HELSINKI UNIVERSITY OF TECHNOLOGY

Department of Industrial Engineering and Management

Institute of Strategy and International Business

Jarno Vähäniitty

Product Strategy Decisions in Small Software Product Businesses – A Framework and Experiences

Master's thesis submitted for official examination for the degree of Master of Science in Technology in Espoo on Jan 8th, 2003.

Supervisor: Professor Casper Lassenius

Instructor: Kristian Rautiainen M.Sc.

HELSINKI UNIVERSITY OF TECHNOLOGY **ABSTRACT OF THE MASTER'S THESIS**

Author:	Jarno Vähäniitty	
Title of the thesis:	Product Strategy Decisions in Small Software Product Businesses – A Framework and Experiences	
Date:	Jan 8th, 2003	**Number of pages:** 72
Department:		**Professorship:**
Department of Industrial Engineering and Management		T-76 Development of Digital Products

Supervisor: Professor Casper Lassenius

Instructor: Kristian Rautiainen, M.Sc., Helsinki University of Technology

In the software product business, success involves managing a complex set of activities and product strategy has been identified as a management area of crucial importance.

Existing literature has mainly discussed product strategy and new product development from the perspective of large companies and software development with the focus on customer-specific projects. Small software product businesses are vulnerable to extensive rework and market failure due to shortcomings in their product strategy decision-making, but the literature offers very little context-specific advice. Without a context-specific understanding of the underlying issues, small companies find it very hard to benefit from existing knowledge on the subject.

This study proposes that product strategy decision-making in small software product business can be supported by identifying what must be accounted for in the strategic management of their product development. A framework of the key decision areas in formulating and enacting product strategy is constructed based on the needs of small software product companies, as identified from the literature and three case studies. The identified key product strategy decision areas are *organisation*, *portfolio management*, *requirements*, *development strategy*, *technology* and *quality strategy*. While these management areas are not equally topical for all companies at a given time, all of them should be considered in the product strategy of a company.

The key product strategy decision areas framework supports product strategy decision-making by helping the key persons acting in multiple and sometimes even contradictory roles and responsibilities to maintain a holistic perspective under the pressures from different stake-holders. The framework can also be used as a checklist in evaluating and improving product strategy decision-making practices.

The effectiveness of the framework for supporting product strategy decision-making is evaluated by using it to examine, analyse and improve the practices of three small software product companies. As a result, the companies' awareness of their problems and challenges improved, tangible and relevant improvement suggestions were made, and over a follow-up period of four to six months, all of the companies had acted on most of the identified problems and challenges as well as the improvement suggestions presented.

Keywords:	Product strategy, software product business, small companies, decision-making, new product development, software development, software engineering

TEKNILLINEN KORKEAKOULU **DIPLOMITYÖN TIIVISTELMÄ**

Tekijä:	Jarno Vähäniitty	
Työn nimi:	Tuotestrategiapäätökset pienissä ohjelmistotuoteyrityksissä – viitekehys ja kokemuksia	
Päivämäärä:	8.1.2003	**Sivumäärä:** 72
Osasto:		**Professuuri:**
Tuotantotalous		T-76 Digitaalisten tuotteiden kehittäminen

Työn valvoja: Professori Casper Lassenius

Työn ohjaaja: DI Kristian Rautiainen, Teknillinen korkeakoulu

Ohjelmistotuoteliiketoiminnassa menestyminen vaatii liikkeenjohdolta monipuolista ja syvällistä osaamista. Tuotestrategia on kuitenkin tunnistettu tärkeimmäksi osa-alueeksi ohjelmistoyrityksen johtamisessa.

Tuotestrategiaa ja tuotekehityksen johtamista on perinteisesti tarkasteltu suurten yritysten näkökulmasta, ja suurin osa ohjelmistotuotannon tutkimuksesta käsittelee suuryritysten projektiliiketoimintaa. Strategisten tuotepäätösten tekeminen pienissä ohjelmistotuoteyrityksissä on usein lapsenkengissään, mikä saattaa ne alttiiksi liiketoiminnan karikoille. Suoraan pienille ohjelmistotuoteyrityksille suunnattuja neuvoja löytyy kirjallisuudesta hyvin vähän. Ilman syvällistä ymmärtämystä tämän kontekstin erityispiirteistä voi olemassa olevan tiedon soveltaminen olla vaikeaa.

Tässä työssä esitetään, että pienten ohjelmistotuoteyritysten tuotestrategian tukemisen tulisi lähteä tuotestrategian kokonaisuuden ymmärtämisestä. Työssä määritellään viitekehys keskeisistä tuotestrategiapäätöksistä kirjallisuuden ja kolmen tapaustutkimuksen pohjalta. Viitekehyksen pääalueet ovat *organisaatio*, *tuoteportfolion hallinta*, *vaatimukset*, *kehitysprosessi(t)*, *teknologia* ja *laatu*. Vaikka alueiden keskinäiset painotukset ja tärkeys vaihtelevat yrityksissä, tulisi kaikkiin kiinnittää huomiota yrityksen tuotestrategiasta päätettäessä.

Keskeisten tuotestrategiapäätösten viitekehys tukee tuotestrategiapäätöksentekoa auttamalla monissa ja toisinaan ristiriitaisissakin rooleissa toimivia pienyrityksen avainhenkilöitä pitämään kokonaisuus mielessään eri sidosryhmien paineiden alla. Viitekehys soveltuu myös käytännön muistilistaksi yrityksen toimintatapoja arvioidessa ja kehitettäessä.

Viitekehyksen arvoa tuotestrategiapäätöksenteon tukena tutkittiin käyttämällä sitä prosessin kehittämisen apuvälineenä ja seuraamalla tuloksia kolmessa pienessä ohjelmistotuoteyrityksessä. Yritysten tietoisuus omista kehittämiskohteistaan ja haasteistaan nousi, ja konkreettisia kehitysehdotuksia pystyttiin antamaan. 4-6 kuukauden tarkastelujakson päättyessä oli jokainen tutkimukseen osallistunut yritys toiminut lähes kaikkien yhdessä tunnistettujen kehittämiskohteiden suhteen sekä reagoinut useimpiin tehdyistä kehitysehdotuksista.

Avainsanat:	Tuotestrategia, ohjelmistotuoteliiketoiminta, pienyritys, päätöksenteko, tuotekehitys, ohjelmistokehitys, ohjelmistotuotanto

Acknowledgements

First and foremost, I would like to thank the supervisor and instructor of this work, prof. Casper Lassenius and Kristian Rautiainen, respectively. Appropriately, they also are the most important mentors during my career as a researcher so far.

Thanks to the rest of the SEMS research team, namely Juha Itkonen, Mika Mäntylä and Jari Vanhanen for their comments and most of all, the great working environment we have.

During the initial case interviews, Ari Torpo helped me by demonstrating how to ask nasty questions from nice people regarding their work. Also, an important thanks goes to Maaret Pyhäjärvi who contributed to the picture of what small software product businesses are like.

Last but definitely not least, I want to thank my family and friends who encouraged and helped me in completing this work.

Espoo, 8th Jan 2003

Jarno Vähäniitty

Table of Contents

1 Introduction

1.1 Background

The software industry has become an important creator of wealth. It is an industry of extreme success, job creation, extraordinary growth and accelerated product cycles. (Hoch et al. 2002) But, as evident from the rise and fall of the hype surrounding the turn of the millennium, failures due to unrealistic assumptions and inadequate planning can be just as spectacular.

In the software product business, success often involves managing a complex set of activities and requires more than the capability to invent and realise new solutions. These activities include, but are not limited to, evolving both the individual products and the technologies they are based on at the same time (Cusumano & Yoffie 1998), and delivering these solutions as a combination of the product and related services to the market at the right time and with the right kind of quality (McGrath 2000). In practice, this means that managing the architecture, contents, timing and roles of the releases plays a crucial part.

Thus, it is not surprising that product strategy has been identified as perhaps the most important management area in the software product business. *Product strategy* answers to the questions *'Where are we going, how will we get there, and why will we be successful?'* from the perspective of what the company offers (McGrath 2000). The function of product strategy is to *link the company's product development to its business strategy* (McGrath, Anthony, & Shapiro 1996) and to *ensure that the firm and its products pursue the right markets from a strategic viewpoint* (McGrath 2000). Product strategy can also be viewed as *the result of making important decisions in managing new product development* (Krishnan & Ulrich 2001; Mintzberg, Ahlstrand, & Lampel 1998). In this thesis, these important decisions are termed *product strategy decisions*, and making them is called *product strategy decision-making*. The process for making product strategy decisions in a company is referred to as the *product strategy process*.

While managers regard product strategy formulation and enactment as the most important topic in technology management (Scott 2000), product strategy decisions are usually made ad hoc in the industry and a systematic approach to strategic decision-making is often missing in practice (Krishnan & Ulrich 2001). Especially in small companies, strategic action is often based on the opinions of the key personnel rather than through deliberate or explicit planning (Brouthers, Andriessen, & Nicolaes 1998).

Small companies play a role of crucial importance because of their innovativeness, popularisation of new technologies, keeping established firms on their toes and sometimes even through creating entirely new industries. Still, much of the literature excludes the perspective of very small firms (Fayad, Laitinen, & Ward 2000; Naumanen 2002) and there has been little interest in the strategic issues of small companies. Even those authors who claim to target small firms often define these as having up to 500 employees (Krishnan & Ulrich 2001; Smith 1998).

In the ongoing SEMS (Software Engineering Management System) research project at Helsinki University of Technology, we are studying the management of software engineering in small companies in the software product business. By small we mean companies having less than 50 people, and by product business that the amount of customer-specific development effort is (or at least is aimed to be) relatively small[*].

According to existing research (Brouthers, Andriessen, & Nicolaes 1998; Mello 2002; Smith 1998) and our experiences, small companies are vulnerable to extensive rework and market failure due to shortcomings in their product strategy process. Some examples of factors affecting the situation are inexperience, time-to-market pressures and the lack of process infrastructure such as requirements management (Carlshamre et al. 2001). To complicate the matter further, key personnel, in our experience, often have truly cross-functional roles, ranging from architecture design to deploying the system, customer-specific tailoring, consulting and sales. Combined with unclear priorities caused by lack of long-range planning, overbooking of resources is common while some important activities do not receive enough attention.

However, both our experience and the literature (Berry 2002; Ward, Laitinen, & Fayad 2000) suggest that systematic approaches to product strategy decision-making help to concretise and communicate the strategy of the company so it can be acted on, refuted, or set aside when necessary, and that a fast learning curve in strategic decision-making is possible. For example, taking a long-term view into release management can assist small companies to bring together the perspectives of business management and software development (Rautiainen, Lassenius, & Sulonen 2002; Vuornos 2002).

A large number of techniques, tools and methods for tackling the problems facing practitioners in product strategy decision-making have been presented in the literature (Cooper, Edgett, & Kleinschmidt 2001). Still, most of these have been designed from the perspective of large companies with multiple business units, each with possibly several product lines. Very little guidance exists for helping very small companies to combine the perspective of business planning with managing product development. While small companies cannot afford the strategic planning staff and personnel that larger firms possess because of their size (Brouthers, Andriessen, & Nicolaes 1998), our experiences suggest that another significant contributor to the problems mentioned may be the difficulty for the key persons to see the 'big picture' in the everyday bustle of multiple, and sometimes even contradictory roles and responsibilities. These factors make it important to find means for supporting product strategy decision-making in small software product businesses.

1.2 Research Problem

The research problem of this thesis is stated as follows:

How can product strategy decision-making in small software product businesses be supported?

[*] See section 3.1 for a more thorough definition of the context.

A solution to the research problem is sought through answering the following research questions:

(1) What issues should product strategy decision-making entail?

(2) How well are these issues currently handled in small software product businesses and is the state-of-practice satisfactory?

(3) What has been proposed in literature to support product strategy decision-making, and how does this support fit the small business context?

(4) Provided there is a gap between the needs and the support found, how can this gap be overcome?

(5) Does the proposition for overcoming the gap between the needs and support found facilitate product strategy decision-making in practice?

The first research question addresses the concept of product strategy and the its role in the software product business based on literature.

The second research question asks how well small software product companies conduct product strategy decision-making and whether a need for improvement exists.

The third research question discusses existing solutions, techniques and frameworks for supporting strategic decision-making in new product development (NPD) and software engineering, and their suitability to support product strategy decision-making in small software product businesses.

The fourth research question asks what kind of support, specifically, would seem most useful in the light of the small software product business context and existing literature, and leads to proposing a framework to answer the research problem.

The fifth research question leads to evaluating the usefulness of the proposition in facilitating strategic product development decision-making in small software product businesses. This is done by using the proposed framework to describe how three companies conduct product strategy decision-making, identifying relevant areas for improvement, presenting improvement suggestions and observing the results of this analysis over a period of time.

1.3 Methodology

Answers to the research questions are sought through literature study, the construction of a framework and applying it in case companies. These activities and their goals in relation to the research questions are described below.

Answering research question (1) ("What issues should product strategy decision-making entail?") starts with discussing a list of key decisions in new product development and continues throughout the thesis in the form of constructing a framework of key product strategy decisions for small companies in the software product business.

The state-of-practice of small software product companies for research question (2) ("How well are these issues currently handled in small software product businesses and is the state-of-practice satisfactory?") is based on comparing our experiences from research on software process improvement in small software companies to a checklist of common problems in new product development.

Research question (3) ("What has been proposed in literature to support product strategy decision-making, and how does this support fit the small business context?") brings forth a review of strategic planning, NPD management and software engineering literature from the perspective of finding existing frameworks and tools to help in product strategy decision-making and evaluating their appropriateness to the context. All of these domains cover the problem area of this study but have slightly different viewpoints. This is discussed in more detail in section 2.3.

Answering research question (3) also shows that there is a gap between existing support and the needs of small software product companies. This gap leads to examining the nature of small companies in the software product business in detail based on NPD management and software engineering literature and the experiences of the author and organising the findings using a framework of key new product development decisions from literature. This examination yields an answer proposal to research question (4) ("Provided there is a gap between the needs and the support found, how can this gap be overcome?") in the form of constructing a framework of the key product strategy decisions that small companies in the software product business face.

The construction process of the framework consists of two approaches conducted in parallel, a theoretical (chapter 3) and an empirical one (chapter 4), and of combining their results in chapter 5.

The theoretical approach constructs an outline of the of key product strategy decisions in small software product businesses by reviewing and analysing strategic planning, NPD management and software engineering literature about the characteristics of small software product businesses and their implications to product strategy decision-making. The results of this analysis are organised as a framework of key product strategy decisions with the help of an existing model of key issues in managing new product development.

The empirical approach abstracts key decision areas from an outline for a product strategy process, and uses the resulting set of areas to describe the practices of three small software product companies. Semi-structured interviews are conducted in three case companies. After the first round of interviews, the data is thematically coded using the elements of the framework outline, and the outline is improved to better cover the areas of importance that surfaced during the interviews. The framework is refined according to the lessons learned. Descriptions of the companies are written based on the improved framework outline, which is also the result of the empirical approach. The answer to research question (4) is gained by combining the outline of product strategy decision areas from the empirical phase to the results of the theoretical approach to create a framework.

To answer research question (5) ("Does the proposition for overcoming the gap between the needs and support found facilitate product strategy decision-making in practice?"), empiria is needed to test the framework that was constructed to answer research question

(4). The effectiveness of using the framework for increasing the company's awareness of its own practices is evaluated by examining whether relevant problems and challenges are found by using it.

The empirical approach used to construct the framework also serves in providing the data necessary to answer research question (5). After writing the case descriptions, they are presented to the companies along with improvement suggestions. Then, a second round of interviews is conducted after an observation period to see whether and how the companies acted on the suggestions given. Together with the answer to research question (4), the answer to research question (5) is proposed to solve the original research problem.

The methodology and the steps taken during this study to answer the research problem are illustrated in section 6.1 (Answering the Research Problem) in Figure 9 and Figure 10.

1.4 Scope

In addition to the scope defined by the research problem, several important issues have been left out of the scope of this study or covered only briefly.

Software development departments of large companies. While relatively independent software development departments of larger companies have been noted to possess characteristics similar to small companies (Brodman & Johnson 1994), this perspective is not discussed.

Hardware development. Although one of the case companies develops both software and hardware for its products, hardware development is not taken explicitly into account in the framework.

Focus on decision-making. The focus of the framework excludes a number of important environmental and contextual variables to product strategy decision-making, such as market size, growth rate, or the competitive environment. These details do not affect the contents of the framework because of the *decision perspective* used (see section 2.1).

Conducting product strategy decision-making. The main emphasis of this thesis is on finding out and defining what product strategy decision-making in the context of small software product businesses is. While the process of conducting product strategy decision-making is given less attention, some basics are presented in section 5.3 for the purposes of understanding how the framework supports the decision-making process itself.

Explicit analysis of the impact of business environment and business model on product strategy decision-making. A business model consists of a set of mechanisms for value creation and appropriation (Rajala et al. 2001), and in recent research (Rautiainen, Lassenius, & Sulonen 2002) it has been suggested that a company should actively seek to reinforce the link between its internal processes, such as the software development process and the other elements of its business model. While the author takes the position that a fit between the intended business models, the business environment and the product development process(es) is crucial, explicitly discussing this perspective further is out of scope.

1.5 Structure and Outline of the Thesis

This section describes the structure and contents of the thesis and the contribution of the chapters to answering the research problem. Note, that answering research questions (1), (3), and (4) is highly interrelated, and is thus the subject of many chapters in this thesis.

In this chapter, the background and the motivation for this thesis were presented. The research problem was broken into research questions, and the methodology and scope were presented. The contents of this work are being described.

Chapter 2 (Small Software Businesses and Product Strategy) examines product strategy decision-making in small software businesses based on literature and the experience of the author, and the need for a new framework is identified. This chapter contributes to answering research questions (1), (2), (3) and (4).

In chapter 3, the construction of the framework is undertaken through a closer examination of the characteristics attributed to small software product companies. These attributes and their managerial implications are identified from the literature and the experience of the author. The managerial implications found are used to identify important product strategy decisions and these are organised as a framework for facilitating product strategy decision-making in small software product businesses. Chapter 3 contributes to answering research question (4).

Chapter 4 presents the design and results of an empirical approach. The empirical approach is used for two purposes. First, the empiria adds depth to the framework being constructed by examining strategy-level decision-making in small software product businesses in practice. Second, it tests the idea behind constructing a framework of key decisions. This is done by attempting to provide relevant improvement suggestions to the practices of three small software product companies based on an analysis conducted using such a framework. This chapter contributes to answering research questions (1), (4) and (5).

Chapter 5 presents and defines a framework that breaks product strategy in small product-oriented software companies into key decision areas for the purposes of describing, evaluating and improving product strategy decision-making in actual companies. Also, the product strategy *process* is briefly discussed. This chapter contributes to answering research questions (1) and (4).

Chapter 6 (Discussion) rounds up the study. The research problem is answered based on the answers to the research questions, the contribution of the study is outlined, the usefulness and limitations of the framework as well as its construction process are evaluated, and the thesis is concluded with highlighting directions for further research.

2 Small Software Businesses and Product Strategy

This chapter examines high-technology product strategy and decision-making in small software businesses based on literature and the experience of the author (section 2.1). It suggests that practitioners in such companies not only face considerable challenges but also fail to benefit from existing literature (sections 2.2 and 2.3). In section 2.3.4, the construction of a framework is proposed to support product strategy decision-making.

2.1 Decision Perspective to Formulating and Enacting Product Strategy in High-Technology Companies

We will now take a closer look at what product strategy consists of using a list of key product development decisions (Krishnan & Ulrich 2001). To construct this list, Krishnan and Ulrich organised new product development literature through using *the decision perspective*. This means that while the *how* in developing products varies across and within companies over time, *what is being decided* remains constant at a certain level of abstraction. The decision perspective can be used to achieve a holistic view of the issues of importance in new product development instead of a functional one. From the decision perspective, product development decisions are organised into two broad categories, those made in *setting up development projects* and those made *within a project*.

The main decisions in setting up projects (Table 1) are *product strategy and planning*, *product development organisation* and *project management*. *Product strategy and planning* involves decisions about the firm's target market, product mix, project prioritisation, resource allocation, and technology selection. *Product development organisation* means the social system and environment in which a firm's design and development work are carried out. Related decisions are team staffing, incentives and reward systems, metrics for monitoring performance, and investments in productivity-enhancing tools and "processes" for product development. In development *project management* decisions are made about the relative priority of development objectives, the planned timing and sequence of development activities, the major project milestones and prototypes, mechanisms for coordination among team members, and means of monitoring and controlling the project.

Table 1 Key decisions in setting up a new product development project (Krishnan & Ulrich 2001)

Key decision area	**Decisions**
Product strategy and planning	Employed technologies
	Portfolio of product opportunities to be pursued
	Product architecture
	Timing of product development projects
	Shared assets (platforms) across products
	Employed technologies
Product development organisation	General organisation (e.g. functional, project, matrix)
	Team staffing
	Project performance measurement
	Team physical arrangement and location
	Investments in infrastructure, tools and training
	Type of development process (e.g. Stage-gate)
Project management	Relative priority of development objectives
	Timing and sequence of development activities
	Project milestones and planned prototypes
	Communication mechanisms among team members
	Project controlling and monitoring

The types of decisions made *within a project* (Table 2) are *concept development*, *supply chain design*, *product design*, *performance testing and validation* and *production ramp-up and testing*.

Concept development decisions define the product specifications, the product's basic physical configuration and any extended product offerings such as life-cycle services and after-sales supplies.

Supply chain design encompasses flows of materials and the supply of intellectual property and services to the firm. Supply-chain design decisions therefore include supplier selection as well as production and distribution system design issues. Note, that some of the supply chain decisions may be of less importance for a company in the software product business, as will be discussed later in this section.

Product design refers to the detailed design phase, which constitutes the specification of design parameters, the determination of precedence relations in the assembly, and the detail design of the components, including material and process selection.

While detailed design decisions are being made and refined, the design is also *prototyped and validated* for fit, function, and fabrication. Typically, the firm has a choice of developing prototypes sequentially or in parallel with different cost, benefit, and time implications. In association with product launch and production ramp-up, a number of decisions must be made. The firm must decide the degree to which test marketing should be done, and the

sequence in which products are introduced in different markets. Launch timing is a decision that trades off multiple factors, including threat of competitor entry and the completeness of development. The firm must be careful in communicating its launch timing to the market, as not meeting pre-announced launch dates may have a significant impact on how it is perceived.

Table 2 Key decisions within a product development project (Krishnan & Ulrich 2001)

Key decision area	**Decisions**
Concept development	Target values of product attributes (including price)
	Core product concept
	Offered variants
	Shared components across variants
	Physical form and industrial design
Supply chain design	Component design
	Component production and assembly
	Supply chain configuration
	Assembly process
	Process technology and environment
Design	Values of key design parameters
	Component configuration and assembly precedence
	Detailed component design
Performance testing and validation	Prototyping plan
	Technologies used in prototyping
Production ramp-up and launch	Plan for market test and launch
	Plan for production ramp-up

Looking at the two categories of product development decisions and their contents, many of the decisions are very topical in the development of software products, but also, some, especially those within a project, seem less relevant from the software perspective. For example, supply chain design and management with respect to software business has many profound differences in comparison with manufactured products. Also, because of the bias toward manufactured products, many of the management issues critical for software development, such as testing and quality assurance are either not discussed, or emphasised correctly.

The list has also been constructed from the perspective of larger companies. This is for example shown in the basic division to conducting and setting up projects, which implies both multiple simultaneous projects and organisational decision-making boundaries between the two decision-making types.

In small companies, there are typically only a few simultaneous projects running, and due to the key personnel having multiple roles and responsibilities, the categories are less clear-cut, providing a degree of flexibility that may be difficult to reach in larger companies.

Because of this, product strategy decisions in small software product businesses are most likely a combination of decisions from both setting up and from within development projects. For example, in a small company, decisions on "assembly" process technology and environment might have company-wide significance both in terms of development process and tool cost, and can be coupled to the technologies employed in the product as well. Another example is the area referred to in the model as supply chain design. Excluding small software companies that use outsourcing, supply chain simplifies in most cases as the distribution channel.

In conclusion, the discussed list is excellent for providing an overview of the issues that are involved in product strategy decision-making, and thus works as a more exact definition of the concept of product strategy. However, from the perspective of applying it to support product strategy decision-making in small software product companies, the list contains issues of questionable relevance, lacks industry-specific detail, and gives a biased and incomplete picture of what kinds of decisions might have strategic implications in a small organisation having more roles than people.

2.2 Problems from Inadequate Decision-Making in Four Small Finnish Software Product Businesses

In literature on managing new product development, product development is seen as the most important way to bring strategy to life (Cooper, Edgett, & Kleinschmidt 2001). Thus, it is not surprising to see several studies suggesting that a formal product or technology strategy and explicit strategic planning are significantly associated with the success of start–ups and small companies (Berry 2002; Smith 1998; Zahra & Bogner 1999).

If product strategy is considered in practice to be the result of making decisions such as those outlined in Table 1 and Table 2, what may happen when such decisions are not properly taken? As an example answer, a list of common problems in new product development is presented in Table 3 (McGrath, Anthony, & Shapiro 1996).

Table 3 Common problems in new product development (McGrath, Anthony, & Shapiro 1996)

Cycle-time and pacing guidelines are not used to validate development schedules
Poor execution of development due to lack of common understanding of the development process, for example, cycle-time guidelines
Unclear product strategy process results in product strategy being formulated superficially as part of annual budgeting
No explicit consideration for company growth, product mix or short and long-range emphasis in product development decision-making
Product strategy is formulated without involving the customer interface
Competitive positioning is unclear and role of competitor analysis shallow
Strategic decisions on where the product is going are made in frustration by developers because senior management has not made them
Decisions are made too late, for example, when considerable costs have already been committed
Fire-fighting decisions are made without the context of development priorities
Failure to invest in current and future core technologies
Decisions are based on inadequate information because the proper level of detail is unknown
Unnecessarily long development cycles because technology development is not decoupled from product development
Decision points or milestones are not defined

Lacking a list more suitable to the context, the key decisions in new product development in section 2.1 (Krishnan & Ulrich 2001) can be elected to represent product strategy decision-making in small software product businesses. Comparing the key product development decisions from Table 1 and Table 2 to the list of common problems in Table 3, it seems that many of the problems are intimately related to product strategy decision-making. Thus, the author proposes that a comparison with the list of common problems is a rough but adequate measure to evaluate the general need for improvement in product strategy decision-making.

Towards this end, four small software product companies we have worked with are examined and their state-of-practice is compared to a list of "product development best practices" derived from the list of common problems in Table 3. The state-of-practice the comparison is made against represents the companies' practices *at the start of the co-operation*[1]. The hypothesis is that if many of the "best practices" thus derived seem missing, it can be concluded that the product strategy decision-making practices of small software product companies are likely in need of improvement efforts.

Table 4 exhibits the results of the comparison. While slight individual differences between the companies existed, the most common situation found at the time is presented for the sake of simplicity.

[1] Those of the companies still operating at the time of writing this have experienced improvements in these areas

Table 4 Results of the comparison

Do cycle-time and pacing guidelines exist?	No guidelines for pacing or project scheduling
Are development schedules affected by cycle-time and pacing guidelines?	Development effort may not be scheduled at all
Does a common understanding of the development process exist?	Development process not defined
Is the product strategy process clear?	No process for product strategy decision-making, scope of product strategy unclear
Does an explicit consideration for company growth, product mix, or short and long-range targets exist?	No explicit consideration (hard to express upon asking)
Is the customer interface involved in product strategy decision-making?	Customer interface and product strategy decision-making not defined
What is the role of competitor analysis and is competitive position-ing is clear?	Limited or no awareness of competitors and the products' competitive positioning
Is decision-making responsibility explicitly role-based?	Responsibility not explicitly allocated. Strategic decisions are made on all levels and senior management overrides as deemed necessary
Do development priorities extend to fire-fighting decisions as well?	No explicit development priorities (Fire-fighting is the most common mode of operations)
Is a 'good-enough' level of knowl-edge to make product development decisions somehow characterised?	No, and no explicit connection to decision types to roles exists either
Is technology development decoup-led from product development?	Companies have started to notice that coupling leads to long lead times in development
Is investing in current and future core technologies actively sought for?	Long-term plans deemed important but do not exist in practice
Does the development process have defined decision points or mile-stones?	Product development work aiming at product releases does not have clear start or end points
Is committing costs controlled together with development progress?	No formal measures of development status, progress or costs

The questions in Table 4 are by no means meant as an exhaustive checklist of NPD best practices. Still, looking the results of the comparison, the examined companies had limited awareness of the issues typically associated with successful product development. Also, tangible applications based on such principles were uncommon.

From this comparison it can be concluded that while practicing product strategy is hard in general, for small companies and start-ups it is even harder because of the low level of awareness such companies seem to exhibit. This notion is also supported by existing research (Berry 2002; Mello 2002).

2.3 Existing Support for Product Strategy Decision-Making

In this section, we will take a look at how existing literature supports product strategy formulation and enactment in small software product businesses.

2.3.1 Strategic Planning

Effective planning is crucial for all companies, and for a start-up it often makes the difference between survival and ruin. This is especially true for companies in the software product business because of shortened cycle times. Ironically, these may also be just the companies that are most tempted to bypass the planning stages in an attempt to shortcut the development process and bring products to market more quickly (Mello 2002). However, practicing strategic planning can be a complicated affair and not easily captured by a single perspective (Mintzberg, Ahlstrand, & Lampel 1998). Thus, a discussion of the role and usefulness of strategic planning in the context of this study is necessary.

Strategic planning often consists of steps such as analysing the industry, selecting a strategy and building tactics around it (Mintzberg, Ahlstrand, & Lampel 1998). These kinds of approaches have been challenged on the grounds that they are based on theoretical ideals having little to do with management realities. Indeed, a major problem with this kind of planning is that the outcome of such an activity, plans, are virtually always to some extent 'wrong' (Brown & Eisenhardt 2002). This is because the analysis may not reflect all relevant factors and plans become obsolete quickly.

In the context of this study it is interesting to note that critics have questioned the value of strategic management procedures in the case of companies operating in the turbulent environment of high-technology industries and small companies lacking the management and financial resources to "indulge in elaborate strategic management techniques". It has been argued that formal strategic management procedures, such as environmental forecasting and long range planning, are inappropriate or at least of questionable value for small companies. This is especially topical for companies operating in environments where conditions change so fast that environmental forecasting becomes meaningless. (Berry 2002)

Despite the criticism, there are several strengths inherent to strategic planning regardless of industry and environment. Strategic planning serves a symbolic role, and helps in coordinating a complex set of efforts among people – without any planning there is chaos. Thus, an important point to be learned from the criticism is that the dilemma is to commit to a future while retaining the flexibility to notice and adjust to the real future as it arrives. In dynamic markets, strategic planning is less about gaining insight about the future, but rather an emotional rallying point that provides a roadmap on resource usage. (Brown & Eisenhardt 2002)

Most studies point out that strategic planning, as defined in the business strategy literature can and should be practiced in small businesses (Smith1998). Also, there is evidence that decision-makers of small companies should rely more on information provided by analysis and less on intuition, the use of analytical approaches should be increased, and improvements in environmental scanning and information search should be sought for (Brouthers,

Andriessen, & Nicolaes 1998). Thus, strategic plans should be treated as a rough roadmap and budgetary guideline, and not as a straitjacket that limits from adapting to the future as it unfolds (Brown & Eisenhardt 2002). Indeed, a general conclusion in literature dealing with innovation management is that strategic planning (Eisenhardt & Sull 2001), as well as the product development process itself (Thomke & Reinertsen 2001) should be no more formal than absolutely necessary. In successful cases, managers often combine limited structure such as priorities, clear responsibilities and formal meetings in some issues with extensive freedom and flexibility in others (Artto, Martinsuo, & Aalto 2001), (Eisenhardt & Sull 2001).

Based on the review on strategic management literature, the author takes the position that strategic planning is vital to small companies' long-term growth and development even in turbulent environments. In balanced development environments, product planning should employ a certain amount of formalism, be coordinated with the company's marketing and operations strategies (Krishnan & Ulrich 2001), and result in mission statements for projects and plans illustrating the timing of future product releases.

Also, the emphasis and the amount of formalism employed should change over time. During the early stages of a company's life, product strategising does not need to employ a highly formalised process, but managers should emphasise its analytical elements, such as scanning the environment, assessing strengths and weaknesses, identifying and evaluating alternative courses of action and reviewing and revising plans. The strategic planning process should then become more formal and sophisticated over the life cycle of the business, and research has shown that those entrepreneurs that are successful on the long-term do embrace this progression (Berry 2002).

Thus, the conclusion of the author is that what is necessary, even at the beginning, is the ability to explicitly recognise the important issues and thus define the scope of the company's product strategy. However, most of current management theory is founded upon large company context and cannot be applied directly in smaller companies (Jennings & Beaver 1996; Ward, Laitinen, & Fayad 2000).

2.3.2 New Product Development

In literature on new product development (NPD), managing development projects or products as portfolios can be seen as a generic approach to create the link between the company's strategy and its product strategy. In practice, this refers to systematic development project selection and prioritisation.

The objectives of portfolio management approaches are threefold: 1) to maximise the value of the development project portfolio, 2) to link it to the company's strategy (Cooper, Edgett, & Kleinschmidt 2001), and 3) to balance it on relevant dimensions. Maximising portfolio value can be supported by investment calculations and other financially based methods and scoring models that build the desired objectives into weighted criteria lists (Artto, Martinsuo, & Aalto 2001). Link to strategy reflects the alignment between projects, the strategic fit of their content and resource allocation with the intended strategy. The link can be tuned, for example, by applying strategies reviews and building strategic criteria into scoring and prioritisation models, project selection tools or go/kill models, or by setting

aside resources for different types of projects. Typical dimensions on which a company's portfolio should be balanced are short versus long term, current versus new platform(s), high versus low risk, research versus development, focus versus diversification and resource allocation between development efforts (Artto, Martinsuo, & Aalto 2001; McGrath 2000). A great variety of concepts, tools and approaches to management of new product development are indeed employed in the industry, and the most successful ones utilise the principles described above (Cooper, Edgett, & Kleinschmidt 2001).

Thus, it seems evident that management of software development has much to gain from advances in managing new product development in other fields. However, most techniques, tools and methods for new product development have been designed from the perspective of large companies with multiple business units, each with possibly several product lines.

While integrated, portfolio-based phased project screening approaches (Cooper, Edgett, & Kleinschmidt 2001) take a holistic enough perspective to support product strategy decision-making as defined in this thesis, their direct applicability to the small software product business context suffers from their design to the perspective of larger companies, and even then, not necessarily that of a software product company. Indeed, portfolio management can be considered relevant in large companies where activities are organised as distinct projects, dedicated resources to make portfolio related calculations and considerations exist, and strategy is (at least to a degree) clear and explicit (Artto, Martinsuo, & Aalto 2001). Also, literature does not yet provide insight into whether portfolio management should be considered in small firms with few projects only or projects being a secondary-type activity, or where strategy is not at all clear, environment is in constant turbulence, and little slack resources exist for coordinating the portfolio (Artto, Martinsuo, & Aalto 2001). As will be shown in section 3.1, most of these characteristics apply to small software product companies.

Thus, it is questionable whether product strategy decision-making can be supported by directly applying such techniques. Various product development models can be very useful as generic guidelines, but they fall short in several areas specific to software development to be used as such (Carmel & Becker 1995). For example, the Stage-Gate™ (Cooper, Edgett, & Kleinschmidt 2001) model adds clarity to the front end of project selection and prioritisation but does not give specific guidance on how to develop the project incrementally. In a small software product company, the emphasis of decision-making would necessarily focus more on the contents of a single product line or even a release project, rather than on deciding of a multitude of development projects concerning different product lines and their extensions.

Due to the difference in perspective, new product portfolio management principles and techniques are not directly useful to product strategy decision-making in small software product company context. However, basic NPD portfolio management principles are likely to be applicable and useful for increasing small companies' awareness of essential product strategy decision-making issues. As small companies often get only one shot at success (Mello 2002), it is especially important that these principles, for example elements from portfolio management are incorporated into the project level to promote flexibility. Also, many of the traditional new product development techniques, for example roadmapping,

are likely to be applicable to small software product businesses' with reasonable adaptation work.

2.3.3 Software Engineering

Much of the literature on software engineering is written from the perspective of large organisations doing individual projects for specific customers (Brodman & Johnson 1994; Carmel & Becker 1995; Demirörs et al. 1998; Kautz, Hansen, & Thaysen 2000; Laitinen, Fayad, & Ward 2000). Also, software engineering literature generally prefers the engineering point of view. For example, software release management has been discussed from the perspectives of its operational aspects (Bays 1999; Johnson 2002), architecture and reuse (Bosch 2002; van Ommering 2001) and requirements (Carlshamre et al. 2001; Carlshamre & Regnell 2000), but the perspective of conducting product, feature and release cycle planning in iterative development and organising for it, in other words, that of conducting product strategy decision-making, is missing.

In general, software engineering literature leaves the link to business management for business literature to handle (Rautiainen, Lassenius, & Sulonen 2002). For example, the Capability Maturity Model (Paulk et al. 1993) refers to "the business goals of decreased cost, increased quality, better schedule performance, and a continuously improving software process". The connection between the business needs and the development process has been addressed by stating that "to a particular organisation, one or more of [the mentioned 'business goals'] may be relatively more important" (Ginsberg & Quinn 1995).

While research on process improvement in small software companies is increasing (Laryd & Orci 2000; Nunes & Cunha 2000; Smith 1998; Sutton 2000) and so-called agile methodologies have recently addressed the small team perspective to software development (Beck 2000; Cockburn 2002; Schwaber & Beedle 2002), a large part of this work addresses the proper execution of the individual development projects.

Quite recently, software product management (Condon 2002) as well as release planning (Penny 2002) and management (Johnson 2002) have received more attention. From the perspective of product strategy decision-making, this work is useful as it provides concepts and definitions for many of the issues that have traditionally received little attention in software engineering literature. Still, most of the work has not at the time of writing this thesis crossed the previously mentioned border between the perspectives of business and engineering management.

Although good guidelines of what product strategy decision-making in small software product business should encompass can most likely be derived from existing literature, direct applications or frameworks to promote such ends were not found. Little context-specific support exists for small companies in the software product business that face challenges in their product strategising.

2.3.4 Conclusion

Based on the literature study in section 2.3 it seems that there is little support for product strategy decision-making available in the form of context-specific tools, techniques or

frameworks. While significant initial improvements to the software development process of small companies are often tool-based (examples can be found from (Otoya & Cerpa 1999) and (Pihlava 1996)), an analogous approach for improving product strategy decision-making seems difficult, since no tried and generally available tools of immediate usefulness to product strategy decision-making in the small software product business context were found. Also, even if such were to be found or developed[2], there still might be considerable effort in justifying the tool overhead (Fayad, Laitinen, & Ward 2000). This, combined with possible differences in perspective and emphasis of management concerns between large and small companies' and the 'not-invented-here' syndrome, suggests that deciding on *how* the more formal approaches should be used to support product strategy decision-making is likely to require a case-by-case examination.

While the author perceives that the principles and ideas behind various techniques in new product development apply to small software product companies as well, the same is more difficult for the approaches themselves. Although important product development decisions usually do get made, they are not necessarily taken in a deliberate fashion. The only guarantee is that these decisions do get eventually made, either consciously, inadvertently or through inaction. Increasing awareness is said to be the first step before systematic and constructive improvement action can be expected (McCarthy & McCarthy 2002). Thus, the author sees increasing *awareness* of the issues essential to product strategy decision-making in small software product companies as a pre-requisite for identifying, adapting and using specific tools and techniques for product strategising.

Furthermore, recent work on strategic planning suggests that starting with increasing awareness may prove useful for small companies because awareness promotes *coherence*. Here, "coherence" means recognising and dealing with the present using actions that make inherently sense to the participants, rather than focusing too much on the future and or the company wants to be (Lissack & Roos 2001). The author also interprets the suggestion that strategy development and enactment should be managed using a dynamic set of relatively simple rules (Eisenhardt & Sull 2001) as emphasising coherence.

Thus, instead of further developing methods to facilitating product strategy decision-making, the author now perceives an explicit discussion of the underlying strategic issues most useful. A logical step with respect to improving small software product companies' practices in formulating and enacting product strategy is that they *must first be supported in identifying what issues must be accounted for in the strategic management of their product development*. As an answer to research question (4), the author proposes that a breakdown of the most important product strategy decisions from the perspective of small software product companies would seem useful for the purposes of supporting product strategy decision-making in such companies.

This kind of a breakdown would provide a context reminding the personnel responsible for product strategy decision-making of the issues necessary for producing a quality product. A framework of important product strategy decisions could be applied, for example:

---

[2] For an example of a roadmapping approach to support product strategy decision-making in small software product companies, see (Vähäniitty, Lassenius, & Rautiainen 2002)

- In a checklist-type manner for process assessment and improvement purposes
- Identifying how the company's business environment and desired way of conducting business should, through conscious product strategy-level decision-making, be reflected in its new product development process

Thus, the construction of a framework outlining key product strategy decisions for small software product business is attempted.

2.4 Summary

This chapter examined formulating and enacting product strategies in small software businesses based on literature and the experiences of the author with the intention of suggesting that practitioners in such companies face considerable challenges and fail to benefit from existing literature.

First, an outline of the decisions through which product strategy is made operational was presented and the applicability of this outline for organising software product development in small software product businesses was evaluated.

Second, common problems in new product development resulting from inadequate management are discussed, and the initial state-of-practice in four of the companies we have worked with was examined. Because of the decision-making perspective presented in section 2.1, it was proposed that the state-of-practice of the companies' product development reflects their product strategy decision-making, and based on this, the general need for improvement in product strategy matters of small software product businesses was evaluated.

Third, this chapter contained a review of strategic planning, new product development management and software engineering literature to examine what has been written about product strategy issues in general, and specifically, in software product development. Strategic planning, NPD management and software engineering literature have been chosen as the target for this review because they address formulating and enacting product strategy from different perspectives.

As a conclusion, it was proposed that a breakdown of the most important product strategy decisions from the perspective of a small software product companies would seem useful for the purposes of supporting product strategy decision-making in small software product businesses.

3 Constructing a Framework of Key Product Strategy Decisions – a Theoretical Approach

How can a framework of important product strategy issues to the context of small software product companies be constructed if literature so far has done little to discuss these issues explicitly? To proceed, the author proposes that the issues important for the context can be derived from two sources: a closer examination of the characteristics attributed to small software product companies and their management implications (see section 3.1), and the issues of general importance in new product development.

Although many of the characteristics of small software product companies are more causes for concern rather than explicit decision areas, they can be used for deriving the basic management issues relevant to the context. However, restricting the framework to these issues only may not be enough, because of possible common denominators in managing software and new product development in general, regardless of company size and degree of productisation. If such denominators exist (and likely so), they for the sake of comprehensiveness must also be accounted for. Towards this end, existing knowledge on what issues are important in product strategy level decision-making is compared and possibly added to the minimum scope defined in the previous step.

3.1 Characteristics of Small Software Product Businesses

In this section, general characteristics assumed common for small software product businesses are presented based on software engineering and NPD management literature as well as the author's previous experience with these kinds of companies.

Software product refers to a packaged solution that meets generic computing requirements and includes, for example, enterprise solutions (for example, inventory-control or accounting systems), software development tools, operating systems, utilities and personal-computing tools (Carmel & Becker 1995). The term *product* has been used within the software community to describe any system which is delivered to the customer, but a simplified categorisation of software products divides the spectrum into the two broad types of *packaged* or *commercial-off-the-shelf* software, sold at a set price, and direct sales *enterprise software*, whose price and support terms may be negotiated with the vendor's representatives (Condon 2002). While both packaged and enterprise software are considered product business, the number of customers ("millions" versus "thousands", respectively (Hoch et al. 2002)) and consequently, the amount of customer specific attention by the firm differs greatly.

Developing one-time solutions for specific customers (commonly used terms are 'bespoke software' or 'software services') consists, in contrast, of development and operations services that are provided to clients on a project basis. These can be for example, custom software development, and implementation or systems integration services (Nambisan 2001). Although the distinction is in practice seldom clear-cut (Condon 2002), there are

considerable differences between the ends of the product/service business axis and a number of factors having important managerial implications.

Focus is sought examining the characteristics along the axes of Figure 1. The discussed dimensions are the implications to management from being in the software product business and being a small company.

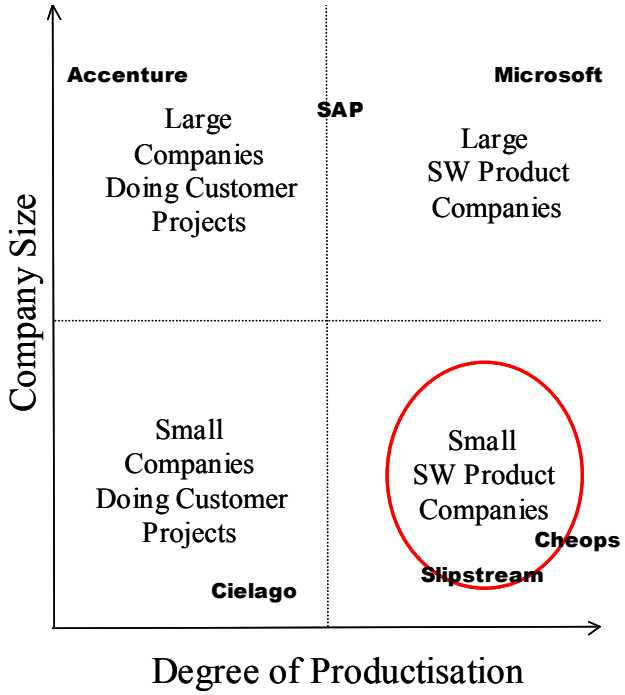

Figure 1 Small software product businesses and the dimensions examined (axes not drawn to scale)

Using a simple interpretation, we can place real-world companies into the upper half of Figure 1: Accenture (upper-left), SAP (up-and-middle; this position along the productisation axis is often referred to as 'enterprise software' (Condon 2002)) and Microsoft (upper-right, 'packaged software' (Condon 2002)). Examples of respective well-known small companies are of course scarce, but taking use of the case companies described in sections 4.2.1-4.2.2, we can place Cielago in the lower-left (with a desire and movement towards the lower right) and Slipstream and Cheops in the lower right. In reality the situation is often complex, especially with large companies having strong service components as part of their total offering. Although it can be argued that without these services they would not be in the business at all, such companies can usually be categorised as companies in the product business by observing where the profits lie.

Still, the main source of profits is only the top of the iceberg when it comes to discussing the differences between software companies in the service or product business. Sections 3.1.1 and 3.1.2 list characteristics attributed to companies in the software product businesses and small software companies (respectively) both in literature and according to the author's experience, and thus provides a more in-depth definition of the focus of this thesis. The results of this analysis are summarised in section 3.1.3.

3.1.1 Characteristics of Product Business and Managerial Implications

Amount of customer-specific effort is (typically) small. Product business denotes that after the initial development costs, manufacturing and distributing additional copies of the software component of the product is relatively cheap (Hoch et al. 2002). Also, the amount of customer-specific development work, including integration, is not significant compared to the total cost of developing the product.[3] This simplifies some aspects of product strategy decision-making, for example, the company has less to worry with respect to the distribution channels and even less about its own "suppliers".

Financial and technical risks are carried out by the company. While the amount of customer-specific effort tends to be small, the risk associated with the cost of developing new product versions and upgrades is usually beared by the developing organisation (Sawyer 2000). Also, billing the customer before releasing the product to the market is usually not feasible. In practice small companies with limited financial resources often have to share risk by compromising freedom to develop the product based on anticipated market needs with satisfying the wants of the initial reference customers.

In the experience of the author, a common way to do this is undertaking customer-specific development work under the hopes of integrating it later as part of the offering for all customers (Vähäniitty, Lassenius, & Rautiainen 2002). However, sometimes this can be difficult because the business and development priorities for doing product- or service-based business are considerably different (Nambisan 2001). Thus, product strategy decision-making should address how long-term business potential is compromised for short-term cash for example through providing complementary services.

Potential for higher marginal returns on scale. A fixed cost structure allows product businesses higher marginal returns on scale (Hoch et al. 2002), and in general, product companies move from developing the product on the basis of needs specific to a handful of important customers ('customer-driven') to addressing the needs of the market as a whole ('market-driven') as the product matures (Nambisan 2001).

A company in the software product business is more likely to require a global rather than a regional market presence and many customers instead of just a few, and market share is often the most important indicator of business success (Hoch et al. 2002). However, as a product becomes more widely used, cost and complexity of customer support and demand for changes and corrections increase and can become extremely expensive (Fayad, Laitinen, & Ward 2000). To take advantage of the potential for higher marginal returns on scale, the company must be able to deal with large volumes of shipped products (especially when compared to the size of the company) through effective distribution of maintenance and feature updates and release management.

Competitiveness and new product versions. Software products typically evolve in releases, with each release including new and improved functionality intended for keeping

[3] Although in some small companies, for example in start-ups, the first reference customers must often be paid more attention, the intent of such companies towards product business is the determining factor for considering them as product businesses in this study. This is justified, because the relative amount of necessary customer-specific activity diminishes as products mature (Moore 1991).

the vendor ahead of the competitors (Regnell, Beremark, & Eklundh 1998). While companies in the product business retain the intellectual property rights to the offering they market even after delivery, obtaining strong legal protection to prevent competitors from copying and imitating is in most cases not possible (Nambisan 2001). Because of the low financial and technological entry barriers (Hoch et al. 2002), the best way to maintain competitiveness is through rapid innovation and bringing out a continuous stream of upgrades to maintain product uniqueness (Nambisan 2001).

A central activity in managing a software product business is deciding when the next releases of the products should be made generally available (release *types*) and what feature enhancements they should contain as to maximise future revenue (*roles*) (Penny 2002). Determining a feasible combination of dates (*timing*), features (*contents*) and resourcing for the next release of a software product is termed here as the *strategic perspective* to release management.

Role of product complementarity. In the product business, successful new products in most cases support or complement the functionality of existing and established products (Nambisan 2001). Product strategy decision-making must take into account the long-term competitive implications of which application area or computing environment they specialise in and undertaking new development projects should be controlled keeping this in mind.

Relative relevance of management areas. Product strategy and marketing & sales are the most critical management areas for companies in the software product business, while superior human resource management and software engineering are most important management areas in the service business (Hoch et al. 2002).

While companies should embrace analytical and formal elements of strategic planning, the other management areas should certainly not be neglected, as success in them is essential in enacting the strategy. Because of the multiple and often cross-functional roles and responsibilities of the key personnel in small companies, product strategy decision-makers should consider integrating the other management areas into the product strategy process. In practice, this may simply mean that there is an explicit strategic vision of the product that is traceable across the management areas. In sales and marketing, such traceability means addressing issues such as distribution channels, pricing and customer segments in compliance with the vision. In human resources traceability means organising the product development as well as the rest of the organisation to support the vision, and in software engineering, it means that the shape of the development process should reflect the strategic ambitions for the product.

Feature elicitation and valuation are based on dynamic criteria and in-house domain experts' judgment. Needs for the requirements engineering process for product development differ from those of developing bespoke software because of schedule constraints and stakeholders' roles (Sawyer 2000). The needs of the market may change rapidly, and no discrete set of users exists for requirements elicitation needs (Regnell, Beremark, & Eklundh 1998). The responsibility of deciding on a competitive set of requirements to implement based on conclusions drawn about the markets ultimately rests with the developers, in-house domain experts and the management (Kamsties, Hörmann,

& Schilch 2002). Also, most software development process models implicitly assume that user requirements are obtainable and that the users are available to provide feedback and are driven more by technical and behavioural rather than market requirements (Carmel & Becker 1995).

Lack of marketing orientation is a frequently cited reason for new product failures. Thus, a top priority for product strategy decision-making is to address and support communication between product development and the market by identifying, finding and involving (the different types of) "remote customer(s)" and creating the marketing interface (Carmel & Becker 1995). The representative of the "remote customer" must be involved with the stakeholders from the company, in other words, sales and marketing, support, engineering and senior management in jointly deciding about the releases' contents. To facilitate commitment and prevent over-reactiveness, or at the other end, paralysis through analysis in the rapidly changing environment, the requirements engineering process should facilitate and allow incremental adjustment of the set of requirements during development.

Complexity of market segmentation and product differentiation. To survive in the long run, the software product must be differentiated on a number of attributes, such as price, features, performance, conformance, reliability, style, services and image. Software products must address other questions concerning market segmentation as well: usage rates, customer and/or user capabilities, technology, preferences and demographics. (Carmel & Becker 1995)

Product strategy decision-making should address differentiating the product and segmenting the market, and the result of such analysis should be reflected in the product development process, for example through (possibly dynamic) criteria used in feature valuation, and requirements engineering. However, transferring the complexity of the markets to the process for managing end-user or market requirements is not feasible or even desirable (Condon 2002). In a small software product company, it is the job of the product strategy decision-makers to provide a framework with good-enough level of detail, complexity and ceremony for deciding about release contents.

Pressures from time-to-market and increasingly shortened release cycles. Software is not a mature product area. The market needs, demand levels and the technology basis are still rapidly changing, and software cannot be marketed in a "steady" state (Condon 2002). In a competitive environment overall calendar time from concept to release must be minimised, and there typically is a major pressure on time-to-market. Windows of opportunity may evaporate and customers may switch to competing suppliers if products or new features are delivered late. Often, marketing departments force development into a very rapid pace by marketing-induced, pre-emptive announcements, and during the last decade, release cycles have shortened from years to several months or even weeks. This has been accelerated by rapid obsolescence and the fact that revenues are associated with delivering product updates (Carmel & Becker 1995).

While practitioners struggle to strike the right balance between excessive intervention and inadequate oversight in new product development, the issue of timing and frequency of project monitoring and intervention has traditionally been addressed only to a limited extent in academic literature (Krishnan & Ulrich2001). A company can effectively control

its time-to-market pressures by choosing how it paces its development, that is, when and how often product releases as well as intermediate products and handoffs are made. Thus, development pacing, determining the general shape of the product realisation process (i.e., waterfall, iterative & incremental) and the software development methodologies and project oversight and tracking methods used are of major concern in product strategy decision-making.

Iterative and incremental product development process recommended. The essential complexity and invisibility of software promote the need for a highly iterative and incremental development process with frequent feedback loops between and within the stages of development (Carmel & Becker 1995; Fayad, Laitinen, & Ward 2000; Marco & MacCormack 1997) to achieve visibility to the development process, increase flexibility and to promote controllability in product development. Failure to induce flexibility to the development process is a common reason for market failure in the product business (Nambisan 2001).

There are a number of attributes that further characterise an iterative and incremental development process, such as the length of development projects and number and type of iterations, feedback loops and milestones. Thus, product strategy decision-making must oversee that the requirements for business success are reflected through these attributes in the way products are being developed.

Simultaneous development of both technologies and products may be necessary. Remaining competitive often requires developing both the products and their underlying technologies simultaneously (Cusumano & Selby 1995; Cusumano & Yoffie 1998), increasing the risks involved (McGrath 2000). Managers should come to expect the added amount of coordination resulting from this, and planning of future releases should recognise and deal with the separated life cycles of the products and the technologies they are based on. Thus, important technological decisions must be addressed in product strategy decision-making.

Higher initial investments in the design of the product architecture. While architecture and technology choices often persist well beyond the development projects in which they are originally made, product companies need to emphasise long-term product support and can not afford to end up with limited design flexibility (Iansiti & MacCormack 1997). Modular product architecture and cross-platform compatibility help here but require complex designs and higher levels of knowledge abstraction, which in turn may reduce short-term efficiency (Nambisan 2001). Often, survival pressures make "hacking" the product to satisfy the needs of initial customers a tempting idea, usually to the detriment of the product architecture. In a small company, the top management should be involved with choosing the technologies and making architectural decisions.

Motivation for process improvement. In the product business it can be more difficult to motivate process improvement because no explicit customer demand on process quality exists (Nambisan 2001). On the surface, customers' are not interested in conformance to software development standards, but cost, utility and interoperability with other existing software (Laitinen, Fayad, & Ward 2000). Improving the processes and practices used in

product strategy decision-making could prove difficult to motivate as well. In the case of small companies, introducing new approaches and techniques, for example for planning of future product releases should not involve heavy documentation or reporting if more-than-moderate resistance is to be avoided, and it should be possible to readily demonstrate the potential value of any improvement work.

Role of quality assurance. Software development has become notorious as an industrial process producing artefacts with many defects, in part due to its inherent complexity. While traditional new product development literature pays little explicit attention to quality assurance elements of product development process, in software product development these should be made very explicit (Carmel & Becker 1995). Also, in software product business, the role of quality assurance is often more on validation than verification (Kaner et al. 2002), although both should be emphasised (Pyhäjärvi, Rautiainen, & Itkonen 2003).

Product strategy decision-making should promote releasing near-defect free products (Carmel & Becker 1995), but the meaning of 'defect-free' depends on the context. From the perspective of product strategy decision-making, the appliance of quality assurance should be based on the product and business risks facing the company.

3.1.2 Characteristics of Small Companies and Managerial Implications

Different definitions exist for the term 'small company'. (Megginson et al. 1997) define small companies having less than 100 employees, with very small companies having less than 20 employees, while (Brodman & Johnson 1994) consider small companies those with less than 500 employees. The regulations of the European Union consider the upper limit of small companies 50 employees, which is also the definition used in this thesis. While one of the case companies is slightly larger than this but judging by the size of its product development organisation, it could be considered a small company. Thus, we refer to all of the case companies as 'small' for the sake of simplicity.

Again, significant differences exist between small and large software companies from the perspective of managing product development.

Potential strengths from low cost-of-communication. In smaller organisations, the entire software development organisation is often a handful of people working closely together, often as a by-product of the company's history (Kelly & Culleton 1999). In our experience, communicating across the organisation requires relatively little effort, which may contribute towards one or more of the following: frequent and extensive communication, new ways of working easier to deploy, and individuals having an opportunity to affect the way work is done. Also, the small size of the organisation makes it inherently more flexible and enables fast reaction time (Nunes & Cunha 2000). Shorter distance between decision-makers and implementers grants visibility (Ward, Laitinen, & Fayad 2000), and the synergy of people with diverse skills working together without management intermediation is claimed to give small teams their "edge" (Laitinen, Fayad, & Ward 2000). However, the author believes that none of these qualities can be taken for granted just because the organisation is small.

Emphasised role of senior management. In small businesses strategic management is enacted in a highly personalised manner and is strongly influenced by the personality, disposition, experience and ability of the entrepreneur/owner-manager (Jennings & Beaver 1996). Also, according to the experience of the author, senior management in small companies is often closely involved in defining not only the timing but the contents of individual product releases as well, and most of the business know-how is in practice held by them. Because hiring outside help is often considered prohibitively expensive, top management or investors have a significant role in providing this kind of expertise in small companies.

More roles than people. In small organisations, there are commonly more roles than people (Otoya & Cerpa 1999). While this means that several important roles have been rolled into one, these roles are not necessarily the same ones that are needed in larger organisations (Laitinen, Fayad, & Ward 2000). In general, roles and responsibilities are not as clear-cut as in larger organisations and, depending on the case, may be imprecisely defined as well (Laryd & Orci 2000; Otoya & Cerpa 1999). While roles are only the carriers of responsibilities, joint roles and imprecise role definition has been known to make for example process improvement more difficult (Laryd & Orci 2000). Also, in the companies we have worked with, roles and responsibilities are often intertwined to resource allocation because the organisational units act as game pieces for resource allocation decisions as well.

Low cost-of-communication, imprecise or unclear role definitions, emphasised role of senior management and consequent low organisational hierarchy all promote the following phenomena from the perspective of product strategy decision-making: important decisions may be done implicitly, by the wrong people, with incomplete information, or with inappropriate timing. Examples include re-prioritising or creating new features or development tasks without linking this to the intended business implications or schedules of upcoming releases, or deciding (or not deciding) on product features and making assumptions about their market value without involving the actual stakeholders. In both of these examples, unclear role definition and ease of communication cause important 'steps' from the perspective of change management to be omitted. In the case of small companies where the same key people possess multiple different roles and participate in product development on many levels, keeping the current role and its responsibilities in mind helps prevent these kinds of problems from happening. This subtle phenomenon of 'changing hats on the fly' can be a great source of flexibility but also work to destroy coherence.

Individuals' skills and competences. The organisational culture is directly shaped by the founders' personal values. This often means that people's abilities and experiences guide how the ways of working are selected, not vice versa (Schein 1999). Also, process quality is often considered less important because the individuals in the company are considered to possess high personal skill level and competence (Grünbacher 1997).

To maintain the innovative and flexible atmosphere typically attributed to small companies (Megginson et al. 1997), product strategy decision-making should exert attention to supporting team collaboration, skills and training and motivation and shared vision (McCarthy & McCarthy 2002). Also, while skilled individuals are often attracted to small companies and start-ups, it is by definition impossible for every small company to have

"employed the best people" as small company managers and personnel sometimes, in the experience of the author, like to put it.

Pressures to secure financing. To secure steady financing, small companies may have to tender for contracts in areas where they have little or no domain expertise (Moses 2000). Also, as small companies may lack crucial business know-how or experience, this may be difficult to realise on their own.

Pressures to secure financing must be balanced with considering the long-term implications of undertaking the opportunities that present themselves. This emphasises the need for a mechanism to remain coherent under pressure, and must be realised in considering what the company sells at a given moment in the context of priorities set by company strategy.

Local area of operations. In most cases, the area of operations for small companies is local, although the markets necessarily are not. Apart from the initial reference customers and direct sales, successful small companies must often employ indirect sales channels to reach their markets.

The dilemma of local area of operations but required global presence poses many important issues to product strategy decision-making. Partnering is crucial for business success (Hoch et al. 2002), and indirect sales channels must be motivated to distribute the product, with the best motivator being an increase in the channel's bottom line. Thus, in product and feature planning it should be kept in mind that in addition to the end customers, products must also be marketable to the channel. A number of channel management issues exist, for example sales promotions, training and incentives to the partners. Besides indirect sales, direct sales keeps playing an important role. In many software firms employing indirect sales there are sales teams that account for a significant portion of revenue, and listening to the 'voice of the customer' heard through these teams must be balanced with that coming from the indirect channels. (Condon 2002)

Small companies (appear to) have less need for formal management procedures. While the management overhead of smaller companies and development projects is indeed smaller (Brooks, Jr. 1995; Russ & McGregor 2000), they may have a larger number of external dependencies increasing the need for process. For example, the development may have a closer association with its customers requiring more interaction from the team members, and small teams more often include several people with only part-time participation requiring more coordination and interaction to effectively utilise their skills. (Russ & McGregor 2000)

In the case of small teams, having inadequate coordination and control mechanisms in place has been observed to be the cause of several generic pitfalls (Russ & McGregor 2000). First, favourite activities receive more attention than others, with the team doing an outstanding job with whichever activities its members care most about and does little or nothing about other activities, possibly even ignoring that such exist, regardless of their importance. Second, in a rapidly changing domain, a part of the work done at the end of a phase is always out-of-date, and the team may get stuck as it tries to compensate for this before moving on. Third, the team may make little progress because it is unsure about what to do next. Lacking a flow of activities, or process to guide it, time is wasted.

Although these pitfalls were identified for software development work by (Russ & McGregor 2000), they in the opinion of the author are very applicable to product strategy decision-making as well. Just as a process or development pacing set forth by decisions on the strategic level is necessary for software developers, the people responsible for product strategy should also have some means to avoid analogous pitfalls in their work. Structure can be sought for example from regular meetings of common format, techniques and tools for strategising, and checklists of important decision areas.

Role of quality assurance. While the importance of software testing is recognised in small companies, the limited resources cause restraints on the use of traditional testing approaches based on a separate testing organisation. (Pyhäjärvi, Rautiainen, & Itkonen 2003)

Process improvement – potential strengths. Smaller, younger companies that are able to hire from the ground up can design a development process that reflects a company-wide collaboration and is fast-paced and productive provided that mature and experienced managers are available (Ward, Laitinen, & Fayad 2000). In a small company it may be easier to change the corporate culture and steer the organisation toward improvement goals because of less inertia and bureaucracy. Also, development projects in small businesses tend to be shorter, which means more frequent feedback loops and can be advantageous for introducing new initiatives as this is easiest in the inception phase of the project (Brodman & Johnson 1994; Russ & McGregor 2000).

Potential for rapid changes in the ways of working exists and a steep learning curve from any improvement activity is possible. Also, significant initial improvements to the software development process of small companies can be made by introducing new development tools (Otoya & Cerpa 1999; Pihlava 1996), provided that effort can be spent in justifying the tool overhead (Fayad, Laitinen, & Ward 2000) and cost. Infrastructure and tool decisions are of major concern, because in small companies changes to these can have company-wide significance both in terms of development process change and acquisition cost of the tool. In addition, infrastructure and tool decisions and are often coupled to the technologies employed in the product as well.

Process improvement – potential pitfalls. While several studies conclude that small companies want to improve their process and product quality, there are many organisational, cultural, financial and technical problems (Nunes & Cunha 2000). First, the need for long-term investments in improvement programs is often not understood. Individuals' 'heroic feats' play an important part and need for accompanying processes is not seen (Grünbacher 1997). While the culture in small organisations can often be justifiably characterised as creative, dynamic and innovative, such organisations easily view software process improvement as the antithesis of these qualities, leading to bureaucracy and paperwork that restrict the freedom of individuals (Brodman & Johnson 1994; Grünbacher 1997; Kelly & Culleton 1999). Existing reference models from software engineering literature are often not applicable, mostly due to the characteristics inherent for small companies. The most important factors are joint roles (Grünbacher 1997; Nunes & Cunha 2000; Otoya & Cerpa 1999), availability of resources for improvement activities (Moses 2000; Nunes & Cunha 2000), perceived expensiveness of implementing a process improvement programme especially when compared to other business pursuits (Brodman &

Johnson 1994; Cater-Steel 2001; Grünbacher 1997; Kelly & Culleton 1999; Nunes & Cunha 2000), lack of software process improvement know-how (Cater-Steel 2001). In the case of software product development, the fact that existing models focus on the software process for developing bespoke software (Rautiainen, Lassenius, & Sulonen 2002) can make for example the key practices of such models inappropriate (Otoya & Cerpa 1999) or at least not directly applicable. Besides these problems, most process improvement models fail to benefit from the potential strengths of small organisations (Nunes & Cunha 2000). Also, developers expect to be involved in deciding about the process. While in large organisations it is not unusual for decisions about the ways of working to be made outside those involved, personnel in small organisations expect to influence decisions that affect the way they work.

The potential pitfalls regarding software process improvement have also several implications to product development decision-making on the strategic level. First, the need for adding formal and analytical elements to product strategy level decision-making may not be understood, and because of the inappropriateness of traditional software engineering literature to software process improvement, there is likely to be little material characterising or supporting product strategy decision-making in small companies. Second, even if management is aware of such issues and wants to steer the development process to better reflect some business needs, imposing process solutions to the development organisation may prove challenging (Kelly & Culleton 1999). Finally, tools and techniques used in the product strategy process should not rely heavily on reporting procedures, which may prevent approaches employed by larger companies from being scalable to small companies as such.

Start-up characteristics. Finally, it must be noted that while most small companies are not start-ups, the contrary is often quite true, and thus characteristics specific to start-ups are discussed here. While all software companies must address the issues of time-to-market versus cost versus quality and are subject to changing technological and economic environments, start-ups face these issues usually with an 'extreme' twist (Sutton 2000). While not all start-ups are expected to grow fast, they usually operate in an extremely turbulent business environment featuring fierce competition. Such companies have often very limited resources, hold development speed and time-to-market as their primary concern and may consciously avoid development for reuse (Fayad, Laitinen, & Ward 2000). In the case of companies aiming for rapid growth, the models used in process improvement as well as the process improvement itself should scale up (Cusumano & Yoffie 1998; Laryd & Orci 2000), and advanced levels of process maturity may be out of place (Sutton 2000). Also, while introducing time-paced development at a very early stage in the life cycle of a business can be problematic (Brown & Eisenhardt 2002), experiences from the software product business suggest that short development cycles are especially important in the start-up phase to get early user feedback, establish market position and keep the technology moving forward (Cusumano & Yoffie 1998). Small established companies usually have fewer internal communication and coordination problems and greater flexibility due to lower organisational inertia, a foundation of established products, partners and customers, and possibly a shared history and vision (Sutton 2000).

3.1.3 Summary

The results of the analysis from sections 3.1.1 and 3.1.2 are summarised in Table 5 and Table 6 below.

Table 5 Software product company characteristics, implications to product strategy decision-making and related management issues

Characteristic(s)	**Managerial implications**	**Important issues for product strategy decision-making**
Amount of customer-specific effort (typically) small.	First reference customers often require more attention; Manufacturing and distributing of additional copies can be negligible	Amount, type and implications of customer-specific effort Supply chain (typically) simplified to distribution channels
Financial and technical risks are carried out by the company	Small companies forced to compromise freedom of development and risk sharing	Amount, type and implications of customer-specific effort
Potential for higher marginal returns on scale	Must be able to deal with a large volume of shipped products	Distribution channels Sales and marketing Release management (operational perspective; includes the release process and configuration management)
Competitiveness and new product versions	Competitiveness through rapid innovation and a continuous stream of upgrades	Release management (strategic perspective; includes release contents, roles, types and timing) Release management (operational perspective)
Role of product complementarity	Long-term competitive implications must be considered	
Relative relevance of management areas: 1) strategy, 2) sales & marketing, 3) human resources 4) development process	Ad hoc strategic management inadequate; Other management areas should be integrated in the strategy process due to the multiple and often cross-functional roles and of the key people to gain holistic perspective	Product strategy Release management (strategic and operational perspective) Organisation, roles and responsibilities Requirements engineering Project management
Feature elicitation and valuation are based on dynamic criteria and in-house domain experts' judgment.	Communication between product development and the market must be paid special attention Flexible requirements engineering process and release scope setting required	Organisation, roles and responsibilities Requirements engineering Change management process
Complexity of market segmentation and product differentiation	Seek balance in process complexity	Requirements engineering Change management process
Pressures from time-to-market and increasingly shortened release cycles.	Timing and frequency of project monitoring and control important	General shape of the product realisation process (feedback loops), Project management
Iterative and incremental product development process recommended.	Address and guide the selection and shaping of the development model	
May be necessary to develop both technologies and products simultaneously	Plan life-cycles of the products and the technologies separately	Technological and architectural decisions
Higher initial investments in the design of the product architecture	Knowledge abstraction and technology Integration	
Motivation for process improvement	All improvement activities must be perceived immediately as useful	Team collaboration
Role of quality assurance	Should be based on product and business risks	Approach to quality assurance

Table 6 Small software company characteristics, implications to product strategy decision-making and related management issues

Characteristic(s)	**Derived managerial implications**	**Important issues for product strategy decision-making**
Potential strengths from low cost-of-communication, Emphasised role of senior management, More roles than people	Decision-making inherently unstructured: important decisions may be done implicitly, by wrong people, with incomplete information, or with inappropriate timing	Organisation, roles and responsibilities
Individuals' skills and competences	Promote awareness of team issues and Collaboration	Team collaboration, Staffing
Pressures to secure financing	Emphasised role of coherence and strategic planning	Product strategy Release management (strategic perspective) Sales and marketing, Servicing and deployment, Amount and type of customer-specific effort
Local area of operations	Channel management and identifying the voice of the customer	Distribution channels Sales and marketing Servicing and deployment Requirements engineering
Small companies and projects (appear to) have less need for formal management procedures	...but appearances may be deceiving and 'less' can still be quite a lot	General shape of the development process (feedback loops) and project management
Role of quality assurance	Integrating testing as part of the development efforts in small companies may be difficult	Approach to quality assurance
Process improvement – potential strengths	Potential for a steep learning curve	Team collaboration, Organisation, roles and responsibilities Development tools and infrastructure
Process improvement – potential pitfalls	Need for formalising product strategy level decision-making may not be understood; Having a small group impose process solutions to the development organisation may turn out to prove challenging; Tools and techniques used in product strategy decision-making should not rely heavily on reporting procedures	
Start-up characteristics	Advantages of established companies in product strategising; Greater flexibility due to fewer internal communication and coordination problems; A foundation of established products, partners and customers; Potential for a shared history and vision	(Affects the relative importance of other management issues)

Finally, the reliability and validity of this kind of analysis must be addressed. Obviously, depending on the references and arguments used to point out the characteristics attributed to small software product businesses, the exact placing of the management areas in the right-hand columns of Table 5 and Table 6 is very likely to vary and is subject to argument. However, the resulting set of management issues gained in this way (listed in Table 7) is relatively independent of the exact arguments used, and thus less subject to change.

A more important concern from the perspective of validity is the comprehensiveness of the resulting framework, which is addressed in the subsequent construction steps where depth and contrast is sought by comparing this framework to existing models on what

issues are important in product development decision-making. For some characteristics, this is directly visible from the tables as well in the form of joint 'important product strategy decision-making issues'.

Another aspect worthy of note is that the characteristics and their implications are clearly not independent, meaning that changes to one are likely to affect others, too. For example, if a company does not have to compromise its strategic ambitions in order to secure its financing, challenges in all areas where the amount and type of attention devoted to specific customers is of importance are less severe. While this does not undermine the usefulness of the kind of framework under construction, different companies have different management priorities, and thus, not every company needs to pay as much attention to all of the issues listed in Table 5 and Table 6.

3.2 Constructing the Theoretical Framework

At this point, the minimum scope for the framework of product strategy decisions for small software product companies can be established. Referring back to Table 5 and Table 6, the key product strategy issues of small software product companies have been collected into Table 7.

Table 7 Key product strategy decisions in small software product companies

Decisions
Sales and marketing
Development tools and infrastructure
Amount, type and implications of customer-specific effort
Distribution channels
Release management (operational and strategic perspectives)
Requirements engineering
Project management
General shape of the development process (i.e. projects, increments, milestones etc.) and feedback loops
Technological and architectural decisions
Quality strategy (what kind of testing is conducted and how)
Organisation, roles and responsibilities
[Investments in] team collaboration

Many of the elements of product strategy decision-making listed in Table 7 are interrelated. For example, project management is difficult without a good understanding of the effort left in the form of understood requirements, and it is questionable whether having milestones or increments to pace a release project should be considered as a part of 'project management' or the development process itself.

Thus, even though the issues are not really independent, it makes sense to restructure, in other words, group, rename and create hierarchy to improve the usability and understandability of the resulting framework. Because maintaining the issues' "orthogonality" is not possible, a more important factor is to keep in mind that the grouping should make sense from the perspective of the intended audience of the model, and that the framework should be reasonably comprehensive.

This restructuring and checking for comprehensiveness is done by contrasting the list in Table 7 to the list of key decisions in NPD development by (Krishnan & Ulrich 2001)

from Table 1 and Table 2. The goal is to reach a simple framework having as independent decision areas as possible with a comprehensive but minimal set of elements relevant from the perspective of a small company in the software product business.

The framework resulting from combining key new product development decisions by (Krishnan & Ulrich 2001) and the managerial implications of being a small software product business is shown in Table 8 below. A detailed explanation of how the key decision areas in Table 8 were come to is given in Appendix C.

Table 8 Key product strategy decision areas in small software product companies

Decision area	**Contents**
Organisation (*by whom, and where?*)	Organisational model, Roles and responsibilities, Team staffing, Team physical arrangement and location, Investments in team collaboration
Portfolio management (*what and when?*)	Sales and marketing (incl. distribution channels), Servicing and deployment Release management (incl. operational perspective: release process and configuration management + strategic perspective: release contents, roles, types and timing)
Requirements (*what and when, specifically?*)	Elicitation, Specification, Allocation, Change management
Development strategy (*how?*)	Development model(s) (incl. type of development process / pacing & pacing, progress tracking and control and communication mechanisms among team members)
Technology (*by leveraging which technologies?*)	Product architecture and employed technologies, Development infrastructure
Quality strategy (*delivered with what emphasis?*)	Decisions on what kind of testing is conducted and why, Project performance measurement

Table 8 also proposes the major questions that answer how product strategy gets enacted through these decision areas. Portfolio management answers 'what and when', with requirements engineering specifying these further. Organisation answers to the question 'by whom and where', and development strategy 'how'. Technology specifies the special leverage the company uses in fulfilling these objectives, and quality strategy determines the emphasis and connotations associated with delivering the promise to the markets.

4 Refining and Evaluating the Framework of Key Product Strategy Decisions – the Empirical Approach

This chapter presents the design and results of the empirical part of this study.

Section 4.1 describes the design of the empirical part, in other words, the interviews, their goal, the version of the framework used and the approach for refining the framework based on the interview data. Section 4.2 presents the detailed case descriptions created with the help of the framework to characterise the companies' practices. It also contains the observed problems and challenges, suggestions made by the researchers involved, the initial feedback received from the companies and the results over a follow-up period from providing the suggestions.

4.1 Research Design

The goals of the empirical approach are twofold. First, we find out how product strategy is formulated and enacted in three small Finnish software product businesses by interviewing product development personnel from each company. Then, these practices are described using the framework and improvement suggestions are presented. The value of the framework in supporting small software product businesses is evaluated by observing the action taken in the companies over time. The second goal is to further refine and add detail to the framework being constructed.

The empirical part aims at expanding and generalising a theoretical framework. A contemporary phenomenon, planning of new product development, is investigated within its context, with the boundaries between the phenomenon and the context not clearly evident. Also, data collection is guided by prior theoretical propositions, namely, the version of the framework used. Based on these characteristics, the research approach of this part of the study can be characterised as constructive action research using a case study approach with multiple cases (Yin 1994).

4.1.1 Data Collection

A round of semi-structured interviews was conducted at three companies, Slipstream, Cielago and Cheops, to describe and evaluate their current practices in the area addressed by the concept of strategic release management (Rautiainen et al. 2002) (see section 4.1.2 for details).

For convenience and accessibility reasons, the case companies were selected from the companies participating in the SEMS research project. All case company names appearing in this thesis are pseudonyms, and some details such as numbers and titles have been slightly altered to for the sake of confidentiality.

In these initial interviews the following people were involved. From Slipstream, the senior product manager was interviewed in a three-hour session. From Cielago, the head of R&D, the head of the software team and the person responsible for customer delivery were simultaneously present in a single two-hour interview session. From Cheops, a total of 8 people were interviewed one at a time using a total of 10 hours for the purposes of this and other studies in the SEMS research project. The interviewed personnel include the head of R&D, the two product managers, a development project manager, a chief designer and two programmers. In these interviews, an additional set of questions focusing more on the implementation level activities (out of scope for this thesis) was also gone through. This material was available for the purposes of this study as well and was used to triangulate the results of the main interview with the head of R&D and provided some additional details to the description.

Slight adjustments were done to the order of the questions and their exact wordings during the actual interviews. The set of questions on the practice of strategic release management containing these adjustments can be found in Appendix A (in Finnish).

All of the interviews were conducted by having two members of the SEMS research team present, with the author being present in all of the interviews. The author lead the discussion for roughly half of the interviews, and both researchers also took notes during the interviews. The interviews were recorded and the tapes were listened through when writing the case descriptions and later referred to as necessary. Based on this material, detailed descriptions of the companies' activities in the scope of the framework used (see section 4.1.2), their perceived problems and challenges, as well as observations and suggestions made by the researchers to improve the companies' practices were written.

The descriptions were sent back to the companies to check correctness. The general findings were then presented anonymously to all of the companies in a joint session. After this, company-specific observations and improvement suggestions were presented and further discussed with representatives from the interviewed companies. Then, informal conversation-like interviews were conducted with some of the interviewed persons to evaluate the usefulness of the suggestions as initially perceived by the case company personnel.

After an observation period (four months for Cielago and Slipstream, and six months for Cheops), a new round of interviews was conducted at the companies to find out what action, if any, had resulted from the first interviews and their dissemination. At Cielago, the head of product development and the person responsible for customer delivery were jointly interviewed in a two-hour session. At Slipstream, the follow-up interview was conducted in a two-hour session with the new manager of the software development function. The senior product manager previously interviewed had now a new post in the organisation. At Cheops, the head of R&D, two product managers, a project manager, a lead developer and the head of the quality assurance team were interviewed in sessions of one hour each.

No formal techniques were used during the interviews to extract or prioritise the current problems and challenges asked. The observations and suggestions attempt to focus on a

minimal set of issues perceived to yield best payback in terms of added controllability versus overhead caused. Identifying the current problems and challenges in one company triggered suggestions for improving the process in the others as well, and some suggestions were intentionally reused across cases.

4.1.2 Framework Used in the Interviews

To make an observation period of reasonable length possible, the interviews took place at the beginning of this study before theoretical work towards constructing a framework of key NPD decisions for small software product business was begun. Thus, the interview questions were formulated around the concept of strategic release management as described in (Rautiainen et al. 2002).

Because the more recent papers discussing strategic release management (Pyhäjärvi, Rautiainen, & Itkonen 2003; Rautiainen, Lassenius, & Sulonen 2002) had not yet been published at the time of conducting the interviews and (Rautiainen et al. 2002) do not provide in-depth details about the set of decisions made in strategic release management, the author utilised both his own insight and the expertise of the SEMS[4] research team at Helsinki University of Technology in forming a more detailed view of what decisions would be relevant in the strategic release management process of a company, and consequently, the interview questions used. The contents of the 'strategic release management key decisions' framework used as the basis for forming the interview questions (shown in Table 9) is based on free brainstorming and team expertise rather than rigorous comparison to existing frameworks. Those strategic release management decisions that were added in this manner and not directly derived from (Rautiainen et al. 2002) are written in *italic* in Table 9 below.

[4] See http://www.soberit.hut.fi/sems/ for more information on the SEMS research project

Table 9 The version of key product strategy decisions used as the basis for the interviews

Element in the outline of strategic release management decisions	**Interviewed issues**
Organisation, roles and responsibilities	*Structure of R&D organisation,* *Decision-making on the organisation,* R&D competences and recruiting
Product mix	Offerings Overview of software (past, present, future & under development) *Services (incl. support and customer-specific development)* *Decision-making* Timing of the marketing of new features
Release contents and requirements engineering	Release planning, Requirements elicitation and specification, Feature prioritisation and change management, Configuration management
Development pacing, release timing and release types	General shape of product development Phasing Concurrency Types of development effort Release types Project types Project management
Product architecture and technology	*Conceptual model of the product and implications to development,* *Architectural design,* *Employed technologies,*
Testing and risk management	Testing; Types, Timing, Reporting, and the rationale behind these Identified product and business risks Self-assessment Quality criteria Customer satisfaction

The areas added to the framework for interview purposes were product technology and architecture, accounting for services in product mix decision-making and the structure and roles in the R&D organisation. These additions were made based on our experience of what issues are important for product roadmapping in three small software product companies (Vähäniitty, Lassenius, & Rautiainen 2002).

4.1.3 Data Analysis and Case Descriptions

Based on the experiences gained from the interviews, the framework under construction was improved and tailored to better suit the small company context, and suggestions were made to the companies.

First, the notes made during the interviews were combined, and the interview tapes were listened through with correcting and making additions to the notes. After this, the data was thematically coded (Miles & Huberman 1994) using the elements of the framework outline from Table 9 as codes. Based on these codes, the data was clustered and the resulting clusters were written out as parts of the case descriptions. During the process, new codes, in other words, new for the key decisions framework were created, and some of the old ones were restructured and renamed to better reflect the underlying issues, or to respond to those elements that surfaced as important to the case companies.

The resulting framework is shown in Table 10 below and the changes made to the outline used in the interviews (Table 9) to reach this are explained in section 4.4. The framework of Table 10 is used in summarising the results of the cases in sections 4.2.1 – 4.2.2 and is the basis for the structure of the case descriptions in Appendix B.

Table 10 The version of the framework used to structure the case descriptions

Decision area	**Contents / Interviewed issues**
Organisation, roles and responsibilities	Organisational structure Roles and responsibilities Use of outsourcing
Product mix	Offerings; Overview of software (past, present, future & under development) For each product; Sales and marketing Revenue logic Servicing and deployment Release strategy (release types and planning) Decision-making Timing of the marketing of new features
Requirements engineering	Elicitation Specification Allocation and initial prioritisation Change management
Development model	Overview of how releases are built Types of development effort Releases (Overview, pacing, phasing, concurrency) Other Progress tracking
Technology selection and software architecture	Product architecture (incl. asset sharing, common conceptual view and design) and employed technologies Development infrastructure
Testing and risk management	Testing; Types, Timing, Reporting, Test documentation plus the rationale for these Test planning Identified product and business risks Release criteria and quality metrics Release success evaluation

Note, that not all of the case descriptions in Appendix B contain all of the elements shown in Table 10. This is because some of the elements were "discovered" as a result of the empirical phase and the analysis, as opposed to being already a part of the framework according to which the interviews were conducted. For example, the importance of evaluating the success of past releases was learned as a result of case Slipstream, and consequently, asking whether such activity is conducted at Cielago or Cheops was not done. While a similar element was incorporated from (Krishnan & Ulrich 2001) into the framework in the theoretical approach to constructing the framework, it did not affect the way the interviews were conducted.

4.2 Results

In sections 4.2.1-4.2.2, the suggestions given by the researchers, the initial feedback received and the results observed over time are described. The full case descriptions as well as perceived and observed problems and challenges can be found in Appendix B.

All of the figures summarising the observed problems and challenges and the results of the cases (Figure 3, Figure 4 and Figure 5 on pages 41, 43 and 45, respectively) are in mind map™ format. This makes it possible to show relationships between the observations, problems, challenges, suggestions and results. The key to the notation used is depicted in Figure 2 below. For details on the cases, see the case descriptions in Appendix B.

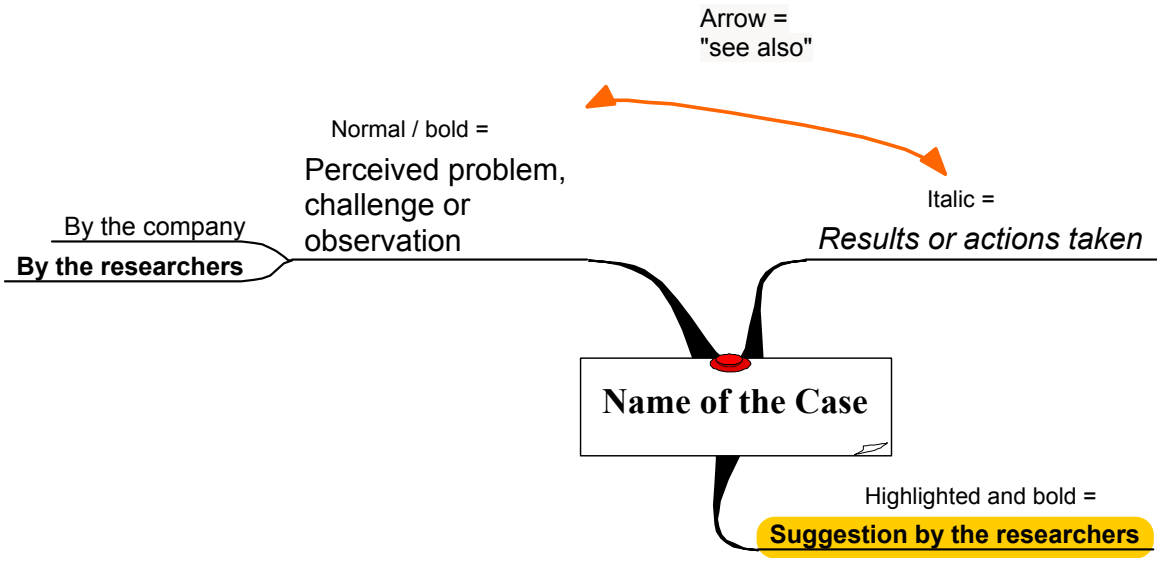

Figure 2 Notation used in summarising the results of the cases

4.2.1 Slipstream Ltd.

Suggestions

The suggestions made at Slipstream address development pacing, release timing, the amount and role of documentation in requirements engineering and testing and progress tracking.

The planned requirements engineering process offers a lot of room for making the requirements process more efficient. In theory it saves effort in the form of updating documents when changes are made to the scope of the release. However, the transition from the old, document-heavy process should be thought carefully. In any case, it seems clear that effort can be saved by eliminating some of the duplicate documentation. Also, a mechanism is needed to make late changes to project scope in a controlled fashion, and the most natural way to introduce this is to keep constant eye on the project scope versus the time left. In theory, the planned requirements engineering process handles this. A minor improvement to the requirements database would be to include non-software requirements for the product as well.

The development model requires clarification with respect to its pacing. If the release projects are currently schedule-driven in theory but feature-driven in practice, a possible explanation is that the prioritisation of requirements is not efficient enough (i.e., too many 'must-have' requirements) or change management practices are inadequate. To see whether this is the case, Slipstream's product development managers should ask themselves the

following questions: How much of the total effort of release projects initially comes from the features of the highest priority class, what amount of features initially allocated to a release project does get implemented, and what part of these is of the highest priority? Also, stronger mechanisms to track project progress should be taken into use. If effort estimation and hour reporting seems too heavy or otherwise suitable, literature on agile methodologies (for example, (Beck & Fowler 2001), (Schwaber & Beedle 2002) and (Cockburn 2002)) offers many alternatives. For example, holding 15 minute meetings to review the progress are a light-but-robust way to improve project controllability (Schwaber & Beedle 2002).

The testing practices yield several improvement suggestions. The feature database could in the future be used in easing with the documentation load in functional testing, for example by including test case specifications to the database and making the developers responsible for producing their first version. The effectiveness of the outsourced testing efforts should be evaluated as well, especially if the new product will require outsourcing as well. Also, although the interviewed person claimed that automatic testing does not help in dealing with the platform-incompatibilities, Slipstream should consider putting more effort into finding out if this indeed is so. The field failure rate metric seems a good idea and should be implemented to provide some quantitative data of the releases' success. Despite the ambitions of the interviewed senior product manager, starting out with a simple metric only should be tolerated since more dimensions can always be added later. Finally, the relationship between the types and amount of testing conducted and the product and business risks facing Slipstream should be explicated.

Initial feedback

The initial feedback to these suggestions was positive, and the senior R&D manager and Slipstream's CEO agreed on their importance. Also, work on the new requirements process was already underway, and to support this, a product management group was being formed. The responsibilities of the product management group would be to elicit those features from the customer interface that the sales personnel perceive important, guide in the specification and prioritisation of requirements and keep the sales personnel up-to-date with what the products can do.

One month after the initial interviews and presenting the suggestions, the interviewed senior product manager reported that they had taken initiatives towards managing the product mix through business case thinking, systematic defect tracking and improving the making of requirements specifications. Product-related documentation had been taken into the scope of quality assurance, and problems with the internal incompatibilities caused by the multitude of Java environments had been tackled by making the product itself inform the user of whether the configuration on which the program is run on is supported or not.

Results

After an observation period of four months, a follow-up interview was conducted. However, the interviewed person had changed from the senior product manager to the new head of the software function. Because the interviewed person had not been fully informed of the study being conducted and he was not aware of the discussions in the

initial interviews, half of the time allocated for the follow-up had to be used in 'selling' the interview and explaining what had been done earlier. Thus comparing the action taken by the company to the suggestions given proved more difficult than in the other cases.

During the three months that had passed since the initial interviews, initiatives towards collecting and disseminating best practices between projects had been taken, and a written final report was now required from each project. However, these were not yet disseminated. Effort estimations were now being made for features, but actual effort spent was not tracked. Although the latter had been tried, it had not received enough support and was put on hold until further notice. The development of the core technology was now organised as projects as well. The most striking improvements, however, were those made in product and release planning. Slipstream had adopted a roadmapping technique similar to the one reported in (Vähäniitty, Lassenius, & Rautiainen 2002) to specify its future releases, with participants from across Slipstream's organisational functions.

The perceived and observed problems and challenges, suggestions made, and the actions taken by the company during the follow-up period are summarised in Figure 3 below.

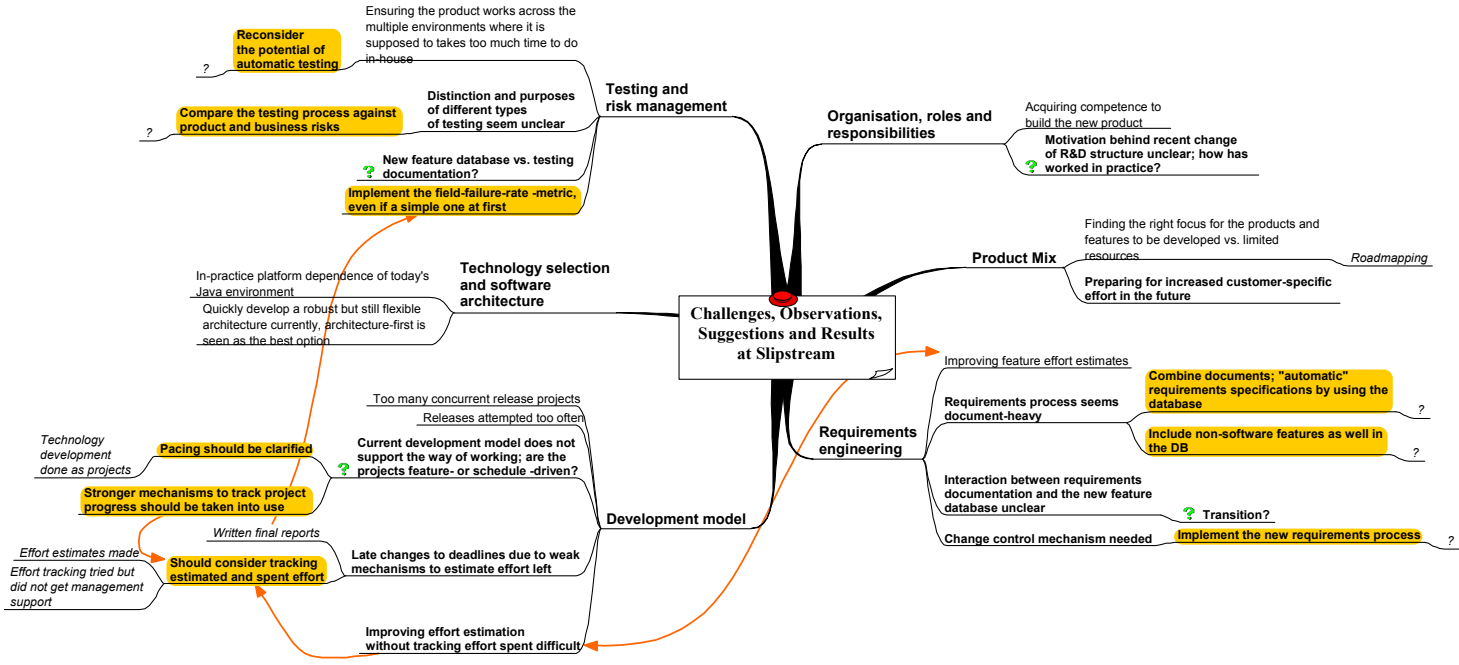

Figure 3 Problems and challenges, suggestions and actions taken at Slipstream

4.2.2 Cielago Ltd.

Suggestions

The most important suggestion made was to improve the communication between the sales and R&D. Towards this end, the roadmap of the future features of the product should be made more detailed, for example by including effort estimations, and some sort of explicit pacing and progress tracking should be introduced, perhaps even by tracking the effort spent by the R&D personnel. Also, we suggested that Cielago should start systematic

defect tracking using a simple tool, for example with the help of MS Excel or an open source system. Success in establishing the basic rules on how development and sales work together at Cielago is likely to require the involvement of someone enjoying a high respect from the entire organisation.

Initial Feedback

One month from the initial interviews resulting in the case description, the interviewed personnel told of their insights gained from the interviews. The personnel perceived that the research had helped them to realise some of the severe weaknesses in their current way of working. The most important realisation had been that the product was being created in a technology-push manner and the relationship between R&D and sales was not working. Effort had been put into thinking how the R&D-sales interface could work better and during the upcoming months, the entire from-idea-to-after-sales process was to be identified and defined complete with inputs, outputs, roles and responsibilities for each phase. As one interviewed developer put it, "realising these things has completely changed our opinion about how the company should operate". The most important target for improvement efforts during the first month had been how requirements for the product were handled.

Results

Four months from the initial interviews, a follow-up interview was conducted. In summary, the most significant changes at Cielago during the observation period were altering the roles and responsibilities of some of the key personnel to stimulate interaction between R&D, sales and the customers, introducing phases to their productisation process and specifying and analysing the requirements for new products more rigorously.

In the organisation, the customer interface had been strengthened with the former head of R&D (now the head of sales) and one of the software developers (sales support), and this had remedied the former problems with communication between sales and R&D.

With respect to product mix, Cielago had explicitly productised some of the services they were originally providing because of the necessity to ease the pressures from the immature markets and the long service cycle. Thus, the business model of the company and the role of the distribution channel had been changed.

For deciding on new products' requirements, a phased process featuring written plan documents and their review had been created, and according to the interviewed persons, it had already shown its worth. Due to written product plans, informed decision-making with respect to the perceived needs of the market and the potential benefits from new products can now be made early on. Improvements to the existing products were still made without formal change management, but the input to these modifications now came directly from existing customers because of the strengthened customer interface.

Distinct phases had been introduced to the from-idea-to-delivery process of the company. It now consisted of six phases, each with defined inputs and outputs. The product development process (the first two phases) had further been divided into nine phases. Except for analysing and specifying the requirements for new products, there was little

experience of the strengths and weaknesses of this new phased development model. While the company understood that the new process was to be iterative, the current description (phased and sequential) did not support this view very well. Progress tracking became more detailed due to the more detailed specifying of product requirements, and thus, the development tasks, but no formal approaches to effort estimation or tracking were in place here. A clear handoff had been introduced between hardware and software development. Now, software development did not start until the respective hardware was completely ready.

The interviewed personnel perceived that making technology and architecture decisions had become easier due to the new process for planning the products, the market becoming more mature and Cielago's increased understanding of their position in the industry value chain. Also, making educated guesses in terms of optimising hardware bill-of-material against future functionality had stopped.

On the basis of the follow-up interviews it was hard to say whether testing practices had changed significantly. A person had been hired to do testing and more effort was claimed to be spent in testing the hardware. What had changed was Cielago's own opinion of the quality level of its products – it was now realised there was still room for improving the quality of the product, which can be interpreted as a healthy sign.

The perceived and observed problems and challenges, suggestions made, and the actions taken by the company during the follow-up period are collected in Figure 4 below.

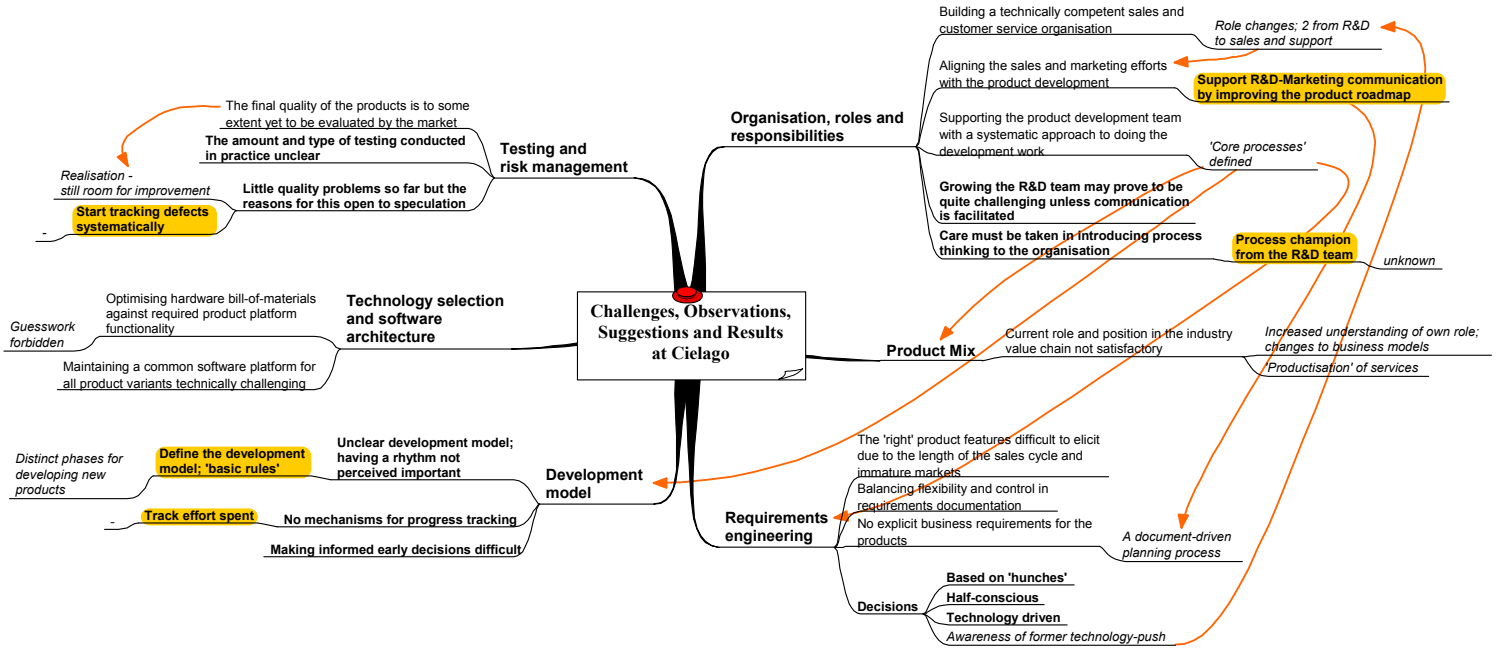

Figure 4 Problems and challenges, suggestions and actions taken at Cielago

4.2.3 Cheops Ltd.

Suggestions

The suggestions made to Cheops concerned mostly the requirements engineering process and its connection to the development model.

First, to tackle the challenges in gaining feedback on features' value as perceived by the customers, and for facilitating understanding of the direction where the product is going based on a requirements specification, the author proposed that a strategic dimension should be added to the features. This proposition was based on the concept of strategic buckets (Cooper, Edgett, & Kleinschmidt 2001) and its application in creating a 'multi-release technology plan' as suggested in (McCarthy & Gilbert 1995). In the proposed model, features in the database are categorised into the five types of strategic, competitive, customer satisfaction, investment and paradigmatic features. After classifying all the features, the managers decide the level of investment in each category. Once the target spending levels are established, multi-release plans can be made by prioritising the features in each category, and reflecting this against the amount of resources available for the releases over time.

Second, an XP-style planning game (Beck & Fowler 2001) was suggested for requirements allocation to releases, subsequent change management, and selecting the tasks for the buffer time at the end of the development phase. In the adapted version suggested, product manager proposes a desired scope, feature priority, release date and technology by the 'preparation' milestone utilising also the strategic buckets –categorisation. Then, by the kick-off meeting of the project, project manager responds with effort estimates for these and tells of the implications of the set for the project schedule. On the basis of this information, the most valuable set of features is chosen to the release project. During the planning phase, the product manager writes user stories of these features, project team estimates the effort for them and asks for clarification to these when necessary. By the end of the planning phase, product and project managers agree on the "final" scope for the release project considering the capability of the team. This "game" is then repeated for each development iteration and the buffer time.

Some minor improvements were also suggested. In requirements engineering, these were to collect ideas relating to the "whole product" to the feature database in addition to software features and recording the origin of the incoming feature ideas.

Initial Feedback

In the dissemination session held separately for Cheops, the general reception of the suggestions described above was positive. The planning game was found interesting. One of Cheops' product managers recalled that adding the strategic dimension to feature prioritisation seemed familiar from somewhere and might have been thought of in the past as well. Still, when asking for additional feedback after the session, the head of R&D did not consider the value from the case description or these suggestions very significant because he perceived that the company knew these issues already.

Results

Five months later, the head of R&D told that Cheops had adopted a strategic dimension to requirements prioritisation, the origin of features was now traceable, and for those features originating from the customers, a new role had been created to ensure that the implementation of the feature satisfies the customer who requested it. An XP-style planning game had been piloted in the initial planning phase of the project, and was to be used later between the iterations as well. Also, having a team in charge of testing and quality assurance efforts for both products had been a considerable help in dealing with challenges identified in the initial interviews, but its responsibilities needed further clarification.

In addition to reacting to the suggestions made, Cheops had worked to solve the problems and challenges that had been identified during the interviews. The most important changes related to clarifying the function and responsibilities of the production team. Also, a separate research effort had been started at Cheops in order to evaluate the usefulness of code re-write in various situations.

The perceived and observed problems and challenges, suggestions made, and the actions taken by Cheops during the follow-up period are collected in Figure 5 below.

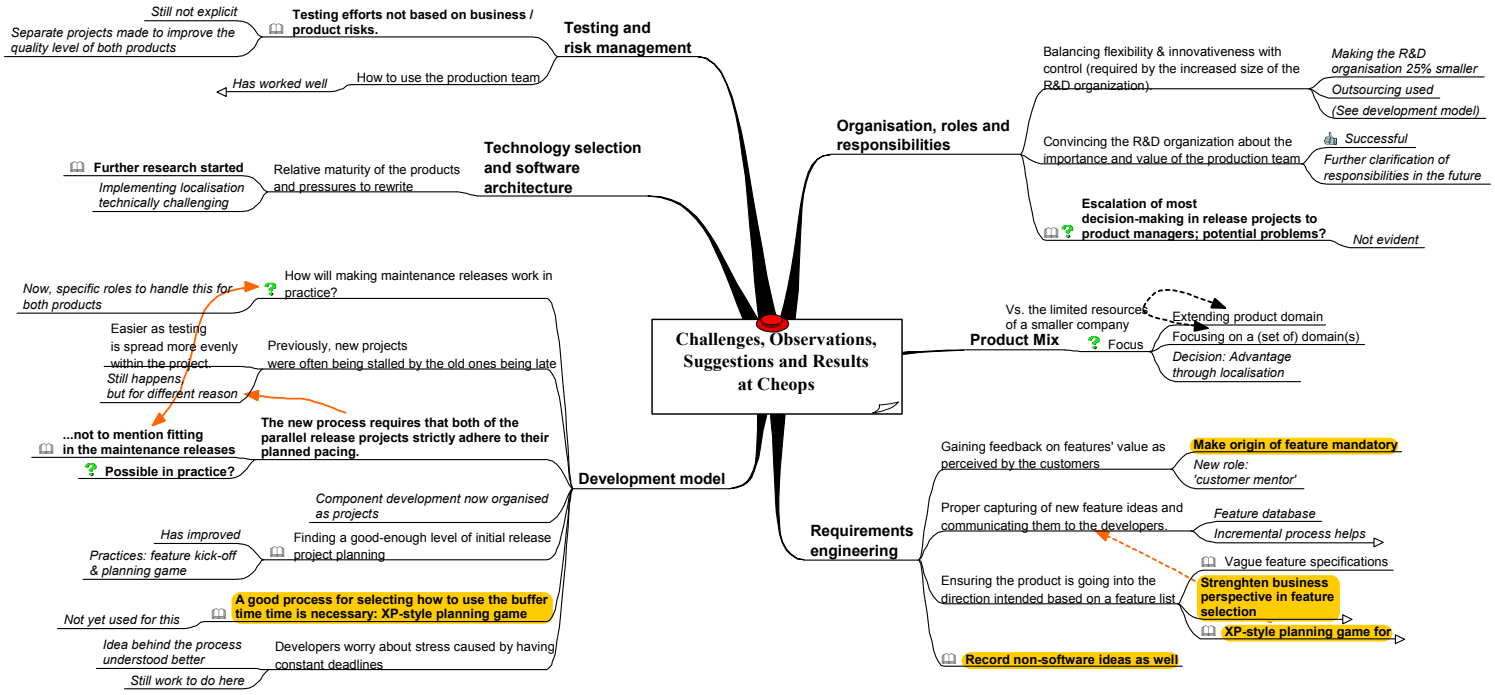

Figure 5 Problems and challenges, suggestions and actions taken at Cheops

4.3 Costs and Benefits to the Participating Companies

To answer research question (5) (Does the proposition for overcoming the gap between the needs and support found facilitate product strategy decision-making in practice?), the added value to the companies from this study is considered to consist of three possible clusters of insights gained: (1) those resulting from participating to the interviews and their

dissemination, (2) those resulting from the observations and suggestions made by the researchers, and (3) those gained from reading and commenting the written descriptions (see Appendix B).

At Cielago, conducting the interviews based on going through the areas of the construction was openly considered very useful by the company personnel, and during the follow-up period, the company had successfully tackled almost all of the problems and challenges identified during the interviews and in writing the description.

The same applies for Cheops, except that the personnel were less enthusiastic at the start of the observation period. However, this could be explained by the higher initial level of awareness of the issues featured in the strategic release management framework.

At Slipstream, the initial feedback was positive, but the results from writing the description, pointing out the problems and challenges and presenting the improvement suggestions were less clear.

None of the companies considered their own description as particularly helpful, which is natural because their contents and the suggestions had already been discussed with the participants. Thus, writing the descriptions contributed most to improving the framework constructed in this thesis. Evaluating whether the companies perceive benefit from reading each other's descriptions is out of the scope of this thesis.

The costs from participating to the study can be expressed in terms of the effort spent by the company personnel to activities specific to this study. At Slipstream, the effort spent by the company personnel was 3 man-hours for the initial interview, no hours for participating in the joint dissemination session (no attendant from Slipstream), 3 hours from reviewing the case description and 2 hours from doing the follow-up interview. At Cielago, the effort spent by the company personnel was 6 man-hours for the initial interviews and 3 man-hours for the dissemination, review of the case description and participating in the follow ups each. At Cheops, the initial interviews took 10 man-hours, participating in the dissemination 9 man-hours, reviewing the case description 3 hours, and participating to the follow-up interviews 7 man-hours. In addition to the costs listed, coordinating the study is assumed to cause overhead equal to one fourth of the effort. Thus the total effort required from the companies to participate to this study was 19 man-hours (Cielago), 10 man-hours (Slipstream) and 36 man-hours (Cheops). In the opinion of the participating companies, the insights gained from participating to the study outweighed these costs.

The costs (in man-hours) and perceived benefits from participating to the study are summarised in Table 11 below.

Table 11 Summary of the costs and perceived benefits to the companies

Case company	Cost	Initial feedback	Actions taken
Slipstream Ltd.	10h	"We agree on the improvement areas and suggestions."	Of the improvement suggestions discussed during follow-ups all had been reacted to. Reactions to many of the problems and challenges and some of the suggestions currently unknown.
Cielago Ltd.	19h	"The evaluation and the framework helped us to realise our strengths and weaknesses. We have started a major effort to improve our way of working."	All problems and challenges identified together had been addressed, and most with success. Approximately half of the improvement suggestions had been taken into action.
Cheops Ltd.	36h	"The results were interesting and most of the suggestions relevant. However, I believe we were aware of many of these issues already."	All of the suggestions made had somehow been acted on in the product development process. Two of the observations made by the researchers proved irrelevant. All of the problems and challenges identified together had been addressed.

As a conclusion, conducting an evaluation of a company's product strategy decision-making practices using a version of the key product strategy decisions framework helped to raise the awareness of the companies and yielded actionable and relevant improvement suggestions with reasonable cost. What is interesting and encouraging is that using the framework in companies already having a higher level of awareness and more mature practices seems to yield at least as much benefits as in companies with lower initial awareness and ad hoc practices.

4.4 Changes to the Framework

This section explains how the key product strategy decisions framework used for the interviews was modified based on the lessons learned.

It was noticed that in small companies, asking about the roles, responsibilities and organisational structure of the entire organisation rather than just about the R&D function provided a better overview of how the company operates with little additional cost. Also, during the analysis, *communication mechanisms* used both within R&D and between the various functions, for example, to the salesmen were noticed to be of importance and thus added explicitly to the framework. The competences and recruiting needs regarding the R&D organisation were discussed in all of the interviews and can be found from the case descriptions in Appendix B under the perceived and observed problems and challenges.

We noticed that to understand the company's portfolio of products (termed *product mix* in the case descriptions) and the way it operates, the way the products were *sold* and *distributed* and how revenue is created (or *revenue logic*) were of importance in product strategy decision-making and thus were included to the model. Also, the concept of *release strategy*, containing release roles, contents, timing (*release planning*) and types, previously scattered around the framework, was taken under the key decision area of product mix.

With the moving of *release contents* under *release strategy* in *product mix*, a practical way of restructuring the key decision area of *requirements engineering* was to divide requirement prioritisation as *allocation and initial prioritisation* and subsequent *change management* to accompany the original steps of *elicitation* and *specification*. *Configuration management* was not

explicitly discussed in the interviews except from the perspective of keeping track of the versions of the product on the market.

With respect to *product mix*, we noticed that asking about the past product portfolio along with the current and upcoming products helped to understand how product mix decisions were made in the company.

After the restructuring of *release types* and *timing* under *release strategy* in *product mix*, the key decision area of *development pacing*, *release timing* and *release types* was renamed simply as *development model*. *Project types* were now considered to be included in *release types*, *project management* was renamed as the better-to-the-point *progress tracking*, and the real-life experiences from the cases helped to understand and structure the variables involved in *types of development effort*.

The key decision area of *product architecture and technology* was not altered with respect to its contents, but its terminology and structure were revised to better explicate the areas found relevant in the case work.

In *testing and risk management*, *test documentation* was now understood as separate issue from *test reporting* (for example, test case specifications do not necessarily result in *test reporting* by themselves or vice versa), and the concept of *test planning* was brought up to describe the decision-making of how different testing is to be 'instantiated' to different types of development effort. *Quality criteria* was renamed as the more explicit *release criteria and quality metrics*, and evaluating *customer satisfaction* was concretised as *release success evaluation*.

5 A Framework for Supporting Product Strategy Decision-Making in Small Software Product Businesses

In this chapter, the results of the theoretical and practical approaches to constructing a framework for supporting product strategy decision-making in small software product businesses are combined. The resulting framework is defined and described, and the process of conducting product strategy decision-making is discussed.

5.1 Combining the Theoretical and Empirical Constructions

The version of the framework resulting from the theoretical approach (Table 8, p. 33) was taken as the basis of the combined framework and compared with the result of the empirical approach (Table 10, p. 38). The changes made to the former are explained below (also illustrated in Appendix D).

In the key area of *organisation*, managing the *use of outsourcing* was added because of its significance in two[5] of the case companies.

In *portfolio management*, the decision areas for each product are how the product is *marketed* (including the timing of marketing new product versions and features), *sold and distributed*, how it is intended to create revenue (*revenue logic*) for the company and what complementary *services* (including deployment) are offered. The other half of the area is managing how the releases of company's products are planned and delivered in practice (*release strategy*). While the contents of this key decision area did not change significantly as a result of the empiria, this structure and naming better reflects the issues in practice.

The key area of *requirements* was not changed as a result of the empiria. The experiences supported the notion made in the theoretical construction of the framework that in iterative software product development, it is practical to break requirements engineering down as *elicitation*, *specification*, *allocation* (i.e. initial prioritisation) and *change management* (i.e. re-prioritisation).

In the key area of *development strategy*, the empiria resulted in two important realisations. First, in addition to *communication among team members*, the communication mechanisms specified by the development model should address the *interaction to the customer interface* and the *rest of the organisation* as well. Second, it was understood that the possible *concurrency* and consequent interaction of different development models is important in a small company because the resources are often shared, and the situation of one development effort easily affects the other one(s) as well.

[5] At the time of writing this, managing outsourcing has become topical in the third case company also.

In the key area of *technology*, the importance of having *a common conceptual view* of the structure of the product and involving stakeholders in making important *design decisions* were added as a result of the comparison.

The key area of quality strategy gained depth as the result of conducting the empiria. Organising testing was concretised with decisions on test *timing*, *reporting*, *documentation*, *quality metrics* and the project-specific tailoring of these through *test planning*. Also, the risk-based approach to testing was concretised through understanding that product and business risks can be managed through setting *release criteria* and understanding how the previous release were viewed by the market (*release success evaluation*).

5.2 Key Decision Areas Defined

Table 12 below summarises the key decision areas of the framework.

Table 12 Key product strategy decision areas in small software product companies

Decision area	**Contents**
Organisation (*by whom, and where?*)	Organisational model Roles and responsibilities Team staffing Team physical arrangement and location Investments in team collaboration Use of outsourcing
Portfolio management (*what and when?*)	For each product; Marketing (incl. timing) Servicing (incl. deployment) Sales and distribution Revenue logic Release management; Operational (management; release process and configuration management) Strategic (planning; release contents, roles, types and timing)
Requirements (*what and when, specifically?*)	Elicitation Specification Allocation Change management
Development strategy (*how?*)	Development model(s); for each, Type of development process (overview, pacing and phasing) Progress tracking and control Communication mechanisms (within, to the customer interface & the rest of the organisation) Concurrency of development models
Technology (*by leveraging which technologies?*)	Product architecture (incl. asset sharing, common conceptual view and design) and employed technologies Development infrastructure
Quality strategy (*delivered with what emphasis?*)	Testing Types, timing, reporting, test documentation, quality metrics, test planning (i.e. tailoring to individual projects) Risk management Release criteria Release success evaluation

The contents of the framework and their grouping have been made based on the understanding of the needs of small software product companies gained from the theoretical and empirical approaches conducted in this study. The grouping of the areas has been made with keeping in mind the purpose of being explicit. For example, the entire *require-*

ments key decision area could be considered to be a part of *release contents* in portfolio management, but the current grouping is more meaningful in increasing the awareness of the target audience.

Another factor justifying the current grouping is that in a small organisation, the responsibilities of the key personnel are cross-functional, broad and possibly imprecisely defined as well. While in larger companies splitting the area covered by the framework into several areas with separate processes, for example, 'technology management', 'product line management' and 'product management' could be useful, the author proposes that in small companies these management issues in new product development should be handled by the same group of individuals. Although a more detailed discussion is out of scope, the author believes that "splitting" product strategy decision-making between different forums is feasible as long as the decisions made in one of the resulting forums (for example, product line decision-making) do not set significant constraints or requirements across others.

Sections 5.2.1-5.2.6 below define and describe the key areas of product strategy decisions in small software product businesses.

5.2.1 Organisation

The decision area of organisation encompasses how the work is organised in terms of organisational structures (*organisational model*), what *roles and responsibilities* are needed, how resource allocation is made (*team staffing*), decisions on the physical work environment (*team physical arrangement and location*), how teamwork is supported (*investments in team collaboration*) and outsourcing is managed (*use of outsourcing*).

Organisational model

Organisational model describes those organisational structures and associated functions that are assumed stable over time unless explicitly changed. Organisational entities can also be hierarchical, for example, instances such as development or testing teams and product management in product development.

While decisions on organisational model determine, for example, whether the company has personnel dedicated to marketing, sales, customer support, product development etc. and what the functions of these entities are, it does not address which people belong to which entity nor their specific roles and responsibilities.

While in small companies there seldom are clear organisational boundaries, deciding about the organisational model is important because organisational model is one of the most important factors in determining whether the company's is capable of operating according to its intended business model (Linder & Cantrell 2002).

Roles and responsibilities

A *role* is a job description that is associated primarily with a function or a process. A role defines the activities the person having the role is responsible for in a process, or in short, his *responsibilities*. A person can have multiple roles, for example, a sales manager in a small

software company can act in the roles of the head of an organisational unit, the supervisor of the company's sales personnel, a salesman or an expert. Also, in certain cases it is meaningful that a single role is associated with multiple persons. (Laamanen 2001)

Defining roles and responsibilities is especially important in small companies because of low organisational hierarchy and the emphasised role of key personnel such as senior management. In the case of a small company, organisational change is more often realised as re-defining roles and responsibilities than changing the organisational structures themselves. Thus, acting under the 'wrong' role in a given situation may result in fluctuation in the company's business model.

As a conclusion, explicating the roles and their responsibilities together with understanding the levels of abstraction in the development process, for example using the *cycles of control* (Rautiainen, Lassenius, & Sulonen 2002) helps product development to act coherently.

Team staffing

Resource usage decisions, or more appropriately for small companies, *team staffing*, is the most visible mechanism to operationalise strategy (Cooper, Edgett, & Kleinschmidt 2001; Krishnan & Ulrich 2001). Inadequate practices in resource allocation can cause wasting of resources on the wrong things while the issues deserving attention are starved.

In small companies team staffing is an important decision area because of the absolute limitedness of the resources. Deciding about the use of the company's product development resources at a coarse level includes allocating resources to product development activities, but also to services requiring attention from product development personnel. These can be for example maintenance, training, tailoring, or systems integration. This is further complicated by the fact that in small organisations, key people usually have several roles.

Team physical arrangement and location

The physical arrangement and location of company personnel and teams[6] affects the effectiveness of communication, and consequently, all work (DeMarco & Lister 1999). In small companies, the people responsible for product strategy decision-making should recognise and address this phenomenon by taking the responsibility of looking around, reflecting on what they observe and discussing their 'best guesses' on what arrangements, conventions and policies could suite the company's and the facilities' strengths and weaknesses (Cockburn 2001).

Investments in team collaboration

The principle of "Team = Software" (McCarthy & McCarthy 2002) means that the behaviour of a team maps directly to the qualities of its output and vice versa. If the product with certain characteristics is desired, the most effective way to promote this is to ensure that the team involved has the 'same characteristics' when it is developing the product. While this is both an insightful and an extreme proposition, an order-of-

[6] 'Team' is not restricted to denoting only the software development teams of a company.

magnitude difference between the productivity of effective and less effective teams exists (Humphrey 2002; McBreen 2002). Thus, decisions on investing in teams' capabilities and collaboration have strategic implications.

Use of outsourcing

Many companies have chosen to outsource less critical resources that can be obtained at better quality and/or lower cost outside the organisation (Kotler 1997). Here, outsourcing can mean anything from hiring programmers to provide additional muscle to help in developing a certain release of the product to the acquisition of subsystems or other components of the offering from outside companies. However, regardless of its exact nature, outsourcing entails both considerable risks and potential payoff, and decisions on whether outsourcing should be used and managed are of strategic importance.

5.2.2 Portfolio Management

Portfolio management consists of deciding on *marketing*, *sales and distribution*, *revenue logic*, *servicing* and *release strategy* across the product portfolio offered by the company. *Product portfolio* is the set of all products and items offered for sale by the company (Kotler 1997), and besides pure software products, software companies' product portfolios usually include maintenance and other services as well (Afuah & Tucci 2002).

Marketing

Marketing refers to the decisions concerning market segmentation, target market(s) and presenting the products to the customers in a way that enhances their perceived value. In the software product business, connecting marketing and the product development through decisions such as what new features can be marketed (and how) at what stage of their construction is of crucial importance

Sales and distribution

Sales and distribution refers to decisions on how the products are intended to reach their markets, in other words, how the products are *sold* and *distributed* to the customers. Besides the initial reference customers and direct sales, small companies must often employ indirect sales channels to reach their markets.

Revenue logic

This area refers to how the company extracts value from its operations – its mechanisms for creating sales revenue, the basic idea behind pricing, and utilising other possible sources of financing (Afuah & Tucci 2002; Rajala et al. 2001). Some examples of decisions regarding *revenue logic* are does the company sell product licenses and provide upgrades and maintenance for free, do the customers get the software product for a negligible price and pay the company for associated services and what share of profits possible indirect sales channels get.

Servicing

The software itself is often not the entire offering – it is often combined with services. The concept of whole product (Moore 1991) implies that the delivery of the core benefit the customer is buying can be enhanced by either modifying the way it is packaged, or by complementing it with services. Thus, restricting product strategy to planning product features only limits the view on what has to be achieved in order to put together a compelling offer. The whole product must be considered, not only its software component.

Incorporating and managing services can be challenging, especially for product vendors (Nambisan 2001). Synergies between the product and the services do not realise automatically, and for example, problems in balancing resources between product development efforts and providing services can cause serious delays in product development schedules (Hoch et al. 2002) and failure to recognise the resource implications of new services in the experience of the author can lead to a crisis (Vähäniitty, Lassenius, & Rautiainen 2002). Small software product companies should thus consider any customer-specific effort induced after the start of the business relationship as deliberate servicing

Thus, customer-specific software development, training and consultation in how to use the product, technical issues regarding the delivery of the product, such as installation, maintenance (delivering new product versions) and support (helping the customer use their current product version) are all types of product-related services. The cost and benefits of offering services should be balanced, and possible long-term implications should be kept in mind when complementing the products with services.

Release strategy

A *release* means passing a software build and associated documentation on to one or more parties outside of development (Penny 2002). The framework divides release strategy into two perspectives, the long-term *release planning* (strategic perspective), and *release management* (operational perspective). Releases can be internal, for example from the developers to the testers, or public, for example to a specific end user, a trade show, or to all customers.

Strategic perspective. The *strategic perspective to release management* provides the interface between business management and product development. Managing contents, timing and intended business goals for future product releases based on the market information available is a prerequisite for timely delivery of good-enough quality. The strategic perspective to software release management combines business planning with managing product development through release planning, which consists of deciding about the *contents*, *timing*, *roles* and *types* for future product releases.

Contents refer to linking product features to business requirements, in other words, the needs the customers are perceived to have for the product (Wiegers 1999), market opportunities, and deciding which features should be included in which release. High-level product requirements are the main interface from business planning to the individual release projects (Rautiainen, Lassenius, & Sulonen 2002).

Timing means identifying and exploiting a window of opportunity and making trade-offs between functionality, quality and time-to-market based on assessing the product against its competitors.

Roles refer to the releases' intended business implications for the company and the planned audience for the release (Rautiainen, Lassenius, & Sulonen 2002).

Types refer to the classification of the release according to some internal or external schema (major, minor, patch etc.) characterising the scope of the changes in the release from for example, the previous versions. Release types are often used also for marketing purposes. Examples of release types at Microsoft are Microsoft Windows 98 and 95 ('major' releases), service packs ('minor' releases) and security updates ('patches').

The strategic perspective to release management should address the company's plans for technology and product development, and result in

- an understanding of where the products are and where they should be going
- identifying the needs and objectives for new product releases, and
- identifying directions for extending and further developing the technological basis of the products

An example of a technique for covering the strategic perspective to release management is product roadmapping (Kappel 2001).

Operational perspective. The *operational perspective to release management* refers to the technical process of making a release happen and the set of services, tools and methodologies used (Bays 1999). It also includes managing how, in technical sense, the product and its different versions are created and distributed as releases. Also, keeping track of the configurations of the product that have reached the market belongs here. Examples of concepts belonging to the operational perspective to release management are builds, technical release classifications, release numbering and aspects of software configuration management (Zeller 1996).

5.2.3 Requirements

Basically, the requirements engineering process of a software product company is the mechanism for capturing product and feature ideas, prioritising them and transforming the viable ones as part of the product offering. Deciding about the principles this process is based on is an essential part of product strategy.

Software *requirements* can be functional requirements or quality attributes, with the term *feature* referring to a group of related software requirements (Bosch 2002). *Business requirements* represent the needs perceived for the product and are addressed in the product as features (with many-to-many relationships possible). As a release gets closer, its contents can be further specified from the level of business requirements to *features*, and then possibly down to *functional* and *non-functional requirements*. While requirements depend on each other in complex ways (Carlshamre et al. 2001), in practice it is often feasible to specify a system using features only (Beck & Fowler 2001).

Note, that in practice all the levels of requirements discussed above, for better or worse, may not be explicit. For example, an organisation may express their perceived business requirements directly as specific functionality, in which case information on why the functionality is important is lost, making for example change management (see below) more difficult.

The *requirements* decision area consists of planning and collecting potential requirements for future product releases (*elicitation*), properly documenting them for the company itself and possibly for the customers (*specification*), *allocating* them to future releases, and keeping this allocation fit in the pressures of changing requirements, schedule constraints and the changes' implications for the whole product (*change management*). Moving features and their parts between releases, in other words change management, should be based on the relative importance of the business requirements in question. Also, incorporating bug fixes to the releases is considered as change management, with the criticality of the known bugs being a major determinant when prioritising these together with new and enhanced features.

5.2.4 Development Strategy

Development strategy refers to the way the company's product development is organised in order to achieve its business goals, and consists of one or more *development models*. A development model means the general way development efforts are organised (if organised at all) for a certain kind of undertaking, for instance a release project. Thus, decisions about a development model include how the *pacing* (i.e. rhythm) of the development is supported or enforced, for example, whether the effort is divided into different *phases*, what measures are used to *track and control progress*, what *communication mechanisms* there are among development team members and the rest of the organisation and how possible simultaneous instances of development models should interact.

The business the company is in should guide its development strategy. This means that the company's business priorities should be evident from the way the development efforts are managed. For example, a company whose customers expect an update to the product every Friday is likely to have a different development process than a company who releases an upgrade for every six months or so and can decide the exact release time on its own.

The cycles of control framework (Rautiainen, Lassenius, & Sulonen 2002) provides means to visualise the relationship between *development strategy* and individual *development models* (Figure 6) and the roles of *pacing* (Figure 7) and *phasing* (Figure 8) in a product development model.

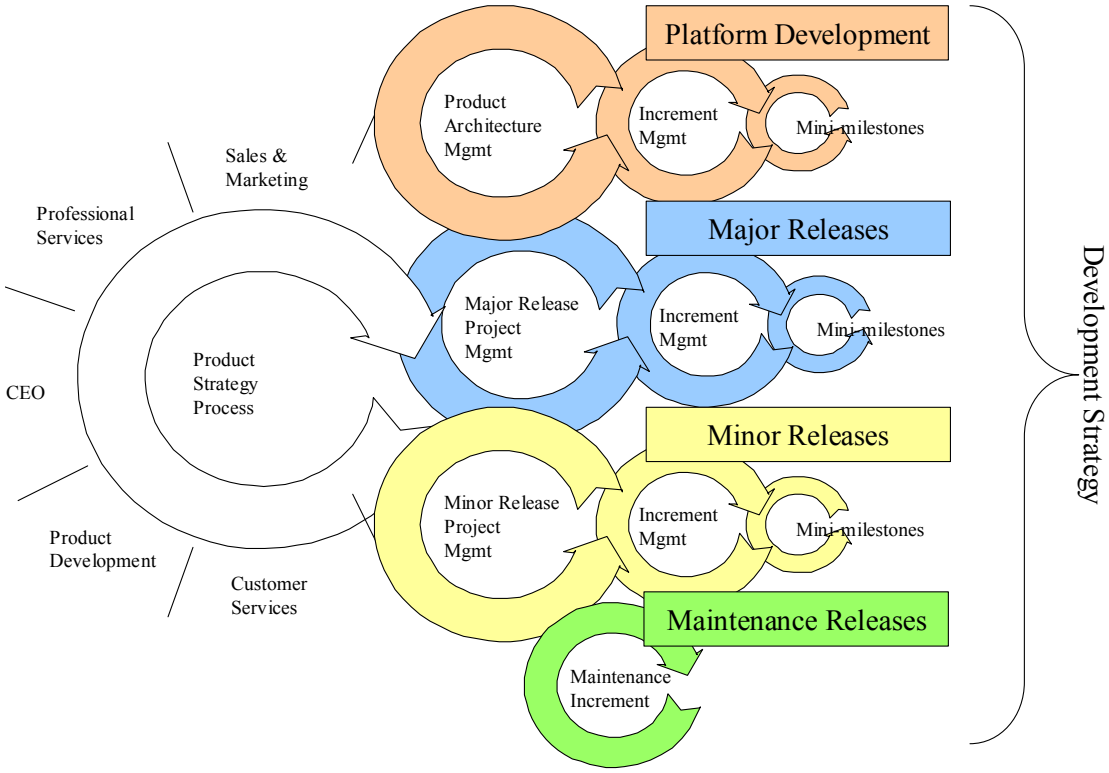

Figure 6 *Development models* for different *release types* as components of *release strategy*

Pacing

Pacing means creating a rhythm to the development efforts, for example by conducting the entire development as projects with clear start and end dates, or pacing the daily work with various practices.

When the development has a rhythm, people can adjust to the beat by adjusting the intensity of their own activities and directing their efforts towards a clear goal in the vicinity. Pacing creates a predictability that makes people feel in control, gives them greater focus and confidence, and often increases performance. (Brown & Eisenhardt 2002)

A simple example of creating rhythm through pacing is deciding that the development effort starts on a certain date, has a targeted end date and it consists of three phases, with a review of the progress at the end of each phase.

Main approaches to pacing are time pacing and event pacing. Time pacing means doing handoffs (for example, creating new products, making new releases, introducing new services and entering new markets) according to the calendar, and event pacing drives evolution according to occurrences, such as moves by the competitors, shifts in technology, or new customer demands. Event pacing is more erratic than time pacing and typically reactive (Brown & Eisenhardt 2002).

Because the future is impossible to predict exactly, managers are always somewhat event-paced, and time pacing is an effective way to counteract this. Time pacing forces both managers and developers alike to look up on a regular basis, survey the situation, adapt if necessary, and then get back to work. Time pacing also helps to keep from making changes

too often. Sometimes, in a fast-paced market, managers try to adapt to every change, which may result in rushing through decisions with inadequate data, pulling out of new markets before these markets have had time to develop, or dropping a promising technology without a sufficient trial (Brown & Eisenhardt 2002). From the perspective of the developers, constant change may cause them to stop reacting entirely, since they keep anticipating the 'next change' that would make any effort committed in the meantime useless.

Synchronising the rhythm of the development with the marketplace can also be used to thwart competition (Zahra & Bogner 1999). In the software product business, Netscape (Cusumano & Yoffie 1998) and Microsoft (Cusumano & Selby 1995) are perhaps the most famous examples of using time paced development.

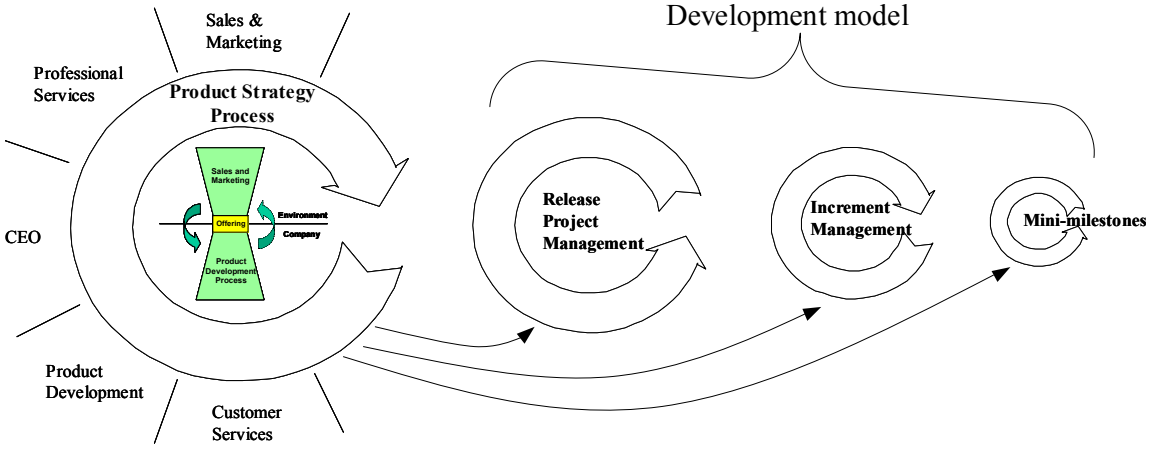

Figure 7 Pacing is realised through the number, durations and types of *cycles of control*

Phasing

If pacing determines whether the development model features clear start and end dates and (more or less) distinct phases, *phasing* defines the nature, emphasis and objectives of such dates and phases. For example in a project-based development model with three phases, the phases can be defined as planning, development and testing with specific goals for each phase, and semantics for their start and end dates. For example, the start date means that resources for development are committed from that point on, and the targeted end date is set because an important trade show where the product should be demonstrated coming up three days after. For example, the waterfall model of software development (Brooks, Jr. 1995) suggests a certain phasing. Figure 8 shows how phases in a development process can be communicated using the *cycles of control* framework.

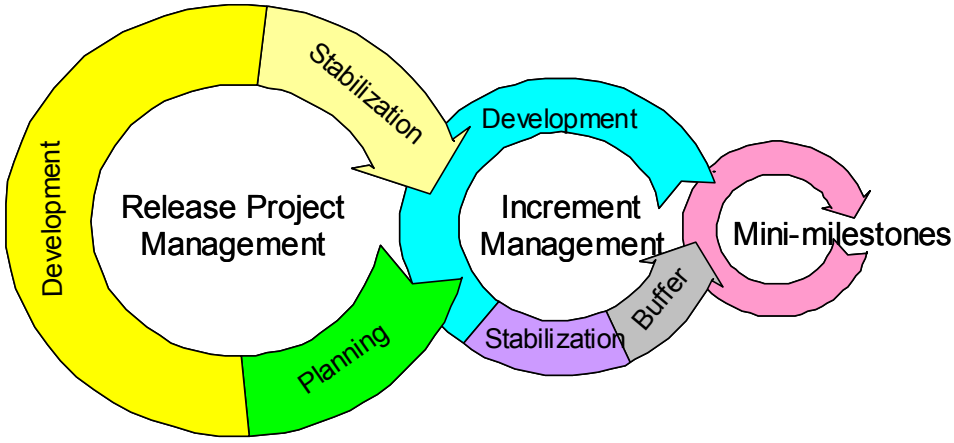

Figure 8 An example of phasing in an iterative and incremental development project

Communication mechanisms

Supporting effective communication is one of the most important tasks of the development model. Thus, when deciding on the development model, product strategy decision-makers should attempt to comment on how communication both among the team members and between the team, the customer interface and the rest of the organisation should be conducted. Examples of deliberate communication mechanisms are different kinds of documents and meetings. While deciding about the internal communication mechanisms of a team is not usually a strategic decision, having inadequate practices in place can certainly have dire consequences.

Progress tracking and control

Although pacing, phasing, and having defined communication mechanisms in place creates visibility and control to the development, the author finds it useful to separate progress tracking and control from these. This is because not all means to track and control progress involve pacing or phasing (or vice versa), and not all instances of pacing or phasing give direct indication of development status. Also, while most communication gives information on the status of the development, defining the communication mechanisms that are explicitly given this purpose is beneficial. Just because some people know where the project is going does not mean that everyone concerned does.

For example, comparing estimated development effort to the actual hours committed and the tasks completed over time can give a very good understanding of development status, but does not require pacing as such. Many other types of measurements fall also into this category. Likewise, checking all program code into the version control system and running a set of automated tests at the end of the day (Cusumano & Selby 1995) is a low-level way to pace product development, it does not give a direct indication of development project progress.

The conclusion here is that pacing, phasing and having defined communication mechanisms in place facilitates progress tracking and control, but the actual mechanisms used for progress tracking and control should be addressed in the framework separately due to the difference in perspective.

Concurrency of development models.

Development strategy also specifies how the development models interact in the situation when there are multiple instances of the development models underway. Examples of possible interaction are coordinating the use of common resources or re-allocation of resources from one development project to another.

5.2.5 Technology

By making the trade-offs involved in architecture and technology choices explicit for other stakeholders, the dangers from product development relying on old experience and skills when choosing perhaps the most pervasive constraints for the new product in the form of architecture and implementation technology are decreased.

Employed technologies

A key component of product planning is the decision about which technologies to incorporate in a forthcoming product. Also, competitive conditions may require a firm to develop technologies and products simultaneously, which increases the risks involved (Krishnan & Ulrich 2001).

Product architecture

Although the actual architectural design is not a part of product strategy decision-making, key personnel from outside of the actual development work should be involved in making technological and architectural decisions, and the product development should take a consulting role instead. Deciding on the architecture of the software and the implementation technologies should be based on the business requirements for the product, and the needs for incrementally developing the product or extending the product line. Also, most non-functional requirements directly affect the set of choices available, and these should be made explicit from the business requirements to the extent possible.

The author's experience from working with small companies suggest that a common language to refer to the parts of the software and their relationship to the envisioned product offering may be lacking even when the product development organisation is very small (Vähäniitty, Lassenius, & Rautiainen 2002). This makes planning the future development of the product difficult, and the author proposes that establishing a common conceptual view of the product is essential to successful product strategy decision-making.

Development infrastructure

Deciding on development infrastructure deals with the selection, acquiring and usage of development tools and environments and their sharing amongst projects. In small companies these decisions have implications of strategic nature, because development tools and infrastructure easily shape the development process (Fayad, Laitinen, & Ward 2000), tool and infrastructure investments can have direct impact on the company in terms of monetary cost, and the development tools and environments used are often coupled to the technologies employed in the product as well.

5.2.6 Quality Strategy

Defining and operationalising "good-enough" quality is one of the major strategic decisions a software product company has to make on a continuous basis (Hoch et al. 2002). However, talking about "good-enough" implies an understanding of the risks involved. Whilst most companies see risk management as a key strategic issue, risk is typically treated tactically and most often in an ad-hoc or piecemeal manner (Clarke & Varma 1999). Also, even recent work on software risk management emphasises the tactical perspective of managing risk within a single development project (Kontio 2001) rather than the perspective of the company releasing a set of product versions.

The area of *quality strategy* aims to combine quality assurance with risk management by basing decisions on what kind of testing is conducted, how it is conducted and for what types of development effort, on an assessment of product and business risks. Thus, quality strategy is essentially about *risk-based management of testing*, and the results of paying proper attention to business and product risks should be evident in the company's approach to validation and verification. This perspective is relatively recent in software engineering literature (Kaner et al. 2002; Pol, Teunisen, & van Veenendaal 2002) and a more comprehensive discussion of these dynamics is out of the scope for this thesis. Instead, the focus is on the variables involved; types of testing and their usage, test documentation, quality metrics and release criteria, and the success of past releases.

Testing

Testing means more than just detecting and correcting errors in the software – it aims to maximise customer satisfaction and provide feedback for process refinement (Pyhäjärvi, Rautiainen, & Itkonen 2003). Decisions on testing fall into the following categories: *types of testing* conducted, *test documentation*, *quality metrics*, and the project-specific tailoring of these (or *test planning*).

Types of testing. While in small companies the technically oriented people may have the best insight on how to carry out the actual testing, management should provide input on the consequences of different kinds of failures in product quality. Thus, specifying the different types of testing effort and their purpose is considered a part of product strategy in small software product companies.

Test documentation. The level of documentation utilised and produced by testing should vary depending on the context. In some cases, complete test case specifications and comprehensive reporting are necessary, while in others, trying out the product ad hoc may be sufficient. Defect reports are considered to be a type of test documentation.

Quality metrics. Quality metrics refer to quantitative information characterising the state of the object being tested. Again, the metrics used, their benefits and costs should be evaluated depending on the development model and the release in question.

Test planning. Testing can prove difficult to integrate into the software development process. In many cases, testing is easily left just to 'happen' at the end of the development project especially if the resources are scarce, the pressure is on time-to-market and efforts are focused on implementation. Different products and different releases may require

different level and type of verification and validation efforts (Pyhäjärvi, Rautiainen, & Itkonen 2003). Test planning means specifying which testing types are to be utilised and how, the amount and type of documentation necessary, and the quality metrics used. In other words, test planning is tailoring the testing practices to individual development efforts, for example, to development projects.

Risk management

Testing is not independent from other product strategy decisions. The development model sets the limits and expectations for test pacing, for example, through the actual builds for which testing can be conducted. Requirements engineering guides the feature selection and provides information on the set features to be tested. Technology selection and software architecture influence the ability to start testing early and the importance of testability should be accounted for. (Pyhäjärvi, Rautiainen, & Itkonen 2003)

Besides these issues, the goals for testing must be derived from product strategy through balancing time, scope and quality with the product and business risks facing the company (Pyhäjärvi, Rautiainen, & Itkonen 2003). For example, a company releasing weekly updates of its product might consider employing an automatic set of test for preventing the most obvious bugs from reaching the end user. Risk management steps into the picture also in concretising what the 'most obvious bugs' are.

Techniques for risk management have been widely discussed in literature (Kontio 2001), and this thesis does not go into the details. However, two basic ways to incorporate elements of risk management into quality assurance decision-making on the strategic level are determining release criteria for the product, and measuring the success of products already released.

Release criteria. Release criteria are the conditions under which the work-in-progress can and should be released (Bays 1999). While establishing and collecting quality metrics is considered in the strategic release management framework as a part of testing, setting criteria for "good-enough" quality plays an important role in operationalising the 'risk' in risk-based testing and should be addressed as part of product strategy decision-making.

Release success evaluation. Understanding the strengths and weaknesses of past product releases both in terms of added value from functionality and (absence of) defects provides depth to decision-making on setting release criteria and usage of quality metrics (Cusumano & Selby 1995). Mechanisms to evaluate the success of past releases are crucial in forming an understanding of the product and business risks facing the company.

5.3 The Framework and Product Strategy Process

While detailed discussion on *how* product strategy decision-making should be conducted is out of focus for this thesis, some of the experiences from our research so far have been collected into this section for the purposes of illustrating the role of the framework in the management of small software product companies. Below, the relationship of the constructed framework to the actual decision-making processes is clarified.

As the basis for this discussion, we will consider the concept of strategic release management (Rautiainen, Lassenius, & Sulonen 2002) as representing the *product strategy process in small software product businesses*. Strategic release management means addressing release and development schedules for the product(s), composition of individual releases, changes to the underlying technology and complementary services and planned resource usage across the company's product line(s) (Pyhäjärvi, Rautiainen, & Itkonen 2003; Rautiainen et al. 2002; Rautiainen, Lassenius, & Sulonen 2002; Vähäniitty, Lassenius, & Rautiainen 2002).

Strategic release management is a process for making product strategy decisions, and the framework of key product strategy decisions guides in setting its scope. In practicing strategic release management or designing this process in small software product companies, the framework of key product strategy decisions constructed in this thesis should be viewed as a context-specific checklist of the managerial issues that are likely to be relevant to small companies in the software product business. Not all of the described issues are equally topical to all companies, and even within a single company, the relative relevance of the areas is bound to change as time goes on. Thus, the described areas should all be kept in mind and their relative emphasis in the strategic release management process and the time perspective taken will vary over time and with changing business conditions.

Strategic release management is a continuous, paced activity having a variable scope. Obviously, conducting product strategy decision-making is continuous in the sense that it does not have clear start or end points, save perhaps those of the company's business itself. However, this does not mean that such decisions should be made without a clear pacing. The author proposes that the strategic release management process should be both time-paced (proactive) and event-paced (reactive). This means that those responsible for the activity could meet at pre-defined intervals to review the status of various strategic release management decision areas in the scope of the meeting, but meetings could also be arranged when a pressing need requiring the attention of the key personnel arrives. For example, a roadmap featuring future releases could be reviewed quickly every month to assess progress and whether scope changes have to be made. A more comprehensive update taking an in-depth look at future releases' contents, timing, roles, and types with participants from the company board could be scheduled at every six months.

While the concept of product roadmapping (Kappel 2001) covered many of the important decisions in strategic release management mentioned in (Rautiainen, Lassenius, & Sulonen 2002), this study has extended the perspective on the issues that should be accounted for in product strategy decision-making beyond the scope and application of our and our associates' earlier work

(Pyhäjärvi, Rautiainen, & Itkonen 2003; Rautiainen et al. 2002; Rautiainen, Lassenius, & Sulonen 2002; Vähäniitty, Lassenius, & Rautiainen 2002; Vuornos 2002). Even though the added depth stems out of the necessity for providing a holistic perspective to product strategy decision-making in small software product companies, the presence of these new elements (for example, roles and responsibilities, products' revenue logic and development models) suggests that designing a comprehensive strategic release management process is not necessarily straightforward.

Strategic release management should involve a cross-functional group of participants. The strategic release management process should involve stakeholders across any functional organisation structures. Thus, in a small company of 16 personnel with 8 developers, 4 salesmen, 3 in sales/technical support and a CEO, a possible configuration for the strategic release management team could be the CEO, the heads of product development and sales and someone from the support team.

Tools and techniques in strategic release management. To facilitate strategic release management, different tools and techniques can be used. For example, product roadmapping is an approach used to document and communicate plans for future releases (Kostoff & Schaller 2001). Scenarios (van der Heijden 1996) can be used to deepen understanding of decisions related to marketing, sales and distribution and revenue logic and communicate the decisions made. Mapping techniques exist for balancing the product portfolio (Cooper, Edgett, & Kleinschmidt 2001), or perhaps more appropriately in the case of small companies, the set of requirements to be implemented. However, according to the experience of the author, small companies often do not understand the underlying issues these kinds of techniques are designed to help with, and easily end up rejecting them as too heavy or otherwise poorly suitable because they do not know what to look for. Based on the understanding gained in the process of constructing the framework, the hypothesis of the author is that many of the techniques and tools proposed to help in new product development are applicable to small companies. Adopting and tailoring such techniques should begin with identifying and matching the underlying decisions the techniques are intended to address with the needs of the company.

5.4 Summary

This study proposes that product strategy decision-making in small software product business can be supported by identifying what must be accounted for in the strategic management of their product development. The identified key product strategy decision areas are *organisation*, *portfolio management*, *requirements*, *development strategy*, *technology* and *quality strategy*.

The decision area of *organisation* encompasses how the work is organised in terms of organisational structures, what roles and responsibilities are needed, how resource allocation is made, and decisions relating to the physical work environment and how teamwork is supported.

Portfolio management involves decisions about the set of all products and items offered for sale, their marketing, sales, distribution, associated services, revenue logic and planning.

The decision area of *requirements* deals with specifying the contents for future releases. This involves eliciting, prioritising and allocating requirements to timed releases, keeping this allocation fit in the pressures of changing requirements and schedule constraints, and the changes' implications for the whole product.

The decision area of *development strategy* refers to the overall form of the process models used (for example, waterfall, iterative and incremental, etc., or in some cases even whether

the work is scheduled at all), with an emphasis to the internal rhythm, structuring and controllability of the development efforts.

The decision area of *technology* deals with involving key personnel of the company in making technological and architectural decisions, and deciding on the infrastructure and tools used in product development.

The decision area of *quality strategy* consists of decisions on what kind of testing is conducted and how, the relationships of testing types on different types of development effort, and balancing the testing practices against perceived product and business risks.

The product strategy process in small software product businesses, or *strategic release management* is a continuous and paced activity conducted by a cross-functional team for making key product strategy decisions in small software product companies. The *framework of key product strategy decisions* guides *in setting the scope* in terms of *what decisions the strategic release management process should address* in small software product businesses. While these management areas are not equally topical to all companies at a given time, all of them should be considered in the product strategy of a company.

6 Discussion

This chapter closes the study. The research questions and the research problem are answered, the contribution of the study is outlined, and the usefulness and limitations of the framework as well as its construction process are evaluated. The thesis is concluded with highlighting directions for further research.

6.1 Answering the Research Problem

It is now time to address the research problem and the research questions set in chapter 1. The methodology and the steps taken during this study to answer the research problem are illustrated in Figure 9 and Figure 10 and explained below.

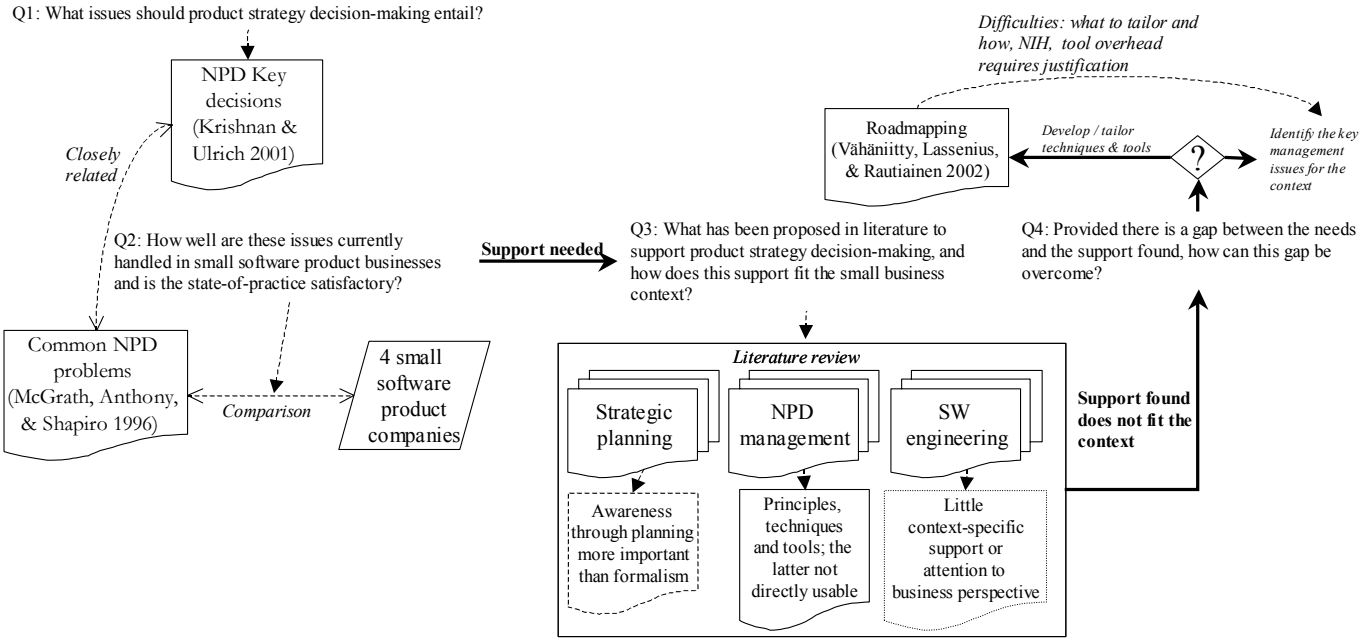

Figure 9 Identifying the need for a context-specific breakdown of product strategy

This study suggests that product strategy decision-making in small software product business can be supported by *identifying what issues must be accounted for in the strategic management of their product development* (research problem). This was seen as the most fruitful route to take because of *small software product companies' characteristics*, the *nature of the difficulties they face* in this area (2)[7] and the *missing support from existing literature* (3).

Towards this end, *creating a framework outlining the key decision areas in formulating and enacting product strategy* was undertaken (4).

[7] The numbers in parenthesis denote the respective research questions

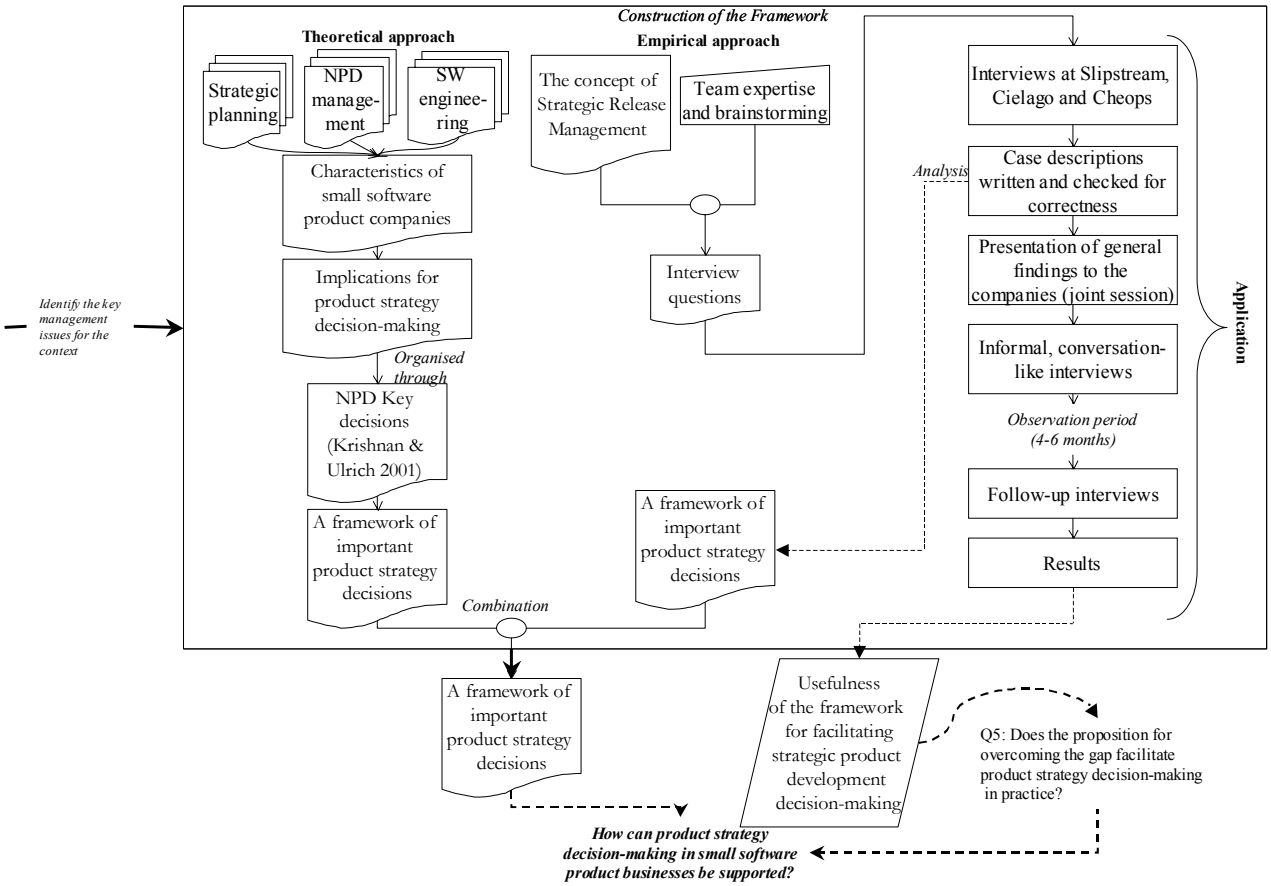

Figure 10 Constructing the framework and answering the research problem

The contents of the framework have been collected based on the understanding of the needs of small software product companies based on literature and empiria. In the framework, the *six key management areas* in small software companies' product strategy decision-making are *Organisation*, *Portfolio Management*, *Requirements*, *Development Strategy*, *Technology* and *Quality Strategy* (1). The naming and grouping used in the framework is a compromise between usability (descriptive naming, low hierarchy) and mutual exclusiveness (each element should be found only once and from a logical place) with the context of small software product businesses determining how possible conflicts between these were resolved. While not all of these management areas are equally topical to all companies at a given time, they should all be considered when deciding about the product strategy of a company.

Conducting an evaluation of three company's product strategy decision-making practices using a version of the key product strategy decisions framework *helped raise the awareness* of the companies of their problems and challenges, and *yielded tangible and relevant improvement suggestions*. Over a follow-up period of four to six months, most of the identified problems and challenges as well as the improvement suggestions had been acted on (5).

6.2 Contribution of the Study

This study adds understanding to product development decision-making in the context of small software product businesses. To date, existing literature has mainly discussed new product development from the perspective of large companies and software development with the focus on customer-specific development, leaving the development of software products in the context of small companies unaddressed.

In this work, we discussed the differences of new product development in small software product companies compared to the situations most commonly described in literature, identified key decisions in managing product development and tailored them to the small software product business context. The main result of the study was a framework of the most important product strategy decisions tailored to small software product companies. The following discussion further describes the contribution of the study by listing the potential uses for the constructed framework:

A definition of product strategy in the context of small software product businesses. The key product strategy decision areas framework supports product strategy decision-making by helping the key persons acting in multiple and sometimes even contradictory roles and responsibilities to maintain a holistic perspective under the pressures from different stakeholders.

Evaluation and improvement of product strategy decision-making practices. The framework can be used to raise the awareness of the key issues in product strategy decision-making of the companies and provide suggestions for improvement. If the areas of the framework have tangible and working real-life counterparts in the company, the product strategy process of the company is most likely in good shape. The framework can also be used as a blueprint to re-engineer the product strategy process of a company or help in building one from scratch.

Evaluation and improvement of software engineering management practices. Based on the case studies, an evaluation of the 'product strategy-level' as defined by the framework yields improvement suggestions beyond the level of detail addressed by the framework itself. Thus, in addition to evaluating the comprehensiveness of a company's product strategy process, the framework also helps improve a company's software engineering management practices through providing an outline of important issues. Based on the experiences from the case companies, the framework was very useful for getting an overview of how a small software product company operates for process assessment and improvement purposes.

6.3 Evaluation of the Research

Answering the research problem. The research problem and the research questions were answered satisfactorily.

The construction process of the framework. The benefits from the parallel approaches to constructing the framework are that extracting insights based on work with real case companies could be started immediately, and experiences of the usefulness of constructing

a framework defining important product strategy level areas in small software product companies were gained early on. The latter also made the validation of the idea behind the framework and its intended function feasible within this thesis. Additionally, not clinging to a single framework definition early on in the study was likely to enhance the comprehensiveness of the final framework. The drawback was that multiple versions of the framework had to be maintained simultaneously and presented in this thesis, and that the case company descriptions could not be made according to the final version of the framework defined in chapter 5. In summary, the approach made the research process and the reporting of the results more complicated but was likely to yield more reliable and comprehensive results.

Actions taken by the case companies during the follow-up period. Because of limited personal communication of the author with the companies during the follow-up period, it is difficult to state with full confidence what part the actions taken by the companies during the follow-up period could be attributed to this study, in other words, the joint identification of the problems and challenges and the presentation of the improvement suggestions after the initial interviews. Also, for Slipstream and Cielago, the fact that only a couple of people (one and two persons, respectively) were interviewed may have given a more positive view of the state-of-practice at the companies than would have been gotten by interviewing other personnel also.

6.4 Directions for Future Work

Based on what was learned during this study, several areas for future research seem promising.

6.4.1 Exploring the Relationship of Business Models and Product Development Processes and Communicating the Business Perspective to the Development

While it is generally understood that a company's product development process should support its intended way of doing business, explaining this relationship has received very little attention in literature.

The key idea behind the research effort this study is a part of is that different software companies produce different kinds of products for different customer groups, and the approach for creating software should fit the company's overall way of conducting business. By understanding the possibilities and constraints set on the product development process by the business model, software process improvement can be focused on the essentials from the business perspective and thus improve product quality and profitability. (Pyhäjärvi, Rautiainen, & Itkonen 2003) If the 'company's way of doing business' is conceptualised as the set of its business models (Rajala et al. 2001), this research space can be illustrated as follows (Figure 11).

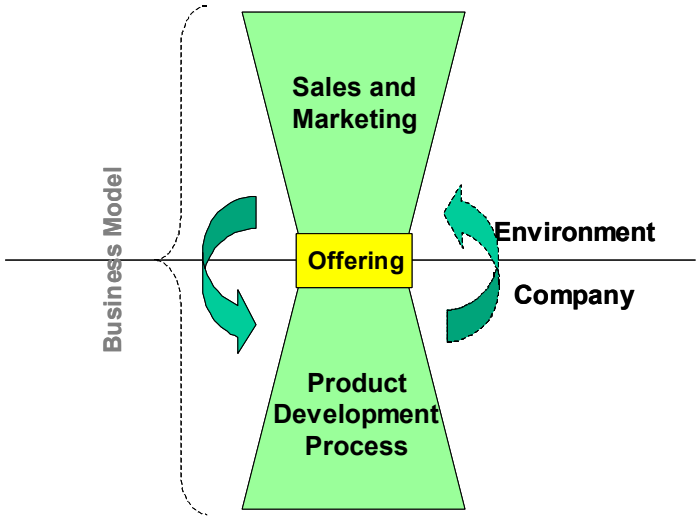

Figure 11 Balancing a company's product development process to support the other components of its business model

The author proposes that the framework lists essential decisions through which strategic considerations of the company should be reflected in the company's product development process. For example, business needs provide constraints and requirements for future releases' contents, timing and quality, and these can be supported by the process through making adjustments to the development model, requirements processes and testing practices used. If some of the decision areas outlined by the framework are being neglected, the company should ask itself why, and whether something can be done about it. Also, if some decision areas are more important than others because of the business the company is in, the software engineering management practices should reflect this as well.

Because the key decision areas of the framework constructed in this thesis are interrelated, changes in one area sets constraints and requirements on how the other areas can and/or should be organised. For example, the need to release a product with a near-zero tolerance for defects poses requirements on how testing should be organised, and this in turn sets constraints on the development model (for example, the length of the release cycle). A hypothesis is that in theory the degree and causality of these interactions and dependencies could be derived from the business model.

Thus, identifying how the company's business environment and desired way of conducting business should be reflected in its new product development process could be approached by exploring the relationship of the key decision areas in contexts with different business priorities. This could shed light on what kind of development processes are suitable for different business models.

6.4.2 Using the Key Decisions Framework for Self-Evaluation and Dissemination of Good Practices

In principle, the framework constructed and presented in this thesis can be used by the companies for self-evaluation and process improvement purposes. While evaluating the

usefulness of the framework from this perspective is out of the scope of this thesis, such experiences are of interest to be studied and reported in the future.

A possible approach to exploring this would be to make a group of small software product companies prepare a presentation of how they manage their product development using the key decisions framework or parts of it as a checklist on what issues the presentation should cover. The companies would then make a presentation of their own practices in a joint workshop, and the experiences and insights gained both in preparing for the workshop and during it would be recorded to improve the framework and assess its value in self-assessment and communication.

Also, descriptions based on the framework could be used to spread knowledge about real-life management practices. As shown in this thesis, creating such descriptions most likely leads to suggestions on how to further improve the process for the participants as well.

6.4.3 Identifying and adopting existing techniques to help with product strategy decision-making

The hypothesis of the author is that the framework assists in selecting, combining and tailoring existing product strategy decision-making methods, techniques and tools. This is because the framework helps identify and conceptualise the issues that need to be addressed and covered in product strategy decision-making. This is to be explored in future research.

6.4.4 Modelling and Instantiating Product Strategy Processes

While the issues a product strategy process should address have been examined in depth, the process itself, in other words, how companies should organise for making these decisions has received relatively little attention.

Clearly, not all of the management areas are equally topical to all companies at a given time. Thus, while their relative emphasis and frequency as part of the product strategy process varies over time and with changing business conditions, the described areas should all be addressed by the product strategy process, or strategic release management (Rautiainen, Lassenius, & Sulonen 2002), as it has been called in the small software product business context. However, while the key decisions framework provides a comprehensive blueprint of what should be taken into account in the strategic release management process of a small software product company, the set of most important decision areas to be included as part of the process is not constant. If the attention given to different decision areas varies depending on the business, the management practices already in place and the overall state of the company, *how should a strategic release management process be instantiated in a given situation*? In this area there is need for further research, and the question and its relationship to the product development process using the *cycles of control* framework (Rautiainen, Lassenius, & Sulonen 2002) is illustrated below in Figure 12.

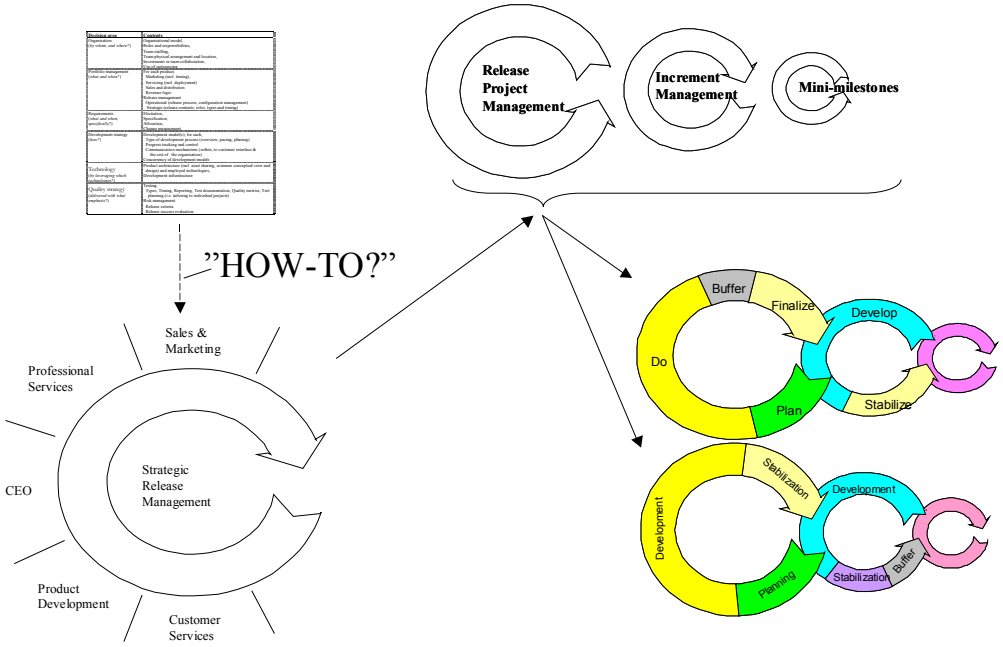

Figure 12 From the key decisions to the development process(es) used

References

Afuah, A. & Tucci, C. L. 2002, *Internet business models and strategies - text and cases* McGraw-Hill Higher Education.

Artto, K., Martinsuo, M., & Aalto, T. 2001, *Project Portfolio Management - strategic management through projects* Project Management Association Finland.

Bays, M. E. 1999, *Software Release Methodology* Prentice-Hall.

Beck, K. 2000, *eXtreme Programming eXplained* Addison-Wesley.

Beck, K. & Fowler, M. 2001, *Planning eXtreme programming* Addison-Wesley.

Berry, M. 2002, "Strategic planning in small high-tech companies", *Long Range Planning*, vol. 31, no. 3, pp. 455-466.

Boehm, B. W. 1988, "A spiral model of software development and enhancement", *IEEE Transactions on Software Engineering*, vol. 21, no. 5, pp. 61-72.

Bosch, J. 2002, *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach* Addison-Wesley .

Brodman, J. G. & Johnson, D. L. "What small businesses and organizations say about the CMM", pp. 331-340.

Brooks, F. P., Jr. 1995, *The Mythical Man-Month: Essays on Software Engineering*, 20th anniversary edition edn, Addison-Wesley, Reading, MA, USA.

Brouthers, K. D., Andriessen, F., & Nicolaes, I. 1998, "Driving blind: strategic decision making in small companies", *Long Range Planning*, vol. 31, no. 1, pp. 130-138.

Brown, N. & Eisenhardt, K. M. 2002, *Competing on the Edge : Strategy as Structured Chaos* Harvard Business School Pr.

Carlshamre, P. & Regnell, B. 2000, "Requirements Lifecycle Management and Release Planning in Market-Driven Requirements Engineering Processes", IEEE International Workshop on the Requirements Engineering Process: Innovative Techniques, Models, and Tools to support the RE Process, 11th Int'l. Conference on Database and Expert Systems Applications (DEXA 2000), Greenwich UK, pp. 961-965.

Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., & Natt och Dag, J. 2001, "An Industrial Study of Requirements Interdependencies in Software Product Release Planning", 5th IEEE Int'l. Symp. on Requirements Engineering (RE'01), pp. 84-91.

Carmel, E. & Becker, S. 1995, "A process model for packaged software development", *IEEE Transactions on Engineering Management*, vol. 42, no. 1, pp. 50-61.

Cater-Steel, A. P. "Process improvement in four small software companies", pp. 262-272.

Clarke, C. J. & Varma, S. 1999, "Strategic Risk Management: the New Competitive Edge", *Long Range Planning*, vol. 32, no. 4, pp. 414-424.

Cockburn, A. 2001, *Agile Software Development* Pearson Education.

Cockburn, A. 2002, *Agile software development* Pearson Education.

Condon, D. 2002, *Software product management - managing software development from idea to product to marketing to sales* Aspatore Books.

Cooper, R. G., Edgett, S. J., & Kleinschmidt, E. J. 2001, *Portfolio Management for New Products*, 2nd edn, Perseus Books.

Cusumano, M. A. & Selby, R. W. 1995, *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets and Manages People. The Free Press* The Free Press.

Cusumano, M. A. & Yoffie, D. 1998, *Competing on Internet Time: Lessons from Netscape and its Battle with Microsoft* The Free Press.

DeMarco, T. & Lister, T. 1999, *Peopleware - productive projects and teams*, 2nd edn, Dorset House Publishing.

Demirörs, E., Demirörs, O., Dikenelli, O., & Keskin, B. "Process improvement towards ISO 9001 certification in a small software organization", pp. 435-438.

Eisenhardt, K. M. & Sull, D. N. 2001, "Strategy as simple rules", *Harvard Business Review* no. 1, pp. 107-116.

Fayad, M. E., Laitinen, M., & Ward, R. P. 2000, "Software Engineering in the Small", *Communications of the ACM*, vol. 43, no. 4, pp. 115-118.

Ginsberg, M. P. & Quinn, L. H. 1995, *Process tailoring and the software capability maturity model*, Software Engineering Institute, CMU/SEI-94-TR-024, ESC-TR-94-024.

Grünbacher, P. "A software assessment process for small software enterprises", 23rd EUROMICRO Conference '97 New Frontiers of Information Technology, Budapest, Hungary, pp. 123-128.

Hoch, D., Roeding, C., Purkert, G., Lindner, S., & Müller, R. 2002, *Secrets of Software Success: Management Insights from 100 Software Firms Around the World* McKinsey & Co.

Humphrey, S. W. 2002, *Winning with software* Pearson Education.

Iansiti, M. & MacCormack, A. 1997, "Developing products on internet time", *Harvard Business Review*, vol. **75**, no. 5.

Jennings, P. & Beaver, G. 1996, "The Performance and Competitive Advantage of Small Firms: A Management Perspective", *International Small Business Journal*, vol. 15, no. 2.

Johnson, M. K. 2002, "Release Management 101: The Basics", *Crossroads News* no. November 2002.

Kamsties, E., Hörmann, K., & Schilch, M. 2002, "Requirements Engineering in Small and Medium Enterprises", *Requirements Engineering*, vol. 3, no. 2, pp. 85-86.

Kaner, Cem, Bach, J., & Pettichord, B. 2002, *Lessons learned in software testing - a context-driven approach* Wiley Computer Publishing.

Kappel, T. 2001, "Perspectives on roadmaps: how organisations talk about the future", *IEEE Engineering Management Review*, vol. 29, no. 3, pp. 36-48.

Kautz, K., Hansen, H. W., & Thaysen, K. 2000, "Applying and adjusting a software process improvement model in practice: the use of the IDEAL model in a small software enterprise", ACM, Limerick, Ireland, pp. 626-633.

Kelly, D. P. & Culleton, B. 1999, "Process improvement for small organizations", *IEEE Software* pp. 41-47.

Kontio, J. 2001, *Software engineering risk management - a method, improvement framework and empirical evaluation,* PhD Tech., Helsinki University of Technology.

Kostoff, R. N. & Schaller, R. R. 2001, "Science and Technology Roadmaps", *IEEE Transactions on Engineering Management*, vol. 48, no. 2, pp. 132-143.

Kotler, P. 1997, *Marketing Management – Planning, Analysis, Control, and Implementation*, 5 edn, Prentice Hall International.

Krishnan, V. & Ulrich, K. T. 2001, "Product Development Decisions: A Review of the Literature", *Journal of Management Science*, vol. 47, no. 1, pp. 1-21.

Laamanen, K. 2001, *Johda liiketoimintaa prosessien verkkona - ideasta käytäntöön* Suomen Laatukeskus Koulutuspalvelut Oy.

Laitinen, M., Fayad, M. E., & Ward, R. P. 2000, "The Problem with Scalability", *Communications of the ACM*, vol. 43, no. 9, pp. 115-118.

Laryd, A. & Orci, T. "Dynamic CMM for small organizations", Sweden, Umeå University, Department of Computer Science.

Linder, J. C. & Cantrell, S. 2002, "Five business-model myths that hold companies back", *IEEE Engineering Management Review* no. 3, pp. 26-31.

Lissack, M. & Roos, J. 2001, "Be coherent, not visionary", *Long Range Planning* pp. 53-70.

Marco, I. & MacCormack, A. 1997, "Developing products on internet time", *Harvard Business Review*, vol. **75**, no. 5.

McBreen, P. 2002, *Software craftmanship* Addison-Wesley.

McCarthy, J. & Gilbert, D. 1995, *Dynamics of Software Development* Microsoft Press.

McCarthy, J. & McCarthy, M. 2002, *Software for your head - core protocols for creating and maintaining shared vision* Pearson Education.

McGrath, M. 2000, *Product Strategy for High Technology Companies* McGraw-Hill.

McGrath, M., Anthony, A. R., & Shapiro 1996, *Setting the PACE in product development* Butterworth-Heinemann.

Megginson, W. L., Byrd, M. J., Scott, R., & Megginson, L. C. 1997, *Small Business Management - An Entrepreneur's Guide to Success*, 2nd edn, The McGraw-Hill Companies Inc.

Mello, S. 2002, *Customer-centric product definition - the key to great product development* American Management Association.

Miles, M. B. & Huberman, M. A. 1994, *Qualitative data analysis: an expanded sourcebook*, 2nd edn, SAGE Publications.

Mintzberg, H., Ahlstrand, B., & Lampel, J. 1998, *Strategy Safari : A Guided Tour Through the Wilds of Strategic Management* Simon & Schuster.

Moore, G. 1991, *Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers* HarperCollins Publishers.

Moses, J. "Learning how to improve effort estimation in small software development companies", IEEE Computer Society Press, CompSac International Conference, Oct 2000, pp. 522-527.

Nambisan, S. 2001, "Why Service Businesses are not Product Businesses", *MIT Sloan Management Review* pp. 72-80.

Naumanen, M. 2002, *Nuorten teknologiayritysten menestystekijät* SITRA.

Nunes, N. J. & Cunha, J. F. 2000, "Wisdom: a software engineering method for small software development companies", *IEEE Software*, vol. 9-10, pp. 113-119.

Otoya, S. & Cerpa, N. "An experience: a small company trying to improve its process", Proc. Software Technology and Engineering Practice STEP '99, pp. 153-160.

Paulk, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. 1993, *Capability maturity model for software, version 1.1* CMU/SEI-93-TR-024, ESC-TR-93-177.

Penny, D. A. 2002, "An estimation-based management framework for enhancive maintenance in commercial software products", IEEE International Conference on Software Maintenance (ICSM 2092), pp. 122-130.

Pihlava, S. 1996, *A Process Improvement Experience in Small PC Software Companies,* Helsinki University of Technology.

Pol, M., Teunisen, R., & van Veenendaal, E. 2002, *Software testing - a guide to the TMAP approach* Pearson Education Ltd.

Pyhäjärvi, M., Rautiainen, K., & Itkonen, J. "Increasing Understanding of the Modern Testing Perspective in Software Product Development Projects", Hawaii International Conference on System Sciences 2003 (HICSS-36).

Rajala, Rossi, Tuunainen, & Korri 2001, *Software Business Models: A Framework for Analysing Software Industry*, Tekes.

Rautiainen, K., Lassenius, C., & Sulonen, R. 2002, "4CC: A Framework for Managing Software Product Development", *Engineering Management Journal*, vol. 14, no. 2.

Rautiainen, K., Lassenius, C., Vähäniitty, J., Vanhanen, J., & Pyhäjärvi, M. "A Tentative Framework for Managing Software Product Development in Small Companies", HICSS-35.

Regnell, B., Beremark, P., & Eklundh, O. "A market-driven requirements engineering process - results from an industrial process improvement programme", Springer-Verlag.

Russ, M. L. & McGregor, J. D. 2000, "A Software Development Process for Small Projects", *IEEE Software*, vol. 17, no. 5, pp. 97-101.

Sawyer, P. 2000, "Packaged Software: Challenges for Requirements Engineering", 5th International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ 2000), Stockholm, Sweden (June 2000).

Schein, E. H. 1999, *The Corporate Culture Survival Guide* Jossey-Bass, San Francisco.

Schwaber, K. & Beedle, M. 2002, *Agile software development with Scrum* Prentice-Hall.

Scott, G. M. 2000, "Top priority management concerns about new product development", *IEEE Engineering Management Review*, vol. 28, no. 1, pp. 5-12.

Smith, J. A. 1998, "Strategies for start-ups", *Long Range Planning*, vol. 31, no. 6, pp. 857-872.

Sommerville, I. 1996, *Software engineering*, 5th edn, Addison-Wesley Publishers Ltd.

Sutton, S. M. 2000, "The Role of Process in a Software Start-up", *IEEE Software* no. July/August 2000, pp. 33-39.

Thomke, S. & Reinertsen, D. G. 2001, "Agile Product Development: Managing Development Flexibility in Uncertain Environments", *IEEE Engineering Management Review*, vol. 28, no. 1, pp. 37-50.

Vähäniitty, J., Lassenius, C., & Rautiainen, K. "An Approach to Product Roadmapping in Small Software Product Businesses", Quality Connection - 7th European Conference on Software Quality (ECSQ2002), Conference Notes, Center for Excellence Finland, Helsinki, Finland.

van der Heijden, K. 1996, *Scenarios: the art of strategic conversation* John Wiley & Sons.

van Ommering, R. 2001, "Roadmapping a Product Population Architecture", European Software Institute.

Vuornos, L. 2002, *Improving Software Development Process in Small High-tech Companies: Case Smartner Information Systems Ltd.,* Helsinki University of Technology.

Ward, R. P., Laitinen, M., & Fayad, M. E. 2000, "Management in the Small", *Communications of the ACM*, vol. 43, no. 11, pp. 113-116.

Wiegers, K. 1999, *Software Requirements* Microsoft Press.

Yin, R. K. 1994, *Case Study Research - Design and Methods*, Second edn, Thousand Oaks: Sage.

Zahra, S. A. & Bogner, W. C. 1999, "Technology strategy and software ventures' performance: exploring the moderating effect of the competitive environment", *Journal of Business Venturing*, vol. 15, pp. 135-173.

Zeller, A. 1996, *Configuration management with version sets - a unified software versioning model and its applications,* PhD Tech, der Technischen Universität Braunschweig.

Appendix A: Questions Used in the Initial Interviews

This appendix contains the questions used when conducting the initial semi-structured interviews of this study in Finnish.

1) Terminologia (Concepts and terminologyy
Mitä sana 'tuote' teillä tarkoittaa?
Mitä sana 'tuotejulkaisu' (tai vastaava käsite) teillä tarkoittaa?
Mitä sana 'tuoteversio' (tai vastaava käsite) teillä tarkoittaa?
Mitä sana 'ominaisuus' (feature tai vastaava käsite) teillä tarkoittaa?

2) Resursointi ja organisaatio (Organisation, roles and responsibilities)
Miten olette organisoineet tuotekehityksenne?
- Kuinka kauan tämä jako on ollut voimassa?
  - Kuinka usein se muuttuu?
- Ketkä päättävät vastuujaosta?
  - Missä määrin vastuujaosta voi sopia tilannekohtaisesti eri tasoilla?
- Miksi tuotekehitys on organisoitu näin?

Onko teillä mielestänne sopiva osaaminen T&K-porukassa?
Ostatteko ulkoistettuja palveluja?
- Mitä?
- Miksi?

Mikä on rekrytointitarve?
Millainen on rekrytointitilanne?
Mitkä asiat koette haasteellisiksi tuotekehityksen organisointiin liittyen?

3) Tuotevalikoima, tuotelinjapäätökset ja kokonaistuote (Product mix)
Mitä tuotteita myytte tällä hetkellä?
Miten tuotevalikoimasta päätetään?
Kenen vastuulla on lopullinen päätös?
Ketkä päättävät mitä tuotteita myydään?
Mitä (kehittämiänne) tuotteita ette enää myy?
Mitä tuotteita kehitätte parhaillaan?
- Kehitättekö täysin uusia tuotteita?

Mitä tuotteita jatkokehitetään?
- Miten päätätte mitä tuotteita lähdetään kehittämään?

Onko jonkin tuotteen kehitys joskus pistetty jäihin tai lopetettu kokonaan?
- Miksi?
- Ketkä osallistuivat päätöksentekoon?

Missä vaiheessa tuotekehitystä uutta tuotetta tai uusia ominaisuuksia aletaan myymään/markkinoimaan?
- Miksi silloin?

Mitä tuotteen myymiseen ja toimittamiseen sisältyy?
- Myyttekö muutakin kuin ohjelmistoa?
  - Mitä tähän sisältyy?
  - Ketkä sen hoitavat?
- Teettekö asiakaskohtaista työtä joka vaatii ohjelmistokehitystä?
  - Millaista?
  - Ketkä tekevät?
- Teettekö muuta asiakaskohtaista työtä?
  - Mitä?
  - Ketkä tekevät?
- Ketkä päättävät millaiseen asiakaskohtaiseen työhön ryhdytään?

Mitä tuotteita ja tuoteversioita tuetaan?
- Ketkä päättävät tästä?

Mitkä asiat koette haasteellisiksi kokonaistuote- ja tuotevalikoimapäätöksiin liittyen?
4) *Tuoteominaisuudet, tuotejulkaisun sisältö ja vaatimustenhallinta (Release contents and requirements engineering):*
a) Tuotejulkaisujen hallinta ja tuoteominaisuudet

Millainen suunnitelma teillä on tällä hetkellä tulevista tuotejulkaisuista?
Ketkä ovat vastuussa tästä suunnitelmasta?
Miksi julkaisut tehtiin juuri noihin aikoihin?
Tietystä julkaisusta (tai joukosta julkaisuja):
Ketkä tekevät lopullisen päätöksen, että tuote on valmis julkaistavaksi?
Miten päätös tehtiin?
Mihin päätös perustuu?
Milloin julkaisulle asetetaan ensimmäisen kerran tavoitepäivämäärä?
Voiko tavoitepäivämäärä muuttua tämän jälkeen?
Miten ja miksi?
Ketkä osallistuivat päätöksentekoon?
Mitä tuotejulkaisuja olette tehneet tähän mennessä?

Miten keräätte ideat uusiksi tuoteominaisuuksiksi?
Ketkä osallistuvat ominaisuuksien ideointiin?
Esittävätkö asiakkaat toivomuksia?
Miten näihin reagoidaan?
Ketkä osallistuvat ominaisuuksien keräämiseen?
Dokumentoidaanko ominaisuuksia jotenkin?
Miten?
Priorisoidaanko ominaisuuksia tuotejulkaisujen ulkopuolella?
Voiko ominaisuuksien toteuttamiseen liittyä muutakin kuin ohjelmistokehitystä?
Kerätäänkö nämä eri tavalla?
Miten arvioitte tuotejulkaisuissa onnistuneenne...
Suunniteltujen tuoteominaisuuksien saavuttamisessa?
Asiakastyytyväisyydessä tuoteominaisuuksien suhteen?
Mihin nämä arviot perustuvat?

b) Tuotejulkaisujen, vaatimusten ja muutosten hallinta
Miten päätätte mitkä ominaisuudet tuotejulkaisuun toteutetaan?
Onko toteutettavat ominaisuudet priorisoitu tuotejulkaisua tehdessä?
Voiko tuotejulkaisun sisältö muuttua tämän jälkeen?
Millaisia nämä muutokset voivat olla?
Mistä ne johtuvat?
Mitä tehdään jos yrityksen sisältä tulee toivomuksia tuotteeseen kesken julkaisun tekemisen?
Mitä tehdään jos asiakkaalta tulee toivomuksia tuotteeseen kesken julkaisun tekemisen?
Mitä vaihtoehtoja on näiden toivomuksien kohtalolle?
Voivatko muutokset tuotejulkaisun sisältöön johtua muusta kuin edellä mainituista tekijöistä?
Huomioidaanko vaatimusmuutoksen vaikutus tuotekehitysaikatauluun tai resursointiin?
Miten?
Miksi näin?
Onko jonkin tuotejulkaisun kehitys joskus pistetty jäihin tai lopetettu kokonaan?
Miksi?
Ketkä osallistuivat päätöksentekoon?
Tehdäänkö asiakaskohtaisia tuotejulkaisuja?.
Milloin ja miten tällaisiin variantteihin tehtävät lisäominaisuudet ja virhekorjaukset siirtyvät ”perusjulkaisuun”?
Mitkä asiat koette haasteellisiksi tuotejulkaisujen sisältöön ja vaatimustenhallintaan liittyen?

5) Ohjelmistokehityksen rytmitys ja tuotejulkaisujen ajoitus (Development pacing, release timing and release types):
Miten olette organisoineet uusien tuotejulkaisujen tekemisen?
Onko tuotejulkaisujen tekemiseen tähtäävä ohjelmistokehitys ositettu jotenkin?
Miten? (esim. rakennetaanko tuotejulkaisut projekteina? onko välietappeja? Mitä välietapeissa tapahtuu?)
Onko eri tyyppisiin julkaisuihin tähtäävä kehitys erilaista?
Limittyvätkö ’projektit’?
Kuinka monta ’projektia’ on käynnissä samanaikaisesti?
Minkä tyyppisiin julkaisuihin tähtääviä?
Onko tuotejulkaisujen tekemisen lisäksi muuta tuotekehitysresursseja vaativaa toimintaa?
Miksi?
Ketkä tämän osituksen/rytmityksen ovat suunnitelleet?
Onko teillä eri toimintatapoja eri tuotteiden kehittämiseen?
Miksi?
Entä saman tuotteen eri osien kehittämiseen?
Miten tuotejulkaisujen etenemistä seurataan? (esim. kokoukset, mittarit, tuotokset, jne...)

Miten arvioitte tuotejulkaisujen tekemisessä onnistuneenne...
- Aikataulupidossa?
- Budjettipidossa?
- Mihin nämä arviot perustuvat?

Mitkä asiat koette haasteellisiksi ohjelmistokehityksen ositukseen liittyen?

6) Tuotteen rakenne, arkkitehtuuri ja teknologiapäätökset (Product architecture and technology):

Kerro lyhyesti (pää-)tuotteenne rakenteesta:
- Piirrä kuva tärkeimmistä osista ja niiden välisistä yhteyksistä.

Koostuuko tuote esimerkiksi ytimestä ja sen päälle rakennettavista palveluista tai lähes irrallisista ohjelmista?
- Kuinka itsenäisesti eri osia voidaan kehittää?

Miten arkkitehtuurisuunnittelu tehdään? Kuka, mitä, milloin?
Mitkä ovat keskeisimmät teknologiat, joita on käytetty tuotteidenne toteutuksessa?
- Miten näihin teknologioihin on päädytty?
- Ketkä osallistuivat päätöksentekoon?

Mitkä asiat koette haasteellisiksi teknologiapäätöksiin ja tuotearkkitehtuuriin liittyen?

7) Testaus ja riskienhallinta (Testing and risk management):

Mitä testausta tehdään?
(Esim. Yksikkötestaus, integrointitestaus, järjestelmätestaus (toiminnot), käytettävyystestaus, suorituskyky, alfaversiot omassa käytössä)?)
- Missä vaiheessa?

Ketkä tekevät testausta?
Suunnitellaanko testauksen
- aikataulu?
- resurssit?
- testitapaukset?
- mittarit?

Ketkä ovat vastuussa testauksen käytännöistä?
Milloin testauksen käytäntöjä suunnitellaan?
Missä vaiheessa eri tasoista testausta tehdään?
Raportoidaanko testaukseen liittyen jotain?
- Mitä ja miten?

Onko olemassa laatutavoitteita?
- Millaisia?

Käytättekö asiakasta hyväksi laadunvarmistuksessa?
- Esim. Betatestaus, hyväksymistestaus tilaustuotteissa?

Mitkä ovat (eri tyyppisen testauksen tavoitteet)?
Mihin testauksenne käytännöt perustuvat?
- Jos tuote- tai bisnesriskeihin,
  - millaisia tai mitä nämä ovat?

Miten arvioitte tuotejulkaisuissa onnistuneenne...
- Laatutavoitteissa?
- Asiakastyytyväisyydessä laadun suhteen?
- Mihin nämä arviot perustuvat?

Mitkä asiat koette haasteellisiksi testaukseen liittyen?

Appendix B: Case Descriptions

In this appendix, descriptions of the interviewed companies, their products, a rough outline of their business model and their product development organisation and process with a focus on product strategy level decision-making are given. When asked for, the companies' management pointed out several challenges facing them in product development.

In the sections describing the perceived and observed problems and challenges, the observations made by the researchers based on the case interviews are *in italic*. Note, that the present tense used in the case descriptions refers to the status quo in 4-5/2001.

Case 1: Slipstream Ltd.

Slipstream, founded in 1996 and re-focused to its current operations in 1999, develops software to package and stream video and audio over the Internet. The products can be used for example for corporate communications (internet and intranet), web portals, banner ads and video e-mail campaigns. At the time of the interview Slipstream employed a total of 30 people.

Organisation, Roles and Responsibilities

Structure. Of Slipstream's 30 employees, 19 are in product development, 5 in sales, 3 in management and 3 in customer support. The company is owned by its founders, venture capitalists and other private investors. The development work is conducted in release projects, and a product manager and a technical project manager are appointed from the development function to head a project. There is also a nominated test manager.

Roles and responsibilities. Product managers are responsible for the feature set to be implemented in the project and end-user documentation, such as manuals and online help. The project manager is responsible for progress tracking, how the product is designed and the features implemented, internal product-related documentation and possible post-release work in the form of a. bug-fix release. The test manager of the company is responsible for ensuring that the end product meets its specifications, test documentation and in general, all test-related effort. The project manager leads teams of developers who do the actual implementation, and the developers consult the product manager directly on feature details when need arises. The senior product manager is responsible for reporting the progress of all ongoing development to the head of product development and the management team. While currently resource spending is a top-management decision, it is planned that the heads of R&D functions start meeting weekly to make joint decisions on allocating resources to projects.

Product Strategy and Portfolio Management

Overview of the products. Slipstream is currently developing and marketing two products that jointly form the offering of the company; one for basic videos and the other for creating and streaming synchronised rich media presentations. Currently, Slipstream plans to

start developing a version of the product for mobile platforms and to package the two current products into one in order to simplify their maintenance, development and marketing. Launching a mobile version of the product has been considered for a longer period of time, and the technological changes in the external environment as well as the superior efficiency of Slipstream's technology are currently viewed to support this move.

Sales and distribution. A typical delivery of the product starts with a visit to the potential customer. The customer may download a version of the product for a 30-day evaluation, and during that time they are allowed to free support and some training, which in most cases amount to less than a couple man-days of effort. After the evaluation phase, the deal is either closed or off. The products are sold mainly to service providers, either directly by Slipstream (4/5 of total sales) or through agents (1/5).

Revenue logic. For both products, 80% of sales turnover comes from licensing, and a maintenance fee (which includes support in the form of a helpdesk and all updates to the purchased product version) accounts for the rest. The products are developed in-house with the help of a handful of outsourced developers. The technical core of the product was licensed from an outside research institute with small profit sharing of Slipstream's sales, and it is being jointly developed further. Slipstream's revenue totalled to approximately 170 000 € in 2001, with costs from product development so far exceeding this. Subcontracting accounted for about half of the total cost of product development. Although the relative amount of subcontracting will decrease in 2002, details about the budget were not available for the purposes of this study.

Servicing and deployment. Customer service and sales agents handle all support required by the customers, and in case of problems, contact the head of development who delegates the task to someone. In special cases, Slipstream has provided additional services to go with the product such as handling the service provider's role (possibly with partners) in order to deliver the end product directly. Also, in some cases customer-specific development effort may be undertaken but these have usually been propagated back to the main product either to a main, or a service release. A third kind of customer-specific servicing, which has been undertaken only once to date, is to deliver a part of the source code and let the customer make changes to it in an OEM fashion. At the time of the interview it seemed that there are pressures towards special cases of this kind actually becoming more common in the future. According to the interviewed person, it is very important to limit customer specific product releases.

Release strategy. Slipstream specifies the overall direction where its products are going with a roadmap document in the format of presentation slides. The purpose of a roadmap is to define the company product development strategy in an easily presentable manner and they are used for communication purposes both internally to the product development and also in the meetings of the management team. The document briefly lists and describes the current products, their future major and minor releases, possible future releases and larger updates to the product platform along with their planned release dates on a monthly scale. Also, the major features to each release are listed. The company board and the management team are responsible for these decisions. The roadmap does not provide indication on the planned developments' dependencies or resource usage. The vice president of engineering proposes the roadmap to the company's management and board who review, possibly revise

and accept the plan. Roadmaps are revised according to company's whenever deemed necessary by the top management. Typically roadmaps are frozen for the next 6 months.

Target release dates for a project are set when the project is launched. However, these change almost always, and schedule slips range up to 50% of the original calendar time of the project (based on a set of approximately ten releases after 1999). The two most common reasons for schedule slip are the addition of major new features late in the project and underestimating feature efforts. Decision to change a release date is made by Slipstream's management team, and "made as soon as it is realised that the deadline is not going to be met". There are no strict guidelines to when marketing product features for an upcoming release starts, and in practice this depends on the sales persons and the situation at hand. The expressed philosophy behind release timing is "to get it to the market as soon as possible".

Maintenance of older product releases is handled by letting the customer download the latest version, as this in all encountered cases was known to remedy the situation. While this is contrary to the license pricing policy, the quality level of the older releases has made this the state of practice. In the future, the current products are to be frozen and supported by making maintenance releases when necessary, possibly including a maintenance fee if their quality level turns out to be adequate and the company's business focus stays in those products. Excluding these cases, product development can concentrate on working on new product releases. Traditionally, bug fixes were made on a shorter notice based on customer feedback, or the development team itself noticing something requiring immediate attention. The project personnel who were working on the next version of the same product were responsible for making such maintenance releases.

Decision-making. The management team makes decisions affecting the product mix, and in general, all such decisions are taken to the company board for final acceptance. In practice, the senior product manager presents available options and characterises them with respect to resource and schedule implications to the management team, with the company board making the final decision, if needed.

Requirements engineering

Elicitation. Ideas for new products and features come from existing customers, sales and marketing, company management and board and the product development. Also, an important source of new product ideas is competitor surveillance. The interviewed product manager estimated that a third of such new ideas come through the customer support team, another third from the customers through sales and marketing, and the rest from inside the R&D team.

Specification and allocation. Traditionally, product managers have documented requirements for the product release as requirement specifications in the beginning of the release project. At this point, requirements come from the feature database, but also from customers, the customer support team and product managers. Based on the requirements specification, functional specification documents, project plans and project breakdown documents featuring effort estimates (as workdays) are written by the project manager, but not all of these documents are kept strictly up-to-date during the course of a release project. Require-

ments specification documents are typically between 20 to 50 pages in length, and their goal is to provide a detailed specification of the product to make the development easier and communicate the features to the product development. In the past, sales and marketing were required to read through requirements specifications but it was noticed that the benefit from this exercise varied greatly depending on the person. More recently, separate summaries of these documents have been written for sales and marketing by product managers, and the contents of the upcoming releases are reviewed with them with the help of a slide show.

Change management. Potential scope changes surface usually in weekly project meetings. If something is to be left out, the product manager must ask for permission from the sales or the management team, depending on how significant the feature to be dropped is. Significant changes are characterised so that alone, they would account for a minor release. A rule of thumb is that changes affecting all but 'nice-to-have' features or the project schedule must be taken to the management team for approval. Decisions and changes on how particular features are implemented can be made without the consent of sales or the management team. Adding features to the scope of an ongoing release project requires less reviewing, although the management team must approve considerable additions. Feature requests from customers are handled similarly to any other features, except they escalated to the management team more frequently, and a common point in the agenda of a management team meeting is to review these. Also, the management team may issue changes to release projects' scope. A change log of the project scope is updated weekly to a slide set describing the outline, schedule and progress of the project, and the functional and requirements specifications are updated immediately after the project meetings. Decisions to kill release projects are made by the company board and the management team.

A new requirements process. A commercial tool has recently been taken into use to help record and maintain a database of features and known bugs.

Elicitation. The product managers have joint responsibility of keeping the feature database up-to-date. Also, the customer support personnel have a key role in collecting feature ideas and improvements to the product and entering these into database. Below is an outline of a planned but not yet deployed requirements management process utilising the new tool. Besides pure software features, requirements can concern for example the look and feel of the user interface, presentation layout templates (or "skins") in the software. However, the process for managing these kinds requirements has not been yet been thought of.

Allocation. First, customer support and sales evaluate whether the features in the database will benefit all of the customers or just one (or a few) of them, how valuable they are for the concerned parties, and how important these parties are for Slipstream, and whether a concrete deal is involved. Product managers meet regularly to assign priorities to the features based on the sales' and customer support's evaluations.

Specification. For an upcoming release project, the product management team takes the prioritised features, and together with project managers estimates their efforts and the amount and type of resources required to implement the features, and based on these decides whether the feature should be included to the release at hand. When the costs and resource constraints for implementing the features are estimated, the sales and customer

support can re-prioritise the feature set. The final set of features is reviewed and approved by the management team.

Change management. After the approval of the feature set, no details had yet been planned on how to do within-project change management and re-prioritisation. The interviewed person noted, that the details of the process have not yet been planned to the point with respect to the other parts either.

Development model

Overview. Slipstream has a waterfall (Sommerville 1996) -like model of release project phasing, with the implementation phase divided into three sections starting from the most important features. In practice this model has not been strictly followed, and the division of the implementation phase based on feature priority is problematic due to feature interdependencies. Also, the phasing of the project and the emphasis on the various waterfall stages varies depending on whether the project is a first release of a new product or an update to older one. Currently, release project duration vary from two to six months. The intention is that initial releases are allowed more calendar-time while updates on older products are to be made on a more frequent schedule. However, no explicit rationale (besides generic fulfilling customer expectations and sales promises) was mentioned to be behind this approach.

Pacing. Early versions of the final product (called builds) are made approximately weekly during the projects. Traditionally, the number and contents of such builds has not been specified beforehand, but this approach has been piloted in one release project, and the experiences were encouraging. While the project had to be terminated due to pressing resource needs in other projects, planning the project build-by-build was seen by the interviewed product manager as the future way to manage release projects because of several perceived advantages. First, it forces the project manager to consider the entire project already in the planning phase, second, it makes the feature dependencies explicit and third, it makes progress tracking easier because of frequent milestones.

Concurrency. Currently, there are three simultaneous release projects, with two making new versions of older products and one launching an entirely new product. The goal is that the developers would be involved only in a single project at a time, and when serious conflicts in resource needs arise, the decisions can be escalated directly to the management team, which issues priorities for the ongoing release projects. Currently, no explicit rules on how to establish release project priorities exist.

Decision-making. The interviewed senior product manager and one of the project managers are responsible for the general model according to which product development is organised. A recent addition to the product development model made by the new head of R&D is to allocate and link the time and resources for other products' maintenance releases in advance to a certain project milestone.

Types of development effort. There are two basic types of development work at Slipstream, the release projects and core engine development. Although the engine development team has its own project plans and other design and specification documentation, it was at the time of the interview not fully integrated into the overall process thinking. The core is developed together with a third-party research institute, and the separation in the way of

working stems from cultural differences and the fact that establishing a common development process has not been crucial to date.

Release criteria and quality metrics. The criteria for releasing a version of the product are that all defined test cases have been successfully completed, no critical bugs are open, and all features of 'must' priority must be finished. The project and product managers review defect data at weekly project meetings. The final decision to release a product is made jointly by the leading product manager, the test manager and the respective project and product managers. In principle, the release projects are schedule-driven except for features of the highest priority class, but in the past there has, in practice, been room of movement here. Typically, a product is not released unless all 'must' features have been implemented.

Progress tracking. The projects' progress is tracked in weekly project meetings. In these meetings, the development project status is reviewed together with the project team, project manager, product manager and test manager. These kinds of meetings have been held even daily (or at best twice a day) during a critical phase in the project. There has been discussion about whether to use hour reporting or other time tracking based mechanisms on to better grasp the estimated versus actual effort spent in release projects, but so far have not yet been taken into use effectively. The interviewed senior product manager felt he had a good knowledge of what product development people were working on at a given time. Also, it is not uncommon for the developers to work from slight to moderate overtime.

Technology Selection and Software Architecture

Employed technologies. Relevant technology decisions from the perspective of the senior product manager were the programming language and other key technologies used, the environment the product supports, and selecting those development tools that essentially shape the development process, such as a bug reporting system. The currently utilised technologies stem from the products business requirement that they should support as many computing infrastructures as reasonably possible. In other words, the client product (written in Java) should work in most common web browsers, and the desktop application (written in C++) in Windows environment. While requests to port the latter to Macintosh or Solaris have been received, the decision to stick to the current environment has been made by the management team.

Conceptual view of the architecture. Although the interviewed product manager was not closely familiar with how the product and its parts were referred to technically, he claimed that conceptual models of the products exist and are utilised by the developers to decide on the product architecture and take advantage of synergies between various parts that are similar or the same between the products.

Architecture design. Product architectures are designed to accommodate larger future enhancements (i.e. architecture does not only support the next immediate product release but also longer term releases).

Testing and Risk Management

Types of testing. The types of testing conducted at Slipstream are system testing, integration testing, module testing, and 'ad hoc testing'. System testing means testing the entire

product from a black-box perspective according to test case specifications, and is in principle conducted for each build. The total amount of system testing depends loosely on the stage of the release project, and respectively, integration testing refers to system testing conducted for the first builds, that is, making sure all the developed SW modules function together. According to an estimate made by the interviewed person, the vast majority (80%) of such testing is regarded as system tests. Module testing means the testing of a specific piece of functionality. Although there has been some effort to spread the practice of test automation, it is currently utilised only in stress and stability tests of the technical core of the product before its integration. Also, for the current release projects, a beta testing program, in which selected customers pilot the release under development and provide feedback, has been arranged. System testing for one of the products has so far been outsourced because of the complexity of the environment it has to work in. The effort of running the tests (estimated to one man-month) necessary for a release is thought to disturb too much the development rhythm if it were to be handled in-house.

Test process. Integration and system testing are performed by Slipstream's test manager, the test team, sub-contractors, and in some cases developers as well. All defects found in this way are reported to the bug reporting system. Also, sometimes personnel outside R&D may be asked to use the products to get general feedback on the products. As soon as the first version of the requirements specification for the project is finished, the test manager allocates a member of the testing team to make preparations on how the features to be implemented can be verified. Based on this, issues such as when testing is started, how much testing should be done and how long running the set of test cases will take are evaluated.

Test documentation. Writing or updating the test case specification starts when the requirements specification is finished. System and integration testing produces bug reports and test logs, which specify the results of running the test cases and the environment in question. Before a release is made, the information from the project's test logs is compiled into a test report document. All test-related data except whether particular test cases were completed successfully on some platform at a given time and what percentage of test cases has been run in a given environment can be fetched from the defect management tool. There are some 300 – 500 documented test cases for each product.

Beta-testing. For the releases currently under development, a beta-testing program has been arranged with selected customers.

Quality metrics. There were no explicitly defined goals or desired levels of effort for conducting different types of testing. According to the interviewed person, the current efforts are based on "improving the testing state-of-practice to enhance the perceived product quality".

Release success evaluation. The success of product releases is in principle evaluated based on how well the schedule was kept, whether the targeted scope could was achieved, what was the quality level at the time of the release, and post-release customer feedback. While there is currently no process for conducting this analysis, some thought has been given to possible metrics and these are to be used for the upcoming releases. Difference between the targeted and actual release date and whether any 'must' features have been left out are to be monitored as well. Measuring the quality level could be done by comparing the total number of

bugs found during the project with the number of bugs open at the time of release, with test metrics such as the number of test cases have been completed versus all test cases, and how many bugs reached and were found by the customers versus Slipstream's own testing. The most important measure of customer satisfaction would be so called field-failure-rate (FFR). This measure indicates how many of the customers evaluating the product does not want to close the deal for product quality reasons. At the time of the interview there was no data on the FFR or reasons behind non-closed deals for the current releases on the market.

Perceived and Observed Problems and Challenges

Organisation, roles and responsibilities. Slipstream feels that main challenges for the product development organisation lie in acquiring the necessary competence for the new product type.

The motivation behind the new organisational model for the R&D was most likely the need to clarify roles and responsibilities. However, this rationale or its effects of the new structure on organising product development were not entirely clear – for example, the interviewed person mentioned "in practice, the new structure can not change the way we have worked so far very much". The open question is then what the idea behind the organisational change from project-based to functional was and how has it worked in practice?

Product mix. The most important challenge in deciding about the product mix is to identify the right focus for the products and features to be developed and balancing this with limited resources of a small company. This challenge is emphasised in developing the new product.

The challenges in deciding about the product mix are evident from the company's history during which major changes to the focus have been relatively common. However, if customer-specific effort is likely to become more common in the future, this should be accounted for in both in terms the product mix and the development models used.

Requirements engineering. Although the company has gone through a learning curve in its current products, the interviewed product manager felt that improving feature effort estimates is a major challenge. This is especially true for the new product whose development is about to begin, because it features new technologies.

Based on the interviews, the requirements process seems quite document-heavy. Also, it features some duplication of information, for example besides the requirements specifications themselves, there may be abstracts of requirements specifications for sales purposes and, slide show versions of the specifications, for example. The interaction of the new requirements database and the specification document is unclear. Examples are updating effort estimations (to database, to the document, or both?) and the timing of writing the requirements specification document. Also, if a transition from the requirements documents to using the database is to be made, how will it be done? Also, there should be a mechanism to make changes to project scope in a controlled fashion.

Development model. According to the interviewed person, there have been too many concurrent release projects in the past, and releases have been attempted too often.

Improving effort estimations without tracking effort spent is problematic. The success of effort estimates can only be made against the set calendar dates, and when these are missed, there is typically little hard data to evaluate the causes for this. Also, while release dates are changed "as soon as it is realised that the target dates are not going to be met", this may, in practice be quite late with respect to the original project schedule.

The author presumes that the problem stems from not having strong enough mechanisms to estimate the completion date at a given time, such as time tracking and a history of effort estimating. The waterfall-shape model has not served the purpose of structuring the development as much as hoped for. For example, different project types and ways of working in them are not explicit, and whether their timing is schedule- or feature-driven is unclear. Even though in the future, the goal is to start conducting schedule-driven release projects, the waterfall-type process model does not support this very well.

Technology selection and software architecture. The most pressing challenge is the in-practice platform dependence of today's Java environment. There are slight and less slight variations between browsers, operating systems and computing platforms, and in some cases, the same solutions simply do not function properly across the variety. This has forced Slipstream to optimise the code to circumvent these kinds of problems. As the Internet environment as well as possible input formats the product has to support are getting more and more diverse, these issues continue to persist in the foreseeable future. A continuing challenge in architecture design is to quickly develop a robust but still flexible structure. Both careful architecture design and more rapid get-the-release-out style approaches to architecture design have been tried, and currently, the architecture-first approach seems the better choice ("demos, prototypes and hacks excluded", as put by the interviewed person).

Testing and risk management. The main challenge is perceived to lie in ensuring that the product works across the multiple environments it is supposed to support. Even with just the most important environmental combinations, the current set of tests amounts to approximately one man-month of effort. Also automating testing on the module level is not perceived to do much good here since the part of the product, which does not suffer from compatibility problems, has traditionally been of good quality.

The distinction between system and integration testing seems unclear, as does the interaction of beta testing program with ongoing development. Also, it is likely that the relationship of the new feature database to testing documentation has not yet been thought of.

Case 2: Cielago Ltd.

Founded in late 2000 by a group of industry professionals from several high-tech companies, Cielago develops and markets devices enabling wireless short-range network capabilities to industrial applications. At the time of the interview, Cielago's employees totalled 15.

Organisation, Roles and Responsibilities

Overall structure. Of Cielago's 15 employees, 8 are in product development and the rest are evenly divided (4+3) in sales and management. The founders and venture capitalists own the company.

R&D structure. Cielago's product development is organised as hardware and software teams of 3 and 4, respectively, and is supervised by one person.

Roles and responsibilities. Both teams have a team leader. One person from the software team is responsible for managing and conducting customer-specific work requiring technical understanding of the product, including installing and delivering the products and training. This is because the sales personnel lack the necessary technical competence and there are no personnel dedicated to customer service only.

One person in the hardware team is responsible for testing and certifying the product. He reports directly to the head of R&D. Besides these responsibilities, each person in the development team has in principle personal primary and secondary focus areas, but the responsibility matrix shown was not consistent with the interview, and most likely not up-to-date.

Two of the software development teams are located in western Finland away from the main office in Lappeenranta. The head of R&D has a leadership-type role and works in the customer interface. Having strong technical skills, the head of R&D was previously responsible for directly managing both of the development teams and the customer interface, but the workload turned out to be too hard for one person to handle. The current organisation structure is seen more successful. Based on the interviews, the head of product development is mainly responsible for successes in sales and marketing efforts as well.

The company's management team consists of the head of R&D, head of operations, head of sales and the CEO. The head of R&D decides on product development organisation.

According to the interviewed persons, the responsibilities within software development are roughly based on the individuals' competencies, and this dictates who will work on what. No formal process exists for allocating responsibility to the software development team for producing product features.

Product Mix

Overview of the products. Cielago's products are based on an open technology standard and are used to build host-less man-to-machine, machine-to-machine, or machine-to-network wireless platforms. Some example applications of Cielago's solution are wireless meter reading, condition monitoring and automation in industrial settings, wireless point-of-sales systems, scanners, ATMs and credit card readers, security and alarm systems, and in various lifestyle electronics such as GPS receivers, fitness equipment and mobile accessories. Cielago's goal is to position itself as a design house with a third party responsible for the hardware manufacturing of their end projects.

Sales, distribution, servicing and deployment. Instead of ready-to-deliver product versions, Cielago's product portfolio is currently best characterised as a group of potential solutions of varying levels of sophistication aimed at slightly different types of customers, from OEM manufacturers to integrators and consultants. Originally, Cielago's product offering was a device aimed for OEM manufacturers who needed a wireless connectivity module in their own solution. The product consists of a development toolkit including hardware and software components, and requires customer-specific hardware and software development effort to create the final OEM wireless application module for the customer's solution.

To utilise this first product offering, a customer has to allocate resources to develop the hardware and software required by the final product, and it takes from 12 to 18 months from closing the deal for the final solution to reach the customer's market. In practice, many prospects have been unwilling to make this kind of commitment to a small start-up company, and thus the concept is problematic.

The next step in refining Cielago's offering was for the company to take responsibility of a part of the work originally allocated to the customer in order to decrease the order-to-delivery cycle to 6 months. However, the current need for the cycle length is 1-3 months, and this has lead to the need to deliver near-complete solutions to pilot various customer-requested applications.

In addition to the device based on the original concept and its development toolkit, Cielago currently markets both an intermediate version and the possibility of building pilot systems for the customers. Currently, the software component is intended and designed to be the same for the entire range of Cielago's offerings.

Revenue logic. Customer-specific training, installation and application development projects have so far been a significant source of revenue for Cielago when compared to license sales, and roughly one-third of sales involves customer-specific development work. The person who closes the deal decides on whether the deal will include customer-specific development effort. In the future, revenues are hoped to consist of product licensing and customer service with an 80-20 ratio, respectively. Cielago's revenue for its first operation year in 2000-2001, was under 1M€ and for the year 2002 it is estimated to be 2-3 M€. Relative costs of product development to revenue were roughly 2/3 during the start-up year and are expected to decrease 50% during the second year.

Release strategy. The head of R&D writes the schedule for high-level requirements on a quarterly scale as product roadmaps and communicates them to the product development teams, who attempt to understand how to fulfil those requirements.

The roadmaps provide no indication of resource usage or the effort the features' implementation should take. Also, the dependencies of the features are not visible from the roadmaps, which was explained to be quite natural since the features shown in the roadmap are often implemented as distinct programs on top of the operating system and thus have no direct dependencies.

In 2002, two releases of the software platform are to be made, a minor one in Q2 and a major one in Q4. Bug fix releases are to be made on the fly as the need arises, and the time required from them is taken from developing new features.

As a rule of thumb, the head of R&D estimated that a bug fix release would be a job the size of one man-week. As there have so far been no defects serious enough to make a separate maintenance release, there is little experience of how this practice will scale up in the future.

Two minor releases have been made so far. These were made on-demand to provide customers with features that were originally scoped out of the respective major releases, and at the same time, minor improvements and bug fixes were made. In addition to the existing basic technology and its development kit, four new solutions with varying sophistication levels are to be created during 2002, with all but of one of them already under development. In addition to software development, making a release involves performing regression testing, updating the documentation and making a release note.

According to the head of software development, there is a detailed software development action plan for the first two quarters of 2002 and at least for him, the features in the

roadmap are traceable to this document. In practice the document is not actively used to track or control the development progress.

Decision-making. Decisions affecting the product mix are in practice the result of the head of R&D making go-decisions to develop certain pieces of hardware based on the signals he receives through his work at the customer interface. While the head of R&D is responsible for characterising and presenting all product mix options and decisions to Cielago's management team and board, he in practice makes the final decisions as well.

The competitor information from the sales and marketing personnel has not in practice influenced the decisions because they are perceived too abstract for taking concrete action in the form of new requirements. As the sales personnel have not provided information for the R&D on what the customers would like, Cielago's sales works in a technology-push fashion. To help with these issues, a new sales director has been hired in Q1/2002 to promote a more systematic way of making decisions.

The interviewed persons considered the product mix decision process to be less than optimal due to its heavy reliance on the intuition of a single person and lack of documented rationale. While proceeding with developing and properly documenting the pieces of hardware according to the current roadmaps entails considerable costs, no explicit or even generally understood rationale (such as descriptions of the product concepts and their target customers or the decision criteria behind selecting these) for product mix decisions exists.

According to the interviewed persons, the basic problem stems from the length of the hardware development cycles – if a customer ordered a specific new product not directly based on the set of offerings available today, the delivery could be promised three quarters from now. As the potential ways to use the technology are in practice unlimited and difficult to predict, making a set of technology probes based on very little information has seemed unavoidable. Also, the perceived need for flexibility is evident from the products' 'more-versatile-than-necessary' Linux-based software platform.

Requirements engineering

Elicitation. Feedback from various stakeholders such as the CEO, potential customers and the product development team can introduce new features to the release under development. In addition to the head of R&D working in the customer interface, some requirements also come from the ongoing customer-specific projects handled by a member of the software team, and although they are usually discussed, the fact that there's no process for extracting or recording these requirements may causes 'random disappearance' of potentially valuable customer feedback.

Specification. The head of R&D bears the responsibility for release contents and high-level requirements and consults Cielago's management team when decisions about including new feature ideas to the products have to be made. Although some documented forms of requirements management have been practiced, there's currently no explicit process for collecting and managing ideas for new product features. With the possible build-up of a technically more competent customer service team in the future, a more formal requirements engineering process is seen as both necessary and beneficial. Currently, the only require-

ments document is the high-level list of features for the two upcoming product releases, and the management team reviews this quarterly.

Allocation. The interviewed persons felt, that up until recently, the 'must-do' requirements for the product have been quite clear, but believe that harder decisions with respect to product features will be faced in the future.

Change management. No defined procedure on how to select features that have to be left out in order to make a release deadline (i.e. how to do triage) exists, and in practice, the developers have made these kinds of decisions independently as well.

Development model

Overview. Product development at Cielago does not have a clear process or rhythm. Some attempts to pace the development effort have been made by setting release deadlines, but so far these have not been made, usually due to the hardware components being late. An R&D process model was designed and proposed for the company by an outside research institute, but never took off. The model was based on Boehm's spiral model of software development (Boehm 1988), and was considered to be too out-of-context and theoretical to provide basis for action.

Development rhythm in practice. In the past, hardware development schedules and the actual set of hardware features delivered by the hardware development team have been erratic at best. These experiences have resulted in the current pacing, where the software development team keeps choosing the next feature from the high-level feature list to be worked on until the hardware team finishes. When the hardware component is ready, a target release date is decided on, implementation of new non-critical software features stops, and testing begins. Over time, visibility to the hardware development process and increased, and the process itself has matured. The interviewed persons estimated that in the near future, it would be possible to start major new software development efforts without having real hardware to back it up at the beginning. However, this is also a question of limited resources, and until there are new employees to help with customer servicing, this is not seen feasible.

Progress tracking. Both the software and hardware teams have recently been using a weekly review to track what has been completed, what problems emerged, what will be worked on next week and what issues currently constitute the team's high-level action plan. Although based on the interviews the developers work considerable amounts of overtime, it stems from their enthusiasm about the work. Mechanisms to establish control on the product development work such as estimating and tracking effort by hour reporting have so far not been considered necessary. One of the interviewed developers pointed out, that a major portion of the software development work is porting the software to the various hardware platforms and considered this inherently less predictable and difficult to plan in advance than regular software development. Estimating features' efforts, tracking the actual time spent, or estimating the amount of work still to be done for a certain release are not conducted.

Technology Selection and Software Architecture

Employed technologies. The technologies have been chosen by the personnel in product development, with the expressed criteria being the price tag and the developers' previous

experience. Most technologies in the hardware have been involved right from the start and are based on previous experience and the company's strategy with respect to main components. Also, certain software technology decisions have been decided on by the company board based on the trends in the larger IT environment, and these are to be incorporated future releases.

Conceptual view of the architecture. Based on the interview and the internal material shown, Cielago's R&D organisation has a common understanding of the product's structure and a language to refer to its parts.

Testing and Risk Management

Types of testing. At Cielago, testing refers mainly to verifying and validating the hardware component of the product. In the hardware development process, considerable effort is used to verify the hardware design before it goes into production. The other types of testing conducted are so-called production testing, debugging parties organised by the special interest group responsible for the wireless technology standard, joint testing sessions with customers or partners and ad hoc release testing of new software platforms. Also, certification testing by the special interest group ('SIG') of the technology is currently on the way. Production testing means going through the final hardware functionality. The reliability of the Linux-based software platform is regarded as excellent, and in practice the approach to testing it is basically to keep it running for long periods of time and see if something unexpected happens.

Test process, documentation, reporting and release criteria. The process of testing Cielago's products is not formally defined, and it has not been linked to business and product risks. A thesis on how testing at Cielago should be organised was written by a researcher from an outside research institute but is not utilised. There are no test case specifications, but test reports are being written. However, their usage was not discussed during the interviews. There are no defined release criteria, or explicit metrics for evaluating product quality. No explicit guidelines or practices have been defined for evaluating product the success of released products.

Perceived and Observed Problems and Challenges

Organisation, roles and responsibilities. Currently, Cielago's product development competence is perceived excellent. Instead, challenges are seen in building a technically competent customer service organisation with social skills to ease the development's workload and aligning the sales and marketing efforts with the product development and supporting the technical perspective of the product development team with a systematic approach to doing the actual development.

Growing the R&D team may prove to be quite challenging if no further effort to structure to the development work is made. This assumption is based on the fact that interaction between the R&D team and its only "outside contact" to date, the sales team, seems based on the interview to be working less than optimally. However, care must be taken in introducing process thinking to the organisation, as Cielago has fresh experience from an attempt to deploy an "irrelevant" model for product development.

Product mix. The basic dilemma regarding Cielago's offerings results from the length of the order-to-delivery cycle. The more sophisticated the end solution delivered by Cielago, the more effort and calendar time it takes from the company's small product development to produce it, and the more costly the bill-of-materials and manufacturing of the product. As this relationship is non-linear, selling and delivering application-specific solutions is "approximately 1000 times less effective" (as put by the head of R&D) from Cielago's perspective than just providing the basic technology and tools for OEM manufacturers to create their own solutions from. While the basic path towards utilising Cielago's core competence seems currently to be building requested pilot devices and letting the customers understand the benefits of the basic technology, the challenge is in identifying a niche where the threshold of customers adopting the basic technology and manufacturing application-specific solutions on their own is relatively low.

Requirements engineering. The most important challenges in requirements management are perceived to stem from the same source as those for the product strategy, i.e. the length of the order-to-delivery cycle and immature markets. As long as the product is a generic platform solution, the requirements are either technical must-haves, or cannot be specified at all because of missing domain knowledge. Another challenge is finding the right amount of control in elicitation and specification in order to direct the enthusiastic and innovative atmosphere of the product development teams into solutions with real market value.

Explicit business requirements for the products were missing, and a lot of decisions were made based on "hunches". Based on the interviews there seemed to be many half-conscious decisions regarding the products also. For example, the software platform used in the product line was mentioned to be well versatile beyond its current needs, but when asked why the technology had been chosen in the first place, only its price, ("free") and previous experience were mentioned. In other words, many of the decisions are technology rather than market driven.

Development model. So far, there has not been much pressure on getting any single release of the product to the market, and these issues have not been considered as especially challenging.

While hardware development constraints the rhythm and pacing of the product development, the development activities seem mostly event-paced. Mechanisms for progress tracking were non-existent. In practice, the development model has no clear shape, pacing and phasing are not employed, and informed early decisions based on progress tracking are most likely not possible.

Technology selection and software architecture. The most pervasive challenge in technology decisions is to optimise the hardware bill-of-materials against the required product platform functionality. As very different uses for Cielago's products are possible, a simpler and cheaper solution would in some cases do, but as a small company Cielago cannot stray from its core platform to create more cost-effective solutions for these customers. While these decisions have to be made very early on in the process of developing new products, the trade-offs in hardware price and functionality are quite well understood. The software has had to compensate for some hardware deficiencies, but even if software components have to be acquired from third parties for this purpose, the total addition to the product's bill-of-materials is minimal compared to hardware costs. Also, maintaining the same software platform for all future product variants is technically challenging.

Testing and risk management. Product quality is perceived good at Cielago. While the approach to testing is not very formal, the added value from test case specifications, reports and metrics does not seem worth the effort. The developers believe that with respect to the open source components of the software, the open source community helps take care of testing. The final quality of the products is in part yet to be evaluated by the market.

Although hardware testing was claimed to be done, information on what it consists of, how much effort is put in and at what time were left unanswered. Overall, it seemed that Cielago had not yet come face to face with quality problems, but whether this is because of extremely good quality or due to other factors remains a question.

Case 3: Cheops Ltd.

Cheops Ltd. was founded in 1996 and develops and markets software products for performance measurement and process management. At the time of the interview, the number of employees in the company totalled 100.

Organisation, Roles and Responsibilities

Structure. Cheops has approximately 100 employees of whom 40 are in product development (30 programmers) and another 40 in sales and marketing. The remaining 20 are evenly divided in management, customer support and IT support. The number of employees has grown roughly 100% during the past two years, and the expected future is some 10% for the ongoing year. Cheops' founders own nearly half of its stock and its employees 15%, leaving a quarter of the company's ownership to private investors, venture capitalists and other shareholders. The product development personnel are situated so that the majority reside at the second company office in western Finland. Both product managers, the head of R&D, one of the chief developers, four developers and one usability team member are at the Helsinki office.

R&D structure. Cheops' new product development consists of product management, project management, a pool of developers and a production team to test the products under development and perform various supporting functions in product development. The product development is headed on the strategic level by a product team and on the operative level by an operations team. The product team consists of two product managers, each responsible for one product, the head of sales support, and one person responsible for a significant customer segment, the two latter being so-called 'product evangelists', and responsible for representing the customer perspective.

Roles and responsibilities. The product managers 'own' the products in the sense that they are responsible for the contents of the products to the rest of the organisation. The product team is responsible for deciding about the products to be offered, where the product is going and what features will be included to which release, a rough allocation of development resources and release schedule. Decisions about the set of products to be offered are done jointly by the product team and Cheops' board.

The product team is helped by application specialists, who are responsible for taking care of non-software aspects of the product offering such as training packages and examples, slide

shows and online material according to the guidance by the product managers, and keeping an eye on the competitors.

The operations team consists of two project managers (one per product), the leader of the production team and a person responsible for technology and research. It runs the product development projects by making a detailed resource allocation from the pool of developers into development teams of two to five persons, and to the production team for the current and upcoming release projects. With the exception of a couple of key developers and a three-person core of the production team, the exact constitution of the development teams and the production team is changed from project to project. The production team handles system and release testing, running the automated builds each night as well as various support functions for both products. New recruits to product development personnel are usually assigned to the production team at first.

Both products have lead developers, who are responsible for supervising the design and implementation of the software. Additionally, one person is responsible for organising internal component development for both products, a team of four people specialises on user interfaces and documentation as well as supports the development teams in this work. One person is at the head of the entire R&D organisation.

Product Mix

Overview of the products. Currently, Cheops markets two products, both of which are developed in-house. The products are used to plan, implement, communicate and commit people to organisational strategies, objectives and business process improvement. The products or their application is not industry-specific.

Sales and distribution. The customers are large private and public sector companies. The products are marketed, sold, distributed and implemented by Cheops itself and approximately 100 retail partners, of which one-third are domestic. Retail business accounts for two-thirds of total sales. Partnerships with management consultants and system integrators are used to facilitate the delivery of the products, and even in the case of direct sales, a team separate from Cheops' R&D handles the delivery.

Revenue logic. Of sales revenue, 70% comes from licensing the products and 20% from maintenance contracts which include access to all product updates, helpdesk support, and the possibility to get fixes to possible critical bugs on a short notice. Many of the older customers do not have maintenance contracts at all, but for new customers, the licenses are in practice not sold without these. The final 10% comes from a mixture of installing the product, training and consulting, and in rare cases minor customer-specific tailoring. According to the interviews, in this kind of business, third-party consultants or integrators charge typically twice the costs of software licenses from their work, while for Cheops this work varies around 0,5-1,5 times the license cost. Retailers get a provision of the sales revenue. Cheops' revenue and profits in 2001 were 12M€ and 3M€, respectively, and for 2002 revenue was at the time of the interview estimated to grow by 30%. Costs of product development in 2001 were roughly 15% of the revenue. Both of Cheops' products are similar with respect to their revenue logic.

Servicing and deployment. Cheops sometimes assists in product installation and provides training and consulting. The R&D is not involved in providing these services. No customer-specific tailoring of the product is generally made. Currently, one large customer has a variant of the product from which copy protection has been removed for convenience reasons. However, so far there has been no need to propagate these kinds of changes back to the main product version.

Release strategy. Cheops aims their releases to take place at the end of a quarter. There are two seasons for higher sales, just before summer and in at the end of the third quarter, but no specific need is perceived to get into this rhythm with the releases because of product maturity. Earlier on in the products' life cycle, pressure on time-to-market was perceived higher. Also, the salesmen are now more focused on marketing the advantages of the product rather than specific new features. Major and minor releases are in principle schedule-driven, and one of each are generally made once per year for both products. Maintenance releases are made when the need arises, typically once per minor or major release. Features of the highest priority level may be communicated in advance to the customers and other external stakeholders.

Release planning. Cheops specifies the plans for its products with two kinds of roadmaps, both of which are documented as simple one-page slides. The first roadmap type shows the planned minor and major releases for both products on a quarterly time scale for one year ahead, and the second roadmap type (one per product) features tentative major releases for the next three years. For each release, the latter roadmap shows the main business goal and lists new key features and significant changes with respect to technology and architecture.

Thus, Cheops' roadmaps communicate the most important new features, their schedules on a quarterly scale, plus a rough indication of the amount of resources to be used through the release types. However, the resource implications of any single feature or their dependencies are not visible in the roadmap. The product managers are responsible for the roadmaps, and their decisions are guided by so-called 'winning criteria' and a general direction, established twice a year in a joint strategy meeting of Cheops' board, the product team and the head of R&D. Also, the 'winning criteria' and the general direction are refined monthly with the product team. The 'winning criteria' specify dimensions on which the products must excel their competitors.

Release types. There are three kinds of releases for the products, major releases, minor releases and bug fixes. The main difference between major and minor releases is that for a minor release, changes are typically small improvements contributing to usability of previously introduced new functionality and bug fixes, whilst for a major release new entities, features and ways to use the product are introduced. Maintenance releases are made to fix a critical bug or for improving the overall quality of the release by addressing several major bugs.

Decision-making. Several decisions to discard a product have been made in the past. Also, in one case, customer-specific modifications were made to an older product version.

Requirements engineering

Elicitation. Ideas for product features stem mainly from the customers, either directly or through the personnel working in the customer interface. These include the marketing personnel, helpdesk and the product managers. The developers are also responsible for providing some of the feature ideas. Although the application specialists of the product team are responsible for competitor surveillance, only few features have been made on the basis of this activity, and its role was not evident from the interviews. Earlier in the life cycle product, it was common that certain customers and their situation could directly affect release contents, but nowadays the focus has shifted from single customers to (perceived) needs of the market.

Specification. The product manager and helpdesk are responsible for entering new feature ideas into a feature database, and everybody can view the amount of new features entered from their personal dashboards. For new features, the business perspective and benefits to the customer must be entered, and it is also possible to write use cases and information about from whom the idea originated, but these are to some degree neglected initially. Thus, no quantitative data exists for evaluating the sources of the current feature base, and the database may contain features obscure to all but the original contributor. Also, only product features that are to be realised in the software are being entered into the database (as opposed to new services or other whole-product related improvements).

Allocation. Features in the database have a target release set for them. Although anyone entering new feature ideas can set the priority, only the product manager can set the target release date, and he will also adjust the priorities as he sees fit. In principle, product road-maps could be constructed bottom-up based on the feature database but this is not currently utilised.

Change management. When the contents of an implementation cycle (or the entire project) have to be changed, the project manager issues a change request to the product manager. Needs for scope changes result from changes in available resources, such as sick leaves, and from actual effort being larger than was originally estimated. Most of the change requests are made by the product manager, and concern new features or changes to existing ones. If the development team feels it can handle schedule deviations internally, the project manager is not consulted, and likewise, when the project manager feels that re-structuring of the development teams can compensate schedule deviations, the product manager does not have to be consulted.

Development model

Overview of the development models. All product releases are done as projects and so that any given moment there is a major release project for one product and a minor release project for the other one going on. There is typically a single development team working on a minor release, and several working on a major one. Traditionally, maintenance releases have also been the responsibility of the product development personnel and have caused serious problems keeping up with the schedule of the ongoing release project for that product. In the new process, the responsibility for maintenance releases belongs to the production team.

The new development model. At the time of the interview Cheops' product development organisation and process had recently been re-engineered from a more waterfall-like to an iterative and incremental process, and the new process had been tried out by one-half of the R&D organisation in a release project for the other product.

The re-engineering effort was originally inspired by the perceived need for developing new product versions faster. Its objectives were to make the development process more market-driven through flexibility, achieve better visibility into the state of the development project, improve quality through introducing testing earlier on and to promote better understanding of individuals' roles and responsibilities as well as their impact on development project success. The new process model and organisation were initially developed by one of the product managers together with a project manager, tested on a major release project, and improved through holding a feedback session with the personnel participating in the release project. This description is of the new product development organisation and process, and thus represents partly an ideal that at the time of conducting the interviews this is only partially deployed. This applies especially to project progress tracking, of which there was little practical experience at the time of the interviews.

Both minor and major release projects consist of preparing the project, a planning phase of 2-4 weeks, an iterative and incremental implementation phase with 4-5 cycles of 3 weeks each, and a phase of 2-3 weeks for finalising the release. At the beginning and end of each phase there are specific milestones, in which the project steering group, consisting of the project manager, the product manager and the head of R&D review the project.

When a new release project is prepared, the product team headed by the product manager reviews the feature database and proposes a prioritised set of features to be implemented. The next step is holding a project kick-off meeting involving the head of R&D, production team leader, product manager and the project manager to start the planning phase of the project.

In the kick-off meeting, main development areas suggested for the product release by the product team are presented, the amount of resources to be allocated to the project is decided on, and a preliminary schedule is made.

During the planning phase, the project manager goes through the features, revises their effort estimations together with the lead developer, creates the development teams for the project and together with the product manager discards some of the features (typically 50-75%) based on the calendar time and resources available. Here, 'features' denote functional or non-functional entities of arbitrary size, and their effort estimations range from one man-day to a man-month.

In the milestone that completes the planning phase, the project manager prepares a project plan showing the schedule of the release, the set of features to be implemented, their effort estimations and relative priorities, the number of iteration cycles and their objectives (with respect to features) and resources used. This is reviewed by the project steering group, which consists of the product manager, the project manager and the head of R&D. After this, the program manager basically handles independently all decisions not affecting the scope of individual release cycles (or the project), and consults the product manager in these cases.

The three-week implementation cycles have been divided into a two week coding and one-week stabilisation phase. For each cycle, a set of features has been allocated for implementation. After the coding phase, an alpha release is made for internal system testing and, bugs are fixed, and the next cycle is planned. The alpha release can also be made available to customers that are especially interested in trying out the new features implemented in the cycle for feedback. Traditionally, possible beta releases could be made so late in the process that in practice it was not possible to react to customer feedback for that release. In the new process, no features are planned for the last implementation cycle and thus it is possible to react to customer feedback, implement more high-priority features, or buffer against overruns from earlier cycles.

Phasing and pacing. At the milestone after the last implementation cycle, all promised features have been completed and the version should be ready for release, except for the system and release tests and finishing the user documentation, which has been started in the implementation phase and intentionally 'lagged one cycle behind' the actual implementation. After the final milestone, the production team prepares the version for release.

Project progress tracking. Traditionally, project progress has been tracked in weekly meetings, where development team leaders inform the project manager about the situation. Also, metrics of the number of completed features and defects have been available, as the release projects have neared their completion. In the new process this is changed so that at the end of each cycle, the production team reviews test reports and gives grades to how well new features are working. Also, two new project metrics dashboards are to be implemented.

Through the first one, project management can view information on the defects found, features implemented (versus target values for each feature priority class set during the planning phase) and whether project milestones were completed on time. Also, a dashboard for product managers for evaluating long-term success on a monthly basis is being planned. This dashboard would feature metrics on motivation (whether the personnel are satisfied with their tasks and whether they believe their team is going to achieve the goals set for them), on implemented process improvement ideas, measuring the amount of courses taken or books read by the personnel, and on the internal component development for the products. Both dashboards are to be taken into use during the next weeks for the upcoming release projects and have been reviewed with and accepted by the program managers.

Project progress tracking (effort estimation). Although coarse effort estimations (the alternatives provided by the feature and bug database are 'day', 'week' and 'month') for the features exist, actual effort spent is not tracked. The only explicit link between calendar time and man-hours is the fact that the personnel are assumed to be able to contribute 30 effective hours during a workweek. The sets of features to be completed in projects, as well as in the cycles, are based on agreements between the product and project manager, and project manager and development team leaders, respectively. Situational factors such as team composition and difficulty and interdependence of features are assumed to be accounted by the judgment and understanding of the product and project managers and the lead developers. The average programmer works regular workweeks while the managers tend on the average to work from slight to moderate overtime.

Phasing and pacing. In the traditional process there were internal and external deadlines for completing the release projects, and while the internal deadlines were usually overrun by roughly a month, the external deadlines were in most cases met due to buffer time reserved.

Release criteria. In the release project piloting the new process, the release was at the time of the interview estimated two weeks late due to changes in development team composition for the last implementation cycle. The product manager makes the final decision whether a product version is ready for release. The most important factors in deciding this is the overall level of quality, and how much the version suffers from leaving possible half-completed features out. The overall quality level for both products is now perceived to be at its highest so far during the products' life-cycles and because there are so many sold products on the market, the product managers are currently more willing to compromise schedule than quality.

Technology Selection and Software Architecture

Employed technologies. The most important technological decisions regarding the two products concern the supported environment, the technologies and tools used in developing the products and major revisions to the product architecture. A person with the title of 'technology and research' has been allocated the duty to keep an eye on the needs to change these, and her responsibility is to 'sell' new technological ideas to the product managers. Making technology decisions is the joint responsibility of the operations and the product team.

Conceptual view of the architecture. On the basis of the interviews, a common language for the R&D organisation to refer to the products, their parts and relationships to each other exists. The developers understand at least the basics of both products, and the product managers, having a background of participating in the development are familiar with this as well.

Testing and Risk Management

Types of testing. In the new process, testing efforts consist of system testing by the production team at the end of each implementation cycle, release testing conducted before an actual release is made, and ad-hoc testing conducted by the development teams as they code. System tests are based on test cases written by the development teams, which in turn are based on the functional specification. In the new process the production team evaluates and rates new features to encourage improving the quality of the implementation. Traditionally, testing has consisted of system and release testing at the end of the project. Also, load testing and 'smoke testing' were mentioned as recent additions to the set of testing approaches; the head of production team is responsible for the latter, and it means automatic user interface tests run nightly for a build to check that the main functionality is working. Project managers conduct intra-project risk analysis addressing issues such as feature dependencies, developer productivity, personnel risks and technology risks together with the product manager, the lead developer and the head of product development.

Rationale for the testing practices. In the interviews, no formal process for doing risk management was identified, and the significance of the practice is unknown. Also, there are no explicit link between possible business risks and the verification & validation practices

used, and establishing such a link is complicated by the fact that budgeting is not product specific. This is because in the past it has been difficult for the sales personnel to accurately estimate the relative amount of sales from each product.

Perceived and Observed Problems and Challenges

Organisational Model. The most pervasive of these in the near future according to the head of R&D and one product manager are to completely deploy and enforce the new product development process and to maintain a flexible, innovative and enthusiastic atmosphere in product development amidst the need for structure and control required by the increased size of the R&D organisation. Also, the R&D organisation must be convinced of the importance and value of the new organisational unit, the production team.

The escalation of decision-making in case of schedule deviations may cause that the problems that reach the product manager have already grown quite big due to optimism by the development team leader and then by the project manager. However, escalating all problems directly to product manager is not a good option either, striking a balance here is important. Because one of the interviewed product managers mentioned this as a problem, it suspected that in practice most problems (even those that could have been solved on the lower levels) are escalated to product manager and take his time and attention.

Product Mix. Two conflicting demands exist for developing the product offering further. To be able to compete against the offerings of larger companies the products' domains should be extended. However, fitting this goal to the limited resources of a relatively small company and maintaining the competitive advantage of the current focus is challenging. Another question is whether the currently generalist products should be focused on certain industries or domains. Both issues are currently very topical at Cheops.

Requirements. In deciding about release contents, the challenge is in gaining feedback on features' value as perceived by the customers. Inventing new features is easy, but there are no easy answers on how to add significant amounts of value to a mature product through incremental development. Additionally, making sure the product is going into the direction intended based on a list of implemented features is not straightforward. An issue in requirements engineering is proper capturing of new feature ideas and communicating them to the developers. Currently, new feature ideas entered to the database are sometimes sketchy, and a valuable feature may be overlooked in the process of evaluating and selecting features for implementation. Vague feature specifications cause confusion and rework during the actual implementation.

Development model. The most pervasive challenge in the near future according to the head of R&D and one product manager is to completely deploy and enforce the new product development process and to maintain a flexible, innovative and enthusiastic atmosphere in product development amidst the need for structure and control required by the increased size of the R&D organisation.

Another challenge in the new product development process as perceived by Cheops' management are that at this point are shortening the planning phase to two weeks (i.e. finding a good-enough level of initial planning – the details are bound to change anyway and overdoing the planning is a waste of time). Also, the developers are worried that the rhythm of the product development may be stressful because deadlines are always present during the

development. However, in the traditional process, new projects were often being stalled by old ones being late, and this in turn is thought to become easier as testing is spread more evenly within the project. The leader of the production team noted that there is little experience in making maintenance releases under the new development process, and thus it is unknown how they will fit in the practice. Also, one project manager viewed the poor accuracy of feature effort estimations and consequently inadequate project planning as the most serious obstacles for maintaining release project schedules.

To work properly, the new product development process requires that both of the parallel release projects strictly adhere to their planned pacing, and it is questionable whether this is possible in practice. For example, pacing so that both development projects don't require production team's efforts at the same time. And what happens when maintenance releases must be made to the products at the same time, if the production team has other responsibilities as well, or if the production team does not at a given time possess the required competence to make a certain hot fix? Also, it seemed likely that in practice, implementing more critical features, making up for being behind schedule, plus bug fixes will all be competing for the time of the last cycle, and to facilitate informed decisions, a good process for selecting how to use the time is necessary.

Quality. In testing, the most prominent challenges were in learning how to better utilise the potential of automatic testing and in revising and bringing the test case specification for one of the products up-to-date.

Like in the two other case companies, testing efforts were not explicitly based on identified business or product risks.

Technology. In product technology and software architecture, the main challenge lies in the relative maturity of the products. A piece of software can be re-worked and altered only so many times before the code becomes unreadable, and keeping up the quality is becoming increasingly challenging as time passes. Also the architecture poses limitations on what kinds of features can be added, and how development teams can efficiently work on different parts of the product concurrently. For the other product, there are pressures to rework the architecture, and although some kind of resolution must be reached in the near future, the problems are yet far from solved. The overall technological change has slowed down slightly from the perspective of Cheops' products, and this eases the situation somewhat. The effects of the problems in deciding about the product mix are visible here as an interviewed lead developer's comments on challenges in scaling up the product architecture.

If program code is to be rewritten in order to make it more maintainable, a rigorous approach to deciding what is to be rewritten and why is suggested.

Appendix C: Details from Constructing the Theoretical Framework

The results of organising the decision-making elements derived from the characteristics of small software product businesses (Table 7) using the list of key NPD decisions by (Krishnan & Ulrich 2001) (Table 1 and Table 2) are shown in Figure 13 below and explained in detail in this appendix.

To keep the construction steps reasonably tidy, definitions of the elements that are being restructured, added or considered to be added to the framework are provided only to the extent necessary. The definitions for the final set of elements of the framework can be found in chapter 5.

The decision-making elements derived from the characteristics of small software product businesses are in *red and italic*, and have been attached to similar issues in NPD key decisions by (Krishnan & Ulrich 2001). The key decisions in setting up development projects have been used to create the basic grouping. Those within-project issues that have direct counter-parts among the issues in Table 7 are considered relevant in the small company perspective have been included here as well and shown as green and underlined branches. Issues of considerable similarity in different branches have been pointed out by dashed red arrows.

Starting from the situation depicted in Figure 13 below**,** the goal is to reach a model as simple as possible in terms of minimised area interdependence (no need for dashed red arrows) (i), a comprehensive but as-small-as-possible amount of elements (i.e. total branches) with a preference of leafs over branches (ii), and a set of elements relevant from the perspective of a small company in the software product business (iii).

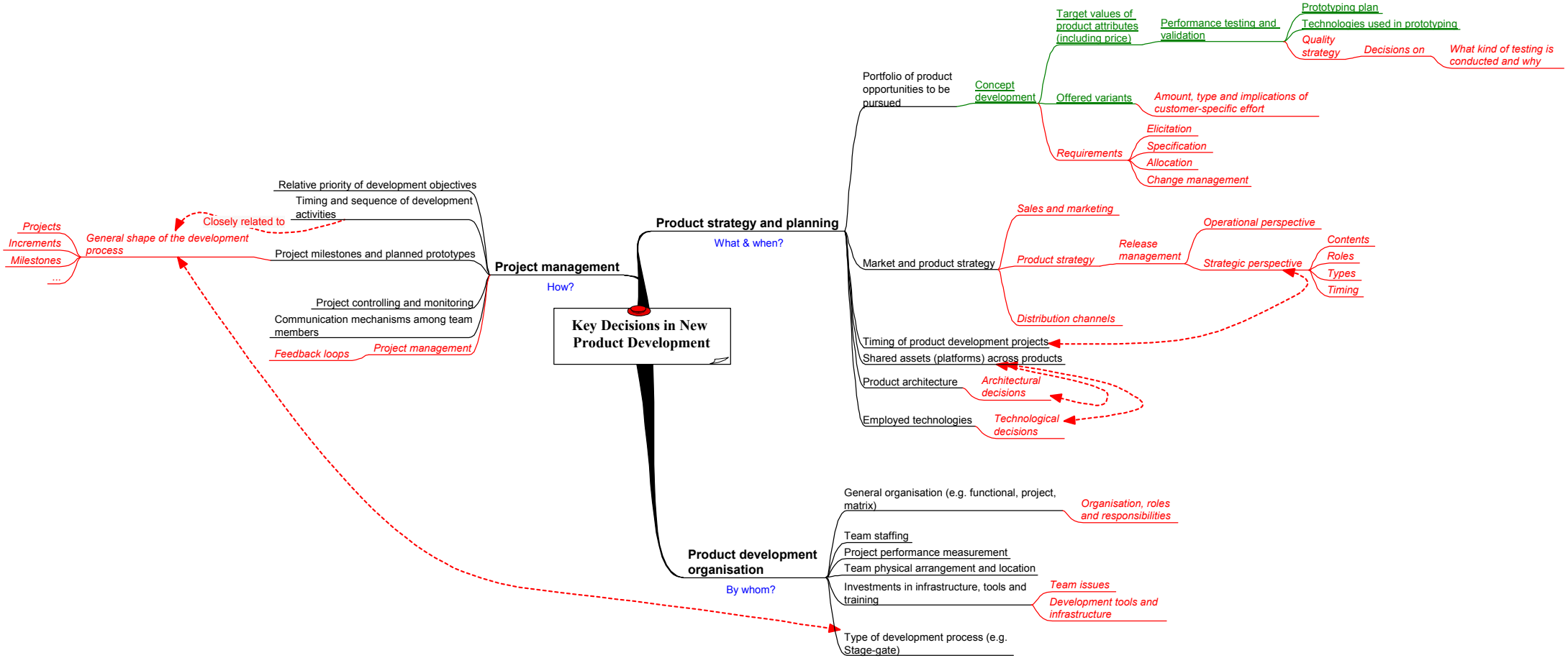

Figure 13 The starting point: relationship of the elements from Table 7 to the key decisions by (Krishnan & Ulrich 2001)

The first step towards these goals is to explain the relationships expressed in Figure 13. For this purpose, we shall refer to the model of key decisions in new product development by its reference (that is, (Krishnan & Ulrich 2001)), to the management issues considered important for small software product business as 'the construction', and the names of branches from either of the frameworks are *in italic* for the sake of clarity. The second step consists of simplifying the framework as far as possible by combining similar issues under a label more suitable for the small software product company context and removing redundant branches, and through regrouping the decision areas to better fit the small software product company context where possible.

First step – Initial Relationships Explained

Beginning the explanation from *project management* in (Krishnan & Ulrich 2001), the *general shape of the development process* and *project management* belong clearly where they are placed in Figure 13. Also, the *timing and sequence of development activities* from (Krishnan & Ulrich 2001) is closely related to this area.

Organisation, roles and responsibilities, *decision-making* and *team issues* are naturally very close to *product development organisation*, and could well be considered a part thereof. Also *development tools and infrastructure* from the construction can be found attached to *investments in infrastructure, tools and training*.

Sales and marketing, *product strategy* and *distribution channels* go easily under the heading of *market and product strategy* in the main branch of *product strategy and planning*. *Architectural* and *technological decisions* have their direct counterpart under this main branch also. However, the closest counterpart to *requirements engineering* from (Krishnan & Ulrich 2001) can be found by raising *concept development* from the within-project issues as an additional detail to *portfolio of opportunities to be pursued*. *Quality strategy* is a relative of *performance testing and validation*, also raised from within-project decision-making and attached to concept development via the *target values of product attributes*. Because of its context in small software product businesses, *amount, type and implications of customer-specific effort* corresponds to *offered variants*, also a part of the within-project *concept development*.

Second Step – Simplify and Regroup

From *Product strategy and planning* to *Portfolio management*. *Timing of product development projects* can be pruned because it is included in the leaf *timing* of the *strategic perspective* to *release management*. *Shared assets (platforms) across products* can be considered a part of *architectural* and *technological decisions*, which in turn can be combined into a single decision area until a need is found to separate them. This is because all three of these issues are in a small company with few products under the control of a handful of key personnel. By replacing *market and product strategy* by raising *product strategy* one level in the hierarchy, we eliminate a few branches with no loss of information and a clearer structure. In Figure 13, product strategy is mentioned in the main branch three times. By bringing sales and marketing and release management as first-level branches, we remove the need for having separate branches for *market and product strategy* and *product strategy*, again without any loss of information. Also, distribution channels can be considered a part of sales and marketing and thus moved as a part of it. Because software component of *offered variants* is included in *release types* it can be

removed safely by bringing *amount, type and implications of customer-specific effort* up the hierarchy. Also, it can be labelled simply *servicing and deployment*, as this at least in the case of small software product companies is the heart of the matter.

Thus, we end up with the two issues of quality strategy and requirements engineering that seem out-of-place under the label of *product strategy and planning*. Although requirements engineering certainly deals with *release contents*, the latter is here considered to denote high-level business requirements for the product, the highest level in a hierarchy of product requirements (Wiegers 1999). Because of the crucial importance of requirements engineering in the software product business and because the current branch is about product strategy and planning rather than about the engineering process, we decide to grant ***requirements engineering*** a main branch of its own. The same line of argumentation is also followed for *quality strategy* – decisions on how testing is organised are essential to be included as the responsibility of product strategy-level decision-making in small companies, forming the separate decision area of ***quality strategy***. Deciding about the portfolio of opportunities to pursue refers to sorting and selecting different product development project types for implementation. While this, too, happens in the small business context in the form of launching entirely new products or complements to the existing ones, it often scales to deciding about which releases and of what type (minor, major, maintenance, etc.) are undertaken. Thus, *portfolio of product opportunities to be pursued* can be removed – but not without propagating the notion of managing a *portfolio* instead of a *single product* to the name of the main branch and removing the word *planning* as redundant – resulting in the branch of *product strategy and portfolio management*. Finally, 'product strategy' is removed from the name of the branch because the 'strategic' nature of portfolio management is essentially contained in the other elements of the branch and here, the term product strategy would have different semantics than in the name of the framework itself. The result is the main branch of ***portfolio management***.

From *Project management* to *Development strategy*. In the project management branch, the first step is to consider the proper hierarchy between the general shape of the development process and project management. Product development may not be even conducted in projects (especially in small companies!), but it certainly always has a "shape", even though not always an explicit one. Thus, the entire branch is renamed as *general shape of the development process*, or more simply, *development model*. This also is considered to include the notion of *development process type* from the main branch of *product development organisation*, and broadened to include different ways to organise development from the more comprehensive ones dictating much of the productisation process (such as the Stage-Gate™ (Cooper, Edgett, & Kleinschmidt 2001), or eXtreme programming (Beck 2000)), or just the development phasing, such as the classic waterfall (Sommerville 1996). Thus, *project milestones and planned prototypes*, *timing and sequence of development activities, feedback loops* and *projects*, *increments*, *milestones*, etc. (renamed *phasing and pacing*) can all be moved under this branch, complete with pruning them all save the latter as redundant. For the sake of being explicit, we leave project controlling and monitoring as a separate branch here, but rename it as *progress tracking and control*, to emphasise the role of control and explicate that the framework does not assume that development effort would be conducted as projects. *Relative priority of development objectives* is considered a part of *strategic perspective to release management* and thus removed from here. Communication mechanisms among team members could be considered a part of the type

of development process. Also, it could be considered a part of the product development organisation. However, because small organisations more often can manage their development efforts without formal communication mechanisms in place (Cockburn 2001), this is left as an explicit branch under *development model*. The final step is to note that a single company, there may be a need for multiple *development model*s, for example for different *release types*. This is accounted for in the model by dropping *development model* one step down in the hierarchy, making the possible plural explicit, and grouping all the other elements under it. Then, the main branch is renamed as *development strategy* to better account for the variables involved

From *Product development organisation* to *Organisation*. First, in the case of small companies, organisation should be broadened to include the entire organisation. First, it is likely that the people whose consent must be had to change structures within the development organisation have also the power and insight to change the structures for the rest of the organisation as well, and second, in order to find and involve the remote customer with the development and integrate the management areas of sales & marketing and human resources to product strategy and development, these people should be involved. In short, any organisational structures in a small software product company should be considered together with all the other possible structures as well, and to emphasise this, the area is renamed as simply *organisation*. *General organisation* is renamed as *organisational model*, because it more clearly describes the meaning of the branch, and *roles and responsibilities* is brought one step up in the hierarchy. Project performance measurement is moved under the new main branch of quality strategy because it is essentially related with risk management (see the definition of the area in section 5.2.6 for explanation), and issues concerning development infrastructure (investments in (development) infrastructure, and tools and respective training) are taken under the branch containing other technology issues in *portfolio management*, since in small companies, decisions on development tools are in practice much coupled with the technologies employed as well. Thus the part of *investments in training* left here deals solely with non-technical training, which is essentially the same as the author's intention behind *team issues*. These can now be combined as *investments in team collaboration*.

***Technology decision-making* separated from *Portfolio management*.** Finally, the entity of architectural and technological decisions, somewhat out of place under the issues on portfolio management, are taken as a separate decision area to emphasise the importance of technology decision-making for a small software product company. This area is named simply technology, and consists of product architecture (including product platform issues) and employed technologies, and development infrastructure.

The Resulting Framework

The framework resulting from combining key new product development decisions by (Krishnan & Ulrich 2001) and the managerial implications of being a small software product business, and grouping them as to make the framework usable from the perspective of small software product businesses is depicted in Figure 14 below.

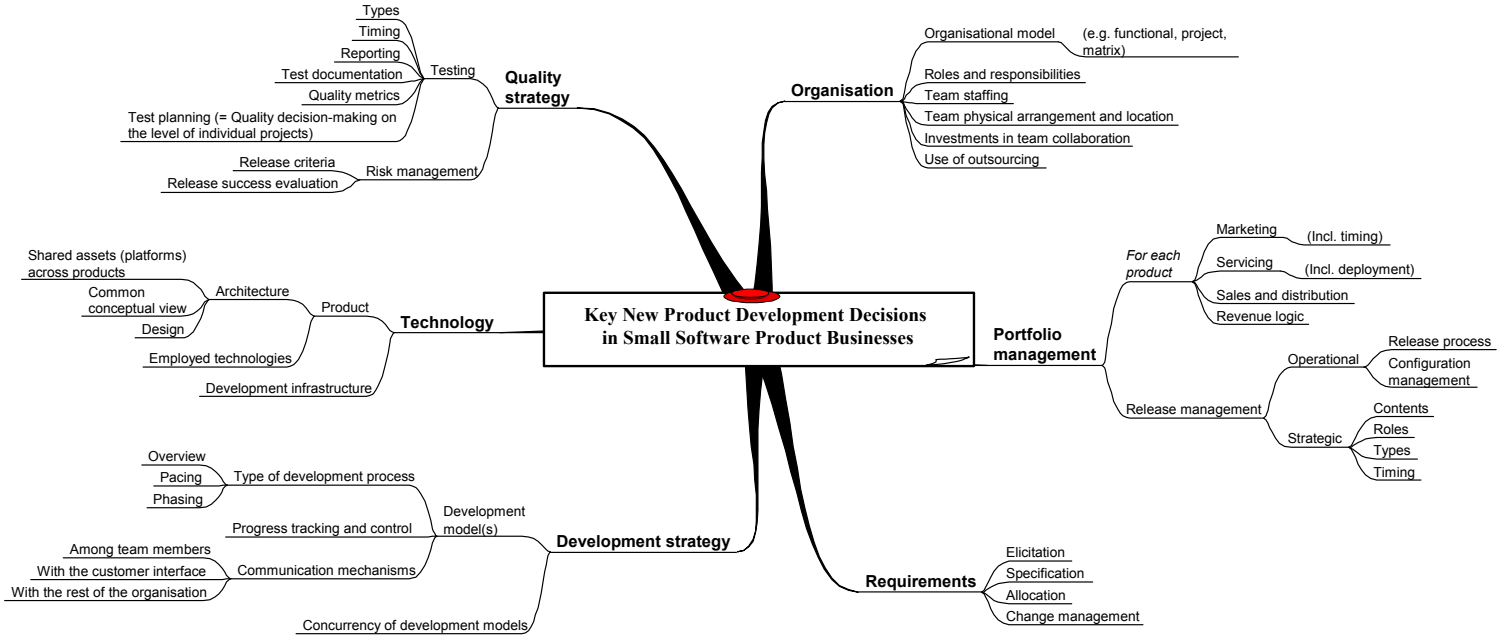

Figure 14 Key product strategy decisions in small software product businesses

Appendix D: Combining the Theoretical and Empirical Frameworks – Details

Below, the modifications to the framework from Table 8 (p. 33) that were made in order to reach the final framework (Table 12, p. 50) are shown. Additions from the empirically constructed version (Table 10, p. 38) are in *red and italic*, and highlighting in yellow denotes that the semantics of an area were changed slightly and the area was renamed to better reflect the new semantics.

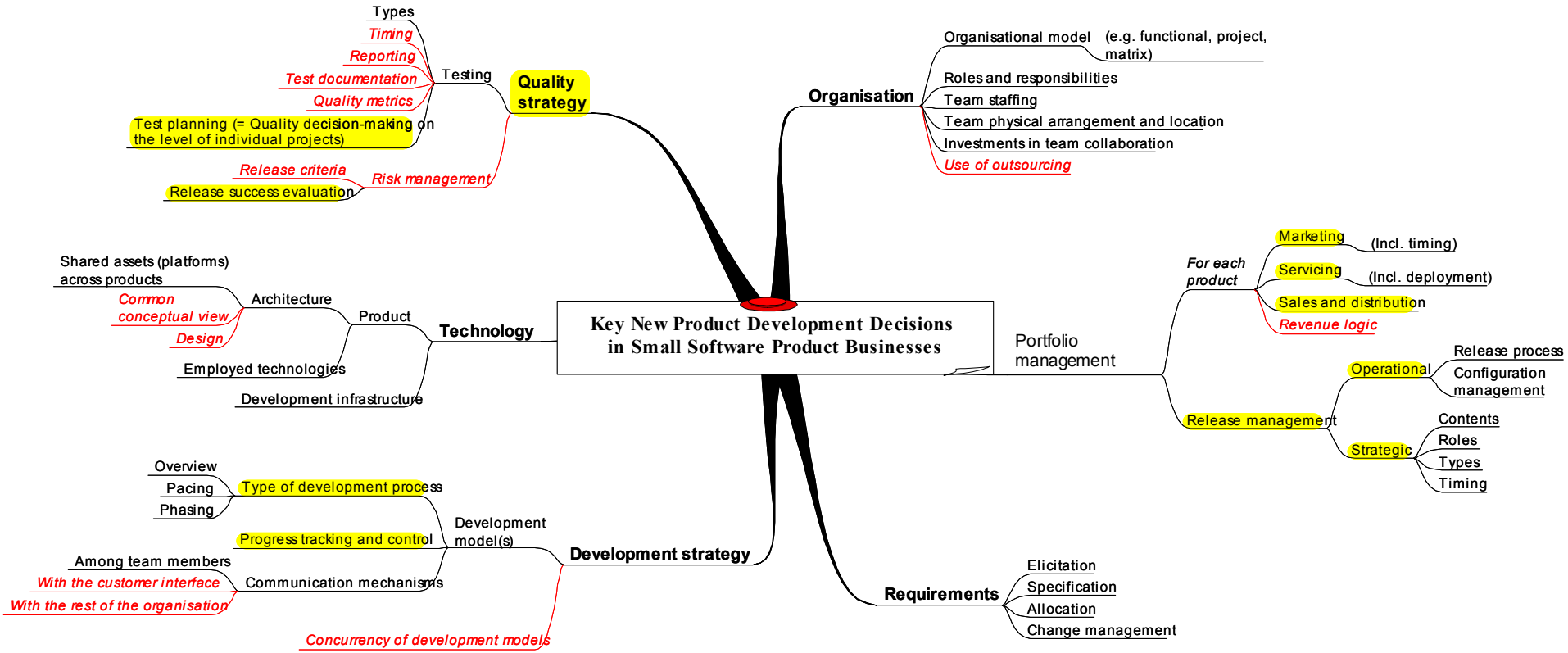

Figure 15 The final framework with changes resulting from the empirical approach to the theoretical framework highlighted