together with increased pressure to review the client software to accommodate new end user organization requirements, caused the case study company to replace the initial client software with a new client that would provide end user organizations more flexibility to integrate with different office productivity environments such as Microsoft Office. The decision was also made to separate product versioning by introducing different version numbers for client and server software modules. The initial renewed client software 1.0 was introduced together with server software 5.7, enabling backward compatibility with old client software. The case study company dropped support for the old client software coinciding with the introduction of the new client software module, and the final release of the old client software module came with server software release 5.7.

With integrated software release 6.0, the Company introduced a new architectural tier to the overall solution, enabling end user organizations to build detail level data warehouses and to deploy large enterprise data warehouses. This solution enabled the Company to address new end user organization requirements that it had not been able to solve before. But the new solution also added one additional layer of complexity that the case study company had to deal with, both from a solution implementation perspective and a development perspective.

The company introduced three distinct client software releases for the new client software module (1.0, 2.0, and 2.4), with corresponding server software releases (5.7, 6.0, 6.1, and 6.2). The last server release, 6.2, includes functionality that enables faster database loads in multiprocessor iSeries/400 environments.

7.2.3 Software Application Frame Release Analysis Overview

A common mistake that software organizations make is to interpret a software product release as a software application frame release. A software application frame release should be the foundation for multiple software releases or derivative products, as the cost of a software application frame is typically much larger than the cost of a product release. We introduced the four principles from Sääksjärvi (2002) in Chapter 2. These four principles are used to identify software application frames within the case study company.

The first principle – that the application frame must improve the effectiveness of application development by applying large-scale reuse of a common application core – requires a predefined aim of the software organization to implement a solution that can be used across any vertical or horizontal market segment without having to change the core software application frame. The second principle – that the software application frame is an implementation of a selected architectural style using a set of underlying technologies that will be conserved in all products generated from the frame – expresses the requirement that the software application frame not be changed from one product release to the other. The third principle – that of software application frame extension - is applied when a company has a need to add new functionality without disturbing existing modules within the software application frame. The fourth principle – that of renewal of the software application frame – happens whenever the internal modules and/or interfaces have been redesigned and therefore changed dramatically.

The task of identifying, separating and describing each historical software application frame release was challenging, as product and product development documentation did not explain in great detail what had been accomplished with which software product release. We compared different types of documentation (product documentation, user group meeting documentation, strategy documents, etc.) with each other and interviewed the original founders of the Company, resulting in both software product release and software application frame release schedules (see also Appendices 2 and 4). As an example of the results of our interviews, the Server Development Manager concluded the following:

"The underlying foundation from version 2.0 to version 4.0 has stayed the same, and not until version 5.x did the server frame have major redesign. All of the server frame releases have been internally the same through version 6.2. We have done several extensions such as adding support for multiprocessor environments, and we have also added support for a relational data warehouse environment that helps us in large environments."

We constructed a table (see Appendix 2) with the aim of listing all the software product releases the Company had introduced since its inception. In parallel with this analysis, we interviewed development team members and management to identify internal changes to the software solution (in both the server and the client software modules). This was triangulated with software documentation and additional requests to confirm our analysis results. The results of our software release analysis, with corresponding software application frame releases, are tabulated in Appendix 4. A summary of these two appendices can be portrayed as follows (see Fig. 29).

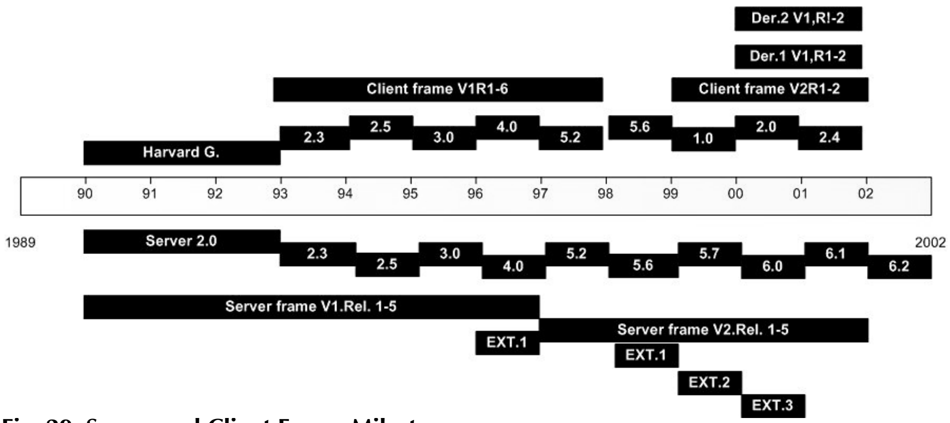


Fig. 29. Server and Client Frame Milestones.

Figure 29 shows the lifecycles of both the software product and the software application frame releases for the case study company. To be able to identify software application frames, software application frame extensions, or software application frame renewals, we had to compare each software product release with the changes that had taken place in the corresponding software application frame release. Fortunately, the Server Development Manager has been with the Company since its inception and is still actively involved in the development of the software product. Many of the changes in each product release were confirmed by him checking in both the manuals and the source code of the server frame releases.

Client frame development has been mostly performed by an external development organization, so that each software release has been under strict version control. We could therefore analyze the changes that have taken place during the years. This was confirmed by the client software

Development Manager, who concluded that the case study company "can go back into any prior release to see what was changed in the client frame."

7.2.3.1 Analysis of server frames

The first server software release from the case study company dates back to 1989. It was more or less a prototype release that demonstrated the software application domain to the founders of the Company. The first initial server product used Harvard Graphics to display flat files from the server. According to the Server Development Manager, "the maintenance of these files became impossible later on as the multidimensionality increased the amount of files." One of the key innovations that the case study company implemented in later development stages was the concept of "navigation," where one single chart/report could be used across dimensions or a combination of dimensions (called a dimension set).

According to our interviews with both the CEO at the time and the Server Development Manager, it became evident that the Company had maintained the idea of having one solution that would be deployed across any vertical or horizontal market segment. This type of large-scale reuse (principle one) across market segments enabled the case study company to deploy the same software solution regardless of selected market segment. The server and client frame architectures and underlying technologies were kept unchanged during their lifetimes. This complies with the second principle of software application frame utilization. It can be argued that the internal architectural models for both the client and server frames were not specifically implemented for derivative product purposes, but the idea of having one solution without modifications was a strong leading idea in the development. The Development Manager for the server products concludes:

> "I have received requests from right and left during the years and if I had implemented all of these requests, the server module would be architecturally a disaster. Our approach has been to enable end user organizations to create a solution that can be applied to any business area or vertical segment regardless of the company."

The first server frame (V1) included five different releases, with one software application frame extension that was needed to enable better integration to Microsoft Office environments. The server faced increased pressure for renewal, and this renewal resulted in a renewed server frame (V2). This renewal included replacement of several internal subsystems, aimed at increasing the performance of database loads and enabling deployment of larger multidimensional databases. The renewal included major new rewrites of application interfaces within the server frame, but these changes did not cause any changes in the communication protocol between the client and the server frame. The renewed server frame (V2) has had several releases (five) and extensions (three) since it came out in early 1997. The underlying technology has been kept the same during the years, with the exception of an added programming language (C++) that has enabled more efficient communication with the new client frame release (V2).

7.2.3.2 Analysis of client frames

The pressure to renew client frame V1 increased when changes in desktop operating system technology took place in the middle of the 1990s. First of all, client software applications transitioned from 16-bit to 32-bit technology. This pressured the case study company to either convert its existing application to 32-bit technology or completely renew the client frame. Secondly, end user organizations became more interested in integrating information with other office productivity tools, such as Microsoft Office. This requirement became an inhibitor given the competition. The core of the client frame V1 was not architecturally solid: it had to be renewed to enable future enhancements. The CEO of the case study company concluded that "we need to find a third-party software company that helps us in this transition, as we do not have the skill set required." The work on a new client frame started in early 1998, becoming one of the largest investments the Company has ever made in any specific product. The client frame was planned to support different derivative products, and it was designed to be component-based, with different architectural layers. This new client frame became the basis for two derivative products. With the release of client frame V2, the case study company decided to separate the release cycles of the server and the client frame from each other. This demonstrates that while the client and server frames have been isolated from each other from technological perspective, they are commercially inseparable.

7.3 Ex-Post Analysis of the Case Study Company Through 1998

The first Analytical Application Frame Strategy framework analysis period covers product development from 1990 to 1998 within the case study company. We presented our elements of analysis in Chapter 6. The following table portrays our analysis results for the software application frame (see Table 6):

	Sub-Strategies for an Analytical Application Frame Strategy				
Software Application Frame Release	**Analytical Application Frame Technology Sub-Strategy**	**Analytical Application Frame Architecture Sub-Strategy**	**Analytical Application Frame Leverage Sub-Strategy**	**Dominant Alignment Perspective**	**Evidence of Alignment Perspective Disconnect**
Server frame V1R1 Year 1990-1992 (Version 2.0)	• iSeries/400 technology on server • DB2 on iSeries/400 relational technology	• Initial server frame, client product still based on Harvard Graphics (third-party technology)	• None	• No evidence of any specific alignment	• No disconnect identified
Server frame V1R5 – Ext.1 Year 1996-1996 (Version 4.0)	• iSeries/400 technology on server, DB2 on iSeries/400 relational technology • Middle-tier technology based on Microsoft COM, C++ and Visual Basic as development language	• Server extensions to support new software application frames on client side (PC) • Change in database model • Addition of dynamic data groups	• Horizontal market segmentation via a new planning and budgeting software solution	• "Harnessing market potential using analytical application frame"	• "Technological disconnect"
Server frame renewal, V2R1 Year 1997-1997 (Version 5.2)	• iSeries/400 technology on server • DB2 on iSeries/400 relational technology • User index technology on iSeries/400	• Major replacement/ rewrite of several server frame components • Use of new user index technology within the database architecture	• Leverage achieved via performance increase in the server frame, which enabled to address the needs of larger organizations.	• "Implementation of technology in the analytical application frame"	• "Segmentation disconnect"
Server frame V2R2 – Ext.1 Year 1998-1998 (Version 5.6)	• iSeries/400 technology on server • DB2 on iSeries/400 relational technology • User index technology on iSeries/400	• Extension of server frame to support calculated data types	• No impact on frame leverage	• "Implementation of technology in the analytical application frame"	• "Segmentation disconnect"
Client frame V1R1-6 Year 1993-1999	• C and C++ as new implementation language • Release development outsourced, proprietary communication protocol	• Client frame based on proprietary defining technology • Six major releases during its lifetime	• None	• No evidence of any alignment perspective	• Weak fit between frame architecture and frame technology strategy

Table 6. Analytical Application Frame Strategy Analysis through 1998.

The first column identifies the software application frame release. The three following columns demonstrate the three sub-strategies within an Analytical Application Frame Strategy framework with our observations. The fourth column portrays the dominant alignment perspective identified for the selected analysis period, while the fifth column expresses possible fits and other alignment perspectives. The final column portrays possible alignment perspective disconnects that the case study company might have experienced during the analysis period.

7.3.1 Analytical Application Frame Architecture Sub-Strategy

Sääksjärvi (2002) set four principles for an application frame to satisfy the requirements of a software application frame. The first and arguably most important criterion is to identify whether the software application frame is built with the intention of large-scale reuse: to be able to have several successive products within a product family. All of the analysis of possible software application frame usage is performed ex-post, which makes it challenging: historical records must be collected to be able to calculate software application frame effectiveness and efficiency. The main goal for the Company was to create a software solution that would satisfy the needs of a large set of vertical and horizontal markets without having to maintain several different versions of the same software product. Our interviews with the founders of the Company revealed that the intent was to create one software solution. There was no plan to create a product family based on a common core (software application frame). The reuse was more or less achieved by enabling a large set of customers to use the same technology and solution base.

In a similar manner, the second principle was realized, as the basic underlying technology (iSeries/400) and architectural style have stayed the same during the years until 1998, when the server frame was renewed. It is also fair to conclude that the management of the Company was exceptionally forward thinking, as the initial server frame included all the basic elements of a defining technology (meta data repository, ETL functionality, etc.) that was expected to be found within a software solution in the given software domain. The case study company built a concept called the "Chart Gallery" that enabled user organizations to save charts/ reports into a common repository with business analytics such as measures (data types) with corresponding exception reporting (with variances). This enabled end user organizations to implement a solution more rapidly when compared to other solutions in the marketplace.

When the software product architecture implemented by the case study company is compared with the analytical application functional architecture in Chapter 3 and the analytical application frame architecture sub-strategy in Chapter 4, we can identify clear resemblances in these models. The biggest difference is mainly in functional components and technical implementation of the software application frame. In the initial server and client frames, the development team did not use a strict layered architectural model. This caused some conflicting designs, such as application logic residing in the wrong software application frame. Even if the development organization recognized these deficiencies, the logic could not be easily changed due to different execution and development environments.

The two last software application frame principles – software application frame extension and software application frame renewal – were also identified during the first analysis period (1990-1998). To be able to satisfy the needs of users who required better integration with office productivity tools such as Microsoft Excel, the server frame (V1R5 – Ext.1) had to be extended to include a new module that enabled flat file extraction for these types of office productivity tools. This extension did not disturb or change any of the existing modules within the server frame. Another clear indication of change to the server frame was introduced during 1997, when the server frame met increased pressure to enable accommodation of larger multidimensional OLAP databases. This prompted the case study company to renew the server frame (V2R1). This was a major new release, with most of the internal modules being replaced or refined. The Company also included new underlying technology that had not been used before in the server software environment (user indexes that are part of the OS/400 operating environment). One additional extension was introduced to the server frame (V2R2 – Ex.1) when new calculated data types were added to the software solution. These calculated data types had previously been part of the client frame, but they had to be added to the server frame to service other client software packages as well and to be part of a common business analytics repository as defined in the analytical application frame architecture in Chapter 4.

7.3.1.1 Server frame architecture for initial server frame V1R1

The initial server frame (V1R1) included the most rudimentary functionality required of analytical application defining technology. The solution was purely based on a server module with an extraction, transformation, and load functionality that enabled the integration from operational applications such as ERP software applications. The first version of the server frame did not include complex report/chart types. The main chart type was a time series chart that enabled end users to track strategic trends. This led to the initial naming of the product ("Trendbank"). The CEO concluded in our interview how the product was positioned:

> "The ability to trend business information gave us the idea to call the overall solution for Trend Bank and this was exactly what business executives needed and we were the first one to provide this type of functionality on the iSeries/400 hardware environment."

The first release of the server frame was dependent on Harvard Graphics from 1990 to 1993. The overall functionality of the solution was cumbersome, requiring lots of manual maintenance.

The Development Manager for the server frame confided in one interview: “if customers only know the underlying structure and what we were doing during the nights when updating and changing the server frame modules based on the feedback that was received during the business hours.” Service revenue from 1991 to 1992 was approximately the same size as the license revenue. It was evident that the software product became more mature in 1993 forward, as software license revenue grew more rapidly when compared to service revenue.

7.3.1.2 Server frame extension (V1R5 – Ex.1) – Buying time for client software

To buy some time for the old client frame (1996 to 1998), the case study company initiated a development project wherein three new software applications were created. Two of these applications were based on Microsoft Excel technology and the third was the software solution which enabled data transfer from the OLAP server engine. The client frame had to undergo several major changes in a technological sense, such as moving from 16-bit to 32-bit environments, but none of these added new functionality to the client frame itself.

However, the existing server frame did not have the ability to align with these new requirements and had to be extended with a server frame extension (Server frame V1R5-Ext.1). This extension came out with software release 4.0. Up until release 5.7, both server software and client software releases were synchronized with same release numbers, but with the renewal of client frame, the software release schedules were changed and numbering reflected each software release separately. Another internal change in the server frame architecture was a change in the data model that enabled the database size to grow beyond its old limitations.

Another server frame extension (V2R2- Ext.1) was developed for software version 5.6 that included major new functionality, enabling calculations to be defined in the multidimensional OLAP server database and calculated on data retrieval. This was a clear extension to the server frame. This extension did not disturb or impact any of the existing modules or interfaces (principle three). This new extension also increased scalability, as most of the calculations were executed on the server and not on the client, as was most common way of doing things at the time.

7.3.1.3 Client frame evolution (V1R1-6)

The server frame did not include any specific vertical market segmentation functionality that would have helped the case study company to address specific needs for selected market segments. This type of market segmentation was at the time not very common. Only recently have analytical application software vendors begun extending their solutions by enabling vertical market segmentation via predefined business metrics and data models. The server frame lasted more or less with a similar internal architectural structure until the mid 90’s, when end user organizations started to pressure the case study organization to add more support for Microsoft Office environments. Due to the outdated client frame architecture, the development team could not amend the existing client frame to include new functionality. The frame had become fragile, requiring considerable work to add any new features. The Chief Architect of the software development organization concluded:

> "Without renewing the client frame we will not be able to add any new features. We keep on adding new stuff into the product and the previously added features do not work anymore. We can not keep doing this anymore."

The quote from the Chief Architect reflects a typical situation when the internal architecture of software application becomes fragmented and the development team spends more time maintaining the software solution and trying to keep it running than adding new features that end user organizations have been asking for. A new client frame was introduced and implemented between 1998 and 2000, one built to satisfy both new technical and functional requirements.

7.3.2 Analytical Application Frame Technology Sub-Strategy

The case study company had five different server frame versions and six different client frame versions during the analysis period. The server frame included one frame extension (V2R2 – Ext.1) that extended the analytics functionality within the analytical application frame architecture.

7.3.2.1 Server frame V1R1

The initial server frame release from 1990 to 1992 was the first crude server software release that the Company built with support for the Harvard Graphics presentation tool. The server frame included the first versions of ETL components that enabled integration of physical files on the iSeries/400 server. The programming language environment was purely based on RPG, as this was the only programming language environment supported by iSeries/400. The technical architecture was more or less a server software solution with rudimentary flat file transfer mechanisms to support the viewing of information using Harvard Graphics.

According to the Server Development Manager "during the days I did consulting at the customer site and during the nights I did programming of the features I realized were missing in the product." The execution and development environment during this server frame release was exactly the same environment (iSeries/400), with the included integrated RPG compilers and programming environment. The company released five different versions (V1.Rel. 1-5) of the server frame during 1990-1997. During this time period, the technological foundation stayed the same.

7.3.2.2 Client frame V1R1 introduced

With software release 2.3 (a combined server and client software release), the Company introduced a new client software solution that was very different from the server frame solution from a technological perspective. The overall architectural model changed to native client/server, with a highly optimized communication protocol between the iSeries/400 server and the client software. This communication protocol was built on top of Advanced Program-to-Program (APPC) communication (part of IBM's SNA architecture). The protocol required that a router to be installed on the PC desktop that used the client software. The initial client software solution was based on the IBM OS/2 operating system environment, as this was seen as a strong component at the time. The CEO at the time explained in an interview that "OS/2 was a natural choice for us as it seemed to be well aligned with other IBM technologies and it was not sure at all that Microsoft Windows would become the standard." Soon thereafter, the Company decided to support the Windows operating system environment as well, using a cross-platform development approach. Similar decisions were made at the time by other companies such as Netscape, which was trying to support Internet browsers in different client operating system environments (Cusumano and Yoffie, 1999). The decision to support cross-platform development within the client frame environment required the Company to implement not only domain-specific application logic, but also infrastructure components that are typically already implemented as part of the integrated development environment. Because of the immaturity of graphical user interface development environments and the need to support two different operating system environments (Windows and OS/2), the Company made a decision to build and use proprietary class libraries in its development.

The decision to support and build core infrastructure components resulted in painful experiences when the IT infrastructure market and end user organizations moved from 16-bit to 32-bit environments. Similar results were reported by Cusumano and Yoffie (ibid) when using a cross-platform development approach at Netscape Corporation. The Company was forced to migrate the existing 16-bit software frame into a 32-bit environment. Due to the proprietary class libraries, the task became major, taking several man months to accomplish. The new client software required a new execution and development environment that was based on Microsoft C/C++ development tools.

The client frame showed initial signs of a weak underlying architecture and structural fragility in the mid 1990s. This could be measured by analyzing the cost of maintaining the solution and the company's inability to extend the solution with new functionality without breaking existing code. The Client Software Development Manager concluded in our interview that support calls were taking more time than new development:

"I keep on spending most of my time on the phone talking with customers and partners about the bugs that we have in our product. If we keep on adding new features, our problems will just accelerate and we will definitely lose the war."

The old client frame technical architecture was technically not componentized, neither did it have architectural layers that would shield IT infrastructure elements from functional domain-specific elements. This mixture of C/C++ implementation language with partly object-oriented and non-object-oriented aspects had to be replaced. The Company did buy some time with this replacement by coming out with new Microsoft Office supported products during the time period from 1996 to 1998. The client frame replacement project was initiated during early 1998.

7.3.2.3 Server frame (V1R5 – Ex.1) – extension to support Microsoft Office solutions

With the requirement for integration with the Microsoft Office environment, the case study company introduced new underlying technology to support the development of Microsoft Excel-based planning and forecasting solutions. To support the integration between Microsoft Office and the iSeries/400 environment, the development organization had to implement a server frame extension (V1R5-Ex.1) that enabled extraction of both meta data and data from the multidimensional OLAP server database. This new server frame extension was still based on iSeries/400 technology, but the middle-tier implementation was based on C++. The planning and forecasting solutions were based on Visual Basic for Applications (VBA). The Company had to add new personnel who had experience in object-oriented languages and OMT++, one of several predecessors to UML. The execution and development environment was Microsoft integrated C++ with Microsoft Foundation Classes (MFC). Microsoft Visual Basic and Visual Basic for Applications were used to implement these new Microsoft Office solutions. The software development approach was also changed, with a move toward component-based software development and defined software development methodologies such as OMT++.

The new software solutions increased the complexity of the software development environment, as new software development languages were added. Also added were new integration requirements between the Microsoft Office solutions and the OLAP server architecture. This new Microsoft Office dependency added additional requirements to the IT infrastructure environment from both the end user organization and the software vendor perspective. The case study company also had to introduce a middle-tier architecture based on Microsoft COM to enable communication between Microsoft Office products and the OLAP Server. The Chief Architect concluded in an interview concerning these new integration requirements:

"There is no way we can add additional complexity in our client frame to enable these new Microsoft Office solutions to communicate with the OLAP server. We must build a middle-tier server software solution that is completely separated

> from the existing client frame. If we do this in the client frame, we will have severe problems in stability and this could lead to problems in our customer-base."

Another dimension that the Company had to evaluate was whether to add core competency in this new technology. The decision was to use external contractors. We concluded in Chapter 2 that software vendors should avoid externalizing core competence, as the accumulation of knowledge outside the software organization could have an adverse impact on future software development. External contractors can be used to develop software assets based on the design of a software vendor and with strict control of source code. The Development Manager for the client software product concluded:

> "Each day the external contractors have to check in their work to the source control software environment that enables us to ensure the integrity."

Use of the outsourcing organization increased the pressure to add version and configuration management procedures into the overall development processes of the Company.

7.3.2.4 Server frame renewal V2R1

In the mid 1990s, the server frame faced increased pressure for a major renewal (principle four), as the competition among iSeries/400 software vendors increased and end user organizations had a requirement to build larger multidimensional databases. In our interview with the Server Development Manager about the reasons for the renewal and the work included in the renewal:

> "I knew it was coming and I had started planning it already early 1996 even if customers really did not complain about scalability. However, I do not even want to think about the other alternative of not having done something about it. It would now be too late."

This renewal that came out in software release 5.2 was based on iSeries/400 proprietary indexing technology (user indexes), which enabled the server developers to utilize features of the iSeries/400 hardware environment that do not exist in other operating system and hardware system environments in the same form. The technological foundation did not change with the last server frame release (Server frame V2R2 – Ext.1) for this analysis period.

7.3.3 Analytical Application Frame Leverage Sub-Strategy

The original founders of the Company had the simple vision of creating a solution for management in the IBM midrange market. This vision was to provide easy access to corporate information by using a personal computer. This original market segmentation included two main factors that would determine the future of the Company. First of all, selection of IBM midrange technology would restrict the Company from offering its solutions on any other hardware and/or operating system environment. Secondly, this market segmentation also included a strong statement of belief in the future of IBM's midrange technology. Both of these selections have been major inhibitors to any future market segmentation strategy for the Company. From the technological perspective, the server frame technology selections have restricted the Company from broadening its solutions to any other hardware or operating system environments.

7.3.3.1 Server frame extension (V1R5 – Ext.1) to add new solutions

Our interviews with management indicated that pressure was mounting from existing customers around 1996-1997 to add new functionality to the solution offering. A key requirement was to be able to have better integration with office productivity tools such as Microsoft Office. According to the Chief Architect of the Company, this type of integration was not possible to implement in the existing client frame without jeopardizing the stability of the client frame. Therefore, the development team decided to add server frame extensions to enable this type of integration. As the existing client frame did not support additional features, the Company decided to build new products, based on Microsoft Excel technology, to enable the use of data extracts from the server product. The chief architect evaluated the success of these changes in following way:

> "In retrospect, we should not have created the middle-tier server, that part of technology should be really part of the server frame and not a product that is installed on client desktop. We have run into so many different problems with this architectural model like distribution of queries, query-sets, scheduling the data updates, integration problems with Microsoft Office due to technological problems, etc. We have also run into a myriad of problems because end user environments have so many different versions of Microsoft Office, desktop operating system environments, and other factors that will increase the instability of the overall offering."

The decision to add support for these new Microsoft technologies was also to enter budgeting and forecasting markets, whereby end user organizations can extract information from decision support server into spreadsheets and then submit budgets and plans back to the OLAP server for analysis purposes. This type of market segmentation goes beyond regular market segmentation, as it provides a completely new software application area in which the Company had no prior knowledge. This was a strategic move into new application areas and underlying technologies on the part of the case study company.

7.3.3.2 Renewal of server frame V2R1 to broaden segmentation

Another key event for the Company occurred when it decided to update its server frame (V2R1) technology to include more robust support for larger organizations. This enabled the Company to move its offering from small- and medium-sized organizations to larger corporations. This new release of the server frame (V2R1) included the use of index technology, enabling faster performance in database loads and maintenance of the database structures. The performance improvement enabled the Company to change its position on the price/performance matrix within the analytical application frame leverage sub-strategy.

The Company did not show any specific market segmentation strategy in the development of client frame V1R1-6 other than predefined chart/report galleries for typical sales, wholesale, and distribution organizations. The whole idea behind the software solution was to provide an easy-to-install software package that could be applied to any vertical or horizontal market segment. The Company did not have any specific features or plans to amend the client frame to other market segments. The product did not have the flexibility to include any variation techniques for further market segmentation.

Overall, neither the server nor client frame included any specific features/functionality that would enable effective analytical application solution variation. The server frame with its ETL functionality and meta data repository did include the ability to predefine operational data sources and create predefined charts/reports, but none of these was implemented with the aim of variation to increase the effectiveness of solution implementation in different market segments.

7.3.4 Summary and Key Findings

To be able to evaluate the success of product development within the case study company for the first analysis period, we need to evaluate it against the six alignment perspectives defined in Chapter 4. The initial server frame release (V1R1), with its corresponding frame technologies, frame architecture and customer segments, targeted any market segment (undifferentiated segmentation strategy) that had iSeries/400 as its hardware foundation. The software solution was based on multidimensional OLAP functionality. The business model was relatively simple, as the technological foundation was one set of technologies (iSeries/400) and the business definition was clear. This is how the CEO explained the situation in our interview:

> "Back in the late 1980s IBM midrange computers where like 'black boxes' for the users, and it seemed to be very difficult to deliver information to users, desktops. Based on this limitation, we saw a tremendous business opportunity to deliver information easily to management users and that is why we decided to categorize our solution as Executive Information System, even if we already had a robust OLAP server architecture as the backend while many others had only a sexy user interface but nothing to show in the backend."

Based on our interviews and documentation (internal memos), it became evident that the case study company did not have any specific Analytical Application Frame Strategy defined when designing and implementing its first releases of software solution. We stated before that the Company had managed to implement a software application frame architecture that enabled large-scale reuse (principle one) when reviewing it against the number of customers. During the years 1990 through 2002, the number of users had steadily grown (see Appendix 4). This enabled the Company to achieve leverage on its software investment. In an interview with the Director of Product Strategies, it became clear that the very early vision had been to provide a solution that could be used across any vertical market segment. The development team did not implement any customer-specific features that could have caused incompatibility problems in future release updates. We argue that this has been one of the key competitive factors for the case study company when compared to other iSeries/400 vendors.

We had to pose the question in our analysis of whether any specific sub-strategy had dominated during the first analysis period, and whether any fit (any relationship between two sub-strategies) had been more strongly emphasized. In our analysis, we could not identify any specific alignment model during the period 1990-1992, when the Company had a server software solution with a Harvard Graphics interface. The case study company was very small and entrepreneurial, with the aim of surviving the economic downturn which took place in early 1990s. When we review the same question from a fit perspective for the first server frame release (V1R1), we recognized "Technological Responsiveness" as the most dominant fit, as the founders of the Company had a strong background in IBM midrange technology, and this was used to build the initial software solution. The management of the Company did not emphasize any given vertical market segment other than the availability of an iSeries/400 server in the end user organization.

The first signs of true strategic market segmentation were identified in mid 1990s, when the management of the Company realized that they had to extend their solution to the budgeting/planning solution market. This decision was more or less driven by a "Harnessing market potential using an analytical application frame" alignment perspective. This resulted in the development of a server frame extension (V1R5, Ext.1) and a new set of Microsoft Excel-based software solutions. The emphasis in this decision was very function/solution oriented, and less emphasis was given to the selected technology. Therefore, there were signs of "Technological disconnect," as the technological foundation for the software got less attention. The Director of Product Strategies concluded:

> "We did not realize the implications of selecting Microsoft as the foundation for the development of a budgeting/planning solution. We later found out that end user organizations have a need to do their budgets online and Microsoft Office development environment does not provide this functionality. It is fair to say that we were locked in Microsoft technology and could not move in any direction."

The strongest fit was "Functional Flexibility." The dominant alignment perspective was "Harnessing market potential using an analytical application frame." The software application frame for the budgeting/planning application was generic, usable across any customer segment. The selected alignment perspective showed potential "Technological disconnect," as most of the focus was given to the analytical application frame and the analytical application architecture sub-strategy.

According to our interviews, the Company met increased pressure (during 1997 and 1998) from larger end user organizations to be able to build larger multidimensional OLAP databases. This resulted in the renewal of the server frame (V2R1), which included major rewrites of internal modules and interfaces (principle four). Our interviews with both management and development personnel revealed that this renewal was not an attempt to change market segmentation, but was more or less a move that had to be done to be able to survive given the competition and to remove the physical limitations on the size of the database that could be created. Another significant requirement on the server frame was to enable users to access the application using a graphical user interface in the Microsoft Windows environment. This task required that two new people be added on the development team, doubling the cost of server frame development when compared to the costs in 1996.

The renewal of server frame V2R1 was a clear attempt to improve "Technological Responsiveness," as the server frame architecture became weak and could not have been the foundation for future customer feature amendments. The same server frame release also had signs of "Functional Flexibility," as the solution enabled the Company to lift its solutions to new price/performance tiers. This type of segmentation did not require any specific variation techniques from the product development team. The main alignment perspective, "Implementation of technology in the analytical application frame," was identified, as the Company focused more on technology and the internal server frame architecture. This resulted in an improved software release that provided the Company with better price/performance segmentation when selling the solution. In a similar manner, the functional improvement brought about with the server frame extension (Server frame V2R2 – Ext.1) was an improvement on the server frame architecture which would specifically enable the Company to better satisfy the needs of business analytics. Even if both the server frame renewal (V2R1) and server frame extension (V2R2 – Ext.1) improved market segmentation, the emphasis was more on application frame technology and application frame architecture. This resulted in increased risk of "Segmentation disconnect."

The client frame evolution did not show any specific signs of an alignment perspective. Most of the time was spent in adding new functionality into the client frame to satisfy the needs of end users. The client software became very fragile, as it lacked a robust internal architecture. Less emphasis was given to architectural soundness and technological foundation. The lack of architectural soundness eventually led to increased maintenance costs, and the lack of a suitable technological foundation led to a large migration effort, supporting a transition from

16-bit to 32-bit environments. We did not identify any efforts of the Company to increase the flexibility of the client frame to accommodate the needs of new market segments.

7.4 Ex-Post Analysis of the Case Study Company From 1998

The second analysis period for the Analytical Application Frame Strategy framework analysis covers product development within the case study company from 1998 to 2002. We presented our elements of analysis in Chapter 6. The following table portrays our analysis results for each respective application frame (see Table 7).

	Sub-Strategies for an Analytical Application Frame Strategy				
Software Application Frame Release	**Analytical Application Frame Technology Sub-Strategy**	**Analytical Application Frame Architecture Sub-Strategy**	**Analytical Application Frame Leverage Sub-Strategy**	**Dominant Alignment Perspective**	**Evidence of Alignment Perspective Disconnect**
Server frame V2R3 – Ext.2 Year 1999-1999 (Version 5.7)	• iSeries/400 technology on server • DB2 on iSeries/400 relational technology • User-index technology on iSeries/400	• Extension of server frame to support multiprocessor environments • Ability to create 100 time larger dynamic addresses	• Ability to address larger databases enabled move on price/ performance matrix	• "Implementation of technology in the analytical application frame"	• "Segmentation disconnect"
Server frame V2R4 – Ext.3 Year 2000-2000 (Version 6.0)	• iSeries/400 technology on server, DB2 on iSeries/400 relational technology • Relational technology using SQL as DML language	• Size of OLAP database increase tenfold • Extension of server architecture to support new architectural tier • New application frame with frame components	• None other than the server would enable new price/ performance tier leverage	• "Harnessing market potential using analytical application frame"	• "Technological disconnect"
Client frame renewal, V2R1-2 Year 1998-2002 (Version 1.0 to 2.4)	• Native Microsoft COM technology • IIS technology • Component-based software engineering	• Major replacement/ re-write of client frame with two major releases	• Strong frame leverage for future client software derivatives	• "Implementation of technology in the analytical application frame	• "Segmentation disconnect"
Derivative products from client frame V2R1 Year 2000-2002	• Microsoft DCOM • Microsoft IIS • C++, Visual Basic, Java Script, VP Script	• Thin client architecture based on middle-tier server, core client frame components reused in derivative products	• New derivative client products from client frame, no specific vertical segmentation strategy	• "Implementation of technology in the analytical application frame"	• "Segmentation disconnect"

Table 7. Analytical Application Frame Strategy Analysis from 1998.

The case study firm had five server frame releases and three server frame extensions during the analysis period. The initial client frame (V1) was replaced with a new client frame (V2) that enabled the case study company to produce two derivative products. Two client frame releases occurred during the analysis period.

7.4.1 Analytical Application Frame Architecture Sub-Strategy

In a manner similar to that of the first analysis period, we need to evaluate whether the four software application frame principles were met when defining the software application frame releases for the second analysis period. The principles of large-scale reuse (principle one) of the server and client frames did not change from server frame version V1 to V2. Even when the server frame was renewed, the internal functionality was kept the same, with corresponding analytical application frame elements such as the defining technology, the service component layer, and the extension component layer. The case study company added one additional data warehouse tier based on relational technology. This enabled the product to address new enterprise data warehousing requirements that had arisen specifically in large end user organizations. The Company had three server frame extensions during the analysis period, and it renewed old client frame V1. The new client frame V2 gave the Company the ability to create the first true product family based on common software assets that were part of the client frame.

The pressure to increase the functionality of both the server and the client frames increased during 1996 and 1997. The Company had to evaluate once again its ability to provide the functionality expected from the software solution. From an analytical application frame perspective, the Company had three options to choose from when deciding on the renewal of client frame. The first option was to keep on adding new features and functions to the existing client frame, which would have increased the maintenance costs and jeopardized the stability of the client frame itself. The second choice was to amend the existing client frame with new extensions without impacting the existing client. The third and more radical alternative was to replace the client frame completely with a new client frame.

In our interviews with senior development managers, we learned that the Company did not really have an alternative other than to replace the existing client frame by using a new architectural model and a new execution and development environment. The existing client frame was outdated and architecturally too complex to modify. Extension or renewal could not be accomplished without major investment. Even if the Company had invested money in maintaining the existing code base, the results would have been unpredictable in the future, as technological changes could not have been embedded into the existing client frame.

7.4.1.1 Two new server frame extensions introduced (V2R3, Ext.2 and V2R4, Ext.3)

The Company initiated a renewal process by renewing the server frame (V2R1) in 1997. The initial product based on the new server frame was released in 1998. This new server frame enabled deployment of larger OLAP multidimensional databases, which had become a major requirement for larger end user organizations. This server frame was later amended with several frame extensions that provided additional functionality without disturbing the core server frame and its stability (principle three). One of the new extensions that came out during 1999 enabled the software to run natively in a multiprocessor environment and take advantage of several CPU's when loading data into the multidimensional OLAP database. This new functionality was made possible by implementing a new server frame extension (V2R3, Ext.2) that was included as a new module without disturbing existing modules or interfaces in the server frame.

A new major extension was released early in 2000 that enabled the Company to extend its solutions to enterprise data warehousing. This new extension can be regarded as a server frame in itself, coexisting with the original multidimensional OLAP server frame. The new extension enables end user organizations to build relational enterprise data warehouse structures. The Director of Product Strategies discussed in an interview the importance of amending the solution to enable enterprise data warehousing:

> "This new relational technology takes us to the next level in the competitiveness as some countries have always been a problem for us. German customers for example want to build huge OLAP databases with too large dimensions, and this has given us negative feedback. With this new technology we are able to build detailed level data warehouses and keep the OLAP multidimensional server as it was originally intended to be used for strategic trend analysis."

The Director of Product Strategies continued explaining that "these two server frames will eventually become one frame with one set of ETL tools that enables us to create a more user-friendly look-and-feel for the products." The server frame was implemented using more architectural layers. For example, the communication module for client/server communication was implemented as a new functional layer above the data structures. The server frame (V2R4, Ext.3) was complemented with the ability to build larger multidimensional OLAP databases by a factor of ten.

7.4.1.2 New client frame V2R1

The management and development organizations of the case study company defined three ambitious goals for the client frame (V2R1) renewal.

1. The software application frame would have to enable derivative product development from a common client frame.
2. The overall architectural model for the client frame would be highly layered, with the business and analytics component layers isolated from the presentation and distribution component layers.

3. The new client frame would have to support other software programs calling its application programming interface to support future client frame extensions.

These requirements reflect the analytical application architecture described in Chapter 4, with the exception of the middle-tier layer that acts as the broker between the analytical application defining technology and the service component layer. This layer was implemented to achieve more flexibility in future client frame development, such as support for other environments and thin clients (see Fig. 30).

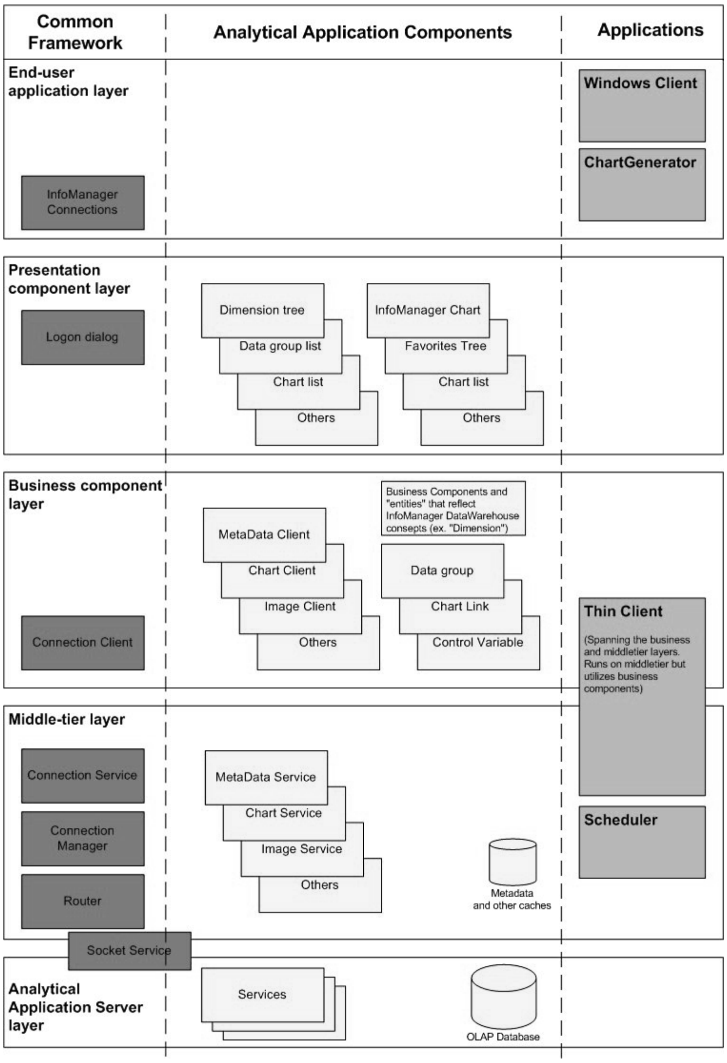


Fig. 30. New Client Frame Architecture.

From an architectural perspective, a layered architectural model enables the Company to amend its solution to support different client environments such as the Pocket PC and cellular phones. If the Company had implemented the solution in the traditional non-layered model, derivative technological product development would have required considerable effort. We interviewed the Chief Architect of the solution, the Development Manager, and several of the development team members. Each interview resulted in the conclusion that layering the software architecture enabled the Company to add features later in the development cycle. The client frame includes predefined design patterns such as database logging, exception management, meta data management, and several others that can be shared regardless of the device used by the end-user analyst, both now and in the future. Each architectural layer within the client frame architecture has a well-defined application programming interface, which enables each software component to work within a single application, with both a presentation and a business logic layer or, alternatively, using a model in which the presentation layer is separated from the business logic layer. This latter model is specifically used in thin client environments, where business logic must reside in middle-tier servers. This kind of separation would not have been possible with a traditional client/server architecture.

The overall client frame architecture is divided into several layers (an analytical application server layer, a middle-tier layer, a business component layer, a presentation component layer, and an end user application layer). The architectural model is also divided into vertical layers. The framework components can be reused across any derivative end products, across any analytical application software components (the components that provide the functionality in the application itself), and finally across the software application that represents the software artifact that is delivered to the end user organization. An interview with the Project Manager from a third-party vendor revealed that the common framework components were defined based on their generic functionality within the selected application domain, while the analytical application components were defined and designed based on the functionality requirements of the software solution. The middle-tier layer acts as a load-balancing tier, providing more scalability for the overall solution. One of the key findings was that each middle-tier service layer has a corresponding counterpart in the business component layer. Therefore, the components that are implemented in the middle-tier layer have divided functionality. They will be controlled via the application programming interfaces that both layers share. In an interview with the team leader for the entire client product family, it was learned that extremely rapid implementation of SQL drill-through was implemented via these paired software components.

A comparison of this client frame architecture with the analytical application frame architecture in Chapter 4 shows that the client frame architecture is only a partial implementation of an analytical application frame architecture, with frame components from the business and analytics component layer and the presentation and distribution component layer. This type of separation of user interface and distribution logic from the business and analytics component layer enables software developers to reduce dependencies between different architectural layers. The development organization had to implement middle-tier software components to manage connection pooling, adding communication managers to the analytical application server layer presented in Figure 30. The client frame architecture model portrays a business and analytics component layer that will be applicable and sharable for all derivative products within the client product family. The same applies to the presentation and distribution component layer. This layer will also implement the functionality needed to display information on different devices. Software development organizations typically expect this type of software infrastructure technology to be found in the software component framework or in the software components that are part of the execution and integrated development environment.

The selected architectural model, with clear separation between the IT infrastructure environment and the application domain environment, enables the case study company to deploy and reuse common software assets across different derivative products. This layered architecture has enabled our case study company to repackage modules in different configurations using different collections of software components. Usage of a software component framework enables the development organization to propagate or create new additional services without having to change the client frame itself.

7.4.2 Analytical Application Frame Technology Sub-Strategy

We portrayed all the needed elements within an analytical application technology sub-strategy in Chapter 4. During this analysis period, there were two major changes in the development environment. First of all, the server frame development team included new implementation technologies such as SQL as part of the data manipulation language. The client frame renewal included several other technologies which have resulted in a more complicated execution and development environment.

7.4.2.1 Server frame to support larger database environments (V2R3, Ext.2)

The server frame version in 1999 (Server frame V2R3, Ext.2) did not include any major new technologies other than the ability to utilize iSeries/400 multiprocessor environments more efficiently. The Server Development Manager concluded in an interview that "large end user organizations will be able to utilize the iSeries/400 hardware to its fullest and the efficiency of our ETL processes will increase dramatically." The release included the ability to increase the size of "dynamic addresses" by a multiplier of one hundred, which enabled end user organizations to create larger databases.

7.4.2.2 New server frame to support a new architectural tier (V2R4, Ext.3)

The management of the Company decided to invest in an additional data warehouse layer that enabled end user organizations to build enterprise data warehouses using relational technology. This server frame release can be regarded as a server frame release by itself, with the same underlying technology as the initial multidimensional OLAP server frame. The new OLAP server frame release (V2R4, Ext.3) extended the size of the OLAP cube tenfold, which benefited larger organizations. According to the development team members, the communication module was rewritten using C++ as its implementation language. This was something that the iSeries/400 included as an alternative development language in the late 1990s. Interestingly, the operating system environment for OS/400 was written using C++ when IBM moved the operating system environment from CISC to RISC processors, and it took a while to get support for the C++ development environment on the iSeries/400. This communication module was implemented as an extension to the server frame, with data exchange between RPG and C++ modules.

The new relational server frame release did not include any major new technologies other than including SQL as part of the data manipulation language. From a technological perspective, both of these server frame releases were still mostly based on RPG. This obviously prevents the Company from moving the application logic to any other operating system or database environments. Moreover, the selection of RPG as main implementation language also impacted the execution and development environment in a way similar to that of the first server frame (V1) release. If the Company wants to extend its solutions to other environments in the future, it will be a major investment for the Company not only in technology, but also in core competence and other factors that are at present unknown to the Company. These types of technological choices are very difficult for any software vendor, as software technologies could potentially move in different directions, as has been seen in the Microsoft vs. Java implementation communities.

7.4.2.3 Client frame renewal (V2R1-2)

The renewal of the client frame was based on several factors, but the most compelling factors were the increased maintenance costs and the inability to add new features to the client product without breaking existing code. Our interviews with the development team members revealed that the selection of a new underlying IT infrastructure environment did not leave many choices, as most of the selections were already chosen by the market: the new client frame had to be based on and supported by Microsoft Windows. Secondly, this new client frame would have to be used in thin client environments. The only choice at the time was Microsoft web server technology, as the comparable web technology in the OS/400 environment did not meet the requirements set by the application group. These technologies also supported distributed component technology (DCOM), needed to distribute functionality across different architectural layers within the selected application architecture.

These selections reflected the core competence of the development team members, as the client frame was heavily based on the Microsoft application development environment with its corresponding execution and development environments. The Microsoft development environment also supported natively distributed computing (DCOM), which was needed to provide load balancing between the client software module and centralized meta data management. Load balancing was planned to help an existing architectural design flaw that prevented executive management end users from using the client software application remotely with large OLAP databases. Instead of loading meta data into a local client, the meta data would be staged in the middle-tier server. Using distributed computing, the client software could act as though the meta data were local. To evaluate each technology selection, the technology group documented each alternative selection and combinations of feasible selections. In the end, most of the selections were decided based on practical reasons, as comparable technologies were either weak or did not fulfill the criteria set for the solution. An excellent example of an alternative technology for distributed computing was CORBA, which at the time of selection had 42 different implementations, each requiring royalty payments.

To meet the functional requirements for the new client frame, the development team concluded that a highly modular multi-tiered architectural model would be best suited to the current and future requirements of the software solution. Modularity had more potential to facilitate the use of common software assets across different derivative software products within a product family. A multi-tiered architectural model could give additional flexibility for better load balancing with larger end user organization analytical application implementations. A multi-tiered architectural model would enable the software development team to build a business logic tier with centralized functions such as meta data management and configuration and installation of software. This type of layered software development approach enabled the case study Company to plan, design, and deploy thin client environments, as most of the business logic resides in the middle-tier server environment. The Development Manager of the client frame team noted in an interview:

> "Middle-tier technology will enable us also to implement a software wrapper around the "legacy" analytical application server and this will therefore minimize the changes that would have to be implemented to the server frame itself."

According to the development team, a layered component model adds dynamics and flexibility in future product amendments, as frame components can be scattered across different architectural layers. A middle-tier solution will increase the scalability of the solution, as client users can use several middle-tier servers instead of one. From a software development perspective, a multi-layered architectural model is more challenging for software developers, as the software must be divided into separate logical layers according to the model-view-controller paradigm.

7.4.3 Analytical Application Frame Leverage Sub-Strategy

The case study company had not practiced any serious market segmentation other than to provide an analytical application solution for the IBM midrange computer market. This segmentation is very coarse and does not provide any vertical market segmentation as such. The first conscious market segmentation was undertaken when the Company decided to enter the enterprise data warehousing market by extending the server frame with an additional database tier (V2R4, Ext.3). This segmentation was more or less a move in the horizontal price/performance matrix, as the Company could charge more for the solution, which was specifically aimed at larger enterprises. A major success factor for the Company was the success of the US subsidiary, which made several new enterprise deals early in 2000 due to the increased performance of the multidimensional OLAP server frame and the relational server frame.

The new server frame (V2) did not include any other features or functions that would have promoted vertical market segmentation, such as a solution for different vertical applications that would include a business analytics repository with predefined business metrics and other critical performance indicators that are typical for a given vertical market segment. These types of applications have become more accepted, and in some cases required, amongst end user organizations (Eckerson, 2002). The value proposition for end user organizations is to buy most of the functionality that is needed. The rest will be modified by the end user organization. In our interview with the Director of Product Strategies, we learned that future server and client frame releases will include support for the development and deployment of ready made solutions. The director explained the following relative to analytical application solution support:

> "There will be several changes in both the server as well as client frame to support some level of solution development. The problem in solution development will become apparent if the end user organization amends the solution that has been provided by the software vendor and the versioning could become a nightmare. These types of solution conflicts are by no means easy to manage and will require extensive work and changes from our part."

Segmentation does not necessary have to be based on creation of a vertical solution, but segmentation can also be based on technical market segmentation. According to our interviews, one of the key decision criteria in the planning of the new client frame (V2) was to include the possibility to use common software assets when creating solutions for different client devices such as Pocket PCs and cellular phones. This was enabled by allowing most of the client frame software components and analytical application components to be reused across the different functional combinations that the software development teams and end user organizations require. The aim of effective reuse of common software assets within an analytical application frame is to use market segmentation leverage either by creating vertical solutions or, alternatively, by using horizontal leverage via price/performance segmentation. Another typical segmentation for analytical application solution vendors is to create horizontal applications, such as budgeting and forecasting or customer relationship management solutions. The case study company had executed this type of horizontal segmentation by implementing a Microsoft Excel-based planning solution.

The renewal of the client frame enabled the Company to use the client frame in derivative product development, mainly to provide alternative user interfaces to the multidimensional OLAP server database. The first derivative product, "Broker" (Der.1 V1, R1-2), enabled centralized meta data management using a middle-tier Windows 2000 server. The second derivative product, "Webulator" (Der.2 V1, R1-2), enabled users to access the same data using an Internet browser without having to install anything on the client workstation (see Fig. 31).

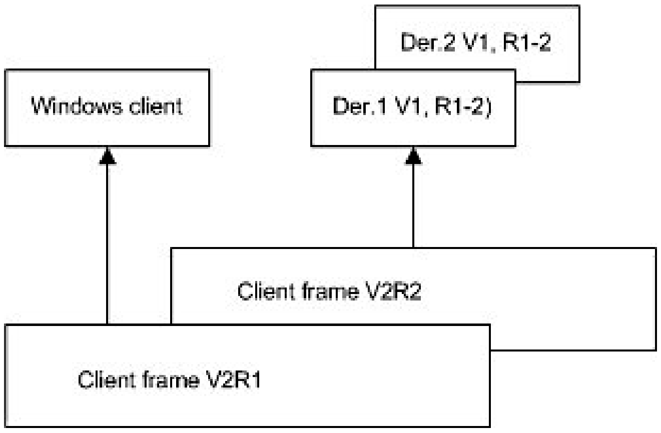


Fig. 31. Client Frames and Derivative Products.

According to our interviews with the development team and the executive management, both derivative products were already planned at the time of client frame V1 renewal.

7.4.3 Summary and Key Findings

The second analysis period can be characterized by the need to either improve the underlying architecture of the solution and/or increase the performance of the software solution. This performance increase could have been achieved either by amending the existing analytical application frame with new technology or using technology as the driver to improve performance bottlenecks. The server frame's (V2R3, Ext.2) internal architecture was changed to improve the size of the OLAP database that indirectly enabled the case study company to address the needs of larger end user organizations. This release included new iSeries/400 based technology (user indexes) that enabled more efficient management of data and implementation of larger databases.

The main alignment model for the new server frame (V2R3, Ext.2) was "Implementation of technology into the analytical application frame." The main emphasis for the new server frame was to improve the internal architecture from the performance perspective. This also enabled the Company to make a move on the price/performance matrix in its application frame leverage sub-strategy. The main fit for the new renewed server frame was "Technological Responsiveness," as new user index technology was introduced, enabling better performance of the multidimensional OLAP server software. The impact of the new server frame was to improve the performance of the server frame, but also to address the needs of larger end user organizations from a sizing perspective. As the focus was more on architecture and technology, the Company did not specifically address the needs of any given market segment, and therefore "Segmentation disconnect" could be identified when analyzing the alignment perspective.

In a similar manner, the Company wanted to address larger multinational end user organizations. This led to implementation of a new relational data warehouse solution with "Harnessing market potential using an analytical application frame" as its dominant alignment perspective. The focus in this perspective was the fit between the analytical application frame architecture sub-strategy and the analytical application frame leverage sub-strategy. This was named "Functional Flexibility" in our Analytical Application Frame Technology Strategy framework. The case study company focused specifically on providing new market segmentation on the price/performance matrix, as well as functional architecture. Less emphasis was given to selected technology; therefore the dominant disconnect was identified as "Technological disconnect." This could become a problem for the Company in the long run, as it is purely based on iSeries/400 technology and is very closely tied to the OS/400 operating system environment.

The client frame renewal (V2R1-2) was a major change in the product development strategy for the case study company. The main alignment perspective for the renewal was "Implementation of technology in the analytical application frame," which was true because several new technologies were selected for the implementation, and these had an impact on the underlying and defining technologies of the Company. The development team also added new domain-specific architectural models that had not existed before. The overall architectural software solution, with both server and client software products, has become more dependent on external IT infrastructures such as Microsoft supported web technologies and programming environments. The original software product architecture was clean and simple, having one server software module with associated client software. The software offering in the new client frame includes middle-tier technologies with distributed computing

(DCOM) and requires specific web-server technology. The Server Development Manager concluded in an interview about this new architectural model:

> "The solution has become more vulnerable due to the more complex technical environment. I do understand that these technological selections were made when iSeries/400 did not provide the technology that was needed to implement the functionality that customers wanted. In retrospect, now we are tied into this technological architecture and Microsoft has managed to lock-in our solution to their proprietary technology."

This comment made by the Server Development Manager portrays well the change in the business model that new technologies and applications have caused the development team during the years. The Company focused mostly on an analytical application frame technology sub-strategy and an analytical application frame architecture sub-strategy, where "Technological Responsiveness" was the fit that the development team emphasized. This emphasis could potentially hamper future marketing segmentation strategies for the company, and its "Segmentation disconnect" could become an issue for the Company and its software development. According to the development managers, these selections have already met some resistance in the end user IT organizations, as the Microsoft web technology and operating system environment have been recently (2001-2004) affected by heavy security vulnerabilities. The renewal of the client frame resulted in two derivative products with the same alignment perspective as the client frame renewal (V2R1-2).

Another issue that the case study company is facing in its client frame development is the versioning of analytical application frame software assets and the use of these assets with derivative products in a product family. The Development Manager for client software concluded in one of the interviews:

> "It seems to be that due to our technological selections and technologies from Microsoft, we are spending more time on these derivative products that we really should."

This statement from the development manager shows clearly that there is a conflict between the selected technology and how this is controlled from a software development perspective. The selection of Microsoft technology has already caused the management grief, as Microsoft has announced that it will discontinue the development environment that the Company has been using. These types of risks can not always be controlled, and might potentially cause considerable harm for independent software vendors.

7.5 Ex-Post Analysis of Economic Metrics in the Case Study Company

We introduced platform related effectiveness and efficiency measures found in the product platform literature in Chapter 2. These measurements were complemented with software business-related economic metrics defined in this study. These additional measurements enable software organizations to evaluate the performance of product development for an analytical application frame with corresponding margin and investment calculations. To be able to calculate both platform efficiency and effectiveness for the combined frame (client and server frame), we decided to use the measurements defined by Meyer et al. (1997): these measure the efficiency and effectiveness of the analytical application frame(s). Platform efficiency depicts the "degree to which a platform allows economical generation of derivative products," while platform effectiveness shows the "degree to which the products based on a platform produce revenue for the firm relative to the cost of developing those products."

7.5.1 Data Sources and Collection of Data

To identify and differentiate software product releases from software application frame releases (both client and server), we conducted several recurring interviews with the founders of the Company and development managers who have been with the Company for several years. Multiple interviews were required was because historical software product releases were not well documented: we had to estimate the release schedules. Fortunately, the server Development Manager had stored old documentation that enabled us to reconstruct historical software product release dates.

The collection of financial data took several weeks, as some of the data had to be collected manually from financial bookkeeping records and transferred onto Microsoft Excel spreadsheets. To verify the collected information, the researcher conducted several iterations of email exchanges and conference calls with the CFO and Controller of the case study company to confirm the conclusions made from the data. The Controller of the Company assisted in the data collection and spent several months collecting licensing information from existing contracts. This enabled us to estimate overall license revenue and related key performance indicators, such as gross margin per customer and per product. We were able to collect service revenue from each subsidiary and parent company. We were not able to collect service revenue from business partners, as this part of revenue is not recognized by the parent company.

Once all of the historical software license and service revenue data and corresponding cost information had been collected and allocated to each software product by year, the data was allocated to each software application frame based on our product release analysis (see also Appendix 2). Therefore it became imperative to differentiate software application frame releases from software product releases.

We presented product platform related effectiveness and efficiency measurements in Chapter 2. To measure these for the case study company, we collected the needed elements from general ledger accounts, accounts payable, and internal analytical application databases.

Several different Microsoft Excel spreadsheets were created during the data collection process. The final spreadsheet includes the following data elements (see Table 8):

Date of sale	Sales organization	End user customer	Product sold	Units	Unit price	New/Existing customer	Share of revenue to parent company	Corporate Gross sales	Royalty percentage

Table 8. Historical Product Sales through 2002.

The columns in Table 8 show the following:

1. the date of sale of the product
2. the sales organization
3. the end user organization
4. what was sold
5. how many units
6. new sales
7. recurring sales to existing customers
8. the share of revenue recognized for the parent company
9. corporate gross sales
10. the royalty percentage paid to the business partner.

We decided to separate the two types of sales (columns 6 and 7), as this information allowed us to see how much the case study company has been able to up-sell to existing customers.

Another spreadsheet was created, portraying overall costs and software license revenue information for each product for each year (see Table 9):

Year	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002
Total Costs License revenue Maintenance revenue Service revenue Total Revenue Gross Margin													

Table 9. Product Gross Margin Calculation.

The first line item in the table corresponds to the overall development cost associated with a given software product. The next three line items show software license, maintenance, and service revenue for each product and year. The final line item (after total revenue) is calculated by subtracting total revenues from total costs. The result is gross margin for each product.

We have divided revenues into three different categories. The first category represents software license revenues, while the second category represents maintenance revenue that enables end user organizations to receive new software updates and support from the case study company. A separate line item (third category) for associated service revenue has also been added to enable comparison of the ratio between software license revenue and service revenue. This service revenue includes installation and/or implementation work performed for end user organizations. The question that every software company must answer is to decide whether it should be a products company, a service oriented company, or a combination of these (hybrid). According to Cusumano (2004), hybrid software companies have survived better in economic downturns, as declines in software license revenue have been replaced with strong service revenue.

The line item for total costs consists of development salaries with corresponding social benefit and third-party software development costs. Total sales revenue numbers are derived from the accounting system combined with an internal analytical application solution that provides the ability to categorize and classify sales by country/business partner and software product. Some earlier software product sales have been estimated using a 70/30 rule, where 70 percent of the software license revenue is from server software sales and 30 percent consists of client software sales. Service revenue that the case study company has recognized is mainly from the domestic service organization and its work with end user customer organizations.

7.5.2 Measurements in the Quantitative Analysis

We introduced product platform measures in Chapter 2, originating from the work of Meyer et al. (1997). According to Meyer et al. (ibid), platform effectiveness and efficiency measures can be calculated on product-by-product basis or for a group of products within a distinct platform level. We decided to apply both calculations methods to achieve an overall view of the product development efforts in the case study company. Meyer et al. (1997) also conclude that both platform efficiency and effectiveness should be analyzed by product category and industry. Their analysis of a case study company showed platform efficiency of 0.25, which shows that the company was able to produce a new derivative product at roughly 25 percent of the cost of the overall platform. The authors conclude that if the platform efficiency ratio is close to 1.0, either the base platform has not been efficient for derivative platform development and should be renewed or the platform has been poorly designed.

We also included additional measurements that we introduced in Chapter 2, as these will provide additional valuable information regarding the product development efforts from a product gross margin perspective. The first group of additional measurements reveals information about overall profitability of software product development (Total Revenues – Total Costs) and how each software product is doing in comparison to every other software product:

- Product gross margin per product (in currency)
- Product gross margin as a percentage of all products
- Total cost (in currency)
- Share of total cost.

The second group of measurements gives us a comparison of how each software application frame release compares to each other in the form of leverage, service revenue, and software application frame margin:

- Total costs, including all costs associated with the product's development
- Total revenue for the products that have been derived from the software application frame
- Service revenue associated with the software application frame
- Frame gross margin (this gives us an idea of whether a frame release has been paid off or if it is still "under water").

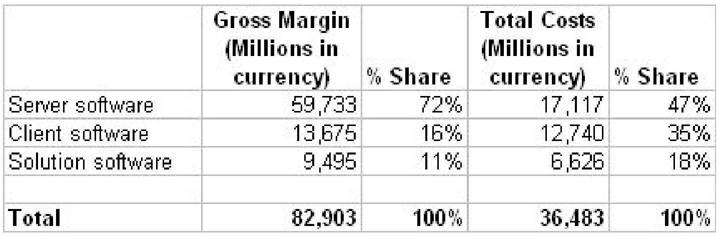
These two sets of numbers are used to calculate the outcome of software application frame development. We also added frame effectiveness with and without service costs. It could be deceiving to assume that a software application frame is effective only by reviewing its associated software license revenue. Service revenue could be extensive, suggesting inefficiencies in the software solution, such as inflexibility or labor-intensive implementation of the data model and integration to source applications. Therefore, the less the software vendor has to spend time tailoring or modifying the software on site, the more efficient the software application frame is from a leverage perspective. If service revenue makes up most of the total revenue, the effectiveness numbers could become meaningless. This type of discussion was not part of the existing software related product platform literature. We also added two additional measurements into our quantitative analysis to give an overall picture of how well the software vendor has been able to provide leverage in relation to customers: numbers of customers and gross margin per customer. The first number demonstrates how many customers the Company has had for a specific software application frame release and the second measure shows the average gross margin per installed customer.

7.5.3 Product Family Profitability

The case study company initially had only two software modules: server software and client software. Over the years the Company has developed new software modules that have kept the overall solution competitive or provided functionality that could not be implemented in the original software offering. Based on this, we divided the different types of offerings into three different product families:

- Server software
- Client software
- Solution software

The last category – solution software - represents Microsoft Excel-based products such as planning and forecasting and reporting solutions and other tools to help analytics reporting. The historical product margins (Total revenue – Total Costs), with corresponding costs for the product categories, are divided as follows (see Fig. 32):



	Gross Margin (Millions in currency)	% Share	Total Costs (Millions in currency)	% Share
Server software	59,733	72%	17,117	47%
Client software	13,675	16%	12,740	35%
Solution software	9,495	11%	6,626	18%
Total	**82,903**	**100%**	**36,483**	**100%**

Fig. 32. Product Family Profitability for the Company.

Figure 32 shows the importance of the server software to the overall architecture and the share of historical investment that it has had in product development. A considerable amount of product gross margin has been generated from value added software modules: they represent more than 11 percent of the historical gross margin. It is also clear that the server software product has been the core for the Company, with an impressive 72 percent of the product gross margin. To our surprise, in our analysis of all products that the Company has released during the years, we were not able to identify any products that had negative gross margin: even a couple of discontinued products had positive product gross margin. Another consideration that a software company should make when investing in new software products is the alternative cost: the cost of doing something else. Even if the Company has not had any products that have had a negative gross margin, the question remains whether the same investment dollars/euros could have been better spent on existing products or other alternative products yielding higher return on investment.

Our interview with the developers and sales executives confirmed that the tradition of the Company has been to sell its products well before each product is released. Each new module has been installed at a customer site prior to its general release to make sure it will fulfill the minimum requirements. By analyzing the historical sales revenue from the Microsoft Excel spreadsheet produced by the CFO and Controller of the case study company, we could identify a few interesting trends in the Company's sales statistics. First of all, the international distribution channel has successfully sold specifically value added products, such as planning and forecasting modules, to markets such as France. Secondly, it became evident that the North American market became the driver of new software sales in the early 2000s. The US business unit was the first one to introduce new products to the market. The more traditional way of conducting business is sell new software in the domestic market to reduce the number of possible first release bugs that are typical for software organizations.

The seemingly high product gross margin for the server family and the much lower number for the client frame do not necessarily translate to poor gross margin performance from the client family, as the pricing model has been different between the two software modules, with the server product carrying a much higher revenue stream. Another factor that might have played into the overall profitability of the client software module is that the pricing model is based on concurrent users and not named users, as many other software organizations price their products. Our analysis showed that there was a limited amount of re-purchase of additional software client modules. We assume that the concurrent user model with less frequent usage of the software will never run into a situation where the concurrency counter is exceeded in the end user organizations. This pricing model was changed at the end of 2002 due to the findings in our study. The Company has put additional variation techniques, based on control code management, into the product. This type of management enables the case study company to ship the complete package with all features and control the functionality on the price/performance matrix of the frame leverage sub-strategy using application control codes.

Once the initial calculations had been performed, the Server Development Manager verified our calculations and provided additional feedback and insight into the historical events and reasons for some of the selections and decisions that had been made. The Server Development Manager concluded in one of the interview:

> "I wish we had these numbers before so we could have seen the profitability by product across the lifetime of the Company. We really didn't do a good job of collecting data but fortunately this was still possible via documentation and interviews."

This statement explains well that aggressively growing software companies could in some cases lose focus on tracking their investments, failing to ensure that each product offering provides adequate product gross margin and return on investment.

7.5.4 Application Frame Efficiency and Effectiveness Analysis

We introduced in Chapter 2 the additional measurements that provide executive management of an analytical application software vendor with information relevant to their business, such as product profitability calculations, ratios between software solutions, and installation revenue (see Fig. 33).

Product	Timeline	Total Cost (Millions in currency)	Total Revenue (Millions in currency)	Service Revenue (Millions in currency)	Analytical Application Frame Effectiveness with Service Revenue	Analytical Application Frame Effectiveness without Service Revenue
Server frame V1	1990-1996	5,399	16,611	6,111	4.2	3.1
Server frame V2	1997-2002	5,742	36,731	9,420	8.0	6.4
Total Server Frame		**11,140**	**53,342**	**15,531**	**6.2**	**4.8**
Client frame V1	1993-1999	3,989	9,537	3,263	3.2	2.4
Client frame V2	1999-2002	5,717	7,049	1,047	1.4	1.2
Total Client Frame		**9,707**	**16,586**	**4,310**	**2.2**	**1.7**
Broker	2000-2002	214	2,151	0	10.0	10.0
% of client frame V2		3.7%	30.5%	0.0%		
Webulator	2000-2002	2,815	3,284	85	1.2	1.2
% of client frame V2		49.2%	46.6%	2.0%		
Total derivative products		**3,029**	**5,436**	**85**	**1.8**	**1.8**
Overall software application frame		**20,847**	**69,927**	**19,841**	**4.3**	**3.4**
(without derivative products)						

Product	Timeline	Analytical Application Frame Efficiency	Analytical Application Frame Gross Margin (Thousands in currency)	Number of Customers	Gross Margin per Customer (Thousands in currency)
Server frame V1	1990-1996		11,212	262	42.8
Server frame V2	1997-2002		30,989	293	105.8
Total Server Frame			**42,201**	**555**	76.0
Client frame V1	1993-1999		5,547	429	12.9
Client frame V2	1999-2002		1,331	82	16.2
Total Client Frame			**6,879**	**511**	13.5
Broker	2000-2002	3.7%	1,937	35	55.3
% of client frame V2					
Webulator	2000-2002	49.2%	469	29	16.2
% of client frame V2					
Total derivative products		**0.31%**	**2,406**	**64**	**37.6**
Overall software application frame			**49,080**		
(without derivative products)					

Fig. 33. Key Indicators for Software Application Frame Development.

Meyer et al. (1997) did not include discussion of any type of installation cost/revenue and its impact on the effectiveness and efficiency of a product platform. Cusumano (2003) discussed in his article "Technology Strategy and Management: Finding Your Balance in the Products and Services Debate" the issues of whether a company should concentrate on building its software business or whether it should provide both software and consulting services. The measures in Figure 30 are divided into two main server frames (V1 and V2) and two client frames (V1 and V2). Each of these four line items includes accumulated numbers for nine different measurement columns in the following way:

- Total costs = Total costs of development of the software application frame
- Total revenue = Total software license revenue plus corresponding maintenance revenue
- Service revenue = Total installation and consulting revenue for the solution at the customer site
- Software application frame effectiveness with service revenue = (Total revenue of all software products derived from the software application frame + Service revenue)/ Total costs of software application frame, its extensions, and derivative products
- Software application frame effectiveness without service revenue = Total revenue of all software products derived from the software application frame/Total costs of software application frame, its extensions, and derivative products
- Software application frame efficiency = Total costs of the derivative product/Total cost of the software application frame version
- Software application frame gross margin = Total revenue – Total costs
- Number of customers = Number of individual customers for the software
- Gross margin per customer = Software application frame gross margin/Number of customers

Both the total cost and the total revenue have been accumulated from annual sales numbers collected by the CFO and Controller of the case study company. Service revenue numbers were allocated according to the domestic end user organizations that could be traced using financial information. The two main indicators from Meyer et al. (ibid), platform efficiency and effectiveness, were complemented with service revenue information that could in some cases demonstrate whether a software application frame has been an effective foundation for the software solution or whether the software application frame requires additional effort when the software is installed at the end user location. Effectiveness number could be dramatically different if installation of the software is difficult, as the numerator will grow, causing the effectiveness number to show values which are too high. When the analytical application solution vendor spends a lot of time customizing the solution for the end user organization, the prepackaged solution loses its intended function of rapid and cost effective delivery.

Efficiency calculations are based on how successfully a software company can derive new products from a software application frame, but as our case study company did not have any true derivative product development from the server frame (even if the same frame was delivered to hundreds of end user organizations), we decided not to include any frame efficiency calculations on the server frame. The case study company had two derivative products for its client frame (V2) that were the foundation for derivative software development. The initial derivative product, "Broker," provides end user organizations with centralized meta data management. The second derivative product, "Webulator," gave end users thin-client web access to the analytical information.

7.5.4.1 Comparison of software application frames

Figure 34 enables us to compare the overall costs and revenue streams for each frame release (see Fig. 34).

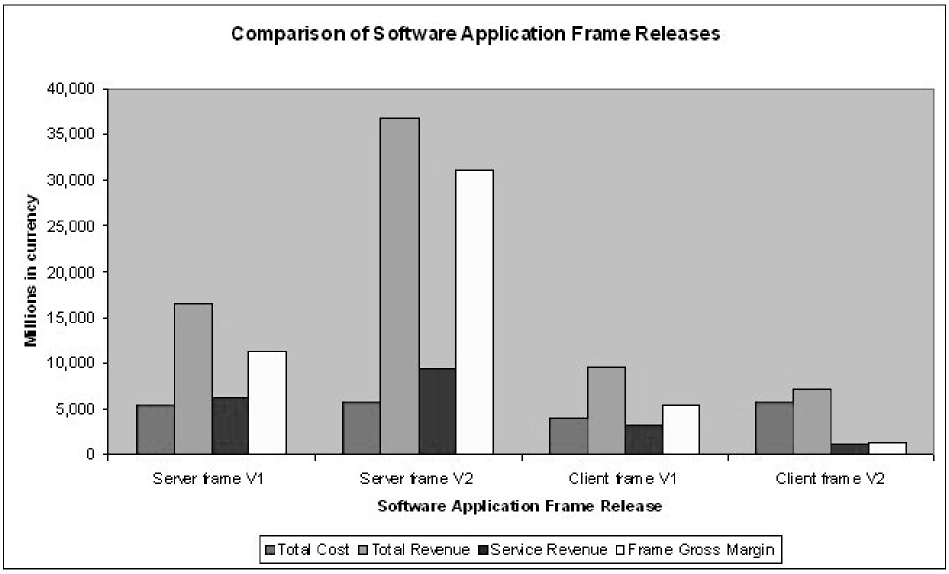


Fig. 34. Comparison of Software Application Frame Releases.

Figure 34 demonstrates that server frame V2 has generated most of the revenue for the case study company when compared with the other software application frame releases. In a similar way, it is obvious that server software has been the core of the solution offering over the years. The client application has enabled the Company to offer a complete package, with OLAP server technology and an associated analytical user interface. Another finding from Figures 33 and 34 is that client frame V2 has yet to show a return on investment (ROI), as the total cost is very close to the total revenue received from the client software offering. It is also obvious that the total cost of client frame V1 is considerably less than the total cost of client frame V2. This can be explained by a much larger development organization working on the replacement of the previous client frame V1. The service revenue from the server frames is also higher compared with client frame service revenue. This is due to the fact that most of the time spent in installing a solution is in the integration of the analytical application solution to operational data sources. This could in some cases take time and effort due to data consistency issues within the end user organizations.

Client frame V2 has been on the market for a few years, having accumulated sales by which the costs have been covered (effectiveness ratio of 1.4). Client frame V1 did not have any derivative products, while client frame V2 had two derivative products that used common software assets from client frame V2. From a cost perspective, the combined client frame cost (V1 and V2) and the combined server frame cost (V1 and V2) are almost identical (client frame cost is 87% of server frame cost), but when the total revenue is compared, the client frame gross margin is only 16% of the server frame gross margin. In an interview with the Server Development Manager, we concluded that the pricing model for the client product has been too flexible and cost effective from the end user organization perspective, as the concurrent user pricing model has not generated significant additional revenue from the existing customer base.

It is highly probable that named user pricing could have brought the case study company ongoing software sales revenue which in turn would have had an impact on the gross margins of the client software. Based on these findings, the Board of Directors of the case study company decided to move into the named user pricing model in late 2002. as it became evident that the case study company had lost a considerable amount of ongoing software license revenue when selling concurrent user licenses to end user organizations. One of the main reasons for this is that the statistical probability of exceeding the concurrent count decreases with the number of users. With a user community of 50 users, a concurrent model of 25-30 users is required, but when the concurrent user license count is increased to 100, it will service a much broader user community than 100 users (anything up to 150 to 200 users). The more concurrent user licenses an end user organization buys, the less probable it becomes that the concurrent count will be reached.

7.5.4.2 Analytical application frame effectiveness

Initial server frame V1's effectiveness without service revenue is around three: the total cost has been well covered over the years. The share of service revenue as a percent of total revenue is 36, which gives an idea of the additional revenue that the Company has received during the lifetime of server frame V1. Server frame V2 has more impressive effectiveness measurements, with frame effectiveness of 6.4. One of the reasons for this effectiveness is that the number of customers increased during the latter part of the 1990s, and the case study company did not increase the amount of investment in the server frame in the same proportion as software license and maintenance revenue grew. The share of service revenue as a percent of total revenue decreased during the lifetime of server frame V2 to 25 percent. The reason for this could be twofold. The first reason is that the share of revenue from internal business grew (it did), and this revenue is not recognized in the company (not known) other than the service revenue generated by subsidiaries. The second reason could be based on the policy of the Company of not customizing its software products to reflect any specific end user organization requirements. Therefore, the same product will be delivered to every organization regardless of vertical or horizontal market segmentation.

The adaptation of the solution to different end user organizations is implemented via built-in parameterization of the software solution. Interestingly, each customer has exactly same data model: only the contents of this database distinguish them from each other. We argue that this type of innovation has enabled the Company to reuse the same server frame in hundreds of different organizations without any customization. Based on our analytical application frame architecture as portrayed in prior chapters, the Company has not utilized the implementation of analytical application solutions using a business analytics and meta data repository as defined in Chapter 4. The Company could have improved its software application frame efficiency and effectiveness by creating vertical market solutions with predefined key metrics and business processes and priced these solution based on the vertical market segment. According to our interviews with the key developers of the Company, these types of market segmentations have to be supported natively in the product to enable effective software version management. This type of management is planned for future releases of the server frame architecture.

7.5.4.3 Analytical application frame efficiency

We concluded in Chapter 5 that an analytical application software vendor has three alternative ways of implementing analytical application solution variation. The first and most common way is to provide a vertical market solution for a specified market segment using components of an analytical application defining technology architectural layer (business analytics repository, business rules, measurements etc.). The second and more difficult variation technique is more or less variation using technology in the software implementation itself. The latter alternative is obviously more error and risk prone and might not provide the same leverage ratios when compared with pure solution variation. The third variation technique is based on using market leverage via price/performance adjustments or by providing the complete solution and controlling the functionality using control numbers. This type of variation technique is by far the simplest, and it has been the main variation technique for the Company throughout its history.

An example of technical variation was demonstrated by the Company when it introduced its two derivative products from the client frame. The variation was pure technical variation using a common software asset base. It did not provide any means of solution content or market segmentation variation. The decision to create a common software asset base was done at the time of renewal of the client frame (V2) to enable different client technologies (such as Pocket PCs, Palms, Internet browsers etc.) to be used for information distribution and access. The result of this decision was the implementation of two derivative products and one prototype from client frame V2. The first derivative product was for centralized meta data management and the other for web-enabled data access. The product for centralized meta data access (Broker) included almost every aspect of the client software product, with the addition of a few additional features such as configuration and scheduling functionality.

Broker has efficiency numbers of 3.7 percent of the cost of the overall client frame. The effectiveness numbers for Broker show that the investment has already been recouped tenfold. These numbers are more or less based on a technology driven efficiency and effectiveness initiative, as both of them are pure technological derivatives and not based on a specific solution or vertical market segment. Broker internally includes a meta data repository created "on-the-fly" from the server software to provide load-balancing and distributed characteristics for the solution. The other derivative product for web-enabled access has required more development effort from the case study company. In a similar manner, a lion's share of the common software assets from the client frame are reused within the Webulator environment, with the addition of specific Internet specific technologies (such as web server technology, scripting language in the Microsoft web environment, etc.). Our calculations show the cost of Webulator to be 49.2 percent of the overall cost of the client frame. Most of this cost is due to additional technologies that have been applied with the client frame, technologies that do not exist in the pure Windows client software. Even if Webulator is regarded as a derivative product, it is evident that the development team had to put a considerable amount of effort into keeping the derivative product in sync with the client frame itself due to additional technologies that Webulator required (such as the Internet environment). As the investment of Webulator has been close to 50 % of the total cost of the overall client frame V2, the question remains as to whether this can be regarded as effective derivative development.

7.5.5 Key Findings in Economic Analysis

We were able to identify one type of variation technique that the case study company had used. This variation was implemented in the client frame environment with reasonable efficiency numbers. We also concluded that the Company had not used variation in its server frame environment, and that the case study company could have achieved considerable additional leverage by using solution variation. This provides a contribution to other analytical application software vendors, as the derivative development of solutions should be easier to implement than solution variation using technological means such as those was introduced in the software engineering literature.

However, these types of software application frame key indicator estimates can be extremely deceiving, as the effectiveness and efficiency of the software application frame do not necessary reveal whether the competitiveness of the software application frame has been kept up-to-date with the required investments. In some cases, the software company could decide to use the product as a cash cow, where the effectiveness ratio would be extremely good due to smaller software application frame investments. In the long run, this would have an adverse impact on the software application frame's competitiveness and robustness. Another factor that could impact efficiency and effectiveness numbers is the architectural robustness of the solution. In our interviews with the development organization, we noted a few deficiencies that had impacted adversely either server or client frame development. Optimization in one software application frame might lead to poor performance in its counterpart.

It is also evident that a software vendor is required to have a good business strategy when setting up its metrics with respect to service revenue. This revenue type as such does not show any impact on how effective the software development organization has been. We argue that if considerable time is spent on installation of the analytical application solution in end user organizations, the analytical application frame architecture is weak or does not provide a solid technological foundation.

All in all, the case study company has been able to show moderate effectiveness measures during the lifetime of both the server and the client frame. It was not a surprise to us that the server frame was by far the more profitable software application frame due to its importance (it is the core of the solution and is priced accordingly). The development organization has little experience in technical solution variation with respect to client frame derivative product development, and it remains to be seen whether similar efficiency numbers can be seen in future client frame releases.

7.6 Key Findings of the Case Study Research

One of our research objectives for this dissertation was to obtain development ideas and provide feedback for the improvement of an analytical application frame construct. No prior empirical studies were made of Software Application Frame Strategy framework usage. This study focuses on the analytical application software domain. We were able to identify a few practical issues that a software development organization could potentially run into when implementing and planning analytical application solutions using analytical application frame constructs.

7.6.1 Discussion of the Analytical Application Architectural Model

In our analysis of the case study company, it became evident that a software vendor must clearly distinguish between the domain-specific architectural model and the IT infrastructure or execution and development environment. The former is typically extremely important for end user organization information technology departments, while the latter is internal to the software development organization. End user organizations do not typically care how the internal software architecture works, as this will be invisible to end user organizations. If the selected internal product architecture requires specific runtime environments due to its architectural model, this will obviously reflect on the external IT infrastructure selections, and therefore will be a consideration for both the users of the software and for the software organizations. This type of analysis is an interesting cross-section of the traditional information systems science and software engineering/computer science research domain.

Both of these environments have different aims; therefore we argue that it would be a mistake to take only one into consideration. We explored several different data warehouse architectural models in Chapter 3 that could become the foundation for an analytical application solution. A more traditional centralized model, with a traditional development approach (not using software components and clear application programming interfaces) could be more appealing for software vendors who lack core competence in modern software implementation methodologies. Unfortunately, these types of centralized applications might not satisfy the needs of highly distributed environments, where subsidiaries or divisions of organizations require their own localized meta data and business analytics repositories. This highly distributed environment could become a nightmare for software vendors to maintain. From a conceptual perspective, a federated architectural model combined with distributed/ multi-tiered characteristics is more appealing to end users, as each of the remote and local databases is transparent to the end user: the user interface will display analytical information regardless of the physical location of the data. The architectural model in our case study company was initially client/server and later multi-tiered, with a middle-tier server acting as the front end to the iSeries/400 environment.

This layered distributed software architecture has enabled the case study company to utilize and scatter software frame components across different architectural layers and build new derivative products by using the client frame architecture and its software components. Several interviews with the development team members also convinced us that the development would not have been efficient without the use of component-based software engineering. Component-based software engineering as such does not provide any competitive edge, but the companies that are able to build and design optimal software application frames that include a multitude of different functionality will have a competitive edge in the future. According to different studies, component-based software development as such has not provided tremendous results, as software organizations have not been able to manage and align the required business processes with the technical implementation.

7.6.2 An Analytical Application Frame and its Identification

The task of identifying a software application frame is by no means trivial. Even though we knew the case study company well, several analysis iterations were required to identify each and every software application frame version from the software product releases that the Company has had in its history. We based our principles of identifying a software application frame on the four principles presented by Sääksjärvi (2002). Sääksjärvi's (ibid) principles are based a collection of definitions that have been defined in the product platform literature. These principles are specifically addressed as generic software application frame principles: they are not geared to any specific software domain.

We initiated our identification of a software application frame within the case study company by listing all historical software product releases in an attempt to establish some type of historical timeline. This gave us a better understanding of how each software release has contributed to the overall value that the Company provides to end user organizations. Once we had a foundation sufficient to allow us to understand what had been accomplished, we interviewed several development organization members and management to find out what types of changes each software product had required in the form or modules, software components, and architectural models. By several iterative rounds of interviews and documentation, we were able to establish the link between a software product release and its corresponding application frame version. We had to identify how each software application frame release had been changed from its previous release to identify whether there were signs of either software application frame extension (principle three) or whether a complete renewal had taken place (principle four). We used the four principles to analyze which type of release was in question. When we identified an extension, we interviewed the responsible development manager or chief architect to determine whether this extension had disturbed any other modules within the software application frame itself.

We recognize the difficulty of separating regular "legacy incremental development" from development of a software application frame with the aim of large-scale reuse. Our view of the historical server and client software development is that the Company was somewhat lucky to have envisioned the concept of building a large architectural construct that could be reused across any given vertical market segment. In our interview and discussion of principle one (large-scale reuse), the Director of Product Strategies stated:

> "We really had the vision of creating a generic product that everybody could use and we had strict rules that we would not implement any customer-specific functionality. Every time we closed a deal with a new vertical market, we analyzed carefully what was missing in the product and if we found something that could benefit every company, we would then implement it."

One could claim that this was not really "large-scale reuse" from a software development perspective, but it is obvious that the case study company had a vision of how it could achieve replicate success by having a large generic application that could be reused across any given vertical market. Another factor that we were also able to conclude based on our ex-post analysis was that the Company did not have any plans for product family development as such during the first analysis period (1990-1998). Only with the renewal of the client frame in V2 was the development team able to create two derivative products that belong to the same product family.

7.6.3 Software Application Frame and Large-Scale Reuse of Common Software Assets

Software product line development and software product family development are treated equally in this study. Some literature sources conclude that in product line development, the underlying IT infrastructure might not be the same for the derivative products. In product family development, the idea is to share common software assets using the same IT infrastructure. The latter concept is closer to the analytical application frame construct, as the overall idea in software application frame development is not to try to satisfy all the possible future market segments, but to maximize the flexibility with the resources and core competence that the company has at each point in time. Based on our Application Frame Strategy framework, there is no single optimal alignment perspective, as each software company is different, with different characteristics and core competence.

A challenging question that each software vendor must deal with is how to define a software application frame and how to maximize the size of this frame to enable maximal software asset reuse in derivative software products when using an analytical application frame construct. The case study company had two different identifiable software application frames with very different underlying technologies. The server frame is highly bound to iSeries/400 technology, while the client frame is based on Microsoft technology and can therefore be used only in Microsoft Windows environments. We interviewed the server Development Manager to find out whether the iSeries/400 had a true module structure in its early days, and the answer was as follows:

> "In the early 1990s, before ILE RPG became available, the development environment was not really set in modules, but more in procedure calls. When IBM introduced ILE RPG (which enables the mix-and-match of programming languages like ILE C), IBM came out with a more structured module architecture."

The Company has included other software modules based on other languages such as C++ and Java in later stages, which has had an impact on how the software application frame is planned and constructed. For example, Java modules will call RPG modules or C++ modules. It is fair to say that during the second analysis period, the internal architectural structure of the server frame has become more layered: internal modules can be replaced with other underlying technologies that support other operating system and hardware environments. In the case of the server frame, we still need to see improvement, specifically in frame efficiency, as the Company has not built any vertical market segmentation strategy whereby the server frame could become a software application frame for derivative product development.

The Company needs to improve its server frame to include functionality to support true versioning of analytical solutions. Our interviews with the senior development managers revealed that the Company is in the process of deploying solution development features in the new server frame, due for release in the next couple of years.

The client frame renewal effort showed the power of software asset reuse, as two derivative products were created from a common client frame. Our efficiency and effectiveness measures were also able to demonstrate this specifically with the first derivative software product, "Broker." As the product line software engineering community has concluded, derivative software development is by no means without challenges. Different articles concerning variation mechanisms and architectural models have been published the last few years, as described in Chapters 2 through 5. According to our interviews, versioning and maintaining a common software asset base is very challenging, requiring careful planning. Each change in the core asset base could have an impact on any of the derivative products.

7.6.4 Identifying the Use of an Analytical Application Frame Strategy Framework

We asked ourselves how these alignment perspectives are best identified in a software company, and whether an "optimal" alignment model will be generic to all software vendors in the same domain or even across different software domains. Our approach to the analysis was very pragmatic, as it was mainly based on interviews of current and prior personnel to identify whether any specific sub-strategy within an Analytical Application Strategy framework had received more emphasis than others. It was of interest to us to see if any one specific fit (relationship between two sub-strategies) has prevailed more than others. We wanted to identify whether the full alignment perspective was evident in any of the decisions made when setting product development strategies within the Company.

Based on the findings in our case study research, we were not able to identify any specific alignment perspective that could have been said to be optimal. There are multiple reasons for this finding. First of all, each decisions made within the Company was made in good faith with the best knowledge available at time. Some decisions would probably be the same today as they were a few years ago due to the background of the Company, such as our close relationship with and knowledge of the iSeries/400 hardware environment. Some decisions would probably be different today, such as the decision to base the planning/forecasting application on Microsoft Excel. According to our interviews with the Product Manager of these Microsoft Excel-based applications, the need in organizations today is to enter their budgets using an Internet browser, not a Microsoft Excel spreadsheet. This type of change in requirements from end user organizations has put the case study company under pressure. Unfortunately, not much can be done due to the technological foundation of Microsoft Excel.

A second reason for not being able to specify an optimal alignment model is that businesses could change quickly based on the surrounding market environment, and therefore one "optimal" alignment perspective could become inefficient in a short period of time. A third reason for not finding an optimal alignment perspective is due to changes in the software application domain itself. An example of this is the decision support market and its transition from highly centralized to distributed architectural models. These types of changes could potentially become too difficult for some vendors to cope with due to weak core competence in the technology or software application domain. Our case study company initiated its software product offering in the IBM midrange S/36 environment with a Harvard Graphics user interface. 15 years later, it has a solution offering for web-enabled analytical application using different types of technologies that did not exist 15 years ago.

Regardless of the challenges that we faced in analyzing the existence of alignment perspectives in the case study company, two specific alignment perspectives were more obvious that others during the two analysis periods. First of all, the more prevalent alignment perspective has been "Implementation of technology in the analytical application frame." The second has been "Harnessing market potential using the analytical application frame." The first alignment perspective was identified when we compared each sub-strategy with every other sub-strategy. Most of our interviews included statements such as "we had to increase the performance of server software" or "the client was very unstable when we added new functionality." These types of statements demonstrate that the development team was very focused on improving the server software performance or meeting the end user organization requirements in the client software. There was very little discussion of market segmentation or implementation of vertical market solutions using the software solution. It is also evident that "Technological Responsiveness" was a key question for the Company over the years. In some cases, we saw signs of a need to improve "Functional Flexibility." The latter was evident when the server frame was improved to handle larger multidimensional OLAP databases in larger end user organizations.

We defined three different types of disconnects in Chapter 4. Each of these disconnects is related to the alignment perspective in question. When the alignment perspective called "Implementation of technology in the analytical application frame" is identified, there is increased potential for "Segmentation disconnect," because an analytical application frame leverage sub-strategy might get less emphasis.

Let's review what this type of disconnect entails in our case study company. We argued that in server frame renewal (V2R1), most of the internal modules were rewritten and the Company selected OS/400 specific technology to enhance the performance of data loading, update of the database, and data deletion. This technology selection (user indexing) was made because the hardware operating system environment was "given" and not because this specific technology could exist in other operating system and hardware environments. Therefore, a technology driven decision was implemented without relating it to future market segmentation. This will restrict and/or increase the costs of moving the solution to other environments in the future if it is so decided.

In summary, selection and implementation of an Analytical Application Frame Strategy within a software company is a multifaceted task, which requires new thinking from executive management and the marketing and software development organizations. Based on the information provided by the case study company, we were able to identify numerous factors in analytical application development that make the use of analytical application frames very challenging. One of the main reasons for the challenge is to keep each sub-strategy within an Analytical Application Frame Strategy framework well integrated/balanced, as each sub-strategy could impact the other if not selected carefully. Finally, we do not believe that one strategy will be the same for all software vendors, as the background of the companies will differ from case to case. Therefore, the framework will be the starting point for the software vendor, which must adjust it to reflect its own requirements.

7.6.5 Management of Dependencies Between Underlying and Defining Technology

Based on the numbers presented in the case study, it is evident that an analytical application frame architecture and its frame component selections are closely tied with the underlying technologies (operating system environment, database management system etc.) that an analytical application solution vendor selects in the process of defining its analytical application frame technology sub-strategy. The linkage between the analytical application frame technology sub-strategy with its corresponding underlying technology and the analytical application frame architecture sub-strategy with its defining technology, "Technological Responsiveness," can result in several dependencies in the future development effort, as could be seen in the case study company. Selection of underlying technology such as the OS/400 operating system with its embedded relational database DB2/400 has kept the Company out of other hardware environments and market segments that are known not to support the iSeries/400 environment. We can therefore assume that two of the most critical selections that the Company can make in the case of analytical application software development are the underlying technology and the defining technology. Unfortunately, it could be challenging to have executive management in software organizations separate shortsighted revenue recognition from a long-term commitment to a robust software application frame that will serve the software organization for years to come and generate revenue on a continuing basis.

The complexity of analytical application defining technology elements such as business analytics repositories, data models, and predefined ETL processes and such are typically not very standardized amongst end user organizations, and therefore it is a true challenge for analytical application software vendors to build a solution that will be a perfect fit to any organization in a selected vertical market segment. Provided with adaptability of these elements, these end user organizations can use these solutions as a foundation for further analytical application solution development. Software vendors must provide a means to deliver new releases of the solution that can be readily merged with the customized business processes and models that these end user organizations might have built using the software package.

7.6.6 Generalizability of Analytical Application Frame Strategy Framework

The question of whether our Analytical Application Frame Strategy framework can be generalized to any other software domain poses a challenge. First of all, each software domain could have different underlying and defining technologies: the fit between software frame architecture sub-strategy and software application frame technology sub-strategy could be considerably different in different software domains. Market segmentation of software products is very different in different cases, as some products (such as office productivity tools) are clearly defined as consumer products while some are built for enterprises. These software categories pose completely different underlying assumptions of the execution environment. Consumer products (such as Windows) are sold by the millions, and their underlying infrastructure must be generally accepted, while enterprise applications could have a combination of requirements that can be fulfilled as long as these enterprises are ready to purchase and support these specific requirements (such as having an iSeries/400 server).

We believe that the process of identifying a software application frame, alignment perspectives, and corresponding disconnects will benefit other software domains. Each software domain will have its own characteristics, but the aim of large-scale reuse with derivative products within a product family remains the same. We believe that software application frame renewals and extensions will remain the same, as these concepts have been kept the same from original product platform theory and we do not see any reason for them to change.

7.6.7 Discussion of Case Study Results in Light of Meyer and Seliger’s Definition of a Product Platform

Meyer and Seliger (1998) concluded in their article “Product Platforms in Software Development” that a platform is “a set of subsystems and interfaces that form a common structure from which a stream of derivative products can be efficiently developed and produced.” Meyer and Seliger (ibid) constructed an architectural model for software products and named this architectural model the "Platform Strategy for Software Products." In this model, Meyer and Seliger (ibid) assume that the product platform itself is made of computing infrastructure and application building blocks, and that the common applications that are needed by any market segment are not part of the platform. We disagree with this, as the platform itself should consist of both the underlying technology and the defining technology with the common applications. Meyer and Seliger (ibid) consider the platform to be a "base software engine" and the platform to be composed of "developers, design strategies, and specific procedures and protocols."

In light of these definitions, the most obvious problem that the Myer and Seliger (ibid) framework could potentially cause is the lack of discussion as to how selected market segmentation might impact both technology selections and domain-specific (analytical applications) and IT infrastructure strategies. When reviewing our case study company, the selections of both the iSeries/400 technology for the server frame and Microsoft technology for the client frame were made a two-dimensional framework where the future impact and dependencies are missing, such as in the architectural framework presented by Meyer and Seliger (ibid). Meyer and Seliger (ibid) made very generic conclusions about the usefulness of the product platform approach for software products without taking into consideration the characteristics of different software domains.

We argue that our study has increased understanding of the domain of software application frame development, and that software related product platform knowledge concerning different software domains must be adjusted based on each domain's characteristics. The work of Meyer and Seliger (ibid) did not cause us to question what type of differences each software domain could potentially have, but when reviewing the analytical application software domain by aligning three sub-strategies (analytical application frame architecture sub-strategy, analytical application frame technology sub-strategy, and analytical application frame leverage sub-strategy), we were able to achieve a better understanding of how each sub-strategy can impact the others in different ways. This was demonstrated via different alignment perspectives and corresponding disconnects that can impact an analytical application software vendor. This type of analysis added new practical perspectives to the existing traditional and software related product platform literature.

Meyer and Seliger (ibid) did not analyze the practical implications of developing common software assets for a product platform. The contribution of this work is to link product line software engineering with other software engineering approaches and explain how these can be used in conjunction with software application frame development. Meyer and Seliger (ibid) did not include discussion of different variation techniques that a software vendor can perform while working with derivative products. Different scenarios determine how common software assets can be used across different products using a reference (product line) architecture.

And finally, had the case study company decided to choose the underlying technologies without binding itself to a specific operating and hardware environment, the results could have been different. Obviously, in ex-post analysis it is very easy to identify problems in the decisions made, and therefore it is important to understand that some decisions are made due to better alternatives, alternatives that occurred once the development had been started and changes were too late.

7.7 Chapter Summary

The question of whether the case study was able to support our research objectives and whether our results can be generalized and applicable to other analytical application software vendors needs to be analyzed from several different perspectives. First of all, no prior research could be found on analytical application software development from any software vendor's perspective: most of the written articles measure the analytical application software domain from the end user organization perspective. Secondly, no prior research had been done to explain how a software vendor should implement its technology strategy and how these technology selections can impact other selections defining technology in an analytical application frame architecture with its corresponding architectural layers, together with its underlying technology and associated execution and development environment. A third area of research in this study was to identify the structure of an analytical application frame architecture with all corresponding functional software application frame components. This architecture will be helpful for any analytical application vendor to be able to identify the needed software application frame components when building analytical solutions. Another clear advantage of our framework and case study was to show the importance of analytical application frame architecture and how each layer in this architecture will impact development of an analytical product solution. This type of analysis has not been published before, and this will be of interest for similar software organizations. We were able to draw parallels between this architectural model and the renewal of the client frame architecture (V2) within the case study company.

We also undertook a thorough discussion of different data warehousing architectural models that an analytical application software vendor can utilize in its analytical solution development. One of the most prominent models is the federated data warehouse model, which enables an integrated view of multiple data marts. An important distinction that a software vendor must make is to separate the technical architectural style from the domain specific architectural model that is specific to the software application. This separation is extremely important, as these two architectural models could be in conflict with each other, and this could impede market segmentation (analytical application frame leverage sub-strategy) for the Company. It became evident in our analysis that the case study company has moved from a highly centralized data warehousing model to a distributed model. Russell (2000) concluded that moving from a highly distributed architectural model back to a centralized model can be laborious. This is what we also found in our case study analysis. This type of architectural movement will typically result in major renewals of subsystems, and therefore analytical application software vendors should consider the architectural basis for their analytical solution very carefully, as any movement from one architectural model to another will be extremely costly.

The fourth major research result in this study was the identification of six different alignment perspectives and how these can be used when analyzing the effectiveness and efficiency of application frame use in derivative product development. These alignment perspectives were used in our analysis of the case study company to identify which perspectives had been used and what type of impact each perspective with its corresponding fit and disconnect has the development of software product solutions. It became evident during our analysis that the case study company could have achieved better results in its software development if it had known about the Analytical Application Frame Strategy framework and the type of impact each sub-strategy would have had on the overall product development strategy. We also argue that these six alignment models can be utilized in other software organizations and software domains as long as the characteristics of each sub-strategy are taken into consideration in the analysis. We also argue that each software domain will have a modified version of the generic Software Frame Strategy framework identified by Sääksjärvi (1998, 2002).

To be able to measure our theoretical findings, we divided our longitudinal single-case analysis into two analysis periods. The first time period represents the time before major application frame renewals and the second analysis period included renewal of both the server and the client frame with several extensions in the server frame to enable satisfaction of the needs of end user organizations. By studying the case study company for a longer time period, we got a better understanding of the product evolution and set the historical events into their own context. The case study company has evolved from a highly entrepreneurial organization into a professional and international organization, with hundreds of end user organizations using its software solution.

During the first analysis period – from the foundation of the case study company through 1998 – the Company built its initial server and client frames for both server and client software products. The Company also had its initial successes in international markets. This time period did not include any major market segmentation strategy other than to provide solutions for the IBM midrange server market. The founders of the Company had a strong vision of creating a product that would enable end user organizations to analyze information that resided on an IBM midrange server. This showed an emphasis on creating a software application frame with a corresponding architecture to fulfill the requirements of end user organizations. Our analysis did not show any sign of an Analytical Application Frame Strategy other than construction of a solution according to the vision set by the management and owners of the Company. The business model for the case study company was initially very simple, with one major underlying technology (iSeries/400) and only one application area (OLAP technology). With the increased pressure from end user organizations, the Company had to add additional layers (Microsoft Office-based solutions), and new technologies had to be put in place.

The second analysis period – from 1998 to 2002 – was mainly marked by two major improvements in product and market segmentation strategy. First of all, the Company purchased technology from a third-party vendor to be able to provide an additional architectural tier that provides to end user organizations the ability to build large enterprise wide analytical solutions. The second improvement included a full renewal and replacement of the existing client frame. This new frame enabled the Company to build several derivative products using common software assets. The renewal of the client frame was a new start for the Company, providing a much needed competitive edge over other vendors in the same market space. The renewal of the client frame was the first intentional strategy from the management of the Company, enabling construction of a true software application frame that would be the basis for derivative product development. We also completed analysis of software application frame effectiveness and efficiency according to the measurements set by Myers et al. (1997). The results of our analysis show clearly that the use of a software application frame helps software organizations to derive products from a common software application frame (measured by efficiency calculations) at a fraction of the cost that it would take to build these products from scratch. Our software application frame effectiveness numbers did not yet show whether the software application frame investment has been successful, as the platform investment has not yet been recouped for all derivative products and the client frame itself.

When comparing all technological selections in the case study company in the light of technology selections, it is very clear that the management did not anticipate future market segmentation strategies, as most of the decision support server technology is based on proprietary technology. This has become in some cases a technological restriction for the Company. Selection of IBM midrange technology could impact future market segmentation strategy, as IBM midrange technology might not be broadly accepted in some vertical market segments. The product development organization based the underlying technological foundation for the client frame on the Microsoft execution and development environment. This is the second time in the company's history that a technological selection could impact the future of the Company. Going forward, the case study company and its management should carefully analyze different alignment perspective alternatives as defined in the Analytical Application Frame Strategy framework. From a business analysis perspective, the second analysis period increased the complexity of both software development and dependencies on the end user organization IT infrastructure. Additional technological selections have caused the product development team to spend more time analyzing IT infrastructure dependencies with associated customer bug reports.

Another objective of this study was to test whether the development of analytical application software using an Analytical Application Frame Strategy framework would change the traditional way of implementing analytical applications. To be able to test this objective, we compared different data warehouse architectural models and concluded that a layered architecture will support distributed computing, which has become a requirement within analytical application development, but that this will also change the way we plan and develop software. No longer will we think in terms of individual functional software components, but software organizations must build software application frames that need to be adaptable to future products that are not yet known and requirements for products which are more or less unknown. As software development firms are moving closer to mass-producing software, software vendors will concentrate on building effective application programming interfaces that are well documented and published to the third-party software development community. This type of development could eventually lead to a large third-party development community that innovates software solutions that complement the application frame developed by the originating application frame software vendor.

We explained the importance of having an adaptable software application frame to accommodate different vertical market analytical solutions. This adaptability must be built into the server frame. The flexibility must also include easy integration of different operational applications such as ERP solutions, financial packages, or other operational solutions that will be the source for integration. Our recommendation for maximum flexibility is to provide a business object layer between the physical and logical representation of the data and business process models. This business object layer enables analytical application software vendors to build solutions that can be more easily modified when changes occur in the operational applications. Some of this flexibility can also be achieved using a data staging area that resides between the operational data source and the data warehouse or data mart application. Our analysis of server frames revealed that the Company had not planned to build analytical solutions by building "plug-and-play" application interfaces to different operational sources. This could be concluded by analyzing the extraction, transformation and load (ETL) functionality ("gateways") and how business rules with their corresponding logic were linked to the database model itself.

The case study company has externalized most of its client frame development to a third-party development organization. Our recommendation for any software organization using third-party development organizations is to set quality criteria for the development and integrate software development with the internal software development processes of the ordering company. The case study company has kept very close relationships with its development partners, providing and exchanging information via email and integrating the third-party organization into its software development process. Once this type of integration has been accomplished, the use of external software development organizations is not a problem as long as the relationship continues long into the future. We expect this type of outsourcing activity to increase in the future as well due to lower cost structures in countries and regions such as India, China, and Eastern Europe.

8. KEY FINDINGS AND SUMMARY

The objective of this study was to explore how the product platform approach can be used in the development of software products (Chapter 2) using software application frame(s), and to combine this research with the analytical application software application domain (Chapter 3). These two chapters provide the foundation for Chapter 4, where we proposed a redefined Software Application Frame Strategy framework specifically tuned to the characteristics of an analytical application software domain. As existing software related product platform literature does not specifically address the complexities of software development, we added a chapter (Chapter 5) to explore possible implementation techniques for derivative software products using a software application frame. In summary, Chapters 2 through 5 provide interpretative analysis of the development of analytical application frames. Our case study provides managerial implications that a software vendor can utilize when defining, managing, and developing an application frame using the Analytical Application Frame Strategy framework.

Using the findings from Chapters 2 through 5, we applied the Analytical Application Frame Strategy framework using a longitudinal single-case interpretative case study of a European software vendor. The case study allows us to review software development by a software vendor over the past 15 years using multiple research strategies, such as quantitative analysis of software application frame effectiveness and efficiency, and also qualitative research methods based on numerous interviews conducted during the past few years. We also benefited from the availability of extensive documentation of product development plans, financial records, and other records that related to the product development efforts of the products under study. To be able to test the validity and reliability of our research, we decided to base our case study research strategy on the Hermeneutic Circle from Klein and Myers (1999), as they have defined seven principles that must be satisfied when evaluating an in-depth single-case interpretative case study analysis. As part of our case study analysis, we analyzed how our findings and case study analysis were applied in light of these seven principles.

It was evident to us that no prior research has been published about analytical application software development from a software vendor's perspective. Most of existing literature concentrates on topics concerning implementation of analytical applications within an end user organization. We also reviewed several years of *Journal of Data Warehousing* articles, none of which had any reference to a software vendor. Therefore, we could conclude that organizations such as The Data Warehousing Institute concentrate mostly on end user organizations: it has not published any reports, articles, or books that would help software organizations to build software solutions or to explain what kind of impact different software related technologies have on development of analytical application software solutions. Therefore, the contribution of this work is to provide more information about the complexities of software development specifically for analytical application software vendors, describing how a software application frame could help these organizations to get better leverage using common software assets that are the foundation for a software application frame.

We also included discussion of whether our research results can be generalized and whether the results can be applied to other software application domains. Software development is still driven by the software application domain, and some generalizations will not be applicable to all software application domains. This is also one of our contributions in light of our refined Analytical Application Frame Strategy framework. Finally, we included discussion of possible future research and the limitations of this study in light of the selected research methodology and research results.

8.1 Use of the Product Platform Approach in Software Development

The first research question in this study, "How can a software vendor apply the product platform approach to its software business and development of software products?" required a thorough literature search. The results of this search are reported in Chapter 2. We reviewed both original physical product platform theory and known software related product platform theory, with the conclusion that existing product platform related theory exclusive of the work of Sääksjärvi (2002) is more or less geared to the theories and characteristics of physical product development. According to Sääksjärvi (ibid), software product development is more complex and requires software organizations to consider other aspects, such as the software process platform, as part of the overall Software Application Frame Strategy framework when setting its product development strategies. Because of this, product platform theory requires that additional strategies beyond product architecture be included in the design process. These strategies include those of software application frame technology and a software application frame leverage sub-strategy. Therefore, we wanted to extend our research to include additional questions about the feasibility of the product platform concept relative to the development of software products:

- What type of software development approach should a software vendor use when building the foundation of a software application frame with associated software application frame components?
- How does existing software product line development literature explain the use of the software platform, and how does that help existing software related product platform knowledge?
- How can the generic Software Application Frame Strategy framework be used in analytical application software solutions, and by how much do domain-specific characteristics change this framework?

We found two main sources of software related product platform development literature, with Meyer and Seliger (1998) introducing the concept of product platforms in software development. The authors related software development very closely to the development of physical products, while the work of Sääksjärvi (1998, 2002) addressed the complexities of software development in more detail using a generic framework without specifically addressing the needs of any software application domain. This framework, with three sub-strategies and their corresponding alignments, was introduced without any empirical consideration.

Existing product platform literature is focused on and borrows concepts from physical product platform development. We argue that development of software product platforms requires more elements that are specifically tied to software development. An important distinction between the development of software artifacts and the development of physical products is the dependence of the prior on an underlying runtime execution environment that is set by end user organization's IT infrastructure. This has not been clearly identified by existing product platform theory, and it is clearly a weakness that our work emphasizes more clearly. Dependencies on the underlying IT infrastructure must be a part of the technology selection that a software vendor will define together with its product and market segmentation selections.

We were not able to find any discussion of software application frame granularity and its corresponding software application frame components and what type of planning methods a software vendor should apply when defining a software application frame for a selected software domain. This type of planning relates closely to software product line development. We were able to identify several applicable methods, such as software domain engineering, that enable software vendors to define the commonalities of products within a product line. This helps these vendors to create common software assets. Bosch (2000) defined the concepts of software archetypes that define recurring patterns across a selected software domain: methods that can be found in component-based software engineering and approaches to defining coarse-grained software components. We also discussed software assets and how these can be categorized and used when implementing an analytical application frame. We found that existing software-related product platform literature did not specifically address different variation techniques that can be used when building derivative products. This study provides discussion of variation techniques that are available specifically for analytical application software vendors. The impact of a software application frame and its leverage on different market segments is measured by different economic metrics that were introduced in prior chapters.

8.2 Identification of Generic Analytical Application Architectures

The second research question, "What types of generic software application frame architectures can be identified for analytical application software solutions?" explored existing knowledge of data warehousing architectural models and how these could be linked to analytical application software solution development. Chapter 3 included an architectural description of all the required architectural elements that are needed in building an analytical application. Elements such as data warehousing, data mart technology, OLAP technology, and EIS technology are all part of an overall analytical application software architecture. We also concluded that software vendors must distinguish between application domain specific architectural models from the technical architectural style (such as client/server, pipe-and-filter, etc.), as these models could have significantly different aims. We introduced three data warehouse architectural models (centralized, distributed/multi-tiered, and federated) in Chapter 3, comparing these models from both the software vendor's perspective and the end user organization perspective (Chapter 4). In selecting an architectural model both from the software application domain perspective and the IT infrastructure perspective, the software vendor should aim to identify an optimal architectural entity that can be used across all derivative products in a product family. Based on this requirement, Chapter 4 portrays an analytical application frame architecture with all the included architectural layers, such as an analytical application defining technology, a service component layer, and an extension component layer. Each of these layers includes additional elements – analytical application frame components - that are the basis for derivative product development.

The emphasis on selection of an optimal data warehouse architectural model for analytical application software development must be a combination of end user organization and software vendor requirements. End user organizations might not support underlying technologies that a software vendor selects, and this could become a major hurdle for the software vendor in future market segmentation strategies. The software vendor should aim to achieve balance between software application frame architecture optimization and the IT infrastructure this software application frame is going to support, paying particular attention to derivative software development. The adaptability of the software application frame will largely dictate future market segmentation for the software vendor. A software application frame should not include all the possible underlying technologies that could possibly be needed for different market segments in the future, as this would cause serious compatibility problems for the software vendor. The analytical application frame should be implemented to be flexible, to accommodate current and future market segments effectively. This can only be measured ex-post and only if the software vendor tracks its software development activities effectively.

The differentiation between the domain-specific architectural model and the architectural style can be measured or evaluated by "Technological Responsiveness," which demonstrates the fit between the analytical application frame architecture sub-strategy with its domain-specific architectural model and the analytical application frame technology sub-strategy, which expresses the type of implementation architectural style the software vendor is going to use in its development. The selection process for the domain-specific architectural model must be based on both end user requirements and the core competence of the software vendor. The change from centralized computing toward distributed computing has also changed the way software vendors implement their solutions. Even though the technological implementation model is based on distributed technologies, end user organizations typically require that the end user experience centralized characteristics. This type of end user experience can be achieved using a federated architectural model, where all different data marts or data warehouses can be viewed using client software that provides an “integrated” view of the different underlying databases. We proposed a hybrid model in Chapter 3, where the selected architectural model is a combination of the federated and the distributed/multi-tier data warehouse architectural models.

We concluded in Chapter 3 that the added value of having a data warehousing architecture with federated characteristics is the ability to shield the layers between the operational applications and the data warehouse by using a data staging area. A data staging area acts as an interface to operational data and helps to stabilize the volatility of operational data. This characteristic is very important within analytical application development. This also helps software vendors to build analytical solutions, as the most frequent changes in ETL mappings are implemented between the operational data source and the data staging area, and not between the data source and the analytical application. Data staging area architecture is not very technology dependent, as most relational database management systems support the functionality of staging data for loading data into subject-specific data marts in the selected architectural model.

One key requirement for most end user organizations is to have the ability to use these analytical application solutions remotely without having access to a centralized meta data repository. This requirement requires some kind of multi-tier/distributed computing environment from the software solution. Selection of the federated data warehouse model from the software development perspective could be most effective for a software vendor, as it provides a centralized repository for both meta data and business analytics. This model also provides a seamless enterprise view of several underlying data mart implementations. This centralized approach assures that all dependent and independent data marts share common business measurements with conformed dimensions. This architectural model enables organizations to build enterprise solutions with a bottom-up implementation approach. Russell (2000) provided important findings in his article “Designing an Adaptable Business Intelligence Architecture,” where he concluded that movement from the centralized to the distributed architectural models is far easier than the other way around. Unfortunately, the case study company in this study had experienced the pain of moving from a highly centralized architectural model to a corresponding distributed architectural model, and now with the renewal of its server frame, the Company is aiming to change its architectural model to be more centralized due to several factors reported in Chapter 7.

From a software implementation perspective, the federated or distributed/multi-tier architectural model is more challenging than the traditional centralized model, as the software development organization must introduce new technologies such as distributed computing and data replication between different architectural layers into the analytical application to permit offline use of the solution. With a layered software architecture, the software vendor could theoretically make changes from a centralized architecture to a distributed architecture easier as long as these layers have appropriate communications protocols. If distributed technology is needed, message marshaling is managed by infrastructure elements in the underlying technology within the analytical application frame technology sub-strategy.

Based on this analysis of software architectures in light of the selected data warehouse architectural model, is obvious to us that irrespective of the selected architectural model, the software development organization must select a software architectural model that enables the implementation of an adaptable analytical application frame that can be used as basis for future derivative product development. The only way to achieve this type of flexibility is to select a robust software architectural style. Software organizations should aim to select an architectural model that will accommodate all future software requirements that might be set for the product solution, such as support for PDA technology or software agent technology that enables intelligent analysis of the data within the data warehouse or data mart databases. A layered software architecture will also reduce the dependences on the underlying IT infrastructure environment, facilitating easier portability for software vendors to move from one hardware and operating system environment to multiple hardware and software environments.

Finally, selection of an architectural model is not only dependent on what end user organizations will support, but the question is, “What is the most effective and optimized analytical application frame architecture that a software vendor can use when developing prepackaged analytical applications?”

8.3 Using a Balanced Software Application Frame Strategy

The third research objective, “How can software vendors balance their software product architectures when changes take place in marketing and/or technology selections?” is intended to give an answer to an analytical application software vendor as to what implications the software vendor can run into when aligning an analytical application frame architecture and analytical application frame technology sub-strategy with a selected analytical application frame leverage sub-strategy. It is evident based on our literature search of software-related platform literature that software development is more complex than the development of physical products. Changes in any of the three sub-strategies within an Analytical Application Frame Strategy framework will have an impact on software development. Unfortunately, these types of changes will typically go unnoticed by executive management, even though the dependencies created by a business decision could impact the future of the company. This result was reported in Chapter 7 in our case study analysis, where the Company extended its solution offering to other application areas such as budgeting/planning and enterprise data warehousing in the mid 1990s and early 2000s, resulting in selection of additional underlying technologies that made the overall software development more complex for the case study company.

We were able to identify six major alignment perspectives in an Analytical Application Frame Strategy framework that a software vendor can potentially utilize when building analytical application software solutions. We explained the impact of each alignment perspective on the overall strategy of a software vendor, and we also argued that one and only one alignment perspective will not be optimal for all software organizations. Each software organization has a different background, with different core competences and software development organizations. Each software organization conducts business in a different competitive environment, and each software organization might have a different business model. The selected business model could change rapidly due to changes in the business environment. This could cause a change in the selected alignment perspective as well. In some cases, the selected software application domain might change in different ways, such as changes in functional requirements or changes in the end user IT infrastructure environment. The case study Company ran into the latter change as the OS/2 operating system environment lost its market presence to Microsoft Windows, causing the Company to change direction in its development. The challenge for software vendors is to define an effective and optimal software application frame to include enough adaptability so that even future unknown product requirements can be implemented as these requirements are identified. It was very clear in our case study research that the selected software architectural model was important to the success of the case study company.

Another important factor to consider when using an Analytical Application Frame Strategy framework is to identify possible alignment perspective disconnects that a software vendor could face when implementing its strategies. We argued that the probability of these disconnects is high, as could be seen in our case study research, but as long as the software vendor is aware of the possible disconnects when implementing any of the three sub-strategies within the framework, the probability of future problems is lessened. "Architectural disconnect" typically leads to a weakened software application frame architecture. The reason for this could be too much emphasis either on market segmentation or on technology. When the software vendor neglects technological selections, the vendor could run into "Technological disconnect" that could lead to future market segmentation difficulties or to a fragile software application frame architecture. The last potential disconnect is "Segmentation disconnect," most likely when software vendors have a strong technological background and/or the emphasize the software application frame architecture, with the result of weak market segmentation. It is important for a software vendor to recognize the existence of these types of disconnects when utilizing an Analytical Application Frame Strategy framework.

The connection between the domain-specific architectural model (part of an analytical application frame architecture sub-strategy) and the architectural style defined in the application frame technology sub-strategy must be severed when planning the analytical application frame architecture. The connection between these two different architectural models became obvious in both Chapter 3 and Chapter 4, where we explained how selection of an underlying technology will impact the defining technology and vice versa. We also expressed the importance of defining technology for an analytical application software vendor, as these elements will be the differentiating factor when competing in the field of analytical application software. This was evidenced within our case study company: the defining technology for the Company had characteristics of an "easy implementation" and an effective communication protocol that provided quick response times with "state-of-the-art" dashboard technology. It was evident that changes in the architectural layers and/ or elements within a sub-strategy could potentially impact other sub-strategies and their elements. If the software vendor is aware of these interdependences, it can replace some of these elements without impacting the overall solution. This can be achieved only via effective interfaces and large-grained modular architecture.

The selection of an architectural model will consider a combination of factors, as we saw in our comparison in Chapters 3 and 4, factors such as the core competence of the software development team, the customer segments to be addressed, and also the IT infrastructure technologies that the customer segment is going to support. The selection of the optimal architectural model must be based on the search for and identification of "an optimal architectural construct that will become the foundation for the analytical application frame." This architectural construct has to be well aligned with technologies discussed in Chapter 3, such as data mart or data warehouse architectural models. This architectural construct – the analytical application frame – will become the foundation for derivative analytical solutions that reflect the needs and requirements of selected market segment. To enable derivative product development, the analytical application frame has to include functionality and flexibility to enable the software organization to maintain and build multiple analytical solutions for different market segments without having to modify the analytical application frame itself.

The adaptability of an analytical application frame can be measured by how easily the solution can be used for derivative solution development. We discussed in Chapter 5 different variation techniques that can be used within analytical application derivative software solution development. The most typical variation technique is solution variation, where the solution vendor includes predefined business metrics for a selected market segment using the meta data and business analytics repository as its foundation. This type of solution variation reflects closely the description of solution variation of information products discussed by Meyer and Lopez (1996). We also expressed the importance of being able to maintain the solutions that are delivered to end user organizations and the ability to upgrade these solutions with new releases.

Another type of variation is the use of an extension component layer within the Analytical Application Frame Strategy framework to customize the analytical application frame for different market segments using software engineering related variation mechanisms. Analytical application frame development will not allow any customer-specific customizations, as the analytical application frame loses its concept of being the core for derivative solution development. According to our literature search into analytical application solutions, the expectation of end user organizations is to have 80% of the common business metrics and business processes covered in the solution. These organizations expect to be able to redefine the remaining 20% of their own specific requirements. It is important that the analytical application frame support the maintenance and parallel development of the solutions so that the software vendor can deliver a new release of the vertical solution without breaking the existing solution that the end user organization has changed and maintained.

Evidence of the use of analytical application frames in analytical application solution development within the case study company could be found specifically within the client frame development effort, where new derivative products could be created. The question still remains whether the case study company could have achieved greater leverage by more active use of the Analytical Application Frame Strategy framework. According to the quantitative analysis, the Company had not utilized defining technology components to implement different solutions for different vertical market segments. The variation within the case study company has been mostly technical variation, using the client frame as a basis for derivative products and variations, by using the price/performance matrix to enable the solution to work with larger multi-national organizations. According to the interviews with development management, the new future software application frame is more compatible with analytical application solution development, having the ability to maintain and update new solution releases in parallel with end user organization development of solutions. This flexibility will include better functionality in maintaining solutions, avoidance of conflicts in parallel development, and better installation routines when delivering the solution.

Finally, to achieve an integrated/balanced strategy, software vendors must continuously re-align all three sub-strategies within an Analytical Application Frame Strategy framework when using an analytical application frame for derivative software solution development. A deviation from this realignment could cause any of the three sub-strategies to get out of sync, as was seen in the case study analysis. Careless selection of underlying technology could lead to an inflexible analytical application frame architecture, and this could prevent the software vendor from extending its solutions to new market segments in the future. A weak underlying analytical application frame architecture could in the future prevent the software vendor from selecting new underlying technologies that are needed in a new market segment that the software vendor would like to address. Therefore, the selected analytical application frame technologies must match well with the analytical application frame architecture. This type of alignment is important when building solutions for selected market segments.

8.4 Changes in Effectiveness Criteria when Applying the Product Platform Approach

The final research question, "How can effectiveness criteria change when applying the product platform approach in analytical application software development?" reflects back to the redefined Analytical Application Frame Strategy framework. According to this framework, part of the product development strategy is to integrate the included sub-strategies in the planning process when building analytical software applications. The traditional approach to analytical application solution development is to focus on one sub-strategy at a time, which could have consequences as identified in the case study analysis. Another clear difference from the traditional product platform approach is the dependence on a software development process, which must be considered when planning the use of an Analytical Application Frame Strategy framework. Selection of underlying technologies will impact not only the execution and development environment but also the analytical application frame architecture sub-strategy and possibly future market segmentation strategies.

Several existing software related product platform references (Meyer and Seliger, 1998; Sääksjärvi, 1998, 2002) conclude that component-based software engineering could be a good foundation for developing software related product platforms. The work of Meyer and Seliger (1998) generalized software development to development of physical products, while Sääksjärvi (2002) argued that software development is clearly different from physical product platform development. This argument led to implementation of the Software Application Frame Strategy framework. Software organizations will not have much use for existing traditional product platform literature, as this literature does not reflect the complexities of software development.

An Analytical Application Frame Strategy framework combines with practical means of creating common software assets by using, for example, software archetypes (Bosch, 2000). Definition of recurring patterns across products in a product line will give the power of true derivative software development to any software vendor. It is clear based on our analysis of analytical application software development that each software product domain will be different with respect to its software frame technology and software frame architecture sub-strategy. Based on this, we can conclude that software organizations must put different emphasis on their product development compared with traditional product development. The overall process for executive management is the same – product market segmentation will be aligned with corporate business strategy - and it is the responsibility of the software product development organization to select appropriate technologies that enable the product development organization to create software assets that are used across different product derivatives. This variation can be implemented using adaptable frame architecture and techniques provided in the software product line and software reuse literature.

Using our Analytical Application Frame Strategy framework as a basis when analyzing the case study company, it became evident that the technology selections in the early 1990s impacted not only market segmentation but also the analytical application frame architecture. The case study also showed how technological selections can hinder other elements within the analytical application frame technology sub-strategy, such as the execution and integrated development environment, with an impact on the overall development of the software solution. Software development using an Analytical Application Frame Strategy requires not only technological selections but also clever identification of recurring functional patterns to enable definition of an adaptable software application frame that can be used as basis for derivative product development. Existing literature on data warehousing, which we argue is the foundation for analytical software applications, does not include any analysis of the impacts of software engineering on related topics, such as common software assets or variation of these common software assets to enable creation of new software products.

A comparison of analytical application software development with or without the use of a product platform approach for a software vendor can be analyzed from two different perspectives. The first analysis perspective is to identify whether the software vendor has used defining technology as the foundation for vertical or horizontal market segmentation. The second analysis perspective is to determine how effective the derivative product development has been, using either technical or solution variation. Traditional analytical application software development does not necessarily include features/functions that make market segmentation easy to deploy and solution development and maintenance of the solution possible in parallel with end user development of the same solution. The researcher notes, with close to 15 years of experience, that software development is still very technology driven, and that customer experience or customer requirements from a solution development perspective are typically completely ignored or receive little attention. The case study company can be categorized as a technology or market-driven software vendor: solution development has been deemphasized, while technology development has been the focus for the last few years.

An analysis of analytical application frame evolution over the years shows clearly that not until early 2000 did the Company achieve relatively successful derivative product development using the client frame. Most of the software application frame development (both server and client) has been more or less technology development, where the server frame has been the “heart” of the overall solution. The Company has been able to extend the server frame with three extensions during its lifetime, leading to more effective price/performance market segmentation. Our interviews with the management and founders of the Company found relatively successful market focus. The company has achieved good penetration in a few selected market segments, such as distribution, car dealerships, heavy equipment rentals, etc., but this penetration has not included any specific features to support these vertical market segments. Maintenance and support for these domains in concert with end user organizations is not implemented.

If the Company aims to be successful with both technical and solution variation in its future analytical application frame development, it needs to increase the flexibility of its server and client frame to include the ability to maintain and build analytical solutions in parallel with end user organizations. Based on the findings in the longitudinal case-study research, it is recommended that solution development be kept separate from the technology development of the software. Therefore, it is recommended that the Company have different personnel for these two different tasks. Software development organizations are traditionally very technology driven and software engineers are typically not skilled or well versed in different market segments. This weakness must be attacked by skilled vertical market specialists who plan and build the analytical solutions using the technology implemented in the technology development department.

Finally, the case study analysis showed the importance of an analytical application frame and the selection of underlying and defining technologies in derivative software product development. The case study also showed that, if the executive management does not recognize the importance of the fit between the analytical application frame architecture and the selected analytical application frame technology sub-strategy, future market segmentation strategies might be impacted by these selections. Unfortunately, software development organizations do not have visibility of these types of frameworks (such as our Analytical Application Frame Strategy framework). This will make the involvement of executive management very ineffective and difficult as, existing methodologies, techniques, tools, and languages such as UML are too difficult for management to learn, understand, and use in their planning process.

8.5 Relevance and Generalizability of the Study

Yin (2003) discusses criticisms of the generalizability of the results of single case studies. The author concludes by stating:

> “How can you generalize from a single case study’ is a frequently heard question … The short answer is that case studies … are generalizable to theoretical propositions ….”

The question of whether our research results can be generalized and applied to other software application domains should be discussed from several different perspectives. First of all, our research concentrated on analytical application software development with respect to this software application domain. As our analysis of product platform related theory in Chapter 2 did not specifically address the applicability of product platform development to analytical applications, we can conclude that the findings in Chapter 2 can be generalized to other software domains (we specifically mention as well the lack of literature investigating the product platform development of software products and how software application frames should be defined). Other researchers can use the refined Analytical Application Frame Strategy framework and apply it to respective software application domains, building specific alignment models that are typical of the combination of software application frame technology, software application frame architecture, and software application frame leverage sub-strategy for the given domain.

Yin (2003) concludes that there are four types of generalizations in interpretative case studies: the development of concepts, the generalization of theory, the drawing of specific implications, and the contribution of rich insight. According to Yin (ibid), these four types of generalizations are not mutually exclusive. Because of this, this case study research theory generalization is based on a combination of factors, drawing specific implications and contributing rich insights. This type of analysis can be also categorized as a critical case (Yin, 2003), as this type of case study analysis could not have been performed without the specific relations that the researcher had with the software development organization.

The study of Jones and Nandhakumar (1993) used a framework to analyze and explain the applicability of a framework in building an executive information system (generalization of theory). Another study from Walsham and Waema (1994) involved specific implications in a particular domain of action (information systems for a financial services company). In this 8 year longitudinal study, the researchers analyzed the relationship between the design and development process and the business strategy. According to Walsham (1995a), Walsham and Waema (1994) use verbs such as 'can' rather than 'will', which implies that the generalizations are more like tendencies and not like predictions.

The refinement of the framework by Sääksjärvi (2002), adapted to analytical application solution development (Analytical Application Frame Strategy framework) and the use of this framework in building different alignment perspectives for analytical application solution development, provides a clear contribution to both the analytical application software domain and to software related product platform theory. This framework was tested in an interpretative single-case study to measure the effectiveness and efficiency of software application frame development within the case study company. As in the case of Walsham and Waema (1994), the implications of this framework were clear, but the differences between analytical application software vendors can be considerable, and therefore our conclusions are more of a 'can' nature and less of a 'will' nature. It is unknown whether any other software vendor would achieve the same results using an Analytical Application Frame Strategy framework in its software application frame development.

The generalizability of the Analytical Application Frame Strategy framework to analytical application solution development can be demonstrated in the case study analysis, specifically in the analysis of client frame derivative product development. It was evident that the lack of proper selection of an underlying technology has impacted current and future market segmentation strategies for the case study company. This could have been avoided with the use of an Analytical Application Frame Strategy framework. The generalizability of this framework applies only to analytical application software vendors, as every vendor must provide all the components within an analytical application frame architecture to be able to provide analytical application solutions to different market segments. The applicability of the framework within other software domains must be re-adapted to each software domain. This could be a good foundation for future research.

An example of this re-adaptation work is in Chapter 4, where the analytical application software domain with corresponding technological elements such as data warehousing architectural models, executive information systems, OLAP technologies, etc. was analyzed and an analytical application frame architecture was built specifically for analytical applications. Existing decision support/analytical application literature does not specifically address the development of analytical applications from the software vendor's perspective, and therefore we had to define an analytical application frame architectural model to be able to analyze the corresponding analytical application frame leverage and analytical application frame technology sub-strategy. The requirement that the product platform concept enable effective derivative product development caused the researcher to identify an optimal architectural construct with all the corresponding analytical application frame components (or elements) that can be used across any horizontal or vertical market segment. We argue that the concept of identifying analytical application frame components can be generalized to any software application domain if it is combined with the corresponding characteristics of the software domain itself.

We also expect that each software application domain will have slightly different criteria for software application frame technology, software application frame leverage, and software application frame architecture sub-strategy. Because of these differences, it is most likely that the six identified alignment perspectives with corresponding disconnects from Chapter 4 will also be different in different software domains, and that they must therefore be adapted to each selected software application domain. We identified the lack of practical software implementation advice with respect to software application frames in existing software related product platform literature. This study provided new insight into the existing literature in our analysis of software product lines, domain engineering, and component-based software engineering. Software application frames are built using common software assets which will become the foundation for a software application frame. We identified different variation mechanisms for derivative product development and found that technical variation mechanisms are generic to any given software application domain. Those variation mechanisms that are specific to analytical application software are not applicable to other software domains, and therefore each software domain must be analyzed separately to be able to identify the specific variation mechanism for a given software domain.

Finally, the single-case interpretative case study analysis reflects the findings of an analytical application vendor. Results cannot be applied to any other software domains as such. Any software vendor can utilize and learn from the thought process that a software vendor undertakes when building and analyzing its Software Application Frame Strategy framework with corresponding alignment perspectives and possible disconnects that could inhibit future development. These alignments describe the behavior of each sub-strategy within a Software Application Frame Strategy framework. The behavior will be driven by the characteristics of each sub-strategy. This characteristic is dependent on the software domain. We do argue that the method of analysis is the same regardless of the software domain, and therefore other software domains and software vendors can learn from the thought process. The case study analysis provided managerial perspectives that any software vendor can utilize when using a software application frame in its derivative software development.

8.6 Limitations of the Study and Future Research

First of all, this study has some exceptional characteristics. The researcher has had access to the Company for several years in different roles and responsibilities. The uniqueness of the case is due to the deep data that was provided. This type of information would be very hard to gather in organizations that are not in one way or another related to the researcher. The researcher used several people in the data collection process – specifically from the financial and accounting departments – to ensure accuracy of the data and that the researcher did not have a subjective view on historical events. We believe that the data we collected will speak for itself. As the data collection process took place over a very long time, we got more insight along the way into what measures and data elements were needed for our analysis. Some of our pre-release data was so strong that the Board of Directors of the case study company decided to take action based on the data. One example of this occurred when the Company decided to change from concurrent user pricing to named user pricing. Another finding that took us by surprise was the fact that almost all products had been profitable, even products that Development had a "gut feeling" were losing money. Even if these products were making money, the question remains whether a better return on investment would have been realized with greater emphasis on either server or client frames. All in all, several interesting results were identified. We also became convinced that the product line engineering approach is a very strong component of derivative software development in the analytical application software domain, even if the Company has yet to experience the power to its full potential. Good results have already been demonstrated via the new client frame (V2) and its derivative products, but still more has be to done to take development to the next level.

We also realized that, as our findings are based on an analytical interpretative single-case study, there are possibilities for deficiencies, specifically in our generalizations. Some of these possible deficiencies are reduced due to the researcher's practical experience in both analytical application solution development and software engineering. We also recognize that the reporting of interpretative case study results by the researcher is not about the reporting of facts, but rather about "reporting interpretations of other people's interpretations" (Walsham, 1995a). We also understand that some "double hermeneutic" (Giddens, 1984) might have occurred in our research, whereby the actions of researchers might impact the interpretations of the people who are being researched.

The case study analysis was carried out over several years, giving the researcher a specific opportunity to make observations over an extended period of time. The founding Chief Executive Officer left the Company in early 2000, enabling the researcher to gain access to all information in the company, including financial records and project related information. The researcher carried out dozens of different interviews during the years that included questions about the decisions made in the company in its early days. It is obviously possible that the interviewees could have forgotten some details either intentionally or unintentionally, but these findings were triangulated with quantitative analysis using financial and project related data and other interviews. Because we had access to all the needed information, we believe that the error has been relatively small and has not impacted the overall analysis of the case study company and its development over the years.

We also recognize that a comparison with other similar software organizations in the same domain could have provided additional insight, but due to the harsh competition, this type of analysis could not be performed. The researcher gained access to records that would not have been available in other similar studies due to several factors: salary information is sensitive, internal financial records are confidential, and a myriad of other internal documents might not even exist. Another question that could be posed is whether the case study company is somehow exceptional compared with other software organizations in the same domain, rendering the research results not generalizable. The researcher has been professionally working within the domain for the last 15 years. Six of those years were spent with another software vendor (a competitor) in the same domain. Based on this experience, it can fairly be said that these two organizations have similar products with similar functionality. The biggest difference is mostly the surrounding environment, such as the size of the company, the importance of the software offering to the organization, the core competence of the software team, etc. These types of differences will not change the foundation of the analysis, but could lead to different results when analyzing software application frame effectiveness and efficiency. Based on this analysis, the researcher has no reason to believe that either the qualitative or the quantitative findings of this study would be biased due to the characteristics of the case study company.

Our research was focused on software development of analytical application software solutions. We were able to determine general findings that can be applied to any software application domain. As for the differences explained in prior chapter, it would be of interest to compare them with another software application domain in the light of our Analytical Application Frame Strategy framework, and to compare how each of these other software application domains would differ with respect to software application frame technology, software application frame architecture, and software application frame leverage sub-strategy. Another future research topic could be to continue to research software product line engineering and how it can be linked more closely with analytical application software development. This will be a good opportunity for additional articles to be posted in journals and corresponding conferences. We strongly believe that software development has to improve and achieve better results in reuse and derivative product development, and that existing integrated development tools must be better integrated to engineering practices such as software product line engineering.

Another interesting future research effort could be to use the findings of this study as compared with other software domains to be able to see what differences our suggested alignment perspectives could demonstrate. If another research group had access to another analytical application software vendor to make a comparable study, differences might be identified. Comparison of the use of an Analytical Application Frame Strategy framework within several organizations with different business models would also be of interest. This could provide valuable information to software organizations that are investing in the development of a common software application frame, showing how service revenue could impact the overall effectiveness and efficiency of the software application frame.

Finally, our effectiveness and efficiency calculations were based on calculations rules that can be found in the existing product platform related literature as amended with a few new measures that we defined in this study. It would be of interest to define new measurements for Software Application Frame Strategy framework effectiveness and efficiency by taking existing measurements from the software product line engineering literature and use these as additional measures for evaluating the software application frame development of a software vendor.

9. REFERENCES

Abatangelo D., (2001). "Increasing the Value of Your Analysis, Layer by Layer." *Journal of Data Warehousing*, Vol. 6, No. 3, Summer 2001, pp. 25-31.

Abowd G., Allen R. and Garlan D., (1993). "Using Style to Give Meaning to Software Architecture." *Proceedings of SIGSOFT '93: Symposium on the Foundations of Software Engineering*, December 1993, ACM, New York.

Adamson C. and Venerable M., (1998). *Data Warehouse Design Solutions*, New York, NY: John Wiley & Sons, Inc.

Agosta L., (2000). *The Essential Guide to Data Warehousing*, Upper Saddle River, NJ: Prentice Hall PTR.

America P., Obbink H., van Ommering R. and van der Linden F., (2000). "CoPAM – A Component-Oriented Platform Architecting Method Family for Product Family Engineering." *Proceedings of the First Software Product Line Conference (SPLC1)*, Denver, CO, USA, August 28-31, 2000, pp. 167-180.

Arango G., (1994). "Domain Analysis Methods in Software Reusability." *In* Schäfer W, Prieto-Diaz R. and Matsumoto M., (eds.), *Software Reusability*, Ellis Horwood, New York, USA, pp. 17-49

Atkinson C., Bayer J. and Muthig D., (2000). "Component-Based Product Line Development: The KobrA Approach." *Proceedings of the First Software Product Line Conference (SPLC1)*, Denver, CO, USA, August 28-31, 2000, pp. 289-309.

Atkinson C., Bayer J., Bunse C., Kamsties E., Laitenberger O., Laqua R., Muthig D., Paech B., Wust J. and Zettel J., (2002). *Component-Based Product Line Engineering with UML*, Boston, Addison-Wesley.

Baker N. and Freeland J., (1975). "Recent Advances in R&D Benefit Measurement and Project Selection Methods." *Management Science*, Vol. 21, No. 10. pp. 1164-1175.

Bass L., Clements P., Cohen S., Northrop L. and Withey, J., (1997). *Product Line Practice Workshop Report, Technical Report (CMU/SEI-97-TR-003, ESC-TR-97-003)*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Bass L., Clements P., Cohen S., Northrop L., Smith D. and Withey J., (1998a). *Second Product Line Practice Workshop Report, Technical Report (CMU/SEI-98-TR-015, ADA 343688)*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Bass L., Clements P. and Kazman R., (1998b). *Software Architecture in Practice*, Reading, Massachusetts: Addison-Wesley.

Bass L., Clements P. and Kazman R., (2003). *Software Architecture in Practice*, Boston, Massachusetts: Addison-Wesley.

Bass L., Campbell G., Clements P., Northrop L. and Smith D., (1999). *Third Product Line Practice Workshop Report (CMU/SEI-99-TR-003, ADA 361391)*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Bass L., (2001). “Software Architecture Design Principles.” *In* Heineman G.T. and Councill W.T. (eds.), *Component-based Software Engineering: Putting the Pieces Together*, Upper Saddle River, NJ: Addison-Wesley, pp. 389-403.

Batory D., Cardone R. and Smaragdakis Y., (2000). “Object-Oriented Frameworks and Product-Lines.” *Proceedings of the First Software Product Line Conference (SPLC1)*, Denver, CO, USA, August 28-31, 2000, pp. 227- 247.

Batory D., Johnson C., MacDonald B. and von Heeder D., (2002). “Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study.” *ACM Transactions on Software Engineering and Methodology*, Vol. 11, No. 1, April 2002, pp. 191-214.

Baumöl U., Jung R. and Winter R., (2000). “Adapting the Data Warehouse Concept for the Management of Decentralized Corporations.” *Journal of Data Warehousing*, Vol. 5, No. 1, Winter 2000, pp. 35-43.

Benbasat I., Goldstein D.K. and Mead M., (1987). “The Case Research Strategy in Studies of Information Systems.” *MIS Quarterly* , Vol.11, No. 3, September 1987, pp.369-386.

Bengtsson P., Lassing N., Bosch J. and van Vliet H., (2000). “Analyzing Software Architectures for Modifiability.” *Technical Report (HK-R-RES00/11-SE)*, University of Karlskrona/Ronneby, Ronneby.

Bosch J., (1999a). “Product-Line Architectures in Industry: A Case Study.” *Proceedings of the 21st International Conference on Software Engineering (ICSE’99)*, May 16-22, 1999, pp. 544-554.

Bosch J., (1999b). “Evolution and Composition of Reusable Assets in Product-Line Architectures: A Case Study.” *Proceedings of the First Working IFIP Conference on Software Architecture (WICSA1)*, San Antonio, Texas, USA, February 22-24, 1999, pp. 321-340.

Bosch J., (2000a). *Design & Use of Software Architectures – Adopting and Evolving a Product-line Approach*, Boston: Addison-Wesley.

Bosch J., (2001). “Software Product Lines: Organizational Alternatives.” *23rd International Conference on Software Engineering (OCSE’01)*, Toronto, Canada, May 12-19, 2001, pp 91-102.

Bosch J. and Bengtsson P., (2001). “Assessing Optimal Software Architecture Maintainability.” *Proceedings of the Fifth European Conference on Software Maintainability and Reengineering*, Lisbon, Portugal, March 14-16, 2001, pp. 168-175.

Bosch J., (2002). “Maturity and Evolution in Software Product Lines: Approaches, Artifacts and Organization.” *Proceedings of the Second International Conference of Software Product Line Conference (SPLC2)*, San Diego, CA, USA, August 19-22, 2002, pp. 257-271.

Bosch J. and Högström M., (2000). “Product Instantiation in Software Product Lines: A Case Study.” *Second International Symposium on Generative and Component-Based Software Engineering (GCSE 2000)*, Erfurt, Germany, October 9-12, 2000, pp. 147-162.

Bragg S.M., (2002). *Business Ratios and Formulas – A Comprehensive Guide*, Hoboken, NJ: John Wiley & Sons, Inc.

Brereton P., Budgen D., Bennet K., Munro M., Layzell P., Macaulay L., Griffiths D. and Stannett C., (1999). "The Future of Software." *Communications of the ACM*, Vol.42, No.12, December 1999, pp. 78-84.

Briand L., Arisholm E., Counsell S., Houdek F. and Thevenod-Fosse P., (1999). "Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions." *Empirical Software Engineering*, Vol. 4 No. 4, December 1999, pp. 387-404.

Brooks F.P., (1995). *The Mythical Man-Month – Essays on Software Engineering*, Anniversary Edition, Reading, Massachusetts: Addison-Wesley.

Brooks F.P., (1987). "No Silver Bullet: Essence and Accidents of Software Engineering", *IEEE Computer*, Vol. 20, No. 4, April 1987, pp. 10-19.

Brown A.W. and Wallnau K.C., (1996). "Engineering of Component-Based Systems." *In* Brown A.W. (eds.), *Component-based Software Engineering – Selected Papers from the Software Engineering Institute*, Software Engineering Institute, IEEE Computer Society, Los Alamitos, California, pp. 7-15.

Brown A.W., (2000). *Large-Scale Component-Based Development*, Object and Component Technology Series, Upper Saddle River, NJ: Prentice Hall PTR.

Brown J.T., Spence I., Kilpatrick P. and Crookes D., (2002). "Adaptable Components for Software Product Line Engineering." *Proceedings of the Second International Conference of Software Product Line Conference (SPLC2)*, San Diego, CA, USA, August 19-22, 2002, pp. 154-175.

Brown, M.G. and Svenson R.A., (1975). "Measuring R&D Productivity." *Research Technology Management*, July-August 1975, pp. 11-15.

Broy M., Deimel A., Henn J., Koskimies K., Plasil F., Pomberger G., Pree W., Stal M. and Szyperski C., (1998). "What Characterizes a (Software) Component?" *Software Concepts & Tools*, Vol. 19, No. 1, Springer-Verlag, pp. 49-56.

Burwen M., (2000). "ERP, BI and E-Commerce: Where are the Winners?" *DM Review*, Vol. 10, No. 7, July 2000, p. 34.

Card D.N. (1995). "The RAD Fad - Is Timing Really Everything?" *IEEE Software*, Vol. 12, No. 5, September 1995, pp. 19-23.

Carmel E., (1995). "Cycle Time in Packaged Software Firms." *The Journal of Product Innovation Management*, Vol. 12, No. 2, March 1995, pp. 110-123.

Chaudhuri S. and Dayal U., (1997). "An Overview of Data Warehouse and OLAP Technology." *ACM SIGMOD Record*, Vol. 26, No. 1, March 1997, pp. 65-74.

Cheesman J. and Daniels J., (2001). *UML Components – A Simple Process for Specifying Component-Based Software*, Boston, MA: Addison-Wesley.

Chilanti M. et al., (1997). *Data Warehousing Solutions on the AS/400*, IBM Red Book SG24-4872-00.

Christensen C.M., (1997). *The Innovator's Dilemma – When New Technologies Cause Great Firms to Fail*, Boston, MA: Harvard Business School Press.

Clements P., (1996). "From Subroutines to Subsystems: Component-Based Software Development." *in Brown A.W. (eds.) Component-based Software Engineering – Selected Papers from the Software Engineering Institute*, Software Engineering Institute, IEEE Computer Society, Los Alamitos, California, pp. 3-6.

Clements P. and Northrop, L., (1999). *A Framework for Software Product Line Practice, Version 2.0*, Pittsburg, PA, Software Engineering Institute, Carnegie Mellon University, July 1999.

Clements P. and Northrop L., (2002). *Software Product Lines – Practices and Patterns*, SEI Series in Software Engineering, Boston, MA: Addison-Wesley.

Clements P., Kazman R. and Klein M., (2002). *Evaluating Software Architectures – Methods and Case Studies*, SEI Series in Software Engineering, Boston, MA: Addison-Wesley.

Cordero R., (1990). "The Measurement of Innovation Performance in the Firm: An Overview." *Research Policy*, Vol. 19, No. 4, pp. 185-192.

Cowan R.D., Mili A., Ammar H., McKendall A. Jr., Yang L., Chen D. and Spencer T., (2002). "Software Engineering Technology Watch." *IEEE Software*, Vol. 19, No. 4, July/August 2002, pp. 123-129.

Cranford S., (1998). "Financial Data Marts." *DM Review*, Vol. 8, No. 2, February 1998, p. 28.

Cusumano M.A. and Selby R.W., (1995). *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*, New York, NY: The Free Press.

Cusumano M.A. and Yoffie D.B., (1998). *Competing on Internet Time: Lessons from Netscape and Its Battle with Microsoft*, New York, NY: The Free Press.

Cusumano M.A. and Yoffie D.B., (1999). "What Netscape Learned from Cross-Platform Software Development." *Communications of the ACM*, Vol. 42, No. 10, October 1999, pp. 72-78.

Cusumano M.A., (2003). "Technology Strategy and Management: Finding Your Balance in the Products and Services Debate." *Communications of the ACM*, Vol. 46, No. 3, March 2003, pp. 15-17.

Cusumano M.A., (2004). *The Business of Software – What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad*, New York, NY: Free Press.

D'souza D.F. and Wills A.C., (1999). *Objects, Components, and Frameworks with UML – The Catalysis Approach*, Reading, Massachusetts: Addison-Wesley.

Dahmus J.B., Gonzalez-Zugasti J.P. and Otto K.N., (2000). "Modular Product Architecture." *Proceedings of DETC '00: ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Baltimore, Maryland, September 10-13, 2000, pp. 1-11.

Debevoise N. T., (1999). *The Data Warehouse Method*, Upper Saddle River, NJ: Prentice Hall PTR.

Dickel D., Kane D., Ornburn S., Loftus W. and Wilson J., (1997). "Applying Software Product-Line Architecture." *IEEE Computer*, August 1997, pp. 49-55.

Doerr B.S. and Sharp D.C., (2000). "Freeing Product Line Architectures from Execution Dependencies." *Proceedings of the First Software Product Line Conference (SPLC1)*, Denver, CO, USA, August 28-31, 2000, pp. 313-329.

Downes L. and Mui C., (1998). *Unleashing the Killer App – Digital Strategies for Market Dominance*, Boston, Massachusetts: Harvard Business School Press.

Eckerson W.W, (1998a). "The Decision Support Sweet Spot." *Journal of Data Warehousing*, Vol. 3, No. 2, Summer 1998, pp. 2-7.

Eckerson W.W., (1998b). "Post-Chasm Warehousing." *Journal of Data Warehousing*, Vol. 3, No. 3, Fall 1998, pp. 38-45.

Eckerson W.W., (2002). *The Rise of Analytic Applications: Build or Buy?* The Data Warehousing Institute (TDWI) Report Series.

Eisenhardt K.M., (1989). "Building Theories from Case Study Research." *Academy of Management Review*, Vol. 14, No. 4, pp. 532-550.

Fayad M., Schmidt D. and Johnson R., (1999). *Building Application Frameworks – Object-Oriented Foundations of Framework Design*, New York, NY: John Wiley & Sons..

Feitzinger E. and Lee H.L., (1997). "Mass Customization at Hewlett-Packard: The Power of Postponement." *Harvard Business Review*, Vol. 75, January-February 1997, pp. 116-121.

Fichman R.G. and Kemerer C.F., (1993). "Adoption of Software Engineering Process Innovations: The Case of Object Orientation." *Sloan Management Review*, Vol, 34, No. 2, Winter 1993, pp. 7-22.

Gadamer H-G., (1976). "The Historicity of Understanding." *In* Connerton P. (eds.), *Critical Sociology - Selected Readings*, Penguin Books, Harmondsworth, UK.

Garlan D., Allen R. and Ockerbloom J., (1995). "Architectural Mismatch: Why Reuse Is So Hard." *IEEE Software*, Vol. 12, No. 6, November 1995, pp. 17-26.

Gawer A., (2000). *The Organization of Platform Leadership: An empirical investigation of Intel's management processes aimed at fostering complementary innovation by third parties*, Ph.D. Thesis, Massachusetts Institute of Technology.

Gawer A. and Cusumano M.A., (2002). *Platform Leadership – How Intel, Microsoft, and Cisco Drive Industry Innovation*, Boston, Massachusetts: Harvard Business School Press.

Geyer L. and Becker M., (2002). "On the Influence of Variabilities on the Application Engineering Process of a Product Family." *Proceedings of the Second International Conference of Software Product Line Conference (SPLC2)*, San Diego, CA, USA, August 19-22, 2002, pp. 1-14.

Giddens A., (1984). *The Constitution of Society: Outline of the theory of structuration*, Berkley, California: University of California Press.

Gill H.S., and Rao P.C., (1996). *The Official Guide to Data Warehousing*, Indianapolis, IN: QUE Corporation.

Gilmore J.H. and Pine II J.B., (1997). "The Four Faces of Mass Customization." *Harvard Business Review*, Vol. 75, January-February 1997, pp. 91-101.

Giovinazzo W.A., (2000). *Object-Oriented Data Warehouse Design – Building a Star Schema*, Upper Saddle River, NJ: Prentice Hall PTR.

Gleason D., (1998). "Decision Support Systems: To Buy or To Build." *Journal of Data Warehousing*, Vol. 3 No. 3, Fall 1998, pp. 12-22.

Gonzales-Zugasti J. and Otto K., (2000). "A Method for Architecting Product Platforms with an Application to Interplanetary Mission Design." *Research in Engineering Design*, Volume 12, pp. 61-72.

Gray P. and Watson H.J., (1997). "New Developments in Data Warehousing." *Journal of Data Warehousing*, Vol. 2 No. 2, April 1997, pp. 2-4.

Gray P. and Watson H.J., (1998). *Decision Support in the Data Warehouse*, Upper Saddle River, NJ: Prentice Hall PTR.

Gray P., (1999). "What's New in Data Warehousing." *Journal of Data Warehousing*, Vol. 4, No. 2, Summer 1999, pp. 12-14.

Griffin J., (1998). "Data Mart vs. Data Warehouse." *DM Review*, Vol. 8, No. 2, February 1998, p. 10.

Griss M.L., (1996). "Domain Engineering And Variability In The Reuse-Driven Software Engineering Business." *Object Magazine*, Vol. 6, No. 10, December 1996.

Griss M., Favaro J. and d'Alessandro M., (1998). "Integrating Feature Modeling with the RSEB." *Proceeding of the 5th International Conference on Software Reuse (ICSR)*, IEEE Computer Society Press, Los Alamitos, California, pp. 76-85.

Griss M.L., (2000a). "Implementing Product-Line Features with Component Reuse." *Proceedings of the 6th International Conference on Software Reuse (ICSR-6)*, Vienna, Austria, June 27-29, 2000, pp. 137-152.

Griss M.L., (2000b). "Implementing Product-Line Features by Composing Aspects." *Proceedings of the First Software Product Line Conference (SPLC1)*, Denver, CO, USA, August 28-31, 2000, pp. 271-288.

Griss M.L., (2001). Product-Line Architectures, *In* Heineman G.T. and Councill W.T. (eds.), *Component-based Software Engineering: Putting the Pieces Together*, Upper Saddle River, NJ: Addison-Wesley, pp. 405-419.

Hackney D., (1998). "Who Are You? Part 1." *DM Review*, Vol. 8, No. 2, February 1998, p.30-31

Hackney D., (2000a). "The Federated Future." *DM Review*, Vol. 10, No. 1, January 2000, pp. 34, 78.

Hackney D., (2000b). "Federated FAQs." *DM Review*, Vol. 10, No. 4, April 2000, p. 52.

Hahnke J., (1997). "Business Function 1st." *Application Development Trends*, May 1997.

Haisten M., (1996). "A History of Access and Analysis Tools." *Journal of Data Warehousing*, Vol. 1 No. 1, July 1996, pp. 46-60.

Hasselbring W., (2000), "Information System Integration." *Communications of the ACM*, Vol. 43, No. 6, June 2000, pp. 33-38.

Henderson J.C. and Venkatraman N., (1993). "Strategic Alignment: Leveraging Information Technology for Transforming Organizations." *IBM Systems Journal*, Vol. 32, No. 1, pp. 4-16.

Henderson R.M. and Clark K.B., (1990). "Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms." *Administrative Science Quarterly*, Vol. 35, pp. 9-30.

Henderson-Sellers B. and Edwards J.M., (1990). "The Object-Oriented Systems Life Cycle." *Communications of the ACM*, Vol. 33, No. 9, pp. 143-159.

Hero S., (2001). "Dependable Information without Complexity." *Journal of Data Warehousing*, Vol. 6, No. 4, Fall 2001, pp. 9-14.

Herzum P. and Sims O., (2000). *Business Component Factory – A Comprehensive Overview of Component-Based Development for the Enterprise*, New York, NY: John Wiley & Sons.

High R.H., (1998). "Component Model for Managed Objects in Large-Scale Distributed Systems." *in Thomas J. (eds.), Component-Based Software Engineering*, Managing Object Technology Series, Cambridge, United Kingdom: Cambridge University Press.

Hoch D.J., Roeding C.R., Purkert G. and Lindner S., (2000). *Secrets of Software Success – Management Insight from 100 Software Firms around the World*, Boston, Massachusetts: Harvard Business School Press.

Hofman D.J. and Rockart J.F., (1994). "Application Templates: Faster, Better, and Cheaper Systems." *Sloan Management Review*, Vol. 36, No. 1, Fall 1994, pp. 49-60.

Hopkins J., (2000). "Component Primer." *Communications of the ACM*, Vol. 43, No. 10, October 2000, pp. 27-30.

Humphries M., Hawkins M.W. and Dy M.C., (1999). *Data Warehousing – Architecture and Implementation*, Upper Saddle River, NJ: Prentice Hall PTR.

Iansiti M. and MacCormack A., (1997). "Developing Products on Internet Time." *Harvard Business Review*, Vol. September-October 1997, pp. 1-10.

Iansiti M., (1998). *Technology Integration*, Boston, Massachusetts: Harvard Business School Press.

IDC, (1997). *Packaging the Data Mart: The Application Centered Data Warehouse Emerges*, Number 13493, May 1997.

Imhoff C., (1999). "Intelligent Solutions: Will the Real Data Mart Please Stand Up?" *DM Review*, March 1999.

Imhoff C., (2000). "Intelligent Solutions: If the Star Fits – Part 1" *DM Review*, Vol. 10, No. 4, September 2000, p. 28, 64.

Inmon W.H., (1992). *Building the Data Warehouse*, Wellesley, MA: QED Publishing Group.

Inmon W.H., (1993). *Developing Client/Server Applications*, Wellesley, MA: QED Publishing Group.

Inmon W.H., (1996). *Building the Data Warehouse*, New York, NY: John Wiley & Sons, Inc.

Inmon W.H., (1997). *Building the Data Warehouse*, Second Edition, New York, NY: John Wiley & Sons, Inc.

Inmon W.H., (1998). "Bottom-Up Warehouse Development." *DM Review*, Vol. 8, No. 2, February 1998, pp. 20, 41, 69.

Inmon W. H., (1999). *Building the Operational Data Store*, Second Edition, New York, NY: John Wiley & Sons, Inc.

Inmon W.H., (2002). *Building the Data Warehouse*, Third Edition, New York, NY: John Wiley & Sons, Inc.

Inmon W.H., Imhoff C. and Battas G., (1996). *Building the Operational Data Store*, New York, NY: John Wiley & Sons, Inc.

Inmon W.H., Imhoff C. and Sousa R., (2001). *Corporate Information Factory*, Second Edition, New York, NY: John Wiley & Sons, Inc.

Jaaksi A., (2002). "Developing Mobile Browsers in a Product Line." *IEEE Software*, Vol. 19, No. 4, July/August 2002, pp. 73-80.

Jaaksi A., Aalto J-M, Aalto A., Vättö K., (1999). *Tried & True Object Development – Industry-Proven Approaches with UML*, Cambridge, United Kingdom: Cambridge University Press.

Jacobsen I, Griss M., Jonsson P., (1997). *Software Reuse – Architecture, Process and Organization for Business Success*, Reading, Massachusetts: Addison Wesley.

Jacobsen I., Booch G. and Rumbaugh, J., (1999). *The Unified Software Development Process*, Reading, Massachusetts: Addison-Wesley.

Jaring M. and Bosch J., (2002). Representing Variability in Software Product Lines: A Case Study, *Proceedings of the Second International Conference of Software Product Line Conference (SPLC2)*, San Diego, CA, USA, August 19-22, 2002, pp. 15-36.

Jarke M., Lenzerini M., Vassiliou Y. and Vassiliadis P., (2000). *Fundamentals of Data Warehouses*, New York, NY: Springer-Verlag.

Jick T.D., (1979). "Mixing Qualitative and Quantitative Methods: Triangulation in Action." *Administrative Science Quarterly*, Vol. 24, No. 4, December 1979, pp. 602-611.

Johnson R. and Foote B., (1988). "Designing Reusable Classes." *Journal of Object-Oriented Programming*, Volume 1, No.2, June/July 1988, pp. 22-35.

Johnson R.A., (2000). "The Ups and Downs of Object-Oriented Systems Development." *Communications of the ACM*, Vol.43, No. 10, October 2000, pp. 68-73.

Jones M. and Nandhakumar, J., (1993). "Structured Development? A Structurational Analysis of the Development of an Executive Information System." *In* Avison, D., Kendall, J.E. and DeGross, J. I. (eds.), *Human, Organizational, and Social Dimensions of Information Systems Development*, North-Holland, Amsterdam, pp. 475-496.

Kang K.C., Cohen S.G., Hess J.A., Novak W.E. and Peterson A.S., (1990). *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, SEI Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.

Kang K.C. et al., (1998). "FORM: A Feature-Oriented Reuse Method with Domain-Specific Architectures." *Annals of Software Engineering*, Vol. 5, No. 1, pp. 143-168.

Kang K.C., Donohoe P., Koh E., Lee J. and Kwanwoo L., (2002). "Using a Marketing and Product Plan as a Key Driver for Product Line Asset Development." *Proceedings of the Second International Conference of Software Product Line Conference (SPLC2)*, San Diego, CA, USA, August 19-22, 2002, pp. 366-382.

Kaplan B. and Duchon, D., (1988). "Combining Qualitative and Quantitative Methods in Information Systems Research: A Case Study." *MIS Quarterly*, Vol. 12, No. 4, December 1988, pp. 571-587.

Kaplan B., and Maxwell, J.A., (1994). "Qualitative Research Methods for Evaluating Computer Information Systems." *In* Anderson J.G, Aydin C.E., and Jay S.J. (eds.), *Evaluating Health Care Information Systems: Methods and Applications*, Sage, Thousand Oaks, CA, pp. 45-68.

Kaplan R.S. and Norton D.P., (1996). *Translating Strategy into Action: The Balanced Scorecard*, Boston, Massachusetts: Harvard Business School Press.

Kaplan R.S. and Cooper R., (1998). *Cost & Control: Using Integrated Cost Systems to Drive Profitability and Performance*, Boston, Massachusetts: Harvard Business School Press.

Karlsson E-A., (1995). *Software Reuse – A Holistic Approach*, Chichester, England, John Wiley & Sons Ltd.

Katzman R., Bass, L., Abowd, G. and Webb, M., (1994). "SAAM: A Method for Analyzing the Properties of Software Architectures." *Proceedings of the 16th International Conference on Software Engineering,* Sorrento, Italy, May 1994, pp. 81-90.

Kelly B.W., (1997). *AS/400 Data Warehousing – The Complete Guide to Implementation*, Second Edition, Carlsbad, CA: Midrange Computing.

Kelly S., (1996). *Data Warehousing - The Route to Mass Customization*, Chichester, West Sussex, England: John Wiley & Sons Ltd.

Kimball R., (1996). The Data Warehouse Toolkit, New York, NY: John Wiley & Sons, Inc.

Kimball R., Reeves L., Ross M. and Thornthwaite W., (1998). *The Data Warehouse Lifecycle Toolkit*, New York, NY: John Wiley & Sons, Inc.

Kimball R. and Ross M., (2002). *The Data Warehouse Toolkit*, Second Edition, New York, NY: John Wiley and Sons, Inc.

Krasner G.E. and Pope S.T., (1988). "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80." *Journal of Object-Oriented Programming*, Vol. 1, No. 3, August/September 1988, pp. 26-49.

Kruchten P., (1995). "The 4+1 View Model of Software Architecture." *IEEE Software*, Vol. 12, No. 6, November 1995, pp. 42-50.

Kruchten P., (1999). *The Rational Unified Process – An Introduction*, Reading, Massachusetts: Addison-Wesley.

Larsen G., (1999). "Designing Component-Based Frameworks Using Patterns in the UML." *Communications of the ACM*, Vol. 42, No. 10, October 1999, pp. 38-45.

Latchem S., (2001). "Component Infrastructures: Placing Software Components in Context." *In* Heineman G.T. and Councill W.T. (eds.), *Component-based Software Engineering: Putting the Pieces Together*, Upper Saddle River, NJ: Addison-Wesley, pp. 263-283.

Lopez-Herrejon R.E. and Batory D., (2001). "A Standard Problem for Evaluating Product-Line Methodologies." *Third International Conference on Generative and Component-based Software Engineering (GCSE 2001)*, Erfurt, Germany, September 9-13, 2001, pp. 10-24.

Macala R.R, Stuckey L.D. and Gross D.C., (1996). "Managing Domain-specific Product-Line Development." *IEEE Software*, Vol. 13, No. 3, May 1996, pp. 57-67.

Malveau R. and Mowbray T. J., (2004). *Software Architect Bootcamp*, Second Edition, Upper Saddle River, NJ: Prentice Hall PTR.

Martin M.V. and Ishii K., (1997). "Design for Variety: Development of Complexity Indices and Design Charts." *Proceedings of DETC 97: ASME Design Engineering Technical Conferences – Design for Manufacturability*, Sacramento, CA, USA, September 14-17, 1997, pp. 1-9.

McGrath M.E., (1995). *Product Strategy for High-Technology Companies: Hot to Achieve Growth, Competitive Advantage, and Increased Profits*, Burr Ridge, Illinois: Irwin Professional Publishing.

McGrath M.E., (2001). *Product Strategy for High-Technology Companies: Hot to Achieve Growth, Competitive Advantage, and Increased Profits*, Second Edition, New York, NY: McGraw-Hill.

McGregor J.D., Northrop L.M. and Jarrad S., (2002). "Initiating Software Product Lines." *IEEE Software*, Vol. 19, No. 4, July/August 2002, pp. 24-27.

McGuff F. and Kador J., (1999). *Developing Analytical Database Applications*, Upper Saddle River, NJ: Prentice Hall PTR.

Messerschmitt D.G. and Szyperski C., (2003). *Software Ecosystem – Understanding and Indispensable Technology and Industry*, Cambridge, Massachusetts: The MIT Press.

Meyer M.H. and Utterback J. M., (1993). "The Product Family and the Dynamics of Core Capability." *Sloan Management Review*, Vol. 34, No. 3, Spring 1993, pp. 29-47.

Meyer M.H. and Lopez L., (1995). "Technology Strategy in a Software Products Company." *The Journal of Product Innovation Management*, Vol. 12, No. 4, September 1995, pp. 294-306.

Meyer M.H. and Zack M.H., (1996). "The Design and Development of Information Products." *Sloan Management Review*, Vol. 37, No. 3, Spring 1996, pp. 43-59.

Meyer M.H. and Lehnerd A.P., (1997). *The Power of Product Platforms – Building Value and Cost Leadership*, New York, NY: The Free Press.

Meyer M.H. and Seliger R., (1998). "Product Platforms in Software Development." *Sloan Management Review*, Vol. 40, No. 1, Fall 1998, pp. 61-74.

Meyer M.H., Terzakian P. and Utterbach J.M., (1997). "Metrics for Managing Research and Development in the Context of the Product Family." *Management Science*, Vol. 43, No. 1, January 1997, pp. 88-111.

Mili H., Mili A., Yacoub S. and Addy E., (2002). *Reuse-Based Software Engineering – Techniques, Organization, and Controls*, New York, NY: John Wiley & Sons, Inc.

Miller A. and Ebert C., (2002). "Software Engineering as a Business." *IEEE Software*, Vol. 19, No. 6, November/December 2002, pp. 18-20.

Moeller R.A, (2001). *Distributed Data Warehousing using Web Technology – How to Build a More Cost-Effective and Flexible Warehouse*, New York, NY: AMACOM.

Monroe R.T., Kompanek A., Melton R. and Garlan D., (1997). "Architectural Styles, Design Patterns, and Objects." *IEEE Software*, Vol. 14, No. 1, January/February 1997, pp. 43-52.

Moore G.A., (1991). *Crossing the Chasm*, New York, NY: HarperBusiness.

Morisawa Y., (2000). "A Computing Model of Product Lines for Distributed Processing Systems, its Product Sets, and its Applications." *Proceedings of the First Software Product Line Conference (SPLC1)*, Denver, CO, USA, August 28-31, 2000, pp. 371-394.

Morris C.R. and Ferguson C.H., (1993). "How Architecture Wins Technology Wars." *Harvard Business Review*, March-April 1993, pp. 86-96.

Morris H., (1998). "The Changing Structure of Information Access Markets: Analytic Applications and Market Forecast." *Journal of Data Warehousing*, Vol. 3, No. 3, Fall 1998, pp. 23-28.

Moss L. and Adelman S., (2000). "Data Warehouse Methodology." *Journal of Data Warehousing*, Vol. 5, No. 4, Fall 2000, pp. 23-31.

Myers M.D. and Avison D., (2002). "An Introduction to Qualitative Research in Information Systems." *In* Myers M.D and Avison D. (eds.), *Qualitative Research in Information Systems – A Reader*, Thousand Oaks, CA: Sage Publications, Inc..

Nandhakumar J., (1993). *The Practice of Executive Information Systems Development: An In-Depth Case Study*, University of Cambridge, Ph.D. Thesis, Cambridge.

Niemelä E., (1999). *A Component Framework of a Distributed Control Systems Family*, Technical Research Centre of Finland, Ph.D. Thesis, VTT Publications 402, Espoo.

Noaman A.Y. and Barker K., (1997). "Distributed Data Warehouse Architectures." *Journal of Data Warehousing*, Vol. 2, No. 2, April 1997, pp. 37-50.

Northrop L.M., (2002). "SEI's Software Product Line Tenets." *IEEE Software*, Vol. 19, No. 4, July/August 2002, pp. 32-40.

O'Grady P., (1999). *The Age of Modularity – Using the New World of Modular Products to Revolutionize Your Corporation*, Iowa City, Iowa: Adams and Steele Publishers.

Orfali R., Harkey D. and Edwards J., (1999). *The Essential Client/Server Survival Guide*, Third Edition, New York, NY: John Wiley & Sons, Inc.

Orlikowski W.J. and Baroudi J.J., (1991). "Studying Information Technology in Organizations: Research Approaches and Assumptions." *Information Systems Research*, Vol. 2, No. 1, March 1991, pp. 1-28.

Paller A., (1996). "Six Trends You should Be Aware of: Data Warehousing Today." *Journal of Data Warehousing*, Vol. 1, No. 1, July 1996, pp. 41-45.

Pancake C., (1995). "The Promise and the Cost of Object Technology: A Five-Year Forecast." *Communications of ACM*, Vol. 38, No. 10, October 1995, pp. 33-49.

Perry D. and Wolf A., (1992). "Foundations for the Study of Software Architecture." ACM *SIGSOFT Software Engineering Notes*, Vol. 17, No. 4, October 1992, pp. 40-52.

Pfister C. and Szyperski C., (1998). "Why Objects Are Not Enough." *In* Jell T. (eds.), *Component-Based Software Engineering (CUC 96)*, Managing Object Technology Series, Cambridge, United Kingdom: Cambridge University Press, pp. 141-147.

Prahalad C.K. and Hamel, G., (1990). "The Core Competence of the Corporation." *Harvard Business Review*, May-June 1990, pp. 79-90.

Pressman R.S., (2000). *Software Engineering – A Practitioner's Approach*, European Adaptation, Fifth Edition, Maidenhead, Berkshire, England: McGraw-Hill International.

Pronk B. J., (2000). "An Interface Based Platform Approach." *Proceedings of the First Software Product Line Conference (SPLC1)*, Denver, CO, USA, August 28-31, 2000, pp. 331-351.

Qunitas P., (1991). "Software engineering policy and practice: lessons from the Alvey program." *Journal of Systems and Software*, Vol. 24, No. 1, January 1991, pp. 67-88.

Robertson D. and Ulrich K., (1998). "Planning for Product Platforms." *Sloan Management Review* Vol. 39, No. 4, Summer 1998, pp. 19-31.

Rosenberg D. and Scott K., (1999). *Use Case Driven Object Modeling with UML – A Practical Approach*, Reading, Massachusetts: Addison-Wesley.

Royce W., (1998). *Software Project Management – A Unified Framework*, Reading, Massachusetts: Addison-Wesley.

Russell T.J., (2000). "Designing an Adaptable Business Intelligence Architecture." *Journal of Data Warehousing*, Vol. 5 No. 3, Summer 2000, pp. 53-63.

Ryans A., More R., Barclay D. and Deutscher T., (2000). *Winning Market Leadership: Strategic Market Planning for Technology Driven Businesses*, Etobicoke, Ontario: John Wiley & Sons Canada, Ltd.

Sach T., (1997). "Building a Data Warehouse – the Key Technical Decisions." *Journal of Data Warehousing*, Vol. 2 No. 4, Winter 1997, pp. 13-20.

Sametinger J., (1997). *Software Engineering with Reusable Components*, New York, NY: Springer-Verlag.

Sanderson S. and Uzumeri M., (1995). "Managing Product Families: The Case of the Sony Walkman." *Research Policy*, Vol. 24, No. 5, pp. 761-782.

Schleicher D.L. and Taylor R.L., (1989). "System Overview of the Application System/400." *IBM Systems Journal*, Vol. 28, No. 3, pp. 398-413.

Schmid K. and Verlage M., (2002). "The Economic Impact of Product Line Adoption and Evolution." *IEEE Software*, Vol. 19, No. 4, July/August 2002, pp. 50-57.

Schmidt M. J., (2003). *Business Case Essentials: A Guide to Structure and Content*, Solution Matrix White Papers, Boston, MA, Solution Matrix, Ltd.

Schuff D. and Louis R.St., (2001). "Centralization vs. Decentralization of Application Software." *Communications of the ACM*, Vol. 44, No. 6, June 2001, pp. 88-94.

Scott M., (1971). *Management Decision Systems: Computer Support for Decision Making*, Boston, Massachusetts: Harvard University Press.

SEI, (2004). *How Do You Define Software Architecture?*, Pittsburg, PA, Software Engineering Institute, Carnegie Mellon University http://www.sei.cmu.edu/architecture/definitions.html

Sharp D.C., (2000). "Component-Based Product Line Development of Avionics Software." *Proceedings of the First Software Product Line Conference (SPLC1)*, Denver, CO, USA, August 28-31, 2000, pp. 353-369.

Shaw M., (1995). "Comparing Architectural Design Styles." *IEEE Software*, Vol. 12, No. 6, November 1995, pp. 27-41.

Shaw M. and Clements P., (1997). “A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems.” *Proceedings of the 21st International Computer Software and Applications Conference (COMPSAC’97)*, Washington, D.C, August 1997, pp. 6-13.

Shaw M. and Garlan D., (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River, NJ: Prentice Hall.

Shlaer S. and Mellor S., (1988). *Object-Oriented Systems Analysis - Modeling the World in Data*, Englewood Cliffs NJ: Yourdon Press.

Singh H.S., (1998). *Data Warehousing – Concepts, Technologies, Implementations, and Management*, Upper Saddle River, NJ, Prentice Hall PTR.

Singh H.S., (1999). *Interactive Data Warehousing*, Upper Saddle River, NJ: Prentice Hall PTR.

Soschin D., (2001). “Meta Data As an IT Platform – The Strategy of Meta Data in Your Organization.” *Journal of Data Warehousing*, Vol. 6, No. 4, Fall 2001, pp. 30-40.

Sparks S., Benner K. and Faris C., (1996). “Managing Object-Oriented Framework Reuse.” *In* Fayad M. and Cline M., (eds.), *Managing OO Software Development Theme Issue*, IEEE Computer, Vol. 29, No. 9, September 1996, pp. 53-61.

Sparling M., (2000). “Lessons Learned Through Six Years of Component-based Development.” *Communications of the ACM*, Vol. 43, No. 10, October 2000, pp. 47-53.

Sprague R.H. Jr., (1980). “A Framework for the Development of Decision Support Systems.” *MIS Quarterly*, Vol. 4, No. 4, December 1980, pp. 1-26.

Stafford J.A. and Wolf A.L., (2001). “Software Architecture.” *In* Heineman G.T. and Councill W.T. (eds.), *Component-based Software Engineering: Putting the Pieces Together*, Upper Saddle River, NJ: Addison-Wesley, pp. 371-387.

Stuart D., Sull W., Pruitt S., Cobb D., Waskiewicz F. and Cook T.W., (2000). “The SSEP Toolset for Product Line Development.” *Proceedings of the First Software Product Line Conference (SPLC1)*, Denver, CO, USA, August 28-31, 2000, pp.413-435.

Surgan M., (2000). “The Coming Packaged Data Warehouse.” *Journal of Data Warehousing*, Vol. 5, No. 2, Spring 2000, pp. 26-33.

Svahnberg M. and Bosch J., (1999). “Characterizing Evolution in Product-Line Architectures.” *Proceedings of the IASTED 3rd International Conference on Software Engineering and Applications*, October 1999, pp. 92-97.

Svahnberg M. and Bengtsson P., (2000). *Software Product Lines from Customer to Code*, Research Report, Department of Software Engineering and Computer Science, University of Karlskrona/Ronneby, S-37225 Ronneby Sweden, ISSN 1103-1581, ISRN HK/R-RES--00/1–SE.

Svahnberg M. and Bosch J., (2000). “Issues Concerning Variability in Software Product Lines.” *Proceedings of the 3rd International Workshop on Software Architectures for Product Families*, Las Palmas de Gran Canaria, Spain, March 15-17, 2000, pp. 146-157.

Svahnberg M., van Gurp J. and Bosch J., (2001). "On the Notion of Variability in Software Product Lines." *Proceedings of The Working IEEE/IFIP Conference on Software Architecture (WICSA 2001)*, August 2001, pp. 45-55.

Szyperski C., (1997). *Component Software – Beyond Object-Oriented Programming*, Harlow, England: Addison-Wesley.

Szyperski C., (1998). "Emerging Component Software Technologies – a Strategic Comparison." *Software Concepts & Tools*, Vol. 19, No. 1, pp. 2-10.

Szyperski C., (2003). *Component Software – Beyond Object-Oriented Programming*, Second Edition, Harlow, England: Addison-Wesley.

Sääksjärvi M.V.T. and Salonen P.I., (1998). *InfoManager Oy's Product Development and Product Frame Strategies*, Technology Report 62/98, Technology Development Centre TEKES (in Finnish).

Sääksjärvi M.V.T., (1998). *Product Frame: A New Thinking in Strategic Development of Software Products*, Technology Report 62/98, Technology Development Centre TEKES (in Finnish).

Sääksjärvi M.V.T., (2002). Software Application Platforms: From Product Architecture to Integrated Application Strategy, *Proceedings of the 26th Annual International Computer Software Applications Conference (COMPSAC'02)*, Oxford, England, August 26-29, 2002, pp. 435-443.

Tanrikorur T., (1998). "Enterprise DSS Architecture – A Hybrid Approach." *DM Review*, Vol. 8, No. 2, February 1998, pp. 62-66.

Thiel S. and Hein A., (2002). "Modeling and Using Product Line Variability in Automotive Systems." *IEEE Software*, Vol. 19, No. 4, July/August 2002, pp. 66-72.

Thiel S. and Peruzzi F., (2000). "Starting a Product Line Approach for an Envisioned Market." *Proceedings of the First Software Product Line Conference (SPLC1)*, Denver, CO, USA, August 28-31, 2000, pp. 495-512.

Thomann J. and Wells D., (1998). "Evaluating Data Warehouse Methodologies: Objectives and Criteria." *Journal of Data Warehousing*, Vol. 3, No. 4, Winter 1998, pp. 11-16.

Thomann J. and Wells D., (1999). "Evaluating Data Warehouse Methodologies: An Evaluation Process." *Journal of Data Warehousing*, Vol. 4, No. 2, Summer 1999, pp. 2-11.

Thomann J. and Wells D., (2000). "Implementing Data Warehouse Methodology: Guidelines for Success." *Journal of Data Warehousing*, Vol. 5, No. 1, Winter 2000, pp. 11-23.

Tipnis V.A. and Epifano P.J., (1998). "An Architecture for Enterprise-Wide Data Warehouses." *Journal of Data Warehousing*, Vol. 3 No. 4, Winter 1998, pp. 23-31.

Ulrich K.T. and Eppinger S.D., (1995). *Product Design and Development*, New York, NY: McGraw-Hill.

Uusi-Rauva E., (1989). *Tuotekohtaisen kustannuslaskennan kehittäminen modernissa tuotantolaitoksessa* [*Development of Product-Specific Cost Accounting in a Modern Production Plant*,] Metalliteollisuuden Keskusliitto, Tekninen Tiedotus 10/89.

van der Linden F., (2002a). "Engineering Software Architectures, Processes and Platforms for System Families – ESAPS Overview." *Proceedings of the Second International Conference of Software Product Line Conference (SPLC2)*, San Diego, CA, USA, August 19-22, 2002, pp. 383-397.

van der Linden F., (2002b). "Software Product Families in Europe: The Esaps & Café Projects." *IEEE Software*, Vol. 19, No. 4, July/August 2002, pp. 41-49.

Van Dyk W., (2002). "The Dangers of Denormalizing in Dimensional Modeling." *Journal of Data Warehousing*, Vol. 7 No. 1, Winter 2002, pp. 40-46.

van Ommering R. and Bosch J., (2002). "Widening the Scope of Software Product Lines – From Variation to Composition." *Proceedings of the Second International Conference of Software Product Line Conference (SPLC2)*, San Diego, CA, USA, August 19-22, 2002, pp. 328-347.

Vayda T.P., (2000). "Components on the Assembly Line." *The Future of Software*, Fawcette Technical Publications, 10th Anniversary Issue, Vol. 1, No. 1, Winter 2000/2001, pp. 86-87.

Wallnau K.C, Hissam S.A. and Seacord R.C., (2002). *Building Systems from Commercial Components*, Upper Saddle River, NJ: Addison-Wesley.

Walsham G., (1993). *Interpreting Information Systems in Organizations*, Chichester, England: Wiley.

Walsham G., (1995a). "Interpretive Case Studies in IS Research: Nature and Method." *European Journal of Information Systems*, Vol. 4, No. 2, pp. 74-81.

Walsham G., (1995b). "The Emergence of Interpretivism in IS Research." *Information Systems Research*, Vol. 6, No. 4, pp. 376-394.

Walsham, G. and Waena, T., (1994). "Information Systems Strategy and Implementation: A Case Study of a Building Society." *ACM Transactions on Information Systems*, Vol. 12, No. 2, April 1994, pp. 150-173.

Watson H.J. and Haley B.J., (1997). "Data Warehousing: A Framework and Survey of Current Practices." *Journal of Data Warehousing*, Vol. 2 No. 1, January 1997, pp. 10-17.

Watson H.J. and Gray P., (1998). "New Developments in Data Warehousing." *Journal of Data Warehousing*, Vol. 3, No. 2, Summer 1998, pp. 8-11.

Watson H.J., Annino D.A., Avery L.K. and Gerard J.G., (2000). "Perspectives on Data Warehousing." *Journal of Data Warehousing*, Vol. 5, No. 3, Summer 2000, pp. 2-7.

Weiss D.M. and Lai C.T.R., (1999). *Software Product-Line Engineering: A Family-Based Software Development Process*, Reading, Massachusetts: Addison-Wesley.

Welke R.J., (1994). "The Shifting Software Development Paradigm." *DATA BASE*, Vol. 25, No.4, November 1994, pp. 9-16.

Werner V., Abramson C. and Kistler K., (2002). "E-Business and the Corporate Information Factory." *Journal of Data Warehousing*, Vol. 7, No. 2, Spring 2002, pp. 21-26.

Wheelwright S.C. and Clark, K.B., (1992). "Creating Project Plans to Focus Product Development." *Harvard Business Review*, March-April 1992, pp. 1-16.

White C., (1995). "A Technical Architecture for Data Warehousing." *InfoDB Journal*, Vol. 15, No. 2, February 1995, pp. 1-7.

White C., (1999). "Analytical Apps – Build or Buy." *Intelligent Enterprise*, Vol. 2, No. 11, August 1999, pp. 22-28, 38.

White C., (2000a). "Packaged Analytic Applications: The Key to Data Warehouse ROI." *DM Review*, Vol. 10, No. 1, January 2000, pp. 38-40.

White C., (2000b). "The Federated Data Warehouse." *DM Review*, Vol. 10, No. 3, March 2000, pp. 58-59, 79.

Wijnstra J., (2000). "Supporting Diversity with Component Frameworks as Architectural Elements." 50-59, *Proceedings of the International Conference on Software Engineering*, Limerick, Ireland, June 4-11, 2000, pp. 51-60.

Wijnstra J., (2002). "Critical Factors for a Successful Platform-Based Product Family Approach." *Proceedings of the Second International Conference of Software Product Line Conference (SPLC2)*, San Diego, CA, USA, August 19-22, 2002, pp. 68-89.

Withey J., (1996). *Investment Analysis of Software Assets for Product Lines (CMU/SEI-96-TR-010*, ESC-*TR-96-010)*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Yacoub S., Mili A., Kaveri C. and Dehlin M., (2000). "A Hierarchy of COTS Certification Criteria." *Proceedings of the First Software Product Line Conference (SPLC1)*, Denver, CO, USA, August 28-31, 2000, pp. 397-412.

Yin Robert K., (2003). *Case Study Research – Design and Methods*, Applied Social Research Method Series, Volume 5, Third Edition, Thousand Oaks, CA: Sage Publications, Inc.

Yoffie D.B. and Cusumano M.A., (1999). "Judo Strategy: The Competitive Dynamics of Internet Time." *Harvard Business Review*, Vol. 77, No. 1, January-February 1999, pp. 71-81.

Zachman J.A., (1997). "Enterprise Architecture – The Issue of the Century." *Database Programming & Design*, March 1997, pp. 44-53.

Özsu M.T. and Valduriez P., (1996). "Distributed and Parallel Database Systems – Technology and Current State-of-the-Art." *ACM Computing Surveys*, Vol.28, No.1, March 1996, pp. 125-128.

10. APPENDICES

10.1 Appendix 1 – Comparison of Product Platform Theories

The following table shows a collection of different characteristics of the product platform strategy literature both in physical and software product development. We compare whether these characteristics can be identified in a few selected literature sources.

Criterion	**Meyer and Lopez (1995)**	**Meyer and Zack (1996)**	**Meyer and Lehnerd (1997)**	**Meyer and Seliger (1998)**	**Sääksjärvi (1998, 2002)**
General definition of product platform	Yes	Yes	Yes	Yes	Yes
Clear definition of software product platform	No	No	No	No	Yes
General classification of product platform, platform extension and renewal	Yes	Yes	Yes	No	Yes
Software manufacturing	No	No	Yes	No	Yes
Process platform	Yes	Yes	Yes	No	Yes
Component-based software engineering (CSBE)	No	No	No	Yes	Yes
Generic product line development	Yes	Yes	Yes	Yes	Yes
Software product line development	No	No	No	No	No
Generic interface management	Yes	Yes	Yes	Yes	No
Software interface management	Yes	No	Yes	Yes	No
Software asset development	No	No	No	No	No
Generic IT infrastructure	No	No	No	No	Yes
Discussion of IT infrastructure from software engineering perspective	No	No	Yes	Yes	Yes
Importance of generic product architecture	No	Yes	Yes	No	No
Importance of software architecture	Yes	Yes	Yes	Yes	Yes
Layered component architecture	Yes	No	Yes	Yes	No
Technology strategy	Yes	No	No	No	Yes
Defining technology in product development	No	No	No	No	Yes
Supporting technology	No	Yes	No	No	No
Market segmentation/ leverage	Yes	Yes	Yes	Yes	Yes
Discussion of platform efficiency and effectiveness	No	Yes	Yes	No	Yes
Core competence or capability	Yes	Yes	Yes	No	Yes
Usage of external development organizations	No	No	Yes	Yes	Yes

10.2 Appendix 2 – Software Product Releases

The following table shows major product releases (both server and client module releases) introduced by the case study company from the foundation of the Company until the end of 2002. The aim of the table is to provide an overview of historical product releases and also to show major changes in each software release when there were also changes in the underlying server frame or client frame.

Major product release	**Server software**	**Client software**	**New software application frames and/or extensions**	**Used technology**
Version 1.0 (test or prototype release) 1989 – 1989	Server software release • Server software based on S/36 technology	Client software release • Harvard Graphics	None	IBM S/36 and RPG, Harvard Graphics
Version 2.0 1990 – 1992	Server software release • New AS/400-based server software	Client software release • Harvard Graphics	None	IBM iSeries/400 with RPG, Harvard Graphics, Synon 4GL
Version 2.3 1993 – 1993	Server software release • No major changes	Client software release • Harvard Graphics • A new own client frame introduced (based on OS/2) and Windows	None	IBM iSeries/400 with RPG, Synon 4GL, OS/2 on client desktop, C on client software development
Version 2.5 1994 – 1994	Server software release • No major changes	Client software release • No major change	None	IBM iSeries/400 with RPG, Synon 4GL, C on client software development
Version 3.0 1995 – 1995	Server software release • No major changes	Client software release • No major changes	None	IBM iSeries/400 with RPG, Synon 4GL, C on client software development

Major product release	**Server software**	**Client software**	**New software application frames and/or extensions**	**Used technology**
Version 4.0 **1996 – 1996**	Server software release • Server frame extension to support new COM-server • Database and database update changes to address larger databases addresses • Support for dynamic data groups	Client software release • Client frame extended to support 32-bit communication routers	New frame extension to support additional software releases • COM server for OLAP Server access • New budgeting and planning tool (based on Microsoft Excel) • New reporting tool for business analysis (based on Microsoft Excel) • New Java client/ server reporting architecture	IBM iSeries/400 with ILE RPG, Synon 4GL, C and C++ on client software development, Visual Basic, Visual Basic for Applications, Microsoft Excel, Microsoft COM, Lotus Notes and Domino, Java
Version 5.2 **1997 – 1997**	Server software release • Major rewrite of data warehouse server platform to increase performance and increase capacity • Use of new user index technology within the database architecture	Client software release • Support for OS/2 discontinued	New product platforms • New native 32-bit GUI builder for the server frame	IBM iSeries/400 with ILE RPG, Synon 4GL, C and C++ on client software development, Visual Basic, Visual Basic for Applications, Microsoft Excel, Microsoft COM, CASE tool development for new server GUI interface, Lotus Notes and Domino, Java
Version 5.6 **1998 – 1998**	Server software release • Support for calculated data types (extension for COM-server for OLAP Server data access)	Client software release • Conversion from 16-bit to 32-bit technology	New product releases • Lotus Notes-based CRM product	IBM iSeries/400 with ILE RPG, C and C++ on client software development, Visual Basic, Visual Basic for Applications, Microsoft Excel, Microsoft COM, CASE tool development for new server GUI interface, Lotus Notes and Domino, Java
Version 5.7 **1999 – 1999**	Server software release • Server frame extension to support multi-processor environments	Client software release • Last version of existing client release, replacement of new client frame	New client replacement • Replacement of existing client frame (first release of the new client software module)	IBM iSeries/400 with ILE RPG, C and C++ on client software development, Visual Basic, Visual Basic for Applications, Microsoft Excel, Microsoft COM, 4GL development tool for new server GUI interface, Lotus Notes and Domino, Java, ActiveX

Major product release	Server software	Client software	New software application frames and/or extensions	Used technology
Version 6.0 2000 – 2000	Server software release • Frame extension to support new chart generator in new client frame • Extension of the architecture with additional data warehouse tier, new application frame with corresponding frame components • Frame extension to support SQL drill through to relational data warehouses	Client software release • New chart generator included in the new client frame • Support for old client software module discontinued	New derived product to product family • New middle-tier server for centralized meta-data access (based on new client module components)	IBM iSeries/400 with ILE RPG, C and C++ on client software development, Visual Basic, Visual Basic for Applications, Microsoft Excel, Microsoft COM, 4GL development tool for new server GUI interface, Lotus Notes and Domino, Java, ActiveX, Distributed COM
Version 6.1 2001 – 2001	Server software release • No major changes to server release	Client software release • No major changes to client release	New derived product to client product family • Support for thin client technology via middle-tier architecture, new product for the client product family	IBM iSeries/400 with ILE RPG, C and C++ on client software development, Visual Basic, Visual Basic for Applications, Microsoft Excel, Microsoft COM, 4GL development tool for new server GUI interface, Lotus Notes and Domino, Java, ActiveX, Distributed COM Microsoft Internet Information Server technology
Version 6.2 2002 – 2002	Server software release • Improved multi-processor support	Client software release • New row- and column calculation editor in chart generator • New architecture for parameterized dashboard and presentations • Ability to integrate to third-party solutions such as CRM	No specific extensions	IBM iSeries/400 with ILE RPG, C and C++ on client software development, Visual Basic, Visual Basic for Applications, Microsoft Excel, Microsoft COM, 4GL development tool for new server GUI interface, Lotus Notes and Domino, Java, ActiveX, Distributed COM Microsoft Internet Information Server technology

10.3 Appendix 3 – Interviews and Other Communication

Place or Activity	Time and length	Interviewee	Topic
Helsinki, Finland	October 1997, 1 hr	Niina Niemi	Company background, financial and personnel information
Helsinki, Finland	October 1997, 1 hr 30 min	Jorma Vesterinen, main server software developer and founding member of the Company	Interview of software releases, specifics of each release, and major changes between releases
Helsinki, Finland	November 1997, 1 hr	Markku Riekkinen, server software developer and founding member of the Company	Interview of product releases, verification of collected information
Helsinki, Finland	November 1997, 30 min	Jyrki Martonen, main developer of initial client software release	Background information on development of the client platform, reasons for selected technology, and deficiencies found in later software releases
Helsinki, Finland	December 1997, 2 hrs	Esa Suurio, founder and CEO of the Company	Interview on the Company's foundation, overall view of the company and its direction and analysis of software development success
Helsinki, Finland	June, 1998, 2 hrs	Esko Jaatinen, Jari Laurila, Patrik Rosqvist	Interview of external third-party software development company; analysis of existing client software and its architectural deficiencies and suggestion for future development
RLI Corporation, Peoria	August 1999, 2 hrs 30 min	Diana Sutterfield, Cindy Brassfield	RLI application structure, interview of the solution built by RLI Corporation
Dallas, Texas	May 1999, 3 hrs	David Ortega, Tina Roos, Patrick Stapells	Requirement analysis of banking solution built on the core software solution of the Company
Dallas, Texas	March 2001, 3 hrs	Esa Suurio, CEO and founder of case study company	Discussion of product development strategies and product platform development with the founder of the Company
Telephone interview	April 2001, 2 hrs	Olli-Pekka Siikarla, Chief Architect	Analysis of product family development using new client software frame
Telephone interview	May 2001, 30 min	Aimo Asikainen, founder of the Company	Analysis of the success of the Company through today's date and suggestions for future development
Email correspondence	January 2001-June 2001	Aimo Asikainen, Esa Suurio, Jorma Vesterinen, Jussi Peltonen, Markku Riekkinen	Strategic discussion of the future development of the company by major shareholders of the Company
Telephone interview	May 2001, 30 min	Jorma Vesterinen, main server software developer and founding member of the Company	Analysis of the development success of the Company from its foundation through 2001

Place or Activity	Time and length	Interviewee	Topic
Telephone interview	May 2001, 1 hr 30 min	Stefan Westerbladh, main developer for budgeting and forecasting solution	Interview on development of platform extensions
Email questionnaire	May 2001	Jorma Vesterinen, main server developer and founding member of the company	Questionnaire on major product releases and the major changes in them, several documents sent via email
Email questionnaire	May 2001	Mika Rihtilä, Chief Financial Officer of the Company	Financial information about sales, product investment, etc.
Dallas, Texas	May 2001, 2 hrs 30 min	Pedro Puig, senior consultant responsible for banking solution	How does the solution support vertical solutions – experiences
Interview in Dallas, Texas	May 2001, 1 hr 30 min	Al Celaya, senior consultant responsible for training of the solution	How well does the solution lend itself to different vertical solutions – a trainers and consultants view
Kansas City, Kansas	June 2001, 2 hrs	Tom Leir, Vice President of Information Systems	Interview on the additional value that the detailed data warehouse architecture has brought Westlake Hardware
Helsinki, Finland	August 2001, 3 hrs	Olli-Pekka Siikarla, Chief Architect	Interview on the development history and evolution of the Adviser product family
Helsinki, Finland	August 2001, 3 hrs	Timo Sinisalmi, Team Leader for Adviser product family development	Interview on development history and accrued investments in development of the Adviser product family
Helsinki, Finland	August 2001, 2 hrs	Jussi Peltonen, CEO Mika Rihtila, CFO, Markku Riekkinen, Product Manager	Interview on accrued sales revenue for InfoManager product family development
Email questionnaire	August 2001	Mika Rihtilä, CFO Timo Sinisalmi, Team Leader	Questions about sales and product development with a historical perspective
Dallas, Texas	July 2002, 1 hr 30 min	Jorma Vesterinen	Review of appendix 1 – Product releases
Email questionnaire	July 2002	Mika Rihtilä, Chief Financial Officer of the Company	Inquiry on historical revenues from the foundation until the end of 2001 and sales per product category
Telephone interview	July 2002	Mika Rihtilä, Chief Financial Officer of the Company	Discussion of the financial results provided
Email questionnaire	July 2002	Mika Rihtilä, Chief Financial Officer of the Company	Inquiry of historical product development costs as a share of overall revenue per year
Email exchange	July 2002	Mika Rihtilä, Chief Financial Officer of the Company	Product sales from September 2000 through June 2002: Microsoft Excel spreadsheet
Email exchange with Finland	July 2002	Mika Rihtilä, Chief Financial Officer of the Company	Overall annual turnover information: Microsoft Excel spreadsheet
Telephone interview	July 2002	Mika Rihtilä, Chief Financial Officer of the Company	Discussion of the financial results provided

Place or Activity	Time and length	Interviewee	Topic
Email exchange	August 2002	Mika Rihtilä, Chief Financial Officer of the Company	InfoManager sales vs. product development costs plus other key performance indicators: Microsoft Excel spreadsheet
Email exchange	August 2002	Jarmo Jämiä, developer of relational data warehouse frame	Question about release dates for each version of the relational data warehouse product
Email exchange	September 2002	Mika Rihtilä, Chief Financial Officer of the Company	Relational Data Warehouse sales per customer and product
Email exchange	September 2002	Mika Rihtilä, Chief Financial Officer of the Company	Update of application frame information: Microsoft Excel spreadsheet
Email exchange	September 2002	Mika Rihtilä, Chief Financial Officer of the Company	Update of application frame information: Microsoft Excel spreadsheet
Email exchange	September 2002	Mika Rihtilä, Chief Financial Officer of the Company	Customer losses by year and customer: Microsoft Excel spreadsheet
Email exchange	September 2002	Mika Rihtilä, Chief Financial Officer of the Company	Update of application frame information: Microsoft Excel spreadsheet
Email exchange	September 2002	Mika Rihtilä, Chief Financial Officer of the Company	Update of application frame information: Microsoft Excel spreadsheet
Email exchange	September 2002	Mika Rihtilä, Chief Financial Officer of the Company	Update of customer losses by year and customer: Microsoft Excel spreadsheet
Email exchange	January 2003	Mika Rihtilä, Chief Financial Officer of the Company	Questions about the collected data and its calculation principles
Helsinki, Finland	January 2003, 3 hrs	Mika Rihtilä, Chief Financial Officer of the Company	Discussion of the results of the financial data collected
Email exchange	January 2003	Mika Rihtilä, Chief Financial Officer of the Company	Questions about the collected data and its calculation principles
Email exchange	January 2003	Mika Rihtilä, Chief Financial Officer of the Company	Update to product profitability and product sales: Microsoft Excel spreadsheets
Email exchange	February 2003	Mika Rihtilä, Chief Financial Officer of the Company	Update to product profitability due to an error in investment calculation: Microsoft excel spreadsheet
Dallas, Texas	February 2003, 2 hrs	Jorma Vesterinen, development manager for server frame	Review of investment calculations and product profitability numbers
Helsinki, Finland	February 2003, 1 hr 30 min	Marko Lehtamo, development manager of future server application frame	Discussion of future application frame and its importance for market segmentation
Helsinki, Finland	February 2003, 2 hrs	Esko Jaatinen, Patrick Rosqvist, Jari Laurila,	Discussion about client frame renewal and derivative development
Dallas, Texas	April 2003, 3 hrs	Jorma Vesterinen	Review of appendix 4

10.4 Appendix 4 – Analysis of Analytical Application Frame Strategy Evolution

The application frame analysis is divided into two separate tables, one depicting the evolution of server software and its corresponding application frame and the second table depictings the evolution of the client frame.

Server frame release and release year	**Number of worldwide customers, Number of domestic customers**	**Total software application frame cost (Millions in currency)**	**Software license revenue (Millions in currency)**	**Domestic Service Revenue (Millions in currency)**	**Product gross margin amount (Millions in currency)**	**Average software gross margin per worldwide customer installation (Thousands in currency)**
Server frame V1R1 Year 1990-1992 (Version 2.0)	44 (25)	2,559	3,368	2,580	0,809	18,4
Server frame V1R2 Year 1993-1993 (Version 2.3)	46(6)	0,668	3,115	0,951	2,447	53,2
Server frame V1R3 Year 1994-1994 (Version 2.5)	39(3)	0,637	2,360	0,406	1,723	44,2
Server frame V1R4 Year 1995-1995 (Version 3.0)	61(8)	0,858	3,803	0,975	2,945	48,3
Server frame V1R5 with extensions Year 1996-1996 (Version 4.0)	72(4)	0,677	3,965	1,199	3,288	45,7
Total server frame V1	**262(46)**	**5,399**	**16,611**	**6,111**	**11,212**	**42,8**
Server frame renewal, V2R1 Year 1997-1997 (Version 5.2)	97(9)	1,009	6,841	1,663	5,832	60,1
Server frame V2R2 – Ext.1 Year 1998-1998 (Version 5.6)	67 (6)	1,045	4,794	1,538	3,749	56,0
Server frame V2R3 Ext.2 Year 1999-1999 (Version 5.7)	47 (5)	1,103	5,510	1,864	4,407	93,8

Server frame release and release year	**Number of worldwide customers, Number of domestic customers**	**Total software application frame cost (Millions in currency)**	**Software license revenue (Millions in currency)**	**Domestic Service Revenue (Millions in currency)**	**Product gross margin amount (Millions in currency)**	**Average software gross margin per worldwide customer installation (Thousands in currency)**
Server frame V2R4 Ext.3 Year 2000-2000 (Version 6.0)	35 (9)	1,121	6,156	1,595	5,035	143,9
Server frame V2R5 Year 2001-2001 (Version 6.1)	22(5)	0,809	6,957	1,438	6,148	279,5
Server frame V2R6 Year 2002-2002 (Version 6.2)	25(3)	0,655	6,472	1,323	5,817	232,7
Total server frame V2	**293(37)**	**5,742**	**36,731**	**9,420**	**30,989**	**105,8**
Total server frame V1+V2	**555(83)**	**11,140**	**53,342**	**15,531**	**42,201**	**76,0**

The following table depicts the evolution of client frames for the Company. Each client frame release must be synchronized with server frame development, as they must be implemented in sync.

Client frame release and release year	**Number of worldwide customers, Number of domestic customers**	**Total software application frame cost (Millions in currency)**	**Software license revenue (Millions in currency)**	**Domestic Service Revenue (Millions in currency)**	**Product gross margin amount (Millions in currency)**	**Average software gross margin per worldwide customer installation (Thousands in currency)**
Client frame V1R1 Year 1993-1993 (Version 2.3)	46(6)	0,202	0	0	-0,202	-4,3
Client frame V1R2 Year 1994-1994 (Version 2.5)	39(3)	0,276	0,498	0,174	0,222	5,7
Client frame V1R3 Year 1995-1995 (Version 3.0)	61(8)	0,378	0,839	0,418	0,461	7,6
Client frame V1R4 Year 1996-1996 (Version 4.0)	72(4)	0,461	0,859	0,514	0,398	5,5
Client frame, V1R5 Year 1997-1997 (Version 5.0)	97(9)	0,864	2,023	0,492	1,159	12.0
Client frame V1R6 Year 1998-1998 (Version 5.x)	114 (11)	1,810	5,319 (sales of from 1998 to 2002)	1,665 (consulting from 1998 to 2002)	3,509	30,8
Total client frame V1	**429(41)**	**3,991**	**9,537**	**3,263**	**5,546**	**12,9**
Client frame V2R1 Year 1999-2000 (Version 1.0)	35 (9)	3,541	2,793	0,178	-0,748	-21,4
Client frame V2R2 Year 2001-2002 (Version 2.0)	47 (8)	2,177	4,255	0,869	2,078	44,2
Total client frame V2	**82(17)**	**5,717**	**7,049**	**1,047**	**1,332**	**16,2**
Total client frame V1+V2	**511(58)**	**9,708**	**16,586**	**4,310**	**6,878**	**13,5**

Client frame release and release year	**Number of worldwide customers, Number of domestic customers**	**Total software application frame cost (Millions in currency)**	**Software license revenue (Millions in currency)**	**Domestic Service Revenue (Millions in currency)**	**Product gross margin amount (Millions in currency)**	**Average software gross margin per worldwide customer installation (Thousands in currency)**
Client frame derivative product (Webulator) Year 2000-2001 (Version 1.0)	14(6)	1,967	2,100	0,049	0,133	9,5
Client frame derivative product (Webulator) Year 2002-2002 (Version 1.2)	15(4)	0,848	1,184	0,036	0,336	22,4
Total Webulator development	**29(10)**	**2,815**	**3,284**	**0,085**	**0,469**	**16,2**
Client frame derivative product (Broker) Year 2000-2001 (Version 1.0)	21(9)	0,211	1,507	0	1,295	61,7
Client frame derivative product (Broker) Year 2002-2002 (Version 2.0)	14(2)	0,003	0,644	0	0,641	45,8
Total Broker development	**35(11)**	**0,214**	**2,151**	**0**	**1,936**	**55,3**
Total Derivative development	**64(21)**	**3,029**	**5,436**	**0,085**	**2,405**	**37,6**