AN INVESTIGATION INTO THE APPLICATION OF SYSTEMATIC SOFTWARE REUSE IN A PROJECT-CENTRIC ORGANISATION

by

MARK JONATHON CHAPMAN

Submitted in part fulfilment of the requirements
for the degree of

MASTER OF SCIENCE

in the subject

INFORMATION SYSTEMS

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: PROF. A J VAN DER MERWE

JANUARY 2007

Abstract

The software development continues to become more competitive and demanding, placing pressure on developers. Changes in the international political climate have resulted in shrinking military budgets, putting developers of defence software under further pressure. At present, systematic reuse is probably the most realistic way of addressing this pressure by improving software development productivity and quality. Software product line (SPL) engineering provides a comprehensive approach to systematic software reuse and is becoming widely accepted.

The focus of this interpretive case study was ground station software development in a small multidisciplinary project-centric company which produces avionics systems for military aircraft. The purpose of the study was to investigate the potential implementation of systematic software reuse in the company.

The study consisted of three phases, a literature study, a contextualisation and a set of field interviews, and used elements of the Carnegie-Mellon Software Engineering Institute (SEI) Product Line Practice Framework to examine the suitability of SPL engineering for the company.

The findings of the study highlight the potential challenges that SPL engineering poses for the company, and emphasise how the company's project-centric structure could impede its implementation of systematic software reuse.

Keywords

Systematic software reuse, software product line engineering, project-centric, defence software, qualitative, interpretive, case study.

Notes on Writing Style

I have chosen to write my dissertation using the first person voice, in spite of this still being regarded as unscholarly by some. I have done so based on Amir's (2005) suggestion that the writer should choose a style which reflects his "world-views, beliefs and values". Amir (2005) reports that in qualitative research the use of the first person is no longer regarded as less scholarly than the third person, and also that it supports the narrative writing style of reporting case studies.

In several places in my dissertation I use the pronouns "he" and "his" in their gender-neutral form, and as others have done, I encourage readers to replace these with the pronouns of their choice.

Acknowledgements

I express my thanks to the following people whose assistance and cooperation made this study possible.

The participants: for their valuable time and willing cooperation.

My company: for granting permission to conduct this study and for subsidising my tuition.

My supervisor, Prof. Alta van der Merwe: for her positive attitude and enthusiastic supervision.

Prof. John Barrow: for his advice and help in getting my study started.

My wife, Hildegard: for her unwavering support, understanding, and patience; also for turning a blind eye to my occupation of the dining room table.

TABLE OF CONTENTS

List of Figures

List of Tables

Abbreviations

COTS	Commercial off–the-shelf
DAU	Data acquisition unit
DBMS	Database management system
FLS	Flight line systems
GSAM	Guidelines for successful acquisition and management (of software-intensive systems)
HUMS	Health and usage monitoring system
IEEE	Institute of Electronic and Electrical Engineers
IP	Intellectual property
IS	Information systems
J2EE	Java 2 Enterprise Edition
PC	Personal computer
R&D	Research and development
SEI	Software Engineering Institute (at Carnegie Mellon University)
SME	Small and medium enterprises
SPL	Software product line

Chapter 1. Introduction

Although software reuse is almost as old as software itself (Frakes & Kang, 2005), it is still seen as potentially the most promising strategy for increasing productivity and improving quality in the software industry (Lynex & Layzell, 1998; H. Mili, Mili, & Mili, 1995; Morisio, Ezran, & Tully, 2002). Although it is a deceptively simple concept, history reveals that, in practice, software reuse is frustratingly difficult to implement successfully (Morisio, Tully, & Ezran, 2000; W. Myers, 1997).

Software reuse research, which has been intense over the past two or three decades (Frakes & Kang, 2005), continues to improve our understanding of the topic; for example, we now know that for reuse to be effective and sustainable it should be *systematic reuse* (i.e. carefully planned and managed) as opposed to *opportunistic* or *ad hoc* reuse (Laguna, González-Baixauli, López, & García, 2003; Schmidt & Buschmann, 2003). In addition, although software reuse was previously considered to be a purely technical issue, evidence shows that organisational issues have a significant effect on the success of software reuse programmes (Birk, Heller, John, von der Maßen et al., 2003; Dikel, Kane, Ornburn, Loftus, & Wilsin, 1997; Fafchamps, 1994; Lynex & Layzell, 1998).

Proven strategies for adopting comprehensive software reuse now exist. An example of one such strategy is SPL engineering, which aims to exploit the potential for reuse that exists within families of similar software products (Bosch, 2002b; Schmid & John, 2002).

This dissertation documents an interpretive case study undertaken to gain insight into the potential implementation of systematic software reuse in a small multidisciplinary company that produces avionics[1] systems for military aircraft. The organisational structure of the company is predominantly project-oriented or project-centric[2].

Chapter 1 provides the background to the study and gives an overview of the research methods used.

[1] Electronics applied to aviation.

[2] The term “project-centric” is used to describe an organisation in which the dominant organisational structure is the project. For a more detailed description *see* section 1.1.4.

1.1 BACKGROUND

1.1.1 General Background to the Study

The company which forms the context of this study produces aircraft avionics systems for military aircraft and has a two-decade history of operation in the defence electronics market. The development of aircraft avionics systems requires multidisciplinary skills which include software development. Operators in this market have traditionally, to a large degree, been project-orientated (C. Jones, 2002) – a situation that can probably be ascribed to issues such as the uniqueness of contracts and special requirements for functionality, qualification and security.

For the past decade, the company has focused on a branch of avionics classified as health and usage monitoring systems (HUMS). A HUMS typically monitors various parameters to establish the status of the health or airworthiness of the aircraft and to record its usage or life consumption. The purpose of a HUMS is to improve safety and to provide data to facilitate the efficient management of a fleet of aircraft.

In spite of narrowing its business focus, the company's level of benefit from software reuse remains limited. This situation is becoming a concern as the increase in competition in the industry translates into a demand for improvements in productivity and quality.

A preliminary search revealed that most of the published software reuse success stories feature large organisations with ample resources, large research and development (R&D) budgets and sizeable development teams, such as Hewlett Packard, Phillips, Nokia, Motorola and Boeing (Gacek, Knauber, Schmid, & Clements, 2001; Knauber, Muthig, Schmid, & Widen, 2000). The same search also revealed that historically, software reuse research focused on the technical issues (Lynex & Layzell, 1998), and consequently there are relatively few papers dealing with the organisational issues. The shortage of software reuse research into organisational and social issues is worrying, especially if one considers that general information systems research indicates that it is these issues that pose the real challenges (Harvey & Myers, 1995).

Much of the literature which does address the organisational issues of software reuse promotes an organisational structure that separates the *development for reuse* function (also called *domain engineering*) from the *development with reuse* function (also called *application engineering*) within an organisation (Clements & Northrop, 2001, pp. 312-326; Czarnecki & Eisenecker, 1999; Jacobson, Griss, & Jonsson, 1997; Laguna et al., 2003). This suggests that the traditional project-centric software development environment is not ideal for supporting software reuse. In a project-centric environment, individual software products are developed by insular project teams that are responsible for their own technical decisions and there is little or no sharing of personnel and reusable assets between teams (Clements & Northrop, 2001, p. 312; Cohen, 2002).

In contrast to large organisations, small and medium-sized organisations often owe their success to their responsiveness and flexibility in satisfying customer requirements (Knauber et al., 2000). As a result, these organisations might become vulnerable if they adopted a reuse strategy that threatens these qualities. Smaller organisations often also lack the necessary financial resources to invest in long-term reuse strategies and organisational restructuring.

1.1.2 Defence Software

The software that this study addresses is categorised as *defence software* and although it has a great degree of similarity to general commercial software, there are significant differences.

According to Jones (2002), *defence software* is a term used for software developed for use by or in support of a country's defence forces. This software is in most cases produced by civilian contracting companies for a procurement organisation acting on behalf of a defence force. It is common practice for large organisations to win major contracts and then sub-contract smaller specialist companies to provide the subsystems. The company on which this study focused falls into this smaller specialist company category.

Defence software is distinguished from other software by its adherence to military standards, which dictate the levels of documentation, quality controls, requirement management, auditing, reporting, and qualification procedures. This difference is the main reason that a military software project typically has three times the *paperwork* of a comparable civilian project, a great deal of which only serves to prove compliance with contractual commitments (C. Jones, 2002). The additional overheads make defence software considerably more expensive to produce, and consequently the potential benefits offered by software reuse are even more attractive.

1.1.3 What is a HUMS?

A typical health and usage monitoring system (HUMS) acquires data from the airframe, the engine and other avionics systems on the aircraft during the flight. The data is processed and analysed to derive information on the health (or condition) and usage of an aircraft's critical components. This information is used to assist the fleet manager to monitor and optimise the use of the aircraft. A HUMS usually comprises three functionally and physically distinct components: the data acquisition unit (DAU) on board the aircraft; the ground station for processing, storing and analysing the data; and the flightline[3] system (FLS) for transferring data between the DAU and the ground station. In general, the ground station will be a personal computer-based workstation located in an office environment and the FLS will be a ruggedised[4] notebook computer.

The HUMS software effort consists of the development of embedded software and the development of personal computer (PC) based software. The embedded software resides in the DAU on board the aircraft and the PC-based software is for the ground support equipment, which comprises the flightline system and the ground station.

Since the case study focuses on the ground station software development, this component of the HUMS is described in extra detail. Although the ground station application is generally categorised as engineering software, there is little difference between it and a conventional information system. The ground station application

[3] The flightline is a designated area of an airfield where an aircraft is parked for the purpose of loading and servicing. The flightline system is typically used to transfer data from the DAU while the aircraft is on the flightline.

[4] A ruggedised computer is built to withstand a moderate level of mechanical shock and to operate in a harsh environment (e.g. with exposure to climatic extremes and chemical substances).

provides for the processing, storage and analysis of data for a fleet consisting of twenty to thirty aircraft over its lifespan of twenty-five to thirty years. A database is maintained to record the configuration status of each aircraft and engine, the serial numbers, accumulated life, and operational status of all critical aircraft components as well as the details of each flight. Data acquired by the DAU is processed by the ground station using complex algorithms to determine the life consumed for each aircraft component per flight. The ground station application also checks for damage-causing exceedances[5] of various parameters. Reports are generated and provision is made for the graphical analysis of processed data for various purposes, such as: to identify components that are approaching end-of-life; to identify fault conditions; and to identify unexpected differences, changes or trends in various parameters. After the ground station has processed the raw data captured by the DAU, the data is stored in a repository[6] to facilitate subsequent detailed analysis of the data parameters of individual flights should the need arise.

1.1.4 What is a Project-centric Organisational Structure?

The Guidelines for Successful Acquisition and Management of Software-Intensive Systems (US Air Force, 2003), commonly known as GSAM, maintains that, from a project management point of view, an organisation is function-oriented, project-oriented or some type of matrix structure in between, *see* Figure 1-1.

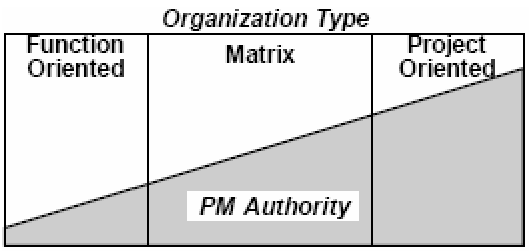

Figure 1-1: Organisation Type and Project Management Authority (US Air Force, 2003)

In a project-oriented or project-centric organisation (*see* Figure 1-2) the dominant unit of management control is the project, which is under the authority of a project

[5] An exceedance is a condition in which a single parameter or a set of parameters represents a state that exceeds a range that represents the normal expected operation of the system.

[6] The repository is a dedicated area on the ground station's hard disk set aside for the storage of raw data from the DAU.

manager. The project manager reports directly to senior management and has almost unrestricted authority within the context of the project.

Project 1	Project 2	Project 3
<activity>	<activity>	<activity>
<activity>	<activity>	<activity>
<activity>		<activity>
<activity>		

Figure 1-2: Project-oriented Structure

In a function-oriented organisation, the dominant unit of management control is the division under the authority of a divisional or line manager, who reports to upper management. In this structure, the project manager's function is reduced to managing activities for the line managers.

software	<activity><activity><activity>
electronics	<activity><activity><activity>
mechanical	<activity>
control	<activity><activity>
integration	<activity><activity><activity>

Figure 1-3: Function-oriented Structure

A mix between the project-oriented and function-oriented structure is the matrix structure in which the project manager negotiates with the line manager for the resources required by the project (*see* Figure 1-4).

	Project 1	Project 2	Project 3
software	<activity>	<activity>	<activity>
electronics	<activity>	<activity>	<activity>
mechanical	<activity>	N/A	N/A
control	<activity>	N/A	<activity>

Figure 1-4: Matrix Structure

The Concise Oxford Dictionary describes *-centric* as a suffix used to form "adjectives with the sense 'having a (specified) centre'". For example: *Eurocentric* means "having or regarding Europe as its centre". In the context of this study, a *project-centric* organisation is one that regards the project as the centre or focus of its organisational structure. The literature more commonly uses the term *project-oriented* to cover this type of structure. The structure of the company at the centre of this study could be described as project-centric at the start of the study, but during the execution of the study a matrix structure was introduced. The ownership of the company changed and the principal reason for the structural change was to align the structure of the company with that of the new mother company. In spite of the change, the company remains predominantly project-oriented and would still be located decidedly to the left of the graph in Figure 1-1.

1.2 PURPOSE OF THE STUDY

The purpose of this study was to investigate the potential implementation of systematic software reuse in a small project-centric organisation.

For software reuse to be successful it needs to be systematic (Laguna et al., 2003; Schmidt & Buschmann, 2003). The adoption of a systematic software reuse strategy involves a transition that imposes organisational and technical requirements on a company. The level of effort required depends on the magnitude of the changes necessary to achieve this transition.

There is limited coverage in IS literature of the difficulties and challenges that organisations might face in adopting systematic software reuse (*see* Section 1.4.2). This study identifies and investigates a selection of these difficulties and challenges for a small project-centric organisation.

1.3 RESEARCH QUESTION

Numerous technical and organisational issues need to be addressed during the introduction of a systematic software reuse programme. Some of the challenges posed by these issues will naturally differ according to the environment or context. The purpose of the study was to identify and investigate the issues that could prove to be challenging when implementing systematic software reuse in the context of a small project-centric organisation.

The primary research question addressed by the research study was:

What are the issues related to the introduction of systematic software reuse in a small project-centric organisation?

During the research design, I divided the primary research question into six subsidiary questions (SQs). The subsidiary questions were of an exploratory nature and served to demarcate and guide the study. The questions are as follows:

- **SQ1***: What is a generally accepted approach to systematic software reuse within the industry?* The aim of this question is to identify a particular approach in order to provide a theoretical context for investigating systematic software reuse.
- **SQ2***: What is the context of the study and, more specifically, what are the historical factors which contributed to the organisation being project-centric?* This question seeks to determine and comprehend the historical and social context of the study.
- **SQ3***: Are there motivating factors for maintaining a project-centric organisational structure?* This question seeks to establish and comprehend the contemporary context of the study.
- **SQ4***: To what degree is there a need within the organisation for systematic software reuse?* This question is intended to assess the level of need for software reuse in the company.
- **SQ5***: How suitable is the generally accepted approach for adoption by the organisation?* This question aims to determine the level of compatibility of the identified software reuse approach for the company.
- **SQ6***: What are some of the technical and organisational issues that might influence the implementation of systematic software reuse within the organisation?* The purpose of this question is to identify potential technical and organisational factors that could pose a challenge for the company.

1.4 RATIONALE

1.4.1 Personal

As a long-time employee involved in software development in the company targeted by this study, I have become aware of, and concerned about, the difficulty experienced in achieving software reuse. A motivating factor for undertaking this study is a desire to understand this phenomenon. My hope is that a more informed opinion will facilitate achieving a valid and convincing critical analysis of the phenomenon, which could contribute to an improved strategy for software reuse in the company.

Diverse statements made in the company on the topic of software reuse have served to increase my aspiration to achieve an improved understanding of the allied reuse issues. Examples of such statements are:

- "Software reuse doesn't work! Company X tried it and found it to be a waste of time and money" – Senior Executive.
- "Why don't we just reuse the software from project X." – Senior Executive.
- "… but isn't your software reusable?" – Project Manager.
- "Why didn't project X reuse my software?" – Software Developer.

These statements are indicative of some of the negativity and misconceptions that characteristically accompany software reuse. Prominent among these misconceptions is the notion that software reuse can succeed without systematic planning, active management, and sufficient resources, or that software reuse is purely a technical issue.

1.4.2 Scientific

There are numerous papers covering case studies of successful reuse projects (Brownsword & Clements, 1996; Clements, 2002; Clements & Northrop, 2002; Cohen, 2002; Vernazza, Galfione, Valerio, Succi, & Predonzani, 2000). There are also several papers covering case studies on software reuse across a group of organisations (i.e. horizontal case studies) (Morisio et al., 2002; Rothenberger, Dooley, Kulkarni, & Nader, 2003). However, a literature search revealed that there are few papers addressing the practical challenges faced by individual organisations attempting systematic software reuse. I hope that this dissertation will contribute towards redressing this situation.

The success of software reuse strategies depends largely on contextual factors such as the organisational structure, organisational culture, the adoption strategy and the supportive processes (Bühne et al., 2004). In spite of this, studies that deal with these factors are scarce (Bühne et al., 2004). It therefore follows that studies addressing these contextual factors in small and/or project-centric organisations are even scarcer. By considering these factors in a small project-centric company, this study will also contribute to addressing this scarcity and will hopefully also create an increased awareness in similar organisations.

As it is a single case study, the findings of this research are not immediately suitable for generalisation. Although, as Myers (1999) suggests, generalisation could be effected across several similar studies, the emphasis of a single case study is more on *particularisation* than *generalisation* (Stake, 1995. p. 8). Accordingly, the emphasis in this study is on understanding the particular case rather than comparing it with others (Stake, 1995. p. 8).

1.5 RESEARCH STRATEGY

I conducted a detailed literature study to determine the background of software reuse and its current trends and practices. In order to set the study in its social and historical context, I expanded the literature study for this purpose and conducted an elite interview with a long-standing senior employee.

I held a series of semi-structured field interviews based on open-ended questions to determine perceptions among participants of the company's need for software reuse and the suitability of a selection of systematic software reuse practices. The discussions that accompanied the interviews also revealed perceptions of software reuse disincentives caused by the company's project-centric environment.

1.6 CONTEXT, SCOPE AND LIMITATIONS

1.6.1 Context of the Study

The context of this study is a small, multidisciplinary, avionics-equipment company that produces health and usage monitoring systems (HUMS). Owing to the importance of the context in qualitative research in general, and in information systems research in particular (*see* Section 3.2.5), a comprehensive analysis of the study context is presented in Chapter 4.

1.6.2 Scope of the Study

This study investigates the potential implementation of systematic software reuse within the context of the small, multidisciplinary, avionics-equipment company with a project-centric organisational structure. To be pragmatic in the light of limited time and resources, I restricted the scope of the study to the ground station software development element of a series of HUMS development projects.

Various other development efforts within the company have equally good reuse potential and are suitable case study material. However, the ground station element was chosen because of the convenience offered by access to suitable interview participants, ease of description, and my personal involvement and familiarity with the material.

1.6.3 Limitations of the Study

The software developed by the company being studied can be separated into three groups, *see* Table 1-1.

Table 1-1: Software Groups in the Company

SOFTWARE GROUP	**CLASSIFICATION OF SOFTWARE DEVELOPED**
On board	embedded data acquisition unit software
Ground support	ground station and flightline system software
Test systems	software for the control of test benches and aircraft simulation benches for testing and qualification of the system

The scope of the study was purposely limited to the ground station element of the company's software development (*see* Section 1.6.2). Given sufficient time and resources, the coverage of the study could be extended either horizontally or vertically.

The coverage of the study could be extended horizontally to address any or all of the other software elements belonging to the other two software groups. As discussed in Section 2.2, in addition to software, there are many physical and intellectual assets associated with the software development that can also be reused (Desouza, Raider, & Davenport, 2003). Taking this idea a step further, the study could be extended to non-software elements, such as electronic hardware and mechanical-housing development.

Perhaps the most practical way of extending this study vertically would be to expand it to a full action research study, in which a software reuse strategy is applied and its effectiveness is measured, the strategy is modified, and so on, until a suitable strategy is reached. Obviously a study of this nature would take significantly more time and effort.

1.7 OUTLINE OF CHAPTERS

Table 1-2 outlines the content of the chapters of this document.

Table 1-2: Outline of Chapters

CHAPTER	CHAPTER OVERVIEW
1	**Introduction** Chapter 1 introduces the study, provides the background to the study and gives an overview of the research methods used.
2	**Theoretical Framework** Chapter 2 establishes a theoretical framework for the study, it provides a brief coverage of the history of software reuse, it shows how the identified systematic reuse strategy evolved, it provides an overview of reuse practices, and discusses models for adopting a strategic software reuse strategy as well as associated advantages, disadvantages and issues.
3	**Research Methodology and Design** Chapter 3 motivates and describes the research design used for this study. Before describing and motivating the research design, it provides relevant background information to contextualise the methods that influenced and contributed to the research design.
4	**Contextualisation** Chapter 4 sets the study in its social and historical contexts and provides a detailed discussion of the factors which have led to the company adopting and maintaining its project-centric structure.

CHAPTER	CHAPTER OVERVIEW
5	**Data Analysis** Chapter 5 documents the results of the analysis of the field interview transcripts. The chapter presents a view based on the perceptions of participants regarding the company's potential for adopting systematic software reuse.
6	**Discussion of Results** Chapter 6 summarises the study and its findings, discusses the findings and provides a conclusion.

Figure 1-5 provides a conceptual view of the study illustrating the relationships between the research question, the data collection methods, the analysis of the data and the study findings. The only data collection method not represented in Figure 1-5 is participant observation (*see* Section 3.3.4).

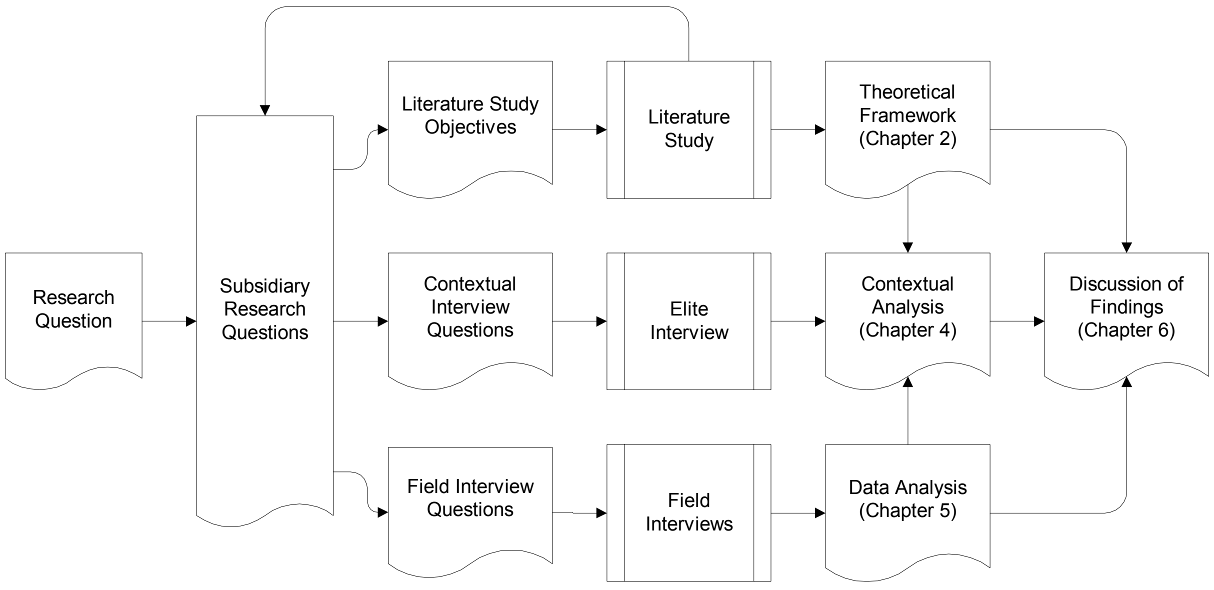
Research Question
Subsidiary Research Questions
Literature Study Objectives
Literature Study
Theoretical Framework (Chapter 2)
Contextual Interview Questions
Elite Interview
Contextual Analysis (Chapter 4)
Discussion of Findings (Chapter 6)
Field Interview Questions
Field Interviews
Data Analysis (Chapter 5)

Figure 1-5: Conceptual View of the Study

Chapter 2. Theoretical Framework

2.1 INTRODUCTION

> Software development cannot possibly become an engineering discipline so long as it has not perfected a technology for developing products from reusable assets in a routine manner, on an industrial scale. (A. Mili, Yacoub, Addy, & Mili, 1999, p. 22)

According to Kranzberg (1964, p. 307), the Industrial Revolution, which began almost simultaneously in Britain, Europe and the USA in the mid 1700s, marked the transformation of society rather than a specific period. Despite its name, this transformation was more of an evolution than a revolution, and was caused in part by technological advances such as improved materials, new tools, the application of science to industry and the introduction of the factory system with production lines. The factory system gave rise to the division of labour and the specialisation of functions. These advances made possible the mass-production of goods.

The Industrial Revolution transformed a society in which individual craftsmen handcrafted one-off products from basic materials, to one where a wide range of product variants could be rapidly assembled using standard components (Greenfield & Short, 2004, p. 5). This transformation resulted in the large-scale reduction of costs and an increased availability of products of consistent quality.

The success of industrial manufacturing can be attributed directly to the broad-spectrum reuse of physical and intellectual assets such as components, designs, processes, skills, experience, tools, machinery, facilities and so on. Anything that can be meaningfully reused can be considered to be a *reusable asset*. Consequently, the success of industrial manufacturing can be ascribed to the judicious application of reusable assets.

The Standish Group (1995) published the now frequently quoted and aptly titled *Chaos Report*, which claimed that only 16% of software development projects finish on time and within budget and that as many as 31% of software development projects are abandoned before completion. Although the situation has undoubtedly improved since 1994, software development remains very much the realm of craftsmen (Czarnecki & Eisenecker, 1999) and has yet to enjoy the benefits of manufacturing processes.

The software development industry has not managed to benefit fully from the valuable lessons of the Industrial Revolution. Greenfield and Short (2003) point this out to be somewhat ironical considering the substantial contribution made by software and information systems to the automation of processes in other industries. The situation can possibly be attributed to the fact that software is predominantly logical or conceptual rather than physical.

The industrialised production of physical goods provides rewards owing to economy of scale and economy of scope. *Economy of scale* is the benefit realised by producing many identical instances of the same product. *Economy of scope,* on the other hand, is realised when a range of different but closely related products is produced from similar designs using common components and shared production processes. In the production of software, *economy of scale* is technically a non-issue owing to the simplicity of copying a software executable. However, in software the real challenge is to achieve *economy of scope* by benefiting from the production of a range of closely related software products. This is also the challenge of software reuse.

This chapter presents the findings of the literature study, the purpose of which is to establish a theoretical framework for the overall study by identifying a *generally accepted approach to systematic software reuse within the industry* as well as the practices relating to this approach. The chapter provides brief coverage of the history of software reuse; describes the evolution of SPL engineering; gives an overview of the product line practices; discusses models for SPL adoption; and addresses SPL advantages, disadvantages and issues.

2.2 THE EVOLUTION OF SOFTWARE REUSE STRATEGIES

> Software reuse is the use of existing software or software knowledge to construct new software (Frakes & Kang, 2005, p. 529).

For software reuse to be effective it must be systematic rather than opportunistic (Laguna et al., 2003; Schmidt & Buschmann, 2003). Systematic reuse is carefully planned reuse that has sufficient resources allocated to it and forms part of an institutionalised software development effort (Schmidt & Buschmann, 2003). Opportunistic reuse, on the other hand, is spontaneous, ad hoc, and neither carefully planned nor managed (Lim, 1998; Morisio et al., 2000). When an organisation exploits software reuse as part of its strategy to enter new markets and gain initiative and competitive advantage, it is called strategic reuse (Lim, 1998).

Deliberate software reuse efforts started in the late 1950s with the introduction of functions and subroutines (Raccoon, 1997). This was essentially opportunistic reuse; programmers would cut snippets of existing code, paste them into new programs and then apply modifications as required.

The introduction of modules and modular programming in the late 1970s advanced the reuse effort because modules tended to be developed specifically with reuse in mind (Raccoon, 1997). Unfortunately, the temptation to modify the reused code often resulted in a multitude of variations of the same module. Although object libraries went some of the way in remedying this problem by denying the developer access to the source code, they often lacked sufficient flexibility, which ultimately resulted in their rejection.

In the late 1980s and early 1990s, the object-oriented development paradigm was unveiled and with it came the concepts of classes, inheritance, polymorphism and data encapsulation (Raccoon, 1997). These concepts facilitate the development of reusable code. Inheritance in particular promotes code reuse, since common code can be located in a superclass, which can subsequently be inherited by any number of subclasses. Inheritance is labelled as white-box reuse because it requires an understanding of the class's internal implementation (Foote & Yoder, 1995). Some of the initial euphoria that accompanied the advent of object-oriented development was based on the misconception that it would automatically guarantee reusability (Jacobson et al., 1997). Although the approach provides powerful mechanisms for supporting code reuse, the ultimate reusability of the software depends largely on the careful selection and implementation of appropriate classes. A significant disadvantage of this technology was that only classes written in the same or compatible languages could be combined to create applications (Pree, 1997).

The perceived shortcomings of classes shifted the attention of the software reuse community to the idea of the *software component*. The idea, proposed by Brad Cox, was to use software components in a similar fashion to electronic components or integrated circuits (Johnson, 1997). A software component differs from a class by generally being language-independent and encapsulating a complete concept, rather than a partial concept (Pree, 1997). Component-based reuse is often referred to as *coarse-grained*, whereas class reuse is considered *fine-grained,* because it usually requires a collection of classes to implement a complete concept (Pree, 1997). For a component to be effective, its interface should be simple and intuitive and it should hide the internal implementation, allowing the developer to consider the component as a black-box subsystem not requiring knowledge of its internal implementation. The coarse-grained, black-box nature of components is deemed more desirable than the fine-grained, white box nature of classes, principally because this helps to boost developer productivity by raising the level of design abstraction.

A common criticism of component-based development is that it is essentially a bottom-up strategy (Bosch, 2000; Matinlassi, 2004; H. Mili et al., 1995). The logic behind this criticism is that component-based development does not address the high-level design or architecture of an application (Ran, 1999). Effectively, the same basic set of components can be used to produce applications with a variety of architectures. Another problem is that the interface of a component intended for general usage is complicated by the variety of options required for an unspecified range of application contexts. Finally, the lack of stable and established component standards makes it risky and difficult to commit to a particular technology, and has resulted in the market for commercial off-the-shelf (COTS) components being somewhat limited. However, there is some optimism that this problem has been be resolved by the recent emergence of the Java 2 Enterprise Edition (J2EE) and Microsoft .NET component technologies (Carlson, 2004; Schmidt & Buschmann, 2003).

Software architecture is a multifaceted subject for which there is little agreement on a standard definition (Clements & Northrop, 1996; Kruchten, Obbink, & Stafford, 2006). Clements and Northrop (1996, p. 3) suggest the use of Garlan and Perry's definition[7]:

[7] *see* (Garlan & Perry, 1995)

“The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time”.

Software architecture serves to characterise the context within which the software components are to function, permitting developers to optimise components for their intended context. Because the design of software architecture is demanding and time-consuming, it adds a large overhead to software development (Pasetti & Pree, 2000). Consequently, in terms of time and effort, the reuse of software architecture is an extremely rewarding practice that results in applications with a high degree of uniformity.

Software design patterns are based on a technique established by Christopher Alexander in the late 1970’s for architecture in the building environment, which addresses the documenting of problem solutions (Bahrami, 1999, p. 72). The idea is to identify a recurring software problem, and to record the solution strategy with the intention of reusing it when a similar problem occurs. Although the concept of patterns specifically addresses design and makes no attempt to address specific code reuse, it is in practice often coupled with object-oriented programming.

Application frameworks provide a way of combining the reuse of software architecture and components. An application framework is basically a set of cooperating classes assembled according to a specific architecture to provide a template for applications in a specific problem domain. According to Bahrami (1999, p. 78), an application framework is “the physical realization of one or more software pattern solutions”. However, Schmidt & Buschmann (2003, p. 697), describe an application framework as "a *semi-complete* application that programmers can customise to form complete applications by extending reusable components in the framework”.

The technologies described thus far all focus on the reuse of either code or software architecture or both, but as with reuse in industrial manufacture, there are many other physical and intellectual assets associated with the software development that can also be reused (Desouza et al., 2003). Software product line engineering is a reuse approach that is comprehensive in that it addresses the reuse of software as well as all other associated assets, such as documents, plans, processes, schedules and tools (Clements, 1997).

2.3 WHAT IS SOFTWARE PRODUCT LINE (SPL) ENGINEERING?

Software product line (SPL) engineering is emerging as a promising and practical approach to systematic software reuse (Böckle et al., 2002; Knauber & Succi, 2002; Northrop, 2002b). The software product line “epitomises” systematic software reuse (Northrop, 2002a) and is currently the “most dominant form” thereof (A. Mili et al., 1999, p. 25). It is aimed at “exploiting the enormous reuse potential” of software product families (Schmid & John, 2002, p. 1). According to Northrop (2002b, p. 40) “the industry trend toward software product lines seems indisputable”. From this information we can conclude that SPL engineering is a generally accepted form of systematic software reuse.

Software applications have become substantially more complex since the early days of computing. This complexity necessitates in-depth knowledge and understanding of the problem domain. End-users, frequently exposed to sophisticated software features, now expect a similar standard from all applications. Customers are also demanding shorter lead times for software production (Lam, 1998). To survive these challenges, organisations sharpen their focus and specialise in specific application areas, also called *problem domains* (Frakes & Kang, 2005). This effectively presents a fertile environment for comprehensive software reuse. It also provides an opportunity for the organisation to accumulate intellectual capital in the form of strategic knowledge and experience in the specific problem domain.

Exceptional software reuse opportunities appear in organisations that produce families of similar applications for a particular problem domain by exploiting the commonalities of these applications (Northrop, 2002b). SPL engineering provides organisations with a set of practices for optimising these opportunities. Although the idea of SPL engineering first emerged in the mid-1990s, the concept of a *family* of software applications was recognised somewhat earlier by Parnas (1976).

All too often organisations ignore their software reuse opportunities and continue to address each new application as a one-off or single product development. The project teams often either ‘do their own thing” and start from scratch or, at best, "clone and own" a previous application. *Clone and own* is a common form of opportunistic reuse in which a new development is undertaken by making a copy of an existing application and then applying the necessary changes (Clements & Northrop, 2001, p. 12; Staples & Hill, 2004). This approach is suboptimal and often results a multitude of similar, but unmanaged and diverging instances of the initial application and has the makings of a software maintenance nightmare. The *clone and own* approach also frequently results in applications inheriting an unsuitable architecture.

The Carnegie-Mellon Software Engineering Institute (SEI), one of the principal proponents of SPLs, uses the following definition for an SPL:

> A software product line is a set of software-intensive systems sharing a managed set of features that satisfy the specific needs of a particular market segment from a common set of core assets in a prescribed way. *(Northrop, 2002b, p. 32)*

The core assets mentioned in this definition "include but are not limited to the reusable software components, architecture, domain models, requirements, statements, documentation and specifications, performance models, schedules, budgets, test plans, test cases, work plans and process descriptions" (Clements & Northrop, 2001, p. 14). The set of core assets developed for a product line will grow as the product line matures and will obviously vary from one software product member to another.

There is an element of disagreement on what the term *software product line* actually denotes. Di Nitto and Fuggetta (1996, p. 51) say that although the term might be "intuitively pleasing" it lacks clear meaning. In their opinion, SEI's definition of a software product line actually describes a *software product family,* whereas and the term *software product line* implies a set of different products that do not necessarily have commonality but complement one another, for example: the products that collectively constitute Microsoft Office. However, Clements and Northrop (2001, p. 14) argue that the concept, rather than the terminology, is the issue of real importance.

Many well-known organisations, such as Hewlett Packard, Phillips, Boeing, Alcatel, Robert Bosch Gmbh, Nokia, Raytheon, Siemens, Schlumberger, Lucent and

CelciusTech (now SAAB Systems) have documented successful SPL engineering ventures (Cohen, 2002; Northrop, 2002a).

In 1996, the Institute of Electronic and Electrical Engineers (IEEE) Software Engineering Standards Committee Reuse Planning Group released an action plan which provided the following list of proposed principles for software reuse these are (Moore, 1997):

- Build a software domain architecture as a framework for reuse activities.
- Use a software development process that promotes and controls reuse.
- Reuse more than just code.
- Practice domain engineering.
- Integrate reuse into the project management and the software engineering activity.
- Organise the enterprise to facilitate partnering to achieve reuse across product boundaries.
- Use automation to support reuse.
- Couple modern reuse theory with natural, traditional organisational reuse practices.

An important assertion made by this planning group was that the main factors that influence software reuse are non-technical (Moore, 1997). SPL engineering acknowledges this fact, while complying with most of the planning group's proposed principles.

2.4 SPL PRACTICE

Implementing an SPL is a significant undertaking involving extensive planning, restructuring, investment, action and commitment at various levels within an organisation (Northrop, 2002b; Schmid, 2002a). Although the concept of SPL engineering is well researched and widely implemented, there is no fixed recipe or formula for putting it into practice. This can be attributed to the fact that the approach to be followed is to a large degree determined by the contextual characteristics of the organisation, such as size, available resources, process maturity, culture and organisational structure, as well as the chosen application domain (Bosch, 2002b; Bühne et al., 2004).

Rather than attempting to provide a prescription for SPL implementation, SEI (2006) proposes a framework of what it perceives to be the required activities and practice areas. SEI (2006) publishes its *Product Line Practice Framework* as a *living* web-based document, which is indicative that this is an evolving body of knowledge. The Product Line Practice Framework is intended as a source of guidance for organisations wishing to adopt or maintain an SPL. Although authors such as Bosch (2000), and Schmid and Verlage (2002) use slightly different terminology, their descriptions of SPL practices are in effect similar to those of SEI.

2.4.1 Essential Activities

According to the SEI there are three activities essential to the implementation of an SPL: core asset development, product assembly and the management of these two activities (Northrop, 2002b).

Core asset development establishes the capability to assemble product line products. The outputs of the core asset development activity are the product line scope, the software architecture, the software components, the application framework and the production plan. The *product line scope* is a document that defines the boundaries of the problem domain to be addressed by the product line. The production plan sets out the documented processes that collectively describe the assembly of a product using the core assets.

Product development effectively becomes the implementation of the processes in the production plan to assemble an application, also referred to as an SPL member.

Both core asset development and product development activities require technical and organisational management. Technical management is required to ensure that plans are made and followed, and organisational management is necessary to establish and support suitable organisational structures and to allocate the necessary resources.

2.4.2 Practice Areas

The practice areas that need to be mastered in order to accomplish the essential activities are organised into three distinct categories, namely software engineering, technical management and organisational management.

2.4.2.1 Software Engineering Practice Areas

The software engineering practice areas support the technical aspects of the essential activities. Examples of the software engineering practice areas are:

- understanding the problem domain;
- requirement engineering and architecture definition;
- component and framework development; and
- testing.

Each of these is addressed below.

Understanding the Problem Domain: The importance of understanding the problem domain is emphasised, because a thorough understanding of the environment within which the SPL members are to be used is crucial to its overall success (Northrop, 2002b). Organisations contemplating SPL engineering have an advantage if they already possess comprehensive knowledge of the problem domain, however, organisations can attempt to fast-track understanding of the domain by consulting domain experts (Clements & Northrop, 2001, pp. 137-141).

Examples of elements of this knowledge are:

- characterisation of the role players that will depend on the product;
- terminology and procedures that they use;
- features of competing products; and
- applicable standards and relevant legislation.

Information of this nature is seldom likely to feature in the client’s requirement specification document, should one even exist.

In-depth knowledge of the problem domain also serves as valuable input for the development of the software architecture and the application framework, because it contributes to products that are more intuitive to the user. Additionally, knowledge of the problem domain puts the software developers and sales personnel in a strong position to negotiate with clients and to guide them in structuring their requirements to coincide with the overall SPL vision and philosophy.

Requirement Engineering and Architecture Definition: Requirement engineering takes on added importance in the context of SPL engineering. Requirements that are likely to be common to all members of the SPL should be satisfied by the collection of core assets. Customer-specific requirements should be met via planned mechanisms for application variability, known as *variation points* (Bosch, 2002b).

The success of an SPL depends heavily on the underlying software architecture (McGregor, Northrop, Jarrad, & Pohl, 2002). This dictates the need for the software architect to make a special effort to identify the architecturally significant requirements. The architecturally significant requirements are those that influence architecture design decisions. An example of architecturally significant requirements is a category termed *quality requirements*, which covers qualities such as performance, testability, maintainability, consistency, and intuitiveness of the user interface. Quality requirements can often only be met if they are specifically addressed at the outset by the software architecture (O'Brien, Stoermer, & Verhoef, 2002). Additional architecturally significant requirements stem from:

- the need to interface with other systems;
- the need to provide support for network or Internet-based operation;
- the choice of the component technology and middleware framework; and
- the desire to exploit COTS components or legacy software.

An SPL architecture must be sufficiently robust to support the necessary range of variability or *optionality*, which may also imply the disabling of functional parts, without the architecture losing its consistency (Bosch, 2000). In some cases the range of variation at a variation point is fully understood and can be implemented as a set of selectable options, in other cases the range of variability may be unknown, or only partially known, in which case the architecture must provide for an interface or component that allows for controlled extension (Greenfield & Short, 2004, pp. 365-366).

Owing to the importance of software architecture, an architecture-centred design approach is widely recommended for the development of SPLs (Bergey, Fisher, Gallagher, & Jones, 2000; Bosch, 2002a; McGregor et al., 2002). This design approach dictates that architecturally significant requirements be identified as early as possible in the SPL development in order to provide for them in the common architecture. Attempting to correct architectural defects or inadequacies at an advanced stage of the development cycle can be costly because of the far-reaching influence of the architecture. In the case of SPLs, architectural defects are potentially even more costly, considering that these defects are likely to be propagated to a number of members of the product line (Kuloor & Eberlein, 2002).

The requirement engineering process must be systematic in the analysis, categorisation and management of all requirements, especially those originating from the customer (Birk, 2002). Eliciting requirements from the customer can prove to be a difficult and thorny process (Kuloor & Eberlein, 2002), which is further complicated if the customer lacks the necessary experience or expertise. Customer requirements are probably the single most sensitive input to be considered when implementing the product line and assembling the individual products, because a prospective customer who doubts that his requirements will be met is unlikely to become a customer.

The outputs of domain analysis also provide valuable inputs for the requirement engineering process. The domain analysis outputs not only serve to identify additional requirements, but can also be used as a framework to support the elicitation and interpretation of customer requirements.

Components and Framework Development: In theory, component development is a well understood concept. In practice, however, it is complicated by the mismatch or lack of interoperability of the available component technologies, as well as the rapidly changing component industry. The selection of a component technology is a crucial decision because of the cost and time it takes to change component technologies late in the product line life cycle. It is also desirable to select a technology that has an established COTS component market because this can serve as a valuable source of standard components.

As already mentioned, a framework is effectively a partially completed application used as the foundation for a new product. An application framework forms the starting point of a new product line application. Products produced for different customers will display differences at the variation points. *Variation points* are locations and components in the software framework, identified during development, at which variation is expected.

The following are examples of mechanisms commonly employed to manage variability:

- **Conditional Compilation:** Most modern programming languages include directives that can be used to conditionally include or exclude code based on constant values defined at compilation time. This mechanism is only valid where there is access to the source code, which is often not the case with COTS components. Conditional compilation has the disadvantage of making code difficult to comprehend and manage effectively.
- **Parameterisation:** The components of a framework are implemented with options that are invoked to customise their performance at run time via parameters and properties. The disadvantage of this is that these component interfaces can become complex and difficult to use.
- **Inheritance**: When a component method needs to be changed, a virtual method replaced, or a new method needs to be introduced, inheritance can be used. Unfortunately not all component technologies support inheritance.
- **Event-handlers or Callbacks** – the application supplies methods that are called in response to predefined events.

The above variability mechanisms are employed during the assembly of the application. There are also mechanisms for managing variations at run time, such as the use of plug-ins and run-time libraries. A benefit of this approach is that certain application enhancements and upgrades can be provided without having to provide a complete rebuild of the application.

Variations will inevitably also be necessary at places where they were not initially expected; this may demand the introduction of new variation points. However, this has to be a carefully judged decision, because injudicious adding of variation points could negatively affect the architecture or result in components that are unacceptably complex.

Testing: The testing of product line software is made more challenging by the need to exercise fully all variation points. It is usually necessary to develop test harnesses to test individual components, especially during the early stages of core asset development, before a sufficient part of the application framework exists. However because core assets are reused many times in the product line, it pays dividends to ensure that the architecture and the components are designed to facilitate testing, with the ultimate goal being the support of automated testing. Built-in test support is of particular value for applications, such as safety-critical applications, requiring proof of the requirement testing for qualification or certification purposes.

2.4.2.2 Technical Management Practice Areas

The *technical management practice areas* are those concerned with the establishment and support of the development infrastructure. Examples of the technical management practice areas are:

- processes;
- configuration management and change control;
- tools; and
- scoping.

Each of these is dealt with below.

Processes: Product lines are inherently process-intensive and the application and maintenance of the product line must be fully documented in the form of repeatable processes which are subjected to regular review and improvement. As with all forms of industrialisation, automation of process tasks, which are repetitive or labour-intensive and thus prone to human error, is of particular value.

Configuration Management and Change Control: Discipline and precision are essential in the management of the core assets and the configuration of developed products. In addition to the control of source code, all other core assets also require configuration management system and change control procedures. Most of the core assets are likely to undergo change as the SPL evolves and matures, making it important to keep track of all versions of these assets and to record the versions used to construct the application framework and the assembled products (Staples, 2004). Since core assets are shared across products and are used by different stakeholders, change control requires considerably more attention than for a single application.

As Birk, Heller, John, Joos et al. (2003, p. 16) explain: "(The) multitude of different products together with the high number of versions leads to a huge complexity, well exceeding the complexity of the existing software development".

The development tools, middleware frameworks, COTS components, DBMSs and operating systems may also be upgraded regularly. Consequently, it is also necessary to record the configuration of the development environments and platforms used for core asset and product development, deployment and subsequent maintenance.

Tools: Industrial production is characterised by the considered application of tools to the automation of processes. In SPL engineering the challenge is to select a suitable mix of tools to support the product-line development practices. These tools must serve to improve the effectiveness of the developers by relieving them of repetitive, time-consuming and error-prone tasks. Tools can also be configured to encourage, measure, or even enforce adherence to coding and design standards and to extract metrics for management purposes.

In addition to code generation, tools can be applied to tasks, such as requirement engineering, software architecture representation, generation of design documentation, database schema design, development of user interfaces, provision of context-sensitive help and user manuals, defect reporting, and collaboration between developers.

Scoping: *Product line scoping* is the process of establishing the boundaries of the problem domain targeted by the product line. The *product line scope* is a statement of the types of systems to be addressed by the product line, and possibly also the types of systems to be excluded. Put another way, “it defines what’s in and what’s out” (Clements, 2002, p. 28).

The scoping practice area is necessary for the planning of possible variations and variation points, for limiting and focusing the core asset development effort and for identifying the target client base for the product line products.

If the scope of the product line is set too wide, costs, timescales and complexity could escalate. Conversely, if the scope is too narrow, variability options are compromised, limiting the applicability of the product line and possibly resulting in missed opportunities (Schmid, 2002b).

2.4.2.3 Organisational Management Practice Areas

Organisational management practices are those related to the implementation of changes to the organisational structure, the support of the new structure and the non-technical management of the product line. Examples of the organisational management practice areas, which will be dealt with below, are:

- business case analysis;
- customer relationship management;
- funding;
- structuring the organisation; and
- training.

Business Case Analysis: It is pointless to establish an SPL that generates less profit than the development of the products in a series of one-off projects; hence it is essential to precede the product line effort with a detailed business case analysis (Boehm, 1999). This analysis should cover the likely costs of establishing and maintaining the product line, the potential profits, as well as the benefits and the perceived risks (Clements & Northrop, 2002). Lim (1998) points out that this form of software reuse should not be seen purely as a long-term cost cutting exercise, but also as a means of gaining competitive advantage in terms of quality, features, performance and time to market. For the product line to be successful, management buy-in is essential (Birk, Heller, John, von der Maßen et al., 2003) and the business case analysis provides a key means of achieving this.

The extent of the SPL implementation cost is a function of the product line adoption model and the prior existence of material suitable for transformation into core assets. It is unlikely that there will be an advantage in implementing a product line for producing one or even two products. Typically, the break-even point occurs at around three products (McGregor et al., 2002) and only beyond this does the approach really start to provide a return on investment.

Böckle, Clements, McGregor, Muthig, & Schmid (2004) provide a comprehensive treatment of the return of investment for SPLs.

Customer Relationship Management: An SPL organisation should carefully manage the relationship with its customers to ensure that they appreciate both the benefits and constraints that this approach presents for them (Clements, Jones, Northrop, & McGregor, 2005). It is also important that a suitable contractual relationship exists between an organisation and its product line customers. If the value of each development contract is based purely on the level of development effort, the financial benefits provided by the product line will accrue to the customer rather than the development organisation.

Elements of the organisation’s problem domain knowledge might be the intellectual property of one or more customers. Care should be taken not to include such proprietary material in the core assets, as this could result in legal action, especially if the product line customers are competitors.

Staff that interface with the customer should understand the product line well enough to be capable of guiding the customer in the specification of requirements. More often than not, the customer can get what he wants, albeit in a form that has been aligned with the product line philosophy. Although this could prove challenging, it is important not to accept customer requirements without careful consideration, especially those that are not within the product line scope. In such cases, the problem should be addressed in consultation with technical management in order to decide whether the scope can be extended without disturbing the architectural integrity of the product line. Failing this, the organisation could consider accommodating the customer by way of a one-off development – inevitably at a greater cost.

As Clements (1999. p. 5) expresses it: “Marketers can no longer agree to anything the customers want but must instead nudge customers to set their requirements so that they can be fulfilled by a version of the product line within the planned scope of variation”*.* Contrary to some expectations, informed guidance of this type usually results in a more confident and satisfied customer (Clements, 1999).

Maintaining an ongoing relationship with customers can help to maintain the alignment of their thinking with the product line vision and provide further opportunities in the form of support agreements and product upgrades, while serving as a valuable source of domain knowledge and feedback for product line improvement.

Funding: SPL development is an expensive exercise that requires a funding plan. This contrasts with one-off development for which the customer ultimately foots the bill (Clements & Northrop, 2002). In SPL engineering, a considerable portion of the development involves the establishment, evolution and maintenance of the core assets which are used to assemble members of the product line. The SPL establishment costs can be amortised over a number of product line members, and the more products that are built, the more the cost can be spread (Böckle et al., 2004), but predicting the number of members to be produced can prove to be difficult.

Structuring the Organisation: This practice area addresses the structuring of the organisation in order to support the SPL initiative. Fundamental organisational changes are needed when moving from project-centric, single application development to an SPL approach (Knauber et al., 2000). Since these changes involve development practices and organisational structures, they require thorough planning and careful management (Birk, Heller, John, von der Maßen et al., 2003).

Morisio et al. (2000, p. 59) see two distinct activities, according to which the SPL effort can be structured: “Usually a product line is built through domain engineering and application engineering. Domain engineering aims at defining and implementing domain commonalities in a generic product. Application engineering produces individual applications for customers starting from the generic product."

Accordingly, implementation and operation of a product line usually requires the establishment of a two-tiered organisational structure. In this structure, one tier, responsible for domain engineering, develops and supports the core assets and the other tier, responsible for application engineering, uses the core assets to assemble the products (Braun, 1999). This structure also makes it possible for the organisation to gain optimum benefit from its domain specialists by capturing their knowledge in the form of core assets for use in assembling products.

Some recent SPL literature disputes the need for a two-tiered structure and Northrop (2002b) provides evidence of successful organisations that have been successful in implementing product lines within a single unit.

Training: The introduction of SPL engineering to an organisation involves exposure to new practices, different concepts and cultural adjustments (Clements et al., 2005). The training practice area is aimed at addressing the demands posed by these changes on the organisation. The set of product line practices should all be described in the form of processes that form a solid framework on which to establish the training programme.

Whereas a series of one-off projects requires a number of teams of generalist software practitioners, an SPL requires less manpower, principally owing to the man-hour savings accruing from reuse. A team of technical specialists is required for developing core assets, as well as small teams of trained product line integrators for assembling the products. The specialist roles are well defined, making it simpler to identify the training requirements. The product development processes can also serve as a useful basis for training newcomers to product line concepts, and the specialist roles offer a valuable opportunity for mentorship of new developers.

Tool support plays an important role in the automation of SPL processes. To use the tools consistently and to gain full benefit, developers require the appropriate training.

2.5 SPL ADOPTION MODELS

There are various approaches for adopting an SPL within an organisation. Krueger (2002) identifies three adoption models: proactive, reactive and extractive. In the proactive model, the organisation plans and implements the product line and then uses it. In the reactive model, the problem domain and application requirements are initially not yet fully understood, so the core assets are developed based on the current, possibly incomplete, knowledge. The core assets evolve as knowledge of the application and domain requirements increases. In the extractive model, the organisation already has one or more existing products at the time that the decision is made to adopt SPL engineering.

With the proactive and reactive models, the organisation effectively begins with a clean sheet and attempts to establish processes and infrastructure before venturing into production. In the extractive model, the software exists before the processes and infrastructure, a situation which could prove to be awkward to manage.

The proactive model is more of a “big bang” approach and stands to provide rewards more quickly, while carrying more risk. The reactive and extractive models depend on an iterative and evolutionary approach, taking longer to produce rewards while bearing less risk.

2.6 SUCCESS FACTORS

Not only is the introduction of an SPL expensive, it is also disruptive and brings with it numerous risks, making it imperative that the decision makers are fully aware of the factors that enhance the likelihood of a successful outcome. These factors include:

Management Commitment: The commitment of the organisation’s management is necessary to enter into an SPL venture. This commitment is required to provide the manpower, time, finance, training and incentives, as well as to establish a suitable organisational structure (Bühne et al., 2004). This is confirmed by Morisio et al. (2002), who after conducting research on software reuse projects in European companies, concluded that it is essential to have management commitment to achieve the changes necessary for a reuse programme.

Management needs to consider carefully the risks and benefits before committing to an SPL initiative, but once a commitment is made it should be "strong and unswerving" (Clements & Northrop, 2001, p. 516). SPLs are primarily suited to organisations prepared to make an honest, long-term commitment to a particular problem domain (Jacobson et al., 1997) and are unsuitable for those seeking a quick, "hit-and-run" venture for quick financial gain.

Process Orientation: The successful establishment of an SPL and its continuing success rely on the establishment and commitment to high quality processes. The SPL is regarded as "institutionalised" once these processes are considered to be "stable and indispensable" (Böckle et al., 2002, p. 56).

Organisations that are already comfortable with the disciplined application of processes tend to make the transition to SPL engineering with greater ease (Clements & Northrop, 2001, p. 517). Bayer et al. (2001) caution that the effort and costs of establishing SPL processes in an "immature environment" could be prohibitive.

Problem Domain: Another priority for success is to ensure that the chosen problem domain is sufficiently stable (Knauber et al., 2000). The risk of SPL failure is significantly reduced if the problem domain is relatively mature and established. Henninger (1996, p. 124) confirms this by saying: "Research in software reuse has observed that most successful reuse efforts have been achieved using collections of components within well-defined and well-understood domains".

Comprehensive knowledge of, and experience in, the targeted problem domain cannot be valued highly enough. A good understanding of the end-users, environment, habits, customs, terminology, problems, desires and trends greatly enhances the probability of successful product line implementation. Since domain knowledge is essential for success and constitutes a valuable intellectual asset, it is important that it be documented rather than being allowed to reside purely in the heads of individuals.

Product Line Vision: The organisation needs to have a clear and coherent vision for the product line (Knauber et al., 2000). The vision needs to address the present state of the product line as well as its medium- and long-term growth paths (Knauber et al., 2000). The vision is built and evolved in a similar fashion to domain knowledge, through a combination of experience and research. Knowledge that contributes to the vision is derived from customer feedback, conferences, industry publications, web sites, trade shows, sales material from the competition and networking with customers and other industry players. Since the vision encompasses strategies from diverse disciplines, various individuals should contribute to it, such as domain experts, technology experts, marketing personnel and the product line champion (*see* Product Line Champion below).

The product line vision presents a long-term view, allowing the company to plan resources, training and marketing, and to invest in infrastructure. This contrasts with the short-term vision that characterises a project-centric strategy. In a project-centric organisation the vision is to a large degree determined by the company's mix of projects which tend to vary over time.

The product line vision has to be dynamic to take account of changing market forces. It must be reviewed regularly and realigned when necessary. Ideally, there should also be an overall corporate vision that addresses the organisational strategy and encompasses the individual product line visions. The corporate vision should ensure that the company achieves its desired portfolio coverage and that there is no unintended overlapping of product lines.

Product Line Champion: According to Clements and Northrop (2001), a *product line champion* is an individual, or a small group, who has an excellent comprehension of product line principles, a detailed understanding of the problem domain, a clear understanding of the product line vision, and the ability to communicate these. The champion should be involved in all product line decisions and must contribute to a free flow of information between all parties involved. The champion must have the resources, authority and management support required to ensure that the efforts and actions of all parties remain aligned with the interests of the product line.

Schmid (2003, p. 9) suggests that the a product line champion should be "a person with strong social and communication skills who is in continuous contact with the development personal and is in favour of product line development ideas and communicates this strongly and convincingly".

Education Drive: Changes to the structure of an organisation could possibly threaten the comfort zones of individuals and sectors of staff. To enhance the likelihood of success, it is important that people be informed of the reasons for adopting a product line, the likely benefits, the implementation plans and the progress being made (Birk, 2002). It is also important for sales staff to be sufficiently informed to uphold the SPL vision and principles when planning their sales strategies.

Morisio et al. (2000) report on how four European companies that successfully adopted SPL engineering avoided “problems with human aspects” by having a management presence, using consultants, conducting presentations and implementing training plans.

2.7 ADVANTAGES AND DISADVANTAGES

The advantages of SPL engineering over one-off development are derived from the production economies that are made possible by developing each application from a common set of assets in a prescribed manner (Clements & Northrop, 2001, p. 5).

The advantages provided by systematic software reuse, in general, are faster time to market, reduced development costs and risks, and significant gains in productivity and long-term quality. Lim (1998, p. 85) points out that organisations should also recognise the opportunity for strategic advantages to be gained from reuse, “such as entering new markets, increased agility in response to a dynamic marketplace and competitive positioning”. Since SPL engineering specifically extends reuse to all aspects of the software development lifecycle, these advantages are enhanced still further.

Other important advantages are:

- **Uniformity and Standardisation:** Members of the product line have a high degree of commonality providing benefits for the organisation owing to uniformity and standardisation. These benefits extend to development documentation, user manuals, user training, maintenance support, marketing and many other functions.

The uniformity of the products in the product line also presents the opportunity for establishing user groups and for holding user conferences. Provided that these are well run, they can serve as an invaluable source of feedback and general domain knowledge, while acting as a useful communication network.

- **Definition of Roles:** In a project-centric organisation, the workforce is of necessity made up of generalists (i.e. jacks-of-all-trades) because project members are required to take on varying roles as the development lifecycle progresses from initial conception through to maintenance.

  In contrast, SPL developers have well-defined roles and responsibilities, allowing for the optimisation of training resources and tools. This also supports the establishment of specialists, who are used to benefit all SPL members and even other product lines.

  The benefit of using specialists is manifested in improved quality in the development and support of the core assets.

- **Continuous Improvement**: The ongoing reuse of core assets and feedback received from users promote continuous improvement, which will benefit all product line members at the same time advancing knowledge of the problem domain. This is in contrast with the development of a one-off application, which typically terminates once all requirements have been met.

- **Explicit Knowledge Representation:** As a result of SPL engineering, domain knowledge is consolidated and explicitly represented in the form of various core assets and processes. The organisation is thus protected from having its intellectual capital reside predominantly in the memory of individual specialists, with the risk of being lost when an individual resigns. *Explicit knowledge representation*[8] also facilitates the sharing of knowledge and provides a valuable basis for the training of new staff.

- **Product Delineation:** Product line engineering naturally provides the platform for the logical identification and delineation of products, which in turn facilitates the establishment and coordination of a product range, product image and

[8] *Explicit knowledge representation* is a term believed to be coined by the Fraunhofer Institute.

branding. This can also contribute to an organisation's strategic planning and marketing efforts.

It is difficult to achieve significant technical progress without introducing elements of cost and risk; similarly, establishing an SPL will initially involve both of these elements (Bosch, 2000, p. 188). However, a successful SPL effort will, in the longer term, result in significantly lower development costs, reduced risk and improved quality (Clements & Northrop, 2001, p. 23).

SPL engineering deliberately sets out to limit unanticipated and unplanned variability. Processes are specifically established to control changes and to prevent unsanctioned modification of core assets (Brownsword & Clements, 1996). This inevitably translates into a reduction of flexibility when compared with one-off development (Knauber et al., 2000).

SPL engineering poses additional challenges for small and medium-sized organisations. It is apparent that most of the widely-published SPL success stories involve large organisations with ample resources (Gacek et al., 2001). Smaller organisations tend to lack the resources and the long-term planning capability necessary for SPL engineering (Knauber et al., 2000).

Smaller organisations can seldom rely on a steady flow of development orders, making it essential to have a versatile, rather than a specialised, workforce that can easily be redeployed according to demands. Product line protagonists argue that a promising business case should exist before committing to the product line, and having committed to it, the organisation is in a better position proactively and selectively to seek out new jobs that fall within the scope of the product line.

SPL practice requires more careful selection and management of customers. Unlike small organisations, large organisations can usually afford to be more selective of their customers. Small organisations are often very flexible and more willing to produce software precisely tailored to each customer's specifications. They also usually cooperate closely with their customers and are more aware of the customer's specific needs (Knauber et al., 2000). These organisations are thus more vulnerable to the potential loss of flexibility brought about by an SPL, especially since this flexibility is often the particular attribute sought out by their customers. Some small organisations, desperate to secure big orders might be tempted to sell their souls – so to speak – to satisfy demanding customers, making them inherently unsuitable candidates for SPL engineering (Gacek et al., 2001).

Smaller organisations also have certain advantages over larger organisations when implementing SPLs, because their small size allows for quicker communication, making it simpler to instil the product line vision, to establish new processes and to share domain knowledge (Gacek et al., 2001).

2.8 SUMMARY

SPL engineering has evolved out of a longstanding need to bring some of the benefits of industrial scale manufacturing to the field of software development. Product lines are well established in other manufacturing industries, and many of the concepts and practices proposed by SPL engineering have actually existed in software development for some time. However, it is probably the first truly comprehensive approach to systematic software reuse.

An organisation with an established SPL owns a collection of core assets that allow it to consistently produce consistently high quality applications that coincide with the product line scope. The staff members working on the product line have well-defined roles, are appropriately trained and can also be regarded as core assets. They assemble an application according to the product line production plan, starting with an application framework, which preserves the product line architecture, and then implement the application-specific functionality via the carefully provided variation points. The components making up the framework are developed by specialists, continually improved and thoroughly tested. The framework and components are designed to facilitate testing, making all subsequent testing comprehensive, quick and inexpensive. Project plans, requirements specifications, design descriptions, test plans, user manuals and all other application-specific documentation are derived directly from existing, proven product line assets. The development effort for new applications is substantially less than for a conventional single development, so timescales, costs and risks are reduced accordingly.

Market forces encourage organisations to focus on problem domains for which they have acquired knowledge, and to produce applications with a large degree of commonality. SPL engineering provides a proactive approach for systematically capturing the knowledge required to produce these applications in the form of a set of core assets. Substantial production economies result when applications are produced using these assets in the prescribed way (Clements, Donohoe, Kang, McGregor, & Northrop, 2001).

Although the adoption of SPL engineering is a major undertaking that initially demands additional overheads and might require organisational restructuring, it can provide a substantial return on investment and could be the solution for many organisations in their struggle to thrive in a competitive market.

The literature study identifies several technical and organisational issues that require careful attention when considering the implementation of an SPL.

Among the technical issues are the following:

- acquiring, managing, applying and protecting domain knowledge;
- establishing, institutionalising and improving processes;
- selecting and applying tools for automating processes;
- adopting architecture-centred development;
- selecting and applying a suitable component technology; and
- strictly and consistently implementing configuration management and change control.

Among the organisational issues are the following:

- funding of the SPL;
- giving special consideration to customer relationship;
- establishing a compelling business case; and
- selecting and implementing an appropriate organisational structure.

This chapter described how software reuse has evolved and identified SPL engineering as a generally accepted, comprehensive solution to systematic software reuse. The *Product Line Practice Framework* was used as the basis of the description of SPL engineering and, in addition, aspects such as adoption models, success factors and advantages were considered.

Chapter 3. Research Methodology and Design

3.1 INTRODUCTION

This chapter starts by providing the background to qualitative information systems (IS) research in order to contextualise the methods which contributed to the research design. The second part of the chapter makes use of this information to describe and motivate the research design.

3.2 BACKGROUND TO RESEARCH METHODOLOGY

This section covers background and supporting information relevant to the research methods used in this study.

3.2.1 Information Systems (IS) as a Research Discipline

Information systems (IS) is a young, rapidly changing field of study that draws upon a large variety of reference disciplines, such as computer science, mathematics, linguistics, economics, political science, ethics, sociology and psychology (Avison & Elliot, 2006, p. 5). Although IS boasts a rich heritage, it is criticised for being reactive, opportunistic, lacking academic rigour, confused and for suffering from an identity crisis (Avison & Elliot, 2006; Khazanchi & Munkvold, 2000; Pather & Remenyi, 2004).

Debates surface regularly on whether IS serves as a reference discipline for other disciplines (Lee & Liebenau, 1997; Wade, Biehl, & Kim, 2006) or whether IS even deserves to be called a discipline or science (Baskerville & Myers, 2002; Khazanchi & Munkvold, 2000).

Information systems is a dynamic and exciting field of study and distinguished IS research authors, Baskerville and Myers (2002), not only recognise it as a discipline, but are also convinced that current trends show IS to be well positioned to address all of its challenges and criticisms.

3.2.2 The Nature of Information Systems Research

"The goal of Information Systems research is to produce knowledge that enables the application of information technology for managerial and organizational purposes" (Hevner & March, 2003, p. 111).

According to Avison and Elliot (2006, pp. 6-7), IS research "focuses more on interactions between people and organizations (the 'soft' issues) and technology rather than on the technologies themselves". In practice, it is also these *soft issues* that prove to be the most challenging in information systems (Harvey & Myers, 1995).

Although natural science research methods have been successfully applied to IS technology, these methods are "inadequate and inappropriate" for dealing with the soft issues (Lee & Liebenau, 1997, p. 3). This suggests that qualitative methods, intended for the study of people in their "social and cultural contexts" (Myers, 2006), are better suited for the research of IS. At the research paradigm level, a *positivist* perspective is more appropriate for *hard issues* and an interpretive perspective for soft issues (Fitzgerald & Howcroft, 1998). According to Myers (2006), the underlying philosophical assumptions of a qualitative researcher can be positivist, interpretive or critical, so the assumption that *qualitative* is simply a synonym for *interpretive* is incorrect.

In IS research, especially in North America, there is a long tradition of academics employing positivist research methods in order to achieve rigour (Evaristo & Karahanna, 1997). Sometimes this rigour is achieved at the cost of practical relevance, and results in dissatisfaction from practitioner interest groups (Harvey & Myers, 1995). This preference for positivist methods, which aim to present an "a-historical and a-contextual" picture, is also in conflict with the special significance that *context* has in the study of information systems (Harvey & Myers, 1995).

Although information systems research has historically been preoccupied with *hard* technological issues, interest is shifting more towards the soft organisational issues (Darke, Shanks, & Broadbent, 1998; Fitzgerald & Howcroft, 1998; Myers, 2006). There are ongoing comparative debates on the hard and soft research approaches for IS research. Table 3-1 provides a useful summary some of the competing dichotomies between these approaches.

Table 3-1: Summary of Soft versus Hard Research Dichotomies (Fitzgerald & Howcroft, 1998)

Soft	Hard
PARADIGM LEVEL	
Interpretivist No universal truth. Understanding and interpretation come from researcher's own frame of reference. Uncommitted neutrality impossible. Realism of context is important.	**Positivist** Belief that world conforms to fixed laws of causation. Complexity can be tackled by reductionism. Emphasis on objectivity, measurement and repeatability.
ONTOLOGICAL LEVEL	
Relativist Belief that multiple realities exist as subjective constructions of the mind. Socially-transmitted terms direct how reality is perceived and this will vary across different languages and cultures.	**Realist** Belief that external world consists of pre-existing hard, tangible structures which exist independently of an individual's cognition.
EPISTEMOLOGICAL LEVEL	
Subjectivist Distinction between the researcher and research situation is collapsed. Research findings emerge from the interaction between researcher and research situation, and the values and beliefs of the researcher are central mediators.	**Objectivist** Both possible and essential that the researcher remain detached from the research situation. Neutral observation of reality must take place in the absence of any contaminating values or biases on the part of the researcher.
Emic/Insider/Subjective Origins in anthropology. Research orientation centred on native/insider's view, with the latter viewed as the best judge of adequacy of research.	**Etic/Outsider/Objective** Origins in anthropology. Research orientation of outside researcher who is seen as objective and the appropriate analyst of research
METHODOLOGICAL LEVEL	
Qualitative Determining what things exist rather than how many there are. Thick description. Less structured and more responsive to needs and nature of research situation.	**Quantitative** Use of mathematical and statistical techniques to identify facts and causal relationships. Samples can be larger and more representative. Results can be generalised to larger populations within known limits of error.
Exploratory Concerned with discovering patterns in research data, and to explain/understand them. Lays basic descriptive foundation. May lead to generation of hypotheses	**Confirmatory** Concerned with hypothesis testing and theory verification. Tends to follow positivist, quantitative modes of research.
Induction Begins with specific instances which are used to arrive at overall generalisations which can be expected on the balance of probability. New evidence may cause conclusions to be revised. Criticised by many philosophers of science, but plays an important role in theory/hypothesis conception.	**Deduction** Uses general results to ascribe properties to specific instances. An argument is valid if it is impossible for the conclusions to be false if the premises are true. Associated with theory verification/falsification and hypothesis testing.
Field Emphasis on realism of context in natural situation, but precision in control of variables and behaviour measurement cannot be achieved.	**Laboratory** Precise measurement and control of variables, but at expense of naturalness of situation, since real-world intensity and variation may not be achievable.
Idiographic Individual-centred perspective which uses naturalistic contexts and qualitative methods to recognise unique experience of the subject.	**Nomothetic** Group-centred perspective using controlled environments and quantitative methods to establish general laws.

Soft	Hard
AXIOLOGICAL LEVEL	
Relevance External validity of actual research question and its relevance to practice vital, rather than constraining the focus to that researchable by ‘rigorous’ methods.	**Rigour** Research characterised by hypothetico-deductive testing according to the positivist paradigm, with emphasis on internal validity through tight experimental control and quantitative techniques.

3.2.3 Qualitative Field Studies

Qualitative field studies, which include in-depth case studies, ethnography and action research, are suitable for “understanding complex phenomena in situ” (Sim, 1999, p. 68). The case study is undoubtedly the most widely used qualitative research method (Darke et al., 1998; Myers, 2006). Yin defines a case study as: “… an empirical enquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident” (2003, p. 13).

In IS research, ethnography is a qualitative research method that is closely allied to the case study, with the distinguishing factors being the greater level of direct involvement of the researcher, the extended duration of the study and the use of participant observation as a method of data collection (Myers, 1999). In ethnography, the researcher becomes *immersed* in the group being studied in order to observe phenomena in their social and cultural context (Myers, 1999, p. 4). Ethnographic studies usually have considerable depth but limited breadth, providing rich insight into a specific situation (Klein & Myers, 1999). This lack of breadth makes ethnographic research unsuitable for generalisation, but as Myers (1999) points out that generalisation can be applied across several suitable individual ethnographies. Whereas the positivists’ preoccupation with rigour has led to dissatisfied practitioner groups, Harvey and Myers (1995) consider ethnography to be a method that can produce IS research with both rigour and relevance, thus satisfying both the academic and the practitioner interest groups.

Action research, sometimes described as learning by doing, is a mix of theory and practice that attempts to test a theory by applying it to a real problem (McKay & Marshall, 2000). It involves a cyclic process of studying the problem, planning steps to apply a theory in an attempt to solve the problem, applying the steps, measuring the effects, reflecting on the results and if necessary modifying either the application of the theory or the theory itself. Avison, Lau, Myers, and Nielsen (1999) aptly summarise action research as follows: “Action research is an iterative process involving researchers and practitioners acting together on a particular cycle of activities, including problem diagnosis, action intervention, and reflective learning”.

3.2.4 Practitioner Research

When the phenomenon being studied lies within the researcher’s own professional field of practice, the research can be classified as *practitioner research*. Practitioner research allows researchers to achieve greater understanding of their own profession and in so doing possibly influence the transformation of their work environment (Tricoglus, 2001). To paraphrase Tricoglus (2001, p. 136) for the purpose of this study: practitioner research is often the only option available to IS developers because in a practical sense they have neither the time nor the opportunities necessary to study the practice of others. Darke et al. (1998, p 209) provide an insightful definition of practitioner research:

> Practitioner research is often portrayed as having the purposes of professional empowerment and transformation of the self, colleagues and the work context. Such a portrayal positions practitioner research as a means for developing deeper understandings of practitioners’ work that, in turn, provide a platform for changing practice(s).

Although frequently associated with action research (Brooker & Macpherson, 1999), the term *practitioner research* reveals the nature of the researcher’s commitment to the study (Tricoglus, 2001), rather than implying a particular research method. Tricoglus (2001), for example, suggests critical ethnography as an appropriate approach to practitioner research.

Whereas academic research is criticised for favouring rigour over relevance (Khazanchi & Munkvold, 2003), practitioner research is occasionally criticised for lack of rigour (Brooker & Macpherson, 1999; McWilliam, 2004). Tricoglus (2001) suggests the need for the researcher to be reflexive and self-aware throughout the research process in order to improve rigour and to ensure that findings can withstand critical scrutiny. It is also important for practitioner research to go beyond mere prosaic description and provide outcomes that do, in fact, make a difference to the practice (Brooker & Macpherson, 1999; Sim, 1999).

3.2.5 The Importance of Context

Because current IS research typically “focuses more on interactions between people and organizations and technology rather than on the technologies themselves” (Avison & Elliot, 2006, p. 6-7) (i.e. on the *soft* rather than the *hard* issues), the social and historical contexts of the information system research take on special significance.

Harvey and Myers (1995, p. 16) comment on the increasing awareness of context in IS research as follows: “It would seem that information systems researchers are becoming more accepting of the need to adopt techniques which consider the historical and contextual aspects of information systems.”

Context is a topic on which the positivist and interpretivist perspectives differ. While positivists attempt to present information systems from an “a-historical and a-contextual” perspective (Harvey & Myers, 1995, p. 6), interpretivists use methods that aim at “producing an understanding of the context of the information system” (Walsham, 1993, p. 4).

The case study, ethnography, and action research are all methods which not only permit, but actually encourage, the researcher to take account of the research context (Stake, 1995, p. 39; Wolcott, 1999, p. 79). In fact, Yin (2003, p. 13) regards the ability of case study research to deal with the context of the case as one of its particular strengths. The boundaries of a case study are seldom well defined, but this is not usually a problem, because the context of the case is at least of as much interest as the case itself (Yin, 2003, p. 13). Ethnographic methods are particularly suitable for the studying the “highly complex, constantly changing, social context” of information systems (M. D. Myers, 1997) and the practical aspects of action research means that it too is highly context sensitive.

This importance of context in IS interpretive field studies is demonstrated by the use of *contextualisation* as a specific principal or tool to establish how a situation under observation came about, by taking account of its “social and historical context” (Klein & Myers, 1999, p. 73).

The importance of context for this specific study is emphasised by the fact that experience indicates that the challenges of implementing systematic software reuse relate mores to the organisational environment than to technical issues (Fafchamps, 1994; Lynex & Layzell, 1998) .

3.2.6 Prejudice and the Participant Researcher

Whereas the positivist perspective considers prejudice to be a serious barrier to knowledge, the interpretivist perspective views prejudice as “the necessary starting point of understanding” (Klein & Myers, 1999). The interpretivist perspective accepts that it is unrealistic to expect the researcher to rid himself of all prejudice, but it does expect the researcher to acknowledge these prejudices and address them during data collection and analysis (Klein & Myers, 1999; Weber, 2004). Prejudgement and prejudice stem from initial or prior knowledge and true interpretivists recognise that these fulfil a valuable role in human understanding, but emphasise the need to distinguish between “true prejudice” which leads to understanding and “false prejudice” which leads to misunderstanding (Klein & Myers, 1999).

In interpretive field studies of this type, the researcher fulfils a role similar to that of the participants: observing, interpreting and analysing (Klein & Myers, 1999). Consequently, the bias or prejudice of the researcher is an issue which is essential to acknowledge (Darke et al., 1998; Gillham, 2001; Yin, 2003). Gillham (2001, p. 47) cautions the researcher to be aware that participant observation, in particular, can be “fallible and highly selective”. Stake (1995, pp. 133-136) recommends reflection as a tool that can be used by the researcher to address bias during the collection and analysis of data.

3.3 RESEARCH DESIGN

3.3.1 Research Question

The research question is:

What are the issues related to the introduction of systematic software reuse in a small project-centric organisation?

The research question and its subsidiary questions are addressed in detail in Section 1.3.

3.3.2 Motivation for Chosen Research Methodology

The research methodology which I have chosen can be described as a practitioner research-based interpretive case study, which has been influenced by the qualitative research methods of ethnography.

The purpose of this study was to investigate the potential implementation of systematic software reuse in a small project-centric organisation. A preliminary scan of the reuse literature confirmed that, in keeping with IS research in general, software reuse programs tend to be dominated by organisational and contextual issues (Fafchamps, 1994; Lynex & Layzell, 1998; Moore, 1997; Morisio et al., 2002). Issues of this nature are best handled by qualitative methods. This study has an interpretive perspective because it is based on the interpretation of the perceptions of individuals in their natural setting (Orlikowski & Baroudi, 1991; Walsham, 1995).

As the researcher, my long and close involvement with the case study and its context, as well as the influence of observation on the study, suggest the suitability of ethnographic study methods. Ethnographic research is regarded as "an inductive mode of enquiry", and, in character with this, the objectives this study started as little more than a general "statement of interest" (Sim, 1999, p. 67). Owing to the iterative nature of this type of research, the *statement of interest* evolved into a *statement of purpose*, which was subjected to regular scrutiny and revision as the study progressed and new facts were revealed. As the focus of the research changed, the research questions were revised and aligned accordingly.

Where this study may differ from traditional ethnography is in respect of the role of the researcher. Although this study corresponds with ethnography with respect to research methods used and the researcher being *immersed* in the group being studied, an important aspect of ethnography is for researcher to enter the research as an outsider who is suitably primed to identify aspects that are unusual or out of the ordinary. In the literature on ethnography, there is frequent reference to the researcher *going native* (Harvey & Myers, 1995; Wolcott, 1999, pp. 146-156). This implies that the researcher is *non-native* to begin with and becomes *native* for the purpose of the study. In this study I, the researcher, was already a native, and, as such, potentially *lost* the ability to distinguish between what is ordinary and what is out of the ordinary. The labels *autoethnography* and *insider ethnography* are sometimes used for ethnography that covers the researcher's own group or has an autobiographical element (Wolcott, 1999, pp. 170-174).

This study stems directly from a desire to achieve an improved understanding of a phenomenon within my professional workplace. Although, I undertook the study from a purely interpretive perspective, the findings include both interpretive and critical elements that could lead to changes to the workplace. Consequently, the study can also be classified as practitioner research.

As already stated, practitioner research is commonly associated with action research, which Avison et al. (1999, p. 95) summarise as follows: "In action research, the researcher wants to try out a theory with practitioners in real situations, gain feedback from this experience, modify the theory as a result of this feedback, and try it again".

The situation covered by this study provides an ideal opportunity for action research. A real problem situation exists (i.e. the need for systematic software reuse), and as part of the study a theory is to be identified (i.e. a systematic software reuse strategy is determined by the literature study). In a true action research study, the next steps would be to apply the theory, determine its success and, if necessary, modify the theory (or the application thereof) and repeat the cycle. However, instead of applying the theory, I tested participants' perceptions of the theory and reported the findings. A simple comparison between the approach taken for this study and action research is illustrated in Figure 3-1 below.

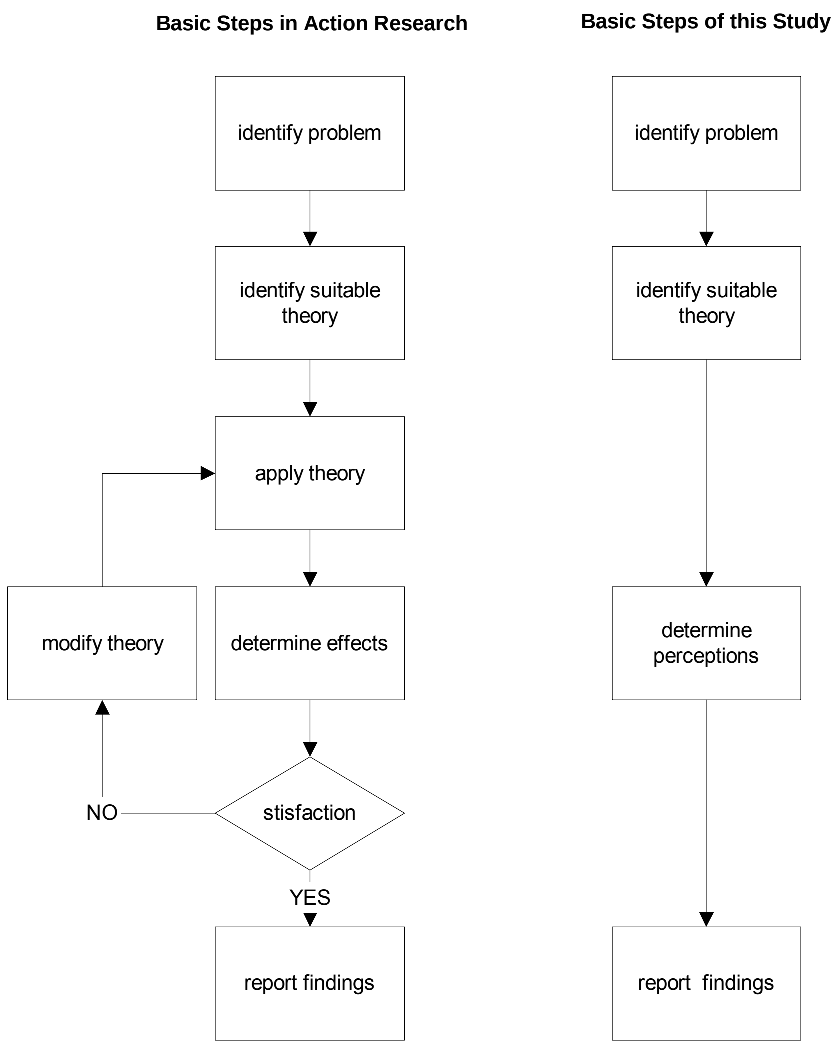

Figure 3-1: Comparison between Action Research and this Study

In summary, this is practitioner research-based interpretive case study employing qualitative research methods.

3.3.3 Scope of the Study

Yin (2003, p. 42) describes how a case study can comprise more than one *unit of analysis*. Although the scope of this case study is the company, we could consider the individual elements of a HUMS as separate units of analysis within this case study. The focus of this case study was purposely limited to the HUMS ground station software element for the following pragmatic reasons:

Ease of Description: Although the ground station software is fairly complex, it has well-defined functions and forms a complete concept that is simple for participants to identify, comprehend and discuss.

Reuse Potential: At least five ground station applications have been developed by the organisation all having a large degree of commonality. Consequently, the chosen case study covers applications with good software reuse potential.

Access to Participants: A number of people have been directly involved in the ground station development projects and they were easily accessible for interview purposes. This is desirable in qualitative research of this nature because, as facts unfolded, it was necessary to revisit certain participants to clarify their responses.

Focus: The purpose of choosing the specific case study was mainly to have a well-defined example that could be used to focus and delimit the scope covered by the interview questions and discussions. There are other development efforts within the company, with good reuse potential, that could also serve as case studies. However, I chose the ground station development projects because of my familiarity with it.

Issues: In spite of the ground station development projects having good reuse potential, a number of issues that conflict with a systematic software development effort are immediately apparent. An example of one of these issues is the relationship with the customer. I wanted the research effort to explore these issues in more detail and possibly expose additional issues.

3.3.4 Data Collection

The study collected data from one secondary and three primary sources:

1. literature study (secondary)
2. semi-structured field interviews (primary)
3. participant observation (primary)
4. an elite interview (primary)

Literature Study: I conducted a study of the published literature to investigate and establish a theoretical framework for the research. The aims of the literature study were to determine the current trends in systematic software reuse, the issues associated with implementing a systematic software reuse programme and the significance of these issues for small and medium-sized organisations as well as project-centric organisations. The literature study also explored the significance of organisational structure in software reuse, the background to SPL engineering and proposed strategies for supporting the implementation of SPLs.

Semi-structured Field Interviews: The main purpose of the field interviews was to uncover facts and determine participants' personal perspectives on the need for reuse in the company and the suitability of a selection of reuse practices. The responses to the field interview questions also contributed to the contextualisation of the research context.

I conducted the interviews using a semi-structured set of open-ended questions. The interview process was semi-structured in that participants were only expected to answer questions relevant to their role in the software development process, but they were encouraged to provide opinions in response to the remaining questions. Open-ended questions allow a participant to respond with as much information as he chooses. Prompts and probe questions were used to steer the conversation and to increase the likelihood of uncovering relevant facts.

Participant Observation: The value gained from participant observation is experience (Wolcott, 1999, p. 46), however, Wolcott (1999, p 44) cautions that the term *participant observation* has become an "umbrella term" used to describe almost everything that researchers do in the field that is not interviewing.

Ebrahim and Sullivan (1995) explain that *participant observation* indicates that the researcher views the study subjectively from within. This contrasts with non-participant observation in which the researcher views the study objectively from a distance (Ebrahim & Sullivan, 1995). The term *from within* could be interpreted in either of two ways: firstly as implying *from within the study* and secondly *from within the phenomenon* (or group) being studied. The first interpretation suggests that the participant observation takes place during the period covered by the study, and the second suggests that it takes place during the period that the phenomenon exists.

The fact that the evidence of participant observation normally is in the form of field notes, implies that the participant observation is a deliberate activity that takes place during the study.

I joined the company in 1988 and have participated in all projects covered by the case study's unit of analysis since 1998. Observations that I made over this period:

- motivated this study;
- sensitised me to the problems relating to the company's reuse problem;
- led me to suspect that certain contextual issues contributed to the problem; and
- guided the research design.

The period covered by this participant observation obviously relates to the phenomenon rather than the study. In order to avoid possibly misleading the reader, I refer to this data source as *personal experience*.

Elite Interview[9]: Taking into account the relevance of context in a qualitative study of this nature (*see* Section 3.2.5), an important element is the contextualisation. In order to supplement the contextual information derived from the literature study and personal experience, I conducted a single elite interview with a colleague who has been associated with the company since 1985.

3.3.5 Interview Process

Systematic software reuse involves technical concepts that are fairly common knowledge among individuals who use these practices, but are otherwise less familiar. This posed a problem for the field interviews for which my intention was to put questions to technical and non-technical participants. As an initial attempt to avoid the problem, two introductory papers were selected for participants to study prior to the interview. However, in a dry-run pilot interview the participant expressed too much uncertainty on the subject, so an introductory paper based on the literature review was used as preparatory reading material for all participants.

In most cases participants were interviewed on site. In one case a participant who has left the company was interviewed at his home. Each interview lasted about an hour, and participants spent about two hours studying the preparatory reading material. The interviews were conducted in English. Although, most of the

[9] An elite interview is an interview with an authoritative person capable of providing a special insight on a topic (Gillham, 2001, p. 63). Wolcott (1999, p. 52) refers to such a person as a "key informant".

participants are Afrikaans-speaking, they are all fluent in English. One participant did however choose to answer certain questions in Afrikaans.

The interviews were recorded using a miniature digital voice recorder, model Sony ICD-P210. The small size and simple operation of this device minimised the intrusive and interruptive nature that is often attributed to conventional tape recorders. The device also allows for automatic voice activation and the convenient organisation of interview responses in participant-specific folders.

3.3.6 Interview Questions

I divided the set of field interview questions into three categories addressing general need and suitability, organisational issues, and technical issues. These categories are summarised in Table 3-2, Table 3-3 and Table 3-4 respectively, which indicate the basic theme of each question, as well as the objectives of related groups of questions. The interview questions for the organisational issues and technical issues categories are based on information from the *Product Line Practice Framework* (SEI, 2006).

Table 3-2: Interview Questions: General Need and Suitability

#	Interview Question	Theme	Objective
G1	Is there a need to make the company's development of software lower risk, more efficient and more deterministic?	Need for software development improvement	To determine the need for systematic reuse
G2	Could the company benefit from software and intellectual asset reuse to a greater degree than it does currently?	Need for reuse	
G3	In general, would you regard software product line engineering as a practical approach to systematic software reuse?	Perception of SPL engineering	To determine the perceived general suitability of SPL engineering
G4	Do you think that it would be practical to implement software product line engineering within the company?	Suitability of SPL engineering	
G5	Do you envisage any negative consequences of implementing software product line engineering for the company?	Negative perceptions of SPL	

Table 3-3: Interview Questions: Organisational Issues

#	Interview Question	Theme	Objective
O1	What do you see as the motivation for maintaining a project-centric structure within the company?	Project-centric structure motivation	To determine the perceived organisational issues with adopting SPL engineering
O2	What do you anticipate that the outcome of a business case analysis would be for a HUMS Ground Station software product line?	SPL business case	
O3	Considering the company's relationship with its current ground station customers, would you anticipate any problems with developing solutions for these customers using a software product line?	SPL customer relationship	
O4	A software reuse effort requires initial funding and ongoing maintenance and support funding. What strategy would the company use to fund a product line effort?	SPL funding	

Table 3-4: Interview Questions: Technical Issues

#	Interview Question	Theme	Objective
T1	Software product line engineering requires the establishment of and strict adherence to processes. Do you foresee that the company would have any problems in this regard?	SPL processes	To determine the perceived technical issues with adopting SPL engineering
T2	Software product line engineering requires considerable discipline with regards to configuration management and change control. Do you foresee that the company would have any problems in this regard?	SPL configuration management	
T3	Software product line engineering requires a comprehensive knowledge and understanding of the problem domain – in this case aircraft health and usage monitoring ground stations. How would you rate the current level of problem domain knowledge within the company with regard to aircraft HUMS ground stations?	SPL domain knowledge	
T4	Do you think that the company would have any problems investing in software development tools, provision of adequate training and enforcing the standardisation and consistent use of these tools?	SPL tools and training	

#	Interview Question	Theme	Objective
T5	The success of a software product line is dependent on having a suitable and robust software architecture. Would this pose a challenge for the company's software developers?	SPL software architecture	
T6	Before investing in the development or purchase of reusable software components, it is necessary to choose a suitable component technology that has a reasonably long-term future. What are your opinions in this regard?	SPL component technology	
T7	Are there any other technical issues, concerning software product lines, which you consider could be a challenge or of interest within the context of the company and HUMS ground stations?	SPL additional technical issues	

3.3.7 Participant Selection

The field interview participants were selected from three groups: organisational management, technical management and software developers. These are all the groups that are involved in, or that directly influence, software development within our company.

The organisational management group consisted of three senior managers who were responsible for the management of the company at the time that the research commenced. The technical management group consisted of two project managers, a systems engineer and a software quality officer who have been directly involved in the projects that are covered by the study. The software developer group was made up of three software engineers who have been responsible for developing ground station software for the projects covered by the case study's unit of analysis.

In total, ten people participated in the interview process. The original plan was to involve twelve participants. However, a senior manager and a system engineer, who were approached, could not find the time to contribute to the study.

As described in Section 3.3.1, the interview questions are divided into three groups – general issues, organisational issues and technical issues. Although the organisational issues are of more relevance to managers, and the technical issues to the technical staff, participants were encouraged to answer whatever questions they had opinions on.

Table 3-5 summarises the groups of interview participants. The direct relationship between roles and identification codes is purposely avoided in order to provide a level of protection for participants' anonymity.

Table 3-5: Field Interview Participant Summary

GROUP	INDIVIDUAL ROLES	IDENTIFICATION CODES
Organisational managers	1 x managing director 1 x sales director 1 x technical director	OM1 – OM3
Technical managers	2 x project managers 1 x systems engineer 1 x software quality officer	TM1 – TM4
Software developers	3 x software engineers	SD1 – SD3

Table 3-6 provides details of the level of education, the interview date, software bias and language of each participant. The *Change* column indicates whether the interview took place before or after the company's change in structure.

Table 3-6: Field Interview Participant Details

CODE	FIELD OF STUDY	TERTIARY EDUCATION	INTERVIEW DATE	S/W BIAS[10]	CHANGE	HOME LANGUAGE
OM1	Business Management Electrical Engineering	Masters Bachelors	1 Aug 2006	No	After	Afrikaans
OM2	Business Management Electrical Engineering	Masters Bachelors	16 Oct 2006	No	After	Afrikaans
OM3	Business Management Electrical Engineering	Masters Bachelors	15 Aug 2006	No	After	Afrikaans
TM1	Electrical Engineering	Masters	14 Mar 2006	No	Before	English
TM2	Electrical Engineering	Masters	28 Aug 2006	No	After	Afrikaans
TM3	Business Management Military Science	Masters Bachelors	5 Oct 2006	No	After	Afrikaans
TM4	Electrical Engineering	Bachelors	9 Sept 2006	Yes	After	Afrikaans
SD1	Electrical Engineering	Masters	7 Sept 2006	Yes	After	Afrikaans

[10] Indicates that the participant has direct experience in software development.

CODE	FIELD OF STUDY	TERTIARY EDUCATION	INTERVIEW DATE	S/W BIAS[10]	CHANGE	HOME LANGUAGE
SD2	Electrical Engineering	Bachelors	24 Mar 2006	Yes	Before	Afrikaans
SD3	Electrical Engineering	Diploma	22 Mar 2006	Yes	Before	Afrikaans
KEY	Electrical Engineering	Masters	29 Oct 2006	Yes	After	English

3.3.8 Interview Transcription

Approximately seven hours of interview data was recorded. Owing to the equipment used, the data was recorded in Sony's proprietary digital voice file format (DVF). Each interview question discussion with a participant was allocated a separate file. The individual files were transferred to a computer hard disk, renamed and systematically organised in a folder per interview question.

As Yin (2003, p. 92) points out, transcription is "a process that takes enormous time and energy". For pragmatic purposes the full interviews were not transcribed, and the transcription was limited to excerpts of particular interest and has been included as an appendix to this document.

3.3.9 Interview Data Analysis

The field interview transcript data was systematically analysed using a basic form of analytic coding, which served to uncover themes in the data and develop categories. Analytic coding involves questioning the data to discover themes, and exploring the themes to develop new categories which allow one to make comparisons (Morse & Richards, 2002). The analytic coding process was implemented via a combination of open coding and axial coding.

The purpose of open coding is to open up the data with the aim of identifying concepts within the data (Morse & Richards, 2002). In order to achieve this, a table was set up for each interview question in a Microsoft Word document. A table row was allocated to excerpts of each participant's answer. Each answer was analysed and Word's highlighter tool was used to identify each segment of text considered to represent a specific and relevant concept. Different colours were used to distinguish between concepts, but no particular meanings were attributed to the colours. A column added to the left of the transcript text was used to record an initial list of the concepts identified. Each identified concept generally consisted of a short phrase, such as *problem with establishing ownership of intellectual property* and *concern over lack of flexibility*.

Whereas the purpose of open coding is to break open, or fracture, the data, axial coding aims to reassemble the data by clustering the themes identified into categories. To achieve the axial coding, an Excel spreadsheet was used to collect the initially identified concept phrases in one column and a second column was used to record the theme relating to each concept. This required a number of iterations for refinement and the Excel data sorting function was useful for grouping the concepts according to theme. Each theme was identified typically by a one or two word title or code, such as *domain knowledge* and *customer management*.

Each concept in the left column of the transcription tables was then translated into its theme code and recoded in a column to the right of the transcription. The transcripts were then studied again in conjunction with the allocated themes, and the list of themes was revised and categorised. An extra column was appended to the extreme right of the transcription tables and used to record the revised themes, which were then used for discussing the data.

3.3.10 Summary

Information Systems is a young and dynamic field of study that draws on a wide range of reference disciplines. It is usually the *soft* issues involving human interaction that produce the challenges in IS practice, and thus qualitative methods are well-suited to IS research. Case study, ethnography and action research are examples of qualitative methods commonly used for IS research. These research methods attribute special significance to the research context and the influence of the researcher on the study. When the researcher focuses on his own profession with the possibility of transforming his workplace, it can be classified as practitioner research.

This study is an interpretive, practitioner case study employing qualitative research methods that are influenced by ethnography. The context of the study is a small project-centric company and the purpose is to investigate the potential implementation of systematic software reuse within this context. Data was collected by way of a literature study, an elite interview, a set of field interviews and personal experience. The literature study contributed to the theoretical framework on which the study was based. The interviews were recorded electronically and key passages were transcribed. The transcriptions were systematically analysed using a basic form of coding.

Chapter 4. Contextual Analysis

4.1 INTRODUCTION

The primary research question addressed by the research study is:

What are the issues related to the introduction of systematic software reuse in a small project-centric organisation?

Experience has consistently shown that it is the “social and organizational contexts of information systems design, development and application which lead to the greatest practical problems” (Harvey & Myers, 1995, p.16). Accordingly, the phrase “in a small project-centric organisation” in the primary research question is intentionally prominent to emphasise the significance of context to this study.

The purpose of this chapter is to provide a detailed understanding of the historical and social factors which have led to the company adopting and maintaining its project-centric structure. The context of the study is analysed using a framework proposed by Korpela et al. (2001). The chapter starts by providing some essential background information to this framework and describes the sources of the contextual information used. It concludes by providing an analysis of the context at the levels suggested by the framework.

4.2 BACKGROUND

The importance of context in qualitative research, and in IS studies in particular, was discussed in Section 3.2.5. Walsham (2000) encourages IS researchers to ensure that their studies consider all the relevant integrative levels of context. Table 4-1 summarises the five levels of analysis suggested by Walsham (2000) and compares them with those suggested by Korpela et al. (2001) and Bühne et al. (2004).

Table 4-1: IS Levels of Analysis

Walsham (2000)	**Korpela et al. (2001)**	**Bühne et al. (2004)**
Society	Society	Market
Inter-organisation	Organisation	Organisation
Organisation	Group/activity	Business unit
Group	Individual	Individual
Individual		

Korpela et al. (2001) propose a framework comprising four levels of analysis and they suggest an extra dimension by addressing each level of analysis from two perspectives: *within* the unit of analysis and *between* similar units of analysis (*see* Table 4-2). In addition, they propose an historical or temporal element, which in effect means that each of the eight (2 x 4) levels of analysis should be considered over time rather than merely as a snapshot.

Table 4-2: Framework Proposed by Korpela et al. (2001)

LEVEL OF ANALYSIS	**INTRA-VIEWPOINT EXAMPLES**	**INTER-VIEWPOINT EXAMPLES**
Society	Country	International
Organisation	Company	Between companies
Group/activity	Project	Between projects
Individual	Programmer	Between programmers

As Korpela et al. (2001) pointed out, not all these levels of analysis are applicable to all studies, but it is important for the researcher to reflect on the significance of each level to the study. I followed the guidelines of Korpela et al. (2001) in analysing the context of this study.

4.3 SOURCE OF INFORMATION

This chapter is based on information drawn from a variety of sources. Details relating to the history and state of the international defence industry were derived mainly from the literature. Details of the local industry were extracted from the literature study, an elite interview and personal experience. Details of the company, the projects and individuals came from the elite interview, the semi-structured field interviews and personal experience. Table 4-3 summarises the source of information according to the levels of the framework proposed by Korpela et al. (2001).

Table 4-3: Information Sources per Framework Level

LEVEL OF ANALYSIS	LITERATURE STUDY	ELITE INTERVIEW	FIELD INTERVIEWS	PERSONAL EXPERIENCE
Society	■			
Organisation		■		■
Group/activity		■	■	■
Individual			■	■

The original literature study revealed the importance of context with respect to software reuse (Böckle et al., 2002; Bühne et al., 2004; Fafchamps, 1994; Gacek et al., 2001; Knauber et al., 2000; Lim, 1998; Lynex & Layzell, 1998). The literature was also extended specifically to provide additional information to support the contextualisation of the study (C. Jones, 2002; Korpela et al., 2001; Rogerson, 1996).

I conducted an elite interview with a colleague who has been associated with the company since 1985. Quotes from this source are identified with the tag [KEY]. Table 4-4 provides a summary of the interview questions used for the elite interview.

Table 4-4: Summary of Elite Interview Questions

INTERVIEW QUESTION		
The company has always been a project-centric organisation. There are various factors that influence an organisation's choice of structure, for example: some organisations tend to mirror the industry in which they operate. Please comment on how the following have influenced the company's choice of organisational structure:		
#		TOPIC
C1	The company's original business model.	Historical context
C2	The nature of the defence electronics industry of the 1980s and early 1990s.	
C3	The business model and development funding model adopted by the state's defence equipment procurement corporation.	
C4	A project-centric structure reduces management effort and overheads.	Contemporary context

Analysis of the field interviews also revealed a variety of contextual information especially at the organisational, group and individual levels. I relied on personal experience and knowledge gained since joining the company in October 1988 to identify information gaps and to guide the literature search and elite interview.

4.4 INTERNATIONAL AND NATIONAL MARKET CONTEXT

This section describes the international and national context that has influenced the defence software market.

During the Cold War, military budgets were massive and this coupled with the fact that neither the defence forces nor their procurement organisations are intended to be profit making organisations, meant that there was very little incentive to improve software productivity. This is confirmed by the fact that the United States Department of Defense software standard in use at the time makes no reference to software reuse (US DoD, 1988).

The Cold War ended in 1991 with the dissolution of the Soviet Union, the easing of international tensions, and the diminishing of the global nuclear war threat (US Congress, 1992). With the end of the Cold War, came significant international disarmament and the down-scaling of military budgets (Abrahams, 2001; Rogerson, 1996), and procurement organisations were forced to spend less and to spend it more wisely. At this juncture, software development costs were consuming a major portion of most defence budgets.

The post-Cold War era brought dramatic changes in the local political climate, which saw the strategic importance of the domestic defence industry wane rapidly. Although the political changes saw local opportunities for the defence industry decline (Abrahams, 2001), these changes also cleared the path for some to the now more competitive, but still potentially more lucrative, international market.

Abrahams (2001, p. 1) reported the situation in the local environment as follows:

> Although the South African defence industry had difficulty in coping with the dramatic decline in the defence budget, there was sufficient prior warning to industries that the cut in the defence budget meant a decline in defence orders. By the late 1980s, South African defence industries were faced with numerous challenges. They needed to convert, diversify and/or increase exports.

4.5 ORGANISATIONAL CONTEXT

This section describes the company within the context of the local industry.

Our company has about ninety employees and produces avionics systems for military aircraft. The company was founded in the mid-1980s as small breakaway group from a large national corporate. Most companies in the local defence industry at the time relied primarily on contracts from the state's armaments procurement corporation. However, the procurement corporation blacklisted our company in an attempt to discourage further breakaways from fragmenting and destabilising the industry.

> [The procurement organisation] had a clear picture for work distribution within the industry and various companies were identified for building up expertise in critical defence areas. With [us] on the outside, this meant that only fragments of bigger contracts would be given to [us] with no significant area of expertise being earmarked for the company. [KEY]

Only in the early 1990s was the company removed from the blacklist and permitted to compete alongside the mainstream defence companies.

Our company's business model and structure, like most other companies in the local defence industry, was influenced by the procurement corporation's project-based contracting model.

> [The company] evolved as a project-centric organisation as from the start it survived on work packages, consulting and engineering studies. All work was broken down into tasks and to a large extent costing was based on an hourly rate. [KEY]

In the local defence industry a contracted company was typically responsible for all aspects of the contract from initial concept to production. Consequently, our company, like many of our competitors, was established as a multidisciplinary company, employing various types of engineers – mechanical, electronic, control, software, industrial and aeronautical.

> People were recruited on the basis of what they knew and not to be trained for a specific role. No provision was made for a synergistic company in which people were focused in any one area with a common vision. [The company]

> was mercenary in the sense that work was accepted from all possible engineering areas, from market studies to the building of [mechanical] prototypes. [KEY]

Each contract was run as a project, typically funded according to level-of-effort plus cost of materials, managed by a project manager in our company and monitored by a project manager in the procurement corporation.

> All orders were linked to a project account and there was no cross-subsidisation. This business model created an efficient company with talented people who had a diverse knowledge, but with a structure that had no depth – a jack-of-all-trades company. [KEY]

In order to survive after the political changes in the early 1990s, the company needed to secure international contracts, which motivated it to increase the level of specialisation and to become more competitive. Although the international defence market was also depressed at the time owing to budget cuts which resulted in downsizing and merging (C. Jones, 2002), new opportunities were presented for small efficient and dynamic subcontractors that could assist the big international conglomerates to fulfil the, now much leaner, major contracts at a profit.

Our past history of successful involvement in the development of aircraft health and usage monitoring systems (HUMS) coupled with a burgeoning interest in the international avionics industry in HUMS, made this a natural choice for our area of specialisation. In spite of its successful specialisation in HUMS, the company continues to maintain its project-orientation structure. Although software development forms an increasing part of the business, the company remains multidisciplinary because it also designs and builds the on board computer, or DAU, which involves electronics, harnessing and mechanical housings.

There are usually between six and ten developers involved in the company's ground station development at any time. From a software development perspective, it would thus be considered as a small software development unit.

In May 2006, the ownership of the company was transferred from a small group of directors to a large international company. This event was followed by a noteworthy structural change; the introduction of a matrix structure. Three line functions –

electronics development, embedded software, and ground support software – were added with a line manager for each. If one views the current projects as vertical elements, the new line functions can be seen as horizontal elements that intersect each project (*see* Section 1.1.4). The function of the line managers is to supply suitable human technical resources to the projects and to ensure that these resources receive training and use the appropriate technology. Line managers also have the responsibility to encourage intellectual asset reuse as well as the consistent adherence to processes.

The change to a matrix structure should, over time, address some of the undesirable symptoms of the project-centric environment, but it falls short of providing the ideal environment for systematic software reuse.

4.6 PROJECT AND INDIVIDUAL CONTEXT

This section describes how projects function within the company and how individual function within the projects.

Projects in our company typically run for a period of between eight and twenty-four months and involve a fulltime project team of between eight and sixteen members. Defence software projects almost always involve lengthy bidding, contract negotiation and specification phases, and as a result, by the time that the contract is eventually awarded, the project is “immediately under schedule pressure” (C. Jones, 2002, p. 26). The pressures are aggravated by the budgetary cutbacks and the now fiercely competitive market conditions.

In a project-centric environment, the project manager reports directly to senior management and has relatively unrestricted control of the project (US Air Force, 2003). The project manager in a commercial company such as ours is tasked with maximising the project profit while delivering a product of acceptable quality. The project manager is responsible for the decisions on how best to reach the target profit, with limited interference from senior management.

In a project-centric environment, the developer effectively *belongs* to a project for its duration and is affected by the project’s short-term, profit-driven goals. From a software development perspective, these short-term goals, coupled with the typical schedule pressures, have the following consequences:

- Software development tools are only bought if they are essential to the project's success.
- Project schedules and budgets can seldom accommodate the training of software developers, so projects use the skills and tools available. A consequence is that projects follow their own inclinations and the resulting lack of uniformity creates a barrier to asset reuse.
- There are few incentives for projects to collaborate on the selection of tools or the scheduling of training, making the sharing of assets between projects impractical.
- Projects hold on to key developers for fear of losing them to other projects. This situation not only limits a developer's career path, but also limits the cross-pollination of ideas between projects.
- Developers are expected to fulfil a variety of rolls on a project. This situation favours individuals with general rather than specialist skills, which in turn limits the level of excellence attainable by the project and thus the company.
- The processes and standards adopted by projects are generally high, but they are to a large degree dictated by the customer. This reduces the payback of process automation and reduces the potential for reusing assets.

These consequences indicate that a project-centric environment is incompatible with systematic software reuse and the optimisation of the company's performance.

In a project-centric company, all development work takes place only under the control of a project, so if reusable assets are to be developed they need to be project by-products. In some cases, project-centric companies use internally funded research and development projects to generate reusable assets for externally funded projects. Historically, our company has funded very few (if any) research and development projects of this nature.

Many of the above issues were confirmed by the filed interview participants:

> [We are] used to processes and procedures to the extreme … actually mind-bogglingly tedious stuff that they really should have tools [for]… it's dehumanising to get people … engineers, who are qualified to do engineering work, to have them sit and do documentation by hand. [SD1]
>
> … it always boils down to "why can't you do the job with the tools we have?" [SD3]
>
> For sure in the past we have … sort of not invested sufficiently in tools and all of that. We have skipped on that. We have tried to maximise our cash flow. [OM1]
>
> … the project manager doesn't want to send off all the software [team] members for a course on new tools … there's no time. There's never time for training. [TM4]
>
> … I mean my goal [as a project manager] is just to win the race and training and that (must wait). [TM2]
>
> Because we are project-based, the project managers don't want to invest in the creation of tools that will lead to long term benefits. [TM4]
>
> [Currently] we're wasting resources. Software people are hard to come by and they're scarce … and if you waste a resource like that it's not in the company's interest. That's one of the big disadvantages [of a project-centric structure]. [TM3]
>
> [In 'n "project-centric" struktuur] jy het baie goed verloor ... jy kyk nie na mannekrag nie, jy kyk nie in die sin van enige opleiding vir die mannekrag nie. Jy voel vere vir die mannekrag en daai goed. Daar's baie goed wat op die grond val ten opsigte van jou mens bestuur. In daardie ["project-centric"] approach is dit a kwessie van jy lewer en klaar. [OM3]

With all these negative consequences, one might wonder why a company would persist with this structure. However, there were good reasons for establishing and maintaining this structure. As described earlier, the project-centric structure was suitable for operating in the defence market, which previously had fewer schedule pressures and no real software reuse incentives. In the past, there were periods when the company attempted to establish more of a matrix structure by establishing a software and a hardware division, but these disappeared when business dropped off. A project-centric structure is suitable for a company in survival mode that needs to minimise costs, especially when the company cannot afford to be selective of the contracts it accepts. This structure carries very little in the way of overheads, and each project serves to encapsulate and limit the extent of risks.

> I think we've come to the conclusion that it's been suitable for the era after [previous owners], where we started off with virtually nothing and we had to get new clients and build a track record and all of that from … very quickly. So I think it was ideally suited for that era and the size of the company and all of that I think now's the right time with being part of [new owners] and also thinking a bit bigger and long-term … it's the right time to change. [OM1]

4.7 SUMMARY

Context is an important aspect of qualitative IS research and Walsham (2000) encourages IS researchers to address all aspects of the context of their studies. The phrase *in a small project-centric organisation* in the primary research question acknowledges the importance of context in this study.

This study makes use of a framework proposed by Korpela et al. (2001), which expands on Walsham's (2000) suggestions. The contextualisation makes use of data derived from four sources: the literature study, the field interviews, an elite interview and personal experience. It places the study in its social and historical context by describing factors at the societal, organisational, group, and individual levels which contributed to the company's project-centric structure.

Chapter 5. Data Analysis

5.1 INTRODUCTION

This chapter addresses the data analysis of the field interview transcripts. The results of the analysis are presented according to themes identified during the coding process. The themes are organised into two classes: the primary themes and the emergent themes. The primary themes, relating directly to the interview questions, are discussed first, followed by a discussion of a selection of relevant themes that emerged during the analysis. The two classes are divided into logical categories and examined in detail using quotes from the interviews where appropriate.

5.2 OVERVIEW

In Chapter 4, the context of the study was explored from a social and historical perspective. In this chapter, a view is presented based on current perceptions relating to the adoption of a systematic software reuse strategy within the case study context. Information for this chapter was drawn from the semi-structured field interviews conducted with the sample group described in Section 3.3.7.

The field interviews were based on a fixed set of questions described in Section 3.3.6. The questions were open-ended and participants were encouraged to discuss their answers. The discussions were recorded and transcribed as explained in Sections 3.3.5 and 3.3.8 respectively. The transcripts were systematically analysed according to the basic coding strategy described in Section 3.3.9. This approach produced two classes of themes, firstly, those that relate directly to the topics of the interview questions, and secondly, the additional themes that emerged during the analysis.

Each class comprises three categories. The primary themes class consists of the same three categories as the interview questions. The emergent themes class is organised into project-centric development environment themes, SPL development environment themes and themes that allow for comparison between the two environments. The organisation of theme categories is depicted in Figure 5-1 below.

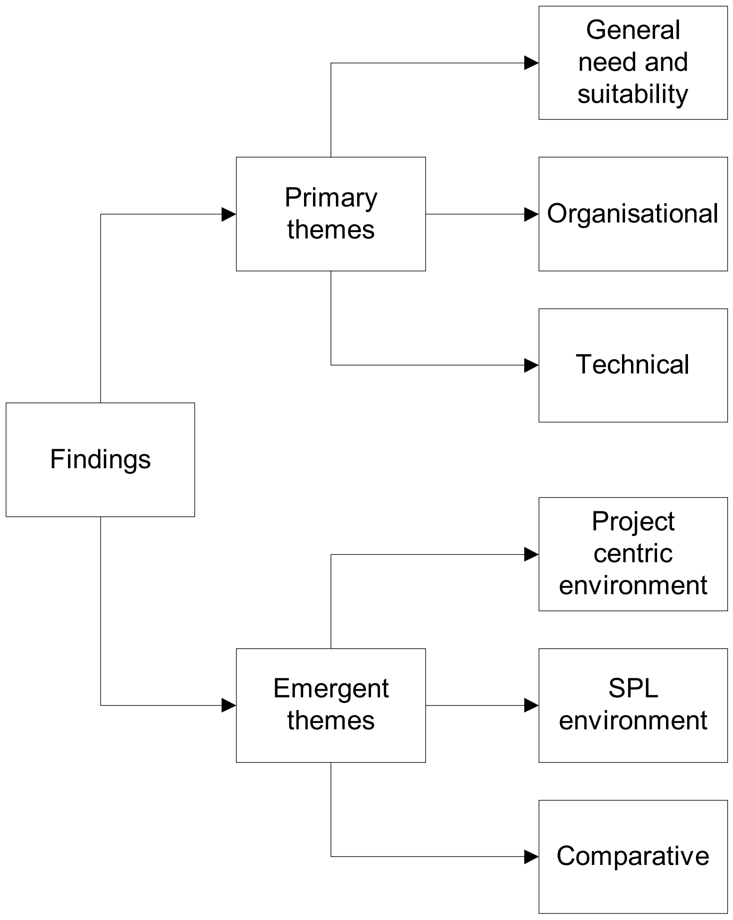

Figure 5-1: Organisation of Data Analysis Theme Categories

5.3 PRESENTATION OF FINDINGS: PRIMARY THEMES

In this section, the findings relating to the primary themes are discussed in the following three categories:

- general need and suitability;
- organisational; and
- technical.

5.3.1 General Need and Suitability

The *general need and suitability* category is made up of the following primary themes:

QUESTION	PRIMARY THEME	SECTION
G1	Need for improvement to the software development process	5.3.1.1
G2	Need for reuse	5.3.1.2
G3	Perceptions on practicality of SPL engineering	5.3.1.3
G4	Suitability of SPL engineering	5.3.1.4
G5	Negative perceptions of SPL engineering	5.3.1.5

5.3.1.1 The Need to Improve the Software Development Process

To determine the participants' perceptions of the need to improve the current software development process, the question asked was:

G1: *Is there a need to make the company's development of software lower risk, more efficient and more deterministic?*

Eight[11] of the ten participants answered this question positively. The most quoted reasons related to the need to address the time, cost and risk pressures currently experienced on projects.

> All the projects I've been working on are late. [TM3]
>
> There is a big cost pressure from our customers. [OM1]
>
> We have to be able to meet out targets and our dates a lot better than we are doing. …I think the time of giving a date and just missing it by two … three weeks is long gone … especially in the aerospace industry. [TM2]
>
> … most of our projects are in high risk and especially in software development we are never in time, or on budget and in time. [TM4]

The participants that gave negative responses did so in what clearly appears to be an expression of frustration with the shortcomings of company's project-centric strategy and its short-term outlook rather than a genuinely negative opinion.

[11] Three organisational managers, four technical managers and one software developer

> … if you think short-term then there might not be any need because you want to make as much money as you want in the shortest amount of time. So it depends on the management. If they want to think long-term … [SD2]

> I'd say there is no serious need due to the nature of the projects. [SD3]

One software developer suggested that as long as the company is contracted to conduct projects on a level-of-effort[12] basis, there is no real incentive to attempt to work more efficiently.

> Ja I think you know [our organisation] has got this strange environment in which they operate and they get paid by the hour for their development work and they are just a small fish in a big pond full of big fishes that do the same thing. So I think in a competitive environment where your development cost is an expense to you. In the environment where they are … you know that's their business … selling man-hours. [SD1]

The latter view fails to acknowledge the time and cost pressures being experienced by many of the current projects.

To summarise, the majority opinion confirms that there is a definite need to make the company's software development more efficient and more deterministic. The rationale behind this opinion is the need to relieve project pressures of time, cost and risk.

5.3.1.2 The Need for Reuse

To determine participants' perceptions of the need for software reuse in the company, the question asked was:

G2: *Could the company benefit from software and intellectual asset reuse to a greater degree than it does currently?*

Two technical managers expressed the view that the company currently gains very little from reuse, and eight[13] of the ten participants agreed that the company could benefit more from reuse. One software developer gave the negative response again

[12] In this form of contracting the contract price is determined by estimating the amount of effort (in man-hours) required to complete the contract.

[13] Two organisational managers, four technical managers and two software developers

as an expression of frustration with the company's persistence with its project-centric strategy, and an organisational manager expressed doubt that a reuse programme would be practical. The latter opinion was based on an experience with a failed attempt to implement a preferred-parts list for machine screws in the company's mechanical designs.

One software developer expressed disappointment that some software reuse was happening in the company, but was going unnoticed.

> I mean ... the management don't even know about the reuse of software that we've reused so far. I've reused software many many times. [SD3]

The problem here is that experience has shown that for software reuse to work it needs to be systematic (Laguna et al., 2003; Morisio et al., 2002) and if management is unaware of it, it is unlikely to be systematic. Systematic reuse requires management support in terms of planning, the implementation of processes, and the allocation of resources.

Four[14] participants identified the project-centric structure as presenting a potential barrier for the implementation of a reuse strategy.

> … we will have to move away from the project structure … [OM1]

> ... if you have a strictly project-driven company then … they [the projects] go in their own direction … [SD2]

In summary, there was general agreement between the participants that the company stands to benefit by increasing software and intellectual asset reuse beyond its current limited levels.

5.3.1.3 The Perceptions on Practicality of SPL Engineering

To determine participants' perceptions on the general practicality of SPL engineering, the question asked was:

G3: *In general, would you regard software product line engineering as a practical approach to systematic software reuse?*

[14] One organisational manager, two technical managers and one software developer

All participants agreed that SPL engineering appears to be a practical approach to systematic software reuse, however four[15] participants voiced reservations. Two[16] of these were concerned that SPL engineering might only be applicable to large organisations:

> I think it's a comprehensive approach. I just don't know how big a company must be to be able to do this properly. [SD1]

> Maybe it's a question of critical mass. It might become more do-able if we were doing 10-times as much ground station work. [OM2]

One participant, an organisational manager, was concerned that the transition necessary to take our company from its current situation to an SPL environment would possibly be too great.

> … within limits. I think the full Monty that you explain here [in the supplied paper] could be quite drastic from where we are here. In between there is just getting used to reusing and then going to the tools and methods and all of that and at the end having a total factory. [OM1]

A technical manager mentioned that the challenges of implementing of an SPL might originate from the customer.

> … the constraints normally come from the customer. So to some extent some customers will not allow the product line mode to work very well … one needs to try and convince the customer that he needs to go in this direction and the long term benefits will pay off for him … [TM1]

Although there was general consensus that SPL engineering provides a practical approach, there were reservations concerning its general applicability and implementation. The reservations concerned the suitability for small companies and customer acceptance of the concept.

5.3.1.4 The Suitability of SPL Engineering

To determine participants' perceptions on the suitability of SPL engineering for the company, the question asked was:

---

[15] Two organisational managers, one technical managers and one software developer

[16] One organisational managers and one software developer

G4: *Do you think that it would be practical to implement software product line engineering within the company?*

Although not all participants answered this question positively, none actually answered it negatively. Among the positive responses were

> I think [the company] is now at the point where they have quite a nice set of products and it can refocus and regroup almost and then say this is what the strategy should be and we should look at moving into product lines. [TM1]
>
> There's a lot of generic functions in HUMS and what you guys have done here can definitely be transferred. The detail [of other systems] will differ but a lot of the scope will not change at all. [TM3]

Most participants qualified their response with provisions or reservations. One of the organisational managers felt that SPL engineering is probably more suitable to the HUMS ground station than it is to the company's on board embedded software. The argument being that the onboard software is customer "unique". This opinion is somewhat contentious, if one takes the HUMS DAU as an example, its primary function is to acquire, filter and store data and it typically has no human user interface, making it a simple system to model. In comparison, ground stations have a very broad human user interface consisting of menus, graphical displays, tabular displays, event logs and reports, making them considerably more susceptible to individual customer requirements, and thus a greater SPL challenge.

Among the reservations expressed were:

> I think it's going to be more difficult because we are already in the process … I mean we've already got products … we've already got a lot of software. I think it's going to be more difficult to do it. But the fact that its going to be more difficult does not mean we should not do it. [TM2]
>
> What's also important is that we have a couple of prima donnas that want to do things their way and that has to stop. An architecture has to be set up and we have to fit into that … even if you don't agree with it one hundred percent. [TM2]

One of the organisational managers saw the customer relationship and loss of flexibility of the product line software as potential challenges.

> In a typical [customer] specification which has got every requirement identified and you have to meet each of those requirements it's going to be difficult. You might have to change so much to fit into that spec that it's not worthwhile. [OM1]
>
> One that I could see is the loss of flexibility to customer requirements … ja exactly. There's quite a great variability in customer requirements you know and to have your variation points for all this variability might defeat the object. I think it's once again suited to something that doesn't vary too much from a central core concept. [OM1]

An organisational manager brought up the ever-important issue of software qualification and certification. In the aerospace industry, especially when human safety is involved, considerable evidence is demanded by the certification authority. The type, format and content of this evidence differ according to the authority and the level of certification necessary. Software certification is a costly exercise which makes exploitation of reuse opportunities attractive. The challenge lies in establishing a strategy, acceptable to certification authorities, that optimises the *level* of reuse, thus reducing the effort and cost of certification.

> The problem is maybe not with [our company] specifically, but it's a bigger problem in the aerospace industry where you've got qualification and certification because each customer's requirement is different. [OM2]

Although participants had various reservations, these reservations largely concerned the anticipation of SPL adoption issues typically experienced during the transition from a project-centric to an SPL *mentality* (Bühne et al., 2004).

5.3.1.5 The Negative Perceptions of SPL Engineering

To determine what negative perceptions participants might have on SPL engineering, the question asked was:

G5: *Do you envisage any negative consequences of implementing software product line engineering for the company?*

OM1 was concerned that because our customers have differing requirements, SPL engineering might restrict the flexibility that we offer our customers at present. From another perspective, he felt that attempting to cater for a wide variety of customers' requirements would make the number of variation points excessive. He did, however, believe that the ground station element of the HUMS solution was more suited to an SPL solution.

One of the technical managers suggested that there are no real negatives but that we would need to keep an open mind on when to attempt total reuse, when to redevelop, or when to do something in between. He added:

> If you design and code [everything] from day one every time … well you are not going to make that much money as when you start reusing… and you're going to make the same mistakes again. [TM3]

Another technical manager considered the negatives to be short term.

> In the short term yes … for sure the rattling of the cages and that is going to have a short-term [negative] effect and because we already have software I think it's going to … in the short-term be a bit of a delay in the process … it will take certain things longer to happen because people are not fully aligned with that. But I think in the longer term you'll get the benefit. [TM2]

A third technical manager suggested that SPL engineering would result in a long-term increase in overheads. This opinion was unexpected because the primary long-term benefits of the product line approach are considered to be reduced overheads and increased profits (Northrop, 2002b; SEI, 2006).

TM4 felt that management would have to change "drastically" to accept that some of the work force would be tied up developing reusable assets and not directly working on a paying project.

According to OM3, for SPL engineering to work it will require a combination of strong leadership and good planning.

All software developers predicted that management would adopt an unfavourable view on the short-term effects that SPL engineering would have on company profits.

Most of the issues that were raised by participants are actually organisational issues relating to SPL adoption. An exception is the concern over lack of flexibility in addressing customer requirements. This was also identified in the literature study as one of the potential disadvantages of SPL engineering that is of particular concern to small organisations. The benefits promised by SPL engineering are dependent on adopting a product-oriented approach as opposed to the typical project-oriented single system approach (Bühne et al., 2004). This implies swapping the unrestrained flexibility of single system development for the carefully planned and managed variability of SPL engineering.

5.3.1.6 Summary of Findings: General Need and Suitability

Table 5-1 presents a summary of the finding relating to the *general need and suitability* category of interview questions.

Table 5-1: Summary of Findings: General Need and Suitability

#	INTERVIEW QUESTION	THEME	FINDINGS
G1	Is there a need to make the company's development of software lower risk, more efficient and more deterministic?	Need for improvement to the software development process	The majority opinion confirms that there is a definite need to make the company's software development more efficient and more deterministic. The rationale behind this opinion is the need to relieve project pressures of time, cost and risk.
G2	Could the company benefit from software and intellectual asset reuse to a greater degree than it does currently?	Need for reuse	There was general agreement among the participants that the company stands to benefit by increasing software and intellectual asset reuse beyond its current limited levels.
G3	In general, would you regard software product line engineering as a practical approach to systematic software reuse?	Perceptions on practicality of SPL engineering	Although there was general consensus that SPL engineering provides a practical approach, there were reservations concerning its general applicability and implementation. The reservations concerned the suitability for small companies and customer acceptance of the concept.
G4	Do you think that it would be practical to implement software product line engineering within the company?	Suitability of SPL engineering	Although participants had various reservations, these reservations largely concerned the anticipation of SPL adoption issues typically experienced during the transition from a project-centric to an SPL *mentality.*
G5	Do you envisage any negative consequences of implementing software product line engineering for the company?	Negative perceptions of SPL engineering	Concern expressed about loss of flexibility in addressing customer requirements.

5.3.2 Organisational Themes

The *organisational themes* category of primary themes is made up of the following individual themes:

QUESTION	PRIMARY THEME	SECTION
O1	Motivation for maintaining a project-centric structure	5.3.2.1
O2	Making an SPL business case	5.3.2.2
O3	SPL customer interface management	5.3.2.3
O4	SPL funding	5.3.2.4

5.3.2.1 Motivation for Maintaining a Project-Centric Structure

To determine participants' opinions on possible motivation for the company to maintain its project-centric structure, the question asked was:

O1: *What do you see as the motivation for maintaining a project-centric structure within the company?*

All three of the software developers interviewed and two of the technical managers shared the view that senior management would be motivated to maintain a project-centric structure by its current success at making low-risk, short-term profits,.

> I think what you're up against here is the company's been successful based on this [project-centric] model and people [senior management] often have the attitude "if it's not broken why fix it? …The project structure has worked even though you haven't ended up with generic products because perhaps of lack of competition in the HUMS sector. [TM1]

> I think in principle, financial control and established [processes] … just the way that they are used to doing it … the approach that they have been doing for the past twenty years and you know to change it would be fairly dramatic... [SD1]

> I'm all for making your life simpler. If I was a manager or project manager or the MD of the company, I'd also want to make my life as simple as possible [by maintaining a project-centric structure]. … So I think that … you know

> that's definitely a motivating factor to have a project-centric company in that sense. But I think that can only go so far. [SD2]

> I think it's a high comfort zone for people. People like having their own project and like focusing on their own project. I think it's more a human thing. [TM3]

In contrast, all three of the organisational managers and the remaining two technical managers were of the opinion that there is no longer any motivation for maintaining a project-centric structure.

> None… ha ha ha … I don't think there is a motivation to keep it. [TM2]

> Well we don't want to maintain it. I think we've come to the conclusion that it's been suitable for the [past] era … [OM1]

> It's going to be a matrix system and that's the way we're going to go. [OM3]

Some of the participants used this question to vent criticism of the project-centric structure:

> [Currently] we're wasting resources. Software people are hard to come by and they're scarce … and if you waste a resource like that it's not in the company's interest. That's one of the big disadvantages [of a project-centric structure]. [TM3]

> [In 'n "project-centric" struktuur] jy het baie goed verloor ... jy kyk nie na mannekrag nie, jy kyk nie in die sin van enige opleiding vir die mannekrag nie. Jy voel vere vir die mannekrag en daai goed. Daar's baie goed wat op die grond val teen opsigte van jou mens bestuur. In daardie approach ["project-centric"] is dit a kwessie van jy lewer en klaar. [OM3]

> But what they do now is reduce the time and reduce the cost and then we upset the customers because we're late and over budget and we upset management… [Everyone on the project] is under extreme stress and extreme unhappiness. [TM4]

This question uncovered general rejection of the project-centric structure. All operational managers rejected the idea of maintaining the project-centric structure, despite suspicions from the other groups that they might want to maintain it.

5.3.2.2 Making an SPL Business Case

To determine participants' anticipations on the outcome of a business case analysis, the question asked was:

O2: *What do you anticipate that the outcome of a business case analysis would be for a HUMS Ground Station software product line?*

Three[17] of the nine participants that answered this question were convinced that the outcome of a business case analysis would be positive, and a further three[18] were also positive, but not totally convinced. Among the factors cited in support of a positive business case outcome were:

- Many of the functions associated with a HUMS ground station are generic, which makes the application a suitable candidate for a software product line.
- The increasing concern for aircraft safety makes the supply of HUMS solutions a growing industry.

Organisational manager, OM1, argued that it would be difficult to quantify the costs, benefits and disadvantages accurately in order to construct a business case, so the decision to adopt an SPL approach would have to be a "gut feel thing" and "a leap of faith".

There was concern among several participants that management's short-term outlook and a reluctance to invest in tools, training and R&D would make it difficult to convince senior management of the SPL business case. Unique and incompatible customer demands, the limited market, and the limited resources of the company were also seen as potential threats to a convincing business case.

Participants raised some issues that might threaten the chances a successful SPL business case. These are mainly organisational issues relating to SPL adoption and the perceived short-term project-centric view taken by senior management.

[17] Two technical managers and one software developer

[18] One organisational manager, one technical manager and one software developer

5.3.2.3 SPL Customer Interface Management

To determine participants' anticipations of the difficulties that an SPL approach might pose for customer relationships, the question asked was:

O3: *Considering the company's relationship with its current ground station customers, would you anticipate any problems with developing solutions for these customers using a software product line?*

During the interviews, various customer-related problems were highlighted, primarily concerning the business relationship with the customer and the ownership of intellectual property.

In the defence industry it was fairly common for software development contracts to be based on level-of-effort plus cost of materials. This model is not economically suitable when the contractor's intellectual property forms a significant part of the product. In particular, if the product is a member of an SPL, the benefits derived from reuse should accrue, not only to the customer, but also to the contractor. This point resulted in a considerable debate.

> … if you can do a job in three months and you ask two man-years of money for it they would want to start squealing … they are actually paying for your background IP (intellectual property) … that's what they have to get used to. [OM1]

> [The difficulty of convincing a customer to pay for 100 hours when it actually took 30 hours] … That's why we have to have intellectual property so that the 30 hours that we sell comes at a premium … it's not just paying the rate. It's not just the 70 dollars or whatever or 80 dollars an hour, it's like 120 dollars an hour. [TM2]

A related problem arises when, after close cooperation between the contractor and the customer to create a product, it becomes unclear which party actually contributed the specific elements of the intellectual property embedded in the product. This situation could prove to be even more problematic when the contractor goes on to produce a similar product for a different customer.

> … we already have a problem with our customer in [country] on IP because he paid for the customisation part but what we said to him was that we're not

> going to give him the [ownership of] IP to the whole product because there is a lot of background knowledge that went into this product . So maybe it will be better if you follow that strategy because you can actually prove that you've got a product line and you only customise a little bit. So you can say to the customer "look you only pay for this customisation bit. The rest is clearly ours. [OM1]

A potential disadvantage of the product line approach is that after adopting it, the organisation possibly becomes less flexible and less responsive to customer requirements.

> …maybe that's one of the reasons why we have been successful is that we've been so flexible in meeting customer requirements. I know the bigger companies like [name of big defence contractor] have the tendency to say [to the customer] "This is what we've got … take it or leave it". [OM1]

This flexibility or willingness to please can become problematic when the customer becomes too involved in the development process, exacerbating the contracting model and IP problems mentioned above.

> … I am afraid to say it, but most [of our] customers [do] micromanage. [TM3]

According to one software developer, it would be more in our interest to maximise development costs, rather than attempt to reduce them, and this business model is to a large degree imposed on us by our major clients, who are very project-centric.

One of the more positive comments was:

> But I think it will be so competitive if we do it right. If you're competing with another company and you can do the job in … you've got that margin to compete with … and that's a good position. [OM1]

The major issues highlighted by responses to this question concerned a possible reduction in responsiveness to customer requirements, contracting model difficulties and the issue of intellectual property ownership.

5.3.2.4 SPL Funding

To determine participants' opinions of the possible SPL funding strategy that the company might follow, the question asked was:

O4: *What strategy would the company use to fund a product line effort?*

All participants were of the opinion that the funding of an SPL effort would somehow be problematic. Participants identified the company's reluctance and the lack of a mechanism to reinvest profits as the prime obstacles to funding.

> I think the major problem for doing this great thing is funding … to tell management or to ask management or to convince management that it will be a good investment to do this. [TM4]
>
> The current structure does not allow such funding, unless on a project… [TM1]
>
> But they seem to think that core business you make money out of and you sell your hours there and that's it. You don't invest in anything there. The customer pays for anything that you do there. [SD1]

The participants who made reference to the company's new ownership had mixed opinions on whether it would make funding more accessible.

> … but our new dispensation with [the new parent organisation]. It becomes a possibility [TM1]
>
> I don't think there's going to be funding coming from [the new parent organisation]. [TM3]

The general opinion was that the product line implementation would have to take place "on the back of" one or more projects.

> … funding will have to come from projects … ja from project profits. [TM3]
>
> …it will most probably have to ride on a project. [OM1]

Several participants thought that the customer would end up having to fund the product line implementation, which is unexpected if one considers that one of the objectives of a product line strategy is to become more competitive (i.e. to provide more economically attractive solutions to customers).

An interesting opinion expressed by a project manager was that the use of research and development funds for product line development would strengthen the company's evidence of its investment in intellectual property. This would in turn help to overcome the tendency of customers to expect to see a direct association between the level-of-effort and cost.

Perhaps the most unexpected response came from an organisational manager who suggested that the funding of the product line implementation would have to come from employees' overtime.

The widely varied responses to this question indicate that there is no obvious solution to the funding of a software product line effort and this issue will require further debate.

5.3.2.5 Summary of Findings: Organisational Themes

Table 5-2 presents a summary of the finding relating to the *organisational themes* category of interview questions.

Table 5-2: Summary of Findings: Organisational

#	INTERVIEW QUESTION	THEME	FINDINGS
O1	What do you see as the motivation for maintaining a project-centric structure within the company?	Motivation for maintaining a project-centric structure	This question uncovered general rejection of the project-centric structure. All operational managers rejected the idea of maintaining the project-centric structure, despite suspicions from the other groups that they might want to maintain it.
O2	What do you anticipate that the outcome of a business case analysis would be for a HUMS Ground Station software product line?	Making an SPL business case	Participants raised some issues that might threaten the chances of a successful SPL business case. These are mainly organisational issues relating to SPL adoption and the perceived short-term project-centric view taken by senior management.
O3	Considering the company's relationship with its current ground station customers, would you anticipate any problems with developing solutions for these customers using a software product line?	SPL customer interface management	The major issues highlighted by responses to this question concerned a possible reduction in responsiveness to customer requirements, contracting model difficulties, and the issue of intellectual property ownership.

#	INTERVIEW QUESTION	THEME	FINDINGS
O4	A software reuse effort requires initial funding and ongoing maintenance and support funding. What strategy would the company use to fund a product line effort?	SPL funding	The widely varied responses to this question indicate that there is no obvious solution to the funding of a software product line effort and this issue will require further debate.

5.3.3 Technical Themes

The *technical themes* category of primary themes is made up of the following individual themes:

QUESTION	PRIMARY THEME	SECTION
T1	SPL software development processes	5.3.3.1
T2	SPL configuration management and change control	5.3.3.2
T3	SPL problem domain knowledge	5.3.3.3
T4	SPL tools and training	5.3.3.4
T5	SPL software architecture	5.3.3.5
T6	SPL component technology	5.3.3.6

5.3.3.1 Software Development Processes

To determine whether participants anticipated problems relating to the adherence to SPL processes, the question asked was:

T1: *Software product line engineering requires the establishment of and the strict adherence to processes. Do you foresee that the company would have any problems in this regard?*

One participant, a system engineer, did foresee problems, but considered these problems to be widespread and not limited to our company.

> I think it's maybe a South African attitude towards engineering. We don't like processes. We don't like documentation. There are too many engineers South Africa-wide that think processes are just there to look nice and have

the quality stamp, but don't actually follow them and that's it … it's window-dressing. You need a major change in attitude … [TM3]

For all other participants, the initial response to this question expressed overwhelming confidence that adherence to processes would not pose problems for our company. The rationale being that, being in the defence industry, the company has years of experience with processes and standards.

I think we already have strict processes in place. [SD2]

We are fairly mature for our size because we have operated in the military environment. [OM1]

…generally the people they know what is expected of them and then they do it that way. [SD1]

…our processes are well defined and it gets followed. [TM2]

We adhere to processes and procedures to the full [in] every project we do. [TM4]

We are probably more disciplined than most companies. [TM1]

From the above evidence one could easily be convinced that processes would not pose a problem, however more issues were revealed as participants discussed the question further.

It's too easy [for us] to deviate from the strategy … more controls are required. [OM1]

[Our tendency towards window-dressing for processes] comes down to the amount of resources available … [TM2]

That's not the approach to take. If you want to do window-dressing, why bother? [TM4]

The more controversial opinions came from the software developers:

…all we do is adhere to the processes absolutely to the minimum that we have to in order to meet the deadline … [SD2]

The current procedures are already a schlep to follow. So additional procedures would just add another burden … [SD3]

> [We are] used to processes and procedures to the extreme … actually mind-bogglingly tedious stuff that they really should have tools [for]… it's dehumanising to get people … engineers, who are qualified to do engineering work, to have them sit and do documentation by hand. [SD1]

The last of these comments highlights another problem: the need for tool support and automation of processes.

Although the company's long history in the defence industry means that it is familiar and comfortable with formal processes, the consistent adherence to processes is an issue that possibly requires attention.

5.3.3.2 Configuration Management and Change Control

To determine whether participants anticipated problems concerning the SPL requirement for configuration management and change control, the question asked was:

T2: *Software product line engineering requires considerable discipline with regards to configuration management and change control. Do you foresee that the company would have any problems in this regard?*

As described in Section 2.6, SPL engineering is much more demanding than single application development when it comes to configuration management and change control.

All participants except one agreed that configuration management and change control are handled very well in the company. The single dissenting voice, a software developer, felt that configuration management and change control were "already a schlep" and product line engineering would only serve to aggravate this situation.

Some participants proposed changes to the current configuration management system:

> Maybe on the software side we need some more tools to do better change control and so on. [OM1]

> I don’t know about CVS [software version control system currently in use] versus something that’s more visual in terms of the source code control. [SD1]

Configuration management and change control have long been part of the company. The perception is that improvements might be required, but in spite of the additional demands of SPL engineering, this area is unlikely to prove a major challenge.

5.3.3.3 Problem Domain Knowledge

To determine participants’ rating of HUMS ground station problem domain knowledge, the question asked was:

T3: *Software product line engineering requires a comprehensive knowledge and understanding of the problem domain – in this case aircraft health and usage monitoring ground stations. How would you rate the current level of problem domain knowledge within the company with regard to aircraft HUMS ground stations?*

Six[19] of nine participants answering this question agreed that the level of domain knowledge was good.

> I think it’s world-leading. I think we need to thank [our customers] for that. [TM3]

> I think our knowledge is pretty good for what we do … [TM2]

> I think we have a very good knowledge of the problem domain. [TM4]

> I think it’s fairly good … if you compare it [with that of our competitors]. [TM1]

Interestingly, of the two operational managers who answered this question, one considered the level of domain knowledge to be weak and the other felt that software developers did not require this knowledge.

> I think it’s fairly weak. It’s maybe better now because we have a few products in the field. [OM1]

[19] One organisational manager, three technical managers and two software developers

> Okay, but that is nothing to do with the software guys whatever… But it's not the software guy or the hardware guy … he's not interested. That is not his interest. [OM3]

A concern expressed by four[20] participants was that, although the domain knowledge is good, to a large degree it exists in the heads of a few individuals, making it vulnerable.

> …it's in a few peoples' heads and if you lose those few people you've lost the capability. [TM3]

> We have an excellent knowledge of the problem domain in some employees. [SD2]

> At the moment the intellectual property of the company lies in people's heads … and that's a big problem. [TM4]

Another issue highlighted in two[21] responses was that as a company we are not proactive in seeking out domain knowledge. A technical manager suggested that the domain knowledge that we have is coincidental and attributable only to our interaction with our customers.

Although valuable HUMS domain knowledge has been accumulated, this knowledge is limited to that which has been necessary to do the job and it is vulnerable because it resides predominantly with certain individuals.

5.3.3.4 Tools and Training

To determine participants' expectations of the company's willingness to provide the tools and training essential for an SPL effort, the question asked was:

T4: *In software product line engineering, tools play an important part in the automating of development processes, ensuring compliance with standards and the eradicating of rote labour-intensive tasks. Do you think that the company would have any problems investing in software development tools, providing adequate training, and enforcing the standardisation and consistent use of these tools?*

[20] Three technical managers and one software developer

[21] One technical manager and one software developer.

All participants agreed that there has been insufficient investment in tools and training.

> Tools are so important … it's underestimated [in the company] what you can do with tools … if you have the right tools. [SD2]
>
> … as a project manager you've got to be quite clever, you've got to time your request to when the cash flow is not a problem. If there's a cash flow problem, you'll get turned down. [TM1]
>
> … management don't seem to regard software engineers' time as money … if they spend money on tools, the see that as spending money, but individuals spending hours and hours of mindless testing [that could be automated] they don't see that as spending money. [SD2]
>
> For sure in the past we have … sort of not invested sufficiently in tools and all of that. We have skipped on that. We have tried to maximise our cash flow. [OM1]
>
> They won't be prepared to invest in software development tools or training. It's just not done … unless the customer pays for it, they just don't do it … [SD1]
>
> … it always boils down to "why can't you do the job with the tools we have?" [SD3]

There is, however, confidence that the situation might improve owing either to the change of ownership or to the introduction of the matrix structure.

> I think to answer the question: in the past we had issues and the issues were really constrained by available funds … capital. That issue should be away now. [OM1]
>
> I think in the structure before we changed it would have been an issue, but part of the reason why we changed the structure is to get advantages of this. [TM2]

A few participants expressed the opinion that a certain critical mass of work in a particular area is required before the investment in tools to automate the related processes can be justified.

> I think it all comes to critical mass. If you've got enough work to justify having fairly expensive tools and then to automate some of the processes … [OM1]

> It's the critical mass thing …if you've got some matrix structure, you can now say "… but look we can use this [tool] on this, this and this project". [OM2]

An interesting secondary theme that emerged was the negative effect that the project-centric structure has on tools and training. Projects that are already experiencing time and budget pressures cannot be expected to invest in new tools and training. Inevitably, this will only aggravate the situation.

> … the project manager doesn't want to send off all the software [team] members for a course on new tools … there's no time. There's never time for training. [TM4]

> … I mean my goal [as a project manager] is just to win the race and training and that [must wait]. [TM2]

In summary, there is acknowledgment across all groups that tools and training are important and have been neglected in the past. Consequently, the fact that an SPL approach depends on the consistent use of tools by trained developers should not pose an insurmountable problem.

5.3.3.5 Software Architecture

To gauge participants' opinion of the ability of the company's software developers to produce a software architecture to support an SPL, the question asked was:

T5: *The success of a software product line is dependent on having a suitable and robust software architecture. Would this pose a challenge for the company's software developers?*

Only four of the nine participants actually ventured an opinion that could be classified as being either positive or negative. One[22] was negative and three[23] were positive.

[22] A software developer

[23] An organisational developer, a technical manager and a software developer

> Yes, I think definitely because [our organisation] is still in the dark ages regarding software architecture and tools and frankly I think they will never catch on. [SD3]
>
> Certainly, I think we've got the right skills and experience in our make-up here to be able to devise such an architecture. [OM1]
>
> I think there are very talented software developers here and I think we will definitely be able to rise up to the challenge. [SD2]
>
> Ja, for sure. I think the senior guys will enjoy doing this. [TM4]

A variety of concerns were expressed. Three[24] participants mentioned the general reluctance of software developers to do properly planning.

> … they don't go through a decent phase of creating what should be done … taking a global view of your system before you start designing architecture. [TM3]
>
> … we are so much in the habit of racing for deadlines … for meeting deadlines and fighting fires so in that sense it would be a bit of a challenge because we'd have to change our mindsets. [SD2]
>
> I think the okes just don't want to do the planning. They want to get into the coding and start the [exciting stuff] ... [TM2]

One software developer, an electronic engineer, had the opinion that the company's lack of software architecture knowledge was due to a tendency of the company to employ engineers and technicians rather than computer science graduates as software developers. An organisational manager foresaw possible difficulty in getting the cooperation of developers to accept a particular architecture.

Considering the participants' responses, one can deduce that the SPL architectural design should be allocated to suitably experienced and trained persons and sufficient time should be budgeted.

[24] Two technical managers and a software developer

5.3.3.6 Component Technology

To determine participants' opinions relating to the selection of a suitable software component technology for an SPL effort, the question asked was:

T6: *Before investing in the development or purchase of reusable software components, it is necessary to choose a suitable component technology that has a reasonably long-term future. What are your opinions in this regard?*

Owing to the obvious technical nature of this question only six participants[25] felt prepared to state an opinion. Of these, four[26] recognised potential challenges of choosing a component technology:

> … there are standards already emerging and you have to now select the right one …[OM1]

> Obviously you need long-term solutions in this industry… You have to find things that will be there for the long term. [TM2]

> You've got to make some key decisions here and they could be tricky and they could be wrong ones and you've got to take that into account. [TM1]

One software developer related a number of problems he had experienced specifically with buying COTS (commercial off-the-shelf) software components and attributed these to the immaturity of the software development profession especially in comparison with electronic development.

> … I think, in terms of the software component industry, you've got a problem with the size of the companies. They give you something and the next week they are not there anymore. The support is bad, the documentation is bad and it just doesn't work. [SD1]

> … it's changing so rapidly that … its so difficult to find something that is stable. [SD1]

> [Unlike electronic components] you don't get something like a data sheet for a piece of software. [SD1]

[25] One organisational manager, two technical managers and three software developers

[26] One organisational manager, two technical managers and one software developer

The participants recognised the importance of not only making a good choice of component technology, but also of understanding some of the practical difficulties involved.

5.3.3.7 Summary of Findings: Technical Themes

Table 5-3 presents a summary of the finding relating to the *technical themes* category of interview questions.

Table 5-3: Summary of Findings: Technical

#	INTERVIEW QUESTION	THEME	FINDINGS
T1	Software product line engineering requires the establishment of and strict adherence to processes. Do you foresee that the company would have any problems in this regard?	SPL software development processes	Although the company's long history in the defence industry means that it is familiar and comfortable with formal processes, consistent adherence to processes is an issue that possibly requires attention.
T2	Software product line engineering requires considerable discipline with regard to configuration management and change control. Do you foresee that the company would have any problems in this regard?	SPL configuration management and change control	Configuration management and change control have long been part of the company. The perception is that improvements might be required, but in spite of the additional demands of SPL engineering, this area is unlikely to prove a major challenge.
T3	Software product line engineering requires a comprehensive knowledge and understanding of the problem domain – in this case aircraft health and usage monitoring ground stations. How would you rate the current level of problem domain knowledge within the company with regard to aircraft HUMS ground stations?	SPL problem domain knowledge	Although valuable HUMS domain knowledge has been accumulated, this knowledge is limited to that which has been necessary to do the job and it is vulnerable because it resides predominantly with certain individuals.

#	INTERVIEW QUESTION	THEME	FINDINGS
T4	Do you think that the company would have any problems investing in software development tools, providing adequate training and enforcing the standardisation and consistent use of these tools?	SPL tools and training	There is acknowledgment across all groups that tools and training are important and have been neglected in the past. Consequently, the fact that an SPL approach depends on the consistent use of tools by trained developers should not pose an insurmountable problem.
T5	The success of a software product line is dependent on having a suitable and robust software architecture. Would this pose a challenge for the company's software developers?	SPL software architecture	Considering the participants' responses, one can deduce that the SPL architectural design should be allocated to suitably experienced and trained persons and sufficient time should be budgeted.
T6	Before investing in the development or purchase of reusable software components, it is necessary to choose a suitable component technology that has a reasonably long-term future. What are your opinions in this regard?	SPL component technology	The participants recognised the importance of not only making a good choice of component technology choice, but also of understanding some of the practical difficulties involved.

5.4 PRESENTATION OF FINDINGS: EMERGENT THEMES

During content analysis a number of secondary themes emerged. Even though some of these themes are located on the periphery of the central focus of the research topic, they were considered to contribute to the richness of the study and the characterisation of its context. These themes are grouped as follows:

1. themes that relate specifically to a project-centric development environment;
2. themes that relate specifically to an SPL development environment; and
3. themes that allow comparison between a project-centric and an SPL software development environment.

Table 5-4 provides a summary of the emergent themes according to their groupings.

Table 5-4: Emergent Themes

THEME	SUB-CATEGORY	SECTION
CATEGORY: PROJECT-CENTRIC DEVELOPMENT THEMES		
Contextual issues	Context	5.4.1.1
Developer attitudes	Software engineering	5.4.1.2
CATEGORY: SOFTWARE PRODUCT LINE DEVELOPMENT THEMES		
Vision	Organisational	5.4.2.1
Implementation strategy	Organisational	5.4.2.2
Management support and leadership	Organisational	5.4.2.3
Organisational structure	Organisational	5.4.2.4
Organisation size	Organisational	5.4.2.5
Mindset	General	5.4.2.6
Advantages and disadvantages	General	5.4.2.7
CATEGORY: COMPARATIVE ISSUES		
Intellectual property	Organisational	5.4.3.1
Human resources and career development	Context	5.4.3.2
Software development frustrations	Software engineering	5.4.3.3
Support and maintenance	Software engineering	5.4.3.4

5.4.1 Project-Centric Development Environment

This section discusses the following emergent themes relating to the project-centric development environment:

- contextual Issues; and
- developer attitudes.

5.4.1.1 Contextual Issues

The issues relating to context were included in Chapter 4 to enrich the contextualisation of the study.

5.4.1.2 Developer Attitudes

Based on a survey, Lynex and Layzell (1998) compiled an extensive list of issues that act as barriers or disincentives to software reuse among software developers.

According to Lynex and Layzell (1998) these issues limit the success of a software reuse programme and can even result in its failure. Table 5-5 itemises a selection of items from this list that were confirmed by responses from interview participants relating to examples of negative attitudes among some of our software developers.

Table 5-5: Disincentives among Development Staff (Lynex & Layzell, 1998)

DISINCENTIVE	**DESCRIPTION**
Not-invented-here	This label is given to the situation in which developers do not trust reuse assets unless they produced them.
Fear of measurement	Some developers feel insecure or threatened by the increased level of assessment and exposure that reusable assets are subjected to in a reuse programme.
Job protection	Some developers regard the sharing of their knowledge in the form of reusable assets as a threat to their job security.
Resistance to change and Fear of the unknown	Some developers resist the change necessary to create a suitable reuse environment because processes in which they have built confidence are replaced by new and unfamiliar processes.
Builders not integrators	Many developers find it more satisfying to develop software from scratch than to integrate components developed by others.
Unprofessional developers	Some developers violate coding standards, produce obscure code, or fail to comment and document their code sufficiently. These developers may live in fear of exposure, while others may become reluctant to reuse assets produced by them.

Table 5-6 presents some examples of comments made by participants belonging technical management based on their experiences with software developers.

Table 5-6: Examples of Disincentives Recognised by Participants

COMMENT	**DISINCENTIVE**
[They say] “It’s much easier to start from scratch … more exciting as well”. [TM1]	Builders not integrators
[People are afraid of people] criticizing [their software] or claiming ownership. If you have done a lot on your software, it’s not nice if I take it, add a line above and add a line below and … “nice software Koos this is really running well” and actually I’ve taken your software and added two lines. [TM4]	Fear of measurement Job protection

COMMENT	DISINCENTIVE
As soon as you put your software in a pool it will be used on various projects so you won't be the HUMS king anymore, because Piet and Koos will use your HUMS idea. [TM4]	Job protection
[They say] "I didn't develop them, so they are badly documented". [TM1]	Not-invented-here
[People's egos get in the way] … they want to put their stamp on it. That's probably my biggest concern to set it up properly with that aim. [TM2]	Not-invented-here
[They say] "Why do you want to use tools or modules from a previous project if they're badly documented or often in your opinion badly written?" [TM1]	Not-invented-here
[Some developers say] "I don't like the way that this is set up … you know … I don't want it like that. I want it just 3 degrees to the left ". [TM2]	Not-invented-here
… we have a couple of prima donnas that want to do things their way [TM2]	Resistance to change and fear of the unknown
… what happens is the technical guys, on the project, they will determine where the project goes. They have got free reign. … [They say] "This is the way I want to design the system…" [TM1]	Unprofessional developers
Some developers … don't want to do documentation. … There are some guys that do really good coding but they can't document … they hate documentation. [TM4]	Unprofessional developers

In summary, participants from the technical management group confirmed the existence of a number of typical disincentives to systematic software reuse. Should the company attempt an SPL venture, it would be sensible to address these disincentives as well as the others in Lynex and Layzell's list.

5.4.1.3 Summary of Findings: Project-Centric Development Environment

Table 5-7 presents a summary of the finding relating to the "Project-Centric Development" category of emergent themes.

Table 5-7: Summary of Findings: Project-Centric Development Environment

PROJECT-CENTRIC DEVELOPMENT ENVIRONMENT THEMES		
THEME	**SUB-CATEGORY**	**FINDINGS**
Contextual issues	Context	A variety of context-relevant information derived from discussions with participants was used to complement the contextualisation of the study.
Developer attitudes	Technical	In summary, participants from the technical management group confirmed the existence a number of typical disincentives to systematic software reuse. Should the company attempt an SPL venture, it would be sensible to address these disincentives as well as the others in Lynex and Layzell's list.

5.4.2 SPL Development Environment

This section discusses the following emergent themes relating to the SPL development environment:

- product line vision;
- implementation strategy;
- management support;
- leadership;
- organisational structure;
- organisation size;
- mindset; and
- advantages and disadvantages.

5.4.2.1 Product Line Vision

The *product line vision* (*see* Section 2.6) is a factor which distinguishes SPL engineering from many other reuse strategies (Knauber et al., 2000). The product line vision is a multifaceted concept because it needs to serve various purposes. It must, for example, provide guidance for technical staff, management and sales staff and if must address the present as well as the short- and long-term future directions.

Knauber et al. (2000) observed that small and medium-sized companies adopting an SPL approach often initially lack a coherent vision for the products that they develop;

however, once they form such a vision it becomes invaluable for their strategic planning.

The product line vision is an essential part of an SPL initiative. The significance of this fact was sensed by certain of the participants.

> We need to go and sit and see what the HUMS of the future is going to look like … and develop that. But to be able to do that we've got to know the market backwards and I am not sure that we [do]. [TM2]

> The company's decided that it wants to be in HUMS but that's as far as it goes. From a product strategy point of view there's not too much in place … that we want to develop generic products. What we basically do is look for projects in the HUMS field no matter what they are. And I think there's quite a big difference if you look at it from that point of view. [TM1]

> I think the biggest technical issue is that of technical vision. You need people to have technical vision of where we should be going. For that to happen … for people to be allowed to have technical vision … you need management with … vision in management … they have to be able to look into the future because what the management decide eventually filters down to what people think. [SD2]

> Currently there's no proactive movement from [the company's] side in terms of looking at the bigger picture technologically-wise. It's a per project thing … what we need, we quickly learn that … and go with that. [SD3]

5.4.2.2 Implementation Strategy

There are various options for SPL implementation (*see* Section 2.5) but, because we are dealing with a fully functional operating company, it is no surprise that most participants who discussed this considered that an evolutionary strategy would be most suitable.

> Ek glo dit [SPL] is die manier hoe jy gaan dit doen. Dis die strategie hoe jy daar gaan kom. We must grow it over a few projects. [OM3]

> I think the "full Monty" that you explain here could be quite drastic from where we are here. In between there is just getting used to reusing and then going to the tools and methods and all of that and at the end having a total factory. So I think we should start with component libraries and still have a bit more

> flexibility in terms of the application framework side. Because if we do the whole strategy now from where we are today I think that would be quite drastic. [OM1]

> I think it's going to be more difficult because we are already in the process … I mean we've already got products … we've already got a lot of software. I think it's going to be more difficult to do it. But the fact that its going to be more difficult does not mean we should not do it. I think it will require a lot more effort and detailed planning and management to get it to work that we can see the big picture at the end. But I think its going to be a bit more difficult than if we were starting from scratch … on a clean sheet. [TM2]

> There are different ways of getting to this product line model. The one is doing it right from the start … sort of from the birth of the company, you have a strategy which says this is where we want to go. [TM1]

5.4.2.3 Management Support and Leadership

An essential ingredient for the success of an SPL initiative is high-level management support (*see* Management Commitment in Section 2.6) in the form of adequate staff, budget and time (Bühne et al., 2004). As Knauber and Succi (2002, p. 44) express it: "… the one thing we can be sure about and that we have learnt in the last 15 years of research on product lines and reuse, is that management support is essential".

The importance of management support is further confirmed in studies conducted by Clements and Northrop (2001) that identified "strong and unswerving management commitment " as a common factor present in successful product line organisations.

Several of the participants, especially those in the technical management group, were sensitive to this fact, as echoed by the following statements:

> They [management] must buy into this heart and soul. They must buy into this and commit to this. [TM4]

> … you have to get the management support, you have to get the structures right, you have to get the champion in place so that everybody is focused, you have to get the market to understand … everybody has to be aligned. [TM2]

> … it’s going to cost money to implement and it will take you three years or five years to sort this thing out. It’s not going to be six months and it will cost money and is going to be blood and guts. I think management will have to realise that. [SD1]
>
> I think if [managing director] should focus … if he thinks product lines is a good idea he should definitely have a structure that supports that otherwise its not going to work. I personally think it’s a difficult thing to manage. You need all the support from organisation, from people like configuration management. Every role player must support it; otherwise it’s not going to work. The message I have is that everything must be in place otherwise it’s not going to work. If you have one link missing the whole thing falls apart. [TM3]
>
> … you also mentioned management support and so on... I think that’s got a good chance of succeeding. [OM1]

Participants felt that another essential ingredient for SPL success is strong leadership:

> Jy moet ’n sterk ou daar hê anderster gaan dit plat val en jy gaan nooit die benefits daarvan kry nie. [OM3]
>
> You either need a software product line software manager or you need a chief engineer that’s in charge of …, but somebody’s got to drive it. Somebody has to be in charge of it or it won’t happen. [SD2]

These sentiments are in accordance with the literature, which highlights the need for leadership at product line and organisational levels. At SPL level, leadership should come from a product line champion, appointed and empowered by senior management. According to Northrop (2002b, p. 35), “This champion must be a strong, visionary leader who can keep the organisation squarely pointed toward the product line goals, especially when the going gets rough in the early stages”.

At organisational level, senior management need to ensure that leadership forms part of the support they provide. Clements (2002, p. 30), asserts, “Putting an organization on the same strategic page requires vision, strong management, technical competence, process discipline, and no small amount of dedicated leadership”.

Leadership should not be limited to the SPL adoption phase, it has to be a continuous, ongoing SPL support effort (Clements & Northrop, 2001, p. 45).

5.4.2.4 Organisational Structure

Early literature on SPL engineering encouraged a two-tired structure (*see* Section 2.4.2.3), one tier for developing *for* reuse and the other for developing *with* reuse.

> Three ideas are common to most successful product line efforts: exploring commonality among products to proactively reuse software artifacts, encouraging architecture-centric development, and having a two-tiered organizational structure. (McGregor et al., 2002)

As indicated in the above quote, there is common belief that for SPL success a two-tier organisational structure is necessary; however, Northrop (2002b) argues that this early principal has now been disproved by experience. Nevertheless, the organisational structure chosen must support the SPL effort.

One of the technical managers considered that a total restructuring of the company would be necessary.

> You need a totally different organisational structure … core to the whole thing is to reorganise the company. [TM1]

However, other technical managers seemed convinced that the company's newly adopted matrix structure was adequate.

> You need a matrix structure otherwise it's not going to work. [TM3]

> I think in the [matrix] structure before we changed it would have been an issue, but part of the reason why we changed the structure is to get advantages of this. So you would have a bigger resource pool and you can actually plan training and development and then I mean there are certain people that are responsible for the tools. I think the way we are structured now will improve. Obviously investment into tools in terms of money and that … money needs to be available. [TM2]

These statements suggest an underestimation of the importance of a suitable organisational structure as well as a misconception that a matrix structure alone might be sufficient to achieve systematic reuse.

5.4.2.5 Organisation Size

Certain participants expressed apprehension about the required organisation size to ensure the successful implementation of SPL engineering.

> I think it’s a comprehensive approach. I just don’t know how big a company must be to be able to do this properly... it looks to me like it’s a major exercise and I think you need a fairly large organisation to implement that. You need to have resources at your disposal. I don’t think a small company can do that. It might not scale down easily. I think a small company can do parts of it. [SD1]
>
> Maybe it’s a question of critical mass. It might become more do-able if we were doing ten times as much ground station work. … So with the critical mass somewhere you’re going to get benefit. [OM2]

The actual concern of these participants was that the small size of our company might make it less suitable for an SPL initiative.

5.4.2.6 Mindset

Numerous participants agreed that a product line initiative would require a significant change of the mindset within our company.

> [We have to] get the people to pull in the same direction … otherwise it’s that mindset change that’s [got to happen]. [TM2]
>
> … the rattling of the cages and that is going to have a short-term [negative] effect … it will take certain things longer to happen because people are not fully aligned with that. But I think in the longer term you’ll get the benefit. [TM2]
>
> … definitely. I think it’s maybe a South African attitude towards engineering. We don’t like processes. We don’t like documentation. There are too many engineers in South Africa wide that thinks processes are just there to look nice and have the quality stamp but don’t actually follow them and that’s it. It’s window-dressing. You need a major change in attitude, even from the software engineers to change that … that’s for sure. [TM3]
>
> … we are … so much in the habit of racing for deadlines … for meeting deadlines and fighting fires so in that sense it would be a bit of a challenge because we’d have to change our mindsets. [SD2]

> [It's a change of culture] and it's a good time now to do this culture change because we need to adapt to a new system and even if we can do this and the convince them [management], it's a good approach. You might just see that they adapt to our system. [TM4]

> I think the other thing is that the management style of product line-based development versus project-based [development] is completely different. I think with projects you've got control and [the company] knows how to do that; they've got a budget and they've got schedules. With a product line it becomes different. I think you'll have to do a mind-switch and go through a learning curve on how to manage that. I think it will be a learning curve and in other words it's going to cost money to implement and it will take you three years or five years to sort this thing out. It's not going to be six months and it will cost money and is going to be blood and guts. I think management will have to realise that. [SD1]

It is clear from these comments the participants recognise that there is a significant variance between the company's current project-centric mindset and the mindset necessary for SPL engineering.

5.4.2.7 General Advantages and Disadvantages

During the interviews there was limited discussion directed specifically at the perceived advantages and disadvantages of the SPL approach. Advantages specifically mentioned concerned improved competitiveness and improved quality, which are also professed benefits of systematic software reuse.

> But I think it will be so competitive if we do it [SPL engineering] right. If you're competing with another company and you can do the job in … you've got that margin to compete with … and that's a good position. [OM1]

> I think there's [sic] advantages in terms of the quality … [SD1]

A disadvantage mentioned was the potential loss of flexibility. The following quote from Knauber et al. (2000, pp. 92-93) explains the importance of flexibility to smaller companies:

> Typical for SMEs is their close cooperation with customers. This offers them a marketing advantage over larger competitors because they not only know

> early about their customers’ actual and potential needs but they are also able to react more flexibly to these needs.

Our company is probably no different in this regard, and one of the operational managers echoed this sentiment.

> On the other hand, you have to quantify the potential loss of business because you are so rigid. You might have an opportunity that you would otherwise have taken but now you’ve got your own strategy which says I’m not going to look at those special cases. [OM1]

Achieving a balance between flexibility (i.e. variability) and complexity is certainly one of the challenges of SPL engineering. Insufficient variability limits the scope of applicability of the product line and excessive variability make product instantiation complex and tedious (Clements & Northrop, 2001, pp. 115-116). As Bosch (2000, p. 21) puts it: “The core element in successful software product lines is the software architecture that should maximise the benefits of the commonalities between the systems in the family, while providing sufficient variability for each family member”.

One of the organisational managers displayed insight into the developer’s psyche by observing that developers might find the product line approach monotonous and lacking in challenge.

> I think a negative aspect might [be] long term because due to the fact that you are getting so efficient on the one side and also focused on the other side with this structure is that if you don’t take your people from one aspect to another. … Dit kan baie eentonig raak en daar’s nie baie uitdagings vir n ontwikkelaar nie. Hy gaan skep maar elke keer met hierdie ou komponentjie en daardie ou komponentjie. So daar’s nie meer vermoë om te skep en dit mag ’n negatiewe aspek wees. [OM3]

On the contrary, because of the limited level of reuse in a project-centric environment, developers often develop similar functionality many times over and are also likely to be frustrated by the lack of training and tool support. However, in a product line environment, the common elements of products are developed once and reused, so developers spend more time on the unique features (Clements & Northrop, 2001, p. 21).

In summary, the prime issue of concern identified in the discussion of advantages and disadvantage is the potential loss of flexibility in satisfying customer requirements.

5.4.2.8 Summary of Findings: SPL Development Environment

Table 5-8 presents a summary of the finding relating to the *SPL development environment* category of emergent themes.

Table 5-8: Summary of Findings: SPL Development Environment Themes

SOFTWARE PRODUCT LINE DEVELOPMENT ENVIRONMENT THEMES		
THEME	**SUB-CATEGORY**	**FINDINGS**
Vision	Organisational	The product line vision is an essential part of an SPL initiative. The significance of this fact was sensed by certain of the participants.
Implementation strategy	Organisational	There are various options for SPL implementation but, because we are dealing with a fully functional company, it is no surprise that most participants who discussed this considered that an evolutionary strategy would be most suitable.
Management support and leadership	Organisational	Several of the participants were sensitive to the fact that management support and leadership are essential ingredients for SPL success.
Organisation structure	Organisational	A suitable organisational structure is required to support SPL development. A section of management appears to underestimate this requirement.
Organisation size	Organisational	Participants were concerned that the small size of our company might make it less suitable for an SPL initiative.
Mindset	General	It is clear from their comments that the participants recognise that there is a significant variance between the company's current project-centric mindset and the mindset necessary for SPL engineering.
Advantages and disadvantages	General	The prime issue of concern identified in the discussion of advantages and disadvantages is the potential loss of flexibility in satisfying customer requirements.

5.4.3 Comparative Themes

This section discusses the following emergent themes that allow comparison between the project-centric and SPL development environments:

- intellectual property;
- human resources and career development;
- software development frustrations; and

- support and maintenance.

5.4.3.1 Intellectual Property

In a project-centric company, software is developed according to customers' requirements, making it difficult to isolate the customers' intellectual property from its own. A possible consequence of this is that intellectual property which evolves is not specifically captured or documented, and therefore only exists in the memory of the individuals who are directly exposed to it. This is obviously an undesirable situation, because the company has no formal mechanism for accumulating intellectual property and, in effect it operates as a labour broker and sells man-hours.

In discussions with the participants, some evidence of this phenomenon was apparent.

> Too much emphasis, that's a syndrome of a small company, is placed on the individual and he's just expected to go ahead and deliver. People assume that, because he is an engineer or comes from a computer science background, he knows what he is doing and I think that is sometimes a bit of a problem. [TM1]
>
> I think it's world-leading. I think we need to thank [our main customer] for that. The problem is that it's in a few peoples' heads and if you lose those few people you've lost the capability. I don't know how you document that. [TM3]
>
> At the moment the intellectual property of the company lies in people's heads … and that's a big problem. If any of the senior guys leaves … what do [the company] do? Do they send them to a doctor to download their information? It doesn't work that way. Unless we put the knowledge in a repository, it's going to get lost as the people move on. [TM4]
>
> We have an excellent knowledge of the problem domain in some employees. [SD2]

In an organisation implementing an SPL strategy, a business factor that requires attention concerns ownership of intellectual property rights. According to Bass, Clements, Cohen, Northrop & Withey (1997), "… intellectual property rights must remain with the developer rather than go out with each product, so that new products based on old ones can continue to be sold …"

This issue was also recognised by some participants as a problem faced by the company.

> … we already have a problem with [one] customer … on IP because he paid for the customisation part but what we said to him that we're not going to give him the IP to the whole product because there is a lot of background knowledge that went into this product . So maybe it will be better if you follow that strategy because you can actually prove that you've got a product line and you only customise a little bit. So you can say to the customer "look you only pay for this customisation bit. The rest is clearly ours". [OM1]
>
> But the problem is that the way we structure our contracts is that most of the intellectual property sits with the customer because we sell our intellectual property and that's where the R&D funds need to be utilised so that we [have evidence of our investment]. [TM2]

The mode of operation of a project-centric company makes it fairly simple for the customer to determine roughly what the man-hour level-of-effort is and thereby estimate the hourly rate being charged for development. This factor, coupled with the lack of clarity on IP ownership, makes it difficult for a project-centric company to charge for more than the obvious level of effort. This issue was acknowledged by one of the technical managers.

> We're a resource broker. [Prompt: Is that what we want to be?] No … not at all! What do we do? We get a spec, we execute the spec and we deliver the product. So we are just supplying resources to a company to execute what they want. We're not adding value to this process by adding our IP. If we add our IP, the price is different. [TM2]
>
> That's why we have to have intellectual property so that that thirty hours that we sell comes at a premium … it's not just paying the rate … it's not just the seventy dollars or whatever or eighty dollars an hour, it's like a hundred and twenty dollars an hour. [TM2]

There is little benefit in an organisation generating or evolving intellectual property if there is no formal mechanism for capturing in some form of knowledge base. The SPL assets provide a highly convenient mechanism for retaining this data in the form of components and processes, which are evolved as lessons are learnt. The focused scope of an SPL and the well-defined set of assets under configuration control facilitate the capture of IP. The Fraunhofer Institute, one of the research institutes that are well known for its contribution to product line theory and practice, aptly call this SPL benefit "explicit knowledge representation". The Institute (Fraunhofer Institute, 2003) describes it like this, "Explicit knowledge representation: Application domain knowledge is consolidated and made explicit. It can be used for development, as well as for training new employees. In addition, the information is not lost when experts leave the organization."

One of the technical management participants expressed frustration at the lack of a mechanism to capture project experience, which is a valuable contributor to IP.

> Since I'm in [the company] we had once a "lessons learnt" [sic]. That was [project manager's name]. He organised this meeting he said "lessons learnt on [the project]" and we all went down there and listed all the things … lessons learnt. So we documented that … but [currently] it's not implemented, it's not fed back, it's not approved. There's no structure to feed it into. [TM4]

Whereas a project-centric structure makes it is difficult to accumulate and benefit from IP, an SPL structure supports the accumulation of IP and facilitates benefiting from it.

5.4.3.2 Human Resources and Career Development

In a project-centric environment, projects tend to hold on to developers out of fear of losing them to other projects, so individuals become attached to projects and are generally obliged to take on various roles as the project lifecycle progresses. Consequently, developers are sometimes reduced to performing fairly mundane tasks, effectively depriving the company of their specialist skills. It follows that a project-centric environment is better suited to generalists, who are good at performing a variety of tasks, rather than specialists.

This phenomenon not only means that products achieve suboptimal functionality and quality, but it can also negatively impact developers' career paths. The situation is further aggravated by the lack of investing in tools and training typical in a project-centric environment.

The following comments confirm that some participants are aware of these issues:

> [In ‘n “project-centric” struktuur] jy het baie goed verloor ... jy kyk nie na mannekrag nie, jy kyk nie in die sin van enige opleiding vir die mannekrag nie. Jy voel vere vir die mannekrag en daai goed. Daar’s baie goed wat op die grond val ten opsigte van jou mens bestuur. In daardie approach [“project-centric”] is dit ‘n kwessie van jy lewer en klaar. [OM3]

> I notice that every year when I came back after the December holidays I think this is what I want to do this year. I’ve got all these things in my mind that I want to do and then what happens after a month and a half or so is you start to regress to your normal habit of just trying to meet the deadline and going home and coming back to work and the three months later you think “what was I [thinking]?” [SD2]

> We are paying engineers to format documents and using up critical resources when we could be pay someone R—— an hour and they would do a better job. [TM2]

> [Currently] we’re wasting resources. Software people are hard to come by and they’re scarce … and if you waste a resource like that it’s not in the company’s interest. That’s one of the big disadvantages [of a project-centric structure]. [TM3]

> I don’t think so. I think [we are] used to processes and procedures to the extreme … actually mind-bogglingly tedious stuff that they really should have tools [for]. You cannot … it’s dehumanising to get people … engineers, who are qualified to do engineering work, to have them sit and do documentation by hand. To type documents … stupid character by character. [SD1]

By comparison, an SPL environment provides certain advantages for career development. Peterson (2003, p. 383) identifies mobility as a career benefit provided by an SPL environment for the individual: [The software product line environment provides] “employees with more career development opportunities by standardizing on the development environment and processes, thereby reducing the learning curve associated with a move to a new project”.

Clements and Northrop (2001, p. 21) interviewed developers and concluded that those working on SPLs displayed “higher morale and greater job satisfaction” than those developing stand-alone products. Career benefits of an SPL environment also include adequate tools and training, time to explore new technology, specialisation opportunities, the satisfaction of producing high quality products, and a reduction of the stress caused by tough schedules.

To summarise, the participants identified problems relating to career development and wastage of human resources attributable to a project-centric structure. Potential SPL solutions to these problems were sought in the literature and presented for comparison.

5.4.3.3 Software Development Frustrations

Certain frustrations concerning software development were prominent and recurring during the discussions relating to several of the interview questions. Interestingly, the frustrations that were identified all concern issues that are typically addressed by a systematic software reuse programme and certainly are addressed by SPL engineering.

A selection of these frustrations was noted, categorised and is presented in Table 5-9 together with comments on how these frustrations are addressed by SPL engineering.

Table 5-9: Software Development Frustrations

PROJECT-CENTRIC ENVIRONMENT FRUSTRATION	**SOFTWARE PRODUCT LINE SOLUTION**
ISSUE: TIME AND BUDGET PRESSURES	
All the projects I've been working on are late. [TM3] I think the time of giving a date and just missing it by 2 … 3 weeks is long gone … especially in the aerospace industry. [TM2] Most of our projects are in high risk and especially in software development we are never in time, or on budget and in time. [TM4]	In an SPL environment, a large percentage of a new product is based on existing assets (e.g. typically around 75% (Pronk, 2002), 70 – 90% (Clements & Northrop, 2001, p. 26)), which results into a significant decrease in overall cost and schedule (Northrop, 2002b).

PROJECT-CENTRIC ENVIRONMENT FRUSTRATION	SOFTWARE PRODUCT LINE SOLUTION
ISSUE: LACK OF THE CONSISTENT USE OF PROCESSES	
… every programme is managed in a different way. System engineering … the standard is different on every programme. [TM3] … the technical guys, on the project, they will determine where the project goes. They have got free reign. [TM1]	SPL engineering involves the definition, implementation and measurement of processes (Northrop, 2002b). "Defined processes set the bounds for each person's roles and responsibilities so that the collaboration is a successful and efficient one" (Clements & Northrop, 2001, p. 175)
ISSUE: TOO MUCH EMPHASIS ON SPECIFIC INDIVIDUALS	
Too much emphasis, that's a syndrome of a small company, is placed on the individual and he's just expected to go ahead and deliver. [TM1] …it's in a few peoples' heads and if you lose those few people you've lost the capability. [TM3] We have an excellent knowledge of the problem domain in some employees. [SD2] At the moment the intellectual property of the company lies in people's heads … and that's a big problem. [TM4]	The detailed documentation of processes, the management of core assets, tool support and training all serve to make SPL engineering less dependent on individual developers.
ISSUE: LACK OF LONG-TERM TECHNICAL VISION	
if you think short-term then there might not be any need because you want to make as much money as you want in the shortest amount of time [SD2] I think the biggest technical issue is that of technical vision. You need people to have technical vision of where we should be going.[SD2] People are not built short term … they are not built to say "oh I've just got to finish this and then it's done". They want to contribute. They want to look back and say "this is what I have done." [SD2]	SPL engineering is based on an overall corporate vision of the product lines to be established and the scope of each product line. "A major distinction between software product lines and other reuse approaches is that the product line is based on a clear vision of which future products will be developed" (Knauber et al., 2000, p. 91).

PROJECT-CENTRIC ENVIRONMENT FRUSTRATION	SOFTWARE PRODUCT LINE SOLUTION
ISSUE: LACK OF TOOLS AND TRAINING	
Because we are project-based, the project managers don't want to invest in the creation of tools that will lead to long-term benefits. [TM4] I think they won't have a problem enforcing standardisation and consistent use of the tools … if it [the tools] were there and that's not a problem. But to actually invest … that's always a problem … [SD1] … it always boils down to "Why can't you do the job with the tools we have? [SD3] The lack off tools and training means that jobs take longer and people become unhappy. [TM2]	Because SPL processes dictate a uniform way in which activities are carried out, it becomes worthwhile and meaningful to automate these processes and consequently easier to train people to follow the processes.

5.4.3.4 Support and Maintenance

In a project-centric structure, the project manager is expected to complete a project within the budget and timescale, using the resources available. Consequently, optimisation occurs at project level rather than at organisational level. Although this approach may optimise project profits in the short term, it lacks the potential benefits of organisational level optimisation, such as identifying and exploiting recurring patterns in terms of reuse. A serious long-term effect of this approach is that the company accumulates an assortment of disparate systems that need to be maintained and supported. Considering that, in the defence industry, some systems need to be maintained for twenty to thirty years after delivery, the magnitude of this problem can be appreciated. Obviously, it is considerably more difficult to maintain ten very different systems, than it is to maintain ten systems based on a similar, consistent design philosophy and shared assets, such as the members of an SPL.

The following quote, from one of the software developers interviewed, captures the essence of this problem in a project-centric environment.

> If you could reuse stuff in a controlled way and benefit from previous years of debugging design and testing, … and guys that were experts at that point might have even left the company but we can still benefit from their work … you know it's excellent. Once again … I remember just when I started on this stuff … we did what you describe as opportunistic reuse … you know you

> copy then go on from that and hack it to pieces. So you've got this new thing you know. So you've got a good step and that's how it's always been working. That's copy and paste at a high level. You get this big initial step and then you go off on your own. You now do work on this and that's fantastic work but you can't take it back into that other product that you copied from. They don't benefit from it. And in the same way product A that you originally copied they now proceed with development and they now do some excellent work there and project B that was copied from it doesn't benefit from that. You diverge and split the two things up and from there on you don't benefit at all and in fact in the long term in terms of maintenance and so on it gets a nightmare. [SD1]

Support and maintenance are important activities in our business, which are complicated by our project-centric structure. An SPL structure would ensure maximum commonality between products and therefore simplify these activities.

5.4.3.5 Summary of Findings: Comparative Themes

Table 5-10 presents a summary of the finding relating to the *comparative themes* category. These are the emergent themes that allow comparison between a project-centric and an SPL software development environment.

Table 5-10: Summary of Findings: Comparative Themes

Comparative Themes		
Theme	**Sub-category**	**Findings**
Intellectual property	Organisational	Whereas a project-centric structure makes it difficult to accumulate and benefit from IP, an SPL structure supports the accumulation of IP and facilitates benefiting from it.
Human resources and career development	Context	In a project-centric environment, developers are sometimes reduced to performing fairly mundane tasks which wastes resources and negatively affects career development.
Software development frustrations	Technical	A number of software development frustrations that recurred in the discussions relating to several of the interview questions. SPL engineering provides solutions to these frustrations.
Support and maintenance	Technical	The wide variety of systems produced by the project-centric approach present problems with respect to long-term support and maintenance.

5.5 SUMMARY

This chapter described the findings of the analysis of the transcripts from the field interviews. The transcripts were analysed using a simple coding process (*see* Section 3.3.9), which was employed to uncover the themes running through data. The themes are organised into two classes: the primary themes, relating directly to the topics of the interview questions, and the emergent themes, which are relevant themes that emerged during the coding process.

Both classes comprise three categories (*see* Figure 5-1). Table 5-11 provides a key to the summary of analysis results for each category.

Table 5-11: Key to Data Analysis Result Summaries

THEME CATEGORY	**RESULT SUMMARY**
PRIMARY THEMES	
General need and suitability	Table 5-1
Organisational	Table 5-2
Technical	Table 5-3
EMERGENT THEMES	
Project-centric development environment	Table 5-7
SPL development environment	Table 5-8
Comparative themes	Table 5-10

Chapter 6. Discussion of Findings

6.1 INTRODUCTION

The study consisted of three phases, the literature study, the contextualisation and the field interviews. This chapter provides a discussion of the results drawn from these three phases. The discussion begins with a brief summary of the study and then proceeds to address the findings relating to the six subsidiary questions and the emergent themes.

6.2 DISCUSSION OF SUBSIDIARY QUESTIONS

The primary research question is:

What are the issues related to the introduction of systematic software reuse in a small project-centric organisation?

This research study consisted of three phases:

- A theoretical framework for the study was established using information derived from a comprehensive literature study.
- The research study was contextualised to gain an understanding and appreciation of the social and historical factors which led to the company's current situation.
- The perceptions of fellow practitioners were determined and analysed, based on a series of in-depth field interviews.

The three phases made use of four data sources: a literature study, an elite interview, field interviews and personal experience. During the research design, the primary research question was decomposed into six subsidiary questions of an exploratory nature. Table 6-1 illustrates how the four data sources contributed to the answering of the subsidiary questions.

Table 6-1: Relationship between Subsidiary Questions and Data Sources

Subsidiary Question	Subsidiary Question Topic	Literature Study	Field Interviews	Elite Interview	Personal Experience
SQ1	Generally accepted approach to systematic reuse	■			
SQ2	Context of the study	□	□	■	□
SQ3	Reasons for maintaining a project-centric structure	□	■		□
SQ4	Need for systematic software reuse	□	■		□
SQ5	Suitability of accepted software reuse approach	□	■		□
SQ6	Potential systematic software reuse adoption issues	□	■		□
■ = main data source, □ = additional data source					

The research findings relating to each of the subsidiary questions are discussed in Sections 6.2.1 to 6.2.6 and are followed by a discussion of the additional topics that emerged from the analysis of the transcripts of the field interviews in Section 6.3.

6.2.1 SQ1: Generally Accepted Approach to Systematic Reuse

Question	*What is a generally accepted approach to systematic software reuse within the industry?*
Aim	To identify a particular approach to provide a theoretical context for investigating systematic reuse.

The literature study (*see* Chapter 2) established the theoretical framework for the overall research study. To achieve this, it was necessary to investigate the theoretical context of software reuse and to identify a single generally accepted and practical software reuse approach to facilitate a debate of its general suitability for the company and the applicability of the practices involved in this approach.

Early software reuse efforts concentrated almost exclusively on the reuse of code, but later progressed to the reuse of design and architecture. Eventually, these efforts were extended to cover all intellectual assets associated with the development and support of software. The literature study revealed SPL engineering as a widely accepted, practical and comprehensive approach to systematic software reuse. The following passage repeated from Section 2.3 provides an apt summary of the findings of subsidiary question SQ1: Software product line engineering is emerging as a promising and practical approach to systematic software reuse (Böckle et al., 2002; Knauber & Succi, 2002; Northrop, 2002b).

6.2.2 SQ2: Context of the Study

Question	*What is the context of the study and more specifically what are the historical factors which contributed to the organisation being project-centric?*
Aim	To determine and comprehend the historical and social context of the study.

As Darke, Shanks, and Broadbent (1998, p. 285) remind us that: "The goal of analysis in interpretive studies in information systems is to produce an understanding of the contexts of information systems and the interactions between these systems and their contexts". This quote also holds true for the study of IS development practices. The primary purpose of the contextualisation was to provide an understanding of the circumstances and environment that led the company to adopt and maintain its project-centric structure. The relevance of this purpose increased when the literature study revealed the sensitivity of software reuse to its organisational context and, in particular, suggested that a project-centric structure is unsuitable for systematic software reuse. The latter view was later endorsed by several of the interview participants.

Section 3.2.5 discusses the importance of context to a qualitative study. The contextualisation of this study is covered in detail in Chapter 4. The context, fundamental to an understanding of the company's historical and contemporary behaviour, was analysed using a simple framework that addresses the societal, organisational, group and individual conceptual levels. In order to provide the reader with an adequate contextual understanding, the contextualisation chapter precedes the chapter on analysis of the field interviews (Chapter 5). The contextualisation describes the circumstances and environment, revealed by this study, which led to the company adopting a project-centric structure and business strategy.

6.2.3 SQ3: Reasons for Maintaining a Project-centric Structure

Question	*Are there motivating factors for maintaining a project-centric organisational structure?*
Aim	To establish and comprehend the contemporary context of the study.

Although the field interviews revealed no real support for retaining the project-centric structure, there were also no convincing indications of a real desire or intention to make a substantial shift from this structure.

As reported, during the study a change to the company's organisational structure was effected, and although a matrix structure was introduced, the company still remains strongly project-oriented. Given the necessary authority and resources, the matrix structure can be used to address some of the problematic reuse issues, such as selection of tools, provision of training, enforcing of processes and reuse of assets. However, unless the company's business strategy is changed from one that is reactive and project-oriented to one that is proactive and product-oriented, the benefits of systematic reuse are likely to be limited.

6.2.4 SQ4: Need for Systematic Reuse

Question	*To what degree is there a need within the organisation for systematic software reuse?*
Aim	To assess the level of need for software reuse in the company.

The majority of the participants were in agreement that there is a need to make the company's software development more efficient and more deterministic. Motivations given for this response were primarily the time and cost pressures experienced by the project teams.

The interviews and ensuing discussions also revealed that there is very limited systematic reuse currently being practised, and the general perception among participants was that the company stands to benefit from the introduction of systematic reuse.

6.2.5 SQ5: Suitability of Software Product Line Engineering

Question	*How suitable is the generally accepted approach for adoption by the organisation?*
Aim	To determine the level of compatibility of the identified software reuse approach for the company.

The dominant perception among participants was that SPL engineering provides a practical approach to systematic software reuse. However, concern was expressed that this approach might not be totally suitable for our company.

One concern expressed was that it might be better suited to larger companies and to new product developments. Our company is relatively small and is already committed to product development, especially with regard to HUMS ground stations. The issue of organisation size is dealt with in Section 6.3.2.

Another concern was that SPL engineering would limit the company's flexibility in addressing customer requirements. This issue is addressed in Section 6.2.6.

6.2.6 SQ6: Potential Systematic Software Reuse Adoption Issues

Question	*What are some of the technical and organisational issues that might influence the implementation of systematic software reuse within the organisation?*
Aim	To identify potential technical and organisational factors that could pose a challenge for the company.

Several topics were identified as potential challenges for the company concerning systematic software reuse. Each of the topics covered specifically by interview questions is now summarised:

Customer Interface Management: The specific issues that were identified on the topic of customer interface management concerned intellectual property and the responsiveness to customers' requirements. The issue of intellectual property was raised in response to several interview questions and is handled separately in Section 6.3.3.

Since a single project or one-off development allows almost infinite flexibility, there is little doubt that an SPL reduces flexibility and ultimately restricts customer responsiveness. However, the key question to be asked is “Do we require more flexibility than an SPL can practically allow for?” If the answer is affirmative, then an SPL is not a suitable solution. For systematic software reuse and SPL engineering to be successful, the problem domain must be sufficiently mature and stable. A mature and stable problem domain has a well-established *modus operandi* and norms that are generally accepted industry-wide, which effectively reduces the required range of flexibility.

To summarise, if the required customer responsiveness is genuinely in danger of compromise, then software product line engineering is possibly not an appropriate solution. After all, SPL engineering is a strategy for exploiting the commonality between products; if commonality is lacking, SPL engineering is not really applicable. A lack of commonality would obviously restrict the potential benefits of any reuse strategy.

Funding: The interview responses showed that there is almost no consensus on a method of funding a potential SPL initiative. Funding is an issue that probably needs to be addressed by a detailed business case analysis, however the study indicates that it could prove to be a difficult issue on which to reach consensus.

Processes: Although the company has considerable experience with software development processes, some participants expressed concern regarding the lack of uniform and consistent application of these processes. Based on participants’ comments, this situation could be ascribed to the current autonomous nature of projects and the level to which customers are allowed to influence our development processes. In an SPL environment the structure makes it difficult to deviate from processes and the customer also has much less opportunity to influence the development.

Configuration Management and Change Control: In spite of the extra demand placed on configuration management and change control, the general perception is that this will be an area that poses little challenge for the company.

Problem Domain Knowledge: The literature study and discussions revealed that there are multiple facets to problem domain knowledge. With respect to HUMS ground stations, the company has, in the opinion of some participants, gained considerable knowledge about HUMS aspects to which the project teams have been directly exposed. Owing to the manner in which this knowledge has been acquired, much of it is incidental and possibly incomplete. Consequently, our problem domain knowledge lacks depth and is limited in areas to which we have been less exposed[27]. There is also a lack of knowledge regarding the trends and future directions of HUMS. An SPL effort must be supported by a deliberate drive to acquire and accumulate knowledge of the problem domain in order to support the SPL vision and strategy.

Tools and Training: The technical staff expressed an element of distrust of senior management's willingness to sanction the provision of tools and training in an SPL environment. However, responses provided by the organisational management group indicate an acknowledgement of the current problems in this regard and recognition of the need to address the situation. In defence of senior management, the interview discussions and the literature study also revealed the difficulties posed by a project-centric environment for standardising on specific tools and training.

Software Architecture: The general perception among participants is that software architecture is a purely technical issue, which should not pose a problem given suitably trained developers, sufficient tools, and enough time.

Component Technology: Based on the recent history in which several component technologies have come and gone, the selection of a suitable component technology is certainly a challenge. However, as pointed out in the literature study the recent emergence of J2EE and .NET component technologies has been greeted with some optimism (Carlson, 2004; Schmidt & Buschmann, 2003). Participants acknowledged the importance of making the *right* choice of component technology.

In summary, a several topics were discussed and some problems were exposed, but it appears that most of the problems are well understood and the solutions are within grasp. Customer responsiveness is probably the main exception and, as explained, if the required responsiveness to customer requirements is genuinely in danger of compromise, SPL engineering is probably not a suitable solution.

[27] This opinion was expressed especially by members of the organisational management group.

6.3 EMERGENT THEMES

In addition to the anticipated themes relating to the interview questions, several other themes emerged from the field interview transcripts. Those that were considered to contribute to the study fall into three categories:

- project-centric development environment;
- issues specific to the software product line development environment; and
- themes that allow for comparison between the project-centric and the SPL development environments.

These categories are discussed in Sections 6.3.1, 6.3.2 and 6.3.3 respectively.

6.3.1 Project-Centric Development Environment Themes

Apart from project-centric issues, which were included in the discussion of context in Chapter 4, the only other emergent theme specific to the project-centric environment relates to the attitude of software developers.

Developer Attitudes: Several examples of problematic attitudes among software developers were raised by participants in the technical management group. A literature search revealed these to be common attitudes which act as software reuse disincentives (Lynex & Layzell, 1998). These disincentives inhibit reuse success and some can even result in the failure of a reuse initiative, but as Lynex and Layzell (1998) express it: “Even though the inhibitors identified may at first seem immense the real message is that it is better to start somewhere than not at all and that all these problems may be overcome provided the will to succeed exists”.

6.3.2 SPL Development Environment Themes

Several themes specific to the SPL environment emerged from the field interview transcripts. The findings on these themes are now summarised.

Vision: Owing to the reactive, single-system development mentality that exists in a project-centric company, there is little incentive to establish organisational and product-level visions (or strategies). However, SPL engineering, being a product-oriented approach, requires the company to become proactive and an important part of this is to establish these visions. These facts were revealed by the literature study and clearly recognised by some participants.

Implementation Strategy: Bosch (2002b) confirms that, whereas a revolutionary implementation strategy promises higher returns, an evolutionary approach carries less risk. In spite of an evolutionary implementation strategy potentially costing more and taking longer (Bosch, 2002b), the general participant perception was that this would be a more pragmatic strategy for our company to follow.

Management Support and Leadership: Management support and leadership should not be limited to the SPL adoption phase, it must be a continuous, ongoing SPL support effort (Clements & Northrop, 2001, p. 45). This opinion was repeatedly stressed by most of the technical managers and software developers throughout the interview discussions.

Organisational Structure: An appropriate organisational structure is required to support SPL engineering (*see* Section 2.4.2.3). As mentioned in Section 6.2.3, there appears to be a perception – especially among some of the technical managers interviewed – that the introduction of a matrix structure constitutes a suitable organisational structure to support systematic reuse. This reinforces the opinion that a major change in mindset (*see* **Mindset** below) would be necessary in the company before an SPL effort can be attempted.

Organisation Size: There was concern among participants that SPL engineering is only suitable for large companies; however, a number of authors provide evidence to allay this concern. Gacek et al. (2001, p. 1) address the misconception that “software product lines may be fine for large organisations, but not for small ones”. They cite an example of a successful SPL development in a small organisation, and maintain that the benefits promised by SPLs are of even greater importance to smaller organisations. Knauber et al. (2000) cite a number of successes in small and medium-sized organisations. Clements and Northrop (2001, pp. 485-511) provide a detailed description of successful software product line development in a small organisation. Thus, the evidence provided in the literature suggests that small organisations are not excluded from successful SPL engineering.

Mindset: The literature also emphasises a need for organisations that adopt SPL engineering to overcome the project mindset and to foster a product line culture (Böckle et al., 2002; L. G. Jones, 1999). The perceptions among the participants suggest that this is potentially one of the biggest hurdles facing our company, particularly at organisational level.

To summarise: the identified SPL development themes could be added to a checklist for special attention, should the company decide to make an SPL initiative. Perceptions suggest that, should the company decided to adopt SPL engineering, the necessary change in mindset might present a major hurdle; however the company would also need to take an evolutionary implementation strategy, work on establishing a clear product line vision, and provide continuous management support and leadership.

6.3.3 Comparative Themes

Four themes allowing for comparison between a project-centric and a SPL environment emerged from the field interview transcripts. The discussion of these follows.

Intellectual Property: The literature study indicated potential problems relating to intellectual property that a project-centric environment might encounter. The field interview discussions confirm that intellectual property problems currently exist within the company. Examples of these are:

- There is a lack of a suitable mechanism for securing IP and consequently much of the company's IP currently exists in individuals' heads.
- Difficulty exists in distinguishing between IP belonging to the company and that belonging to the customer.

Whereas a project-centric structure makes it is difficult to accumulate and benefit from IP, an SPL structure supports the accumulation of IP and facilitates benefiting from it. Participants anticipated that there might be difficulty in convincing customers to pay for access to IP embedded in products. A participant mentioned that evidence of internally funded R&D would strengthen our case for proving ownership of IP and justifying its value.

Human Resources and Career Development: Interview participants identified problems relating to career development and the wastage of human resources that are attributable to the company's project-centric structure. Although these problems can possibly be ignored in the short-term, they are certain to have consequences in the longer term, such as staff dissatisfaction and resignations. The literature indicates that an SPL environment offers solutions to these problems in terms of providing "more career development opportunities" (Peterson, 2003, p. 383) and resulting in "higher morale and greater job satisfaction" (Clements & Northrop, 2001, p. 21) among developers.

Software Development Frustrations: Discussions with participants highlighted a variety of frustrations experienced by software developers in the project-centric environment. I consulted the literature to see whether SPL engineering provides solutions in this regard. Based on the evidence in Table 5-9, SPL engineering provides potential solutions for addressing these frustrations.

Support and Maintenance: Long-term support and maintenance of a large number of products with little or no design similarity could escalate into an unmanageable situation, owing to the wide range of skills and knowledge required. Conversely, support and maintenance is considerably easier if the products share a common architecture, components, documentation and other assets. In fact, the SPL approach presents an opportunity to optimise long-term support and maintenance and could even turn it into a lucrative business opportunity.

To summarise: four comparative themes were identified, each representing a problem or a class of problems that exists in a project-centric development environment. In each case, SPL engineering offers a potential solution. This information could be useful in motivating the adoption of an SPL initiative in the company.

6.4 SUMMARY OF FINDINGS

The purpose of this study was to investigate the potential implementation of systematic software reuse in a small project-centric organisation and the primary research question was: *What are the issues related to the introduction of systematic software reuse in a small project-centric organisation?*

In Chapter 5, the findings are addressed in two classes: findings related to the subsidiary research questions and findings related to the emergent themes. Table 6-2 provides a condensed summary of each subsidiary question, its aim and findings.

Table 6-2: Condensed Summary of Findings Relating to Subsidiary Research Questions

SQ1	
Question	*What is the generally accepted approach to systematic software reuse within the industry?*
Aim	To identify a particular approach to provide a theoretical context for investigating systematic reuse.
Findings	Software product line engineering is accepted as one of the most promising approaches aimed at achieving software reuse.
SQ2	
Question	*What is the context of the study and more specifically what are the historical reasons for the organisation being project-centric?*
Aim	To determine and comprehend the historical and social context of the study.
Findings	The contextualisation describes the circumstances and environment, revealed by this study, which led to the company's project-centric structure and business strategy.
SQ3	
Question	*Are there motivating factors for maintaining a project-centric organisational structure?*
Aim	To establish and comprehend the contemporary context of the study.
Findings	Although the field interviews revealed no real support for retaining the project-centric structure, there were also no indications of a real desire or intention to make a substantial shift from this structure.
SQ4	
Question	*To what degree is there a need within the organisation for systematic software reuse?*
Aim	To assess the level of need for software reuse in the company.
Findings	The majority of participants were in agreement that there is a need to make the company's software development more efficient and more deterministic.
SQ5	
Question	*How suitable is the generally accepted approach for adoption by the organisation?*
Aim	To determine the level of compatibility of the identified software reuse approach for the company.

Findings	The dominant perception among participants was that SPL engineering provides a practical approach to systematic software reuse. However, concern was expressed that this approach might not be totally suitable for our company.
SQ6	
Question	*What are some of the technical and organisational issues that might influence the implementation of systematic software reuse within the organisation?*
Aim	To identify potential technical and organisational factors that could pose a challenge for the company.
Findings	Customer responsiveness is probably the main challenge and, if the required responsiveness to customer requirements is genuinely in danger of compromise, SPL engineering is probably not a suitable solution.

Table 6-3 provides a condensed summary of the findings relating to the emergent themes.

Table 6-3: Condensed Summary of Findings Relating to the Emergent Themes

THEME	FINDING
PROJECT-CENTRIC DEVELOPMENT THEMES	
Developer attitudes	Problematic attitudes exist among some of the software developers which form reuse disincentives. Having identified these attitudes, it is necessary to address them, otherwise they will inhibit the success of the reuse strategy.
SPL DEVELOPMENT THEMES	
Vision	A clear and well-defined product vision is an essential foundation to an SPL.
Implementation strategy	The general perception is that an evolutionary implementation strategy would be a more pragmatic strategy for the company.
Management support and leadership	The technical managers and software developers stressed the need for continuous management support and leadership to ensure success.
Organisational structure	An appropriate organisational structure is needed to support SPL engineering.
Organisation size	Although a number of participants were concerned that SPL engineering was only suitable for large organisations, the literature indicates that it has been successfully adopted by small organisations.
Mindset	The perceptions among the participants suggest that the change of mindset from a project-centric company to a product-oriented company is potentially one of the biggest hurdles facing our company.

Theme	Finding
Comparative Themes	
Intellectual property	SPL structure supports the accumulation of IP and facilitates benefiting from it.
Human resources and career development	SPL environment provides “more career development opportunities” (Peterson, 2003, p. 383) and results in “higher morale and greater job satisfaction” (Clements & Northrop, 2001, p. 21) among developers.
Software development frustrations	SPL engineering provides some solutions for addressing software developers’ frustrations.
Support and maintenance	The SPL approach provides an opportunity to optimise long-term support and maintenance.

The contextualisation describes the national, international, inter-organisational and organisational environments within which the company evolved and how factors in these environments contributed to the company adopting its project-centric structure. The contextualisation explains how changes in the industry have led to increased pressure on software developers. It also discusses the negative consequences that this project-centric context has for projects and individuals, especially with regard to software reuse.

Although there was general agreement on the company’s need for systematic software reuse and that SPL engineering is a practical approach to this, there was concern that SPL engineering might be better suited to a larger organisation. The literature, however, suggests that this concern is unwarranted.

Of the technical and organisational practice areas addressed by the field interviews, management of the customer interface in general and responsiveness to customer requirements appear to have the potential be the most problematic. The findings of the literature study concur with this opinion (*see* Section 2.7).

The emergent themes provided insight into both problems currently existing within the company, and possible problems, with the introduction of SPL engineering to the company. SPL engineering offers potential solutions for all of the existing problems identified, with the possible exception of the problem of software developer attitudes. The advice offered in the literature for the developer attitude problem is to provide training, education and incentives (Lynex & Layzell, 1998). A crucial task for a company attempting the transition from single system development to SPL engineering is the adoption and institutionalisation of a product line culture (Böckle et al., 2002). This task involves all stakeholders and requires considerable amounts of training, learning, debating and adapting. Only once the company is well into this task can it be determined whether the potential problems of SPL engineering will become real problems.

6.5 REFLECTION

6.5.1 Methodological Reflection

This section provides reflection on the appropriateness of the chosen research methods and the level to which they might have influenced the findings.

As stated in Section 2.1, the purpose of the literature study was to establish a theoretical framework for the overall study, by identifying a "generally accepted approach to systematic software reuse within the industry" as well as practices relating to this approach to provide a framework for debate. I considered certain of the reuse practices identified during the literature study to be of special interest because of the potential challenges that they might pose for the company. The findings indicate that practices such as:

- making a business case;
- SPL funding; and
- configuration management and change control

actually contributed little in the way of interest to the study. The selection of specific reuse practices for inclusion in the interview questions was subjective, and possibly resulted in the omission of practices deemed by some participants to be of greater relevance. This matter could have been addressed by adopting a two-phased interview process to allow the participants in the initial phase first to identify the reuse processes considered necessary for discussion.

Even among experienced software practitioners, software reuse is a widely-known but generally poorly understood topic. In the field interview process, my aim was to obtain a broad spectrum of opinions by targeting all of the parties directly involved.

My initial plan was to supply the interview participants with two papers that were carefully selected to provide essential background information. However, as mentioned in Section 3.3.5, a pilot run revealed that more comprehensive information was required. A pragmatic solution to the problem was to give participants an initial copy of the findings of my literature study (*see* Chapter 2). Although this solved the specific problem, in retrospect it created a bias-related problem owing to the fact that the literature study reflects a personal interpretation of the literature, which to some degree influenced the perceptions of the participants.

An alternative solution might be to provide additional independent background material, but this, too, has drawbacks, since it would inevitably require the participant to do a lot more reading and it would also be necessary to ensure that the selection of the background material carried no specific bias.

As mentioned, during the study the company changed ownership and an adjustment was made to the organisational structure by introducing a matrix system. In spite of this change, the company maintained its strong project-orientation. As shown in Table 3-6, certain of the participants were interviewed before and others after the change. A possible influence that the change had on the findings relates to the opinions of the three organisational managers, who, prior to the change, are likely to have argued in favour of maintaining the project-centric structure.

The purpose of this study was to investigate the potential implementation of systematic software reuse in this company. Apart from the issues already addressed, the chosen research methods were successful for achieving this purpose.

6.5.2 Scientific Reflection

This section provides reflection on what has been learned in this study and the contribution it makes to the scientific body of knowledge.

Rothenberger and Nazareth (2002) claim that “despite the potential rewards from an effective reuse program, it appears that widespread software reuse is not particularly prevalent”. Lim (1998) points out that few of the organisations practicing reuse actually achieve its full potential unless the reuse initiative is integrated into the company’s business strategy in the way that SPL engineering permits.

Our company benefits to a limit degree from software reuse. The study highlights the company's need for software reuse and tests the perceptions of participants regarding the applicability of SPL practices in the company. The study also reveals the level to which the company's project-centric structure impacts on reuse related issues and how the SPL approach might remedy this.

Although software reuse, like most other aspects of information systems, is highly context dependent, the findings of this study should provide useful guidance to small companies contemplating a reuse initiative, especially those that are project-oriented in structure and strategy.

Prominent among the aims of practitioner research is a transformation of the "self, colleagues and work context" (Darke et al., 1998, p 209). In line with these aims, the findings of this study will certainly contribute to the company's understanding of itself and could, in the future, contribute to a transformation of the workplace at various levels.

Within our company, this study has:

- served to raise the awareness of the need for software reuse;
- improved the understanding of software reuse;
- highlighted the negative effects of a project-centric structure; and
- identified the potential challenges and solutions provided by SPL engineering.

6.6 RECOMMENDATIONS FOR FURTHER RESEARCH

SPL engineering optimises the level of reuse by addressing it at all levels of the software development process, but an issue which this research accentuates is that SPL engineering is specifically for use in a product-oriented environment. There are companies that provide the service of custom software development and are therefore non-product-oriented by choice. A company of this type is unlikely ever to achieve the same level of reuse as a company producing a family of products with a high level of commonality. However, there must be some benefits that a non-product-oriented company can extract from software reuse. Possible areas of future research might be to explore the differences between product- and non-product-oriented reuse and the establishment of a non-product-oriented software reuse framework and guidelines similar to those available for SPL engineering.

Another issue accentuated by this research is the comprehensiveness of SPL engineering. As has been pointed out, because code only comprises a small portion of a software development effort, it makes good sense to extend reuse efforts to all aspects of the software development cycle. This idea could be taken further by extending the reuse philosophy beyond the software development cycle to all aspects of the product. In the case of our company, this would include aspects such as electronic hardware and mechanical-housing development. A study could be done to create a product line framework and guidelines for this which form a superset of the SPL framework and guidelines.

6.7 CONCLUSION

The purpose of this study was to investigate the potential implementation of systematic software reuse in a small project-centric organisation and thereby identify the specific issues involved. Although recent history indicates that the company has been successful with its project-centric organisational structure and business strategy, the study highlighted a need for the company to improve its software development efficiency via a systematic software reuse strategy.

The literature study identified SPL engineering as a generally accepted approach to systematic software reuse. Elements from the SEI Product Line Practice Framework were used as the basis of interview questions used to determine perceptions on the suitability of this approach for the company. Although there was general agreement that this is a practical approach to systematic reuse, the principal concern was that SPL engineering could hamper the company’s responsiveness to customer requirements.

Following the tradition of interpretive research, the study revealed useful findings beyond those originally expected. The majority of these additional findings related to problems in the company that are attributable to its project-centric structure and issues anticipated with SPL engineering adoption.

The problems relating to the project-centric structure concern important issues such as the management of intellectual property, career development and long-term product support. These issues need to be addressed by the company irrespective of whether a specific software reuse strategy is adopted. However, it is noteworthy that SPL engineering offers potential solutions to the problems identified: a fact that could be used to motivate the company's adoption of SPL engineering.

Although the field interviews addressed some of the potential problems that the company might experience with SPL engineering, additional potential problems were also identified by participants. These related largely to organisational issues such as establishing a product line vision, an implementation strategy, and a suitable organisational structure, as well as ensuring the correct mindset and management support. A company wishing to make the transition to SPL engineering needs to start by establishing a product line culture. Assuming that this culture is successfully adopted, the organisational requirements will be better appreciated, and some of these potential problems should disappear. It would nevertheless be prudent to manage a checklist of the potential problems identified.

The literature study indicates that SPL engineering optimises the level of reuse across software products. It also recommends that a suitable organisational structure is required to support SPL engineering. Although, the field interviews identified certain perceived problems with adopting SPL engineering, they also uncovered barriers to software reuse that are attributable to the company's project-centric structure. From the literature it appears that SPL engineering holds solutions to these problems. These findings suggest that to achieve an optimal level of reuse, the company needs to relinquish its project-centric structure and adopt a product-oriented reuse strategy, such as SPL engineering.

In conclusion, the potential implementation of systematic software reuse was investigated in a small project-centric company. In this respect, the study identified potential issues as well as existing reuse-related issues attributable to the company's project-centric structure.

References

Abrahams, D. (2001). *Defence Conversion in South Africa: A Faded Ideal?* : Security Sector Transformation Programme, Institute for Security Studies.

Amir, D. (2005). *The Use of "First Person" Writing Style in Academic Writing: An Open Letter to Journal Editors, Reviewers and Readers*. Retrieved 15 February 2007, from http://www.voices.no/columnist/colamir140305.html

Avison, D., & Elliot, S. (2006). Scoping the Discipline of Information Systems. In J. L. King (Ed.), *Information Systems: The State of the Field* (pp. 3-18). Hoboken, NJ: J Wiley & Sons.

Avison, D., Lau, F., Myers, M. D., & Nielsen, P. A. (1999). Action Research. *Communications of the ACM, 42* (1), pp. 94-97.

Bahrami, A. (1999). *Object Oriented Systems Development*: McGraw-Hill.

Baskerville, R. L., & Myers, M. D. (2002). Information Systems as a Research Discipline. *MIS Quarterly, 26* (1), pp. 1-14.

Bass, L., Clements, P., Cohen, S., Northrop, L., & Withey, J. (1997). *Product Line Practice Workshop Report* (No. CMU/SEI-97-TR-003). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Bayer, J., Muthig, D., & Göpfert, B. (2001). *The Library Systems Product Line: A KobrA Case Study*: Fraunhofer IESE.

Bergey, J., Fisher, M., Gallagher, B., & Jones, L. (2000). *Basic Concepts of Product Line Practice for the DoD* (No. CMU/SEI-2000-TN-001): Carnegie Mellon University, Software Engineering Institute.

Birk, A. (2002). *Three Case Studies on Initiating Product Lines: Enablers and Obstacles.* Paper presented at the PLEES'02.

Birk, A., Heller, G., John, I., Joos, S., Müller, K., Schmid, K., et al. (2003). *Requirements Engineering for Product Lines*. Kaiserslautern, Germany: Fraunhofer IESE.

Birk, A., Heller, G., John, I., von der Maßen, T., Müller, K., & Schmid, K. (2003). *Product Line Engineering Industrial Nuts and Bolts*: Fraunhofer IESE.

Böckle, G., Bermejo Muñoz, J., Knauber, P., Krueger, C. W., do Prado Leite, J. C. S., van der Linden, F., et al. (2002). *Adopting and Institutionalizing a Product Line Culture.* Paper presented at the SPLC2 - 2nd Software Product Line Conference.

Böckle, G., Clements, P. C., McGregor, J. D., Muthig, D., & Schmid, K. (2004). Calculating ROI for Software Product Lines. *IEEE Software*, pp. 23-31.

Boehm, B. (1999). Managing Software Productivity and Reuse. *IEEE Computer*, pp. 111-113.

Bosch, J. (2000). *Design and Use of Software Architectures – Adopting and Evolving a Product-line Approach*: Addison-Wesley.

Bosch, J. (2002a). *Architecture-Centric Software Engineering.* Paper presented at the International Conference on Software Reuse (ICSR), Austin, Texas, April 2002.

Bosch, J. (2002b). *Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization.* Paper presented at the 2nd Software Product Line Conference (SPLC-2), San Diego, CA.

Braun, C. L. (1999). *A Lifecycle Process for the Effective Reuse of Commercial Off-the-Shelf (COTS) Software.* Paper presented at the SRR'99, Los Angeles CA USA.

Brooker, R., & Macpherson, I. (1999). Communicating the Processes and Outcomes of Practitioner Research: an opportunity for self-indulgence or a serious professional responsibility? *Educational Action Research, 7* (2), pp. 207-221.

Brownsword, L., & Clements, P. (1996). *A Case Study in Successful Product Line Development* (No. CMU/SEI-96-TR-016): Carnegie Mellon University, Software Engineering Institute.

Bühne, S., Chastek, G., Käkölä, T., Knauber, P., Northrop, L. M., & Thiel, S. (2004). *Exploring the Context of Product Line Adoption.* Paper presented at the PFE 2003.

Carlson, B. (2004). Software Reuse is Dead, Long Live Software Reuse. *Weblogic Developer's Journal, 1*.

Clements, P. C. (1997). *Successful Product Line Engineering Requires More Than Reuse.* Paper presented at the WISR8 - Eighth Annual Workshop of Instituitionalizing Software Reuse, Ohio State University, 23- 26 March 1997.

Clements, P. C. (1999). Software Product Lines: A New Paradigm for the New Century. *SEI Interactive*, pp. 1-7.

Clements, P. C. (2002). Being Proactive Pays Off, Point-Counterpoint. *IEEE Software*, pp. 28-30.

Clements, P. C., Donohoe, P., Kang, K., McGregor, J., & Northrop, L. M. (2001). *Fifth Product Line Practice Workshop, Technical Report*: SEI.

Clements, P. C., Jones, L. G., Northrop, L. M., & McGregor, J. D. (2005). Project Management in a Software Product Line Organization. *IEEE Software*, pp. 54-62.

Clements, P. C., & Northrop, L. M. (1996). *Software Architecture: An Executive Overview*: Software Engineering Institute, Carnegie Mellon University.

Clements, P. C., & Northrop, L. M. (2001). *Software Product Lines – Practices and Patterns*: Addison-Wesley.

Clements, P. C., & Northrop, L. M. (2002). *Salion, Inc.: A Software Product Line Case Study*: SEI.

Cohen, S. (2002). *Product Line State of the Practice Report* (No. CMU/SEI-2002-TN-017). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.

Czarnecki, K., & Eisenecker, W. (1999). *Components and Generative Programming.* Paper presented at the Joint 7th European Software Engineering Conference and ACM SIGSOFT International Symposium on the Foundations of Software Engineering (ESEC/FSE'99), Toulouse, France.

Darke, P., Shanks, G., & Broadbent, M. (1998). Successfully completing case study research: combining rigour, relevance and pragmatism. *Information Systems Journal, 8*, pp. 273-289.

Desouza, K. C., Raider, J. J., & Davenport, T. H. (2003). *Intellectual Asset Reuse in Software Development*: Accenture Institute for Strategic Change.

Di Nitto, E., & Fuggetta, A. (1996). *Product Lines: What are the Issues?* Paper presented at the 10th International Software Process Workshop, Dijon, France.

Dikel, D., Kane, D., Ornburn, S., Loftus, W., & Wilsin, J. (1997). Applying Software Product-line Architecture. *IEEE Computer*, pp. 49-55.

Ebrahim, G. J., & Sullivan, K. (1995). *Mother and Child Health: Research Methods*. London: Book-Aid.

Evaristo, J. R., & Karahanna, E. (1997). Is North American IS Research Different from European IS Research? *The Data Base for Advances in Information Systems, 28* (3), pp. 32-43.

Fafchamps, D. (1994). Organisational Factors and Reuse. *IEEE Software*, pp. 31-41.

Fitzgerald, B., & Howcroft, D. (1998). *Competing Dichotomies in IS Research and Possible Strategies for Resolution.* Paper presented at the International Conference on Information Systems (ICIS), Helsinki, Finland.

Foote, B., & Yoder, J. (1995). *Evolution, Architecture, and Metamorphosis.* Paper presented at the Second Conference on Patterns Languages of Programs (PLoP '95), Monticello, Illinois, September 1995.

Frakes, W. B., & Kang, K. (2005). Software Reuse Research: Status and Future. *IEEE Transactions on Software Engineering, 31* (7), pp. 529 - 536.

Fraunhofer Institute. (2003). *PuLSE™ Product Lines for Software Systems*. Kaiserslautern, Germany: Fraunhofer Institut Experimentelles Software Engineering (IESE).

Gacek, C., Knauber, P., Schmid, K., & Clements, P. C. (2001). *Successful Software Product Line Develop Development in a Small Organization: A Case Study*: Fraunhofer IESE.

Garlan, D., & Perry, D. (1995). Guest Editorial. *IEEE Transactions on Software Engineering* (April 1995).

Gillham, B. (2001). *Case Study Research Methods* (1st ed.). London: Continuum.

Greenfield, J., & Short, K. (2003). Software Factories – Assembling Applications with Patterns, Models, Frameworks and Tools, *OOPSLA '03*.

Greenfield, J., & Short, K. (2004). *Software Factories – Assembling Applications with Patterns, Models, Frameworks and Tools*: Wiley.

Harvey, L. J., & Myers, M. D. (1995). Scholarship and practice: the contribution of ethnographic research methods to bridging the gap. *Information Technology & People, 8* (3), pp. 13-27.

Henninger, S. (1996). *Accelerating the Successful Reuse of Problem Solving Knowledge Through the Domain Lifecycle.* Paper presented at the International Conference on Software Reuse (ICSR96), Orlando, Florida.

Hevner, A. R., & March, S. T. (2003). The Information Systems Research Cycle. *IEEE Computer*, pp. 111-113.

Jacobson, I., Griss, M., & Jonsson, P. (1997). Making the Reuse Business Work. *IEEE Computer*.

Johnson, R. E. (1997). Frameworks = (Components + Patterns). *Communications of the ACM, 40* (10), pp. 39-42.

Jones, C. (2002). Defense Software Development in Evolution. *Crosstalk: The Journal of Defense Software Engineering*, pp. 26-29.

Jones, L. G. (1999). *Product Line Acquisition in the DoD: The Promise, The Challenges*: Carnegie Mellon University, Software Engineering Institute.

Khazanchi, D., & Munkvold, B. E. (2000). Is Information Systems a Science? An Inquiry into the Nature of the Information Systems Discipline. *The Data Base for Advances in Information Systems, 31* (3), pp. 24-42.

Khazanchi, D., & Munkvold, B. E. (2003). *On the Rhetoric and Relevance of IS Research Paradigms: A Conceptual Framework and Some Propositions.* Paper presented at the 36th Hawaii International Conference on System Sciences (HICSS'03).

Klein, H. K., & Myers, M. D. (1999). A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly, 23* (1), pp. 67-94.

Knauber, P., Muthig, D., Schmid, K., & Widen, T. (2000). Applying Product Line Concepts in Small- and Medium-Sized Companies. *IEEE Software* (September 2000), pp. 88-95.

Knauber, P., & Succi, G. (2002). Perspectives on Software Product Lines. *ACM SIGSOFT Software Engineering Notes, 27* (2), pp. 40-45.

Korpela, M., Mursu, A., & Soriyan, H. A. (2001). *Two Times Four Integrative Levels of Analysis: A Framework.* Paper presented at the IFIP TC8/WG 8.2 Working Conference, Boise, Idaho, USA.

Kranzberg, M. (1964). Industrial Revolution. In *Encyclopaedia Britannica* (Vol. 12, pp. 307): William Benton.

Kruchten, P., Obbink, H., & Stafford, J. (2006). The Past, Present and Future of Software Architecture. *IEEE Software* (March/April), pp. 22-30.

Krueger, C. (2002). Eliminating the Adoption Barrier, Point-Counterpoint. *IEEE Software* (July/August 2002), pp. 29-31.

Kuloor, C., & Eberlein, A. (2002). *Requirements Engineering for Software Product Lines.* Paper presented at the 15th International Conference on Software & Systems Engineering and their Applications (ICSSEA'02), Paris, France.

Laguna, M. A., González-Baixauli, B., López, O., & García, F. J. (2003). *Introducing Software Reuse in Mainstream Software Process.* Paper presented at the EuroMicro'03 - 29th Euromicro Conference.

Lam, W. (1998). A case-study of requirements reuse through product families. *Annals of Software Engineering, 5*, pp. 253–277.

Lee, A. S., & Liebenau, J. (1997). *Information Systems and Qualitative Research.* Paper presented at the International Conference on Information Systems and Qualitative Research, Philadelphia, Pennsylvania, USA, 31 May - 3 June 1997.

Lim, W. C. (1998). Strategy-driven reuse: Bringing reuse from the Engineering Department to the Executive Boardroom. *Annals of Software Engineering, 5*, pp. 85 - 103.

Lynex, A., & Layzell, P. J. (1998). Organisational Considerations for Software Reuse. *Annals of Software Engineering, 5*, pp. 105–124.

Matinlassi, M. (2004). *Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA.* Paper presented at the 26th International Conference on Software Engineering (ICSE'04).

McGregor, J., Northrop, L., Jarrad, S., & Pohl, K. (2002). Initiating Software Product Lines. *IEEE Software* (July/August 2002), pp. 24-27.

McKay, J., & Marshall, P. (2000). *Quality and Rigour of Action Research in Information Systems.* Paper presented at the European Conference on Information Systems, Vienna, Austria.

McWilliam, E. (2004). W(h)ither Practitioner Research? *Australian Educational Researcher, 31* (3), pp. 113-126.

Mili, A., Yacoub, S., Addy, A., & Mili, H. (1999). Toward an Engineering Discipline of Software Reuse. *IEEE Software*, pp. 22-31.

Mili, H., Mili, F., & Mili, A. (1995). Reusing Software: Issues and Research Directions. *IEEE Transactions on Software Engineering, 21* (6), pp. 528-562.

Moore, J. W. (1997). *Fundamental Principles of Software Reuse.* Paper presented at the WISR8 - Eighth Annual Workshop of Instituitionalizing Software Reuse, Ohio State University, 23-26 March 1997.

Morisio, M., Ezran, M., & Tully, C. (2002). Success and Failure Factors in Software Reuse. *IEEE Transactions on Software Engineering, 28* (4), pp. 340-357.

Morisio, M., Tully, C., & Ezran, M. (2000). Diversity in Reuse Processes. *IEEE Software* (July/August 2000), pp. 56-63.

Morse, J. M., & Richards, L. (2002). *Readme First for a User's Guide to Qualitative Methods*. California: Sage Publications.

Myers, M. D. (1997). *Critical Ethnography in Information Systems.* Paper presented at the International Conference on Information Systems and Qualitative Research, Philadelphia, Pennsylvania, USA, 31 May - 3 June 1997.

Myers, M. D. (1999). Investigating Information Systems with Ethnographic Research. *Communications of the Association for Information Systems, 2* (23), pp. 1-20.

Myers, M. D. (2006). *Qualitative Research in Information Systems*. Retrieved 22 May, 2006, from http://www.qual.auckland.ac.nz

Myers, W. (1997). Software Reuse: Ostriches Beware. *IEEE Computer*, pp. 119-120.

Northrop, L. M. (2002a). Foreword. *IEEE Software* (July/August 2002).

Northrop, L. M. (2002b). SEI's Software Product Line Tenets. *IEEE Software* (July/August 2002), pp. 32-40.

O'Brien, L., Stoermer, C., & Verhoef, C. (2002). *Software Architecture Reconstruction: Practice Needs and Current Approaches, SEI Technical Report*: SEI.

Orlikowski, W. J., & Baroudi, J. J. (1991). Studying Information Technology in Organizationa: Research Approaches and Assumptions. *Information Systems Research, 2* (1), pp. 1-28.

Parnas, D. (1976). On the Design and Development of Program Families. *IEEE Transactions on Software Engineering, 2* (1), pp. 1-9.

Pasetti, A., & Pree, W. (2000). *A Reusable Architecture for Satellite Control Software.* Paper presented at the IEEE/AIAA 19-th Digital Avionics Systems Conference, Philadelphia, PA, USA.

Pather, S., & Remenyi, D. (2004). *Some of the Philosophical Issues Underpinning Research in Information Systems: From Positivism to Critical Realism.* Paper presented at the SAICSIT.

Peterson, D. R. (2003). *Economics of Software Product Lines.* Paper presented at the Proceedings of the Fifth International Workshop on Product Family Engineering (PFE-5), LNCS 3014, Siena, Italy, November 2003.

Pree, W. (1997). *Component-Based Software Development - A New Paradigm in Software Engineering?* (Vol. 18): Springer-Verlag.

Pronk, B. (2002). *Product Line Introduction in a Multi-business Line Context - an Experience Report.* Paper presented at the PLEES'02.

Raccoon, L. (1997). Fifty Years of Progress in Software Engineering. *ACM SIGSOFT Software Engineering Notes, 22* (1), pp. 88-104.

Ran, A. (1999). *Software Isn't Built From Lego Blocks.* Paper presented at the SSR '99 Los Angeles, USA.

Rogerson, C. M. (1996). Defence Economic Restructuring and Conversion in South Africa. *GeoJournal, 39*, pp. 3-12.

Rothenberger, M., Dooley, J., Kulkarni, U. R., & Nader, N. (2003). Strategies for Software Reuse: A Principal Component Analysis of Reuse Practices. *IEEE Transactions on Software Engineering, 29* (9).

Rothenberger, M., & Nazareth, D. (2002). *A Cost-Benefit Model for Systematic Software Reuse.* Paper presented at the ECIS 2002, Gdańsk, Poland.

Schmid, K. (2002a). *An Assessment Approach to Analyzing Benefits and Risks of Product Lines.* Paper presented at the 25th Annual International Computer Software and Applications Conference (COMPSAC'01).

Schmid, K. (2002b). *A Comprehensive Product Line Scoping Approach and Its Validation.* Paper presented at the 24th International Conference on Software Engineering (ICSE'02), Orlando, Florida, USA., 19-25 May 2002.

Schmid, K. (2003). *People Management in Institutionalizing Product Lines* (No. IESE-Report No. 101.03/E): Fraunhofer IESE.

Schmid, K., & John, I. (2002). *Developing, Validating and Evolving an Approach to Product Line Benefit and Risk Assessment.* Paper presented at the EuroMicro'02 - 28th Euromicro Conference

Schmid, K., & Verlage, M. (2002). The Economic Impact of Product Line Adoption and Evolution. *IEEE Software*, pp. 50-57.

Schmidt, D. C., & Buschmann, F. (2003). *Patterns, Frameworks, and Middleware: Their Synergistic Relationships.* Paper presented at the 25th International Conference on Software Engineering (ICSE 2003), Portland, Oregon, USA.

SEI. (2006). *A Framework for Software Product Line Practice Version 4.2*. Retrieved 6 January 2007, from http://www.sei.cmu.edu/productlines/framework.html

Sim, S. E. (1999). *Evaluating the Evidence: Lessons from Ethnography.* Paper presented at the Proceedings of the Workshop on Empirical Studies of Software Maintenance, Oxford, England, 3-4 September 1999.

Stake, R. E. (1995). *The Art of Case Study Research*. Thousand Oaks, California, USA: Sage.

Standish Group. (1995). *The Chaos Report*. Retrieved 6 January 2007, from www.standishgroup.com/sample_research/PDFpages/chaos1994.pdf

Staples, M. (2004). *Change Control for Product Line Software Engineering.* Paper presented at the 11th Asia-Pacific Software Engineering Conference (APSEC'04).

Staples, M., & Hill, D. (2004). *Experiences Adopting Software Product Line Development without a Product Line Architecture.* Paper presented at the 11th Asia-Pacific Software Engineering Conference (APSEC'04).

Tricoglus, G. (2001). Living the Theoretical Principles of Critical Ethnography in Educational Research. *Educational Action Research, 9* (1), pp. 135-148.

US Air Force. (2003). *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems Command and Control Systems Management Information Systems, Condensed Version 4.1*: Department of the Air Force: Software Technology Support Center.

US Congress. (1992). After the Cold War: Living With Lower Defense Spending. In Office of Technology Assessment (Ed.) (pp. 1-229): U.S. Government Printing Office - Washington, DC.

US DoD. (1988). *DO-STD-2167A: Defense System Software Development* (February 29, 1988 ed.): Naval Publications and Forms Centre.

Vernazza, T., Galfione, P., Valerio, A., Succi, G., & Predonzani, P. (2000). *Moving toward software product lines in a small software firm: a case study.* Paper presented at the ICSE 2000 Workshop on Software Product Lines: Economics, Architectures and Implications, Limerick, Ireland, June 2000.

Wade, M., Biehl, M., & Kim, H. (2006). Information Systems is not a Reference Discipline (and What We Can Do about It). *Journal of the Association for Information Systems, 7* (5), pp. 247-269.

Walsham, G. (1993). *Interpreting Information Systems in Organizations* (1st ed.). Chichester, UK: John Wiley.

Walsham, G. (1995). The Emergence of Interpretivism in IS Research. *Information Systems Research, 6* (4), pp. 376-394.

Walsham, G. (2000). Globalization and IT: Agenda for Research. In R. Baskerville, J. Stage & J. I. DeGross (Eds.), *Organizational and Social Perspectives on Information Technology* (pp. 195-210). Boston: Kluwer Academic Publishers.

Weber, R. (2004). Editor's Comments - The Rhetoric of Positivism Versus Interpretivism: A Personal View. *MIS Quarterly, 28* (1), pp. iii-xii.

Wolcott, H. F. (1999). *Ethnography - A Way of Seeing* (1st ed.): AltaMira Press.

Yin, R. K. (2003). *Case Study Research: Design and Methods* (3rd ed.): Sage Publications.