# Software Process Improvement as Emergent Change: a Structurational Analysis

by

**Ian K. Allison**

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Industrial and Business Studies, Warwick Business School

University of Warwick, September 2004

# Table of Contents

## List of Figures

# List of Tables

## Dedication

This thesis is dedicated to Carol, Beth and Chris, three special people.

## Acknowledgements

The guidance, suggestions and observations of my supervisors, Ms Y. Merali and Professor J. Mingers are gratefully acknowledged. My sincere appreciation goes to them for their continued support throughout the length of this part-time study. Their suggestions and stimulation have inspired features of this study. I would also like to thank all those researchers I met at workshops, conferences and simply in passing whose willingness to listen to, question and confirm my ideas helped to shape my thinking along the way.

I thank the School of Computing and Informatics, Nottingham Trent University for their sponsorship and, specifically, Professors R. Whitrow and A. Hopgood for their faith in my ability to pursue this programme of study.

I appreciate the willingness of the case company to participate in the study and the interviewees for their time and their frankness, thereby providing the raw material for this study.

Finally, many thanks go to Carol who has worked tirelessly over the last few months reading and checking my original manuscript, and also for her encouragement, humour and support when I needed them most.

## Declaration

This work is entirely my own original work. The data is based on that gathered and analysed from the case site; all extracts and other sources have been appropriately attributed and referenced. This work has not been submitted for a higher degree elsewhere. All interpretations and suppositions are the sole responsibility of the author and do not in any way represent the views of the case company, its employees, or the University of Warwick.

One conference paper has been published from this research:

Allison, I. and Merali, Y (2003) Software Process Improvement: Towards an Emergent Perspective, In: Levy, M. Martin, A. and Schweighart, C. (eds.), Proceedings of the 8th UKAIS Conference, 9 – 11th April, University of Warwick.

## Abstract

This thesis differs from the technological perspective of SPI by identifying and analysing the organisational features of process improvement. A theoretical understanding is developed of how and why software process improvements occur and what are the consequences of the change process within a specific case. A packaged information systems organisation forms the basis for a substantive case study. Adding to the growing body of qualitative research, the study takes a critical hermeneutic perspective. In doing so it overcomes some of the criticisms of the interpretive studies especially the need for the research to be reflexive in nature.

By looking at SPI as an emergent rather than deterministic activity, the design and action of the change process are shown to be intertwined and shaped by their context. This understanding is based upon a structurational perspective that highlights how the process improvements are enabled and constrained by their context. The work builds on the recent recognition that the improvements can be understood from an organisational learning perspective. Fresh insights to the improvement process are developed by recognising the role of the individual to facilitate or resist the improvement. The understanding gained here can be applied by organisations to enable them to improve the effectiveness of their SPI programmes, and so improve the quality of their software. Lessons are derived that show how software organisations can support the ongoing improvement through recognition of the learning and political aspects of the change by adopting an agile approach to SPI.

## Abbreviations

| | |
|---|---|
| CMM | Capability Maturity Model |
| CASE | Computer-Aided Software Engineering |
| IDEAL | IDEAL is a continuous improvement model and is an acronym of the five phases: initiating, diagnosing, establishing, acting, and learning |
| IEEE | Institute of Electrical and Electronics Engineers |
| IS | Information Systems |
| ISO | International Standards Organisation |
| MAP | Market Analysis Package |
| OO | Object Oriented |
| SEI | Software Engineering Institute |
| SEPG | Software Engineering Process Group |
| SPECTRUM4 | Version 4 of the SPECTRUM product |
| SPI | Software Process Improvement |
| SPICE | Software Process Improvement and Capability dEtermination |
| TQM | Total Quality Management |

# Chapter 1. Introduction

## 1.1 Introduction

Information systems quality has long been a subject that has interested researchers because of its relevance to practitioners. Poor quality software ultimately loses companies money, making this an issue vital to the overall success of organisations. This is doubly so when, as in this study, software is the marketable product of the company. Despite previous predictions by information systems managers that the problems with the quality of software were about to be overcome (Galliers et al, 1994), these problems remain. Indeed, the information systems (IS) section of the software industry has one of the worst track records for failure (Jones, 1996). Developing quality software, therefore, remains an important issue and ranks high amongst the priorities of IS managers (Brancheau et al, 1996; Ravichandran and Rai, 2000a).

It is widely recognised that the quality of software is related to the software processes used to create the software. In an attempt to create a sustained approach to improving the quality of the software processes used by an organisation the practice of software process improvement was developed. Software process improvement (SPI) facilitates the identification and application of changes to the development and management activities in order to improve the product. Much of the current understanding of software process improvement has been derived from the work of the Software Engineering Institute (SEI). However, the majority of the work to date has concentrated on developing prescriptive models rather than understanding the reasons improvements occur and the factors that shape these improvements. Consequently, it is now acknowledged that too much attention is paid to the technological aspects of

software development at the expense of organisational and social factors. Subsequently, Ravichandran and Rai (2000a) criticise existing literature for the lack of systematic examination of the enabling and constraining roles of contextual factors.

This thesis, therefore, differs from the technological perspective of SPI by identifying and analysing the organisational features of process improvement. Unlike the software engineering literature that tends to be restricted to a rational, deterministic view of change, here the ongoing nature of the software processes is placed at the heart of the analysis: opening up the facets of the change not taking it as a given. To address this goal the key influences on a SPI initiative are examined as it is enacted, revealing a perspective on software process improvement based on organisational theory to highlight how the process of change occurs. By looking at SPI as an emergent rather than deterministic activity we can understand how the design and action of the change process are intertwined and shaped by their context. The work will build on the recent recognition that the improvements can be understood from an organisational learning perspective. Fresh insights to the improvement process will be developed by recognising the role of the individual to facilitate or resist the improvement. The understanding gained here can be applied by organisations to enable them to improve the effectiveness of their SPI programmes, and so improve the quality of their software.

## 1.2 Background

Since it was conceived over thirty years ago, the software engineering field has made significant progress towards improving the quality of software development. To date, the focus of software engineering has been 'the application of a systematic,

Organisations that follow these cyclic improvement models tend to document their current software processes and define the preferred processes. The definition and evaluation of new processes is seen to be an important part of these change models. However, whilst many organisations have documented their software process, there is often a mismatch between the defined methods and the processes actually performed. To research what improvements occur and how they happen, it is necessary to look beyond the software process documentation at the actual processes performed and changed.

Relatively few organisations have succeeded with SPI programmes to date. This is partly because disciplined software engineering is difficult and requires a change in attitudes and behaviour. A number of organisations have begun but failed in their attempt to implement SPI programmes because of human and organisational factors. However, there is a paucity of existing work that extends into understanding the way that these factors actually affect the process as it is enacted, and how they affect the success or failure of the improvements. This gap in the literature has resulted in two current developments in the SPI literature. One attempt to understand the changes has drawn on the Diffusion of Innovation theory. Diffusion is seen to occur through the dissemination of an innovation as it is communicated and transferred through a social system. These studies consider that the diffusion concept helps to understand how ideas spread through the industry or organisation, but others have shown that these changes are more complex than the diffusion theory specifies. It therefore does not address the dynamic relationship between those involved.

Another new strand of the literature has begun to show that to understand how process improvements occur we need to encapsulate the way that organisations learn to improve over time. How organisations learn is not fully understood, with the relationship between individual learning and organisational learning widely debated. In this thesis it is argued that we need to look towards an active view of knowing and learning. An action based view of knowledge leads us into seeing SPI as a process of sensemaking, where agents are seen to draw on their knowledge during the action, and learn through reflection on their experience. Thus, if we are to understand how continuous change occurs we need to take into account the organisational context it is occurring in, the actions of the various actors and how these interact to enable the learning and knowledge creation said to be required to improve the process. This thesis adds to this growing body of work by drawing on social theory to analyse the process as emergent change.

## 1.3 Research objectives and methodology

The IS field has seen a change of focus from technological issues to managerial issues, resulting in more interest in the interaction between the context and innovations (Benbasat et al, 1987; Kautz and Larsen, 2000; Avison et al, 2001). However, as discussed above, the research into software process improvement initiatives has been predominantly deterministic, with minimal work focusing on the relationship between contextual factors and the efficacy of the improvement. Lehman (1997, p.550) suggests that 'the essential lesson to be derived from current improvement approaches is that one must develop a global view and a comprehensive insight as to how, through their processes, organisations achieve and maintain quality products'. Jarvine (1994, p. 260) argues that to develop a good approach to

improvement we need to obtain 'additional coverage of the complex world of software production as the software process is only part of it'. Similarly, Perry et al (1994) show that without understanding the technological, social and organisational aspects of software development we cannot hope to significantly improve processes. So, in response to these criticisms of the traditional SPI literature, work has begun towards understanding the way that human and organisational factors affect the software processes as they are enacted and changed.

Work in organisational studies, and the broader information systems literature, shows that the factors that shape process improvement activity are contextual and behavioural. The consideration of factors that shape the introduction of specific technologies has shown that software process and product variations emerge from particular interactions of institutional context, key players' intentions and actions, and the new technology or method (Orlikowski, 1993). It is therefore appropriate to examine whether organisational issues arise as software development groups move towards a more structured, process-oriented environment.

The aim of this thesis is to develop a critical, explanatory theory of the SPI change process based on the experiences of a specific IS group. The theoretical development critically reflects previous findings in the study of organisational change. A theoretical understanding of the change is developed that goes beyond the software engineering basis for SPI. A theoretical understanding is developed of how and why software process improvements occur and what are the consequences of the change process within a specific case. This understanding is based upon a structurational perspective that highlights how the process improvements are enabled and constrained

by their context. In summary the main question of this study, with related sub-questions, is:

*How and why do organisational and social factors shape the adoption of software process innovations within a packaged software organisation?*

*a) How does the software process improvement initiative unfold within the context of a packaged software organisation, and how does this compare to their stated intent?*

*b) What are the critical influences on the software process improvement activity as it is enacted? How and why do these influences enable / constrain changes to the processes?*

*c) In relation to adopting process innovations, how and why do the behaviours of individuals affect the dynamics of the software practice?*

A longitudinal case study is undertaken to investigate these issues. Case study approaches are well suited to empirically investigating such issues and understanding the interactions with the context (Hartley, 1994; Darke et al, 1998), as knowledge is captured from practitioners to enable theory development that is relevant to other organisations (Benbasat et al, 1987). A particular instance of SPI initiatives will be investigated to observe the process of change within its context. This case was designed following Pettigrew's (1990b) ideas on processual analysis and Eisenhardt's (1989) ideas on developing theory from case study research. SPI is an on-going exercise, so to see how the situation develops is an essential aspect of this study. Longitudinal case studies are able to show how the contextual relationships develop and evolve over time and not just at one point in time. This approach helps to develop

research that is theory-based and context-rich. The quality of the access enabled an in-depth study that provided a rich view of the software practice located within a specific, situated context. Here the research builds on existing research not in a theory-led fashion, but by 'enfolding' the literature in during the analysis.

A dialectical hermeneutic perspective is adopted as it enables a critical, explanatory theory to be developed. One advantage of this approach is that it helps to portray the complexities of the process of change as it helps to understand the influences, both realised and hidden, and the consequences of any action, both intentional and unintentional. These influences and consequences of actions are not always evident to the actors themselves and are identified through the use of multiple sources and observations. This view is important in developing perspective of the events that goes beyond the views of the actors, thus highlighting the structurational aspects of the change.

Rather than random sampling, the case was deliberately chosen for properties relevant to the study. The case organisation, InfoServ (a pseudonym), are a multi-national, information services company. The process improvement initiative was in a software development unit within one of the UK divisions. The study focuses on the activities of the software development unit for the period from the start of the introduction of a formalised process definition in 1995, through the initiation of the software process improvement initiative in 1997 and up until the process improvement activity was evaluated in 1999. This period of study was selected to capture the historical perspective of the events as they unfold, and to see the patterns of change as new process and product innovations were encountered. One year was spent as a

participant-observer to examine in detail the actions, intentions and perceptions of the human actors, the context in which these actions take place and the consequences of the actions. Walsham (1993, p.248) states that 'this style of empirical research is appropriate for the view of the nature of knowledge embedded in a broadly interpretive philosophy which emphasises the need for detailed understanding of human beings in context'.

## 1.4 Significance of the study

The thesis addresses four areas of information systems theory and practice. The primary contribution of the work is to reveal the nature of the SPI activity in the case study. The literature suggests that the purely methods driven perspective is insufficient for understanding SPI and the associated problems. This work extends the existing literature by investigating the effect of contextual and social factors on the changes in software processes as they are enacted. It adds to this work by showing such initiatives to be a form of situated change. By recognising that software practice is a social activity it is theorised that process improvement emerges from the rich interplay of actions, not simply the intentional adoption of new methods. The social process of the change is central to the analysis. The organisational theory literature is used to show how an emergent view of the change and software process can help to understand the way the actions intertwine to inform each other, and shape and are shaped by the context they are in. This work adds to the growing body of recent work investigating the social aspects of software process improvement initiatives. The case study highlights the way in which the changes emerge and develop through time, often differing to intended actions or from the way in which the literature suggests the process improvement initiative or software engineering techniques should be

instigated. The software engineering and information systems management literatures and practice are informed through a critique of this case study. The learning and political aspects of the case offer particular insights to the nature of the change.

Secondly, from these insights, lessons are derived to inform future SPI practice. The suggestions discussed include a view of how organisations can support the ongoing improvement through recognition of the learning and political aspects of the change. A provisional model is proposed to inform the management of these issues that reflects the understanding of SPI as an emergent activity.

Additionally, the case is a packaged software organisation. The packaged software market is a considerable global industry. At the beginning of the study, this market had sales in excess of $72 billion and over 2 million people employed in these organisations in the US alone (Carmel and Sawyer, 1998). Yet, much of the existing literature focuses upon the development of software within internal IS departments or the producers of technical and systems software. This study, therefore, adds to the theoretical understanding of the creation of IS packaged software.

Finally, this study adds to a growing body of qualitative work in IS development. The strength of this approach is now widely acknowledged, but still only a minority of the work in the IS field adopt this approach. Specifically, the research takes a critical hermeneutic perspective, building on the work of Myers (1994a), to overcome some of the criticisms of the interpretive studies. The research methodology adopted adds to our understanding of Myers' work as it demonstrates how the techniques used can implement the general principles that he outlines, especially the need for the research

to be reflexive in nature, to challenge preconceived ideas and bias, and the need to look for contrasts in the data not simply confirmatory findings.

## 1.5 Outline of the thesis

The remainder of the thesis is organised into six chapters. The initial chapters develop the justification for the research topic and method. Chapter 2 critiques the development of the software process improvement approaches from an engineering perspective. By highlighting the shift in the literature towards understanding the activity of improving, the chapter draws on an understanding of SPI as learning and innovation. Through this discussion, it is shown that the process of improvement can be understood as emergent change, with the improvements occurring through a process of sensemaking and improvisation. Based on this understanding of SPI, Chapter 3 shows why it is important to investigate this phenomena by observing the events as they occurred, using a longitudinal case study. The case study approach is detailed, showing that a combination of critical hermeneutics and Structuration Theory are key to developing an explanatory theory.

As the case study covers events over period of time, Yin (1994) argues that one approach to present the data is in the chronological order. The chronological approach is particularly relevant here so as to draw out the emergent nature of the findings. This approach serves an important purpose in explanatory case studies because the patterns can be seen to occur over time. The sequence of both Chapter 4 and Chapter 5 – the what and how of the study – are organised into early, middle and later stages of the case. The case study approach has allowed the capture of in-depth data. Chapter 4 describes the case study events as they unfolded. The description of events, with

interpretations, as they took place in a specific organisation is an effective means of communicating the contributions of the research to practice (Benbasat and Zmud, 1999). Chapter 5 develops an analytical framework based on an emergent view of change and underpinned by Structuration Theory. This framework helps to communicate the findings to other settings. The process innovations are analysed through time drawing on the framework to explain the reasons for the changes - the why of the study.

The emergent nature of the change is shown to be more than a random set of outcomes from a period of history. The intent to change informs and shapes the emergence of the processes. The emergence can be seen both for individual software processes and for the organisational context, as each enables and constrains the other. Two features of this emergence were revealed by analysing further the learning and political aspects of the case that occurred throughout the whole period. Understanding the emergence through these concepts is an essential part of developing the implications for software process improvement practice and theory.

From this discussion suggestions are made for the management of future SPI projects in Chapter 6. The model developed, however, is not another prescriptive model to be followed in a step by step approach, but shows the features that IS management need to consider to support their improvement initiatives. In the conclusions, lessons and findings are drawn together so that they can be applied more generally in other cases. The research approach adopted is critiqued so as to inform the research methods literature. Suggestions are made for future research of process improvement. Three proposals are made: one in the area of continuing the study of software package

organisations, one in exploring further the political aspects of SPI, and one in the relationship between education and practice of SPI.

# Chapter 2. Literature Review

## 2.1 Introduction

The chapter begins by providing a critical review of the manufacturing perspective of quality management and how SPI models have been derived from this thinking. The focus of the SPI literature has moved towards continuous approaches for improvement as a means of managing the introduction of these models and encouraging ongoing learning. The recent literature that highlights SPI as an ongoing learning and innovation activity is explored to show how they inform our understanding of the SPI activity. Current perspectives of change and process improvement provide new ways of understanding how the software processes change over time. Drawing from these different views, it will be shown that we can understand process improvement as an emergent intertwining of improvised design and action rather than a pre-planned, predetermined activity. The chapter will show that this view enables a fresh understanding of how and why software processes change.

## 2.2 Software Quality

With the importance of quality in the software industry, more organisations are emphasising customer satisfaction and product reliability as well as delivery on time. One of the problems is that quality is not absolute or objective in nature, making it difficult to define and measure. Following the work of Crosby (1979) and Juran (1979) in manufacturing, the quality of software is often defined in terms of its conformance to customer requirements or fitness for purpose. This emphasis has dominated quality management thinking. The rhetoric has continued with the formation of the term software factory, engendering the view that software development can be automated. By treating software development as a production line

the emphasis has been placed on the introduction of techniques to improve the quality from the outset of the process, not simply by post-development testing, and the introduction of computer-aided software engineering tools.

Specific techniques and measures have been developed to assist in the management of quality and early identification of problems in the software production activity. Software inspections (Fagan, 1986), mathematical approaches for specification and reliability (e.g. Dyer and Kouchakdjian, 1990; Mills et al, 1987) and software metrics (e.g. Fenton, 1991) help to review and control the development output as it becomes available. This encourages the early identification and correction of defects. Product and process attributes help to monitor progress and to indicate areas for future improvements in the process.

The introduction of additional techniques to improve quality initially appears to be an overhead cost, but Crosby (1979) argues that the real cost is in failure. The costs of failure are incurred in four principal areas: cost of correcting defects; overruns in time and budget; high maintenance; indirect user costs. In addition to these costs, there are the less quantifiable items of staff dissatisfaction, formation of low expectations for productivity, consequent difficulties in acceptance of new procedures, and the failure to use current experiences to learn for future projects (Frewin and Hatton, 1986). By comparing the cost of failure with the cost of investing in the prevention of defects through planned activities, the overall costs of development can be reduced (Wallmüller, 1991). By removing defects throughout the process, and not after the product is complete, the cost of defect correction is reduced and the quality of the product is increased. Moller (1991) shows that 39 percent of the costs of a software

project are in fault correction. The earlier a fault is found the cheaper it is to correct but it is better, of course, to avoid faults altogether.

The objective of quality management, therefore, is the prevention of errors, the detection and eradication of errors that occur, and the improvement of processes, procedures and tools to make the development more effective (Frewin and Hatton, 1986). Quality management concepts in the IS field are derived from the work of total quality management (TQM) in manufacturing. TQM focuses on customer satisfaction, a company-wide quality culture and continuous quality improvement. The principle behind TQM is for all members of the organisation to improve their work on a regular basis. TQM is now more widely understood, accepted and applied by software developers (Fox and Frakes, 1997).

ISO 9000 is a family of international standards that has been widely adopted for total quality management. ISO 9001 is particularly targeted at organisations that design and produce software products. This standard is a generic model of a quality process, describing the processes and standards that should exist within the organisation. To meet the standard, each organisation needs to create a quality manual defining a set of quality procedures that is used to create a specific quality plan for each project. The quality plan should describe how the organisational processes and standards will be applied for the particular project. The ISO 9000-3 guide interprets ISO 9000 for software development. The Department of Trade and Industry also developed the TickIT scheme to help software organisations to implement quality systems. TickIT was aligned to the standard but added its own clauses. It is outside of the scope of this thesis to discuss ISO 9000 or TickIT in any detail (Hall (1995) provides a

comprehensive explanation of these approaches). Increasingly, ISO 9001 certification is a pre-requisite for customers purchasing software products. The standard has been helpful in setting up and verifying quality management systems in order to improve customer satisfaction (Rigby et al, 1990). However, improvement beyond the quality management system was not specifically covered by the original ISO 9000 standards. Avison et al (1994) found that although TickIT had enhanced the general quality standard, ISO 9001 is still limited because it is a general manufacturing standard.

Another criticism of the manufacturing perspective is that it encourages the view that quality is simply characterised by fitness for purpose. Fitness for purpose is judged by conformance to a specification proven through inspection and testing. This is a limited perspective of quality, especially in a turbulent business environment. A broader view of software quality is the 'totality of features and characteristics of a product or service which bear on its ability to satisfy a given need' (Frewin and Hatton, 1986, p.29) as defined by the British Computer Society and IEEE, suggesting that there are other factors, such as maintainability and reusability, that are also important for overall quality.

In an attempt to improve these other features a process oriented view has been adopted by many, where the processes used to produce the software are defined and managed so that variability is reduced. The software process can be seen to be the set of activities, methods, and practices used in the production and evolution of software (Humphrey, 1989). The standard software life-cycle models focus on development activities and the output generated from those activities. The software process is wider than just the development activity and includes the planning and control of those

activities, the management of the people involved in undertaking them, the tools available in support and the standards used to undertake the work. With numerous approaches available, practitioners have adopted a wide variety of models for the process of software development.

The Software Engineering Institute (SEI) states that 'the quality of software is largely determined by the quality of the software development and maintenance processes used to build it' (Paulk et al, 1995, p.8). So, the application of a consistent process within an organisation is believed to help improve productivity by reducing the variability between projects. The aim of software quality management is therefore to create a stable and reliable software process based on rigorous project management and process control. However, which process model is adopted is dependent on the organisational context of the development and its application in each project is contingent on the problem being addressed and the variability of the application by the project team. Whichever process models are adopted software process improvement, in its broadest sense, involves the identification and implementation of changes to the software development process. Once more it is anticipated that because of the work in software process improvement 'we have turned the corner of the software crisis' (Paulk, 1998, p.xv).

## 2.3 Software Process Improvement Methods

This section will discuss the current software process improvement literature beginning with the normative maturity models used to assess and improve the processes within organisations, and then how the process improvement activity is

understood as a continuous improvement and learning activity. This will be used to lead into the development of an understanding of SPI as situated, emergent change

### 2.3.1 Normative Software Process Maturity Models

To support improvement programmes, the software engineering community have developed a set of normative maturity models for organisations to follow and enable the assessment of current capability. These models are being adopted by organisations and anecdotal evidence shows that benefits are achieved as a result of this adoption, but the models are criticised for the rigidity of the predefined actions and the deterministic nature in which they need to be implemented. The normative models will be explained and critically analysed below.

Improvement in the software process is considered to result in the *maturing* of the activities undertaken by a software development group (Humphrey, 1989). A mature organisation is one that applies appropriate software engineering techniques, measures its activities and is able to identify and control changes to those processes using the data captured. Maturity models have been formulated to address the problems of using the appropriate techniques in managing software development.

The maturity models and the ISO 9001 quality standard are similar in many respects (Hailey, 1998; Paulk, 1995). This similarity often means that organisations with ISO 9001 accreditation are assessed at above the base levels of the different maturity models (Paulk, 1993; Jung et al, 2001).The quality standard defines the minimum criteria for a quality management system, with the emphasis on documented processes (how to do it) and records of product quality (what was achieved). ISO 9001 is not, however, a process model or process improvement framework and offers little support

for improving beyond the lower levels of software process maturity. So, the biggest difference between the quality management standard and the maturity models is their emphasis. The models add to ISO 9001 by focusing on implementing revised practices that are in line with the higher maturity levels, drawn from the software engineering literature rather than generic quality management. Three capability determination approaches are reviewed below: Capability Maturity Model (CMM), the BOOTSTRAP Model, and the more recent ISO 15504 (SPICE) standard. The evaluation will establish their strengths and the concerns expressed in the literature.

In the mid-1980s the SEI began to develop a process maturity framework to enable organisations to assess and improve their software process. The original use was the evaluation of the capability of subcontractors for the US Department of Defense. This framework was evolved into the Capability Maturity Model (CMM). Zahran (1994, p.223) describes the CMM as a 'common-sense engineering approach to software process improvement'. It is intended to assist in the improvement of the software process maturity through an evolutionary path. The CMM describes principles and practices that support software process maturity. The model is organised into five maturity levels. The lower levels show how to progress from a chaotic approach to the application of predefined processes for a controlled output. The higher levels involve optimising the management of the processes. Each level is a well-defined stage towards the goal of achieving a mature software process containing a number of key process areas. A key process area is a group of related activities that collectively achieve the goals set for that level. The process capability is institutionalised when the goals of key process areas are continually achieved in different projects. The five levels are:

*Level 1 Initial*: at this level the software process is ad-hoc and chaotic. Success can be achieved, but it relies upon the heroic effort of the developer. Schedule overruns are common and few processes are defined. In Humphrey (1989)'s seminal book *Managing the Software Process*, he gives an example of a level one organisation who had just a 'small amount' of programming to add to a system before completing, that turned out to be an extra 250,000 lines. The company had thought the change so simple no one had bothered to plan it. Even though the technical work was good, the customer refused to pay for much of the work.

*Level 2 Repeatable*: by this stage the basic project management activities are undertaken thereby achieving more reliable project schedules. The planning, estimating and control processes are defined.

*Level 3 Defined*: The processes for managing and engineering the software product are documented. These defined processes represent the standard activities undertaken across the organisation. Tailoring of these standards by project is predefined and approved. The application of software engineering tools and techniques are introduced at this stage.

*Level 4 Managed*: the managed level captures detailed measures of the software processes and product. The data is used to control the management of the development and product quality. Significant improvements in quality are often achieved at this level.

*Level 5 Optimising*: By this stage the organisation is undertaking continuous improvement using the measures for identifying changes. The management of improvement is now focused on optimising the process.

The CMM became the *de facto* standard for process maturity models. As discussed later, in section 2.3.2, many companies have reported benefits from using CMM. Others felt that it was too rigid and that the genuine maturity level cannot be indicated by a single number. So following the pioneering work of the SEI, a number of alternative approaches were developed.

Of the alternative models, the most widely adopted in Europe was BOOTSTRAP. A multi-national consortium of industrial companies and research institutes developed BOOTSTRAP in an ESPRIT project funded by the European Community. This model combines the features of both the ISO 9000 and the CMM, and extends these approaches to include the European Space Agency's standards (Kuvaja and Bicego, 1994). The processes included in the BOOTSTRAP 3.0 model, released in September 1997, are shown in Figure 1. The major differences with the CMM are found in the organisation, technology and life-cycle function boxes, where new features are incorporated. Version 3.0 of the model was updated to align with the ISO 12207 life-cycle and ISO/IEC 15504 (SPICE) reference model requirements (Kuvaja, 1999).

BOOTSTRAP addresses some of the criticisms about the CMM through a more detailed and transitional assessment of the maturity level. By not adhering to a maturity model like the CMM, with distinct key practices at each level, BOOTSTRAP allows alternative development approaches to be accommodated. BOOTSTRAP is

therefore 'suitable for all kinds and sizes of software development organizations' (Stienen, 1999, p.59). It takes into account the typical size and cultural differences across organisations and countries. This flexible and constructive approach is a key factor in BOOTSTRAP's uptake.

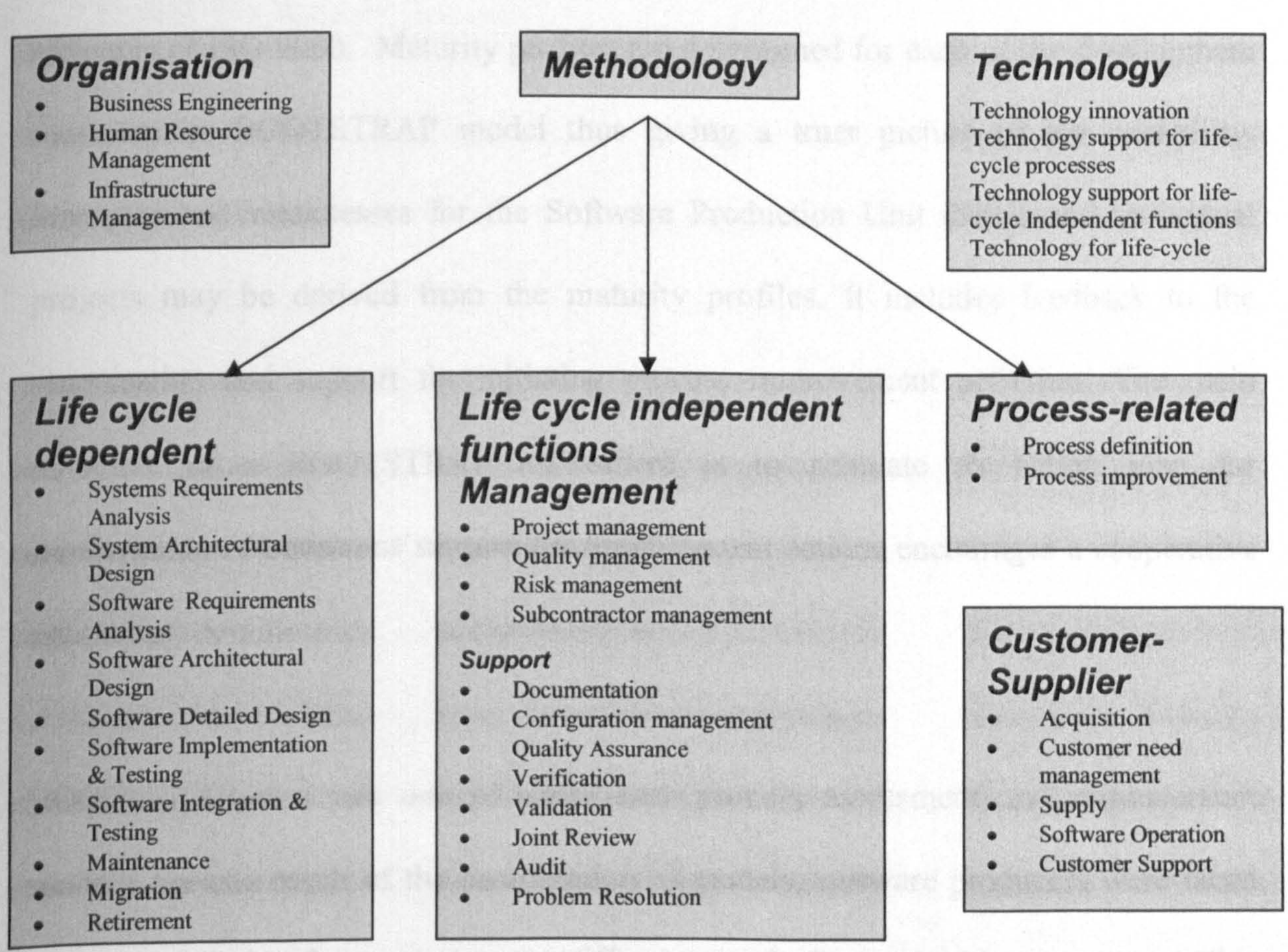



**Figure 1 BOOTSTRAP model**

BOOTSTRAP provides detailed maturity levels and capability profiles of organisations and projects. Early versions of BOOTSTRAP used the same five maturity levels as the CMM, but has now adopted the six levels from ISO/IEC 15504 (see below). The assessment, like CMM, uses questionnaires and interviews to gather data from the company. Two questionnaires are used, one at the organisational level and one at the project level, so that the variations across the company can be

identified. The results are used to guide the interviews. The main difference to CMM is that the answers are scaled rather than a simple yes or no (Kuvaja and Bicego, 1994). When the results of an assessment are presented, each level is sub-divided into quartiles to provide a more detailed assessment and to allow improvement to be monitored. It optionally allows the determination of the degree of fulfilment of the 20 elements of ISO 9000. Maturity profiles are determined for each of the development areas in the BOOTSTRAP model thus giving a truer picture of the capability. Strengths and weaknesses for the Software Production Unit (SPU) and individual projects may be derived from the maturity profiles. It includes feedback to the organisation and support for initiating process improvement activities. The main objective of a BOOTSTRAP assessment is to generate an action plan for improvement. Continuous support for improvement actions encourages a cooperative approach.

BOOTSTRAP was just one of many new process assessment and improvement models. So as a result of the proliferation of models, software producers were faced with a multitude of assessments for different standards required by customers. This proliferation created the impetus for a common approach to process assessment to be developed. After a study report called Improve-IT in the UK, the ISO/IEC committee in charge of developing software engineering standards established a working group (known as WG10) to develop an international standard for software process assessment. The Software Process Improvement and Capability dEtermination (SPICE) project was initiated. ISO/IEC 15504 resulted from the SPICE project. As it stands today, ISO/IEC 15504 can be divided into two categories (Drouin, 1999):

- Normative : parts 2 and 3 must be adhered to for an organisation to claim that it has conducted an ISO/IEC 15504 conformant assessment.
- Informative: parts 1 and 4 to 9 inclusive provide guidance on aspects of process assessment, supplier capability determination, etc.

| The Capability Maturity Model | ISO/IEC 15504 |
|---|---|
| **1. Initial**<br>*Ad hoc and undefined, depends on the individual*<br>**2. Repeatable**<br>*Formal project management methods used to control the process*<br>**3. Defined**<br>*A defined software engineering process utilising good engineering practice*<br>**4.Managed**<br>*Process metrics used to inform improvement*<br>**5.Optimising**<br>*Continuous process improvement integral to the organisation informed by the quantified data* | **0.Not performed**<br>*Failure to perform a process*<br>**1.Performed informally**<br>*Not planned, depends on individual knowledge*<br>**2.Planned and tracked**<br>*Verified process conforming to specified standards*<br>**3.Well-defined**<br>*Process well-defined and uses approved, tailored versions of standard approaches*<br>**4.Quantitatively controlled**<br>*Detailed performance measures used to evaluate process*<br>**5.Continuously improving**<br>*Continuous improvement using metrics to drive the changes* |

**Table 1 Process Maturity Levels**

The standard contains a reference model composed of nine generic attributes. These are used to characterise the capability of any given process and are grouped into six capability levels (from 0 to 5) (see Table 1). The purpose of the reference model is to provide a common basis for different models and methods for software process assessment, allowing a common approach to reporting.

As well as processes, the standard is concerned with 'people, technology, management practices, customer support and quality as well as software development practices' (Dorling, 1993, p.404). The standard can be used in capability determination mode to help in the assessment of a potential supplier, in process improvement mode to improve internal software processes, and in self-assessment mode to help an organisation determine its own capability.

In summary, after the innovative work at the SEI there are many different maturity models in use today. These models are used both to assess current capability levels and to set the priorities for improvements. An international standard has recently been produced to unify the reporting of findings from assessments by different process maturity models. These models have played an important role in raising the awareness and supporting the implementation of software engineering techniques to produce a stable, reliable software process. The next section shows the limitations and drawbacks that the models have.

### 2.3.2 A Critique of Normative Process Maturity Models

A number of case studies have been published on CMM programmes (eg Hughes Aircraft (Humphrey et al, 1991); Raytheon (Dion, 1993); PRC (Hollenbach at al, 1997); Motorola (Diaz and Sligo, 1997); Tata Consultancy Services (Keeni, 2000)).

These studies describe how specific organisations have applied CMM or similar models for improving the maturity of their internal processes. Each study shows impressive business benefits in key performance areas and in measures of software quality (see Table 2).

| Benefit achieved | Hughes Aircraft | Raytheon | PRC | Motorola | Tata Consultancy |
|---|---|---|---|---|---|
| Reduction in defect density / improved product quality | ✔ | ✔ | ✔ | ✔ | ✔ |
| Improved team spirit / less staff turnover | ✔ | ✔ | | | |
| Increased productivity / reduced time to market | ✔ | ✔ | ✔ | ✔ | |
| Reduction in cost of rework / testing | ✔ | ✔ | ✔ | ✔ | ✔ |
| Improvement in meeting schedules / accuracy in predicted cost | ✔ | ✔ | ✔ | ✔ | ✔ |
| Return on investment in SPI | | ✔ | | ✔ | |

**Table 2 Key business benefits reported**

Whilst these cases give positive results from their SPI programmes, the evaluation was limited to a few self-selecting, anecdotal examples. In an attempt to address the paucity of evaluation data, Herbselb et al (1997) gathered data from 13 case studies

and undertook a survey of 138 individuals in organisations involved in CMM-based process improvement. The case study data showed productivity gains averaging 35 percent per annum and a return on investment for the SPI activity of 5:1 on average. The survey results showed that as organisations progressed through the maturity levels they perceived an improved performance in a number of key areas: product quality, customer satisfaction, productivity, ability to meet schedules, ability to meet budgets and staff morale. These results are supported by Jones (1999a) who estimates that there is a seventy-five fold decrease in the relative number of software defects between organisations at level 1 and level 5 of the CMM. Gray and Smith (1998) though question the economics and cost-effectiveness of process improvement as the picture is incomplete with 2 out of 3 cases of SPI failing. The perceived flaws in the models as concepts and the problems encountered in trying to implement the models are highlighted below.

One of the reasons for the high non-completion rate is that reaching the higher levels of the maturity models takes time. A survey in 1995 found that the vast majority (81 percent) of organisations had remained at level 1 with no organisations achieving levels 4 or 5 (Goldenson and Herbselb, 1995). By 2003 over 200 organisations had reached levels 4 and 5, and only 17 percent of assessed organisations had remained at level 1 (Software Engineering Institute, 2003). Results show that after five years of BOOTSTRAP the levels of maturity were similar to early surveys of the CMM, with only 10 percent of organisations reaching an overall level 3 (Stienen, 1999). A later study of 50 Danish companies who had used BOOTSTRAP found that the highest organisational maturity level was 3.25, but individual projects had reached level 5 (Pries-Heje et al, 2001). These companies had shown an improvement in their

maturity levels between two sets of assessments indicating a continued adherence to the maturity model. The constraint in moving to the higher levels was the need to obtain organisational consensus for areas such as standards. Persistence and dedication from senior management are required if the effects of the programme are to be long-lasting.

One of the problems in just looking at the grades achieved in the assessments is that the capability of any organisation is a far more complex picture than that shown by any number. The results from the SPICE trials show that organisations are able to partially achieve aspects of the different process areas at each of the maturity levels (Jung et al, 2001). The organisation's genuine level of maturity, therefore, is not clear from the simple maturity grading (Bollinger and McGowan, 1991; Zhiying, 2003). The assessment process elicits and provides additional detail, but other agencies focus on the numerical result. In addition to this one dimensional grading, the actual levels are criticised as being too general, inappropriate and ill-conceived. The CMM grading system is thought by some to be 'seriously and fundamentally flawed' (Bollinger and McGowan, 1991, p.41), as the 'lack of theory informing the conceptualization of the stages raises questions about the rationale for the suggested sequencing to develop process capabilities' (Ravichandran and Rai, 2000a, p.384). This simplification of an organisation's level of capability is, as already noted, partially addressed by BOOTSTRAP and other models with a more flexible framework. Gray and Smith (1998), however, criticise any normative maturity models as being synthetic in nature as they are based on an ideal organisation that never existed. Critics of process maturity models suggest that whilst they represent a good initial attempt to improve process capability and provide a useful reference point that they should not be used

prescriptively. Organisations, however, tend to try and adopt the key process areas in a mechanistic fashion and consequently have problems following the recommended sequence (Card, 1991).

Consequently, Curtis (2000) states that although many software development organisations learn how to implement successful process improvement programmes others struggle through without making lasting improvements. This lack of progress is evidenced in a survey by Herbselb et al (1997) where nearly half the organisations were disillusioned by the problems encountered and the lack of progress. They found that the SPI activity had been overcome by events or crises in 42 percent of organisations and 72 percent had suffered due to time and resource limitations. It is often stated that SPI programmes are not a quick fix, but this is not necessarily the perception of software managers setting out on a new SPI programme. Maintaining momentum and visibility of the programme can be difficult once the initial enthusiasm wears off (Hollenbach et al, 1997). Thus, relatively few organisations have pursued SPI programmes to date because disciplined software engineering is difficult and requires a change in attitudes and behaviour (Paulk, 1998), not just a simple change of methods.

So whilst there are a number of spectacular results from some case studies the business benefit of SPI in general is questioned, even when apparent progress is made through the maturity levels. The focus on improving the process emphasises the internal activity over the external competitive position. The primary reasons for organisations performing an assessment using SPICE (ISO 15504) reflect this internal focus, with little importance being placed upon the external perception or market

benefits (Jung et al. 2001). In the SPICE trials approximately 60 percent did not observe a major impact on the organisation (reported in Conradi and Fuggetta, 2002). The concern is that the process has become more important than the quality of the product or any resultant business benefit.

This internal focus is an untenable position for commercial software producers, where reacting to the market is seen as more important than following a prescribed process. Software production is operating in a turbulent technical and commercial environment, where the competition and technology are fast changing, so there is uncertainty about the software products (Mellis, 2001). Despite the documented benefits some leading commercial software producers do not, therefore, follow software process improvement according to the maturity models or quality standards (Conradi and Fuggetta, 2002). This is probably because 'modelling and formalism have not solved the problems' in software quality (Lehman, 1998, p.42). The aim of the process oriented paradigm is a stable and reliable software process, as a result of software process improvement based on rigorous project management and process control. The idea of a stable, repeatable software process contrasts with the turbulent environment of commercial software product development. Microsoft have found the need to balance autonomy for development teams within an overall framework that enables the synchronisation and stabilisation of the software (Cusumano, 1997; Cusumano and Selby, 1997). Fast, innovative development is required compared with rigorous and slow, which makes stability a risk to competitive potential. So, 'it is from the vantage point of innovator that CMM seems most lost' (Bach, 1994, p.16), as Card (1991, p.103) puts it: 'maturity is necessary but not sufficient for true

improvement', as the real prize is increased competitiveness not a better maturity assessment score.

It is well known that contextual factors are important in a variety of aspects of information systems development (e.g. Doherty and King, 1998; Lee et al, 1995). We need to recognise that the interaction with contextual factors also affects the way in which the SPI initiative is undertaken and its eventual success or failure. Edgar-Nevill (1994) concludes that 'no matter how good the process model or how skilled the technical specialist, it is the management in all its aspects which gives the key to success'. Yet, SPI research does not pay enough attention to the organisational factors that enable or constrain the process improvements (Ravichandran and Rai, 2000a) and has only recently begun to see these factors as important. Perry et al (1994) believe that a holistic approach is necessary to study all the relevant factors in a given software context. The results of their study of developers showed that elements of organisation and interpersonal behaviour were just as important, if not more so, than technology.

In summary, the maturity models are seen as a useful source of informative advice. Despite the successes reported, the existing maturity models have been criticised in the literature. So whilst they have become a generally accepted order of priority for implementing the software engineering techniques, the principles of the models are not developed on any strong theoretical basis and are not applicable to all organisations. The application of the models are criticised, also, for their inflexibility and the emphasis of technology rather than people (McGuire, 1996). Advocates of prescriptive process-oriented approaches to software quality often discuss them as

though they occur in a vacuum, but as shown above, organisational issues are very much part of SPI. The CMM 'only touches on other, nonprocess (sic.) factors ... although these issues are crucial to a project's success' (Paulk et al, 1995, p.13). Subsequently, not all companies have found software process improvement to be beneficial with many abandoning SPI programmes.

In order to manage the software development process within the dynamic technical and business environment, the software industry must continuously evolve and improve its software development practices. Recent research and practice have recognised that software process improvement needs a long-term view and 'must be considered a repeating cycle' (Curtis and Paulk, 1993, p.386). From this perspective many authors have argued for a continuous software process improvement process (e.g. Briand et al, 1999; Birk and Rombach, 2001). The next section discusses the approaches for the continuous improvement of software processes, showing how these support the normative models. It will show that the primary emphasis in the literature has been on developing frameworks to support the 'how to' of continuous improvement. This literature has developed alongside, and in support of, the deterministic view that the normative models bring.

### 2.3.3 Continuous Improvement Methods

Maintaining an ability to react to the environment whilst achieving a quality standard means embarking on a programme of continuous change. Here the approaches suggested for continuously improving processes will be explored with a view to understanding SPI as a process rather than an outcome. The research in this area has recognised the difficulties of managing the introduction of new processes but provides little understanding of these difficulties. Whilst the literature helps to understand the

need to manage the ongoing changes these continuous improvement models remain deterministic in nature.

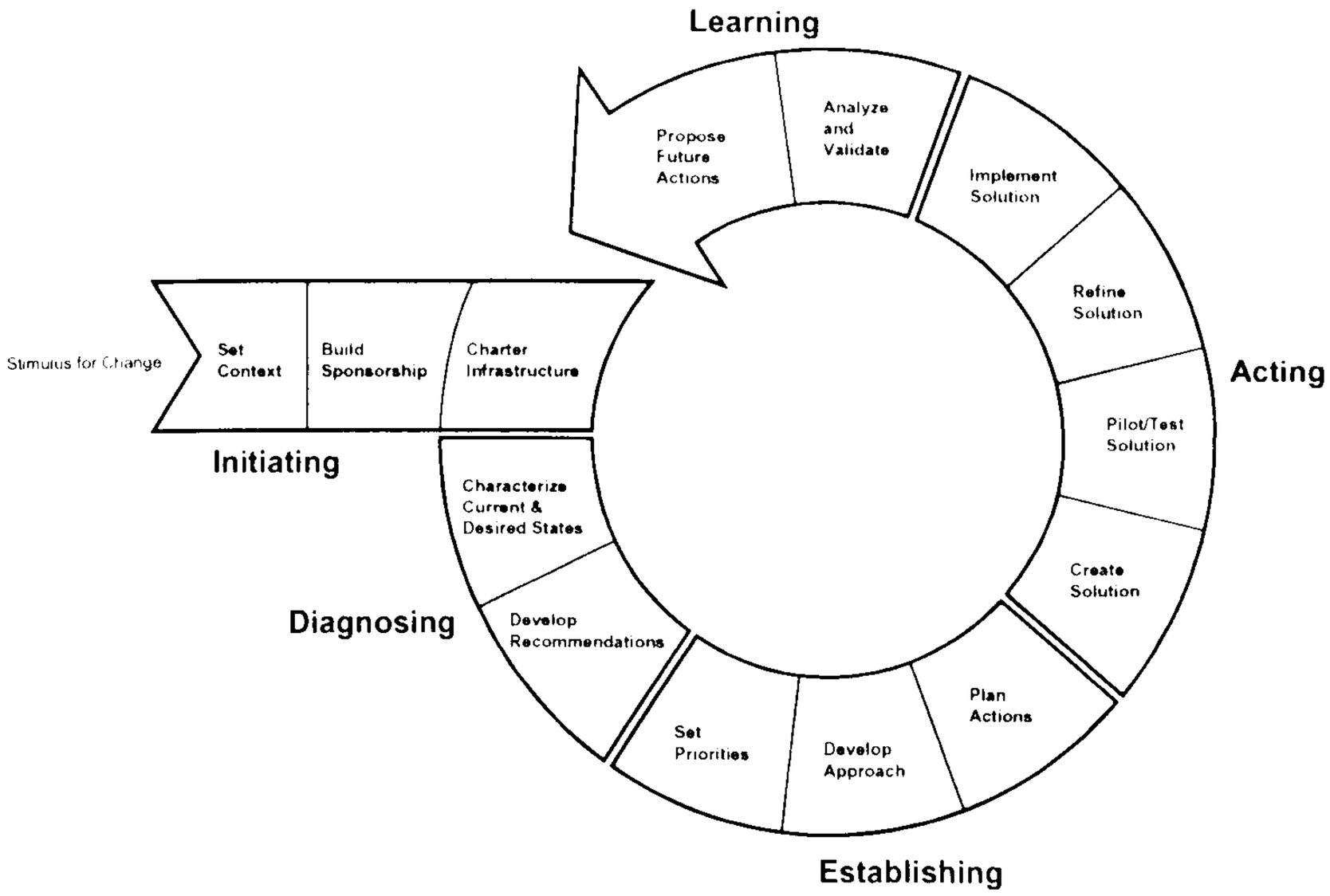



**Figure 2 IDEAL SM model (source: Software Engineering Institute)**

The SEI created the IDEAL model as an iterative model of SPI (see Figure 2) to complement the CMM (McFeeley, 1996). ISO 15504 has an in-built continuous improvement approach that is very similar to the IDEAL model. IDEAL is an acronym of the five phases: initiating, diagnosing, establishing, acting, and learning. The latter stages of the IDEAL model are similar to the typical continuous process improvement cycle: plan-do-check-act. The widely adopted IDEAL model will be used as a structure for discussing approaches for continuous change and learning to enable the improvement of the software process.

The initiating stage establishes the business reasons for starting the SPI initiative. The change often begins with a stimulus for changing the software process; examples of stimuli include internal problems (e.g. rising project costs) or exogenous factors (e.g.

a paradigm shift in a technical or business area). These stimuli need to be taken into account when setting the priorities for the improvement project. So, the aims and needs of the organisation may be quite different to the requirements of the current or subsequent stage of a normative model. Consequently, successful process improvement projects are seen to be tailored to the specific business goals and characteristics of a software organisation (Birk and Rombach, 2001; Stelzer et al, 1998; Sweeney and Bustard, 1997). Fitzgerald and O'Kane (1999) show that identifying critical success factors appropriate to the organisation's needs for each level of the maturity model can help to tailor the improvements. Once the needs are identified it is necessary to ensure there is the support to proceed with the programme. Previous studies have emphasised the role of management support for quality and process improvement (e.g. Jansen and Sanders, 1998; Rainer and Hall, 2003; Ravichandran and Rai, 2000a). The need for improvement should be communicated to the executive and a case made using data from within the organisation. It needs to be understood though that it is not just management commitment that is important. At Bose it was found that a shared vision of the problems and potential solutions helped to integrate improvement work and resource planning for the project (Harkness et al, 1996). In software process improvement the perspectives of management and development staff are equally important in order to get true commitment and support on all levels. During the initial phase the key steps are, therefore, setting the business goals and obtaining support for the initiative.

The second phase of the IDEAL model is diagnosing. This phase is used to develop a common understanding of the current processes. An appraisal of the processes helps to identify the priorities for improving the software processes. One way to undertake

this appraisal is through the participative methods of the Soft Systems Methodology (Bennetts et al, 1998), seeking to develop a consensus on the appropriate way to change the process from the views of the various stakeholders. An alternative goal-oriented approach to identify the priorities is through defect data analysis (Bhandari et al, 1993), whereby problems in the process are identified from the problems in the product. With the generic SPI models, such as IDEAL, priorities are set according to the business objectives and the process assessment. This objective setting exercise will draw from the results of any assessment, the business aims and risks, industry norms and benchmarks, and the process areas defined in a relevant normative model. However, it is recognised that the process requirements will change over time as a result of changes in company goals and improved understanding of the organisational process (Christie et al, 1996). Consequently, such approaches can be tailored to the business needs overcoming some of the concerns about the normative models.

During the establishing phase the SPI infrastructure, outlined in the initiating phase, is set up and the action plans are identified. The IDEAL model anticipates that winning support will result in the approval to strengthen the process infrastructure in order to reduce the overall product costs. According to the SEI the infrastructure changes start with the establishment of a management steering committee, whose responsibility it is to set the business objectives (Curtis and Paulk, 1993). A software engineering process group (SEPG) is formed by the steering committee. The SEPG is a full-time group created to deliver the changes in the software process. It consists of a small number of respected software engineers, who initiate an assessment of the current processes and coordinate the resulting actions. Subsequently, technical working groups are set up to work on specific changes. The term used in IDEAL for these

groups is process action teams (this will be adopted here as the generic term for working groups). Action plans are developed for each process action team. The plans are drawn from the priorities set, the concerns about the existing processes and possibly the relevant normative model.

A company engaged in SPI typically documents its current 'as is' software processes and defines the preferred 'to be' processes. Defining processes is seen to be an important part of the acting stage of IDEAL. The acting phase introduces these changes and trials them. To complement the acting stage of the IDEAL model, Kellner et al (1996) tailored the continuous process improvement cycle into a heletical cycle for SPI. The cycle clarifies what should be done at each stage: evaluate process, develop process, install process, and monitor use. They propose that the heletical cycle supplements the high level IDEAL model by using it for each specific process area that needs improvement, such as project estimation or software design methods. The cycle continues through time encouraging on-going improvement to the same process.

Whilst many organisations have descriptions of the software life cycle, often they do not correspond to the processes actually performed during software development and maintenance (Curtis et al, 1992). The aim of process modelling is to define and analyse the software processes at a detailed level, in an attempt to overcome any previous imprecision. However, it is debatable how useful this is because 'process modelling approaches have little, if anything, to contribute to the search for process improvement' (Lehman, 1997). So we need to treat simplified prescriptive models of the process with some caution, both in terms of whether they could ever provide an

absolute description of the process and whether they are used in the manner anticipated by the authors of the process model.

Zahran (1998) argues that a process model should evolve to incorporate knowledge learnt from prior practice. So the final stages of IDEAL reflect on the changes made. Changes are evaluated using the measures captured. Lessons learned are documented and used to review the overall approach to process improvement. Birk and Rombach (2001) consider this stage to be the most important and should look to trigger further iterations of improvement. This evolution of the documented process is deemed to enable knowledge to be shared with others who do not have the same experience. Any such documentation therefore should be written to facilitate knowledge transfer and affect behaviour. However, this view of knowledge transfer and learning is limited as it does not explain how learning occurs within an organisation and the way in which software developers utilise knowledge. Each company, as well as the software engineering community as a whole, must accumulate and share its knowledge about improvement strategies, the infrastructure created should support this knowledge sharing (Birk and Rombach, 2001). Continuous improvement therefore requires a commitment to learning on the part of the organisation.

Much of the work by the SEI and others presumes that the SPI activity is at the organisational level. Yet it is clear from results of BOOTSTRAP and SPICE assessment work, that project teams are often at different levels of maturity and have responded with different levels of enthusiasm to any SPI programme (Koch, 1993). The Personal Software Process (Humphrey, 1997) and other bottom up approaches have been developed as a result. As process improvement activities are often local,

rather than the corporate-based activity envisaged in CMM, the learning stage of IDEAL could include the diffusion of the improvement to other parts of the organisation, to institutionalise the change. If the improvement activity is limited to a single development group then the focus of this stage is on learning from experience.

IDEAL and other continuous improvement models are often used in conjunction with maturity models such as CMM. The maturity models provide the key processes to be improved and the recommended stages of improvement. The continuous improvement models assist in the management of the implementation of the maturity model or of other software engineering practices per se. They also support the continued improvement beyond the foundational stages of the maturity models. So whilst maturity models have usefully informed practice and encouraged the introduction of software engineering techniques, with resultant benefits for many companies, a continuous view of the improvement activity therefore appears to offer greater flexibility and opportunity than applying the key process areas of a particular process maturity model.

Software process improvement is, however, considered to be more complex than following a set of steps. The focus of the SPI literature has moved towards a further understanding of 'how to' undertake process improvement and how process improvement occurs in practice. The 'how to' literature recognises the need for continuous improvement but remains embedded in a deterministic / contingent perspective of change. Other contemporary work has taken quite a different view: the need to understand how the improvement occurs in order to inform future practice. Ideas from the organisational learning discipline have been used to inform this

practice by recognising that improvement is a learning activity that takes place in a group situation. Others have considered SPI as a form of innovation, and thus considered the change to be a form of diffusion of innovation. By exploring these different strands of the SPI literature the next section will show how the key to understanding process improvement comes from understanding the activity as a process of continuous change.

## 2.4 Software Process Improvement as Learning and Innovation

Above it has been shown that the software process improvement (SPI) literature provides a set of normative models for organisations to follow to improve their software capability. These models are being adopted by organisations with some benefits, but criticism of these models shows that they rigidly state the practice to be adopted and organisations face difficulty in implementing the prescribed actions (Gray and Smith, 1998; Herbselb et al, 1997). Continuous improvement approaches have been adopted, partly to enable flexibility and partly to support the implementation of the normative models. Moving between the maturity levels of the models implicitly requires the organisation to learn new practices, but the models have no emphasis on learning. The IDEAL model of continuous improvement identifies the need for learning from the actions, but this element of the improvement process is dealt with as a relatively minor component at the latter stages of the model. In contrast, Van Solingen et al (2000) found that the most important factor was for organisations to learn how to improve their activities.

As discussed earlier, much of the quality management literature has built upon a manufacturing perspective. Improving a manufacturing process, where the process is

repetitive, can be achieved through quantification of the process. Using statistics it is possible to see where faults are occurring in the production activity; corrective action can then be taken. Software engineering has tried to adopt this approach with limited success. Measurement can support the improvement of quality, but measurement is susceptible to problems of consistency across projects with separate teams and development activities. Even if it were possible, the appropriateness of adopting a consistent development process across the organisation has been questioned (Fitzgerald, 1996), with diversity characterising the current era (Avison and Fitzgerald, 2003). So whilst the software process can be repetitive and features may be measured, often any repetition only exists in the sense of multiple projects with quite different characteristics. Understanding of the problems in software development and identifying improvements therefore can only come from those involved in the process reflecting on their experience rather than through measurement. The SPI literature has begun to recognise that both competence of an organisation's members and their ability to learn are important aspects of improving the quality of products and processes. Mathiassen et al (2002, p.7), for instance, suggest that:

> SPI is driven by knowledge about practices and perceived needs, insights gained during the improvement process, software industry standards, and state-of-the art methodologies and tools. SPI efforts also depend on the implicit, individual knowledge of participants. However, the general idea is to make knowledge explicit and to share knowledge.

This section will establish how learning and innovation theory can inform our understanding of how process improvements occur over time. From a brief review of

the literature, the relationship between process improvement and learning will be outlined here and expanded further in Chapter 5. The link between continuous improvement and ongoing learning will be developed to show that continually enhancing the underlying knowledge-base develops the competence in development projects necessary to be able to assimilate the new ideas. It will be shown that we need to look beyond this resource based view of knowledge towards an active view of knowing and learning. An understanding of learning and the use of knowledge as a process will be explored, showing that to understand how process improvements occur we need to encapsulate the way that organisations learn to improve over time. This action based view of knowledge leads us into seeing SPI as a process of sensemaking, where agents are seen to draw on their knowledge during the action, and learn through reflection on their experience.

An organisation's knowledge-base and competencies are often considered the sources of long-term advantage (e.g. Feeny and Willcocks, 1997; Pfeffer, 1995): a knowledgeable workforce enables a company to take on new approaches. This competency theory proposes that organisations compete through managing resources that are peculiar to an organisation and inimitable by competitors. Prahlad and Hamel (1990) argue that the core competencies are the collective learning of an organisation, harmonising diverse skills and streams of technology. They contend that it is through management's ability to develop these competencies that businesses are empowered to adapt to changing opportunities. Fichman and Kemerer (1997) confirm that organisations are more likely to innovate or take on innovation when the skills required are close to those in the organisation. New methods or development technology are readily identified when employees are knowledgeable about current processes. Successful application of ideas increases confidence and knowledge,

encouraging further changes (Allison, 1999). Competence of software developers can encourage commitment to projects and process improvement. When software practitioners 'understand and appreciate the process, they are empowered to use their discretion and adapt the process to meet the needs of both the situation and their customers' (Aaen et al, 2002, p.34). Without such ability or readiness to learn, the company is likely to face difficulties in delivering change and adapting to the demands of the market, and will therefore lose out in a competitive environment.

In an attempt to address this need, organisations are moving towards smaller IS groups staffed by talented people (Cusumano, 1997; Feeny and Willcocks, 1997). The moves towards leaner, more capable IS functions is a difficult task. Highly able software professionals are scarce, so IS executives need to take responsibility for developing their own staff. Moreton and Chester (1997, p. 190) state that 'it follows that education and training are needed throughout the systems function if staff are to become competent in - and then master - their changed roles and the demands those changes will make of them'. Development and training cannot be a one-off process at the beginning someone's career nor can it be periodically dropped in en bloc. Learning has to be continuous or *just-in-time* (McKeen and Smith, 1996). So, when implementing process improvement programmes, organisations are recommended to encourage learning amongst its members. Ravichandran and Rai (2000b), for instance, suggest that an IS group's ability to learn needs to be nurtured, as acquiring technical know-how places significant demands on developers. The enhancement of software process knowledge is a fairly complex endeavour because of the need to adapt to changes in the organisational environment. Organisations seek to address this learning by developing programmes and incentives to encourage this to happen (McKeen and Smith, 1996; Opolski and Szemborska, 1997).

These human resource management strategies do not offer immediate solutions. Rather, Scarbrough (1998, p.221) states that whilst this resource based view of knowledge traces 'a clear and compelling causal path from endogenous characteristics of the firm through its product and innovation portfolio to competitive performance', there are problems with tapping knowledge as knowledge is distributed and embedded in different social groups (communities-of-practice). So, he challenges the desire to orchestrate the distribution of knowledge in a formalised way, arguing that it tends to institutionalise the communities-of-practice from which competence emerges. The problem, therefore, is one of integration and not just a problem of combining, sharing or making data commonly available (Scarbrough, 1999).

The process of learning across an organisation, though, is not fully understood. Many recognise the separation of tacit understanding of individuals from the explicitly documented knowledge captured in formal systems, such as software process models and standards. The problem with seeing knowledge as only the explicit dimension is that much of organisational knowledge is not in its formal systems (the espoused theory) but in its beliefs, habits, routines, and memory of its members (the theory-in-use) (Argyris and Schön, 1996). This is why Choo (1998) encourages organisations to find ways to move the tacit to the explicit through sharing experiences, stories, reconfiguring existing knowledge and through internalisation of the tacit by experience. Pourkomeylian (2001) suggests that externalisation through socialisation occurs in software process improvement, such as through the creation of SPI plans and process descriptors. Whether it is possible to consciously share tacit knowledge in this way is debatable, but it recognises the importance of the dynamic, social element in the sharing of knowledge. What it misses is the reflective nature of

learning through action. Senge (1990) states that organisations are continuously changing through active implementation and reflection on their theory-in-use. Argyris and Schön (1996, p.16), likewise, state that:

> individuals within an organisation experience a problematic situation and inquire into it on an organisation's behalf. They experience a surprising mismatch between expected and actual results of action and respond to that mismatch through a process of thought and further action that leads them to modify their images of organisation or their understanding of organisational phenomena and to restructure their activities so as to bring outcomes and expectations into line, thereby changing organisational theory-in-use.

March and Olsen (1976, p.56) see this experiential learning as sensemaking, in which 'individuals and organisations make sense of their experience and modify behaviour in terms of their interpretations.'

McGuire and Randall (1999), also, argue that the software development activities involve multiple iterations and feedback loops that result in a maturing sequence of mental models for the human actors. Process models and software engineering features can be understood as frameworks for learning (Mathiassen, 1998). These can be drawn on by the developers in sharing and adopting ideas. Mathiassen (1998) uses Schön's concept of reflection-in-action to show how software developers learn by making use of their past experiences and adapting these to fit the current situation. Reflection-in-action assists organisational actors to gain insight into the background context of the change, thus it helps to reshape and restructure the organisational context. Schön (1983) considers that we show ourselves to be knowledgeable through

our intuitive actions and decisions in everyday life. Our knowing is often tacit, as seen in our patterns of action and feeling for the given activity. Schön (1983, p.49) therefore posits that 'our knowing is *in* our action'. So our work life depends upon our tacit knowing-in action: the ability to recognise phenomena, patterns and symptoms. So the know-how is in the action.

If we recognise knowing-in-action, Schön (1983) claims that we should also recognise that we sometimes think about what we are doing. This thinking implies that we consider what we are doing during the action so as to improvise and respond to the context. This 'reflection-in-action hinges on the experience of surprise' (ibid, p.56); when things please us or lead to undesired results we may respond by reflecting-in-action. At such moments, Schön considers that reflection interactively focuses on the intuitive knowing that is implicit in the action. A practitioner's reflection helps to surface and correct previous tacit understanding; they may reflect on the tacit norms and appreciations which support a decision. Practitioners also reflect on their knowing-in-practice through a post-activity review: they think back on a project.

Gasston and Hallorn (1999) show that software process improvement is dependent upon an organisation's ability to continuously see things differently, gain new understandings and produce new patterns of behaviour. Another way organisations often try to see things differently is by bringing in ideas from outside. The boundary scanning activity is often considered important in the adoption of external innovations. To achieve this adoption individual agents utilise a variety of external networks to identify and assimilate new knowledge relevant to the organisation (Merali, 2002). However, Lyytinen and Robey (1999) contend that external

knowledge may not be transferable into an organisation as it is based on the experience of others and, even when it is transferable, external knowledge often adds little competitive benefit as it is in the public domain. So, rather than looking for a simple solution to solve the problems, those involved in software development need to be constantly learning how to apply new ideas, whether they come from internal or external sources.

Lyytinen and Robey (1999) cite the failure of organisations to learn from their own previous experience as a reason for the recurrence of problems in IS development. They argue that learning from experience runs against the common practice because the incentives are not there to do this. When failure occurs the pressure is on IS management to externalise the problem, identifying deficiencies in the infrastructure, or its supply, rather than poor internal application. The result of this pressure is that improvements in systems development approaches have focused on the adoption of new tools, technologies and techniques from outside, such as the introduction of computer-aided software engineering (CASE) tools, new programming languages and development methods, but this emphasis does not produce the experiential learning necessary to avoid making the same mistakes with the next set of tools and technologies. To deliver this improvement an organisation needs to create knowledge from the inside and thereby feed the continuous improvement activity (Nonaka and Takeuchi,1995). To bring new ideas in from the external environment, organisations need to encourage this assimilation to be part of their ongoing learning, rather than as a simple solution to their problems.

Deliberate attempts to transfer knowledge and the spread of software process innovations have been considered to fit the theory of Diffusion of Innovation (Rogers, 1995). Diffusion is the process by which an innovation is communicated and transferred through a social system. An innovation is the implementation of an idea, not simply the creative idea itself, that is considered to be new by the adopting organisation or individual. Rogers' theory provides a classification of the groups of adopters from the innovators to the laggards, on the assumption that there is a predictable rate of progress of the use of any innovation. This theory has been utilised as a means of analysing quality management and process improvement innovations (e.g. Quintas, 1994; Gibson, 1998). These studies purport that Rogers' theory helps to understand how ideas spread through the industry or organisation, but these changes are more complex than Rogers' theory specifies.

The diffusion literature treats innovations as explicit, formalised knowledge that can be distributed. Innovations are seen as a singular item and something to be implemented in a fixed, mechanistic fashion where decisions are made by an expert. Software process improvement technologies and methods are not discrete packages because each implementation requires interpretation and will evolve through time (Swanson, 1994). The innovation activity is therefore contextual. Lyytinen and Damsgaard (2001) point out that technologies do not diffuse in a homogenous and fixed social ether but rather that a complex set of contextual constraints shape these changes. As such, as an innovation unfolds it is richer and more varied then the diffusion literature suggests.

Diffusion often assumes rationality in the adoption decision. The Diffusion of Innovation theory illuminates the role of the change agent, the decision process involved in adopting (or rejecting) the innovation, and the different types of knowledge drawn on during the adoption. Mustonen-Ollila and Lyytinen (2003) state that this adoption decision is made by those with resources and decision rights to change behaviour, but Giddens (1984) considers that we all have resources at our disposal, questioning the ability of a hegemony to impose an innovation upon the rest of the organisation. Lyytinen and Damsgaard (2001) also contest this view of diffusion by claiming that adoption choices are not the function of available information, preference and the properties of adopters, but of local, fluctuating factors.

Mustonen-Ollila and Lyytinen's (2003) own results offer a better understanding by stating that majority of innovations resulted from internal learning and experimentation. So, successful innovation involves learning, iteration and the emergence of new ideas, because processes 'are improvisational in that they combine real-time learning through design iterations and testing' (Eisenhardt and Tabuzi, 1995, p.108). The integration of any new innovation is not simple and requires intensive learning; the adoption should not be seen as a stand alone action (Lyytinen and Damsgaard, 2001).

Looking at the introduction of new approaches as diffusion does not address the dynamic relationship between all those involved (Kautz and Larsen, 2000). Many institutions and actors have a role in diffusion and so it 'forms part of a complex interplay of multiple technical systems' (Lyytinen and Damsgaard, 2001, p.179).

Mustonen-Ollila and Lyytinen (2003) therefore argue that empirical research on IS process innovations is largely lacking organisational, behavioural, and contingent factors and how these factors influence the adoption of the software process tools and methods. One of the reasons that organisations do not learn from experience is 'internal tension, power play, and time demands' (Lyytinen and Robey, 1999, p.88). Scarbrough (1998) suggests that a view of knowing as a relational activity locates it in the terrain of organisational politics.

It is widely accepted that organisations cannot be changed successfully based solely on rational approaches. Organisational change involves learning, iteration and emergence of new ideas and initiatives as the process evolves (Mathiassen, 1998, p.45). Lyytinen and Damsgaard (2001) also point out that the timescales in the implementation of any innovation are not necessarily short, so recognising the time-based factors in the emergent change is necessary. So we need a context and process based model that helps us to understand this emergent change as occurring through reflective learning. It is therefore necessary to accommodate iterative experimentation, use, and learning in any model of change. So it is not surprising that recently there has been significant work in the SPI field trying to understand the activity of process improvement as a learning activity.

If we are to understand process improvement as a cycle of ongoing change then we need to understand what is happening within the change activity. The following section will show that much of the previous work takes a deterministic view of the change: believing that goals set during the initiating stage will be delivered and produce a beneficial effect. Originating with the maturity models an outcomes based

view of process improvement has dominated the field. By assuming that each of the maturity levels will improve the work of the development organisation key process areas defined by the models are used to determine what should be changed. Whilst continuous process improvement should be more attuned to the local problems and requirements it is still seen as a transformation of the development organisation towards a pre-determined goal. IDEAL and other similar approaches take little account of the organisational context. 'In designing a good improvement path we need means for obtaining additional coverage of the complex world of software production as the software process is only part of it' (Jarvine, 1994, p.260). Thus, if we are to understand how continuous change occurs we need to take into account the organisational context it is occurring in, the actions of the various actors and how these interact to enable the learning and knowledge creation said to be required to improve the process.

## 2.5 SPI as Emergent, Organisational Change

Outcomes rather than the process of improvement have been at the forefront of SPI research. Software process improvement, though, is more complex than the planned and clearly defined activity the software engineering hegemony would have us believe. It has been shown that deterministic approaches to software process improvement have their limitations. A number of organisations have successfully used such models, but many others have failed. Learning to improve has been highlighted as an important element of understanding how SPI occurs in practice, but this is only one aspect. Other factors involved in the change are considered to be organisational in nature so there is a need to understand how these factors shape the improvement activity. The consideration of factors that shape the introduction of specific

technologies has shown that software process and product variations emerge from particular interactions of institutional context, key players' intentions and actions, and the new technology or method (Orlikowski, 1993). However, the existing literature does not extend into understanding the way that these factors actually affect the process as it is enacted. In this section, it is posited that SPI can be understood as a form of situated, emergent change. This argument will be based on an understanding of activity as a form of sensemaking. The outcome of this discussion will provide the basis for the framework of analysis developed in Chapter 5.

This section will begin by comparing previous models of change with the emergent views now being proposed in the organisational and IS literature. Orlikowski (1996) identifies three perspectives that have influenced studies of technological change based on organisational transformation: planned change, punctuated equilibrium and technological imperative. Each of these perspectives is discussed and related to previous SPI research. To understand how process improvement occurs within its situated context previous models of change will then be compared with the situated, emergent views now being proposed in the literature.

The literature on planned change presumes that managers are the primary source of organisational change and that they deliberately initiate and implement change in response to perceived opportunities to improve organisational performance. Lewin (1951) first proposed a three stage model of change that now forms the basis of much of the theory of organisational change and development. This approach sees change as an intervention to disrupt the status quo, requiring managers to unfreeze, change and refreeze their organisation. The interventions often take the form of standard solutions

from vendors or consultants 'driven through the organization by directives from top management' (Weick, 2000, p.232). The change activity is seen to involve learning and changing the behaviour of individuals that can be captured in the new stabilised organisation. Benchmarking, training, and consultants can all be used to reinforce the need for, and facilitation of, the change. The purpose of this change model was to increase the likelihood of successful change, but this approach disregards the context of change and assumes that the variables of the change are within the control of the change agent. The management of change on this basis can be criticised for being overly reliant on solutions from other contexts (Lyytinen and Robey, 1999). N-step models ignore 'the variety of influences and orientations which impact upon people at work and their experiences of change' (Collins, 1998, p.100), so a more socialised model is required. Research of change as a planned activity is also criticised for the implication that it is controlled as intended, because looking back at a change will always give a neat, rational view of the process (MacDonald, 1998) and for simplistically treating change as a discrete event (Orlikowski, 1996).

Similarly, punctuated equilibrium assumes changes to be 'rapid, episodic, and radical' (Orlikowski, 1996, p.64), where periods of relative stability are interrupted by short periods of radical change, often triggered by changes in the internal or external environments. Researchers have adopted punctuated equilibrium as the primary model of organisational change, but how appropriate it is for studying change within dynamic environments, where change is frequent and perpetual, is being questioned (Brown and Eisenhardt, 1997). Weick (2000) shows that to hold this perspective we must see organisations as inert; developing structures that hold the organisation in place. In order to change, it is considered that we must move from a given state to

another by making an intervention that disrupts the equilibrium. In both the punctuated equilibrium and the planned models of change, organisations are seen to move from their current state to a better or pre-planned state by the application of substantial force.

The technological imperative perspective of change takes a different view of organisational transformation. Rather than there being an intervention by a change agent, technology is considered as the primary driver of change. The introduction of new technology is seen to create predictable effects in organisational structures and processes. In this view of change, technology has a positive impact and problems are solved solely as a result of the technology. It is assumed that technology is beyond our control and influence: organisations are forced to adopt the new innovations to maintain their competitive position and work is designed to fit with the demands of the technology. However, this view is flawed because there is a lack of consideration of the actor's role, contrasting with current organisational discourses 'where assumptions of agility and flexibility require actors to explore, learn and innovate new alternatives for working and organizing over time and in different circumstances' (Orlikowski, 1996, p.64). There are choices of technology to implement, the way the technology is implemented, how the work is organised around the technology, and our individual response to those choices.

Each of these perspectives on change, and their perceived weaknesses, helps to shed some light on the software process improvement literature. The SPI literature adopts aspects of each of the above models. Whilst SPI does not deal directly with the

introduction of new technology, the adoption of engineering methods can be equated with technology as they are prescribed solutions from external sources.

In line with the planned view of change, the maturity models have a strong view about how the change can be managed from the top. They assume that the managers and senior engineers are the principal change agents, initiating and setting up organisational structures, such as software engineering process groups (SEPG), to implement pre-determined approaches. At each stage the SEPG identify the next key process area from the model to be disseminated. By treating each change in this way the decisions are isolated from the different contexts across the organisation, thus the events involved in introducing these changes and the intentions of those involved are ignored. The process of change is expected to follow Lewin's three stage approach with the introduction of the process areas associated with each level of the capability model. A major change programme is assumed to move from one level to the next before consolidating and stabilising around the new software process areas. The CMM assumes that the whole organisation moves forward as one, but the application of tools and methods will vary in depth and specifics across teams and team members. Each instantiation will be different as the developers use improvisation and bricolage to solve their short term problems (Ciborra, 1999).

The associated continuous change models, such as IDEAL, take a punctuated equilibrium view. Following the discussion about the maturity models above, each cycle of change adopts a new set of ideas for improvement. Between each major cycle of change a number of adjustments are made in response to any evaluation made. In line with the punctuated equilibrium model the changes are driven by a champion who

perceives the change to be an opportunity to improve. IDEAL and other similar approaches in the literature also view change in a deterministic manner: following a rational action plan to reach pre-determined goals. After using the IDEAL/CMM approach to undertake an SPI programme, Kautz et al (2001, p.108) found that this is not how it occurred in practice and concluded that, in contrast to the theory, the change was 'not a straight forward, rational process'.

The normative maturity models, such as the CMM, apply the concepts of the technological imperative view where each maturity level is seen to be better than the lower levels. It is assumed that the software engineering ideas are right in all circumstances and will enable improvements in the quality of the software activity, but, as shown above, these models are criticised as being artificial in nature as they are based on an ideal organisation that never existed. Indeed, Briand et al (1999) consider that strict adherence to an activity-driven approach can be disadvantageous because the defined outcomes may not be *correct* for the specific organisation. Organisations, as discussed earlier, are mechanistic in their use of the models and consequently can have problems following the recommended sequence. Gray and Smith (1998) postulate that often achieving the next level is seen as more important than the improvement or having a good process. A lack of adherence to the defined *mature* process was evidenced in the early SPICE trials. Stienen (1999) found that 80 percent of level 2 organisations had not only most of the level 2 processes in existence but also 60 percent of level 3 processes. In practice, however, only 40 percent of all level 2 criteria were fulfilled as organisations did not follow the documented processes. So we should not understand software process definitions as accurate prescriptions of practice because, as Ciborra (1999) noted, even written formal

instructions are interpreted by experienced workers. When used in practice they are adapted, selectively used in combination with other methods, and applied based on the experience of the developer.

By taking a deterministic view of SPI, previous research of this form of organisational change neglects the context of the process, the events involved in adopting new approaches to developing the software product and the intentions and actions of the people involved. Mathiassen et al (2002, p.29) state that 'software process improvement is an evolutionary process in which you implement changes incrementally, over time, rather than in a few dramatic transformations.' They argue that most SPI approaches now emphasise stepwise improvements, as those approaches that are technology-driven or revolutionary have not been effective. The problem is that much of the process modelling and assessment work has forgotten that software is the product of people not of a conceptualised process. Hosking and Anderson (1992, p.6) argue that the:

> ability to instigate, plan and direct all forms of change has been taken for granted, creating a present-day misinterpretation appropriately termed 'the illusion of manageability'. By this we mean that the extent to which managers are able to direct or predict change processes is overestimated, and the way in which they might contribute to change is fundamentally misconceived.

Indeed, they state that there is considerable evidence to suggest that change is far from controllable and can only be influenced to a limited extent; the intended purpose of the intervention is often overcome by unexpected or unintended outcomes. So the

challenge is to understand change not as a predictable or designed causal outcome, but as an emergent process developed from the relationship between people and their context. To further our understanding of software process improvement, it is necessary therefore to move beyond these prior perspectives of technological and organisational change. What is required is an integrative theory, building on research that examines the enabling and constraining factors on quality management practice (Ravichandran and Rai, 2000a): an understanding of change that reflects a more complex, dynamic and unpredictable world. So, 'a perspective that posits change rather than stability as a way of organizational life may offer a more appropriate conceptual lens with which to think about change in contemporary organisations' (Orlikowski, 1996, p.65). However, all three of the traditional models of change discussed neglect the distinction between deliberate and emergent strategies, or changes.

An emergent change is realised through action. Orlikowski (1996, p.65) argues that a situated change perspective helps to explore the ongoing practices of organizational actors that emerge 'out of their (tacit and not so tacit) accommodations to and experiments with everyday contingencies, breakdown, exceptions, opportunities, and unintended consequences that they encounter'. Each action either changes or reproduces existing organisational properties and practices, so the organisation will metamorphose 'through a series of ongoing and situated accommodations, adaptations, and alterations' (ibid, p.66). As these amendments are sustained over time then fundamental changes occur:

> The recurring story is one of autonomous initiatives that bubble up internally; continuous emergent change; steady learning from both

> failure and success; strategy implementation that is replaced by strategy making; the appearance of innovations that are unplanned, unforeseen, and unexpected; and small actions that have surprisingly large consequences. (Weick, 2000, p.225)

The concepts of situated change are founded on the social theories of sensemaking and improvisation. A situated change perspective helps to explore the ongoing practices of organizational actors that emerge out of everyday practice. March and Olsen (1976, p.56) see sensemaking as part of experiential learning in which 'individuals and organizations make sense of their experience and modify behaviour in terms of their interpretations'. To engage in sensemaking is to frame the boundaries of the problem, and impose coherence upon it to allow us to decide how the situation needs to be changed. We are always in the middle of complex situations as there are no absolute starting points: sensemaking is an ongoing activity. In improvised change the solution applied is refined to fit the need on each occasion. To do this, organisations need to create new knowledge and to draw on the competencies available. Sensemaking helps us to understand situated, purposeful change in dynamic organisations in a different way to the deterministic view. Taking this perspective can help us to understand the unfolding changes in SPI through the actions of the human actors, whether intended or not. The intention of the key actors and their design for the change are important aspects of understanding the reasons for and the consequences of the changes.

Weick (1995) considers sensemaking to be the process of constructing an interpretation of the unknown by active agents. Whilst not all authors agree, he argues

that action results from sensemaking, as interpretation is only part of sensemaking not synonymous with it: 'sensemaking is about authoring as well as reading' (Weick, 1995, p.7). It is an activity or process of invention not simply a discovery of 'the interpretation' (ibid). Individuals try to make sense of their experience. Even when the experience is complex and ambiguous they impose order, attribute meaning and provide explanations (March and Olsen, 1976).

Sensemaking is an ongoing activity. We are always in the middle of complex situations; there are no absolute starting points. Weick (1995) shows that problems do not present themselves as givens, but need to be constructed from complex, fuzzy, puzzling and uncertain situations. To engage in sensemaking is to frame the boundaries of the problem, and impose coherence upon it to allow us to decide in what direction the situation needs to be changed:

> There is a strong reflexive quality to this process. People make sense of things by seeing a world on which they have already imposed what they believe. People discover their own inventions, which is why sensemaking understood as invention, and interpretation understood as discovery, can be complementary ideas. (Weick, 1995, p.15).

Interpretation and invention, or improvisation, are at different ends of a continuum (Weick, 2001). With improvisation greater modification of the original model is implied than with interpretation which is seen as just giving meaning to the original. Improvisation is a mixture of the pre-composed and the spontaneous (Weick, 2001). It does not materialise out of nothing. Ciborra (1999) points out, however, that the improvisation is planned not haphazard; as people practice they develop the skills and

understanding necessary. Weick (1995) develops the idea of enactment to show that we receive stimuli as a result of our own action. We do something (say to improve our context) which then helps us to do our job better so we continue to improve it. Sensemaking is, therefore, more than simply a cognitive process, but one that exists within the body. Mingers (2001a, p.123) therefore argues that the physical embodiment of our action 'suggests that much that we "know", in the sense that we are able to undertake particular actions and activities, is essentially tacit, habitual, and beneath our consciousness'. Knowledge is therefore learnt through action by practice and habituation.

Weick uses the example of the improvisation of a musician, which is based on a prior understanding of the score, the music as it has been played thus far and the skills and repertoire the musician can bring to playing the piece. To do this the musician utilises pre-composed fragments and embellishments to compose in real-time, using patterns that have occurred in the music so far. These patterns are discovered retrospectively, so as the action of playing occurs sensemaking happens. Also, on each occasion the solution applied is refined to fit the need. So we can see improvisation and sensemaking as 'purposeful human behaviour which seems to be ruled at the same time by chance, intuition, competence, and outright design' (Ciborra, 1999, p.136).

If we are to understand changes in the software process as this combination of events, we need to be clear about what is meant by design because planned and emergent perspectives of change hold somewhat different views of design or plans. Suchman (1987) was one of the first to apply the ideas of situated action to problems of computer development. Whilst her work focused on human-computer interaction it

provides a useful critique of the traditional views of plans and human action. Traditional views of plans see them as prerequisite to and prescribing action. The architectural metaphor, that is implicitly a planned view of change, has plans as a blueprint of the outcome, but as shown above this contrasts with the idea that we do not know what will happen and can only make sense of what has happened retrospectively. The traditional view of plans considers that developers will develop mutual intelligibility because the process model has enabled 'common conventions for the expression of intent, and shared knowledge about typical situation and appropriate actions.' (Suchman, 1987, p.27). Plans (or defined processes) are created to produce order and to achieve the planned change through the intended design, but whilst change unfolds as if intended it actually consists of patterns that emerge gradually.

Weick (2001) argues that design understood to be planned is flawed because, at best, it can only represent one point in time. The emergent view of change shows that:

> while a course of action can always be projected or reconstructed in terms of prior intentions and typical situations, the prescriptive significance of intentions for situated action is inherently vague...these are tied to local interactions contingent upon the actor's particular circumstances. (Suchman, 1987, p.27)

Emergent, continuous designing is sensitive to incremental changes in local conditions. So design viewed as emergent is continuously updated as people and conditions change, thus they are more sensitive to what people pay attention to than architectural blueprints (Weick, 2001). Blueprints can often be seen as the intention of the designer, but as individuals can only make sense of the past then designs must be

based on what has gone before. So plans fall short of action and their only purpose is to orient the actor in such a way as to use their embodied skills.

Pettigrew (2000, p.246) is concerned, however, to avoid the situation where research in organisational change adopts a dichotomy of planned against emergent, and argues for a move towards a recognition of 'mutualities and complementaries of planned and emergent change'. Whilst the emergent properties of actions mean that they cannot be known a priori, that does not mean that there is no intended design for the action. Rather, design and emergence coexist and that emergence alters design, as 'the capabilities of a living system in any place and time arise from the interplay of design and emergence' (Senge, 2000, p.78). This combination of planned and emergent activity provides us with the *mutualities* of the two views, but is potentially criticised for mixing the views of action being based on the past, as understood in the sensemaking perspective, and being based on our planned view of the world, as seen by rationalists. Ciborra (1999, p.144) sums up this dilemma when he asks 'how can the orientation to the future, which seems to characterize both improvisation and planned decisions, be reconciled with a view where subjective, highly circumstantial interpretation of the past gives the ultimate meaning to action?' He uses Schutz (1967) to show that action is carried out according to a projection, which envisions the act as if already accomplished. The projection contains both *in-order-to* and *because-of* motives. The *in-order-to* motives represent the conscious, explicit plans and goals but are only the 'tip of the decision-making "iceberg"' (Ciborra, 1999, p.144), whereas the deeper reasons for the problem are encapsulated in the *because-of* motives drawn from our experience. There is therefore a dynamic relationship between our beliefs

and our actions. Actions generate beliefs and actions are based on our beliefs (Weick, 1995).

So by understanding changes in software process as situated, purposeful action, they can be seen to be 'simultaneously rational and unpredictable; planned but emergent; purposeful but opaque; effective but irreflexive; discernible after the fact, but spontaneous in its manifestation.' (Ciborra, 1999, p.137). It is argued that software process improvement can be best seen as ongoing improvisation enacted by the development team whilst making sense of the world. The following section will explore how this situated change can be understood through Structuration Theory.

## 2.6 A Structurational Perspective of Software Process Change

Sociologists have long debated the relationship between social structures and the action of members of those societies. Functionalism and structuralism dominated the social theories for much of the 20th Century. Both these theories, which can be traced back to Durkheim, tend to strongly emphasise that the social structures in societies exert constraint over their members. These social structures are conceived to be external to us, comparable to material structures. In contrast, hermeneutics and other interpretive forms of sociology consider social phenomena to be the products of human agency based on their interpretation of the world. These views place action and meaning as primary in understanding human activity, with social structures given little importance in the action but derived from the action. Structuration Theory is a process theory that attempts to resolve the debate between the social theories of human agency and those of structure by using a duality of structure. These are not separate but interconnected phenomena, with human action occurring by drawing on

social structures and in so doing reproducing the social structures. Structural properties are therefore 'the medium and outcome of the contingently accomplished activities of situated actors' (Cassell, 1993, p.132), and are considered to be enabling as well as constraining, with all human agents having the option to act according to their own will.

In understanding social structures in this way Giddens considers that they are not simply an exogenous force independent of human action nor only a product of human agency, but are 'rules and resources, or sets of transformation relations, organised as properties of social systems'(1984, p.25). The sense in which Giddens uses the term 'rule' is relatively weak, as it is not a causal relationship but it is something that occurs habitually and it is part of the routine. More specifically, Giddens (1999, p.124) defines the rules of social life as:

> techniques or generalizable procedures applied in the enactment / reproduction of social practices. Formulated rules – those that are given verbal expression as canons of law, bureaucratic rules, rules of games and so on – are thus codified interpretations of rules rather than rules as such. They should be taken not as exemplifying rules in general but as specific types of formulated rule, which by virtue of their overt formulation, take on various specific qualities.

He is at pains to point out that it does not imply that social life can be reduced to a set of mathematical formulae, on the contrary he argues against the production of 'laws' akin to the natural sciences. Generalisations can only, at best, be historical in nature. Thompson (1997) criticises the proposal to conceive of structure in this way because the notion of rule is too vague to fully analyse structure. Giddens (1999) rejects this

criticism stating that Structuration Theory allows both the personal and institutional aspects of structure to be analysed.

Additionally, Giddens (1979, p.202) considers social interaction is often treated as just a snapshot in time, arguing that because 'patterns of interaction that exist are situated in time', then only by examining them over time are any patterns formed at all. Consequently, Giddens places time at the centre of much of his writing. The duality of structure is the primary basis for continuities of social practice through time and space. Through this duality, he ties together the individual and institutional levels of social practice and points to the recursive nature of social life. Thompson (1997, p.328), despite his criticism above, concludes that Giddens has done more than any other contemporary author 'to advance our understanding of the complex ways in which action and structure intersect in the routine activities of everyday life'.

Giddens (1984) does not see actions as isolated phenomena but sees them as a flow of events, with agency taking place with knowledge and practical consciousness. Actions are purposive and intentional, but an agent is not always conscious of all the consequences of their action. These consequences become conditions of new actions, therefore structures change constantly but in unpredictable ways. So an agent's actions are always bound by the unacknowledged conditions and unintended consequences of action. Bertilsson (1997) argues that this view of agency can lead to an inconsistency between a view of actors having a will capable of choice and the actors having an affiliation with structures that make them conform to the routines. Addressing this criticism, Giddens (1979) shows human beings to be knowledgeable about the structural framework in which their conduct occurs because they draw upon

that framework in producing their action, at the same time as reconstituting it through their action. Everyone must have knowledge of the society in which they operate because of their participation in it, but also as Orlikowski and Robey (1991, p.149) note:

> the role of structural properties in shaping human action and interaction is transparent to human actors. Actors often believe they act freely within organisations, and hence structural properties remain unacknowledged as the condition of their actions. Whether individuals are conscious of the influence of these properties or not, their action is not possible without the interpretive schemes, resources and norms they use to realise their intentions.

So, human actors are seen to be continuously, reflexively monitoring their actions. The reflexive monitoring of action refers to the intentional behaviour of human beings. Giddens (1979) states that intentionality is a routine feature of human conduct, but this is not the same as having clear view of the outcome. A theory of motivation therefore has to recognise unacknowledged conditions of action that are outside of the self-understanding of the agent. Giddens (1984, p.6) distinguishes the reflexive monitoring and rationalisation of action from its motivation: 'if reasons refer to the grounds of action, motives refer to the wants that prompt it'.

Giddens's duality of structure interlinks action and social structures through a set of modalities: interpretive schemes that enable communication through shared stocks of knowledge; facilities for utilisation of power; and norms that are used to maintain structures of legitimation. The dimensions of the duality are shown in Figure 3

(Giddens, 1984). The separation of these modalities is only for analytical purposes, as they occur together in practice. Incorporating these modalities into the analytical framework therefore sensitises the researcher to these unconscious features linking the process of change to the emergent contextual structures. Each of the modalities and how they help us understand process improvement will be explained further below.

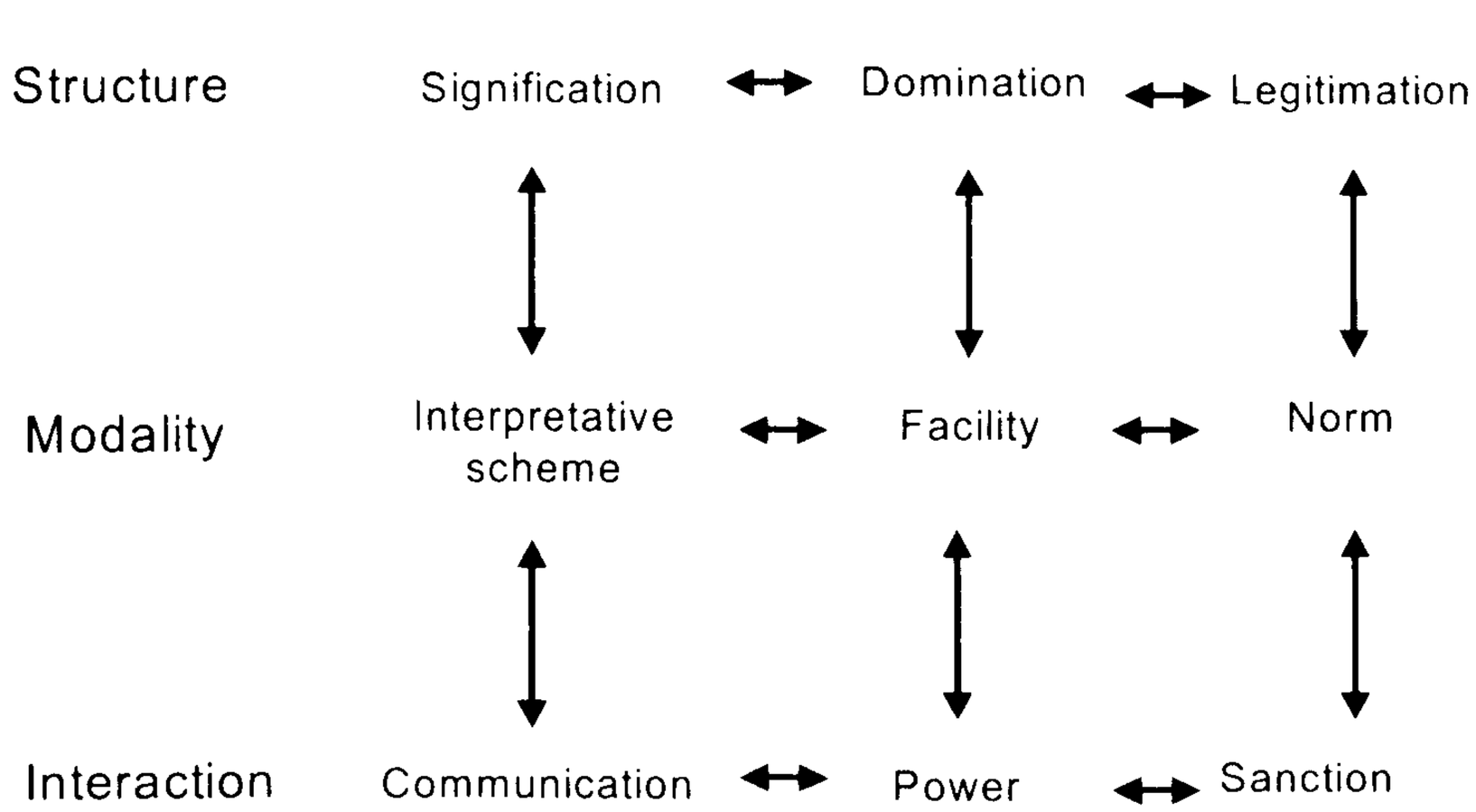



**Figure 3 Dimensions of Structuration Theory (from Giddens (1984))**

Firstly, human communication uses interpretive schemes, which are drawn upon to make sense of the actions as they are played out. Interpretative schemes are shared stocks of knowledge that we use to make a meaningful interaction by interpreting events. Interpretive schemes do more than enable shared meanings through communication; they also serve to impose structural constraints. Structures of signification represent the social rules that enable, inform, and inhibit the communication process. In drawing on the interpretative schemes those rules are reproduced and recreated. So the communication of information and meaning through signs and language change over time. Thus, in any interaction, shared knowledge is

not merely background but an integral part of the communicative encounter, in part organizing it, and in part being shaped by the interaction itself" (Orlikowski and Robey, 1991, p.149). This perspective fits the active learning concepts discussed earlier where defined processes can be understood as frameworks for learning and drawn on by the developers in sharing and adopting ideas. Systems development and improvement is the outcome of a complex process of interaction and communication influenced by the situated characteristics of the actors (Mathiassen, 1998). Checkland and Scholes (1990 cited in Mathiassen (1998, p.19)) summarise this by stating that software development is an 'ongoing learning process, or interaction process, integrated into the streams of ideas and events as they unfold in practice'. Software process improvement, therefore, can be seen as ongoing improvisation, not just a pre-planned action, that occurs as the development team make sense of the specific development activity facing them. This reflection may be a speculative or deliberate attempt to learn for the future, but it is implicit in Giddens's view of actors being knowledgeable and continuously reflexive.

The degree that managers and practitioners work together to change the process is dependent upon the trust between them (DeMarco and Lister, 1987; Fincham, 2002; McGill, 2001). Trust is a generalised attitude of mind that underlies our decision making that relates to our development through reflexivity. It is determined by interplay between people's values, attitudes, and moods and emotions. Trust evolves and influences cooperation and teamwork: it can be considered to be a spiral. Whether the spiral is positive or negative depends upon the reciprocal reaction to expectations. This development of trust is a function of the organisation's ability to create a setting within which trust can develop. (Jones and George, 1998). Trust is stronger than a

weak form of confidence, it presumes a commitment (Cassell, 1993, p.292). Trust underlies a host of day-to-day decisions that all actors take in the course of orienting their activities.

Human agents also draw on facilities, such as human and technical resources, to utilise power in interactions and in so doing maintain or modify structures of domination. Power should not necessarily be understood negatively or as solely in the hands of those in authority, or as a means to achieve the desired outcomes. Power is enabling and productive as well as constraining, it is what enables the existence of different positions within the organisation. Walsham (1993, p.40) suggests that 'power and its use in political activity pervade all action and discourse in organisations, that the exercise of power is a continuous process that has subtle local properties, and that local actions are linked in a complex way to more general networks and institutional frameworks.'

Politics and influence are the processes, the actions and behaviours through which the potential power is utilised and realised (Pfeffer, 1993). Giddens (1984) argues that as the use of power in organisations is through the way individuals are able to apply resources (organisational and personal). Jermier et al (1994) support Giddens's view that employees are active and wilful agents, negotiating social realities that are only partly of their own making. It is therefore within the capacity of each agent to change or retain current approaches, through the utilisation or withdrawal of those resources. The operation of authority also relies on the acquiescence of the other actors as all human agents have the option to act according to their own will, even when the threat of sanction is extreme. Power is therefore only effective to the extent that others allow

it to shape their actions. Jones and Karsten (2003, p.14) note that 'this voluntarist view of power as being instantiated in action rather than a type of act ...provides a distinctive approach to a central issue in organisational analysis'. From this perspective then power is part of all organisational relations and is the process through which structures of domination in organisations are sustained, reproduced and changed. This position is different to the two principal views of power: that it is either the capacity to achieve desired or intended outcomes or that it is the property of society. By combining these views, power can be seen to be part of the duality of structure, where resources are structured properties of social systems, drawn upon and reproduced by knowledgeable practitioners.

Thirdly, Giddens (1984) argues that we sanction our actions by drawing upon norms thereby creating or recreating structures of legitimation. Norms are the rules or standards that govern appropriate conduct, constraining and enabling action. Norms have an important part in shaping organisational understanding of appropriate, and sustaining established, behaviour. Ricoeur (1991) describes norms as organising action: configuring it, giving it form and sense. As mentioned above, the meaning of any action depends on the systems of conventions, or structures of signification therefore the precise context and timing of the action are important in their symbolic interpretation. Giddens uses the concept of legitimation because it does not suppose that everyone adheres to a system of norms, but that the structures of legitimation are used to justify people's actions (Craib, 1992).

Sanctions can be classified according to whether the resources used to produce the sanction are internal, i.e. involve elements of the actor's personality, or external, i.e.

draw upon features of the context of action. Several sanctions may operate simultaneously, and external sanctions are only effective if they impinge upon an internal one (for example, a reward is only such if it is something someone wants). Normative elements of social systems are a set of contingent claims which have to be sustained and 'made to count' through the effective mobilisation of sanctions in the contexts of actual encounters (Giddens, 1984, p.30). Normative sanctions express the structural asymmetries of domination identified above. Human action is shaped therefore by the way others use sanctions.

However, Giddens's lack of attention to culture, viz people's histories, habits, customs, feelings, and other aspects of non-agency, is criticised by Mestrovic (1998). Mestrovic (1998, p.110) states that 'without such emotional faith – not just rational trust – human agents could not believe in the objectivity of human institutions ...sufficiently to be able to act on their own perceptions and conceptions.' Giddens certainly makes little attempt to engage with emotions and feelings, other than by drawing on the work of Freud to develop an understanding of motivation. Yet, an understanding of culture can be seen to be embedded within the modalities and the social structures. Like norms, culture can be conceived as 'habits of the heart' (Mestroviz, 1998, p.219). Culture directs attention to sources of coherence and consistency in organisational life, and to the dominating beliefs or ideologies (Pettigrew, 1987). Within Structuration Theory these facets of culture are part of the unconscious motivations that inform our reflexivity, and the shared cognitions run through the structures of signification, domination, and legitimation drawn on to interact through communication, utilising power and applying sanctions.

Many authors in the SPI literature have also discussed the importance of organisational culture, emphasising the need for a cultural change (e.g. Conradi and Fugetta, 2002; Thompson and McParland, 1993; Jansen and Sanders, 1998; Gibson, 1999). The results of these studies have assumed that organisational culture is a unitary item, but there is not a single overriding culture that can be understood as *the* corporate culture. To take this position is to assume that everyone in an organisation has a singular view of their world. Static definitions of this nature do not bring to life the nuances of the organisation. Rather than seeing culture as a single variable, i.e. something that an organisation has, culture is understood here as something emerging from social interaction. This sensemaking perspective of culture reflects the subjective way in which we interpret and develop meaning from previous experience and act upon the attitudes determined from that experience. Structuration Theory understands *culture* to be represented in the structures, as interpreted and acted upon by each human agent.

Weick (1995), however, shows that culture is not entirely at the individual level, because human thinking and social functioning are aspects of one another. Members of a society have a feeling of some form of common identity, which may manifest itself in both practical and discursive consciousness (Giddens, 1984, p.165). Individuals may be aware of belonging to a definite collectivity without necessarily agreeing that this is right and proper. Within any organisation there are sub-cultures, or communities-of-practice, who have a distinctive set of shared meanings. These groups develop from organisational infrastructures, an affinity towards each other and regular interaction.

## 2.7 Conclusion

The problems with implementing SPI are understandable if we consider it to be organisationally situated. In contrast to the dominant deterministic views in the SPI literature, this chapter has shown that current thinking has moved towards a learning perspective of process improvement. Aspects of this work on SPI as learning and situated change will be incorporated into the theoretical framework developed in Chapter 5, highlighting the intertwining between software development and process improvement.

From this background, using the organisational change literature, it has been shown that by understanding change as a situated, emergent activity that is founded on action as sensemaking we can understand the interplay between the activities of software development and process improvement within their given context. This alternative view helps us to better understand the relationship between context and action. Systems development and improvement is the outcome of a complex process of interaction and communication shaped by the context of the actors. It has been shown that this interaction can be seen as occurring through a structuring process.

Using Structuration Theory, it has been shown that the modalities that enable and constrain the duality of structure and action can inform our analysis of improvements in the software processes as they emerge. Giddens's views have been supplemented by identifying the way the concept of culture, and communities-of-practice in particular, support Giddens's Structuration Theory in explaining change. There are numerous examples of IS research papers that have made use of Structuration Theory (Jones and Karsten, 2003). Not all these researchers have remained true to Gidden's own thinking, especially where ideas have been added so as to incorporate a

technology specific focus. Many of these vagaries are derived from a desire to incorporate a realist view of technology and information systems. This dissertation is less concerned about interpreting the impact of the technology but is an analysis of human action, and so this issue does not apply. Jones (1999b) suggests that researchers should pay special attention to the contextual sensitivity; the complexity of human intentionality; and the subtlety of social constraint. Each of these points are essential features of this research project. The theoretical framework will incorporate these views to provide a basis of the analysis of SPI within the case to understand how the process of change occurs as it is enacted. The following chapter will discuss the research methodology adopted to enable this analysis to occur.

# Chapter 3. Research Methodology

## 3.1 Introduction

The purpose of this chapter is to discuss how the approach adopted here addresses the issues of relevance to software process improvement, whilst achieving the quality of research required. The interdependence between practice and research is considered important for the development of the information systems field, but this symbiosis is not always evident in the work published. By comparing the list of most important issues for IS executives and the subject of IS publications, Galliers (1995) shows that the IS field prefers to research topics that have little relevance to practitioners. Choosing to select topics that are only of interest to the research community results in the research remaining in a theoretical, isolated world (Davenport and Markus, 1999). These authors and other prominent IS academics have called for an improvement in the relevance of our work to practitioners, whilst retaining the quality of research required to ensure rigour is maintained.

Before moving on to discuss how this study plans to meet this aim, it is worth considering what is meant by relevance. At its basic level, we could define this term to mean 'of use to practitioners'. Benbasat and Zmud (1999) suggest that this can be achieved by selecting appropriate topics that are of interest to and develop outputs that can influence practitioners, and then write up findings in a way that encourages their implementation. However, as Lyytinen (1999, p.26) points out, relevance goes beyond simply 'something that suggests immediate solutions to CIOs and that they can digest in one afternoon'. Rather, areas of significant impact are achieved through a complex, intellectual debate. The danger of seeking relevance in its simplest sense is that researchers only describe what happens in practice. This may be useful in encouraging

the dissemination of innovative practice (Lee, 1999), but often does not add to our understanding nor encourages radical changes in thinking.

In the IS research field there has been a move towards discussing how to research as well as what to research (e.g. Boland and Hirschheim, 1987; Nissen and Klein, 1990; Cotterman and Senn, 1992; Mingers and Willcocks, 2004). With little history or tradition to fall back on the discipline has leant on a multiplicity of reference disciplines for its inspiration, methods and underpinning theory. Many paradigms and methods have been adopted over time (see Galliers, 1992 for an indicative taxonomy). This study will adopt an critical interpretive, longitudinal case study approach, drawing on social theory within the analysis. The remainder of the chapter will outline the design of research for the case study, showing how the choices were necessary to aid our understanding of the concepts under investigation and to maintain the relevance to others.

## 3.2 Research Design

### 3.2.1 Selection of Strategy

Robey (1996) shows that it is important to design the research according to its aim. This study will develop an explanatory theory of the SPI change process based on the experiences of a specific IS group. It will focus on how and why those changes occurred and evaluate their efficacy. In order to study a given group within their context a number of options were available, including field experimentation, action research and different forms of case study research. Both field experimentation and action research assume that the researcher can impose a theoretical position upon the group being studied. Field experiments take laboratory experiments into a more

realistic environment, but the narrow form of an experiment would exclude many of the factors that shape SPI activities in practice. Whilst action research can be an effective form of research that encourages praxis (Baskerville and Wood-Harper, 1998), it too seeks to control variables and places the emphasis on the action researcher to bring about change within the context rather than studying the organisation to understand the events in their own right. In contrast, case study research does not seek to control or manipulate variables (Cavaye, 1996).

Case study research can be seen as an overarching research approach or methodology (Cavaye, 1996; Benbasat et al, 1987). The term case study has been derived from sociological and anthropological studies, as an in-depth study of the individual, group, event or feature being investigated. These studies incorporate a selection of methods or techniques, styles of research and underpinning philosophy. Case research consists of detailed investigations, often over time, with one or more groups or organisations. The aim of case study research is to reconstruct and analyse the case from a given perspective, so that the case is understood on a different plane to the purely descriptive level. It is particularly useful where:

- the theory and understanding is poorly developed;
- there are numerous variables and the relationship between them is unclear;
- the boundaries between the phenomenon and the context are not clear;
- multi-disciplinary and multi-method research is required.

(Yin, 1994; Aitken et al, 1996)

Case study can be used for exploratory theory building or explanatory theory testing (Yin, 1994). A theory building approach is appropriate in a situation where the existing theory is weak or missing; theory testing process is used where existing

theory is being proven or extended. To develop a theory of SPI as emergent change it is necessary to investigate the phenomenon so that it is contextually situated, thereby allowing the variables in the case to be interpreted as it unfolds. Also, as this work breaks out of the existing engineering mould, an inter-disciplinary study is required as shown in Chapter 2. The IS field has seen a change of focus from technological issues to managerial issues, resulting in more interest in the interaction between the context and innovations (Benbasat et al, 1987). Case study is well suited to empirically investigating such issues because the phenomena under investigation is not isolated from its context, and it is precisely for this reason that it is of interest (Hartley, 1994). By not separating the components from related matters a holistic examination is possible. The methods adopted are ideal for capturing the knowledge and views of practitioners, with longitudinal case studies helping to show how the contextual relationships develop and evolve. Where the implementation of IS methods and solutions requires us to see how the situation develops over time 'there is no choice but to adopt in-depth case studies for this type of work' (Walsham, 1993, p.248). This type of study enables the researcher to describe the empirical findings in detail and draw out specific events and interactions that are key to building our understanding of the changes occurring. Section 3.3 furthers this discussion by showing how the case study method was applied, and how the limitations of this method were addressed. Having selected a case study approach, alternative philosophical stances and methods are still possible. The remainder of the chapter will describe and justify the approaches adopted.

### 3.2.2 Philosophical Stance

The question of the assumed epistemology and ontology needs to be part of the consideration of the qualitative researcher, as they can adopt different philosophical

stances. Whilst there are a rich variety of possibilities, the various approaches are simplified largely into a dichotomy between positivist and interpretivist views. In the early 1990s, Orlikowski and Baroudi (1991) found that positivist approaches dominated the research in the IS field, with as little as 5% of the work published being classified as interpretive. The reason for the dominance of positivism is that the approaches are often considered synonymous with quality. Even when it is not justified, rigour has often been associated with research that has emanated from the positivist philosophical perspective, namely that using quantitative and deterministic methods. This predominance has resulted in much sterile research because the hypotheses being tested were trivial and lacked relevance to practice (Fitzgerald and Howcroft, 1998; Benbasat and Zmud, 1999).

Despite these concerns, Lee (1989) suggests that case studies should be conducted under a natural science framework, i.e. a positivist approach. Positivist approaches assume *a priori* relationships within the phenomena. They are theory-led, seeking to test theory through the use of hypothesis testing, measuring variables and drawing inferences from the data (Olikowski and Baroudi, 1991). By taking an objective scientific approach researchers are seeking to adopt the methods of natural scientists in controlling the study, thereby enabling controlled deductions to be made from the study and the replicability and generalisabilty of the findings. However, to take a positivist approach is to make the assumption that research is neutral and that observers are impartial and detached from their study. This assumption is undermined by Orlikowski and Baroudi (1991, p.11) who state that 'information systems research enters into the very constitution of the phenomena it studies'. Whilst neutrality could be argued for distant, survey work or controlled experimentation, it is impossible to

achieve in engaged studies such as this one. The emphasis on the activity of interest can subliminally affect the decisions and processes, thus case studies designed to be objective are flawed from the outset. So the view adopted here is that whilst case study methods can vary in the extent of their engagement with the subjects, at some level there will be a degree of influence and interaction.

In response to these concerns, and a desire for more relevant research, interpretive approaches have gained more prominence in IS over the last decade, with 18% of published work now classified as interpretive (Mingers, 2003). The recognition that social issues are important in the development and implementation of computer based IS has been a primary reason for this change (Walsham, 1995). Interpretive research is particularly applicable in complex real-life situations and can give a deeper understanding of the underlying processes of organisational change (e.g. King, 1996; Orlikowski, 1993; Walsham, 1993). In contrast to positivist approaches, interpretive research does not look to falsify existing theory, but draws from the literature to develop the concepts to be studied. The epistemology of anti-positivism is 'firmly set against the utility of a search for laws or underlying regularities in the world of social affairs' (Burrell and Morgan, 1979, p.5). Interpretive studies therefore refrain from causal hypothesis because this would imply that the findings could be used for predictive purposes.

Interpretive approaches assume that reality is subjective and constructed by the human actors. Access to reality is therefore only possible through the interpretation of social constructions such as language, consciousness, shared meanings and instruments (Myers, 1997). According to this view, people create and recreate their own subjective

and intersubjective meanings as they interact with the world around them. Researchers therefore seek to understand how actors, through their participation in a social process, give meaning to their particular realities. From this we can understand how 'these meanings, beliefs and intentions help to constitute their social action' (Orlikowski and Baroudi, 1991, p.13). Nadhakumar and Jones (1997) suggest that studying social phenomena from an interpretivist perspective requires the researcher to interact closely with the phenomena and the human subjects (or actors). They draw from the work of Giddens (1993) to show that it is not just the researcher who is involved in interpretation but the subjects too resulting in a 'double hermeneutic'. Whilst some may be concerned at this layered interpretation, the analysis of interpreting subjects is the role of a researcher (Alvesson and Sköldberg, 2000). So, as the human actors' interpretations will be influenced by the context in which they are placed, we need to move away from looking at isolated issues to looking at people and processes in context (Pettigrew, 1979).

Interpretive studies have been criticised because the outcomes were based on the intrinsic experiences of the social actors in the study (Hamel, 1993). It is therefore difficult to validate the theory emanating from the work, and by implication any theory developed cannot be generalised. In other words how could one case be truly representative of the whole? From this perspective, the same physical or socially constructed artefact can have a different meaning for each person. This leads to the criticism that it fails 'to explain the unintended consequences of action, which cannot be explained by reference to the participants and which are often a significant force in shaping social reality.' (Doolin, 1998, p.302). Yet, Giddens (1979, p.59) claims that the unintended consequences of actions is of 'central importance to social theory'.

Another criticism is that such studies tend to lack any temporal dimension in understanding change. Pettigrew (1990a, p.9) argues that much research in organisational change is 'ahistorical, aprocessual and acontextual'. Walsham (1993, p.5) points out that process dynamics need to be addressed 'by taking seriously the transformation and change which take place over time'. So whilst interpretive approaches offer advantages over positivist ideas in the engaged nature of case study work it is necessary to address the concerns outlined. The criticisms arise because interpretive researchers understand the world from the perspective of those being studied without critically analysing those views.

An interpretive approach combined with critical theory can develop a deeper understanding of the phenomenon under investigation (Thompson, 1981). Dialectical hermeneutics, whilst based on the concepts of hermeneutics and critical theory, is different to pure hermeneutics and to critical theory in the Marxist tradition. Hermeneutics is concerned with the analysis of the meaning of a text, or text-analogue. Although the traditional use of hermeneutics comes from theological analysis of written texts, the concepts have been transferred to organisational analysis and more specifically IS development. To do this we need to understand the case study as a 'text'. Hermeneutics is used to explore the socially constructed contexts of organisations. This is done through interpreting the underlying sense from the ethnographic data gathered through interviews, observation or documents. The hermeneutic circle is a dialectic between the understanding of the whole text and the interpretations of the part (Gadamer (1976) cited in Myers, 1994a). This is a reflexive, iterative approach to deriving meaning, from the historical context of the phenomena.

Dialectical hermeneutics builds on the theoretical position of 'pure' hermeneutics but also recognises the dialectic between text and interpreter. Texts (e.g. documents and interviews) can be interesting for what they leave out as well as what they say. A 'pure' interpretive approach will not address these gaps. Hermeneutics encourages us to move away from the view that documents simply report social reality. Written texts can be seen to have authority, as they can be viewed as the authorised version of knowledge (Olsen, 1980). We can 'utilise our own cultural understandings in order to 'engage' with 'meanings' which are embedded in the document itself' (May, 1993, p.138, emphasis in the original). This engagement encourages the dialectic between the researcher's own frames of meanings and those found in the text. A dialectical approach understands that knowledge is based on practice, and that practice is constantly changing. We must understand what the changes are and why they take place, through revealing the content of the actions and the way they change over time. Dialectics focuses on process rather than structures. Structures are considered as 'complementary abstractions' (ibid, p.23), reflecting Giddens' (1984) view of them as traces of the mind.

Dialectics also focuses on relations rather than entities, helping to understand social phenomena as they relate to each other in a particular context. The contradictions and the relations that constitute them are highlighted. Whilst dialectical hermeneutical approaches are critical in the sense of being reflexive, there are differences to most critical theorists who have focussed upon a 'critique of class-based societies and capitalist forms of production' (Myers, 1994a, p.57), but it is possible for researchers to be reflexive without working from a socialist political stance or a realist ontological

view. By implication 'it would seem reasonable to conduct research from an emancipatory cognitive interest which critically interprets various empirical phenomena, with the purpose of stimulating self-reflection and overcoming the blockages of established institutions and modes of thought' (Alvesson and Sköldberg, 2000, p.128). A researcher therefore does not have to assume from the outset what the important oppositions, conflicts and contradictions are in the organisation (Myers, 1994b). Thus, to be critical we need to challenge the positions that are taken for granted, the perceptions and values of the participants and our own pre-constructed perspectives.

Any attempt to combine research founded on mutually exclusive ontological, methodological, and epistemological views, however, will be the subject of criticism that such synthesis is not possible because the underlying beliefs are contradictory. The basis of this criticism is that, as Jones (1999b) points out, if researchers are philosophically inconsistent they have to hold different views of the world concurrently. This argument is based on the seminal work of Burrell and Morgan (1979) who state that linguistic symbols have different meanings in each paradigm, thereby making it impossible to translate between paradigms. Accordingly, their conclusion is that research based on different philosophical views is incommensurable. This argument is flawed because it is based on Kuhn's original concept of a paradigm that was less dogmatic than their interpretation (Allen and Ellis, 1999; Mingers, 2001b). Mingers (2001b) argues that any paradigm is a human construct and provides us with a different perspective and that we need to adopt a pluralistic, multi-paradigmatic approach to address a complex reality underpinned by a 'critical pluralist' philosophy.

In line with this view, Myers (1997) cites Bernstein (1983) to argue that there is common ground between the critical theory of Habermas and the hermeneutics of Gadamer. Whilst critical research has been identified as a third category of research philosophy separate from positivist and interpretivist approaches (Orlikowski and Baroudi, 1991), Ngwenyama (1990, p.271) states that this category 'spans the objective – subjective spectrum'. Critical theorists, like interpretivists, consider that the social world is constructed by the human actors, but acknowledge the physical and organisational structures that may be drawn upon or that can constrain actions. Giddens (1984) also considers structures to be drawn upon, but strongly emphasises the enabling as well as constraining aspect of the structures and that they are only 'real' at the moment of action.

The advantage then of a dialectical hermeneutic approach is that it enables the portrayal of the complexity of organisations as social, cultural and political systems (Myers, 1994a). It therefore suits the purposes of this research project enabling a critical, explanatory theory to be developed. Dialectical hermeneutics is also proposed in order to understand the influences, both realised and hidden, and the consequences of any action, both intentional and unintentional, thereby addressing the first area of criticism of interpretive studies.

The other area of criticism, related to the nature of change, can be partly addressed by looking beyond the historical development of the software process and taking a critical look at the organisational and political structures that form the context. The context is, therefore, important in understanding the situated change process. To

understand that context, researchers need to understand the organisation at different levels of analysis, i.e. taking sectoral and socio-economic context and change into account not only the local context. So, as well as ensuring that the perspectives of the different participants in the study are understood, the context will be extended 'into the larger economic and societal framework within which developments occur' (Doolin, 1998, p.303). Pettigrew (1990b, p.273) also argues that to study change we have to do this over time because the 'longer we stay with an emergent process and the further back we go to disentangle its origins, the more we can identify continuities', which requires the researcher to spend time in the field investigating these factors.

To support the understanding of the interplay between the context and the actions in this case study, a frame of reference has been developed that uses Structuration Theory (Giddens, 1984) as the underpinning view of social action. Structuration Theory, however, is not intended as a concrete research programme or methodology. It is therefore necessary to identify a suitable approach to support this social theory, hence the combination with dialectical hermeneutics as the way to achieve a richer, more integrative view of IS development (Myers, 1994a). This strategy is followed here to develop an explanatory theory of the SPI change process based on the experiences of a specific IS group. The link between Structuration Theory and dialectical hermeneutics is justified below by considering how they address the criticisms of the general interpretive research described above.

Giddens strongly emphasises the context, or social structures, as both enabling and constraining. The emphasis of both the subjective meaning of the human actors and

the social structures, which shape their meanings and actions, is a central reason for selecting both these frameworks for studying the changes associated with information systems. By extending the interpretive paradigm to include thinking from the critical theorists, as dialectical hermeneutics does, it is possible to understand the constraining and influencing aspects of the context and to analyse the result of actions beyond the intentions stated by the actors. This combination could be criticised because Giddens (1999) suggests that to look for the reasons for actions of which an actor is ignorant is a mistake, but he does recognise that the rules that shape people's action are part of the structures that are reproduced by the actors, often unconsciously.

Structuration Theory also addresses the temporal dimension of organisational change. Giddens (1979) considers that much social analysis to be flawed as it treats social interaction to be a snapshot in time. He (ibid, p.202) states that 'patterns of interaction that exist are situated in time' because only by examining them over time are any patterns formed at all. Structuration Theory has been described as a process theory, encouraging an emergent perspective of action that is inherently situated in its context (Jones and Karsten, 2003). Myers (1994a) argues that dialectical hermeneutics, also, is a constitutive process theory that emphasises 'both the importance of subjective meaning for the individual actor and the social structures, which enable and condition such meanings'(p.55). He shows that constitutive process theories provide a way of researching the social and contextual aspects of IS.

By combining these theories and research approaches the situated action of the SPI programme within the case can be analysed through time, highlighting the emergent nature of the software process changes and the context that they occur within. The

enabling and constraining features of the context will be highlighted to show how they shape the action, and in turn are reproduced through that action. The reasons for the changes will be drawn out to identify the intentions, both stated and unconscious, and the outcomes intended or unintended to help to understand why the changes occur and their efficacy.

So, the emphasis of both the subjective meaning of the human actors and the social structures, which shape their meanings and actions, is a central reason for selecting this strategy for studying the changes associated with IS. It allows contextual factors to be considered in the analysis. By moving beyond a simple description of the incidents within a case study to challenge the existing literature the approach is vital to facilitate findings that are relevant to a wider body. The following sections will highlight the selection of the case study and the specific approaches used, detailing the methods applied and show how these relate to the dialectical hermeneutic perspective.

## 3.3 Case Study

### 3.3.1 Case study approach

Following the identification of the strategy above, it was necessary to add to Myers' (1994a/b; 1997) work that justifies the reasons for applying dialectical hermeneutics in qualitative IS work but does not specify how to undertake the case research and analysis. The methodology for this case has been designed following Pettigrew's (1990b;1997) views on longitudinal study and processual analysis, and Eisenhardt's (1989) ideas on developing theory from case study research. The analysis for the case

utilises Miles and Huberman's (1994) work on qualitative data analysis. The following sections will explain the reasons for selecting these approaches.

Eisenhardt's (1989) roadmap of how to undertake case study research (see Figure 4) was used to structure the overall stages of the research. She suggests that an initial set of research questions is important in building theory from case studies. Without a clear focus, it is easy to become overwhelmed by the volume of data. Yin (1994) suggests that case studies are suited to how and why questions. The questions should be also of interest to both the research and practitioner communities to enhance their relevance (Benbasat and Zmud, 1999; Darke et al, 1998). The questions outlined in the introduction address a gap in the theoretical understanding of how SPI occurs within its context, and thereby inform future practice.

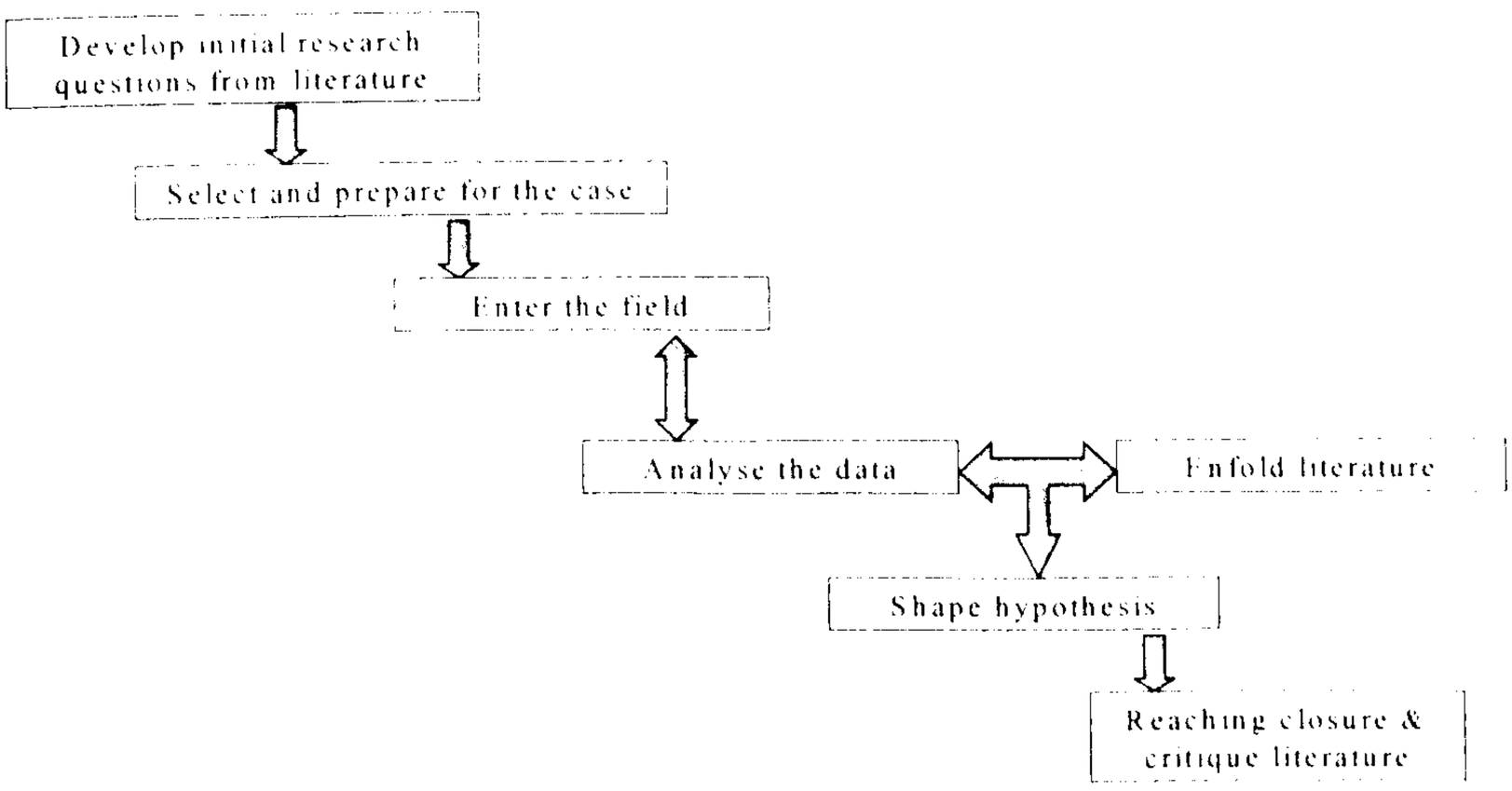



**Figure 4 Case Research Methodology. Adapted from Eisenhardt (1989)**

Interpretive studies do not predefine hypotheses or the variables but focus on the complexity of human sense making. However, critical interpretive approaches accept

that researchers have pre-constructed perspectives. The identification and reflexive criticism of the pre-constructed view is an important part of a critical interpretation. The way in which the researcher's worldview influenced the development of an a priori specification is defined here, and discussed in further depth in §3.6.2.

An *a priori* specification of constructs can help in the building of theory because they direct attention to those things considered relevant to the study. By challenging this position against the meanings of the participants the researcher's view is recreated. Yet, any use of early constructs should be applied cautiously, recognising that they are tentative. Indeed, Eisenhardt (1989) suggests that it is important that any pre-existent theoretical framework is minimal. Like in Soft Systems Methodology, where Checkland (1981) highlights the need to avoid imposing systems too early, Eisenhardt argues that researchers should formulate the problem and identify some relevant variables, but should avoid firming up on the relationships early in the process. This ensures the theoretical understanding being built up is open to the novelty and nuances of the interpretations received during the study.

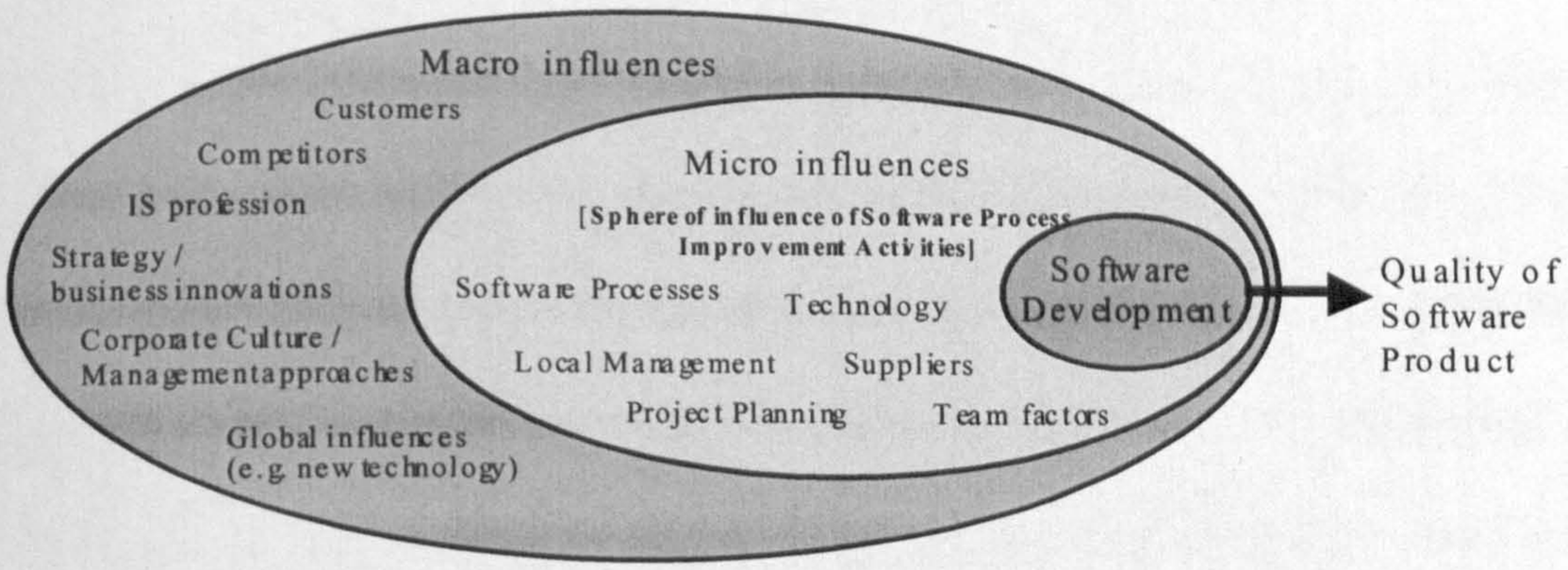



**Figure 5 A schematic of the contextual factors influencing the software development process**

An initial framework was used here to identify contextual factors that affect the software quality and the process initiative. The framework is shown as a simple schematic in Figure 5. This schematic identifies the main contextual factors from the software engineering and IS literature and attempts to categorise them into two groups: those within the sphere of control of the team and those external to it. The view that the software product is dependent upon the processes used is clearly informed by the software engineering literature discussed in Chapter 2. From the reading of that literature it was anticipated that the organisation would adopt a staged approach to maturity akin to the Capability Maturity Model, with a strong emphasis on measurement and causal analysis. Additionally, the commercial experience of the researcher influenced the view that the process innovations would be context dependent and influenced through the intervention of senior management. Other IS literature confirms the influence of the context and a management driven view was

supported by the SPI literature that states the need for senior managers to champion the changes.

Data collection occurred over a year working with the company as a participant-observer, as explained in §3.4.3. During the data collection and analysis it was recognised that the relationship between the context, the SPI activity and the software development was not a simple linear one, but intertwined and interdependent, as discussed later. By systematically reviewing the evidence from the case, the initial framework was adapted to reflect the themes and relationships found. The interpretation of the data, with the literature enfolded, has shaped the theoretical framework to the one used for the final analysis (see Chapter 5). This was a highly iterative process, during which the definitions of constructs and variables were carefully managed. It was necessary to reanalyse the data in the study in the light of changes to the framework. In light of the recommendations of Benbasat and Zmud (1999) the framework was designed to be intuitively meaningful to practitioners to enhance the communication, and relevance, of the findings to them.

### 3.3.2 Case Selection

A single-case strategy is adopted here with the emphasis on the richness of the data collection over a significant period. The decision to adopt a single or multiple case strategy is central to case study research projects. Benbasat et al (1987) suggests that single-case projects are most useful at the outset of theory generation. The single case can be used for exploration and to develop theory, and followed up later with further cases, surveys or other approaches to test the theory developed. There are many exemplars of single-case research in IS (e.g. Markus, 1983; Myers, 1994b;

Orlikowski, 1993; Currie and Willcocks, 1996; Nicholson and Sahay, 2001). Each of these cases enables an in-depth understanding of the phenomena in question.

Walsham (1995) argues that this form of case study assists in drawing out the details of the events. Benbasat and Zmud (1999) suggest that such cases often prove to be effective means for communicating conceptual developments to practice. The alternative of spending a few weeks with several organisations does not allow relationships to develop sufficiently to create the deep familiarity necessary to overcome the distortions and misrepresentations given by participants (Nadhakumar and Jones, 1997). Multiple cases are appropriate where there is a clear comparison to be made (Yin, 1994) or to develop theoretical saturation (Glaser and Strauss, 1967). However, the danger of multiple cases is that they can only focus on one moment in time, unless there is a team of researchers, and thus tend to lose the temporal nature of the change which is considered important in this case. Longitudinal studies as suggested by Pettigrew (1990b) are naturally time intensive studies, but do allow the researcher to follow the process of interest 'in flight' and capture the emergent nature of change through time.

Rather than random sampling, the case was deliberately chosen for properties relevant to the study and where the issues were most likely to occur, i.e. the case was chosen for theoretical rather than statistical reasons. This purposive sampling allows the generalisations from the case to be made in terms of the theoretical propositions not the entire population of cases. The case organisation, InfoServ, is a multi-national, information solutions company. The company and participant names have been

changed to retain confidentiality. The company is described in more detail in Chapter 4. They were selected because:

- they hold a market-leading position in their domain and therefore have a proven high performance capability in their field;
- the group studied are primarily a commercial software producer with a predominance of packaged information systems products, so the study could highlight issues related to IS vendors;
- they were undertaking an SPI programme at time of the study, providing the opportunity to record the events as they occurred;
- moderate prior contact with the company facilitated access and communication with participants (see §3.3.5), and helped to identify the issues to be studied.

### 3.3.3 Timeframe

The period of study was selected to capture a perspective of the problems and the events as they unfolded. The opportunity to collect data over a significant period of time was important to examine in detail the actions, intentions and perceptions of the human actors, the context in which these actions took place and the consequences of the actions. The in-depth observation and study spanned the period April 1998 to September 1999. Prior contact had been established since early 1997 and visits continued to 2003. The data collection also retrospectively established the historical background of the study, especially the period 1993 to 1997. The case study is split into chronological periods. These periods are not intended as distinct stages of development but are provided to simplify the discussion. The key episodes were

related to product and process developments, and were selected to mirror naturally occurring episodes within the case (see Table 3).

| | Period | Product development | Process Improvement |
|---|---|---|---|
| Pre-study | Pre 1995 | Establishment of the DOS-based SPECTRUM Systems | Haphazard use of processes for each development |
| | 1995 - 1996 | Redevelopment to produce Windows Market Analysis Package version 1 | Formalisation of a 'standard' process for the MAP product development |
| Main study | 1997- 1999 | Enhancement to Windows Market Analysis Package version 2 and product diversification | Refinement of the formalised process for the MAP software |

**Table 3 Significant Periods within the Case**

### 3.3.4 Unit of Analysis

The unit of analysis (UoA) is what constitutes the area to be studied. To achieve the research aims, the UoA should provide sufficient breadth and depth of data to be collected and should be related to the research question(s). Darke et al (1998) suggest that it can be an individual, a group, an organisation or it may be an event or some other phenomenon. To bound the study tightly based on organisational structure implies a fixed view of the study from the beginning. Miles and Huberman (1994) suggest that we identify the 'heart' of the case and build outwards into the context.

Given that the focus of this study is to understand the process of improving the software process, rather than making the UoA a specific organisational group, it is appropriate to make it the continuous software process improvement activity in context (Pettigrew, 1990b). So, the SPI activity was chosen as the UoA to give a theoretical boundary to guide the data collection and analysis. This allows the case to be compared at a later date with another improvement initiative, even where the organisational forms are different.

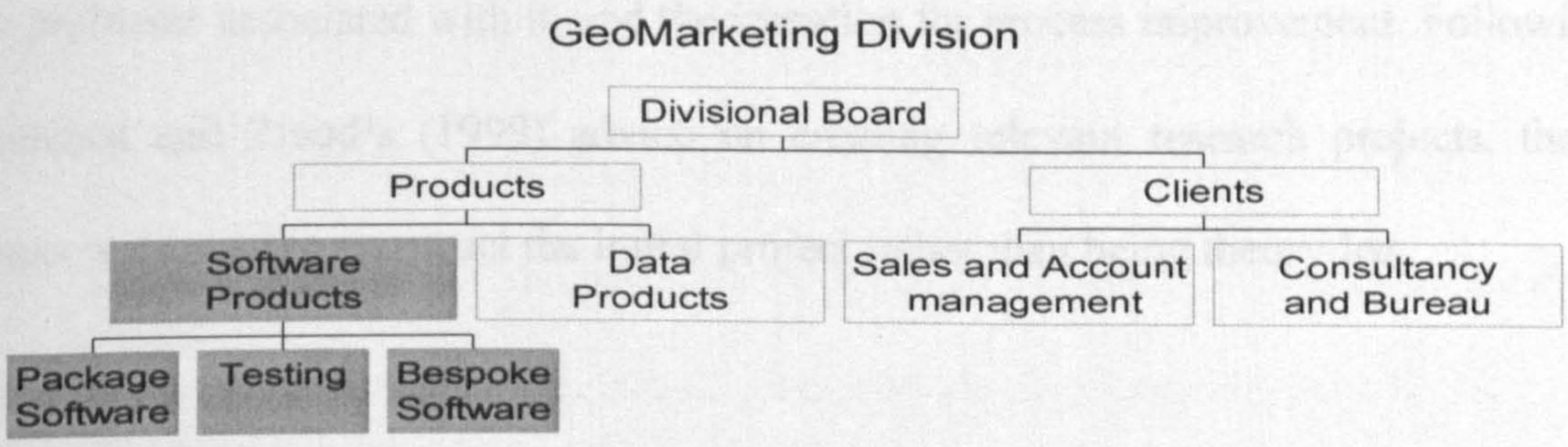



**Figure 6 GeoMarketing Division**

So whilst the research is primarily focused on the package software development group, the SPI activity permeated each of the teams within the software product group (see Figure 6). So, it is the activity of change to the software process within that context that forms the UoA. By restricting our view to the process improvement within the one group, it is necessary to be clear that this limits the findings to an understanding of what occurred within that group. Other process improvement initiatives elsewhere in the company are not incorporated into the data unless they

directly impinge upon the work of this development team. Further details of the group and the improvement activity are given in Chapter 4.

### 3.3.5 Access

Prior to the beginning the continuous, in-depth study of the case during 1998 and 1999, the study site was visited on six occasions. Two initial visits were made in an academic role to monitor the progress of undergraduate students on work placement. It was during these visits that the issues to be investigated in this case were initially identified. The IS management explained the current software development activity, the problems associated with it, and the intention for process improvement. Following Benbasat and Zmud's (1999) advice on creating relevant research projects, these issues were used to construct the initial project rather than being theory-led.

Following these initial visits a short review of the software management techniques was undertaken during the period July to September 1997. The review involved discussing approaches with key members of the development group, attending a small number of selected meetings, and evaluating software plans. A brief report of the findings was provided for the software management.

Whilst this contact could be considered to have biased the research, it assisted the access and relationships with the participants of the study. In order to facilitate more appropriate questioning in interviews, Darke et al (1998) show that researchers should arm themselves with prior information about the site. Any bias that existed in the research design and findings was a result of the prior knowledge and experience of the researcher, rather than the short visits to the company. The review could have influenced the actions of the development group. However, the IS management were

already instigating improvements, indeed the review had been done at their request. So, in line with dialectical hermeneutics, it is recognised that bias in the researcher and the participants is inevitable. This prior contact is not considered to have unduly influenced the study.

Access was negotiated with the software director and approved by the divisional Board of Directors. No limits were placed upon the contact. As Pettigrew (1997) states many people are willing to open their doors to researchers providing there is some form of reciprocity in the activity. It was agreed that, where relevant, information would be given to team members about techniques and ideas they were interested in, that assistance would be given in organising SPI review meetings and an end of project report would be provided. In order to recognise any bias arising from the reciprocal arrangement, further details of these interventions are given in 3.6.2.

## 3.4 Data Collection Phase

### 3.4.1 Data Collection Approach

The choice of data collection methods is partly dependent upon the access the researcher has and the degree of engagement with the subject that is required. This case study, in line with the longitudinal approach, is a prolonged study to examine changes over time. To undertake this type of research we need to use engaged methods to penetrate the world of the actors. Yet, whilst there has been a rise in the use of qualitative methods, Nadhakumar and Jones (1997) found only 16 percent had used engaged methods. This study was carried out using an 18 month participant-observation approach. This engaged research was undertaken using a variety of data sources, allowing contrasts and congruencies between and within social groups to be

identified by taking the different views into account. The following sections will explain the application of each of the methods.

### 3.4.2 Interviews

Two forms of interview were used: a set of formal semi-structured interviews and a set of regular reviews with two of the IS management. The formal interviews covered all the development staff within the software development team, their line management, and associated staff (other IS staff and related non-IS staff) (see Table 4 for the numbers). A full list is provided in Appendix 1.

| Categories of staff | Number formally interviewed |
|---|---|
| Developers (including testing team and project manager) | 15 |
| Directorate (including IS director) | 5 |
| Other IS staff in Division (e.g. help desk manager) | 3 |
| Other non-IS staff in Division | 6 |
| Non-divisional staff | 2 |
| Total staff interviews | 31 |

**Table 4 Number of interviews by group**

The formal interviews, with two exceptions, were tape-recorded and transcribed afterwards. Those that were not recorded were 'non-IS staff' members who were unwilling to have the interview recorded and where the interview was written up

immediately afterwards. There are both limitations and advantages to recording interviews. Tape-recording can inhibit interviewees and researchers can become over reliant on the recording (Darke et al, 1998), but they also free up the researcher to listen to the interviewee. Tape-recordings allow the researcher multiple opportunities to reanalyse the same texts, drawing out new meanings and understanding, thereby reducing the likelihood of misinterpretation. A database of tape-recordings was maintained, enabling the researcher to re-access the interview in oral form to ensure nuances were interpreted appropriately.

A semi-structured approach was adopted for the more formal interviews. For each category of staff, a standard interview question instrument was created to enable consistency and avoid missing items (see Appendix 2). However, these sessions were not conducted in a rigid fashion, but were similar to what Patton (1990, p.288) describes as an interview guide approach: 'topics and issues to be covered are specified in advance, in outline form; interviewer decides sequence and wording of questions in the course of the interview'. One advantage of this approach is that it allows for a systematic coverage of the material using a more conversational style: improving on the comprehensiveness of the data through the openness of the interview, and allowing the interviewer to respond to any gaps, misunderstandings or follow up required. Whilst it could be argued that this makes the research less standardised and different wording can result in different responses from interviewees, these are minor risks compared to the risk of stale interviews that have less informed results. Attempting to remove this interaction would be futile because it makes the interview more 'interpretively active' (Holstein and Gubrium, 1997, p.114) and allows the researcher to challenge the perceptions and values of the respondents

consistent with a critical interpretive study. It therefore becomes a 'meaning-making' occasion (Holstein and Gubrium, 1997, p.114). The risk of misinterpretation of viewpoints is overcome through the multiplicity of data sources and length of the relationship with the group studied.

For interviews to work effectively there needs to be a level of trust between the interviewer and the interviewee. This is achieved in part by building a rapport and assuring confidentiality (Miller and Glassner, 1997). The advantage of the prior and ongoing contact was that it assisted in the building of trust. A mutual understanding of the social world helps to build understanding and enables the interviewer to respond to the items raised in such as way as to make the interview active rather than passive, thereby adding value to the basic interview instruments that had been created. The assurance of confidentiality within the organisational context was explicitly stated prior to each interview. Whilst it is possible to relate items back to individuals from this thesis names are protected as far as possible. Each interviewee was offered the option of a non-taped session.

The project manager and the software director were key people in the management of the software and the process. Twenty-seven review meetings were held with them throughout the period of study to assist in understanding the actions and decisions as they occurred. These were informal events and were mostly not taped. The meetings were based on a generic structure. Primarily the purpose was to discover their intentions and reactions to what was happening in the product development and process improvement activities. The questions emerged from the context of the

current events, so there was no predetermined set wording or detailed topics. Notes were taken during the meeting and written up shortly afterwards.

### 3.4.3 Participant-Observation

Klein and Myers (1999) consider that the researcher needs to place themselves into the historical context to gather the data. The facts of the case are a product of the interaction between the researcher and the sources or subjects. Interviews are a limited option as the facts are interpreted initially by the interviewee and then again by the researcher. Silverman (1998) considers that we get a better understanding by examining experts in situ rather than relying on interviews. Nadhakumar and Jones (1997) draw from the work of Giddens (1993) to show that the human actor's interpretations will be influenced by the context in which the actor is placed. This is why we need to move away from looking at isolated issues to looking at people and processes in context (Pettigrew, 1979).

Engaged studies allow the observer to participate in the everyday lives of the observed. Atkinson and Hammersley (1998) categorise the approach used here as observer as participant. The researcher watches, listens and asks questions over a significant period of time. In this case the researcher was given wide-ranging access, and drew on information from sources as required. The role was undertaken overtly, as would be expected for critical research, then this enabled open interviews to be conducted, attendance at meetings, access to a wide range of documents, notes to be taken immediately and interviews to be taped for later transcription. The nature of the research was not explicitly communicated to the informants, but they were aware that the research was related to the software process improvement programme.

Jorgensen (1989) shows that participant observation also provides access to the insider's world of meaning. From the perspective of the participants, the researcher can describe what goes on, who or what is involved, when and where things happen, how they occur and why. Nadhakumar and Jones (1997) argue that this enables the researcher to develop a more holistic response to the events, but highlight the need for the researcher to remain reflective and critical.

The degree of participation can vary. Some participant observers adopt the position of a member of the group under study. So in a software team they would also produce software, or on an assembly line they would work alongside the other assembly workers. Here, however, the researcher remained an outsider to the daily functions of software development, yet participated in meetings about the software development and process improvement projects, and became more involved when requested by team members and management (see §3.6.2). Whilst the informants became acquaintances and there was a level of familiarity in conversation, the position of the researcher as outsider remained clear throughout. However, by being involved in these meetings, training, and discussions with the actors, a deeper awareness was developed than would have been the case if the work had relied on interviews or been undertaken externally.

Following the advice of Fox (1990) a journal was used to note thoughts, observations, ideas, events, etc. Eisenhardt (1989) suggests writing down whatever impressions occur, these can be picked up or discarded later as patterns emerge. The journal is a combination of the collection of data, analysis, the formulation of concepts and tentative theory. During the study two volumes were completed. The journals were

used as data for the data analysis, as well as containing ideas that shaped that analysis. This is one example of where the overlap of data collection and analysis is evident.

### 3.4.4 Documentation

| Document type | Information gathered |
|---|---|
| Software development output | Sample requirements, analysis and design documents |
| Software development plans & strategy | Project Gantt charts<br>Work-breakdown and estimation<br>Product strategy documents<br>Risk analysis documents |
| Process information and improvement actions | Definitions of the software process<br>Output from the action teams |
| Quality assurance information | Standards<br>Group quality manual |
| Non-software documents<br>Product and in-house publications<br>Financial and company data<br>Trade and financial press | Information about the market and the competition, customers, technology and global influences. Used to provide background and understand external influences. |

**Table 5 Documents gathered from the case study**

Documents were collected from the company in two principal categories: those specific to the software development and process activities, and those general to the

organisation and its products. Some of the items in the latter category were from external sources such as trade press. The items collected are summarised in Table 5. These provided information about the activities of the software team, their stated intent, and the outcomes

### 3.4.5 Software Process Data

In addition to the documents outlined above, internal data collected by the software group was also acquired to assist in the analysis of the perceived efficacy of the SPI activities. Unlike in traditional software engineering texts, these measures are not taken as an objective and accurate calibration of the organisation's work. The perception of the data, as well as the numbers themselves, is taken into account. The types of data collected are listed in Table 6.

| Data | Notes | Purpose of Collection |
|---|---|---|
| Help Desk Calls | Data over time and type of problem | Perceived as an external measure of service quality |
| Defect Metrics | Data over time and type of problem | Perceived as an external measure of quality of the software products and changes from the SPI actions |
| Quality control data | Information about defects found in software inspections | Perceived as an internal measure of quality of the software products and changes from the SPI actions |

**Table 6 Software Process Data**

### 3.4.6 Team Questionnaires

A survey of the team was undertaken to assess perceptions about the software quality and the changes that had occurred in the software process. The survey was taken in November 1998 and was based on McGuire's (1996) questionnaire. McGuire's

original study examined the effects of a SPI effort on a software development team. The questionnaire was targeted at those directly involved in the development and maintenance of software products within the GeoMarketing Division. Twenty-five responses were received, which was a 100 percent response from the whole of the software group. The survey captured the team's perceptions of its current strengths and the impact of the changes. The questionnaire used a Likert scale to capture the perceptions. The questionnaire was in two parts: the first about the team member and the second about the process and related factors. The second part asked for a comparison to a year previous, before the SPI project was initiated. A summary of the questionnaire response can be found in Appendix 3. The results provided supporting data to the interviews. However, given such a small sample a statistical analysis would not have been meaningful.

### 3.4.7 Summary

So, in summary, the data collection methods were designed to gather a variety of data to meet different purposes. The study has been based on a period of participant observation during which data was captured from across the software team and beyond. The data was captured by: recording observations in a journal; using a variety of interviews and a questionnaire to capture participants' views; collecting documentary evidence of events within the software development and process improvement projects, and from the wider organisational context; identifying metrics used within the team for managing software quality. The methods of capture were based on proformas, protocols, and well defined approaches in the literature to enhance the reliability of the collection. The following section shows how the data was analysed.

## 3.5 Data Analysis and Interpretation Phases

### 3.5.1 Stages of analysis and interpretation

In line with the philosophical stance outlined above, the analysis of this case utilised a critical hermeneutical approach whereby the text of the case was analysed to draw out thematic concepts. The various data sources were analysed to identify these patterns. To undertake the analysis, the overall case study approach explained in §3.3.1 was supplemented with qualitative analysis techniques. Miles and Huberman's (1994) expansive sourcebook on methods of analysis was used as the primary source because their own philosophical position is similar to that adopted here. In particular, they recognise that conveyed 'facts' are subjective and socially constructed, but look to establish explanations that go beyond the individual perspective. Their preference is for social anthropological studies that provide data about human activity that can be seen as a 'text'. Finally, in line with Eisenhardt, they recognise the importance of the interaction between formulating the concepts, data collection and data analysis.

Alternative examples of how others have analysed SPI cases are found in Bennetts et al (1998), and Fitzgerald and O'Kane (1999). Bennetts et al (1998) utilise aspects of Soft Systems Methodology as a framework for SPI. This approach helps to analyse the variety of perspectives within an organisation about any current problems and to identify the changes required. This methodology is used as a framework for an action research project. Similarly, to assist Motorola in their SPI activity, Fitzgerald and O'Kane (1999) undertook an action research project. They analysed the business needs by identifying critical success factors to be achieved by the organisation for each level of the CMM. By identifying these factors the company avoided the pitfalls

of addressing just the minimal requirements of the particular level of the maturity model and the case was analysed against each of these factors.

Both of these approaches are active processes of analysis that involve making decisions about the SPI priorities with the development team. These actors are therefore fully engaged in the SPI process itself rather than observing and analysing the activity. Both approaches are valuable for action research projects, or could be used where the level of participation is relatively high. Here though the intention was to analyse the actions undertaken by others and to apply these approaches would have significantly interfered with the natural events in the case.

The analysis stage for this research overlapped the data collection stage: the development of formative concepts and initial theories were documented in the research journal; the interviews were transcribed soon after they were recorded with some initial coding; and the literature was explored and enfolded during the period of engaged study. This overlap is a natural and expected feature of qualitative analysis. A significant proportion of the analysis stage, though, was undertaken after the data collection period. The stages of analysis after the data collection followed those suggested by Miles and Huberman (1994), including:

- Completing, organising and summarising the data into a usable database.
- Highlighting the key concepts and themes through coding, the creation of data displays to summarise the themes and events, and further enfolding of the literature throughout to assist in the development of the explanations.

The detail of how the analysis stages were undertaken and the techniques were applied is provided in the following sections.

### 3.5.2 Completing, organising and summarising the data

The data from this case was voluminous. An early step in analysis therefore was to organise and summarise the data into a usable form. After ensuring items and notes were legible and comprehensible, they were sorted into files ensuring the data was easily retrievable. The files were organised to ensure items were accessible, and copies placed in multiple sites and secured to reduce the risk of loss. Electronic and manual data were stored to ensure ready access to the data at any point. The materials were organised within each type of item. Appendix 4 shows how the different categories of data were stored.

The next stage was to develop summaries of the interviews and contacts, meetings and events, and periods of observations noted in the journal. The purpose of this was to reduce the volume and improve accessibility of the data. Primary coding and analysis used the full text versions, but these summaries helped to sift out the core features of the items and improved scanning across the case. Three types of summary sheets were developed from the ideas suggested by Miles and Huberman (1994): event; contact and document summaries. These highlight the main issues and salient features, summarise the data, and prompt for speculative theory and queries from the interaction (see Appendix 5). These speculative theories were used to develop the theoretical framework discussed in Chapter 5. The interaction between the context, the process improvement and the product development came through from a number of interview summaries and repeated incidents in the log.

### 3.5.3 Coding and concept development

Miles and Huberman (1994) suggest developing a predefined list of codes that are then amended from the data. To achieve this is it is necessary to have a clear conceptual framework. The initial framework that was used to support the interviews identified factors in the macro and micro software development environment (see Figure 5). The micro environment factors were those deemed to be within the control of the SPI activity of the project team. This framework was intended as a schematic of the different influences for discussion. This was not used as an analytical tool.

During the initial case study data capture it was thought that the process improvement activity could be interpreted using Roger's (1995) Diffusion of Innovation following Swanson's (1994) work. An analytical framework and a related set of codes were developed using ideas from innovation theory. However, after analysing a number of interviews and part of the observation journal, it was discarded because the data would need to be forced to match the framework rather than the other way round. Following this initial abortive attempt to code from a pre-defined framework, the concepts and themes developed were drawn from the data using an inductive approach. The principal elements of the inductive analysis were initial conceptualisation, iterative coding, further analysis of the literature, and the development of themes.

The initial conceptualisation began with rereading the materials to ensure familiarisation and making notes in the margins. During this stage the material was not simply read for factual data but to identify nuances and to reflect on what the significant patterns were through the study. As discussed above, the summarisation process helped in this conceptualisation stage by analysing the interview, journal,

meeting, and document-based data for patterns and themes. The reflexive nature of the analysis enabled the theoretical framework to mature. Starting from a deterministic view, the theoretical framework underwent a number of iterations during the data capture and analysis. These iterations increasingly reflected the dynamic, dualistic nature of the improvement. This approach to theory development is similar to Pettigrew's (1997) cycles of deduction and induction, which he claims is where the real creative process takes place. The iterative nature of the research approach is key to the success of case research (Klein and Myers, 1999).

The analysis stage developed through an iterative coding and pattern generation process. Codes were developed from the data and related to the new schematic framework (a list of codes is given in Appendix 6). This interpretation of the facts occurred through reflection about the data, as part of the construction of the theory. As described above in §3.2.2, hermeneutics is concerned with the analysis of the meaning of a text, or text-analogue, and in case study research is used to explore the socially constructed contexts of organisations (Berger and Luckmann, 1967). At InfoServ, this was done through interpreting the underlying sense from the ethnographic data gathered through interviews, observation or documents. It is a reflexive, iterative approach to deriving meaning from the historical context of the phenomena that is achieved by asking questions of the data, seeking common points, and looking for surprises and contrasts (Riley, 1990).This engagement encouraged a dialectic between the researcher's own frames of meanings and those found in the case.

The coding was done manually, writing in the margins. Computer aided analysis has become widely accepted because of the apparent speed at handling large volumes of data. Seale (2000) claims that it adds rigour. However, this claim reverts to viewing the counting of items as having equivalence with rigour. Whilst for multi-researcher projects a more consistent approach can be applied this is less of a problem for a single researcher project. There is a danger that the computer tool acts as a barrier between the researcher and the data, moving the researcher a step further from their data. The coded data was sorted into topics manually.

Further analysis of the literature was undertaken to review how the case supported and challenged the existing materials. A key part of the theory building process is to establish how the new theory supports, relates to or contradicts existing theory. By fully considering existing literature the final results from the study will be more robust, can be generalised to a wider body and will strengthen the body of knowledge in that area by tying together different aspects (Eisenhardt, 1989). It was from this enfolding of the literature that the view of the process as emergent and intertwined with the product development was identified. This emergent framework was developed using Giddens's Structuration Theory as the underpinning theory, as shown in Chapter 5. The adoption of a structurational perspective was heavily influenced by work within the IS field that analysed the development and implementation of IS as a form of organisational change using Giddens's work (Walsham, 1993; Orlikowski, 1992; 1993; 1996). However, this time the identification of their work followed from the data and had resonance with the tentative theoretical memos made in the journal.

The analysis identified patterns of data from different sources that pointed to common themes and facets. Memos, diagrams and data displays were used to portray and abstract the patterns and contrasts in the data.

Memos were used to write up ideas about the data and the relationships as they arose during the analysis. The memos did not just report data but tied together data into clusters to show how they represent a concept. An extract from a memo gives an example of this theory development that reflects an early view of the political aspects of SPI discussed in Chapter 5:

> At one level we can see SPI initiatives as an emergent, sensemaking activity. The individuals deal with each action and the formation of their position as a sensemaking act. But part of this sensemaking is to consider the political nature of the act: their own position; the implications on resources in their own function; the relative strength/perception of their function; career development; etc. So, considering this, actions are taken to further their own position.

Diagrammatical representation helped to visualise the relationships between aspects of the data. Diagrams varied from concept maps, that showed how aspects of the data could be organised into different themes, to abstractions of the framework, such as the final diagram in figure 14. In part some of these diagrams overlapped with the time-ordered displays discussed below that were more detailed in nature.

According to Miles and Huberman (1994) qualitative data preserves chronological flow, enabling us to see precisely which events led to which consequences and derive

fruitful explanations. Care has to be taken with the assumptions about causality, however, as we cannot give meaning to the event just because one thing proceeds another (Harré, 1981). Time based analysis was conducted to show the temporal relationship between different parts of the data and from these explanations were built. The type of time-ordered displays and matrixes used were based on those suggested by Miles and Huberman (1994). For instance, a substantial matrix was developed that showed the relationship between the initial problems identified with the processes, actions planned and taken (or not taken), and outcomes, both intended and unintended. Also, for different key process areas the relationship between the context and the actions in the software process improvement and software development were identified through time. It was from these data displays that the emergent framework discussed in Chapter 5 was developed.

### 3.5.4 Summary

A process of iterative analysis enabled the data to be considered from multiple perspectives and in different frameworks. Whilst the activity was done manually, the process followed the suggestions of authors in the qualitative methods field, so that the analysis used established techniques of data consolidation and summarisation, concept development through memoing, coding, and time-based analysis, and the development of a framework that acts as a means of analysis and explanation. An evaluation of the approaches followed is given below.

## 3.6 Critique of Research

### 3.6.1 Validity, Reliability and Generalisation of Findings

It has already been stated that it is not the purpose of this thesis to adopt a natural science perspective to experimentation design and application of the results. However,

it is important that the research design is rigorous so that the findings can be trusted as valid and reliable, and they can be used appropriately in other research and to inform practice. This section will discuss how the research has been designed to ensure validity, reliability, and generalisation of the findings. The following section will critically discuss the approach adopted by the researcher.

Validity is considered to be different things dependent upon the case study perspective, with positivist approaches tending to seek a greater level of objectivity in the data collection and analysis (Yin, 1994). However, here (internal) validity is essentially considered to be about ensuring that the research is credible, authentic and reflects a plausible explanation of the events. Silverman (2000) defines validity as truth; not some absolute truth, but that the account is a true representation of the case. One way to ensure that this is met is through what Silverman (2000) calls the refutability test, where early ideas and theories are dismissed and replaced with more considered views. It has already been shown (see §3.5.3) that a number of iterations occurred during the analysis, with the rejection of early frameworks: ensuring an original rather than a preconceived framework was created.

Validity requires that the researcher is unbiased in the selection of data from the potential or collected items, and that the items used are representative of the whole. One approach that has been adopted to improve the validity of the study is triangulation. Triangulation is the use of multiple sources and techniques to view the phenomena from different perspectives. Lee et al (1990) suggest that qualitative researchers can overcome bias in informants' views by drawing from multiple informants. Fielding and Fielding (1986) note though that the main sources of bias are

selecting data to support a preconceived idea, and to select more exotic data at the expense of the ordinary. Data has been drawn from across the organisational strata and sub-groups, avoiding the tendency to concentrate on a management or development group perspective. By accepting that each person has a perspective which incorporates their own bias then, by taking all the views into account within the analysis, contrasts and congruencies have been identified between and within social groups.

However, triangulation has its critics. Trauth and O'Connor (1990) consider triangulation a fallacy because it assumes that we can identify an absolute truth through multiple sources. Certainly to assume that simply using multiple data sets is in itself a guarantee of validity would be wrong. A compromise is given by Denzin and Lincoln (1998, p.4) who also agree that triangulation will not achieve validation because 'objective reality can never be captured', but they consider that triangulation is an appropriate alternative to validation. Fielding and Fielding (1986, p.24) support this view, stating that the advantage of triangulation is that it 'puts the researcher in a frame of mind to regard his or her own material critically, to test it, to identify its weaknesses, to identify where to test further doing something different'. It is only the researcher who can ensure a critical selection and review of the data.

Data should be used to develop an account that is meaningful, convincing and shows how the data links or contrasts to support the explanations or challenge them. In this study, the qualitative data shows why an event is occurring, which helps to establish internal validity in the findings (Eisenhardt, 1989). Concentrating only on the meanings of the participants can limit the understanding of the broader context

(Pettigrew, 1990b). These broader aspects can be identified though by drawing on other sources, such as trade literature, informants from outside the unit of analysis, and using information that provides a wider perspective such as from clients and suppliers. Other contextual features can be ascertained by observing actions first-hand, this also reduces misunderstandings that can arise from interviews.

Whilst quantification should not be undertaken for the sake of it, as Jorgensen (1989, p.34) warns 'quantification risks distortion', but quantitative data can give another perspective and that can be compared to the qualitative views. Here two forms of quantitative data have been used. A survey of development staff was taken to ascertain their views on the SPI activity and how the change was managed. However, as stated in §3.4.6, the survey has been used as qualitative data rather than analysed for statistical patterns. The other form of quantitative data is process and quality data collected by the organisation; these are used to inform the discussion through the way the changes were perceived by the informants.

To ensure that the account is plausible, it is important that theoretical abstractions are related to the details of the study. An IS researcher can only access the subtleties of the interpretation of the different social actors by using 'thick' description (Walsham, 1995), which is done through using a selection of field notes in the analysis. A process theory is developed, reflecting the interaction of human actions and institutional contexts over time. The findings reflect a sense of change and dynamism. The methods allowed data to be captured retrospectively and in real-time. The analysis of the data draws out and conceptualises the contexts and their interaction with the change process.

The value of an explanation is judged in terms of the extent to which it allows others to understand the phenomena and makes sense to those being studied. Two forms of feedback have been sought: on the case accuracy and interpretations, and the framework and analysis. The first of these is called respondent validation, where informants are asked if they recognise the validity of the account. Informants have reviewed the accuracy of the case study data. The process – product interaction was agreed as a strong theme, and its influence had continued beyond 1999 when the focus of the process widened with the move to multiple products. Whilst this evaluation should ensure that no significant misrepresentation has been made, Fielding and Fielding (1986) note that informants do not have a privileged position as commentators because their perspective is limited. In line with their view, the responses have been used to further inform the analysis rather than as a single judgement on the generalisation. The framework has also been presented to colleagues and at a conference to seek feedback on the interpretation. The relationship of an emergent process to agile software methods was formed at the conference through discussion with peers.

In addition to the validity of the study, we need to consider whether the research study has reliably captured the data. Miles and Huberman (1994) suggest that reliability comes from ensuring that the research design is congruent with the research questions. The purpose of much of this chapter has been to link the methods used to the aims of the study, so this aspect has already been covered in some detail (see §3.4).

More specifically, the underlying issue for reliability is whether the process of study is consistent. As a single researcher on a single case there is no concern about the issue of ensuring consistency between these aspects, as there would in a research team or multi-case design. However, even within the case, consistency could vary over time, so data collection features like the use of an interview protocol have been adopted to assist. Reliability, especially in quantitative studies, is considered the ability to replicate findings. However, this does not apply to ethnographic studies (Hammersley, 1990), but the earlier parts of the chapter have shown how methods have been applied in a diligent and orderly manner. The principles for interpretive research developed by Klein and Myers (1999) have been used to guide and assess the process and outcomes of the research. These principles are aligned to the critical interpretive approach followed here, i.e. iteration of the reasoning, contextualisation of the data, interaction with the subjects and a reflexive interpretation.

The data collected, the records kept and the way they have been organised has already been defined. Areas for improvement in the research approach and lessons for future projects include ensuring that the transcriptions are made the same day as the interview to reduce the chance of error. However, this potential flaw was countered by listening to the audio version of the interviews on multiple occasions (and any errors identified corrected in the transcriptions).

For some qualitative researchers there is a tendency to ignore the generalisation of their findings, focusing simply on the particular case (Silverman, 2000). However, the need to produce explanations from a single case that have resonance for practitioners and academics is evident in most IS case study work. By comparing the findings from

the case study with the literature a final descriptive hypothesis of the contextual factors affecting the introduction of SPI is developed. However, care has to be taken in generalisation as the epistemology of anti-positivism is 'firmly set against the utility of a search for laws or underlying regularities in the world of social affairs' (Burrell and Morgan, 1979, p.5). Walsham (1995) suggests that generalisation from case studies can be used to develop concepts, produce theory, draw specific implications and to provide rich insight. The developed theory indicates tendencies that explain past data rather than predictions of future events.

One of the main tasks in interpretive IS research is to clarify the context the phenomena occurs in. The discussion of the subject in this study has been set within the social and historical context. This allows the reader to see how the situation has evolved and they can interpret it for other contexts, enhancing the research's external validity. Contextual factors are directly linked to the phenomenon and not just provided for background material.

To generalise from one case is always open to criticism because it is just a single instance in a substantial population of possible cases. However, here it is not intended to produce a statistical inference but the creation of a plausible account that has general applicability to other situations. Placing these findings in a well-established theoretical framework, such as Structuration Theory, enables other researchers to relate this work to other cases and findings. The usefulness of the findings for other cases has been drawn out by developing an understanding of the implications for IS managers in Chapter 6. It might be argued that multiple cases could have provided stronger grounds for external validity but, unless there is a clear theoretical reason for

two or more cases, there is no more ground for generalisation from multiple cases than from one given the potential variance amongst all the possible cases.

### 3.6.2 Critique of Researcher's Role and Perspectives

Throughout the chapter it has been the aim to evaluate the approaches and views adopted. However, it is necessary to highlight and evaluate the biases. The role and perspectives of the researcher is critically discussed here to enable the reader to evaluate the work. To know how the researcher construes the world and the impact that has had upon the research activity and analysis is important in orientating this work with the reader's own understanding. The critique will look at three areas: the researcher's the prior understanding of the SPI field and its influence upon the case study and theory development; the research strategy and approaches adopted within the case study; and the influence of the researcher in the case study activity.

Prior to undertaking this research project, there were two major influences upon the researcher's view of SPI. Firstly, ten years' experience in the software development field as a programmer, analyst and project manager included the adoption of new methods, experience of the introduction of a corporate software quality system and the pressures involved in developing software within commercial constraints. This experience had encouraged a sympathetic view of structured methods and software quality approaches. Later, a reading of the software engineering literature reinforced the perceived need for a methodical and controlled approach in order to improve the software product quality. It was therefore from this perspective that the preliminary work in the project was focused upon the 'best' ways of reaching the goals and approaches in the literature. It was anticipated that InfoServ would seek to adopt this perceived 'best practice'. It was an important step therefore to move from that bias

towards seeking an understanding of how that activity was actually undertaken and how the changes occurred. As the review of the literature in Chapter 2 identified, and the discussion of the case study results will show, the improvement is not as clean as many of the engineering texts show and by understanding this change we can seek to support the improvement activity in other organisations from a more informed position. The shift in position, from seeing the SPI as an engineering activity to one of situated change, occurred through a reflexive analysis of the case in conjunction with a reading of the literature that challenged the traditional software engineering perspective.

Given the prior perspective it was a considerable risk that the researcher would unduly influence the improvement approach and outcomes. This risk was especially evident as there was prior contact between the researcher and informants in the software group, and a desire from the software group for a high level of support. However, this risk was identified early and the research purpose and approach communicated to the software team. Throughout the project the researcher remained closely involved with the activities as an observer, but it was agreed that the team could request information from the researcher to support their improvement activities. Information was provided about four process improvement areas, but only at the level that could have been easily accessed in any standard software engineering text book.

During the course of the project two sets of short training were provided on request: one on the moderation of software inspections and one on planning for the testing team, both lasted less than half a day. Additionally, members of the team attended seminars and training organised through the university. Two paid activities that had an

influence on the SPI activity were a series of satellite broadcasts by Eli Goldratt attended by the project manager, and a week long training course on object oriented design and programming organised for the development team (which a university colleague ran).

Throughout the data collection activity the review meetings with the software managers discussed the progress of the software product development and the software process improvement project. These discussions are described in §3.4.2 and were informal in nature, therefore opinions were sought and offered during these sessions. Also two written reports were provided. During the project a summary of the team questionnaire was circulated to the team. At the end of the participant-observation phase, as part of the access agreement a short report was produced to evaluate the improvement project. There is no doubt that these interjections influenced the specific actions of the team, but they were based on ideas that were initiated by the team and did not significantly influence the approach to the improvement project. Each of these interactions could have influenced the reflective actions of the participants, but were not designed to do so.

The researcher had a dual role with the organisation, as observer to research and as participant to share expertise where relevant. This relationship did not affect the rigour of the research. Any group being observed, particularly in the short-term, will act in different manner to usual, even if it is only to work to a higher standard that usual. Overt observation can therefore influence the behaviour of the observed group to some extent (Jorgensen, 1989). At least having the added participative role gives a legitimate other role to the research, which lessened the self-consciousness of the

observed group. As discussed in §3.4.3, the participant role was relatively limited and only undertaken on request.

By retaining a position physically within the project team the observation of daily activity was improved but this did lead to a more informal relationship with team members. However, as mentioned above, the position of outsider was retained throughout, principally as there was no involvement in the activity of software development or the SPI actions. The closer researchers become to their subject then the more they will affect what is being researched. Even where the purpose is not to change the situation being studied, the researcher will tend to project self onto it (Reason and Rowan, 1981). It is also true that the researcher is dependent upon those from whom they are collecting 'data'. There is less unilateral control and a greater pressure to work using other people's definition of the situation because social processes are ephemeral (Elden, 1981). This dependency makes research of this type 'more personal and interpersonal than methodological' (Reason and Rowan, 1981). It is therefore argued, as discussed earlier, that in case study work there is a need to move away from the idea that there is one truth, bringing us back to the 'soft' views of knowledge and reality.

## 3.7 Conclusion

This chapter has explained the research methodology in the light of the aim of the study and the wider goal of enriching the IS literature with relevant research. The study has been designed to follow an interpretive, longitudinal case study approach. Within the case Eisenhardt's stages have informed the process of data collection, analysis and enfolding of appropriate literature. The detailed analysis has utilised

methods from the qualitative methods literature, especially the analysis methods from the work of Miles and Huberman (1994) as their own philosophical preferences, and therefore methods, support dialectical hermeneutics.

The approach adopted throughout has been reflexive. The analysis has been interpreted to draw out inferences beyond the actors' own interpretations and from what has been left unsaid; underlying, unconscious motives and unforeseen consequences have been identified; the actions and interpretations are critiqued; the hegemony of software engineering is reconsidered to avoid 'narrowing down' the conclusions to fit existing theory, and the researcher's own biases are recognised and challenged.

The relevance of the study can be assessed against Benbasat and Zmud's (1999) recommendations. They consider that relevance can be achieved by selecting appropriate topics that are of interest to and develop outputs that can influence practitioners, and then write up findings in a way that encourages their implementation. The topic was identified directly from the case scenario but also through the recognition of the ongoing importance of quality to IS management. The area has been well covered but the problems have not been resolved. Here the case is used to identify a way forward in the understanding of process improvement as a form of organisational change.

To develop research that is useful to the IS community the research should be cumulative, theory-based and context-rich. The outputs should be intuitive, meaningful and useful for practitioners; they should be readable by crafting them in a

simple, accessible style. Here the output builds on existing research not in a theory-led fashion, but by 'enfolding' the literature in during the analysis. The theory developed is based upon Structuration Theory that is widely used in IS. The case study approach has allowed the capture of in-depth data, allowing the subsequent use of 'thick description' in the case analysis. The emergent framework and subsequent lessons for practitioners are designed to enable the communication of the findings to other settings. The following chapter highlights the history of the case.

# Chapter 4. Case Results: SPI at InfoServ

## 4.1 Introduction

The purpose of this chapter is to give the reader an insight into the context of the case study and the principal events that will be referred to in the following chapters. The chapter provides a brief overview of InfoServ, especially its GeoMarketing division. An introduction is given to the division's business and competitive environment. An overview of the organisational history highlights the development of the software products and changes to the software processes. Accordingly, the chapter reveals the changes that occur to the software processes within this packaged software organisation. The approach adopted by InfoServ is evaluated using prior SPI theory. To achieve this aim the final section examines the software process improvement project in more depth.

## 4.2 The Company

InfoServ, a wholly-owned subsidiary of HVT, are a leading global information services company operating in over 60 countries. The company was formed in 1996 following the merger of NNC in the United Kingdom and USI in the United States of America. NNC was created over twenty years ago from its parent company, HVT, as a credit reference agency. Reusing the credit control skills and knowledge from managing the credit risk of HVT's clients they began to offer these services to other companies.

USI began in the US in the late 1960s. It became a leading US company in credit and marketing information, with high quality information products and services, and data resources. Like NNC, USI had built up massive databases on consumers, business and

property as a basis for providing marketing services. The £1 billion acquisition in 1996 by HVT enabled NNC to have a prominent position in the US market, and the opportunity to deliver a stronger global presence as the renamed InfoServ. For simplicity InfoServ will be used as the company name throughout even when the discussion is drawn from time prior to the formation of that company as well as after.

In 1980 InfoServ's UK sales were £3 million, rising to over £60 million by 1990. By 1995 the company had customers in 12 countries and had a turnover of £120 million. In 1998, InfoServ had 7000 employees worldwide. The annual turnover had risen to £600 million from customers in over 40 countries. By 2003 the company had over 13,000 employees worldwide with an annual turnover of £1.2 billion. The rate of growth indicates the strength of the company and its ability to adapt to the demands of its market. The company has diversified to develop a broad range of business information services. The key business areas are now:

- Information Services: provision of information about consumers, property, vehicles and businesses.

- Decision Support and Account Processing: customer management through risk analysis, credit application processing and authorisation, customer account management and communication, credit and arrears control.

- Business Support Services: the provision of services to allow organisations to outsource their telemarketing, call centres, data capture or payment processing.

- Target Marketing: the identification of prospective customers, analysing existing customers and markets, planning the position of outlets.

Within the UK, InfoServ uses a divisional structure organised on its key business areas listed above. Each division operates as an independent business unit, with its own internal board. The divisional managing directors sit on the UK board. Each division acts autonomously, within agreed targets from the UK board, reporting its own profit and loss.

InfoServ is now one of Europe's largest credit referencing agencies. The credit referencing and related areas, such as fraud detection and bulk transaction processing for banks and other credit card companies, still form a substantial part of the business. Recent acquisition of a leading US direct marketing and consumer database provider enabled InfoServ to become the UK market leader in target marketing. Other acquisitions and organic growth have enabled the company to reach the size it is today. InfoServ remain a significant part of the HVT group, by 2003 they contributed £256m pre-tax profit to the £700m HVT total.

## 4.3 GeoMarketing Division

This study relates to the GeoMarketing division based in the UK headquarters. GeoMarketing is a form of target marketing. By segmenting customers and potential customers into different groups, the objective is to enable companies to be more precise and effective in their communication with customers or in their planning for sales and marketing. The development of the SPECTRUM geodemographic

classification system was a key step in achieving InfoServ's prominent position in this business. SPECTRUM, the principal product of the division, classifies each postcode area into a pre-defined type that describes the location and typical people living in that postcode. This classification system has been developed using data about individuals drawn from census and electoral roll data, combined with data from other divisions and data purchased from other sources.

SPECTRUM was developed by the Division's Managing Director, Richards, who is arguably one of Europe's leading authorities on geodemographic segmentation. He was perceived as the 'ideas man behind the organisation'[1]: a 'powerhouse of good thinking, creative thinking, ... high intellect, [and] innovative'.[2] In the early 1980s he developed the predecessor and main competitor to SPECTRUM, Galaxy, for direct mailing using the electoral roll. InfoServ at that time was a customer of Galaxy Systems and resold Galaxy to its clients. In the mid-80s, the management at InfoServ decided that there was more to be gained from being a data owner than being a reseller. Richards, building on his experience with the Galaxy classification system, moved to InfoServ and developed SPECTRUM using the data available and thus removing the dependency on Galaxy Systems. The SPECTRUM classification system was initially used on mainframe systems to provide a bureau service for clients. By the late 1980s the PC technology had developed to support the type of analytical processing required.

---

[1] Log
[2] Interview 3

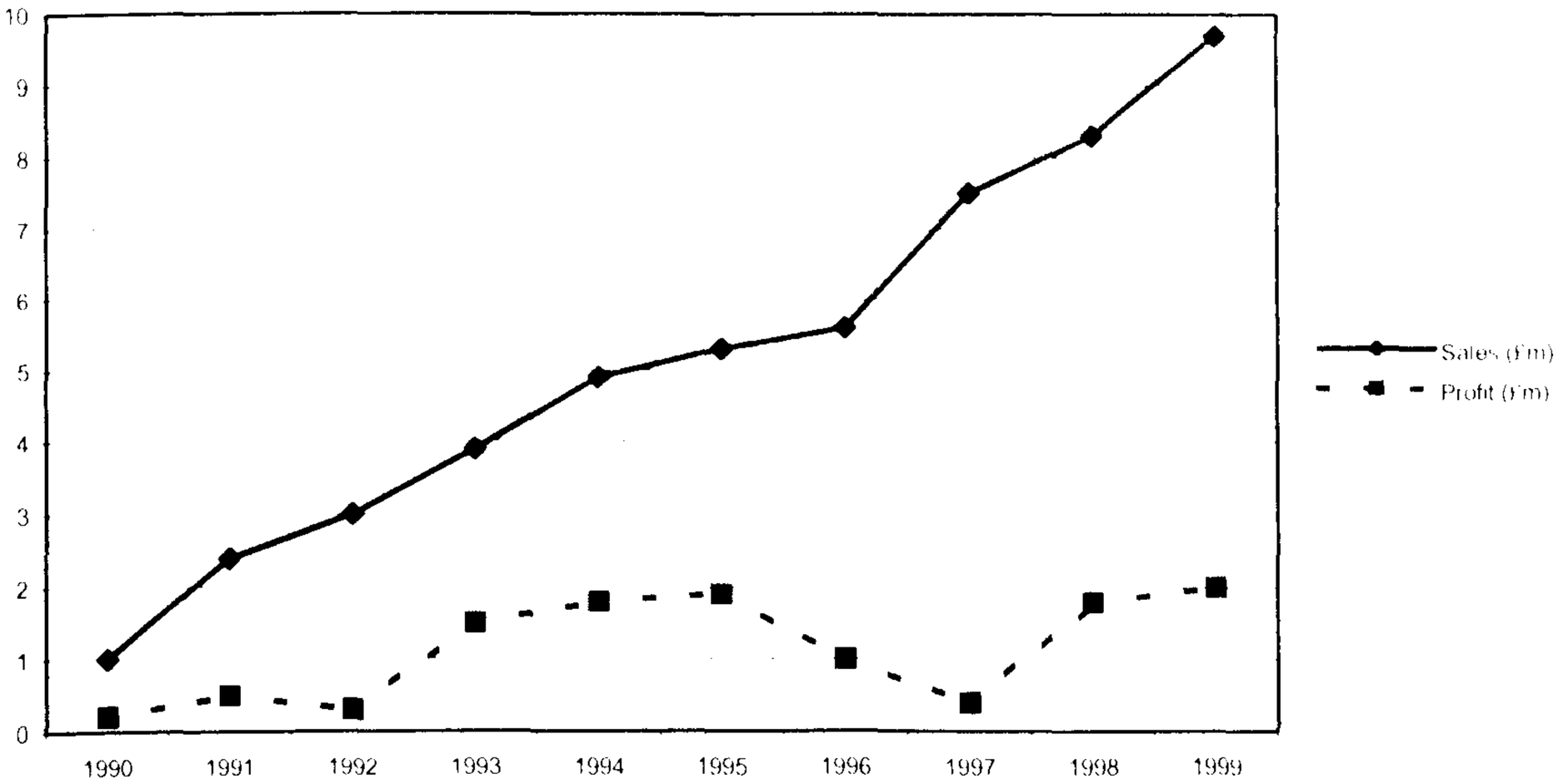



**Figure 7 GeoMarketing Turnover and Profit**

The GeoMarketing division is one of the smallest in the company. It began with a number of the current divisional directors, drawn from geographic, economic and marketing backgrounds. The directors had only limited prior knowledge of IS in practice: Richards had used computers to develop the statistical models in previous organisations and the product director, Herriot, had studied information management as a post-graduate. In 1987 there were just five people and the division turned over approximately £300,000. By 1999, it had over 170 employees and turned over nearly £10 million (see Figure 7). Since the conception of the SPECTRUM product, the software and its associated development team have grown. Initially the development was integrated with other activities, but through the growth of the division a set of specialist areas have evolved. The software team hierarchy during the period of the SPI project is shown in Figure 8. The teams highlighted in solid background are those that were known as the Market Analysis Package team and are the main focus of this study. The bespoke software group and help desk were influenced by and influenced the process improvement and so were included in the study but were not the main

focus of the study. They are therefore only discussed in relation to their influence on the process improvement activity in the packaged products team and vice versa.

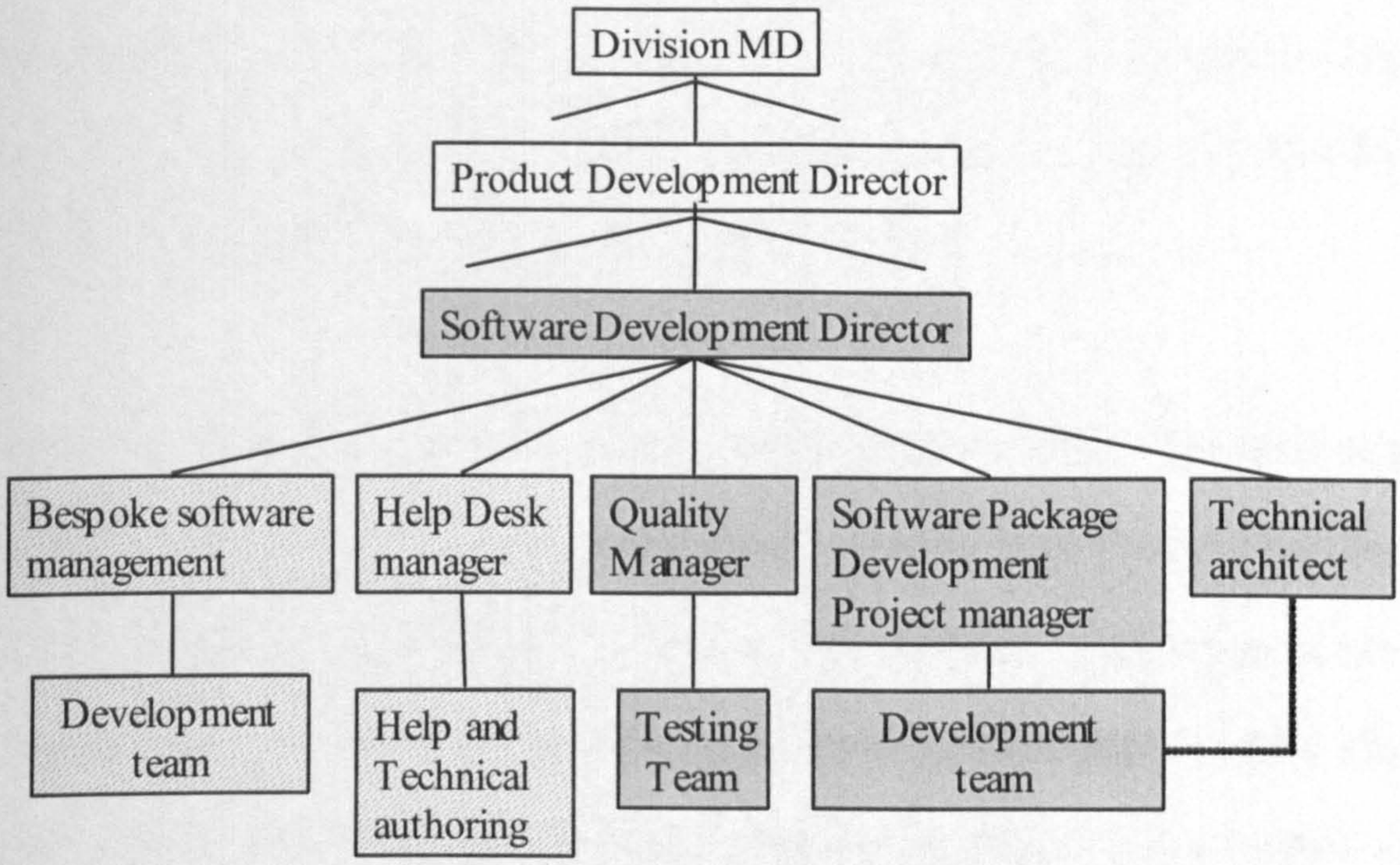



**Figure 8 Software product group - organisational chart 1997-1999**

## 4.4 The Competitive Environment

The competitors for the software products fall into three broad categories: companies that provide generic geographic information systems (GIS), those that have combined software and data products to provide a more integrated product, through to those who provide specific geomarketing solutions.

The GIS providers have little impact on the geomarketing market, with only 10-15 companies using these products for this purpose. These suppliers tend to be technology based rather than solutions based, focusing on the provision of the mapping technology. Integrated software and data suppliers are a more serious threat

to InfoServ but they do not provide a full geomarketing solution. Many of these organisations started out as software suppliers but increasingly the suppliers have realised that the value is in the data rather than the software. Most of these competitors take their data from third parties and integrate it with their software. Indeed, not all the companies are clear competitors as some of these companies use InfoServ data and resell some of the InfoServ products.

InfoServ see themselves as a specialist geomarketing solutions provider. Galaxy Systems form the most significant threat as they have the most presence, building on their development of the Galaxy product. The specialist solutions providers all combine data and software products for the market analysis required by clients. InfoServ claim that SPECTRUM geodemographic segmentation system is the world's most widely used for classifying consumers. The software developed to facilitate the use of this segmentation system is now called Market Analysis Package (MAP). The Market Analysis Package software product directly contributes approximately half of the GeoMarketing Division's turnover. Consultancy and other activities also require the product to function effectively. Over the ten years 1990-99 the revenue of the division rose tenfold to nearly £10 million.

The customers for the geomarketing products came from a variety of sectors, including retail, leisure (such as food and drink), motor sales, media, and finance. The products were used for the analysis of existing clients, the identification of new clients, and the targeting of products and outlets. For example, companies identify what type of clients purchased different types of their products, develop marketing and sales strategies based on an analysis of current and future clients and identify

appropriate locations for a new store. Due to the costs involved many of the traditional clients for the software were blue chip organisations with large marketing departments. Smaller companies could make specific job requests to the service bureau. One of the developments in InfoServ's products was to provide ways of supporting smaller companies through cut-down or market specific versions of the products. The remainder of the chapter will reveal how the product and the processes changed up to 1999.

## 4.5 Product Development History

During the late 1980s a mainframe based bureau service was provided to service clients' requests. At this time, InfoServ had a centralised approach to developing software on mainframe systems. By moving away from this platform to the PC, the division was allowed to develop its software with little influence from the centre. The skills that they needed for developing PC based systems were not available in the company, therefore they were allowed to form their own development group. This autonomy was highly valued by the directors, who did not want to be constrained by the centralised systems department's approach and attitude to developing software. It was felt that the software produced by the central systems department 'didn't meet what the commercial people were looking for... [there was a] distinct lack of understanding of what the needs were of what was a small specialist business in this area'.[3]

By 1990 the organisation had made a major step towards becoming the leading player in the geodemographic marketing field. By creating a PC product 18 months before their main competitor, InfoServ were able to establish a market leading position as the

main supplier of PC market analysis tools. So whilst the competitors were only able to offer a bureau centre, SPECTRUM Systems also allowed the organisation to sell a PC solution to clients. At that time Galaxy was widely known in the marketing field, but InfoServ's move to selling the whole solution to the client opened up a new market. This period was an important period for the growth of the business, as one of the GeoMarketing directors states:

> We got a good lead there...we were well ahead and we have always managed to maintain that and a really important point about that is software and data are the mainstay of this business, it's where the profits are. So we managed to build up a base of profitable managed leased contracts by renewal each year and that is the foundation on which the business is built each year so very, very important is this (sic.) integrated software and data packages.[4]

SPECTRUM Systems was a compilation of different pieces of software. The territory management part was produced by an external vendor. InfoServ developed their own facility for mapping SPECTRUM postcodes. To this set of software other products and facilities were added: a sector ranking product, a segmentation system and a customer analysis system. These elements of the system were partially integrated to allow data to be passed between them and to allow results to be produced on maps. However, customers, and consequently sales staff, complained of the need for a truly interoperable system. So during the period to 1993, InfoServ gradually developed the system to improve the functions available and enhance the integration between the different sub-systems. Despite their market strength, the changing technological

---

[3] Interview 21
[4] Interview 21

environment was to prove to be something of a stumbling block in redeveloping their product.

The development of SPECTRUM Systems version 4 (referred to as 'SPECTRUM4') is a critical point in the history of the division's software development. The project began in 1993. The aim of this development was to provide additional features that would provide the company with an interim product that would allow them the time to develop a Windows-based product. The SPECTRUM4 development added new analysis functions and resolved some of the integration problems. Initially, the development of SPECTRUM4 was intended to be something that was quite straightforward: just a set of minor updates. However, little control was placed on the functionality to be included in the product, so what started out as a relatively simple increment turned out to be complex and oversized:

> there were many more little ad-hoc innovations which were treated in a rather unsystematic way. And consequently you got design inconsistencies for the user. It was very difficult because you'd do the same thing in totally different ways in different parts of the program. There was masses of redundancy (sic.). It was very difficult to test the thing because it had so many different bits to test, and the whole thing was rather informal, organic and a bit shambolic.[5]

> Anyone could ask for any change they wanted and, because there was a desire to build a product that would last, these requests were incorporated. Requests were made direct to a particular developer who would change their part of the system as they saw fit. This resulted in a functionally rich, but distended product. So the ad hoc, informal

[5] Interview 22

approach to development that had been considered a significant strength when the division and its products were small, had become a constraining factor.

In retrospect the scale of the change put into SPECTRUM4 was seen as a mistake. The upgrade had taken on too much, with the redevelopment of whole sections of the product. One of the development team members stated:

> Basically they tried to do too many things all at once. The area analysis software was a complete rewrite out of the old Fortran code into Pascal. That was as complete rewrite job. The mapping side got a complete overhaul of its user interface and tactics, the segmentation side got a complete overhaul of its data dictionary structure. As well as half a dozen other things all going in: cosmetic changes to make it all look the same. So there was tons, tons that went into it, and so not surprisingly it took a long time and there were lots of problems to extract out of it.[6]

The overhead costs in maintenance, support and training were subsequently high. The product was delivered in the end but, as one director remembered, the clients expressed concern because 'whilst technically very capable it was a monster to use because it had grown and grown'.[7] Accordingly, many of the clients had to rely upon the help desk to assist them in their use of the software, only the more advanced analysts found that the powerful functionality was an improvement over version 3 of SPECTRUM Systems.

---

[6] Interview 25
[7] Interview 21

The delayed development meant that once it was ready it was already too late to be successfully marketed, as the sales director explained: 'the world was moving on very rapidly into a Windows environment - Windows 95 environment - and here we were still stuck with a software product from the DOS era'.[8] The decision to undertake this development rather than switch immediately to Windows was made by the product management team. They felt it was important to develop a product that would cater for most of the requirements to give the development group time to create a new Windows-based product. The systems management 'did not have a good understanding of the way in which PC software processes were moving or even the PC software marketplace was moving'.[9] With the advent of Windows 3.1 and subsequently Windows 95, integrated software became the norm. Sales for SPECTRUM4 soon dried up, as no one wanted to purchase a DOS product.

During this period the divisional Board of Directors and software management had to decide on a strategy for retaining and continuing to enhance their DOS product or switch to a Windows product. They all recognised that in due course a Windows product would be necessary but selecting the right time was difficult. It was decided to build a further version of the DOS product to allow time for the creation of a Windows product. In hindsight, this decision was recognised as a mistake by the divisional managers. The speed at which customers wanted to drop DOS had not been anticipated; the changes in the technological environment had overtaken them. The upgrade had come too late to pacify customers. This was a legacy that hung over the development of the Market Analysis Package, and prompted changes in the software development process. After reflecting on the mistakes and problems in the

---

[8] Interview 21

SPECTRUM4 development, the GeoMarketing Board decided to change the software development competence through recruitment. This recruitment policy was designed to break away from the past. New recruits were 'brought in from outside to change [the] culture, to change the processes, give the whole thing a clean sweep'.[10] In particular, the need for specialist skills in managing the software development had become clear to the Board. A new project manager, Tyler, was recruited with PC development expertise. He was brought in with a deliberate policy of trying to change the organisation away from being 'enthusiastic amateurs'[11] towards managing software as a professional discipline; he was given a specific remit to implement formal processes. Tyler's background in a manufacturing organisation had exposed him to successful process engineering and improvement.

A technical architect, Jones, was transferred from another division of the company. He was recruited for his knowledge and skills in PC software development. Later in 1995 the project manager recruited a quality assurance manager to head up the testing of the product. The rest of the team members were chosen from the existing SPECTRUM Systems development group or were graduates recruited direct to the team. The project team was organised into three development teams and a testing team. One of the development teams, known as the 'core', was run by the technical architect. The other two development team leaders had been drawn from the SPECTRUM4 project team.

Following the difficult and overdue development of SPECTRUM4, the development of the replacement Windows-based Market Analysis Package (MAP) product got off

---

[9] Interview 21

to a late start. The move to develop Market Analysis Package officially began in 1994, but did not start in earnest until spring of 1995, when personnel were free to move to the new project from the SPECTRUM4 team. By this time, a Windows product was already an urgent business priority. Existing clients were looking to upgrade and new clients were not willing to buy the DOS product.

Despite these pressures, this project was seen as a major opportunity to rethink the product. The first task was to initiate a wide scale review of the business and user requirements to ensure that there was a clear understanding of the future needs. A number of joint requirements planning workshops were organised. Over a hundred of the existing clients were involved in the design process from specifying the functional content to reviewing the prototype. The requirements capture activity was beneficial but not without its problems. It was difficult to obtain the commitment of sales and management staff for involvement in these meetings because of the continued focus on the recently released SPECTRUM4 product. The sales team, who had been heavily involved in the SPECTRUM4 product, was now trying to sell this product. The desire to open up the discussion on requirements early attempted to draw on a previous affinity and comradeship between systems and sales staff. However, there was a growing animosity between the groups: the sales team now mistrusted the delivery dates and were resentful about the affect it had on their jobs. Eventually prototype designs were produced by the software team, they were discussed with clients and senior management.

---

[10] Interview 21
[11] Interview 6

The functional design of the product therefore reused much of the team's knowledge of SPECTRUM Systems. The user interface, though, was redeveloped entirely. The interface was designed using the wizard style that Microsoft had made popular. The product was intended to offer the analytical facilities required by all users whilst being accessible to people with different IT skill levels. To overcome the complexity of many of the analytical tasks, wizards were used to lead inexperienced users through difficult functions. A data view interface allowed users to combine information together from different sources, such as the internal data with the SPECTRUM codes, other InfoServ data about retail locations or externally sourced data such as media catchments.

The technical architect drew on his external contacts and prior knowledge to develop a nine-layered architecture for the system. The lower layers contained the 'core' of the product. It featured the database management system for user data and an internal repository for holding system data. The top layers were the user interface. The middle layers were designed to do the processing to deliver the analysis from the data to the user interface. Layered architectures are a common feature of object oriented (OO) systems now, but in 1995 they were in their infancy. This is one of a number of examples where the technical architect acted as gatekeeper to bring about the implementation of a design or process feature from external software engineering practice.

The application provided mapping, reporting and area profiling, built on a relational database that integrated spatial and non-spatial data. Market Analysis Package was initially developed for Windows 95, and later upgraded to a full 32-bit version for

Windows NT. The product was written mainly in-house. The team did look at external sources for the mapping and database facilities. For mapping, they decided to develop facilities of their own because the available technology was inadequate for their requirements. An external database management system was used initially, but was replaced shortly after Market Analysis Package was implemented.

The move by InfoServ to a Windows development environment brought its own problems. Knowledge and skills in this area were in short supply across the industry. With the exception of the technical architect, the team's knowledge and experience of OO was minimal. The software development team was faced with significant change in the development tools and environment they were using. These changes created uncertainty and additional complexity in the development. As one of the team leaders stated:

> Anything new has a major impact, no matter how much of a saver it is meant to be. We'd got new design methodologies, we'd got new tools, we'd got [a] new language, [and a] new operating system. We'd got so many new things that [...] it is surprising we managed what we did ... we wasted a lot of time doing work on certain other things as well like the work on [the external mapping tool] for example.[12]

One example of how this lack of knowledge affected the adoption of new methods can be seen in the introduction of a CASE (computer aided software engineering) tool. This tool was used to support them in the application of OO design methods, but the tools for OO development were still immature during this era. The lack of

---

[12] Interview 25

knowledge in the team of both the methods and the tool, combined with the ineffectiveness of the tool, meant that a lot of time was spent trying to make use of it without any perceived benefit. This experience had a damaging effect on the team's confidence of CASE tools.

In response to the problems that some developers were having with aspects of OO, the technical architect undertook to explain these topics in team workshops. However, the response to those workshops was not as he had anticipated, with dwindling attendance and negative comments about their usefulness. The other team members found Jones condescending, impatient and unable to explain the ideas at their level. He felt they made little effort to learn and to try ideas for themselves. It was not long before Jones discontinued the sessions, frustrated at the team's lack of motivation to learn and self-reproachful over his inability to communicate the knowledge. Eventually the tool and some aspects of OO design were temporarily abandoned, with developers preferring to resort to familiar, conventional approaches.

The impact of the division's strategy to develop SPECTRUM4, the last DOS version, was evident to those involved in the sales activity, with a flattening out of the revenue and profit growth during this period (see Figure 7). The Board's response to the sales pressure was to try to find a commercially acceptable date for the launch. So at their annual customer conference in autumn 1995, they announced that the product would be ready for August 1996. The team immediately felt that this target was unrealistic. Despite the evidence and feelings of the team an internal memo, in April 1996, described the development progress as progressing 'extremely well' and that confidence was high to achieve the August target. Some of the sales staff challenged

the management on this view, claiming such information and progress reports as 'propaganda'.[13]

The sales staff felt they were proved right with the first internal release not occurring until December 1996, with client releases starting to occur in spring 1997. Following the development of the MAP product significant problems were encountered with the quality of the software. One director summing up the consequences of the pressure stated that:

> in reality the very first installation we did, which with all due respect was held together with black tape and fuse wire, was in December. And we began taking the product out in March time. That was [version]1.1 and there were almost releases coming out every other day. So we had clients who said "hey you told me August it is now March, and why doesn't it work very well?" So you had this pent up demand. [14]

So whilst the first version of the Market Analysis Package was available for sale in February 1997, most of that year was spent consolidating the original development. Despite the commercial pressure to re-establish the product as a leader in the market, account managers and sales staff were still worried about selling the product as a replacement for the existing SPECTRUM4. The MAP development team and the sales staff remained dissatisfied with the quality of the initial product. The functionality of the product also required a lot of work to have the same capability as SPECTRUM4. During the first five months a release was made every two weeks to

---

[13] Interview 31 / Internal memo

rectify defects in the software, after which releases became monthly. By the middle of 1997 a number of performance, stability and technical releases had been made.

Many of the developers and the account managers, who look after the existing clients, wanted the balance of resources to continue to be skewed towards reducing the problems. The management was more concerned about the lack of functionality as it did not match the capability of the existing product and developments in competitors' products. A strategic plan was formed to enhance the existing Market Analysis Package product to cover the prime areas of SPECTRUM4 and to incorporate a data mining capability. Once the software management considered the product to have stabilised to an acceptable level in mid-1997, the team were moved onto the development of the product's functionality. Significant improvements to the reporting and mapping interfaces, and the introduction of additional market profiling features to the existing sales management functions were released in spring and summer 1998 as versions 1.6 and 1.7 respectively. Throughout the next two years the tension surrounding the trade-off between developing of new functionality and improving the existing product remained prevalent.

In addition, by the late 1990s, many blue chip companies were using geodemographic products as an essential part of their strategic marketing decision making. The complexity of products like MAP was such that they were only likely to be purchased by larger organisations, who have the capability and need to use a sophisticated product. To counter this situation, it was the software management's intention from soon after the initial release of version 1 of Market Analysis Package to produce a

---

[14] Interview 26

portfolio of products. A portfolio of products can assist packaged software developers in delivering products earlier than their competitors (Meyers, 1997), and enables a company to gain market share in different segments of the market. InfoServ envisaged the portfolio as a set of small tools intended to complement the Market Analysis Package. It was envisaged that this strategy would open up the sale of the data and analysis concepts to different sections of the market. The target market can be viewed on two dimensions: the type of market the product addresses and the sophistication of the product (see Figure 9).

Prior to the MAP project it was expected that the software, whilst integrated, would be modular and therefore usable as separate items, but that had not been possible. One unanticipated consequence of this lack of modularity was the need for additional software development to produce the portfolio of bespoke and smaller packaged application, each with little or no reuse. The expansion of the portfolio beginning with the enhancement of the Market Analysis Package product, including its data mining facility, is discussed below. The development of the other products in the portfolio is briefly explained to give the context for the changes in the software processes.

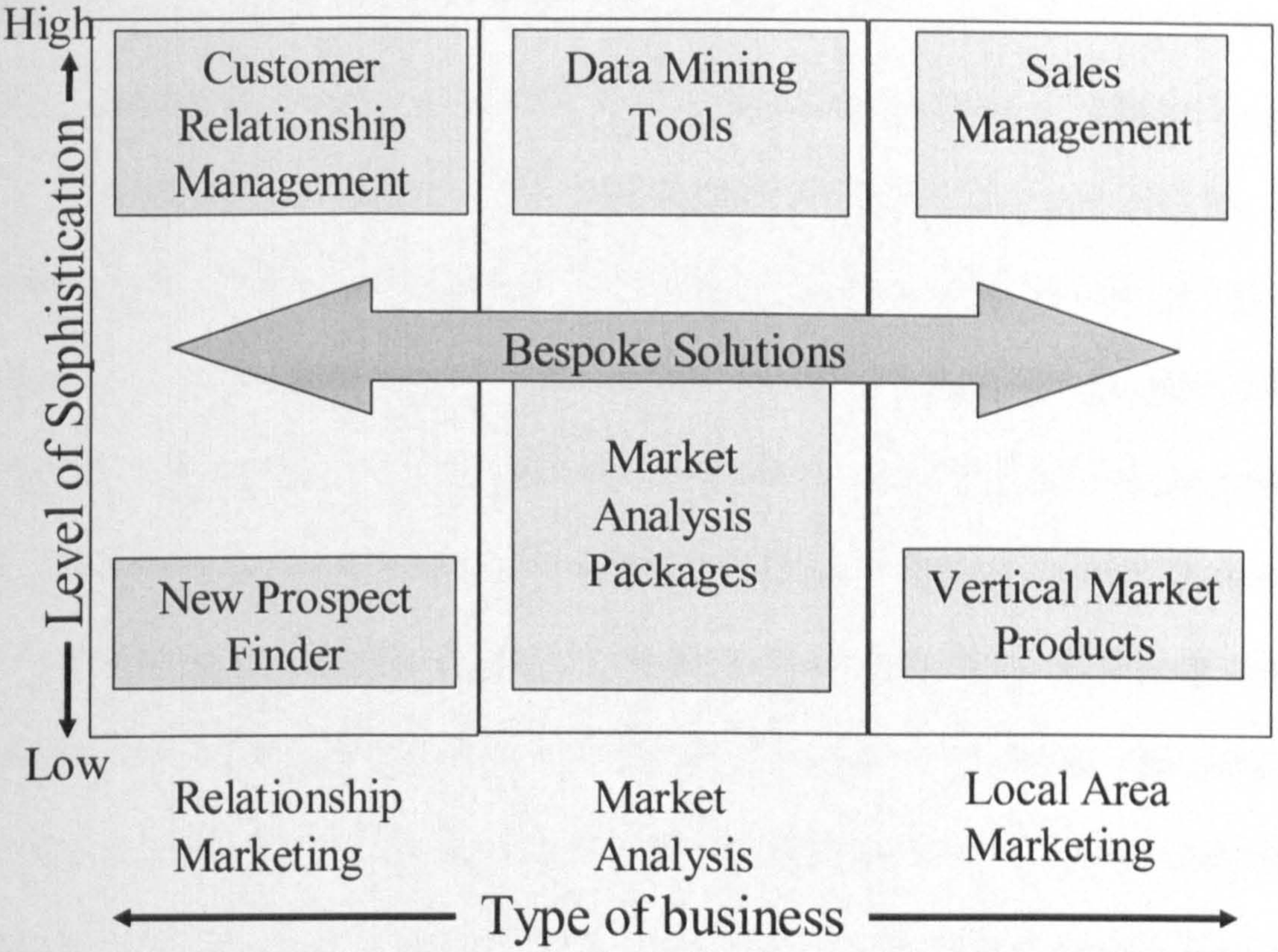



**Figure 9 Software product portfolio**

Market Analysis Package Version 2 was a major release that had taken over a year to produce. The push for new functionality from MAP version 1 culminated in this new version ensuring, at last, that it was equivalent to SPECTRUM4. Many of the functions introduced had been requested by key clients and were prioritised over other features.

In order to enhance the package's capability to analyse customer databases and associated customer activity data, Market Analysis Package Version 2 was extended by the addition of an optional data mining facility. The data mining software used an embedded modelling engine sourced from the US. This engine supported different statistical modelling techniques to allow the user to automatically find the best predictive model for a given application. In a customer briefing the company stated

that the Gartner group positioned the supplier in the top three of producers of data mining tools. Despite this reputation the development took much longer than anticipated due to problems incorporating the engine into the Market Analysis Package tool. Many of the problems were considered to be due to the product director's decision to adopt the product without involving the software team; consequently there was little evaluation of the software and the vendor prior to the purchase. The software director thought that this example highlighted the domination of the directors in these decisions reflecting the 'problem in justifying investment internally and [the] tendency to go outside because it is cheap…the data mining engine was a bit of a dictum "you shall go forth and use this"; the deal was done before I had even seen the product'.[15] Despite these problems, once it was complete the product was widely praised within the organisation. Following the release of version 2, the number of Market Analysis Package installations continued to increase (see Figure 10) but the rate of increase slowed below the planned growth of 10–20% per annum, as the market was beginning to saturate.

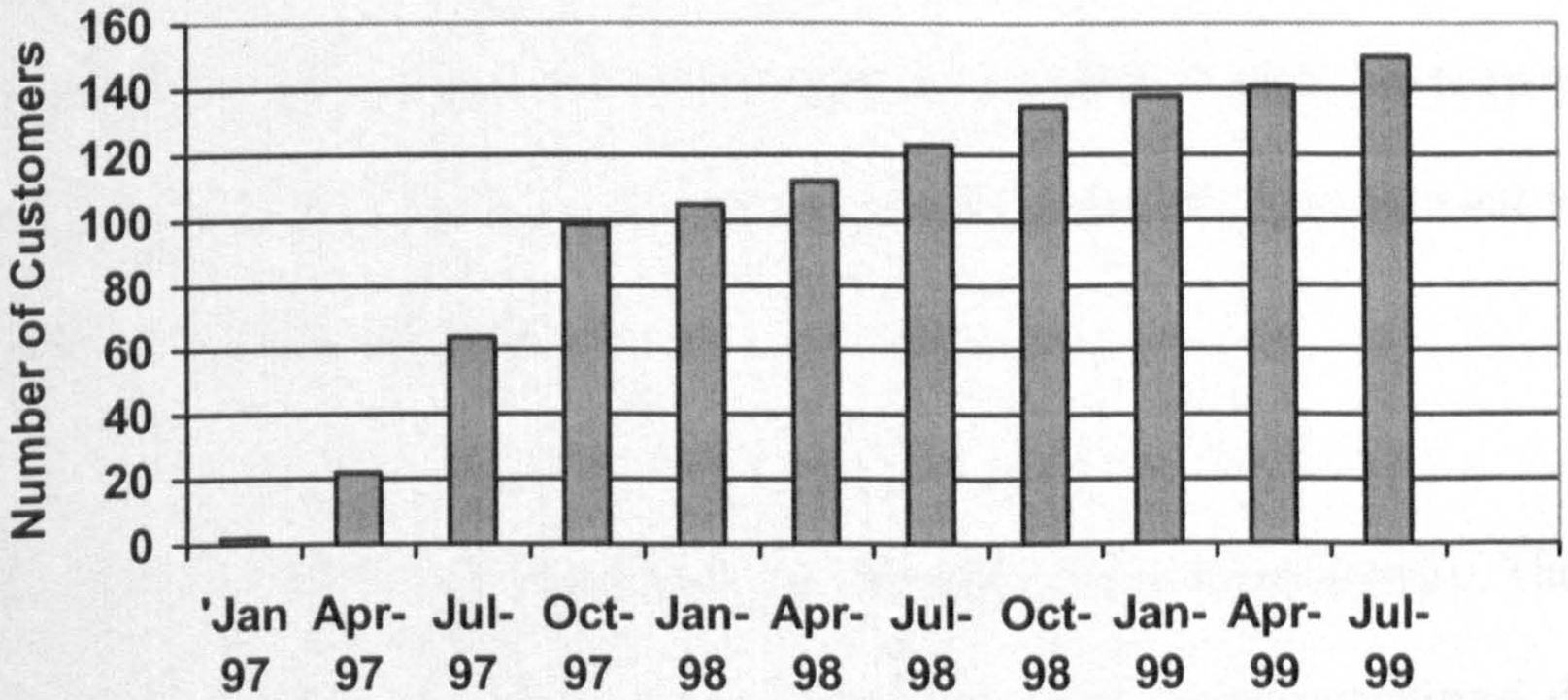



**Figure 10 Number of Market Analysis Package Customers by time**

[15] Review 6

At approximately the same time as Market Analysis Package version 2, two products were developed in response to events external to the development group: the links with the US meant that the organisation had to fulfil internal promises to deliver a market profiling product for the US market and, in response to a competitor's product, a tool was created to classify customers on an individual/household level rather than at postcode level. These developments were outsourced. Also, a new prospect finder system was developed by a external vendor using their specialised database tool. Its focus was on the creation of names and address lists from the names on the electoral roll. Using the same company and technology, a vertical market product was produced that focused on the analysis and mapping functions so that the motor industry could examine possible permutations of vehicle make and model characteristic penetration in any geographic area. Another specialised analysis product was developed for the UK's local newspaper market that allowed newspaper sales teams to perform area analysis against their sales territories and the sales areas of their local clients. During 1999, a new venture was being made in customer relationship management using the customer classification. In addition, a bespoke development team was formed to develop small stand alone applications to meet the requirements of specific customers. The team used desk top and multimedia development tools to provide this new sales channel.

Each product was developed individually to minimise risk and complexity. The MAP development team were given little say in this approach. Some considered that the investment of the same time and effort in creating a product that could be reused would have been more beneficial, stating that the business had been 'poor at leveraging the underlying technical elements in the Market Analysis Package

product'.[16] The unanticipated consequences of these decisions were the duplication of software, a lack of sharing of knowledge about the product and development processes between the MAP developers and the other teams, where ad hoc and short-term development approaches remained.

Following the release of the initial Market Analysis Package product, the subsequent expansion of the product portfolio and move into bespoke software, Tyler was promoted to software director. Then one of the team leaders, Bridges, was promoted to MAP project manager. Overall the team questionnaires and interviews were consistently favourable about the team leaders: their political and technical strengths complementing their concern for staff development and teamwork. As one developer put it Tyler was considered 'a good manager...[who] knows how to get people motivated...[and has] enough willingness to use his clout to push things [with the directors] that affect the development team'.[17] The new project manager drew on his knowledge of the system and ability to get on with people to lead the team. His non-confrontational nature was perceived as either a benefit and drawback by different members of the team.

The software management team acted as a relatively cohesive unit in its approach to developing the products and improving the processes. The technical architect, Jones, held differing views to the others on aspects of software engineering process. Within the management group these differences were cordially debated during the team meetings and a consensus was reached on most points. The differences helped to hone

---

[16] Interview 19
[17] Interview 20

the views and shape the understanding of the other managers. These differences were more striking within the team itself, generating ongoing conflict.

The package development team could be seen as having three communities-of-practice; the testing team could be seen as a separate sub-culture again. These groupings represent a relatively crude generalisation as individuals did not conform to any single pattern, but this representation helps to analyse how the norms and habits of those groups influenced the actions. The groups, also, were not fixed over time as individuals changed perspectives, but they characterise the groups during this period. The communities-of-practice relate partly to the physical and logical groups of the team, but also to personal relationships. There were two main logical sub-units within the MAP team during this period: those who worked on the advancement of the standard MAP system and those who were allocated to integrating a data mining engine. Each of these groups sat in different rows of desks that were divided by head height barriers (see Figure 11). This relatively limited physical divide was highly symbolic as one team had 'definitely has nothing to do with the other side'.[18] Two communities-of-practice were within these logical sub-units: the nucleus of the data mining team (i.e. those who started and continued the development throughout) and a group of MAP developers centred around a set of long-term employees from the SPECTRUM package days. Each of these groups developed their own culture and refined versions of the processes; they were different in the way they worked and despite a close proximity and supposedly common defined process, one developer noted that 'very rarely are ideas brought across'. The third group cut across these

[18] Interview 12

teams and can be categorised as a set of *disciples* of the technical architect, i.e. those who saw him as someone to look up to and follow in software engineering technique.

The social relations between the members of the team tended to be supportive, but Jones and the related community-of-practice continued to be isolated both through their critical attitude of others and their action to introduce external software practices. It was not that the other groups were intellectually weaker. The package development team were generally a highly talented group. Most were educated to university level with over a third holding postgraduate qualifications, and two-thirds of them considered that their qualification was relevant to their job. The view of the team, and of others in the organisation, was that they were technically strong. This capability had been devised by the software director, who deliberately organised the whole software team such that a talented group of developers were supported with others to do subsidiary work, thus allowing the team to be as productive as possible.

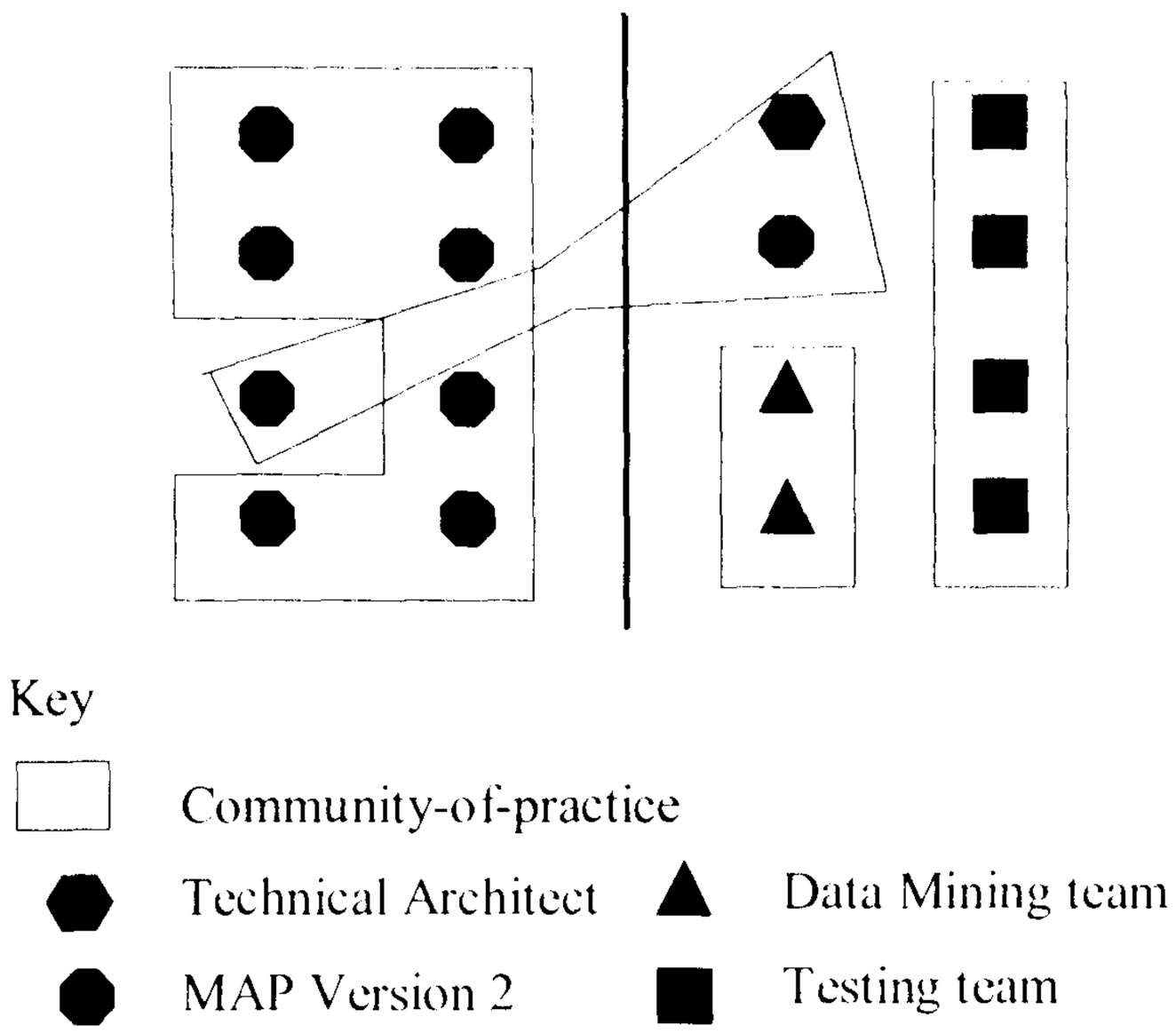



**Figure 11 Physical Layout and Communities-of -Practice**

The context in terms of the product strategy, the events external to the organisation, the capability of, and relations within, the team all informed and shaped the action of the team in improving the software processes and developing the products. The following section shows how the process improvement activity was influenced by these circumstances and events.

## 4.6 Software Process History

The development processes used throughout the period discussed above changed from the initial ad hoc, individual approach, through a time of prescribed methods, to a situation where the methods provided a referential framework that the practitioners applied in a contingent manner. This section will outline the key changes and highlight how some of these changes occurred.

For the SPECTRUM4 project the processes within the team remained undefined and informal. As shown above, each developer held a parochial view of the system and of the methods used to develop the software. The approach selected for each development was based on the developer's experience and preference. During the early versions of the product, the consequence of this inconsistent approach to the development had been problems with the product quality, but the market dominance enabled the organisation to cope with the problems. Despite the project manager's prior experience in the central systems department no formal methodology was adopted at this stage.

Once the MAP development began, the new project manager and technical architect agreed to produce a formalised development process. This action coincided with the organisational wide initiative to become ISO 9001 accredited. The support of the UK Government through the TickIT scheme for ISO 9001-3, the software specific version of the standard, was an encouragement to many software organisations to follow this path. The division fell in with this initiative as a way of formalising their endeavour to become more disciplined. This move also helped the division corporately in response to the internal criticism over the SPECTRUM4 project and the subsequent poor sales record in 1995. The divisional management 'wanted to keep in with [the initiative to become ISO compliant] and look like we were supporting them and not against them as some of the other divisions might be'[19] because they were in a difficult position politically. So within the broader company context the timing was right for them to try to formalise their processes.

---

[19] Interview 25

In attempting to follow ISO 9001, the organisation introduced a corporate quality manual as expected by the ISO standard. The quality manual included guidelines and standards in quality management, project management, software development practices, verification and validation, and change control procedures. Although this corporate initiative was abandoned later, the processes and procedures were defined to reflect the ISO standard. The Market Analysis Package development team implemented many of the practices outlined in the manual. The software team adopted the standards in principle and added to them through writing relevant programming standards for the company. The quality manual covered quality assurance and standards, but was not specific about the development approach. The technical architect defined the development process to be adopted and the way the corporate quality management system was to be applied.

The consensus in the team was that object orientated (OO) development was the right approach. The team were keen to adopt object orientation to allow for future reuse. Object-oriented development is different to traditional functional development in that the objects act independently rather than as a whole. Object-oriented systems are considered to be easier to maintain as classes can be changed as stand-alone entities, and changes should be more localised. Classes can easily be reused in different applications because each one is tested independently, thereby creating a library of classes.

It was through the influence of the technical architect that the team selected C++ as the development language. Pascal had been used in SPECTRUM4, but that was not an option as it does not provide support for OO development. Other options such as early

versions of Delphi, which had evolved from Pascal, might have offered some advantages in the reuse of skills and in simplifying the development of the user interface. Whilst Delphi had some support in the team, it was generally thought that C++ would allow for the complex programming required in the Market Analysis Package product and would support the OO development well.

All three teams initially adopted the new software development practices. However, the precise nature of the implementation varied amongst the teams and through time. Some of the practices in the quality manual were ignored or only partially implemented. Many of the informal practices previously used remained prevalent. Commercial and political pressures began to build. This pressure to deliver started to undermine the new processes that had been put in place, as one developer described:

> some things we [had] decided we were going to do were abandoned. We had the core team which did inspections. We had the UI team which said we are getting close to the deadline so we will abandon these inspections and that's what happened. The intention originally was to go through and do this, that and the other and then it was just panic.[20]

Whilst the project manager protected the team from much of this pressure, it was evident to all concerned. Those who had less experience of the new software engineering approaches abandoned them. Their initial experience had not shown a significant benefit, rather they found the additional processes a burden. The team resorted to a short term view of producing software as quickly as possible for the

---

[20] Interview 9

testing team, believing that they could 'worry about the process later'.[21] It was only when asked to reflect on the project after it was completed that they realised that if the process was under control it might help rather than hinder.

Project planning was introduced such that tasks were identified and scheduled. The team leaders and project management did the estimation of time. The project management did develop a fully integrated set of plans for the product launch into the global market. The plans were thorough and incorporated the software, data, help, training, user documentation, multimedia tools, translations, installations, and marketing aspects of the project implementation. These overview plans supported the coordination across the teams, and were built on the plans from each team. However, these team plans were perceived to be flawed by the developers because the planning was often done by the project manager without a full understanding of what was required in the tasks. The team leaders 'would come up to a developer and say "you have got to build a wizard, we don't know what it is going to do yet but how long is it going to take you?" That would be built into the project plan which we wouldn't really get explained to us'.[22] Thus, the plans were treated as insignificant by the developers.

Once the Market Analysis Package project started to slip behind its scheduled delivery date, management pressure came on to speed up the development. Short cuts in the processes were made in an attempt to achieve this demand affecting the way the system was designed. The core system team, managed by the technical architect, did maintain confidence in the process. By taking their time to get the design right,

---

[21] Interview 25

ensuring that each class and document was inspected before proceeding, they found that they quickly implemented that part of the system. This does not imply that there were no problems with this part of the package, as processing speed was a key problem that had to be fixed early on, but they did not have the level of defects and crashes that the other areas experienced. This relative success shaped the process changes both through changing the defined processes and the enacted processes; the processes used were consolidated and those involved in the team learned through their actions.

These events could imply that the change of personnel or processes had no impact. This is far from the case, but it does mean that the division had not resolved the problems in one move. The Market Analysis Package development had been an improvement over SPECTRUM4:

> what we have been able to do with Market Analysis Package is bring the thing under control and say right these are the limits of it and we are not going beyond that because it is just not economic to do so. In SPECTRUM4 we said 'oh we could do that'. And so much more discipline was put in place with the development process. much more discipline was put into place in terms of the functionality that went into it. Much more discipline was put into the testing. So it was all about discipline, planning, control, [and] bringing order to the development.[23]

---

[22] Interview 20
[23] Interview 21

A review of the processes used was undertaken soon after the initial release of the Market Analysis Package. The Software Package Development team within the GeoMarketing Division initiated a set of improvement actions in 1997 and, in due course, this was formalised into a software process improvement programme in April 1998. The technical architect revised the previous life cycle document, incorporating lessons from the original MAP project. It was intended to provide a usable, pragmatic approach for the development of PC systems in the division. The new document was not overly prescriptive but did state that it was 'compulsory for all new development work'. However, whilst the formal view of the developments was that the life cycle was followed the actual process varied depending upon the developer. The style of documentation, the order of activity and the avoidance of tasks were all areas of variance that were noticeable. In part this can be explained by different working approaches within the framework provided by the new procedure. However, it was evident that many of the developers were more comfortable in the latter stages of the life cycle, especially the programming. A continued lack of knowledge and skill in OO design remained an area of concern and a source of conflict between some developers and the technical architect, who saw himself as the keeper of standards. He thought that the team were 'still struggling with how the development process should work'.[24] An alternative explanation highlighted the contrast between the language and the actions of the actors in the case. The project manager purported that one reason why the team did not always follow the process was despite their statements that it was useful 'they don't just go and do it' implying that they did not always find it helpful for producing the product.

---

[24] Interview 19

Another area that was intentionally changed in 1997 was the project planning method. Following the initial release of MAP, the new project manager attended a course that included some fundamentals of project planning, such as the use of Gantt charts and work breakdown structures. Gantt charts had been used before, but the course introduced a greater degree of rigour in the creation of the work breakdown structures (WBS), using the three categories of tasks identified in PRINCE: Quality, Management and Product. The work breakdown structures and schedules produced for the developments starting in the autumn 1997 all followed the new development life cycle. The WBS for the data mining system was developed from a mixture of the course notes and the development process description. These ideas were shared with the other members of the team. This WBS format became the de facto standard for the others. This helped in ensuring some consistency in approach. However, each developer interpreted the content of the template slightly differently. Ideas about the interpretation of skill levels, task complexity, loss factors and contingency varied between the developers. Another example of the variation was in the estimation of the effort required for testing. The estimates for testing were often on the low side. So it was widely felt that further improvements in planning were still required.

Following these changes to the defined processes, it was decided to formalise the changes under a process improvement project. In April 1998, a divisional away-day provided a suitable opportunity for the software management to initiate the SPI activity. This meeting may be seen as a watershed, but it should not be seen as an absolute starting point because, as identified already, many changes and suggestions had already been made. It did, however, allow for a formalisation of the improvement

activity. The official SPI project lasted 18 months and is discussed in the following sections.

## 4.7 Software Process Improvement Project

### 4.7.1 SPI Life Cycle

From the beginning there was no intention to follow a normative model, such as the Capability Maturity Model. The perception of these models within the division was that they were constraining and would not suit their needs. In fact the organisation did address, or attempt to address, most of the key process areas at levels 2 and 3 of the CMM (see §4.8), but in the order that they felt suited them. No attempt was made to investigate other options, such as Bootstrap, principally because they were unaware of them. The software management team preferred an action based approach, designed to address the issues as they perceived them. This action based approach was similar to the cyclic, continuous improvement models. The approach undertaken will therefore be compared to the IDEAL model discussed in Chapter 2, as this has become an industry standard.

### 4.7.2 Initiating Phase

The team management were explicit about the stimuli for the SPI project and the problems they were trying to address. The principal aim of the process review was to the enable the team to move from an ad hoc to a more disciplined process but there was no definition of what these meant. The SPI project was initiated because the new software director felt that they had come off 'such a bad streak' and did not want things to remain that way. It was expected that, by improving the processes, the software development would be more predictable and controllable.

It is often stated that successful process improvement projects are linked directly with the business goals, such as being able to produce software more profitably or to reduce the time to market. The project was not specifically coupled with such business goals, other than implicitly through the team's desire to achieve the targets set them. Had the improvement project goals been explicitly linked to the business goals it may have helped to ensure that the necessary resources were made available because, whilst the senior management approved of the SPI initiative, it was seen as a non-critical activity for the business. The literature emphasises the need for management support in the process improvement. There is no doubt here that the management wanted the development process to improve, but their actions sometimes contrasted with this expressed objective. They consciously overrode development processes and SPI tasks were sidelined in favour of development tasks perceived to be more urgent (see below).

### 4.7.3 Diagnosing Phase

The second phase of IDEAL reviews the current state of the processes in order to be able to set the priorities for the future. It is assumed that the existing processes are documented and assessed during this stage, but here the team had recently revised their process documentation. The revised process definition at InfoServ reflected some of the nuances of the practice from the MAP version 1 development. However, as discussed in §4.6, the espoused process was not the same as the practice, so improvement against a documented process would only solve part of the problem.

There was no formal assessment of the existing competence in each of the process areas against a benchmark. This lack of assessment meant that the improvement was not objectively measurable, and so it was difficult for the organisation to know how

much they had improved. However, subjective evaluations were made and are discussed later. The SPI initiating meeting in April 1998 acted as an assessment of the existing processes. The meeting itself was a recognition of the need to address the quality, management and development problems. In his approach to the meeting the software director wanted this to be an inclusive activity. All members of the development group were involved, but it was observed that many of these were less than eager to participate as demonstrated by their body language. The software director recognised that the developers would not naturally and willingly want to be involved but by including them at this stage he was able to: acquire knowledge about what they perceived to be the main problems; develop a group consensus about the principal objectives; generate some level of ownership of the problem. Once the meeting was underway, peer pressure built to encourage participation through a series of group and voting activities. A brainstorming and voting session identified the key problems facing the software development team. The voting identified a 'top ten' list, which was discussed by the software team leaders after the meeting.

### 4.7.4 Establishing Phase

The IDEAL model identifies the need to strengthen the process infrastructure. The SPI infrastructure set up at InfoServ was relatively lightweight. The steering and software engineering process groups recommended by Curtis and Paulk (1993) were not set up. The software management team acted as both. Also, no specific time or resources were allocated to the project but the whole team was involved in this project. The software director believed from the outset that SPI should be an inclusive activity as participation helps to generate a common perspective and engender a cooperative approach (in line with Mathiassen et al 2002; Conradi and Fuggetta, 2002). The approach and infrastructure for the SPI project were therefore designed to

achieve this purpose. Process action teams were set up to look at the key areas for improvement.

A set of actions were identified from the initial SPI meeting and passed to the process action teams. The selected actions were not a direct match with the most voted for but bore a close resemblance to that list. Each of the action teams had a month to further investigate their problem area before producing a short statement about the problem and deriving a set of more detailed, measurable objectives. They were asked to prioritise the list with three 'quick hits' or high priority actions that could be achieved within three months, three other 'big hitters' that could be achieved within the following six months, and three others that are good ideas but not likely to be achieved in that timeframe. The lead members of each group came together to discuss their findings. However, it was a relatively quiet meeting: each presented their points and the others listened with minimal discussion or questions; the software director did most of the talking – championing the project. Actions from the working parties were agreed and allocated to individual team members. The allocation was sanctioned by integrating them into the appraisal activity.

As mentioned above, the team was encouraged to engage with the improvement activity, both prior to and during the formal SPI project. Before the project, the desire to improve the processes was communicated to the team and sanctioned through the company pay scheme. Those who were perceived to improve their own activity, or more importantly improve the team activity, were rewarded accordingly. This prior position was one reason that Tyler wanted to link tasks to appraisals. However, not everyone was interested with some tending to avoid the actions unless prompted. The

eighteen months following the initial SPI meeting showed that the individuals who were interested in enhancing the process put the effort into making the change happen. The benefits of undertaking such a change programme were unclear to many, as they had seen how the rigour of software engineering was negated by events outside of their control. They felt that the quality of the development was undermined by senior management decisions on product development, such as the outsourcing decisions, late changes to the requirements and short-cutting on the planning. These perceptions compounded the opinion held by some developers that the SPI task was a management role and therefore should not include them. This attitude, of course, contrasted strongly with Tyler's stated position.

Perhaps the biggest problem resulting from the non-separation of the process improvement management from the product development was that there was no specific co-ordination of the process improvement actions: the individuals were left to get on with the tasks at their own pace. This enabled the project to be flexible to the local needs, but meant that some developers could let their tasks drift without fear of management pressure. By having this dual responsibility the leaders themselves tended to prioritise the product development work. The focus of the software management remained firmly on the product development as evidenced by the content of the minutes from their team meetings. Out of the 22 team leader meetings between April 1998 and September 1999 the SPI project represented a minor part of the discussion: only four specific references were made to the SPI project, seven ad hoc process improvements were mentioned in passing and on eight occasions implemented SPI tasks were identified.

### 4.7.5 Acting Phase

Defining new or revised processes is seen to be an important part of the acting stage of IDEAL. So, as described above, each person was asked to address an action that was appropriate to their level of experience, with more junior members of the team receiving the more straight-forward tasks. Each change was introduced and evaluated separately. The monitoring of the introduction and the lessons to be drawn from the innovation relied on the judgement of the developer(s) / manager involved. Two further team meetings were held to inform the whole team about the process improvement initiative, and these acted as the foremost means of sanctioning the project. The first meeting in November 1998 outlined what was planned and why. The second meeting in July 1999 allowed the team to share their progress to that point. During the SPI project the actions fell into three key areas: project planning, software development life cycle, and quality assurance and control. The planned actions and unplanned actions in each of these areas are discussed below in turn.

Project Planning Processes

As described above, there had been a number of initiatives to adopt more repeatable project planning and control processes. These changes to planning were perceived to have helped, with the survey in November 1998 showing planning to be more effective in the MAP team than a year before. The feeling was that the improved planning had reduced the level of the problem so that 'dates do not drift as much'.[25] By contrast the estimation of elapse time and staff effort was still not considered effective, and indeed the recording of actual effort and maintenance of the plan still require improvement. So a relatively high number of actions related to planning and control were identified in the initial working groups. The actions primarily related to:

improving the estimating of effort and time required for each project; improving the planning between teams; improving the progress reporting and post-release reviews. Each of these planned areas followed a different course. Work was undertaken in an attempt to address the need for improvement in project estimation, but little was to come of it. The planning between teams was addressed but not through the formal SPI action. How this occurred is discussed as an unplanned action. The release meetings were initiated as planned and proved useful.

A group involved in the estimation area formed themselves into a process action team. Existing mathematical models such as Boehm's COCOMO model were evaluated to see how helpful they would be if applied in future. The models did not correspond to the team's experience. (In fact the software had been produced with much less effort than suggested by the standard models.) Instead of adjusting them as the literature recommends, they were rejected. No attempt was made to try to develop their own. The software engineering literature emphasises the need for capturing the actual effort taken and comparing it to the original estimates. By creating this feedback loop, the estimations should improve by adjusting personal perceptions or building mathematical models. The problem here remained the lack of useful data about past projects.

Attempts to collect data by introducing time management systems had been made in the past, but either the data collected was not precise enough or the system fell into disuse before the dataset became useful. Despite the view of the process action team that collecting data 'would be a good thing to get away from the problems of lack of

---

[25] Interview 4

time which leads to sloppy coding',[26] the team did not consider it a priority to make this part of the routine project activity. A short term attempt to address this need was made by collecting a snapshot of data but this provided minimal useful information. The action team's unwillingness to put their own effort into collecting this data can be explained by the tendency for deadlines to be changed. Developers felt that plans were frequently changed to suit customer demands, affecting the team's perception that the plans were only temporary so getting them perfect was a false hope. These perceptions were reinforced by senior managers sometimes pressurising the developers to come up with instant estimates of a job without full consideration of the tasks. One unforeseen consequence of this management action was that poor estimations were communicated to the customer as though it was absolute and so it then became a problem when it was missed. By the end of the SPI project no substantial progress had been made in resolving the estimation problem from the formal activities.

However, the process in use continued to change through practice; one example was the estimation of inspections and reviews. Whilst there was some allowance made in the original plans for reviews, software inspections and document reviews were not explicitly defined in the initial work breakdown structures, but this changed over time. Developers recognised the need to estimate based on an understanding of the volume of work and number of people to be involved, not just their own input.

Review meetings were held at the end of each phased release from version 1.7 onwards, in line with the planned action. These meetings ensured that the whole team

---

[26] Interview 20

was informed about the work that had been undertaken in the development. The actual information provided was rather superficial, but the review meetings provided an opportunity to discuss issues in the development. There was no formal audit of the project to look at the detail of what happened, whether the plan was followed and specific problems that have been identified. This type of audit could have helped to identify the lessons before the next project. One outcome from the review meetings was the identification of a lack of confidence in object/class design, which was addressed with a training course.

Two substantive unplanned changes for project planning were also introduced following external training: estimation of testing and the adoption of Goldratt's Critical Chain approach to planning and control. The quality assurance manager attended a specialised course on test management that included estimating testing effort. An attempt was made to use an approximation of the number of defects as a guide for the number of test cycles and therefore the effort required. The team found this helpful but the estimates were disputed by the management, as they were often higher then desired (testing was usually the area squeezed for time). However, this data gave the team a clear basis to refine estimates as they gained experience, and adjust the test plan when time was reduced by management decisions on release times. The priority for using organisational resources for testing was low. So there was difficulty in justifying the plans. The problem was that 'management think it will take a couple of weeks to test something when it will take months'.[27] This view contrasted with the voiced desire by the managers for improved testing.

---

[27] Interview 14

The project manager first became aware of Goldratt's Critical Chain approach to planning and control during a seminar series organised by the local university. Critical Chain is slightly different to the normal critical path approach as it allows for the scheduling of resources as well as tasks, and uses the concept of buffers to control the progress and any overrun. The project manager drew on the theory as a framework to be adapted for his own use in the project: aspects were applied that he felt suited the needs of the team and others were dismissed as unnecessary.

The introduction of Critical Chain planning proved successful on two sub-projects: a sub-system in MAP 2.0 was ready ahead of schedule; the whole of the version 2.1 development was also programmed on schedule despite problems in one of the sub-systems. Problems of the nature encountered in version 2.1 would have traditionally caused several weeks delay, but the advance warning given by the new approach helped to avert any problems. A key part of the theory that was used was the setting up of a buffer by reducing the main task estimates. Buffer management helped the team to recover, as extra resource was allocated when it was first noticed that the buffer was being used too quickly. Previously, it would not have been until much later in the project that the delay would have been noticed. Despite these initial successes there was scepticism from the others in the team about the methods, because they felt it was unrealistic to reduce the task estimate to create a buffer. Despite this resistance the ideas were disseminated across the team because the project manager felt that it addressed the original issues that remained unresolved by the process action team, that is the achievement of release deadlines announced to customers.

Planning across the whole software team had also been identified as a problem at the initial SPI meeting and was allocated to the software director. However, it was not until Bridges, the project manager, recognised that it was also necessary for him to solve this problem in order to avoid the constant complaints about the products being late that something was done. He noticed that whilst he could develop the software on time for his schedule, it would take much longer to test the system and get all the periphery material together. Bridges felt that the managers above him were placing too much emphasis on the software delivery and not the whole product. So he applied a second aspect of the Critical Chain theory, the ability plan across multiple projects and areas, helping to bring the planning of the whole product together by understanding the constraints. The project manager worked through the theory, identifying the testing team as the critical resource helped to ensure that the work flowed through that area at a better pace.

Software Development Processes

The second area of actions was related to the defined software development life cycle. The software director was a strong believer in having a life cycle to work to and, despite the noted variations, both he and the package development team felt they were now in a much stronger position than when he joined. The defined process had helped to re-establish business confidence in the team and assisted in establishing consistency in the process. In general the development was more organised with initiatives, such as a defect database and the use of configuration management software, helping to manage the changes to the software. Many of the developers noted that the improvement in the design documentation and the programming style was evident when reading through previous areas of the development.

Once the initial pressures to improve the life cycle reduced, each of the communities-of-practice in the software team responded differently. There was no attempt to make the decisions about how to apply the defined process for each sub-project congruent with each other. The actual processes enacted were not primarily related to the sub-project but to which community-of-practice the developer was associated with. The technical architect's disciples were very keen to define everything following a purist object-oriented view, and followed the life cycle as defined. The data mining group also had a strong emphasis on the design stage, but with less emphasis on the programming aspects of OO. They tried to follow the defined life cycle, but when the deadline pressure came on the group they abandoned or delayed aspects such as the inspections. In the third group, the MAP version 2 developers, the focus on programming was evident with object oriented design undertaken only as a consequence of the implicit approach of the team and the system architecture.

In the initial action planning, maintenance and bespoke software activities were identified as areas that would benefit from a defined process. The software director used the apparent benefit of a life cycle to persuade the bespoke team to define their own process. A process was defined based on an iterative approach refined to resemble previous bespoke projects. The help desk manager defined the maintenance approach that reflected the current approach to maintaining MAP. So, definitions for each of these areas were produced based on current approaches reflecting the current norms. However, neither defined process was actively adhered to: these definitions had fulfilled their obligations under the SPI project but the software director's rhetoric

about using a consistent process was countered by the actions of the relevant team leaders.

One area of the life cycle that received significant attention during 1999 was component based development (CBD). CBD was not on the original actions list but addressed a long-term desire of the managers to produce reusable software. CBD offers a different approach to the development of multiple software systems. It is a more manageable form of software generation. Software development is achieved through 'component selection, evaluation, and assembly processes where the components are acquired from a diverse set of sources, and used together with locally-developed software to construct a complete application' (Brown and Short, 1997, p112). Externally sourced components in the Market Analysis Package included generic software such as a database management system, freeware available from the Internet such as Windows control objects, or specialised software that requires expertise not found internally such as the data mining technology. The size of components can vary, but the focus across the industry had been on small, technically oriented components. During 1999, InfoServ began a strategy to create a set of 'business components' to support future developments. By raising the profile to a business level it was anticipated that the components would be used to support the development of an application portfolio. This strategy evolved from an experimental activity to show how one of the outsourced developments could have been developed using software from within the MAP package. The components were perceived as the business building blocks that make up the final products, which would have a 'closer

affinity with our business. So the people who actually make the business decisions can actually understand it, can understand what we are doing'.[28]

There was substantial interest in this approach to developing the software because of the opportunities it gave to the organisation in meeting fresh market opportunities and increasing the profitability on the bespoke work. Its implementation was agreed at a Board meeting in early 1999. According to the software director, it was 'a fundamental objective for Bridges in the next six months' to introduce the right environment such as 'training, control management, testing and documentation standards all that sort of stuff, and for actually making sure we do start work on it'.[29] However, despite the clear intention to bring this strategy about the company never achieved this goal. Two constraints were perceived across the team: knowledge of the development techniques to succeed and the lack of motive for many in the team who felt threatened by this change. The reasons for this lack of success are discussed in the next chapter.

Quality Assurance and Control Processes

The third area of the SPI project was quality assurance and control. Significant improvement has been achieved in checking the correctness of software. In the SPI plans four specific actions were identified for this area: to clarify who should be involved in which inspections; to identify and document the current development standards in use; to specify criteria for each software release; and to hold end of project review meetings for each release. The degree of completion of each of these actions varied and other actions were introduced especially in the software testing

---

[28] Review 20

area. For each area the actions prior to and during the SPI project are discussed, both planned and unplanned.

The wider use of the software review and inspection process enabled early identification of problems in the design and the programs. The members of the team involved in defining the life cycle saw early review of the software as important. The inspections were widely seen to have had a positive impact on the quality of the software. The documented process in the quality manual followed the original theory as introduced by Fagan (1986). The actual process was different as aspects were removed or changed to suit their needs. Whilst this revised process had not been formalised by the team in the documentation, the activities undertaken in each inspection were relatively consistent. Inspections were planned for in the schedules at appropriate times, but whilst the team followed a consistent process for each inspection they tended to leave the actual review until later than planned. This delaying action had become a norm in some communities-of-practice within the team. Whenever there was any perception of time pressure these were the first tasks to be rescheduled, a habit that had formed in the version 1 project. The delay was not just some unforeseen drift, but a deliberate act of avoidance, as one developer stated: 'I purposely put off these things because ....I can't be bothered to do all the setting up'.[30]

So, as well as drawing on tradition to sanction their delaying actions, some developers began to believe that the inspections were not always that valuable. The technical architect, who found this delaying tactic illogical, felt that they did not see them as

---

[29] Review 20

'real work - it's not part of the development process for them'.[31] Significant defects, therefore, were not discovered until too late or they had already been resolved by continuing the development, thus adding to their own reflective sensemaking that the inspections 'don't often pick up things of value'.[32] This action was countered by one member of the original core team, who took on the role of championing inspections and trying to get the developers to schedule them for the MAP development.

It was this developer who had the responsibility for the action of specifying the roles required for each type of inspected item (e.g. user interface design, class design). Because of his previous role and a belief about the importance of inspections in the software life cycle, he actively addressed this action. He consulted others across the division, and achieved consensus about the categories of inspectors required for each document type. It was also agreed to distinguish between documents that needed full inspection, and should be done immediately and those that could be more informally reviewed. This resonated with the developers' views that not all inspections were valuable. In differentiating between the different tasks this way, he managed to persuade the other developers to follow the defined inspection process in future developments.

The inspection activity would have been even clearer had the standards to be applied been defined for each document type too. It was for this reason that the action plan highlighted the need to obtain a clear index of the standards in use, where they could be found, and where there were needs for new standards to be defined. The developer that was allocated this task felt apathetic about the task, perhaps not understanding its

---

[30] Interview 12

purpose. She was involved in a major development that was time critical, and so ignored this action until required to report on progress.

The remaining planned item was to define release criteria for software. This action came from a product / process duality that had been observed when the pressure to release the software means that the developers agree to release it too early. However, the quality manager redefined this action to something he felt should be done (and would also achieve this goal): the implementation of quality planning. For each new project or major version a quality plan was to be produced. Each plan would link with the scheduling activity to consider key QA areas, such as: risk, release criteria, testing strategy, inspections and reviews required, and change control. This idea is a key part of ISO 9001 thinking, ensuring that the project manager has pre-empted any potential problems and risks. It also ensures that the process to be followed is based on the standard and any deviation is identified for all to consider. Yet despite this intent the action was not followed through. The reason for this is believed to be, in part, the personal skill of the quality manager to facilitate the change; he could not persuade others to support his view.

One area that had no planned actions was software testing. Testing had been improved through the application of an automatic testing tool for running tests in 1997. The introduction of a testing tool enabled test plans to be created within the software and then automate those tests, so they can be rerun when changes are made to the software. So whilst test status was still rather unknown at times during the

---

[31] Log
[32] Log

development, knowledge of outstanding defects prior to the release of software was improved by 1999.

One change that was introduced by the testing team members was the simplification of test planning using summary test plans rather than full test scripts. This reduced the costs involved in developing a full set of test scripts. It was considered possible because of the increased experience of the team, and therefore less need for prescriptive scripts – especially on smaller projects where the cost / benefit of full tests was less. This change helped to make the tests more flexible to changes in the system. The testing team were also involved earlier in the development life cycle, which had been mentioned in the initial SPI meeting, but was not part of the specific tasks. This early involvement enabled them to build an early understanding of the new functionality and to test areas of the system at the earliest possible moment.

Another significant, unplanned change to testing was the introduction of a beta test process to address the need for client involvement. In an attempt to stabilise the product before it was released to the whole client set the concept of beta releases was gradually incorporated and then formalised. The team and business staff widely commended this change. The benefits of this action were an improved image when the product went to market and an improved confidence in the account management team. The downside is that there was a delay in making the product available for wider release – approximately three months for MAP version 2.0. However, in business terms this delay may not be that significant as account managers were previously withholding new software releases from their clients until the product stabilised.

The beta testing concept is common in the software product development environment. It had not been used before in the GeoMarketing Division because of the belief that getting to market quickest was most important. The idea of using beta testing began to form when the team released early versions of the product to the international offices to enable them to perform sales demonstrations, and comments about the software were received in response. This reaction caused some unanticipated impact upon the development team as they had to react to these suggestions and defects. In an attempt to handle these changes the software management began to describe the process as a beta testing process, and thus tried to justify the effort required by the developers.

One debate that arose from this experience was about the term 'beta testing': different directors and managers accused each other of 'abusing' the term. The rhetoric showed their discomfort with the term and how it had been interpreted by different people. Looking back at one of the releases to an international branch that had received a high level of criticism, one of the team members referred to it as a 'pre-alpha' trying to make sense of the quality of the product in terms of this emerging process and terminology. So tainted was the term that it was felt necessary to define what was meant in an attempt to unify the interpretation. Indeed, by the time the process was documented and formally agreed the use of 'beta' was dropped altogether, instead the process became known as a 'feedback release' to avoid association with the rhetoric that had developed through this emergence.

So in conclusion, as Chapter 5 will show, change occurred where there was a specific and urgent reason for it happening, where there appeared to be more relevance and purpose in the task. Whist it is recognised that the priorities will change in any business, the project was not controlled in a manner to achieve these goals. The actions were tailored, abandoned, and altered according to the varying perceptions in the organisation. A number of actions arose from practice rather than formally being included in the improvement programme.

### 4.7.6 Learning Phase

The final stages of IDEAL document the lessons learnt and review the overall approach to process improvement. Birk and Rombach (2001) consider this stage to be the most important and should look to trigger further iterations of improvement. At InfoServ this review officially occurred in the meeting in July 1999 that indicated the end of the SPI project, but further improvements were not specified at this stage. This meeting consisted of each individual explaining the progress they had made through a short presentation. This helped to ensure the ideas were disseminated across the whole of the software team; there was no dissemination of the ideas to other parts of the organisation.

In terms of the planned SPI actions, the pace of change could be considered slow. By the SPI review meeting, only five of twenty-one action tasks were completed. A further eleven actions were in some stage of progress. Of those in progress some team members had written up a proposal for change and others did some work to establish the problem or the options, but in these cases significant effort was still required to complete the task. However, as will be shown in the evaluation below, the improvement was far better than this initial view.

In addition, the changes that had been introduced were discussed in the related project review meeting following the process innovation, and in smaller events and group meetings. So again, not all the action was officially organised. These impromptu gatherings were able to provide valuable points of discussion as the topics raised were those that team members felt needed further revision or further explanation.

No use was made of data to help the learning and evaluation. In part, this was because no benchmark had been used at the beginning, but also there was little software engineering data retained by the development team, such as cost data, so it would have been impossible to use. However, there were data sets that the software management were aware of, such as the help desk statistics and defect count, that influenced their views on the progress of the team. Examples of these statistics are shown in the next section.

## 4.8 Evaluation of the Software Process Improvement Project Outcomes

Despite the slow progress on the official SPI actions, significant progress was achieved through changes that emerged through practice and individual reflection. There had been a number of unplanned, improvised changes. Some of these were to address problems identified during the April 1998 meeting, others were to address new issues that had arisen since that meeting, but many simply arose from the ongoing activity of developing the software and were formalised. The project manager admitted that not that many things had resulted from the original SPI meetings, but by having the process improvement initiative it had encouraged the team to be more reflective so during the course of the project the process had been

reformed. Overall, substantial changes had been made over time, as Tyler put it: 'if you look at our history we have made a sizeable journey ... not ideal but practical … [it] isn't where we need to be …but […] all things will come together … [helping us to reach] another level'.[33]

There are two measures available to reflect the level of problems in the software: help desk statistics and defect statistics. Figure 12 shows the relative number of calls per client over the years 1997-99. The number of client calls reduced from 400 in July 1997 to 300 in July 1999, with an increase in client organisations from approximately 65 to 150. So, in this two year period, the average number of calls per client decreased by 65 percent.

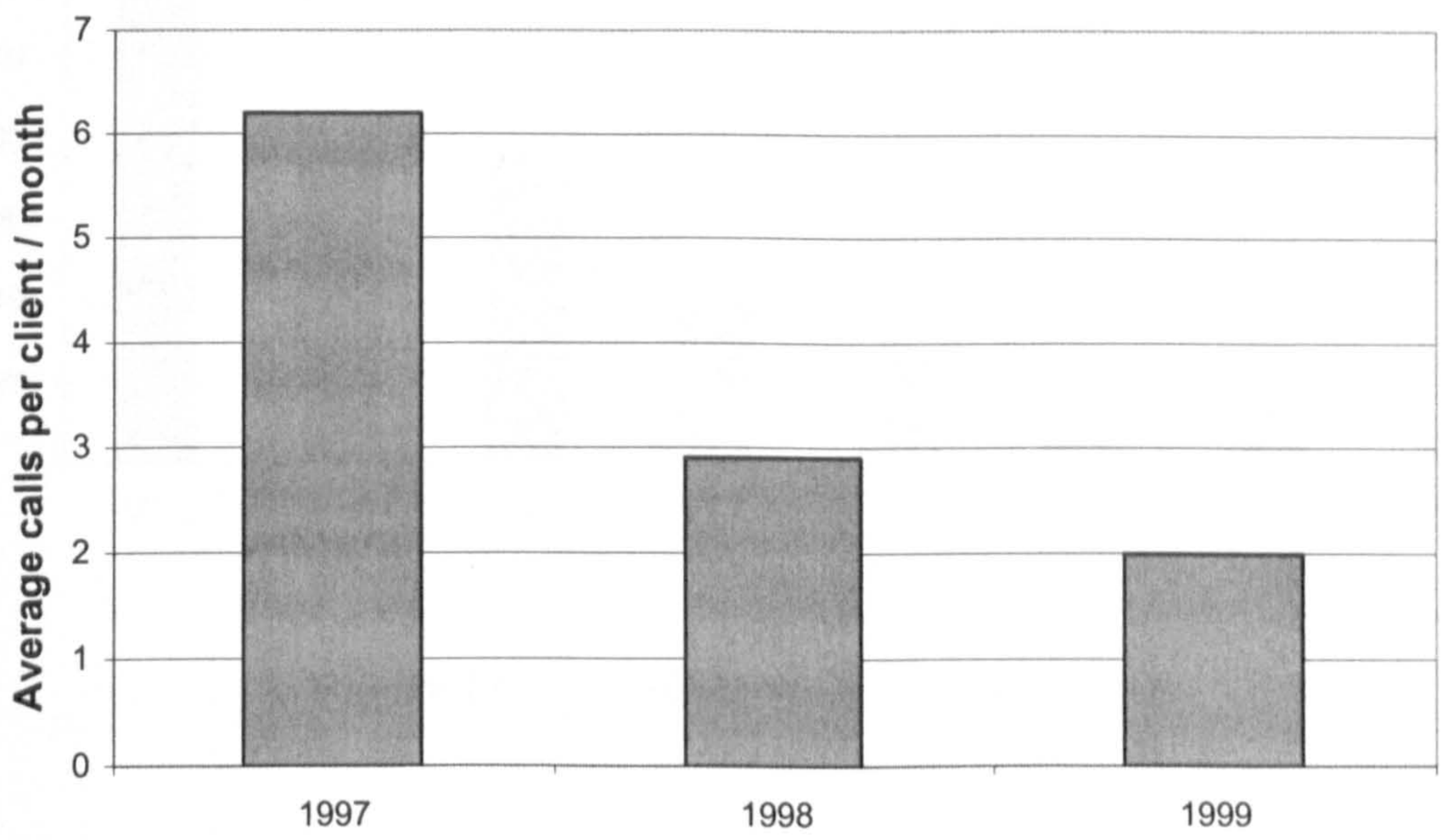



**Figure 12 Average number of calls per client in a month (based on July figures)**

Figure 13 shows the defects recorded against each of the versions of the MAP. Again it can be seen that the number of reported defects dropped significantly. Using the approach adopted in other SPI cases studies, the potential savings can be estimated from the decrease in the number of reported defects, by comparing versions of MAP

[33] Review 20

before and after the SPI programme. To assess the full impact of the improvements, it would have been ideal to compare version 1.5, which was released in 1997, with one of the version 2 releases that occurred in mid- 1999. The problem is that version 2 was far more substantial than 1.5 and, more importantly, had only been released for four months so the defect data may not reflect the final situation, and so may over state the benefit. Version 1.7 was released in autumn 1998, so whilst it did not incorporate all the changes, it did benefit from the process changes made after version 1 and some of the earlier changes in the SPI project. Versions 1.5 and 1.7 incorporated similar sized enhancements, so help to overcome any issues of defect density.

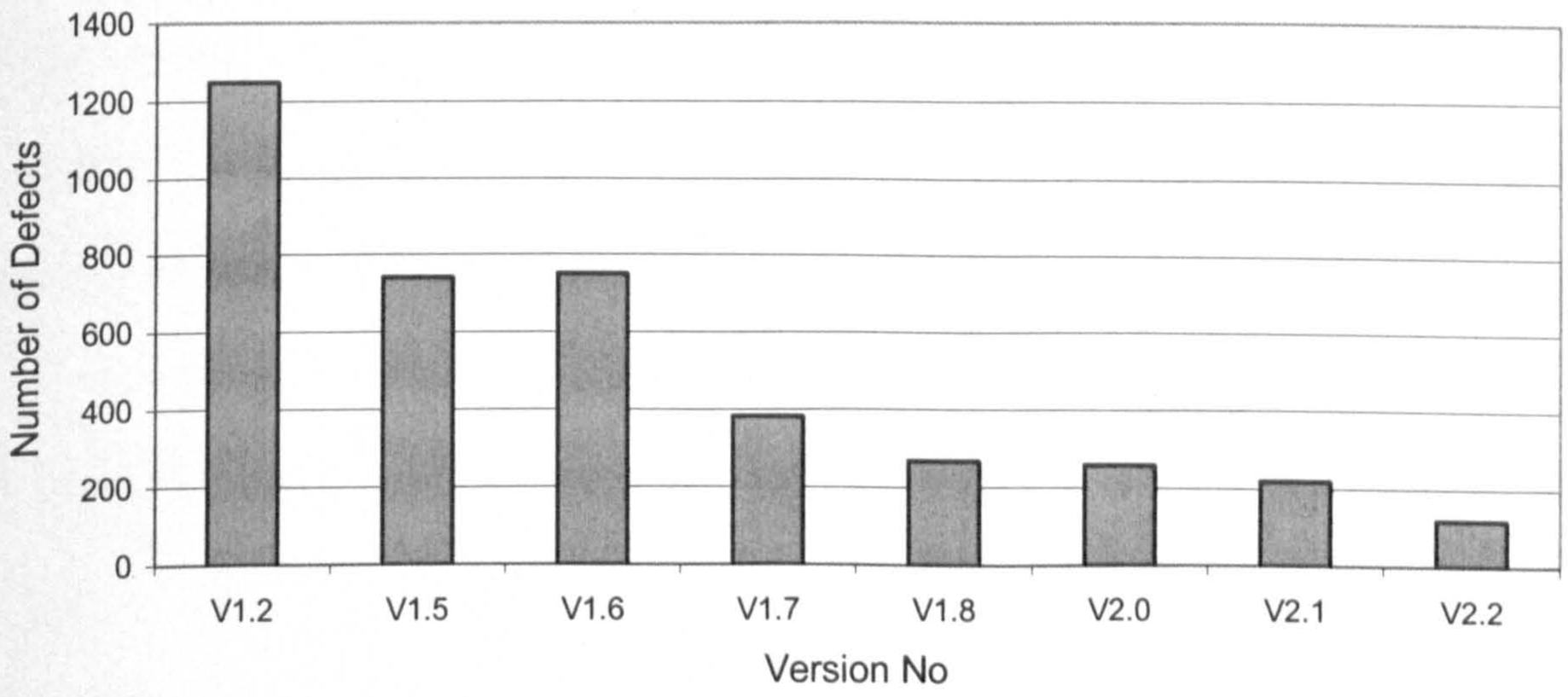



**Figure 13 Defects recorded by version**

From this data it is possible to estimate the financial benefit of the SPI project. The reduction in the number of defects evident between 1.5 and 1.7 was approximately 350. It could be argued that version 1.5 had an inflated defect count because of residual problems from the initial versions that were only discovered once more clients purchased the product. So assuming only half of the new defects recorded were problems from the new development then the difference between releases was 175 defects. From the recorded data on defect fixes, the average time to fix and test a

defect was 4.8 person days, including management overheads. So by reducing the defects by 175, then this represents a saving of 840 days saved. Assuming a cost of £200/day is a benefit of nearly £170,000 for that release alone – ignoring the time spent placating clients, time taken to address result of the error on the clients' data and the impact on the resource available for developing new releases. In addition, the number of help desk staff was reduced by one and a half in the same time period, even taking into account the other products being released, resulting in an additional saving of approximately £25,000.

The company invested in the region of 130 days working on the SPI tasks, in meetings, and relevant training, costing approximately £26,000. So as a conservative estimate, this gives a simple return on investment of 6.5, which is in the middle of the range of examples from other cases (van Solingen, 2004).

However, whilst these measures are useful for highlighting the improvements, they need to be treated with caution. The calls to the helpdesk could have fallen naturally as established clients became familiar with the product. The defect count does not take into account the density of the defects, and so does not show the improvement per size of the change. Also, for the early releases the defects recorded will include residual defects from the significantly larger and more complex, original version. Recognising these potential flaws, though, they are similar to the measures used in other cases to show the efficacy of SPI, so we should treat other case study data with equal care. More importantly, it was the indirect perception of these metrics that was assimilated around the organisation that was most important. The view of all the

business directors was equally positive, recognising the business impact that the process improvement had. For example Herriott considered that:

> things are a 100% better than what they were [...] that is largely achieved through much more structured methodology, much better documentation, much better specs, [and] a reasonable QA process. I think to some extent it is function of resource, much more realistic expectation in terms of what you get for a given unit of resource. So you know, asking for a realistic productivity. There's some quality problems still in there clearly [...] but overall I think we have come a long way.[34]

It is, of course, difficult to tell the impact of SPI to the exclusion of other factors, but it is possible to look at the business variables and how these have changed over the course of the exercise. Table 7 provides a comparative review of the case against the benefits identified in the other case studies shown in Chapter 2, it shows that similar benefits were achieved.

By the time the MAP version 2 was released, it was felt to be of a high standard and consequently a 'very strong selling point'.[35] The product director felt that the developments were 'far more under control that it ever has been'[36] because of the process improvements. He stated that the organisation would previously 'deliver things faster but would then spend a whole pile of effort fixing stuff post delivery...now we try not to deliver stuff that is going to require rectification work'.[37] Discussions with sales staff also showed that they believed that business income

---

[34] Interview 6
[35] Interview 3
[36] Interview 6
[37] Interview 6

would have been further constrained, and potentially decreased, if the improvements had not occurred. Clients had provided good feedback in comparison to Galaxy.

| Benefit achieved | Hughes Aircraft | Raytheon | PRC | Motorola | Tata | InfoServ |
|---|---|---|---|---|---|---|
| Reduction in defect density / improved product quality | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Improved team spirit / less staff turnover | ✔ | ✔ | | | | |
| Increased productivity / reduced time to market | ✔ | ✔ | ✔ | ✔ | | ✔ |
| Reduction in cost of rework / testing | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Improvement in meeting schedules / accuracy in predicted cost | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Return on investment in SPI | | ✔ | | ✔ | | ✔ |

**Table 7 Key business benefits reported**

During the review of the data mining engine, it was asserted that this was the most stable product the division had ever produced. The developments were a better quality as a result of less time pressure. The data mining development in particular had received very positive customer feedback. This part of the development team had been more rigorous in its application of the object oriented design methods, the quality control methods and following the life cycle than other sub-teams. Also, looking back at different parts of the software, the subjective view of the developers

was that there was a perceptible improvement in the calibre of the software from a design and programming perspective.

There is no evidence to suggest that the team spirit improved during this period, however the team membership was very stable. The team was proud of its achievements with version 2, but other factors counteracted this buoyancy, especially concern about how the product strategy would be implemented.

The productivity of the team was not measured in terms that software engineers would understand (e.g. function points or lines of programming / day). However, the level of variability to the project target was certainly improved during this period. In the early versions, drift in the release dates and the tactical delay of specific features until the following version were common place. In each of the version 2 releases the internal target dates were met with the planned product features. In part, this improved productivity and predictability has to do with the reduction in defects shown above. Previous versions had suffered delays because of the need to remove defects towards the end of the development life cycle. Thus, the cost of testing and rework was much reduced. Other improvements in project planning and control, design quality and quality control had also had a considerable impact on this assessment of their improvement. The estimated financial benefits of the SPI project are shown above.

In terms of the division's process capability, Appendix 7 provides an informal assessment of their position at three points in time: at the beginning of the process formalisation (1995); before the process improvement project (1997) and after the process improvement project (1999). This assessment takes each process area at level

2 and level 3 of the CMM and makes a simple judgement of the capability. The judgements are: not implemented; informally implemented; defined but not always implemented; implemented. The definitions of these are given in the appendix. It can be seen from the summary provided in Table 8 that there was a gradual move towards the process areas being informally applied and / or defined, and then fully implemented. So by the end of the SPI project the division had improved its implementation of software processes across levels 2 and 3 of the CMM, but would still officially have been at level 1 as some of the level 2 processes were not implemented.

| | Perceived level of competence | 1995 | 1997 | 1999 |
|---|---|---|---|---|
| **Level 2 : Managed** | Not implemented | 5 | 2 | 1 |
| **7 process areas** | Informal | 1 | 1 | 2 |
| | Defined | 1 | 3 | 0 |
| | Implemented | 0 | 1 | 4 |
| **Level 3 : Defined** | Not implemented | 3 | 3 | 3 |
| **13 process areas** | Informal | 6 | 2 | 2 |
| | Defined | 4 | 4 | 1 |
| | Implemented | 0 | 4 | 7 |

**Table 8 Summary of process capability by time**

## 4.9 Reflections

In this chapter the software process improvement initiative has been presented in the context of the product development history and prior software process experiences. This section reflects on the approach taken in order to establish the lessons other packaged software organisations can take from this case. To undertake the SPI

project, as shown above, the division devised and followed their own approach which closely resembled a cyclic, action-learning model. The organisation would have been better adopting and tailoring a known approach such as IDEAL because it would have informed the software management about the key stages and decisions that needed to be taken and in particular how to organise the improvement. The team leaders did discuss how to organise their efforts, but by integrating it with their development structures the emphasis and control were lost. The reason for this decision was primarily that the people who occupied the two roles that would have most naturally lead such a project, the technical architect and the quality manager, did not have the personal characteristics to do so. The technical architect had the knowledge but not the rapport with the team; the quality manager did not have the leadership ability and depth of software engineering knowledge required. So the software director retained the responsibility and delegated it through the team leaders group.

The IDEAL model stresses the need for suitable structures and assignment of resources, as well as the stages of improvement. InfoServ's only attempt to assign resources was through the implicit allocation of actions in the appraisals. No costs were estimated for these actions nor was any time allowance made in the product development project planning. The perception of the developers, both in the action setting stage and through events that followed, was that the SPI tasks were to be done when they could spare the time. To have allocated an SPI project leader with specific time and responsibilities would have had two effects: it would have ensured some actions were achieved directly through the leader's available time, and it would have given more credence to the project and thus given SPI tasks higher priority.

Initially the timescales were kept short, with the process action teams given a month to investigate and report back to the software management team. This tight timescale maintained the momentum from the inaugural meeting. However, this focus was lost after the teams were disbanded and tasks allocated to individuals. By allowing team members to do the actions according to their own priorities and motives meant that little progress was made with some tasks. So, whilst an inclusive approach can support the development of common goals and views, doing so with no control assumes that all will be motivated to undertake their actions. Perhaps a more productive approach would have been to involve just a sub-set of the team who were motivated to change the processes and were aware of alternative methods across the profession. Also, allocating tasks to individuals tended to discourage the sharing of ideas. Continuing with process action teams beyond the initial priority setting could have helped to maintain this teamwork. One group did continue in this fashion and whilst the end results were ineffective, the team did encourage one another to try to make progress. By keeping the initial groups together their initial investigation would have acted as a foundation, the group's 'quick hits' could have been achieved within weeks and thus encouraged each team to work on further improvements.

One way of keeping the tasks on track, as Jakobsen (1998) suggests, is through small, informal review meetings. He found that such meetings also help to spread knowledge about the innovations across the team. The reviews that were done within the team certainly helped, but having specific reviews as milestones for the SPI project would have acted as a counter balance to the demands of the product development. However, product development is critical for packaged software organisations and will always remain the priority.

Due to the style of the process improvement, the majority of the actions were problem-based. The IDEAL model also recommends identifying key process areas that need addressing as longer term actions. The usefulness of this recommendation is debatable as process areas could be arbitrarily identified from normative models, but looking at other ideas would have enlightened the development team and avoided overly focussing on current problems. The team focussed on current problems both when setting the initial actions, such as the estimating tasks, and when introducing ideas informed from external sources, such as the critical chain planning. The latter actions tended to mix problem and process-based reasons for their initiation. They were prompted by the needs from previous software practice but used externally devised methods to solve some of these problems.

As discussed earlier, the project was not coupled with the business objectives. Indirectly the objectives were taken into account through the software management team's awareness of the business priorities. However, to have identified specific business goals or targets, such as reducing the time to market, less variability of the product release against the planned deadline, or reducing the cost of reuse, would have enabled the tasks to be better aligned to these goals and the benefits of the SPI project would have been evident to the Board. Relevant business goals such as these are important for what Bach (1994) calls true process improvement: the adoption of specific processes that happen to be in a maturity model does not mean that the business will be any more competitive. A customer based perspective is the best judgement for a commercial software organisation such as InfoServ; if they do not continue to pay the licence fee, or the level of complaints rise, or the market share

reduces then the company's product quality is insufficient for its purpose. At InfoServ, whilst there was no internal assessment of these business targets, the sales continued to grow, customer complaints dropped (as defined by the defect count and help desk statistics), and their market leadership was strengthened.

So in summary, it could be argued that the variance from SPI theory was the reason for the lack of success on the actions identified from the initial SPI meeting. However, despite the non-conformity to traditional models, improvements occurred in the process that benefited the organisation. This does not invalidate the existing models, but it does show that the process of improvement is more than simply following a prescribed model.

The following chapter will discuss further how the improvements occurred through the situated practice of the developers. It is argued that the reasons for the changes go beyond the traditional view of software engineering and towards an understanding of the process improvement as situated change. However, this does not imply that the changes are somehow a random set of events that would have occurred with or without the SPI project. The project had a significant impact on the processes, the resultant products, and therefore the business. The project manager believed that the project:

> increased the awareness, it has rationalised the processes that we have been doing. So we now do what it is useful to do rather than things not so useful and most people now know what they are expected to do, or what they should do to keep a process running. [There are] probably not that many things that you could just pinpoint and say we are doing

that, that and that as a result of [the SPI project, but I think we] work differently as a result.[38]

[38] Interview 25

# Chapter 5. The Emergence of Software Process Improvements

## 5.1 Theoretical Framework

### 5.1.1 Formation of the framework

Chapter 4 provided a brief history of the events of the case as they unfolded. The results shows that the software process and its related context changed considerably over the four year period after the process was first defined. The analysis in the previous chapter was limited to understanding the changes made to the processes from a traditional SPI perspective. The discussion highlighted some of the contextual factors that shaped these interactions. The inter-relationship between the products and the processes is implied through the narrative, but so far little detail has been placed upon the way this interaction occurred. This chapter extends the understanding by analysing the process improvement as an emergent, situated form of change. It is shown that changes occurred through ongoing, incremental, negotiated improvisations and adaptations enacted by the developers and managers at InfoServ.

To aid in the analysis of these changes a theoretical framework was developed. This framework is summarised in Figure 14. The framework both provides a conceptual understanding of the process of change drawn from the case study and acts as a lens through which the case can be discussed. This conceptual model was conceived during the early analysis of the case, through reflection on the data, and underpinned by enfolding literature from the SPI and organisational theory fields. It therefore rests on a combination of the case study data and elements of the existing literature. The framework was devised to enable the analysis to be more attentive to the emergent nature of the SPI activity. Like all models this framework is a simplification of a complex reality intended to attune the researcher to key concepts within this form of

change activity. It is used to inform the research and is not seen as a rigid model to be adhered to, thereby inhibiting the interpretation of events. The aspects highlighted are not intended to be a definitive or exhaustive list, but these concepts sensitise the analysis of the software process improvement initiative at InfoServ. The theoretical framework will be used as a structure to analyse the case and will be expanded on through explicit reference to the case. The remainder of this section will outline the sources that provide a theoretical underpinning for the framework.

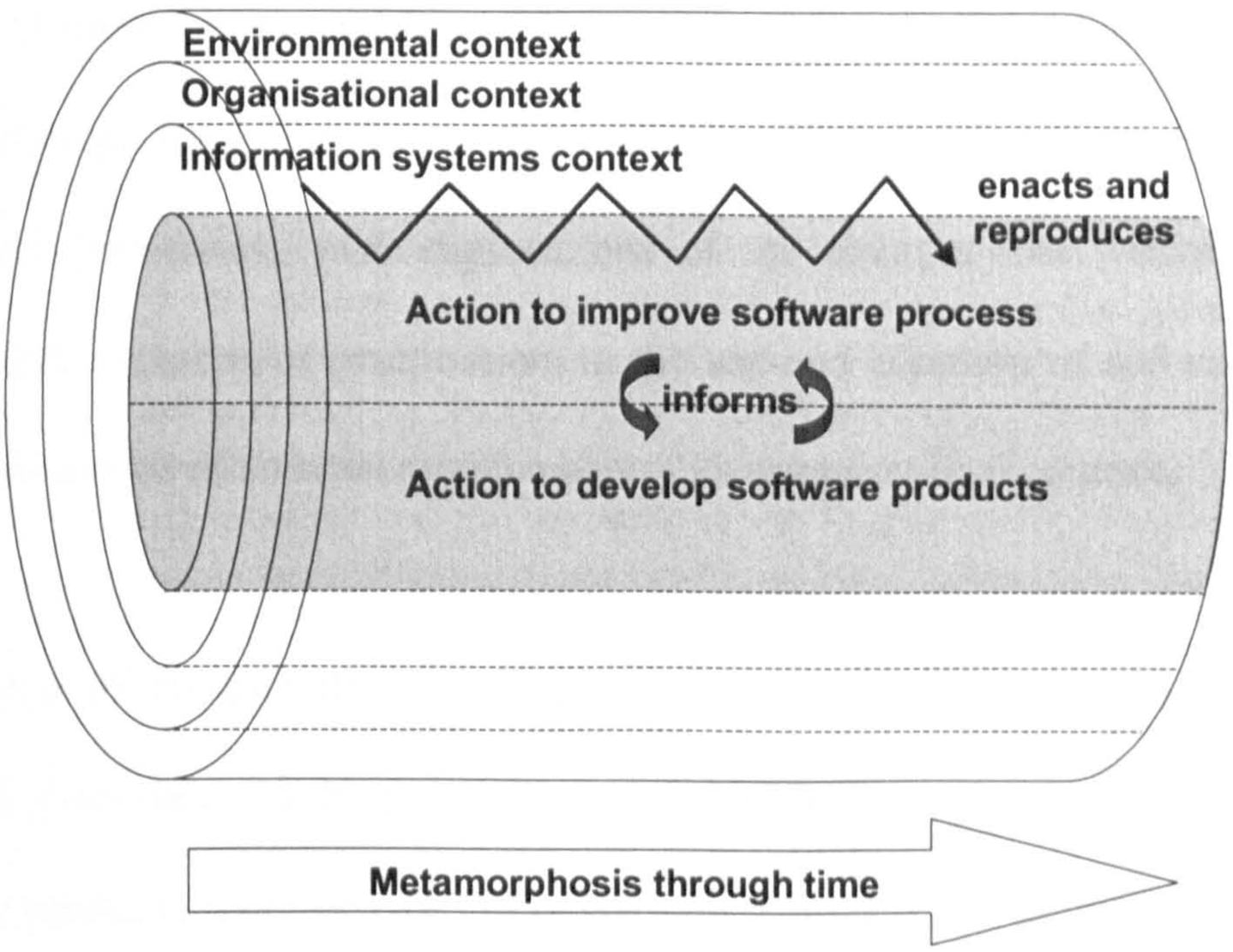



**Figure 14 An emergent view of software process improvement**

A central theme arising from the interpretation of the case was the way that the process improvement interweaved with the product development, within their specific historical context. Change is seen as an intertwining of design and action; each action either changes or reproduces existing organisational properties and practices (Weick, 1995). The background to this understanding is outlined in Chapter 2. This dynamic and emergent view of the software process, whilst contrasting with the rational nature

of much of the software engineering literature is now widely accepted by those authors seeking an organisational perspective of software practice.

In order to highlight the emergence, interplay and outcomes occurring from the software practice a contextualist and processual perspective is adopted here. The principles of this perspective are that organisational change should be viewed over time and related to the historical and contextual circumstances of the actions. Pettigrew (1987) suggests that to achieve this understanding we need to analyse the content of the changes, and how this interconnects with the context and process of change through time. The data analysis therefore includes this view of practice in the theoretical framework, with explanations of the changes that weave together the actions and decisions of practitioners in the use and adoption of software processes, and the dynamics of situated practice within its organisational context.

The content of change within the case is seen to encompass both the adaptation of the software processes and the practice of applying the processes in the development of software products. The activities of developing software and improving the processes by which it is developed are not mutually exclusive, even though for simplicity they are normally considered as separate. Process improvement incorporates the interplay between the two aspects of the software development activity: the action of process improvement and software development each inform the other. Individual actors draw on the structures within the context as they enact the software practices and thus reproduce the context. The process of change is understood to occur through the linkage between action of software practice and its context. So through time there is a metamorphosis of the context, the actors' understanding and intentions, and the

software process as it is enacted. Fundamental changes can occur when these emergent changes are sustained over time. This framework helps to recognise the difficulty in changing current practices through the adoption of new methods, and shows the need to recognise that developers apply far more sophisticated approaches to developing the software than any static process model can capture.

### 5.1.2 Context

Pettigrew (1987) shows that, by looking at the multi-layered context of the change process, we can understand how the context both constrains and enables changes to emerge and at the same time how the context evolves through individual actions. The process of change refers to the ongoing actions, reactions and interactions from the various agents requiring an understanding of how the organisational history shapes social structures. The context of the changes varies from the particular area under examination to the wider corporate and socio-economic environment. Previous SPI research has identified three layers of the context that shape the adoption of specific innovations: the environment of the IS function, the organisation and the wider business and professional context (Orlikowski, 1993; Swanson, 1994). These three layers are adopted here for analysing the SPI programme, but are not seen as entirely separate entities as the edges are blurred, rather as a way of illuminating the different aspects of the context.

The context of the IS function is the environment within which the software practice occurs. So whilst the organisational and socio-economic contexts can have significant influence upon the changes, these changes occur through the individuals within the IS context and the social infrastructure they operate in. The infrastructure is not simply the physical, hierarchical and logical groupings of teams that form the setting for the

changes, but also the relationships and group dynamics within and across communities-of-practice (see below for further discussion).

Behavioural, personality, and competence factors have been shown to have a major influence on group processes and team performance, and vice versa (e.g. Baron et al, 1992; Cusumano, 1997; Krishnan, 1998). Behavioural attitudes to the SPI initiative, arising from underlying motives, affect the introduction of new ideas and the interaction with other team members. These attitudes and motives are, partially, evidenced through individual behaviour, consequently affecting the outcomes of the SPI. Managing the social aspects of the SPI activity is often seen as the most difficult because established patterns of behaviour may need to be modified (Curtis and Paulk, 1993). So, the management and leadership style of the software activity, and process improvement programmes in particular, can be seen to shift the emphasis through the instigation of change at a local level (Jansen and Sanders, 1998; Rainer and Hall, 2003). Indeed, management strategy and behaviours are often deemed to be closely related to the success of SPI initiatives (DeMarco and Lister, 1987; Stelzer et al, 1998).

These individual and organisational attributes are seen to be part of the context of the action, but it is also recognised within the framework that there is a close relationship between the behavioural attributes and the behaviours as evidenced through action. The separation of these facets assists the analysis of the reasons for the changes, but this duality is explored further through the discussion of the linkage between the context and the action.

The infrastructure also includes the defined processes and standards, which are drawn on during the software practice. The process models, local and industry-wide standards, documentation and development tools act to enable and constrain the software practice. These features of the IS context are amended as they are used: models are adjusted, standards are reinterpreted, additional documents produced, and tools amended to suit the needs of the organisation. Each SPI activity has its own infrastructure, such as process activity teams, steering groups, and reporting processes. The nature of these structures will shape and reflect the improvement activity.

The resources within the organisation comprise both the relatively permanent elements, such as process models and skilled personnel, and the capabilities to produce these resources on demand, such as the ability to create new documentation or hire new personnel. Competencies of individuals have a bearing upon the application and introduction of software processes. Individual capability can enable the introduction of specialised design and programming techniques. However, in part these capabilities can be changed through training (Hutchings et al, 1993). Indeed, the SPI activity itself is often seen as a knowledge creation activity (see Chapter 2), and so it needs to be recognised that as individuals change and adapt, their expertise and knowledge are part of the emergent context of the SPI and software development.

The organisational context relates to the broader infrastructures, relationships and competencies of the organisation. It also shapes the activity of any process improvement initiative through the response to the pressures, reactions, and standards of the wider environment. The perceptions of customers to the quality of the software

product, comparisons to alternative products or suppliers, the organisation's perception of the competition dynamics, such as threats or opportunities in the marketplace, vendor support for the changing practices and the constraints of any regulatory regime have all been shown to play a part in the activity of process improvement. Strategy at a business and IS level are all shaped by these perceptions and can affect the activity of the development teams, potentially increasing the pressure to deliver products in restricted time.

Packaged software organisations function under an intense time to market pressure relative to IS departments. They are under pressure to innovate, and upgrade their existing product set. When one company releases 'some critical functionality "A" in the latest release, all other firms developing for this market place are compelled to modify their development to include "A" in the next release' (Carmel and Sawyer, 1998, p.9). These time pressures are increasing as release cycles reduce in time. So, the propensity to deliver products earlier than their competitors is an essential feature of the marketing strategy for these organisations. Companies will only want to purchase one brand of a given type of application, so 'capturing pole position' in the race to the market is crucial for business success (Feeny and Ives, 1997, p.56).

The external environment also includes aspects from the professional context, such as the expertise and methods available in the IS profession, and technological developments are evident. Radical changes are often thought to require new knowledge or information from external sources (Nonaka, 1991; MacDonald, 1998). In this study, it was recognised that SPI is not solely about taking on new ideas from external sources nor are all new methods, technology, tools, etc. necessarily taken on,

even if they were known about. Even if we were to assume that novel approaches could be identified and adopted as a whole, the features of an innovation are not fixed but *reinvented* during the adoption (Swanson, 1994). However, the SPI initiative and related software development were influenced by the external environment as developers scanned for new approaches and as suppliers and customers changed the operational dynamics. So, as important as the structures and dynamics of the wider context is the perception, within the organisation, of any potential initiatives identified from the environment and the way they are incorporated within the organisation.

Additionally, the formative context for the changes comprises the organisational and personal history, and the actors' experience relevant to the software process. The conditions before and during adoption shape the decision to adopt and the use of the method in practice (Orlikowski, 1993). Organisations can be pre-occupied with ongoing concerns and commercial pressures, and focused on pre-existing arrangements. So, the analysis needs to take into account the previous experience of the organisation and the individuals involved, as their experience of success and failure in software development will influence the approach.

### 5.1.3 Emergence of Software Process and Products

Mathiassen et al (2002) show that, within any given software context, there is a recursive relationship between the software process improvement effort and the software development practices. Understanding this relationship is important if we are to see how the software processes emerge through time. Previously the link between product and process improvement has been identified on both an industry-wide and local level (McGarry and Decker, 2002; Quintas, 1994; Van Solingen et al 1999), but there is a paucity of previous work showing the ongoing interaction as discussed in

this case. Here the relationship is understood through the micro-dynamics of the interaction that is a constant and a fundamental aspect of the SPI activity.

The ongoing dialectic between the defined processes and the processes in use shape the changes to the software practice. Software development in any particular project is informed by the process improvement activity. In any given project the development activity makes changes to the software artefacts through the decisions and actions of the developers. The means they use to do this are drawn from the context, and through the application of skills and knowledge developed through previous experience. The context of software development includes the defined and routine processes, which are recreated through the actions and experiential learning in the development activity. The adoption of new processes is considered to comprise: the intentional introduction of new methods; the 'bubbling up' of ideas through developers' improvisation; and the adaptation of existing processes as a consequence of the ongoing interpretation and application of practiced processes.

Firstly, the introduction of new methods occurs through the planned activity of the formal process improvement programme. Within the planned SPI activity specific needs are explicitly identified and solutions proposed as part of the organisational improvement activity; these solutions are then designed, disseminated and evaluated. As the case results in Chapter 4 show improvements are also improvised as individuals identify and seek to solve perceived problems, or find an external solution to a problem previously identified. These changes occur through the individual efforts of the practitioner to bring about the change. These changes are designed because, even though they are improvised, they are considered and discussed prior to action.

Both the planned changes and the improvisations are therefore intentional actions in the sense that they are deliberately undertaken with the purpose of changing the long term process. Changes also occur unintentionally, or without the explicit purpose of changing the organisation's practices, as existing processes are adapted through situated practice. Small adjustments are made as a consequence of the ongoing interpretation and application of practiced processes to develop new software products. They occur as a consequence of the ongoing interpretation and application of practiced processes to develop new software products. So, whilst the actor is conscious of their action, they do not design the action to change the process.

In line with Giddens's work discussed in chapter 2, we can see that the consequences of this interplay between product and process changes are both the anticipated and unanticipated outcomes in the software process. Anticipated outcomes relate to the intentional actions, where the action results in a change that is perceived as useful and successful. Ravichandran and Rai (2000a) consider that the anticipated outcome in terms of quality is a combination of product quality and process efficiency. Unanticipated changes occur both through the developers' spontaneous adaptation during situated actions and when the preconceived changes do not occur as intended. The theoretical framework therefore recognises unacknowledged conditions of action are outside of the self-understanding of the agent.

### 5.1.4 Process of Change

Based on Structuration Theory (see Chapter 2), it is argued that the emergence of the software processes occurs through a structuring process. The structuring process acts as a linkage between the context, and the content to help to explain the process of change. The following section will explain further how Structuration Theory supports

the analysis of software process improvement in a given context. As outlined in Chapter 2, Giddens (1984) specifies three modalities to explain the structuring process: interpretative scheme; facility; norms. The enactment of, and changes to, the software processes are seen here to embody the modalities of the structuring process.

Systems development and improvement is the outcome of a complex process of interaction and communication influenced by the situated characteristics of the actors (Mathiassen, 1998). To undertake this interaction actors draw on stocks of knowledge and shared experiences to communicate about how to undertake, and change, the software practice. In drawing on the interpretative schemes the processes are reproduced and recreated, and software engineering features and defined processes act as frameworks for learning. The communication of information and meaning occurs through signs and language that change over time. 'Thus, in any interaction, shared knowledge is not merely background but an integral part of the communicative encounter, in part organizing it, and in part being shaped by the interaction itself' (Orlikowski and Robey, 1991, p.149). So, related knowledge is contested and shifting: knowledge is drawn on and changed through action. Software process improvement, therefore, can be seen as ongoing interaction, not just a pre-planned action, that occurs as the development team make sense of the specific development activity facing them. This sensemaking enables the mental models to mature through the actors' reflection-in-action.

This interaction supports an active view of knowing and learning, as outlined in §2.4. SPI understood as a reflective, sensemaking process helps to show that knowledge is changed through the ongoing use of the processes. The theory-in-use (or process-in-

use) is challenged and reinterpreted through this new experience. This single loop learning develops into double loop learning as it is shared with others within the communities-of-practice (Argyris and Schön, 1996); espoused theory changes initially through changing the interpretive schemes that individuals hold and then in turn the social structures such as the organisational patterns and defined processes. Ideas from training and other external sources are assimilated through these communities-of-practice, and the necessary discussion and negotiation, drawing on structures of signification to enable this communication.

Human agents also draw on facilities, or resources, in interactions. To draw on (or not) personal or organisational resources in order to retain or alter existing software approaches is within the control of all practitioners. This position presumes that there are relations of autonomy and dependence between actors, not simply dominant unidirectional relations. Giddens (1984) terms this the dialectic of control, whereby subordinates can also influence superiors. This influence can be both in the sense that developers can introduce initiatives and they can shape the thinking of the managers, by inspiring them, honing their initiatives and resisting them. Each member of the organisation has the power to conform or challenge a suggested change. Individuals and groups may exercise power to resist in some circumstances and not others. Individuals can monopolise knowledge that facilitates their operational practices, but it is not just possessing knowledge that is important but the way that it is deployed. The use of knowledge to legitimate action then depends upon how that knowledge is perceived, given that knowledge is contested and shifting.

Organisational politics and a lack of commitment of the technical staff towards the change programme have been identified as one of the main barriers to successful SPI (Goldenson and Herbselb, 1995). This lack of commitment is a form of resistance, where there is an unwillingness to commit the time to the task. Resistance is a feature of SPI programmes (McGill, 2001; Somasundaram and Badiru, 1992; Thompson and McParland, 1993). Resistance is often thought to relate only to subordinates, but managers also can resist initiatives from employees through their disinclination to mobilise resources. So important questions in the analysis are who resists, why they do so and how they do so. However, we must not assume that all problems are a result of a lack of personal commitment as McGuire (1996) found that even strong project teams would suffer difficulties. So the challenge is to understand the use of power but not simply from a hierarchical view.

Giddens (1979) states that structures of signification are mobilised to legitimate the sectional interests of hegemonic groups which rely on the asymmetrical distribution of resources to satisfy wants. An example of the relevance of this to the software process would be for the inappropriate use of software metrics within staff appraisals, or for a given process action group to have an unduly significant influence on the processes without the consent of the other groups, or a solution may be undervalued if the idea comes from someone with low status in the team. However, SPI structures, positions of authority, or the application of a software process do not imply that dominance is inevitable, as individuals retain a choice to conform or resist.

So, software process improvement is a constant process of negotiation, communication and establishment of norms through the everyday relationships

enacted within the process improvement programme and software development process. The norms of the practitioners become the traditions for future practice. Giddens (1979, p.219) asserts that routine is strongest when it is sanctioned by tradition. Giddens uses the idea of 'reversible time' to contend that the events of daily life do not have a one-way flow to them: routines, seasons, and recursive activities indicate the repetitive character of day-to-day life. In software practice we can understand that routine is repeated between projects, and through the reuse of knowledge embedded in methods and tools. Being careful not to assume a structuralist view of the historical events, Giddens uses this understanding of time to show that language can be used as an active process of legitimation, as it is grounded in the mutual knowledge of the institutional tradition. Developers and managers sanction their actions through the narratives of meaning and action (Fincham, 2002), by which the defined process acts as the norm and the practised process becomes the norm.

The recognition of communities-of-practice, or sub-cultures, raises our awareness of the differences, complexities and contradictions within an organisation. Thus, there are multiple meanings about particular events within the organisation. However, it is acknowledged that attempting to define specific communities-of-practice within any company is subjective and open to criticism. One view would be to assume that the software engineers were a single sub-culture within the organisation (Sharp et al, 2000). However, these groups are not necessarily consistent with organisational infrastructure or other physically identifiable units, and sub-cultures are not exclusive sets, in that individuals can belong to more than one. Communities-of-practice are informal groups bound together by shared knowledge, motivations and interests. Contemporary research that recognises SPI as an on-going activity has shown that

communities-of-practice within organisations hold multiple meanings about the software process and the SPI programme (Kautz et al, 2001; Mathiassen et al, 2002; McGuire, 1996; Ravichandran and Rai, 2000a). Rather than trying to categorise each software engineering group, here we are trying to understand the cultural features, or *habits of the heart*, that characterise sub-cultures and the process whereby they retain their distinctive character and how they interact with each other.

Each of the modalities enables and constrains the interaction within the communities-of-practice by drawing on the structures created by that interaction. Software development occurs in communities-of-practice using idiosyncratic language developed *in situ* to access relevant knowledge close to the events to be faced (Ciborra,1999; Mathiassen, 1998). Communities-of-practice therefore 'help us to understand important factors that support and foster learning and others that restrict and limit learning' (Mathiassen, 1998, p.29). Divergent interests could result in structural conflict as power is central to all software development activity within the organisation, so changes occur through a negotiation process within and across communities-of-practice. The balance between control and autonomy helps to influence the SPI action. Software processes and methods used within the group are drawn upon as norms, and in so doing recreate structures of legitimation. The norms that develop are legitimated through the shared language of the community-of-practice developed from mutual knowledge of their traditions.

### 5.1.5 Chronological Analysis

The theoretical framework has been developed from a contextualist, processual perspective, where the context of the software practices is seen to enable and constrain the practices. The interaction between the practice of developing the

products and the practice of improving the processes is considered to be intertwined, each informing the other through an on-going act of sensemaking. Software process improvement is viewed as ongoing change, occurring through a structuring process. The duality of structure interlinks action and context through a set of modalities. This view of change as a structuring process helps to recognise the difficulty in changing current practices through the adoption of new methods, and that developers apply far more sophisticated approaches to developing the software than any static process model can capture.

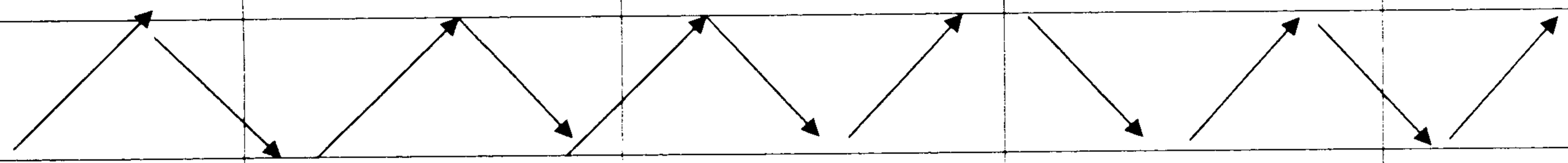
A chronological approach is adopted to analyse the emergent nature of the processes and their interplay with the context and software product development. The theoretical framework provides a lens to consider the dialectic between actions to improve the software process and instantiation of those processes in the software product development. In the analysis, the ongoing software practices will be shown to emerge from a series of situated choices. Within the case the processes can be seen to have changed through a combination of intended, planned activity and adaptive activity according to situated factors: ideas for planned changes were triggered by a perceived product need, process improvements were overcome by immediate demands for product development, and ongoing changes emerging from the product development activity. The situated practice of software development is shown to bring about a metamorphosis in the context and the software processes. The details of the changes in the processes and products are provided in the previous chapter.

Based on the diagrammatical representation of the theoretical framework in Figure 14, Table 9 summarises the key actions and the contexts in which they occurred. The

context is represented as layered, as per the framework, with the IS layer split into general IS function and software process layers for ease of clarification. The actions related to product development and process improvement inform each other through time. The arrows indicate that these actions are enabled and constrained by the context(s) and thus enact and reproduce these social structures; the placing of the arrows is not intended to be significant.

The remainder of the chapter will analyse the way that the changes emerged through situated practice. The purpose of the following sections is to discuss how and why these changes occurred. It will consider how the context influenced the process, how the software practice changed the context, how the product / process dialectic unfolded and why these things happened in this way. The discussion is split into three periods: the early years of development forms the historical context, and then the action – context linkage is discussed over the period of process formalisation and the period of the software process improvement (SPI) initiative. The framework is used both to inform the discussion and as a structure for each of the latter periods, with a section highlighting the contextual factors and a section analysing the emergence of the processes and products for each period. The factors in the internal and external contexts are highlighted so as to support the discussion of the reasons for the actions in the process improvement and product development. The nuances of how and why the processes changed are discussed in each period by drawing on structurational concepts. The reasons behind each of these actions help to explain the primary factors that influenced the shaping of the process improvement, both of a formal and informal nature, and in turn changed the social structures within the organisation. The final section provides a reflective analysis of the framework and the process of change.

**Table 9 Emergence of processes and products**

| Context | To 1995 | 1995-96 | 1997 | 1998 | 1999 |
|---|---|---|---|---|---|
| External | MS Windows becomes norm; TickIT introduced; Rise of OO | Competitor Windows product introduced; Outsourcing becomes more popular | Rise of component based development concept; multimedia development tools | Household level analysis offered by competitor. | OO Methods and CASE tools refined |
| Organisa-tional | SPECTRUM classification; Centralised IS | Move to become ISO 9000 accredited | Merger with USI; lean sales period; pressure for new functionality | Pressure from international partners for data mining tool | Move towards Internet products |
| IS function | Decentralised PC development ; project overrun | Recruitment of new IS staff; Lack of experience of OO / C++ in team | Bespoke product team formed | Portfolio of package products created; SPI infrastructure established | Switch personnel from MAP development to portfolio |
| Process Structures | No formal processes or change control | Quality manual; Standards defined; OO life cycle defined | Process redefined. Project planning framework developed | Testing tool introduced; new standards; Process action teams | New processes defined |
| Structure/ Action |  | | | | |
| **Action** Product develop - ment | DOS System becomes amorphous / distended | New Windows product using layered architecture | Poor reputation of initial MAP product; Bespoke applications become popular | MAP product stabilised & enhanced; develop new products via suppliers, with problems | Major release of MAP2 with data mining tool is 'most stable yet' |
| Process Improvement | | Develop QM but application variable; OO approach partially adopted | Redefinition of process; use of additional S Eng features | SPI formalised across the team; process refined through use | Progress with formal SPI actions slow; new process changes |

## 5.2 Historical Context: Period 1

To recap, prior to the development of SPECTRUM4 the division had gained a strong position in the geomarketing domain. They had achieved this success because of the additional sophistication that the SPECTRUM classification scheme had over its rivals. The development of the SPECTRUM PC based software package, which used this classification scheme, had helped to expand the market share for the company. The history of the development of this product, and its predecessors, influenced the software practice during the period of study by establishing norms in the approach and social structures to support their recreation.

The infrastructures of the division at this point were informal and close knit as the division remained relatively small. These relationships effected a spirit of cooperation between members of the division. The management sought to facilitate an innovative environment, whereby ideas for new products were encouraged: it reflected their responsiveness to market opportunities. As a small, growing part of the company they had seized opportunities to sell products, sometimes making changes to the software to achieve a sale. This approach reflected their perception of themselves as a small business working under the InfoServ umbrella. Whilst the division was cooperative internally, the management of GeoMarketing considered the division to be different from other areas of InfoServ, which tended to provide large-scale support services for organisations.

Another reason for the perceived gap with the rest of the organisation was the technical platform. Whilst this group were using PC based software, the technology platform across most of the organisation remained essentially mainframe and mini-computer based reflecting the external business environment. The division also

reacted against the more controlled approach of the central systems department because they thought this to be flawed in delivering business solutions. One consequence of this disparity was an early intent to be self-sufficient in IS capability, or at least not reliant upon the rest of the organisation. In order to produce this PC product, the GeoMarketing Division initially overcame the lack of PC experience by customising externally available software to produce SPECTRUM Systems. The initial success of the product formed structures of domination that the divisional Board of Directors drew on to persuade the organisational management to allow them to set up their own development team. The growth of the development team was partially achieved by buying out the principal software vendor, thus bringing the development capability in-house. The team's expertise in programming grew during this period, but typical of many PC developments of this era the methods used remained ad hoc.

The GeoMarketing management, who had no previous commercial software development experience, in due course recognised that they had insufficient knowledge of IS management. In view of the perceived need for further skills and experience, a project manager was transferred from the centralised systems department. His experience was all mainframe based, developing systems to support the service areas of InfoServ. The PC-based, product-oriented position proved to be quite a different challenge.

The problems with SPECTRUM4 arose from a combination of the ad hoc approach to the development process, the lack of control over which changes were incorporated and the reactive norm of the management. As described in Chapter 4, the development

team was cooperative but operated as a multifarious group: each person had their own functional specialism and worked on their own area. The development methods were undefined. The division's reactive strategy was a key feature of the dynamic between the context and the ongoing software development: previous incidents of reacting to commercial prospects promoted the haphazard development approaches, with requests for product changes interacting with attempts by developers to design, develop and release the products.

The organisation may have overcome the problems in the SPECTRUM4 development but for the changes occurring in the external technological environment. Until the early 1990s, MS-DOS was the dominant PC operating system in use, but by the time of this project Microsoft had introduced, and were actively marketing, their Windows platform. This change of operating system had a major impact on the SPECTRUM product set, as the industry norm altered to compatible desktop products the expectations of many customers reflected this change. The development techniques and tools are different for Windows than MS DOS, so it was not a simple change. The reasons for not switching to Windows earlier can be summarised as a mixture of a reliance on the market strength of the product, the team's knowledge of the existing technology, and the shared understanding about the software provided the directors and the software team with unconscious motives to retain their existing strategy: preferring to retain something they knew well.

The interview data shows that the directors and the developers placed different meanings on what had happened. The problems with the SPECTRUM4 development were attributed by the Board to the project management: poor control of the volume

of change, poor management of the timescale and a lack of foresight about the pending dominance of Windows in the PC arena. One director commented that:

> The chap that was in charge of programs he literally felt that there was nothing wrong with DOS and that Windows was a new fashion, a new fad and it wasn't worth moving in that direction at that time, which is unusual for us because we are usually in first with new technology but because he had what was perceived to be the most inside knowledge, we held back from changing. [39]

These traits were considered by the directors to be residual from the project manager's previous experience in the central systems department. The software team, however, perceived these views to be reinforced by the actions and leadership of the Board members, who were the ones making the demands for new functions and were equally sure that their product was strong enough to withstand any change in the PC market. The decision to retain a DOS based product was supported by the Board. This might have been a result of ignorance, as complacency about their position in the market had meant that they had not maintained a currency of understanding and relied on the judgement of one or two people.

The individual developers sustained this situation through their approach to the software development. Whilst previous developments had problems with software defects, the strength of the product in the market had reinforced the previous ad hoc approaches as the norm. So despite the problems, individual sensemaking had associated the approach with the success of the product. During the development of SPECTRUM4 the traditions from previous developments were drawn on to legitimate

---

[39] Interview 26

the informal and reactive approach. Thus, the project manager, whose experience had been different, was unable to substantially change the structures of signification. He was unable to transfer the practices from one division to another as there was disquiet about the applicability of that division's approach to software development. Individual developers and managers resisted any move to make the life cycle more structured by drawing on the team's shared experiences and norms to sanction their actions when requesting and making changes to the software. These changes to the software were legitimated as the directors did nothing to stop them, and expressed their desire for a product that would last during the development of a Windows-based product.

One consequence of these difficulties was the impact on the relationship between the sales and software teams. The informal and relatively small environment had encouraged the software and business personnel to work closely together. The sense of trust and mutual support from working together in earlier versions of the product began to break down during the SPECTRUM4 project. The project manager's personality, traditions and behaviour were blamed for this conflict, as he was 'definitely on the systems side of things' and he waged 'an us and them battle'.[40] Residual elements of the conflict remained part of the next development because 'everyone on the development team lost faith in managers'.[41] The sales staff were asked to spend a lot of time testing the system, but did not see the benefit afterwards in sales, so they became disenchanted and therefore less inclined to support the work of the software team.

---

[40] Interview 4
[41] Interview 4

In summary, the historical context exhibited the results of the strong financial growth created from an innovative approach to product development and a strong desire to support clients in their use of the products. The software development was characterised by changes to the products in response to opportunities and client requests. The developers' ad hoc processes had the effect of producing an inconsistent design and significant levels of defects.

By the end of this period the senior management were concerned about the perceived problems of the division's software development. Reflexively monitoring the situation, they were not satisfied that the current approach to software development was going to be successful in the future. They realised that the existing norms of an ad hoc approach needed to change. Richards's and Herriot's domination over the project had constrained the project manager's ability to alter the situation, and reduced his power to control the team. However, it was these problems that triggered a move towards a more structured software process. The following section analyses how the process changes occurred, given that the SPECTRUM4 project manager had been unable to change the structures of signification, and why some of the problems remained. The consequences of the SPECTRUM4 development were reflected in the Market Analysis Package project: previous good will had been eroded, the need to get a new product out soon became critical for the business, and the informality of the development approach had become a problem.

## 5.3 Case History Period 2: Formalisation of the Software Process

### 5.3.1 Context

In addition to the formative context from the previous development there were three external contextual factors that had a key influence on the software practice at InfoServ during this period: the spread of ISO 9000 across all industries; the actions of competitors in releasing a Windows product; and the lack of suitable expertise and supporting tools for object oriented development in the professional environment. These external factors interacted and blurred with organisational factors, with the customers frustrated at the DOS product, and with the existing software team competencies.

The adoption of the ISO 9000 set of standards, and its British equivalent BS 5750, became synonymous with formal quality management systems throughout the manufacturing and service sectors. The specific development of the ISO 9000-3 interpretation for software development and the UK government's TickIT initiative gave rise to a profusion of computer service and product organisations adopting the standard, partially as a marketing ploy. This external trend predisposed the corporate management of InfoServ to adopt the standard across the company.

The competitive environment became more difficult for InfoServ at this time. Once Galaxy Systems had launched the Windows version of their product in late 1995, about the same time as InfoServ were designing their new system, the pressure on the organisation dramatically increased. The customers started to become agitated. Getting the product to market became the top priority, as one director noted:

> By the time we had got to stage one, competitor products were out so we had to balance up getting something out there and getting

> something robust. We also had to balance up demand from clients. Galaxy Systems are very good at promoting themselves. Over the years they had pretty much been kept out of the client base, but all of a sudden we were getting asked when our product was going to be ready.[42]

The IS management responded to this pressure by announcing a target release date. This competitive dynamic thereby shaped the introduction of the quality management within the division through restricting the time available to become competent in the new defined processes.

The new methods adopted were identified from new ideas in the software profession (see below). However, the professional environment did not assist the adoption as there was little expertise available and the supporting technologies were immature at this point. So the knowledge of how to implement the methods was initially limited to a few individuals in the team, limiting the team's capacity to adopt and apply the new methods. So, as discussed below, these contextual factors shaped the changing software processes.

### 5.3.2 Emergence of Software Processes and Products

The key concepts developed during the analysis of the emergence of the software processes during the MAP project were: how the organisation responded to the past; the dialectic between the process improvement and product development activities; the influence of individual capabilities and related social structures; and the influence of the context throughout on each of these facets of the organisational transition.

---

[42] Interview 26

The organisation responded to the past, intending to learn from their experience by adopting a different approach. Following the SPECTRUM4 development the senior managers interpreted the events in such as way as to externalise the problems: the SPECTRUM project manager was considered ill-equipped for the role and was replaced. New industry standard methods were thought to be a solution to the development problems. There was a belief in the directorate that the division needed to improve their knowledge of PC development processes, and thereby the structures of signification began to change. Also, those directors who had previously sanctioned a reactive culture were, at least outwardly, changing their position. Their statements indicated a fresh approach and they changed the infrastructure to encourage this change. Bringing organisational resources to this through recruitment slowly changed the existing norms.

Tyler, the new project manager, was recruited with the authority to introduce new development practices. He used his own and team resources to facilitate the introduction of new methods recreating the structures of domination. The organisational move towards ISO 9000 accreditation supported these changes. The motive for this action was that they were in a difficult position corporately because of the criticism of the SPECTRUM4 project and the subsequent poor sales record in 1995. The division's management 'wanted to keep in with [the standards] and look like [they] were supporting them and not against them as some of the other divisions might be'.[43] This politically sensitive situation acted as a structure of domination, enabling the recruitment to be shaped so as to support the new approaches, thus sanctioning Tyler's agenda for change. He was able to communicate the intent to

[43] Interview 25

change by drawing on the shared understanding expressed through the ISO 9000 initiative, and to legitimate the definition of the software process by calling on the industry norms and the norm of product quality.

The team members from the SPECTRUM developments had become concerned about the way the previous processes had contributed to the defects and subsequent business problems. Lessons about the design, development and testing processes had been learnt from the previous development. They were ready to listen to new ideas. The new team members were able to draw on other experiences and education to enable them to interpret Tyler and Jones's views about the new practices. The software process models as defined acted as a framework for learning, a reference model that initiated a change of approach.

The process of changing the software practice is seen to emerge through an ongoing dialectic, or interaction, between the product development and the process improvement. The dialectic can be interpreted as how the actions in each area influenced and contrasted with each other. The establishment of new procedures, the inclusion of new techniques for software design and development, and the use of new development tools each intertwined to form a changing development environment. The emergence of this environment occurred not only through the intended documentation of a new procedure or standard for the quality manual, but also through the interpretation of these defined approaches in practice. As developers implemented the system their interpretation of the defined processes evolved. For example, software inspections initially followed the theory-led definition in the

quality manual, but by the end of the period practice had changed this process by removing, altering and adding elements.

The organisational context affected the software processes too through the reaction to the pressure of the competitive environment. The pressure to deliver the product sanctioned the developers' view to dispense with aspects of the defined process. During the development, approaches were considered to be not assisting the developers to produce their design / product, but were seen as a burden by them. The unanticipated outcomes of dispensing with parts of the defined process is evident in the lack of modularity, in the problems in implementing the abstraction in the tiered architecture, and in the product defects. So despite an initial willingness to adopt new ideas, this experience led some developers to revert, partially, to their previous approaches with which they were confident. The remnant of the SPECTRUM team drew on their shared knowledge of the previous approaches, with their norms acting to legitimate this reversion. The core team, however, pursued the defined process as the knowledge of their team leader contrasted with this team norm. So whilst the process as defined had changed, the processes-in-use varied by developers and through time.

The software product and processes were also shaped by the process capability of the team. Process capability is understood not just as the defined process but the team's experience, knowledge and skills in using the methods, techniques and tools. As shown above, the OO capabilities within the IS function varied. The new ideas were understood to different degrees depending on these prior skills. The software practice reflected this capability: techniques were not used as intended, and for many

developers unfamiliarity with the techniques and tools restricted their use to novice level. The outcome of these capabilities was the seen in the vagaries of the MAP product. However, during the development of the product the developers' knowledge of object oriented techniques grew through observation, training, and their own learning-in-action.

The context shaped, and was shaped by, these experiences. As discussed, the historical context formed the backdrop for the introduction of the new ideas, and so traditions acted to sanction previous approaches, but also the experiences of others such as the new project manager and technical architect, contrasted with this sufficiently to retain the principles of the new processes and the overall intent to move towards these processes.

In contrast to the SPECTRUM4 development, the new project manager and the technical architect changed the software processes through their own capability, and related social structures. Much of the improved control was attributed to the new project manager. One example of this was the way he reduced the reactive nature of the team using his own position of dominance to protect the team by filtering the demands from the directors. His political authority made the team's job easier. The developers saw the benefit and began to trust him as a manager, thus improving their willingness to work with him to change the software processes.

Whilst Jones, the technical architect, did not have the language to draw on the same structures of signification as Tyler, he too was able to use his power to make changes to the processes. Initially, Jones was able to legitimate the new processes appealing to

the successes in his previous division. His knowledge of software engineering, which was constantly updated by networking across and outside the organisation, was drawn on to sanction his views. However, he was considered a 'double-edged sword'[44]: using his knowledge he heavily influenced the choices of design approach and programming language, but he was unpopular because of his negative attitude to others. So, his ability to draw on the software engineering features as frameworks of learning for himself did not transfer to his ability to communicate that understanding to others, as he was unable to explain the ideas in terms of shared knowledge. However, he did not let this put him off introducing new ideas. His self-belief and dogged nature meant he continued to insist that his views were right and appropriate, legitimating them through his external sources.

Despite their apparent change of perspective, the management continued to react to perceived market pressures at the expense of the quality of the software. The management sanctioned their decisions by highlighting the actions of Galaxy Systems, and reinforced the importance of the publicised deadline through a bonus scheme. So, whilst the espoused theory had changed through the creation of the quality manual, the organisational norms meant that many habits were retained, especially when under pressure to deliver a new product. This position was sanctioned through the division's norm of releasing software with residual defects.

Accordingly, during this period the software processes were enacted through a constant process of negotiation between the developers, the technical architect, and the software management. The different competencies, characteristics and experiences

---

[44] Interview 18

of the software team shaped their actions. The combination of the organisation's experience with software, the lack of experience of process-based development, and the commercial pressure to respond to Galaxy Systems shaped the attempt to change the development approach. Despite the efforts to implement a defined process there was a lack of appreciation by some in the software team and the senior management of the long term benefits a repeatable process could bring. Nevertheless, a new defined life cycle based on an object oriented approach had been introduced during this period. Other software engineering approaches had been incorporated both formally via the quality manual and through the needs of the development. These software engineering techniques did improve the approach, but the enacted processes did not always follow the defined version, reducing the anticipated improvement. Whilst the outcomes of these actions was not always as anticipated the process-in-use had changed and the organisational understanding of the espoused process had shifted through the ongoing organisational inquiry, negotiated practice, and shared learning.

Through this ongoing, changing software practice, the structures and context emerged. The emergence of the organisational aspects of the case highlights the duality of the change, reflecting Structuration Theory. The actions reinforced, or altered, the context at all levels: the documents and techniques were altered; the individuals' capability and knowledge changed through their shared experience; the software product was an outcome of the process-in-use. A broader consequence of organisations like InfoServ developing Windows based software and using OO techniques was their continued growth.

## 5.4 Case History Period 3: Software Process Improvement

### 5.4.1. Context

The historical context of the software function in 1997 had been shaped by the outcomes of the intentional and adaptive actions within the previous development. During this period the influential factors were similar to previous periods ranging from changes in the professional environment and the commercial context, to the organisational factors within the IS function and in its relationships with other areas of the company.

The significant factors arising in the professional environment were the changing approaches to sourcing and delivery of IS provision, and the continuing changes to the PC technology and applications. The tendency towards outsourcing software development in the industry grew during the 1990s to £1.7 billion (Willcocks et al, 1997). The GeoMarketing division's use of external sources also expanded during this period. They were primarily acting to solve short term resource constraints or a perceived need for expertise, rather than outsourcing the whole IS function. The vendors used can be categorised as: specialist organisations supplying software developed in their area of expertise, suppliers of underlying technologies, and organisations (or individuals) contracted to develop a specified application. The underlying technology was often industry standard development tools, but others were involved in new database or mapping technology. So in some cases software was brought in as a sub-component of the main package, in others it provided enhanced development capability, and for some whole applications were created by the supplier.

The continued development of PC and client-server systems meant that these became a standard architecture for many organisations. The Windows operating system

became 32 bit, thus changing some of the programming techniques required in MAP. Also, during this period the use of the Internet escalated, with the side-affect of web-based systems being established for a variety of business uses. The development tools that were created reflected these technological changes, such as the establishment of new languages, the creation of multimedia and web development tools. Whilst these languages and tools did not change the MAP product directly, the growth of the portfolio reflected these opportunities.

The delay in moving to the Windows-based product had challenged InfoServ's dominant market position. The resultant commercial pressure remained evident throughout the period until MAP version 2 was released. Even during this latter period of the study, Galaxy was able to continue to offer new product developments before InfoServ. One consequence of this pressure was that the software team perceived that the management were always taking a short-term view because of their reaction to clients' demands and the perceived demands of the market.

The senior management team operated in a relatively informal manner, which made them approachable and close to what was going on in the company, but they often 'set off on projects without [the] full agreement [of those involved]'[45] and so the 'agenda moves and changes depending on what Richards is thinking'[46]. The management allowed market forces to influence their decisions about software products, thus changing the priorities in the software development.

---

[45] Interview 3
[46] Interview 3

Within the IS function the management team changed to reflect the expansion in the portfolio. The software management team acted as a relatively cohesive unit in its approach to developing the products and improving the processes. The technical architect held differing views to the others on aspects of software engineering process. Within the management group these differences were cordially debated and consensus was reached on most points. The differences helped to hone the views and shape the understanding of the other managers. These differences were more striking within the team itself, generating ongoing conflict. As highlighted in Chapter 4, the package development team could be seen as having three communities-of-practice: the MAP developers, the data mining team and the disciples of the technical architect, with the testing team as a further sub-culture. The technical architect's disciples were ardent followers of his views, where others tended to challenge his views. There were blurred edges to these groupings that changed over time, so they are not seen as fixed structures, but the relationships did form congruent norms.

The context in terms of the product strategy, the events external to the organisation, the capability of, and relations within, the team all informed and shaped the action of the team in improving the software processes and developing the products. The following section shows how the process improvement activity was influenced by these circumstances and events.

### 5.4.2 Emergence of Software Processes and Products

The conceptual themes of emergence during the previous period remained evident in this period. The response to the previous experience and the dialectic between the process improvement and product development activities will be discussed to show how they continue. Here though, partly because of the opportunity to collect

additional data through observation, two additional concepts were found and will be explored further: the way that the norms within the communities-of-practice and the motives of individuals shaped the emergence.

The previous period illuminated the difficulties in trying to change the actual software processes through implementing a new defined process. The tendency to revert to tactics and habits previously employed, whether successful or not, was evident. The actions of the management in response to the commercial pressure had legitimated the developers' decisions to take short cuts. However, the robustness of the core sub-system changed that rhetoric and the support for the adoption of a repeatable process grew. Given the problems with deadlines, different teams each going their own way in the development process and high levels of defects retained in the Market Analysis Package system after release, the software management felt that it was important to continue to refine the processes for developing and managing the development.

The professional norms that valued structured approaches as important in developing quality software on time were used to legitimate the continued work on process improvement. After the initial attempts in 1995 to document the process, the software director wanted to continue to change the norms away from the ad hoc development approach. Tyler had an affinity with a more structured approach and was able to draw on the norms of the wider software environment to suggest that this was the way to overcome the problems they were facing. The success of the core sub-system had increased the dominance of this view. Drawing on his experience in MAP version 1, Jones reinforced the structures of signification by updating the defined process to reflect the practice, or the perceived practice.

During the SPI project the link between the process-product continued to be influential in shaping the emergent nature of the process. A good example of the development process–product dynamic was in the area of software design approaches. In particular, the attempted move to component based development was directly linked to the organisation's need to produce bespoke products and a portfolio of packages profitably. The intent in MAP version 1 was to produce a modular package but the software director explained that any possibility of achieving this desire was dismissed early on 'because the company was so far out, adrift from the marketplace where it needed to be...design to that level became a luxury that we couldn't afford'.[47] This approach resulted in the duplication of software between applications. With the development of the product portfolio, however, it was not long before the rising maintenance cost became apparent and component based development became significant.

The component concept came from the external software engineering domain, but its relevance became evident through an improvised experiment to show how parts of the MAP product might be reused. This experiment was sufficient to change the structures of signification, as people from across the division began discussing the idea of component development. However, the changes in the structures of signification were insufficient to change the practice in the long term. The organisational norms for reacting to the market meant that this predicted change to the development approach was never implemented. The dominant position remained fixed on producing specific applications for market demands because the resources

---

[47] Review 14

were never made available for this initiative despite the perceived benefits. For the foreseeable future the benefits were considered to be intangible and tenuous.

Reluctance in the team to adopt components also constrained Tyler's power to complete this innovation. Their reluctance can be attributed to a lack of belief that the approach would succeed because they had little knowledge of it, thus preferring to retain their existing approaches. Jones tried to persuade them of the value of the approach. He had been able to persuade Tyler and other senior directors about the approach, but the previous conflict between him and other team members reduced their trust for his suggestions. However, it was not only the developers whose traditional belief systems were drawn on to sanction their current approach. Bridges too stated that the current approach was 'not hurting us in business terms'.[48] So the message he communicated to the team sanctioned the retention of the existing approaches rather than supporting the emergence of the new structures of signification about components and object design.

Component based development is an example of a process initiative that in the end was not implemented. The process-product dynamic, however, was also evident in a number of successful innovations to the software processes. Most innovations arose from needs identified during the development, as individuals reflexively monitored their own action. It was therefore through this continual, situated learning that the practice changed. Whilst some of the new ideas were drawn from outside of the organisation they were only brought into the development practice when the idea complemented the perceived need, such as the introduction of the Critical Chain as a

---

[48] Journal

way to reduce project overruns. The champion of the idea would legitimate it through a personal success or external norms. In each case the communication drew on language that others could relate to. Conversely, when structures of signification had not been shared, say in the case of external knowledge, then the idea's relevance was more difficult to communicate, as shown in the component initiative. A common pattern with such changes was that they were considered relevant and valuable by the person championing the introduction. When someone saw a clear purpose in introducing a new technique, or revising a current method, they were prepared to apply their own resource and the team's resources to its introduction (either directly through delegation or by winning others round through negotiation). By recognising the relevance of changes in their approach the practiced process was then recreated as the norm.

Norms, i.e. the rules and habits we apply, relate to individuals and shape their action. Individual developers followed particular approaches that related to their own experiences and knowledge-base. There was, though, a level of commonality within the communities-of-practice. The individual competence, experience and personalities tended to encourage the formation of communities-of-practice that did not simply reflect the team infrastructure (see §4.5), but grouped people with similar beliefs. Each community-of-practice shared their approach to the development and therefore the norms within the group were similar and emerged relatively independently of the others.

The data mining development was an example of how the communities-of-practice differed. Members of all three communities-of-practice worked on the product at

different times. The data mining group implemented the defined process but were willing to make compromises; the technical architect followed new, advanced programming techniques that sometimes confused the others because of their complexity; other developers, temporarily allocated from the MAP development, followed the approach they were used to in the MAP enhancements for version 2. The MAP sub-culture often delayed what they considered to be non-essential tasks, such as software inspections, until pressed by the team leaders. The successes, or otherwise, of specific groups reinforced their beliefs about particular features of the enacted process. Thus, when an inspection was delayed the value of such validation was reduced because most of the problems had already been discovered in the later stages of development, reinforcing that group's view that the cost of the inspection was too high.

Each subculture, also, was slightly different when it came to putting their own resource into improving the process. The MAP enhancement group, who were seen to be least interested in following the defined processes, were also relatively uncommitted to the SPI activity perceiving it as a management task. However, to show that communities-of-practice are not entirely homogenous units, one member of the MAP group, who had been a member of the original core team, and therefore had successfully used software inspections, continued to champion the use of these and undertook his related SPI action with enthusiasm.

Individual motives were therefore seen to be significant in shaping the activity, but these motives did not always equate to the stated intent. The new software director stated that the purpose of formalising the SPI initiative was 'to concentrate on

continuous improvement. [I want] people to focus on things which will enable me to get better and better.'[49] This stated reason was perhaps sufficient in itself to legitimate the change, but only shows part of the underlying motive, which can be identified through the director's actions and statements. As partly indicated by the egocentric nature of the above statement, the software director's motives were related to personal desires: he wanted to strengthen his group's position in the organisation. This motive was in direct reaction to the continued questioning of the other sections of the division, and senior directors, about the ability of the team to produce good quality software. To change the structures of domination, the software director needed to react to this developing political situation.

By encapsulating the problems into an SPI project the software management were able contain the problem. This action gave them credence and began to redress their political situation. Thus, with the problems appearing to be addressed through the SPI project, the divisional directors' focus switched away from the previous problems to the actions of the competitors, thereby fulfilling the software management's motive for undertaking the project. By July 1999, after MAP version 2 was released, this underlying motive had been fulfilled, allowing the software director to state that it was 'time to put [the SPI project] to bed'.[50]

Also, the divisional management were often perceived by the team to not be committed to the SPI project. This developers' perception reflected the management's actions that removed people from process improvement activity, by-passed the defined processes, and worked against the long-term development of the team's

---

[49] Interview 1

capability by outsourcing the development. These actions drew on the norm of needing to get products to market quickly to legitimate the short-term action of applying organisational resource in this way. This desire runs counter to the Software Engineering Institute's view of how to achieve quality products through a mature process, but was sanctioned through the profit-based norms of the package software market. This contrast is important here because it was this underlying motive that, unwittingly, in some cases gave rise to actions that worked against the attempts to improve the process. In doing so it reinforced the view that the SPI project, and following of the defined processes in general, was a relatively low priority. This is not to say that the managers did not want to improve the software process, just that it was not their primary motive. The process improvement enabled the organisation to re-establish its strong market position. From this position they were able to begin to move towards a whole new product set in web-based systems by making use of their new capability.

## 5.5 Reflections

### 5.5.1 Emergence in Software Process Improvement

This section will look beyond the chronological view by drawing together features identified within the process improvement activity. It will be argued that the emergence can be understood as situated change, as a learning activity and as a political process. Existing theory will be used to assist in understanding this emergence, and to generalise the concepts and insights so that they are valuable in contexts other than that analysed here, thereby identifying the issues for SPI practice.

---

[50] Review 25

The first conceptual theme discussed below is that of situated change. The discussion will critically assess how the framework was supported by the case analysis and how the analysis was supported by the framework. The discussion of the case argues that the changes in the processes occurred as an emergent, sensemaking activity. Actions in the latter stages are often seen to be influenced by the earlier events and the traditions that have developed in the organisation, with actions legitimated by drawing on the traditions and norms of the sub-cultures within the organisation. Stocks of knowledge, from shared experiences, facilitated the communication of the changes.

Building on the literature outlined in Chapter 2, the second theme of emergence is the aspects of learning in SPI. It can be argued that the improvement at InfoServ was primarily a process of learning through experience, as the mental models matured. Reflection by developers during their development tasks informed their process improvement. The initiatives were not necessarily (indeed rarely) optimised solutions or industry best practice, but simply a solution that appeared to address the need. The changes were derived from innovations they felt to be directly relevant to their work, and would address problems in the software or the life cycle. Ideas that were tried out as experiments to address a particular need during a development activity were communicated to other developers through interpretive schemes that reflected a shared experience of the MAP development, so sharing stocks of knowledge, principally within their own sub-culture.

These changes are seen to be shaped by the power individuals used to apply, or resist the use of, resources. So the final area for discussion, for which there is a paucity of existing work in SPI, is the politics of improving. The individuals associated with the

change were able to either utilise their power to insist that the project team introduce the idea or draw on early successes with the idea to legitimate the change. The use of power was not just by managers, as the team were able to use the SPI project as a structure of dominance to draw on. Alternatively, developers used their power to resist a change by withholding their effort from introducing the idea. Specific instances of innovations that were not introduced successfully were related to such resistance. The reasons for the resistance varied but related to the perceived relevance or usefulness of the change. Yet where people had perceived an advantage in making a change they had mobilised their own resources and that of others to bring about change.

### 5.5.2 Emergence: the metamorphosis of the process

The purpose of this section is to reflect on the usefulness of the framework outlined at the beginning of the chapter, and in particular look at the emergence of the software processes as a means of sensemaking. Throughout the study, the processes were seen to change through an on-going dynamic with the product development. The case data shows that without running the SPI project according to the theory, and without even following their own plan, a number of successful changes were made that over time were perceived to make a significant impact on the capability of the group. The resultant products reflected this improvement. The development of the products was influenced by the process capability of the group, a consequence of their actions from the previous software development. The product strategy shaped the process improvement by heightening the attention of the team on certain aspects of their work and influencing the processes as they were enacted. Kautz and Nielsen (2004) recently arrived at similar conclusions showing that context shaped the social processes as they

unfolded, but their work assumes a stronger structuralist position and so does not highlight the emergence of the context through the actors' actions and decisions.

The analysis here shows that software processes emerged as they were enacted, actions were shaped by the context but also the context was recreated through the actions. New methods were introduced, or existing approaches revised, at InfoServ on a planned, improvised and adaptive basis. The practiced processes emerge from the attempts to apply and refine new techniques within the context of the software development. Changes were therefore realised through the actions of developers as they developed the products, and ideas for changes originated through reflection-in-action. The definition of new process models and the introduction of methods from external sources acted as a form of intended design, but the actuality of the change was seen through the implementation of these designs in practice. So whilst it may appear that the change was as intended, it consisted of practices that emerged gradually.

The individual's actions and the formation of their understanding was a sensemaking act, in that they constructed an interpretation through an active process of invention. Sensemaking was ongoing. The application of project planning techniques, for instance, emerged from a mixture of an understanding of historical practice and associated problems, the introduction of ideas from external sources, and the ongoing interpretation of those ideas to suit the needs of the work of the MAP team. The concepts were communicated through a mixture of documentation and sharing of experience. Individuals suggested methods of working and disseminated these to their colleagues. Ideas though were not disseminated *en bloc* as each planner moulded the

approach to suit their individual project, experience and perception. Reciprocal adjustments were made to the practice. Actors made sense of their action by imposing their own worldview, interpreting the application of specific methods according to their perspective. Once established the process-in-use became the norm and in time was formalised through documentation or training sessions.

The analytical model in §5.1 has provided a useful lens through which the process of change has been reviewed as occurring through a structuring process. The way in which the changes occurred was explained using aspects of Structuration Theory to show the linkage between the changes to the processes, products and their emergent context. The emergence was shown to not be simply a random process, but something that occurs to achieve an intended vision where the detail of that designed future is not fully understood at the time of the action. This view is supported by Weick (1995) who argues that sustained action comes from commitment, which focuses sensemaking. Thus, similar to Truex et al's (2000) view of information systems emerging in their own particular way, software processes emerge to suit the local needs, experiences and situated context.

The emergence of the context was seen to occur, as well as the process-in-use. Within the IS function, people's knowledge and capability developed through the experiences, and the formal infrastructure and dynamics of the communities-of-practice changed through deliberate and unanticipated changes. The process changes affected the organisational and external context and structures through the product strategy and the dynamics of the commercial environment. This emergent form of organisation matches that presented by Truex et al (1999), who suggest that the

socially constructed nature of the organisation forms the basis for the next version of the organisation: as it changes it does so with reference to its former self in a process of recreation.

However, the model could have been clearer about the roles of individuals in the changes. Like in Klein and Nielsen's (2004) study, here it was evident that certain individuals, or roles, were important in shaping the process improvement. This was implicitly accounted for in the framework as a feature of the social structures or context, but further analysis of the role of individuals might have enhanced the theoretical development. This aspect could have been illuminated through the use of an alternative underpinning social theory, such as Foucault's views of power and knowledge or Habermas's analysis of communicative action, or by drawing on further reference disciplines, such as social psychology. It is proposed that these options are subjects for future work. In addition, whilst the model identifies learning and political aspects through its structurational perspective, these are relatively underplayed. The analysis showed these to be important features of the emergence and so are expanded below.

### 5.5.3 Emergence: a Learning Perspective

A recurring feature of sensemaking is the reflexive nature of the individual actor. The analysis of the case illustrates that the learning occurs through reflection-in-action during the enactment of the software process. Individuals learn, share lessons and draw on this shared knowledge to undertake the process of developing products. Figure 15 identifies separate elements of the learning activity to be discussed. This is not to imply that these elements can be separated other than for analytical purposes,

rather that learning and practice occur together, synergistically informing each other. The model reflects the dualistic view of action and context, where the individual draws on the organisational context to undertake their practice and through that practice changes the context. The discussion highlights the linkage between the two facets.

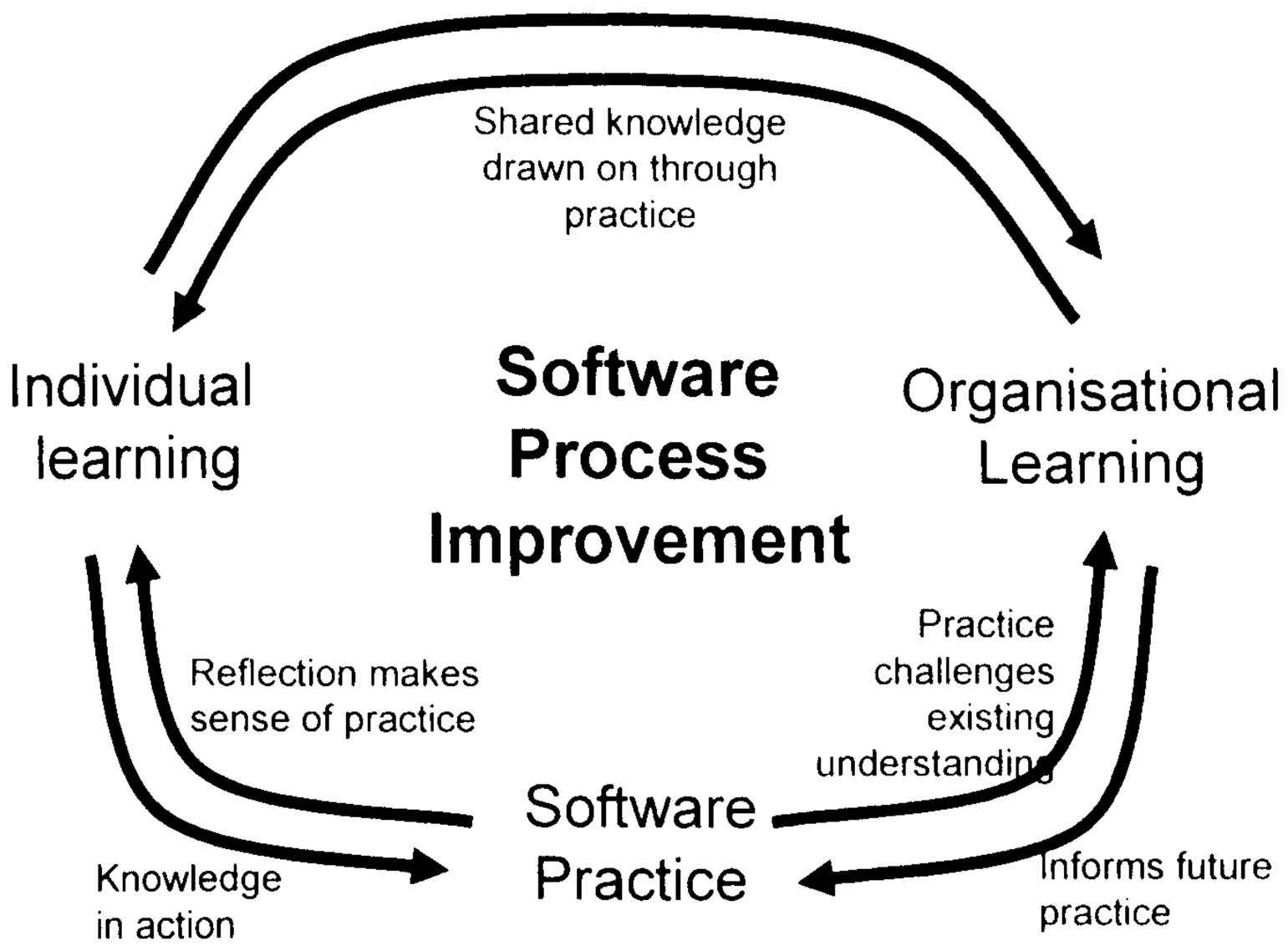



**Figure 15 SPI as learning**

As discussed in Chapter 2, sensemaking is considered to be more than a cognitive process, but rather it is embodied in our physical action: actors improve their software process which helps to improve the product development, and so continue to refine their process. Actors therefore learn through action, by practice and habituation, bringing prior experience and conceptual knowledge from other sources to support their action. This learning is akin to the model of learning through use developed by Daft and Weick (1984), and Kolb's (1984) experiential learning model.

Learning to change is often taken to be a transfer of information from those who know to those who do not. This 'reifies knowledge and de-emphasizes its collective nature as well as [the] social processes of knowing' (Swan and Newell, 2000, p.592). This view implies that prior knowledge is the only thing that is important, which tends to reinforce what is already done in the organisation. MacDonald (1998, p.40) argues, on the contrary, that this knowledge sharing is 'a process which cannot be directed and controlled'. It has been shown at InfoServ, that their local adoption was influenced by the context, the desire of the developer, and the perceived relevance of the innovation for the product development. The assimilation and application of ideas in the case, whether from internal or external sources, were seen to be related to the perceived value of the method or idea. The value related to their level of trust of the sources as well as the perceived usefulness or applicability of the technique.

At InfoServ, product and process innovation occurred through individual experimentation. One example was the creation of a component from the MAP software that encouraged the software management to explore the adoption of component-based development. More specifically in the area of process innovation the adoption of both the Critical Chain planning and test estimation approaches followed external training. In each case these ideas were assimilated through an individual who was able to identify how these approaches would help to resolve existing issues that they were facing. The techniques were introduced through improvisation and experimentation: each idea was reinvented for the needs of the team, debated with other members of their community-of-practice before exposing it to members of the wider team, and then evaluated in given instances to test its usefulness. The willingness to put their own resources into trying out new ideas

combined with the creativeness to achieve a useful result were critical personal attributes. The experimentation often allowed the individual to explore an idea without the pressure to succeed: the result could be thrown away if it did not work out. Experimentation can be encouraged, but the instances observed did not result from any organisational scheme but from individual capability and desire. In both the examples above it was the motive to change the current approach that encouraged them to put their own time into the experiment. This type of successful assimilation reflects Ciborra's (1999, p.137) understanding of smart improvisation as an action 'which contributes to the individual or organizational effectiveness' rather than that of a novice or someone just acting extemporaneously which has no effective link with the demands of the situation.

Formal training and development programmes are not required to enable external ideas to be incorporated into an organisation. Merali (2002) shows that individuals who are part of a fluid, external network of professionals will reflect on those interactions and seek to apply ideas appropriate to the perceived needs of their own software processes and products. The danger is when external innovations are introduced en-masse because they have worked elsewhere. Lefebvre et al (1995) show that the level of benefits derived from the adoption of new ideas is dependent upon the level of penetration of those approaches within the organisation, which requires that the solution applied is refined to fit the need on each occasion. This is a form of bricolage, that is it makes use of whatever resources and skills are at hand, and therefore contrasts to the engineering view of change where projects are not started until all resources are identified and available (Weick, 2001).

Bricolage encourages the use of existing tools and routines by people at the operational level to solve new problems. Local cues are used to obtain ad hoc solutions that bubble-up serendipitously from the normal daily activity. Thus, these changes 'emerge from the enactment and reinforcement of local innovation' (Ciborra, 1994, p.16). However, this is not to suggest that the development of the process or product is entirely random. By combining the approaches of *bricoleurs* with those of engineers there are no limits to the possible implementations of software engineering ideas (Dahlbom and Mathiassen, 1995).

Ciborra (1994) finds that only by encouraging improvisation through tinkering, or bricolage, and having a willingness to fail will innovation occur. When new ideas were introduced at InfoServ not everyone fully understood them, not even those who were introducing the idea, but people were willing to experiment with ideas to see if they had value. The changes to the software process therefore continued to emerge as individuals considered how their actions were supporting the development of the software products.

Individuals' mental models matured through the enactment of the process, as highlighted by the technical architect, who recognised that he had a maturing view of design that took into account perceived mistakes as well as successes: 'I know a lot more about designing large, relatively large software projects in C++ than I did before doing this one. There is a whole host of design approaches which I would no longer take (sic.)'.[51] This maturing of understanding was evident across the team. Individual

---

[51] Interview 19

learning is linked to the organisational learning that takes place as a result of the organisation's experience with any innovation (Lefebvre et al. 1995).

Such organisational learning is more than simply documenting new processes but is about improving the knowledge of members of the organisation by sharing experiences. Indeed, Conradi and Fuggetta (2002) found that developers considered formally documenting processes ineffective in transferring knowledge. So whilst, software process models documented in manuals can be changed to reflect current practice, to incorporate new ideas from outside or in an attempt to develop a more *mature* process, such manuals at best only reflect the desired practice of a software development group. Truex et al (2000) suggest that methods are discarded early in the development process and practice is often inconsistent with the defined methodology. So, even if we assume that teams attempt to instantiate the process as defined in a particular project, by looking at the defined processes our understanding of the process is limited to the espoused theory.

Argyris and Schön (1996) show that individuals maintain their own theory-in-use by interpreting the espoused theory and other previous experience. It is the theory-in-use that the individual draws on to respond to the particular problem faced. We therefore cannot assume that the espoused theory is the same as the theory-in-use, and that it is the same as the action within the development activity. However, it is as the theory-in-use changes and becomes the norm that the espoused theory also changes, in the way that the defined processes were rewritten or communicated to new members of the team, reflecting Argyris and Schön's (1996) concept of double loop learning.

Sharing understanding can therefore be supported through process documentation but this has a limited role in communicating understanding. We therefore need to recognise that knowledge emerges from 'patterns of social relations and dynamic practice' (Scarbrough, 1998, p.227). Knowledge can be viewed as an emergent property of organisational interaction with the wider environment, and in terms of social practice and relations. As organisations learn through practice the norms of their communities-of-practice emerge. As practice unfolds it challenges those norms and refines them through individual learning, and then future practice (and related process change) is informed by the norms as individuals draw on them to sanction actions.

These norms can be seen in terms of individuals learning to routinely follow the process. Whilst there was some resistance to following the processes as defined, as the actual processes used became the norm people knew what was expected of them and therefore 'it gets instinctive...[and so they] follow [the] method...despite themselves'.[52] The application of object oriented programming was a good example of this routinisation. As indicated in Chapter 4, the move to a C++ development environment was difficult because knowledge and skills in this area were in short supply. The level of prior understanding varied and therefore some developers found the abstraction and encapsulation discipline difficult, sometimes returning to old habits. However, through repeated use, listening to others in design meetings and subsequent inspection meetings, and both formal and informal training their personal understanding developed, albeit to a varying extent. The improvement was evident to those looking back at software developed at different times.

---

[52] Interview 16

So as ideas are shared across a group, tacitly and explicitly, the degree of *systemness* increases within the communities-of-practice through shared meaning. Giddens (1999) defines social systems as the reproduced relations between actors or collectives, organised as regular social practices. These collectives are not unified, but draw on commonly understood rules to communicate and act. So there is a greater ability to draw on similar rules as ideas are shared across communities-of-practice. This shared knowledge resides with individuals, but the team's capability increases through an improved ability to access knowledge held within the community and a greater understanding of its relevance to their needs. As this capability improves, so the team's agility to respond to development triggers improves. As norms are drawn on within the practice, then further experience will challenge the understanding embedded in those norms – at the individual level but through discussion and negotiation the norms will evolve – and those norms will be used to inform participants' future practice. The case showed that sharing ideas to develop group solutions is difficult, as individual motives tend to engender a personal perspective. Trust between the participants enables this sharing and learning to happen. These aspects of the improvement process are discussed in more detail below.

### 5.5.4 Emergence: a Political Perspective

Hosking and Anderson (1992) highlight that a key challenge in the area of organisational theory is to understand the political issues in organisational change. In particular, they consider that politics 'as processes, as texture intrinsic to organization and change – are entirely ignored' (ibid, p.8), with authors tending to assume a view that looks at how structural power drives change. Similarly, Knights and Murray (1994) consider that the literature tends to assume either that politics is a deviant that

disrupts the smooth running or it is an uncomplicated reflection of class or gender. There is a need, therefore, to adopt a view of polities as central to contemporary organisations, where the character of negotiated order is examined. Figure 16 shows the emergence of process improvement as ongoing through a series of political actions as enacted in the midst of the product development. Again, the elements of this diagram are representing analytical features rather than attempting to segregate the process of change. The remainder of the section will discuss how this model represents the findings from InfoServ. The model mirrors the relationship in the original framework between the software practice, the context of the community-of-practice and the linkage through the behaviour of the individual. The structurational perspective helps to stress this linkage.

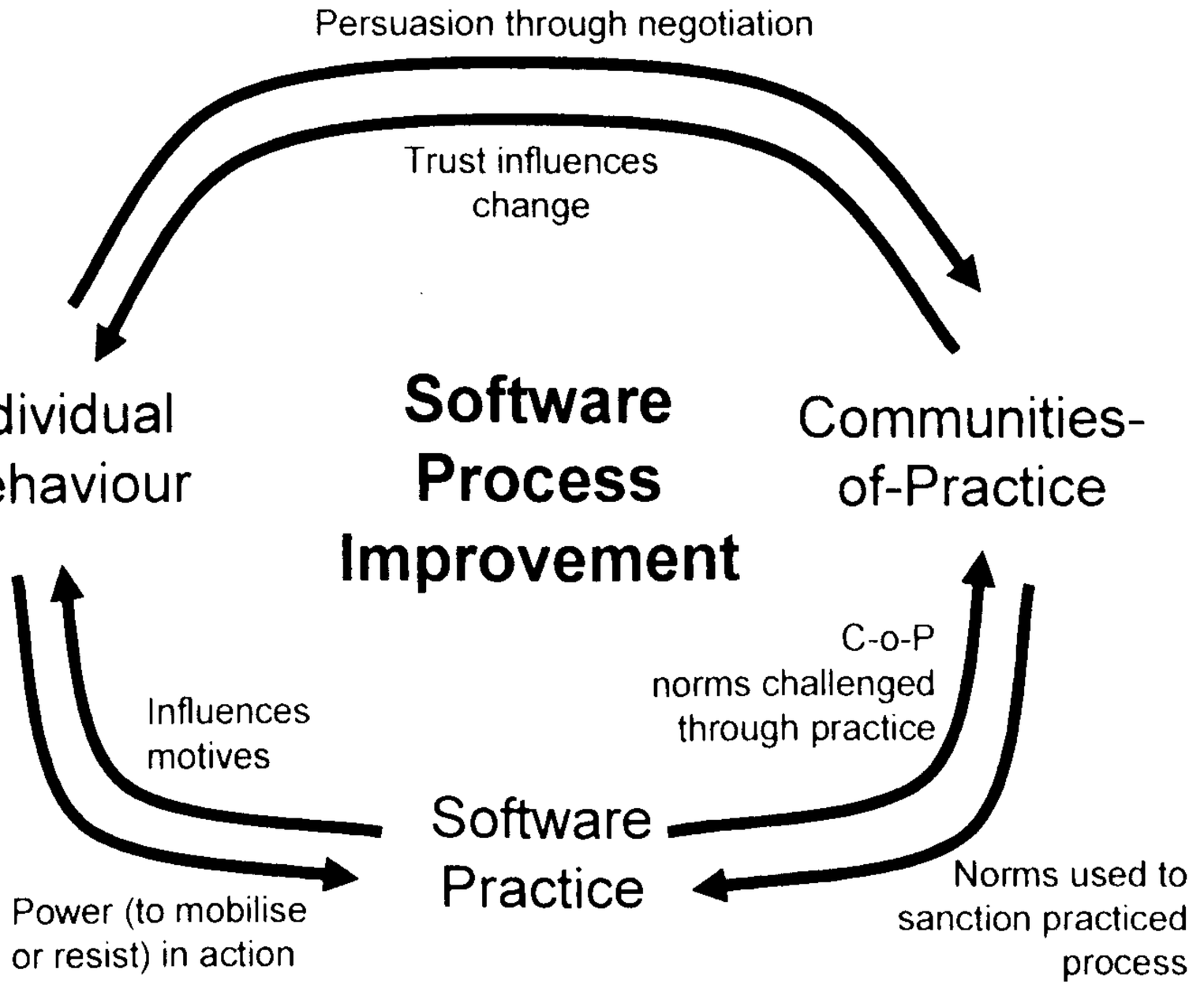



**Figure 16 SPI: a political perspective**

The case history suggested that an element of individual sensemaking incorporates (consciously or unconsciously) the political nature of the act. Actions are taken to further their own position or raise the status of their own area. Examples of the self-

seeking nature of actions include: promoting their own perspective (on the process or product design, say); protecting the resources in their function; enhancing their own career development. Whilst generally he has little to say on political aspects of sensemaking, Weick (1995) does highlight that there is a link between beliefs and actions. March and Olsen (1979) are more specific in that they state that any decision making activity stems from self interest, which in turn is connected to beliefs and attitudes. Coopey et al (1998) argue that this behaviour can be understood by recognising that individuals transform the social structures of their context by:

> repositioning themselves discursively in attempts to make sense of some disruption to everyday routines or, spontaneously, to promote some innovative idea. Engaged as they are in the flux of organisational activity, those individuals are able to tap into its 'flows of social and psychological information about possible ways of life' (ibid, p.272).

The overt purposes for initiating the SPI project at InfoServ were shown to be different to the underlying motives. The software management explicitly stated that the reason for the process improvement activity was *in order to* enhance their capability for delivering projects on time, but the deeper motive was *because of* the pressure from senior management and related career aspirations. Hosking and Anderson (1992) argue that people mobilise power in order to further their own interests. In the process improvement tasks, actors were seen to utilise personal and organisational resources to bring about a change that supported their perspective. One example of this was the project leader's application of the broader features of Critical Chain planning (i.e. Theory of Constraints) to planning across the whole software

area. This initiative identified the testing team as the constraint, who were frequently switching tasks away from his team's testing to the bespoke projects, as these projects were often more urgent, consequently delaying the MAP development. By getting the support of the QA manager and other members of the software management team, Bridges was able to bring about a change in the planning process. During the negotiation the innovation was presented as an improvement for all, but the underlying purpose was to overcome the problems his team encountered when delays in testing occurred.

One of the major factors influencing the uptake of the SPI initiative was the resistance by some actors to partaking in the activity. The nature and cause of resistance therefore needs to be placed within both the specific organisational context and compared and contrasted with the concept of resistance as part of Structuration Theory and the wider organisational literature.

Within the ongoing software practice individuals drew on the structures of domination to follow or avoid the processes as defined (and as desired by other members of the team). Two examples involving the product director and a member of the data mining team reflected how this happened across the organisational hierarchy. In a risk planning meeting the product director overrode the analysis that suggested that the third party supplier would be a major risk. The lead developer frequently referred back to this point when the problems with the supplier arose highlighting it as a critical error of judgment, undermining her trust in him. Rather more subversively, the same developer managed the director's pressure for progress by holding two versions

of the project plan (one for her own project management and one to be shown to Herriot).

Not all resistant action was hierarchical in nature. Ongoing software practice was a constant process of negotiation and conflict with different developers holding views about how to undertake the work, such as the inconsistent use of object oriented design and programming. These variations in practice transpire because realities are co-constructed by people with different relationships to their context: who work with different people, have different histories, perform different tasks, and so on (Hosking and Anderson, 1992). So individuals differ in what they see as important, power is relational and action based rather than structural, and norms are constantly changed through this ongoing narrative.

Resistance to change has also previously been identified by other SPI researchers as a factor in SPI projects. Resistance, or the unwillingness to deploy personal and organisational resources to achieve a goal identified by someone else, is a feature of power relations. In the above analysis individuals were seen to avoid undertaking improvement tasks, legitimising this resistance by arguing that SPI was the role of managers or that they were too busy on the product development. Wilson and Hall (1998) suggest one reason for this variety of views within the team is that whilst practitioners share a commitment to achieving quality software they have different perspectives about what it means. They found that managers saw it as needing to implement process-based quality management systems, whereas developers see these as not contributing to the production of quality software.

The findings at InfoServ, however, showed that the views were more mixed than this simple dichotomy. The case indicates that resistance was from a variety of actors depending on their purposes at the time. Jermier et al (1994) concur, suggesting that a dualistic model of control and resistance is to be avoided as all resist and respond to situations of domination: this is not a manager / worker split. This more complex picture of resistance is explained by Collinson (1994, p.49) who suggests that actors 'have a variety of options, knowledges, cultural resources and strategic agencies through which they can and do initiate oppositional practices. Workplace resistance may seek to challenge, disrupt or invert prevailing assumptions, discourses and power relations.'

In one way then, resistance can be seen to be a deliberate attempt to take alternative action (or inaction) to others, particularly when opposing dominant individuals or groups. However, some perceived resistance may be due to a lack of trust between individuals rather than an act of deliberate conflict. The literature suggests that trust is an attitude of mind that influences the interplay between wilful agents. March and Olsen (1979, p.66) propose that a 'participant will come to believe that people he trusts cause events he likes and that people he distrusts cause events he dislikes'. In the analysis of the case, trust is seen as an important element of the formation and reformation of communities-of-practice: trusting relationships provide an integration of existing social systems and are drawn on by those involved in innovation projects (Coopey et al, 1998). Hertzum (2002) particularly highlights the importance of trust in the acceptance of new ideas by software engineers from colleagues; this is one reason that internal sources of ideas are preferred to external sources.

Confidence in the person's behaviour, not just their ideas per se, was therefore important in the way individuals accepted specific suggestions for process improvement, such as changes to the software inspections and the Critical Chain planning. Whilst some people did not value the software inspections, they were willing to attempt to refine the process because the person championing the changes was considered a trusted colleague. However, in the latter example it was noticeable that developers felt vulnerable with the introduction of this new planning approach because the time for the planned task was reduced and placed in a 'buffer'. Depending on their trust for the project manager, and the software management in general, they either willingly undertook a trial of the method to see if it would work or resisted because they were suspicious of the management's motives for reducing their task estimates. Bridges's conciliatory manner tended to overcome most of the concerns, but there was a secondary, residual fear of how Herriot would use the buffer in these plans, implying that trust is more complex than simply the relationship between any two individuals, but relates to the wider team relationships.

As in the above examples, one way that trust was generated was through dialogue about the proposed change. Shared meanings across the community-of-practice helped to unite support for the application of a process or a change in the process. Coopey et al (1998, p.276) state that:

> though friends and other close acquaintances share the goals to which change is directed, they may still need to be persuaded of the wisdom of the action proposed, engaging rhetorically in a form of social dialectic through which arguments can be refined and agreed on as a basis for action.

Negotiation then is an ongoing feature of the organisational narrative, with persuasion and debate being constant elements of this conflict. In this sense, the conflict encouraged the emergent change rather than being a negative source of tension. Conflict encouraged reflective evaluation of both self and others, and so challenged existing conventional wisdom and theories in use that acted as norms. Only rarely did the conflict become overt.

Another aspect of perceived resistance was differing perspectives on what is deemed to be appropriate action. Collinson (1984, p.28) states that knowledge is a 'crucial resource and means through which resistance can be mobilised'. Knowledge is contested and shifting through time, so he argues it is not simply the possession of knowledge that determines consent or resistance but the way this knowledge is deployed in particular organisational conditions. The various actors across the software team held differing views of quality. Even within the programming style some would consider that a high level of abstraction was good object oriented programming and therefore helpful for maintenance; others considered this style inaccessible to other programmers and therefore more difficult for maintenance. In terms of the SPI project particularly, individuals who avoided the official tasks would at the same time be ready and willing to make suggestions to improve their own practice. This contrast indicates that conformance and resistance are not simple dichotomies, but are related to a complex sensemaking activity that weighs up the value of the action based on their stocks of knowledge and experience.

Rhetorical persuasion thereby influences the behaviour of others by recreating the structures of domination in the organisation. Symbolic labels are used within the

rhetoric to persuade others. Fincham (2002) found that groups seeking to employ new approaches may purposefully label the past as a failure to construct future successes. Similarly, here the emphasis on previous failure was used to endorse the changes, but this emphasis was made initially by others across the business and drawn on by the software management. So, as well as using labels to persuade the team of the change, the SPI project itself became a label to persuade other managers of their intent to improve. The rhetoric used to promote the SPI project suggests that it was a way of dispelling previous concern: making it simply a palliative rather than a fundamental change of culture. The creation of the SPI project at InfoServ, therefore, could be understood as a symbolic action, that combines an expression of individual desire with social action.

The reason for acting goes beyond the immediate practical reason, as it incorporates a more inclusive notion of the deeper psychological condition (Ricoeur, 1991). Huff and Schwenk (1990) suggest that managers give themselves credit when things are going well and seek to blame others for bad times. One consequence of this is that changes can result from bad times as management doubt the current approach. Problems with the product at InfoServ twice raised doubts about the software development process, and infrastructural changes were instigated. Once the motivation of addressing the bad times reduced there was less concern, enabling the software director to draw the SPI project to an end.

Symbolic labels were also used through the ongoing software practice to justify changes and persuade others of the value of the change. One example of how labels were used (or one might say misused) to persuade those outside the team that action

was being taken to improve the quality of the delivery surrounds the introduction of the 'beta testing' process. Through the period at the end of 1998 and beginning of 1999 the term 'beta' was often used by a variety of people to indicate that an early release of software was to be made but different people held contrasting views of what this meant. One view of the new process was the standard software engineering view of beta testing that is an attempt to improve the full customer release by incorporating a hand picked group of customers to trial the system following internal testing. However, it also came to mean a way of justifying a release that was not fully working, thereby appearing to meet deadlines. In a team leaders' meeting about the data mining tool, the discussion suggested releasing an 'alpha' because the system was not internally tested and it would be good to release 'something'. The effect of discussions such as these was that the various members of the management team claimed that the others had abused the term by releasing poor 'beta' versions and so when the process was formalised a new term 'feedback release' was coined to regain its legitimacy.

Throughout improvement programmes an organisation is managing not only technical change but also behavioural change (Curtis and Paulk, 1993; Ravichandran and Rai, 2000a). Managing the social aspects of the change process is often the most difficult because established patterns may need to be modified. The action of individuals is shown to be governed by norms. At InfoServ, one of the developers recognised that implementing process improvement was more than introducing a new process because it was 'very difficult to change someone's working habit'.[53] To change the culture within a software team the norms of the individual developers need to change through

[53] Interview 9

their interaction with others in the varying communities-of-practice that they operate in. Norms are more than simply history: they are the habits and routines embedded into the organisational knowledge base of the communities-of-practice. In the continuous improvement of the paperless integrated manufacturing system at Motorola, Dawson (1994) found that it was harder to change the thoughts and habits than to change technology. Cultural readiness is therefore seen to be an important consideration before the introduction of an innovation (Thompson and McParland, 1993). The case showed how the norms of different communities-of-practice influenced their practice. It is only through new experiences that these norms began to be reformed. Through these shared experiences the individual's use of power may invoke 'consent to everyday practices' (Knights and Murray, 1994, p.36).

### 5.5.5 Summary of Issues for SPI

In line with MacDonald (1998), it has been shown that to look back at the SPI project as a simple planned change would tend to give an overly neat, ordered view. Such views of software activity imply a picture of fake rationality (Truex et al, 2000). The analysis of this SPI project has therefore considered the improvement as an emergent process of change. The process of improvement has been intrinsically linked to the situated practice of developing the software products, with changes perceived to be relevant to the needs of the team. The software process improvement was considered part of a complex intertwining of multiple changes, enabled and constrained through a dynamic relationship between agents. The improvement was understood as a negotiated process of change, occurring through structuring process. The changes to the software processes were shown to emerge through the reflexive nature of the software developers, shaped by the context and traditions of the organisation. The enacted processes form the norms for future practice and recreate the context for

future developments. The complexity of the relationships, the learning intensive nature of the change and the political motives that shaped the behaviour of the actors have been shown as important facets of the process of change. Overall, the reasons for undertaking the SPI project were more complex than the initial rhetoric would suggest. The following chapter will identify the lessons and implications from this analysis for IS practice and research.

# Chapter 6 Implications for Practice

## 6.1 Introduction

The findings from this substantive case study have been discussed during the previous two chapters. These findings have been generalised through the theoretical development, supported by existing social and organisational theory. The purpose of this chapter is to look beyond this theoretical analysis of the case and to consider the impact of the research for other organisations. These views relate primarily to package software organisations, where the product is perceived to be paramount. Other types of organisations, such as service organisations or internal IS functions, can tailor the ideas to their requirements.

Initially, this chapter draws together the implications direct from the case study. It is not the intention to repeat the previous chapters, but simply to summarise key lessons that may inform practice in other organisations. The chapter then discusses the potential future implications for SPI practice by considering how SPI can be more agile. These proposals have not been tested and are limited by the scope of this single case study, and are therefore subject to further research and evaluation. The concept of an agile approach is supported by use of the existing literature.

## 6.2 Implications from the Study

Within the SPI activity at InfoServ it was recognised that the changes were not all based on pre-planned solutions. Indeed, the maturity models played no part in the definition of the process changes. Planned actions were identified according to perceived needs, but often these were changed or abandoned in favour of improvements identified through practice. The case study has shown that the actions

and outcomes from the formal SPI project were only part of the story, with changes occurring through the ongoing practice and improvisations of the practitioners. The innovations in the software process were primarily based on the reflective considerations of the individuals involved during the practice of developing the software. Some ideas were externally derived but even these initiatives were directly related to the needs identified at the project level. This finding is consistent with Mustonen-Ollila and Lyytinen's (2003) recent study that shows that the majority of innovations originate from internal sources. This view of innovations contrasts with much of the traditional SPI literature, and with aspects of recent studies such as the diffusion of innovation literature, which assumes that ideas are mostly externally generated.

The process of improvement needs to account for these reactive, reflective changes if the processes are to be improved not just extemporised. One of the difficulties at InfoServ was the uncoordinated manner in which they allowed the SPI project to progress. The sanctioning of the tasks through management channels was sometimes insufficient to overcome other constraining factors in the context. The infrastructure at InfoServ was minimal, but further support mechanisms might have helped to maintain and coordinate the actions.

There is a need to promote sustainable development of the processes by integrating the experiences of the developers, their learning through action, and sharing that learning. The learning processes that informed the SPI activity were ongoing, not simply delivered via training. Training was seen to assist in the identification of suitable innovations, but not all initiatives from training were incorporated. Rather it

was when a need was clearly answered, often serendipitously, within a training event that it was incorporated into the practice.

Training also provided a means for developing shared language. Joint training can help to support a change in structures of signification and social integration (Aladwani, 2002), but another way that this can happen is by providing space for story telling as this will help to share knowledge and experience (Ciborra, 1999; Watson, 1994).

The InfoServ team identified two formalised means of supporting the dissemination of process improvements: project reviews of each release discussed the process innovations, and managers later were able to set aside a meeting room for ad hoc discussions to consider more immediate issues. However, these lessons were limited to the project team. Further value may have been gained by looking for lessons across projects, teams and divisions.

At InfoServ, this dissemination of ideas occurred through ongoing negotiation. In this sense then conflict should not be regarded as a negative aspect of organisational life, rather debate should be encouraged. For the debate to engender a common feeling of improvement, trust between the actors is important.

Sweeney and Bustard (1997) suggested that it is important to approach SPI tasks carefully if genuine overall improvement is to be achieved, as change is sometimes seen as disruptive and, to many of those affected, unwelcome. Involvement of all the developers was a fundamental strand of the philosophy of the SPI initiative at

InfoServ. This involvement was an attempt to develop a spirit of togetherness, reflecting the literature that suggests this is helpful. Here examples were evident of how individuals, and sub-cultures, both enabled and resisted changes to the processes through the continuous exercising of power. Power should not necessarily be understood negatively or as solely in the hands of those in authority, but as a means used by the individual seeking to fulfil their self-interest. It is enabling and productive as well as constraining; it is what enables the existence of different positions within the organisation. From this perspective then power is part of all organisational relations and is the process through which organisations are sustained, reproduced and changed.

The norms of the organisation, as understood at local levels of communities-of-practice, shaped the retention of existing ideas or the introduction of new ideas. These habits and traditions were drawn on by agents to sanction their actions. However, actors were not passively moulded by their culture, fitting with Giddens (1984) view that agency takes place with knowledge and practical consciousness. The process-in-use within any community-of-practice acted as a norm: informing, guiding, and organising future practice. Such norms thereby sustained existing approaches. The norms changed as they were challenged through experience, negotiation within the group, and through the introduction of new ideas from other sources.

Within packaged software organisations product quality and customer delivery on schedule are more important than following a prescribed process. At InfoServ there were a number of key examples of process changes that occurred because they were important for the business, but this link was not explicitly made and therefore not

measured. By seeking process changes that enable an organisation to achieve good quality product development rather than process rigidity per se, key business measures should improve. The judgement of this improvement can be achieved through client-based assessment, reflecting the value of the improvement to them (Harkness et al, 1996).

Changes in the process-in-use at InfoServ were seen to occur through different forms of innovation, reflecting Brown and Eisenhardt's (1997) proposal that successful innovation involves improvisation, communication, experimentation, and choreographed transitions. Finding a way to facilitate this level of inventiveness within the software process is an important lesson from this case study. The theoretical development provides a step towards that understanding through the recognition of the situated nature of the improvement. The following section will pick up and develop the implications from the case to suggest the how this might be taken forward to inform future practice.

## 6.3 Implications for Future Practice

### 6.3.1 Towards an agile approach to SPI

The lessons discussed above from the case resonate with the contemporary views of agile software development, where evolution and appropriateness are seen to be paramount. The purpose of this section is therefore to explore how these agile concepts can inform the SPI practice. This section contributes to this development by linking agile methods with the SPI concepts. The suggestions will link findings from the study to other recent developments in the SPI literature that have adopted a similar position.

The agile software development manifesto (see Cockburn, 2002, p.213, bold in original) highlights four values:

**Individuals and interactions** over processes and tools

**Working software** over comprehensive documentation

**Customer collaboration** over contract negotiation

**Responding to change** over following a plan

Whilst the items on the right have value, the authors of the manifesto value the items on the left more. Similarly for process improvement, it is argued that the SPI activity should be agile rather than following deterministic and normative based approaches.

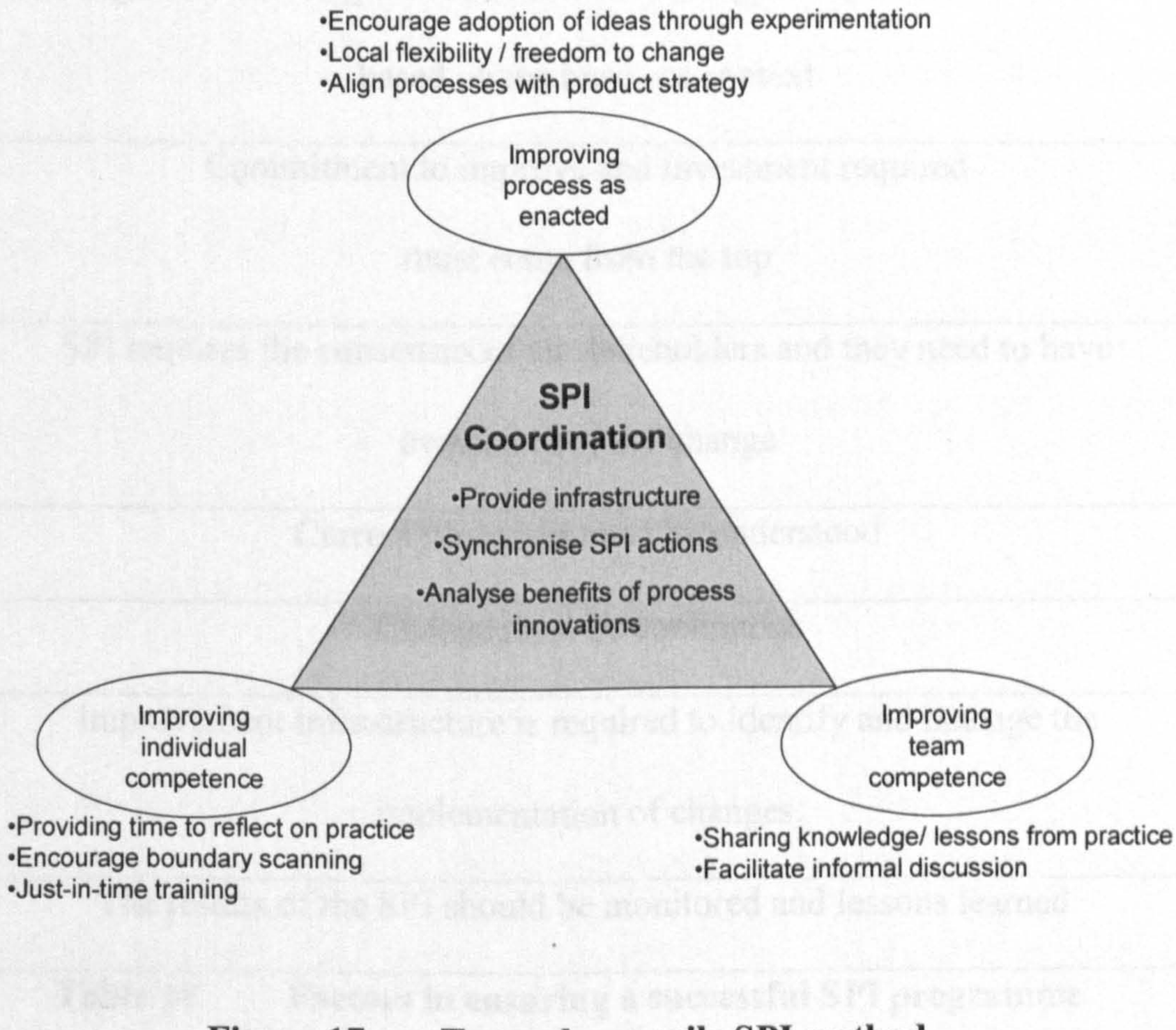



**Figure 17 Towards an agile SPI method**

An agile approach to SPI would be responsive and flexible to local needs, encourage innovation in the process, build SPI projects around those who are motivated,

encourage self-organising competent teams, and promote sustainable development of the processes. Figure 17 conceptualises the features considered necessary to support an agile approach to SPI. Each of these features is discussed below by drawing on the existing literature and the findings from the case. The basis for these discussions is that, rather than continuing the quest to find a perfect software process, researchers need to recognise that 'every individual software organisation must develop its own practices of continuous self-improvement and establish an appropriate improvement infrastructure' (Birk and Rombach, 2001, pp.34-35).

| **Factors in ensuring a successful SPI programme** |
|---|
| Changes must be aligned with business strategy and priorities should be based on the business context |
| Commitment to improve and investment required must come from the top |
| SPI requires the consensus of all stakeholders and they need to have ownership of the change |
| Current processes must be understood |
| Change must be continuous |
| Improvement infrastructure is required to identify and manage the implementation of changes. |
| The results of the SPI should be monitored and lessons learned. |

**Table 10 Factors in ensuring a successful SPI programme**

This discussion is not intended as another prescriptive model but to highlight aspects and issues that need consideration. Above all agility in the process is an attitude of mind not just another formula to follow. Nor is the purpose of this model to replace

existing continuous improvement approaches, but to complement them. Indeed, the ideas encapsulated within the discussion reflect the principal change management factors that have been identified as important in successful SPI projects (Humphrey, 1989; Curtis and Paulk, 1993; Zahran, 1998). These factors (summarised in Table 10) are integral to the agile approach discussed below, which addresses each aspect of the above model.

### 6.3.2 Improving processes as enacted

Improvement in process management is 'a continuous effort to design an efficacious process by understanding the relationship between process configurations and process outcomes and embedding the knowledge in the process through routines and formal process definitions' (Ravichandran and Rai, 2000b, p.202). In the context of a package software organisation the outcome of the process is directly associated with its product base. However, the danger, as seen in this case, is that because of commercial pressures organisations 'prevent, hamper, and even abort well-planned, low-key SPI efforts. Further, many of the cited SPI frameworks are hardly suited or applicable in their present forms' (Conradi and Fuggetta, 2002, p.93). The software engineering community therefore need to understand and develop improvement approaches that have constant change at the core.

'The real issue is product innovation not process stabilization and refinement' (Conradi and Fuggetta, 2002, p.95). To be responsive and flexible to local needs process innovation needs to be aligned to the business needs. This does not imply that the process improvement activity should be 'driven' by the business strategy but that changes in the software process should be informed by the planned product base. The need to focus on product improvement as well as process improvement has recently

been recognised as essential by one organisation reaching CMM level 5 (McGarry and Decker, 2002), but in a product based organisation this link is fundamental to the successful improvement of the process from the outset.

Nonaka and Takeuchi (1995) state that organisations must maintain a highly adaptive and flexible approach to new product development because it rarely proceeds in a linear or static manner and instead involves an iterative, dynamic and continuous process of trial and error. They believe that self-organising project teams should oversee new-product development. Project teams can be organised in organic structures that grow and shrink according to the needs of the product under development. Ayas (1996) shows that self-managed teams, organised into a dynamic structure that changes with the need of the product, share knowledge through association with different colleagues. Organisations should be willing to give a high degree of autonomy to such teams especially as they cope with ambiguity, fluctuation and creative chaos in the start up phases of new product development where little prior knowledge exists and taking initiatives and risks are necessary. The concept of self-organising, however, raises the question of who is responsible for the decisions about resource priorities and how to avoid an idiosyncratic approach to the process improvement, especially in the intrinsic political nature of the SPI activity. This aspect will be discussed further in the coordination section below.

Explicitly moving away from a software process philosophy of stability to one that is adaptive and flexible to local concerns will encourage a change in the actors' intentions, as the norms of the organisation will change from remaining consistent to finding the most appropriate means of undertaking a task. This is not to suggest that

methodologies or process life-cycles should be abandoned, rather the aim should be to produce a successful product rather than following a pre-defined methodology at all costs. Indeed, the objective is not change for its own sake, as there is a need to balance between optimising the current approaches and experimenting with new ideas (Van Solingen, 2004). Yet, we have to move away from the belief that every instance of a process will be the same, towards recognition that individuals should be the judges of what is good practice.

The primary means for adjusting the process to suit the product development needs will be through learning from practice as individuals identify adjustments in the life cycle. Gasston and Hallorn (1999) advocate that, through leadership, norms should be established that will encourage learning. This view concurs with that of Conradi and Fuggetta (2002) who contend that SPI is about learning not control. Nevertheless, the reasons for adapting the defined process in each case should be understood so as to enable consideration as to whether the adaptation is just for this instance or a lesson for future projects. Learning from the action of developing the products will encourage the innovations in the process to be based on practice not just introduced in isolation. In part this implies we need to capture the existing practice and share it, but also teams should seek out ideas that are appropriate to its needs and avoid simply introducing a new technique because it is available. Much of this innovation capability lies with the individuals and is discussed below, but aspects can be encouraged at the organisational level.

To achieve this ongoing learning from practice we need to do more than just recognise that it happens gradually, we need to encourage innovation in the process.

Or, as Bach (1995, p.96) puts it, we need to create conditions where 'useful heroes are born and healthy heroes thrive'. Encouragement to innovate, whether by bringing in new ideas from the software profession or developing approaches locally, requires a suitable mixture of experimentation, training, boundary scanning, and time to reflect on the current experience. The case showed that experimentation assisted both in the individual learning and in persuading others in the team as the idea was tried in practice. This sharing helps the competence of the team to improve not just individuals.

### 6.3.3 Improving individual competence

The enactment of the software development practices depends upon the existing knowledge of practitioners. No amount of process improvement activity will produce good quality software if the people are not able to understand how the latest approach can be used. So if processes are to improve the individual's knowledge of them needs to be developed. The knowledge of practitioners is changed through ongoing practice. However, the learning process is not the same for everyone, even if they shared similar experiences, as learning is also influenced by the actors' original knowledge base and their interaction with networks of external practitioners. The ability to reflect on their own work and competence, and critically assess them is an essential feature of improving the quality of the work. The creativity to improvise and the imagination to bring in ideas from outside their own prior knowledge are facets of the developers' competence that are important, but relatively under developed skills in software engineering education. So whilst it is outside the scope of this study, further consideration needs to be given to developing the reflective, resourceful, innovative competences within higher education.

To encourage the reflection and introduction of external ideas, organisations can create a learning environment, whereby the IS staff are encouraged to continuously augment their knowledge. By doing so, companies will benefit through learning from previous experiences and increasing their ability to take on advanced techniques or new ideas. Learning should be continuous. Organisations sanction this learning by developing programmes and incentives to encourage this to happen. An example of this can be seen at BP Exploration, who consider ongoing education to be a high priority, as highly trained and motivated staff are key to their future success (Cross et al, 1997). Professional development, however, is not always well served by external training courses. Timely and tailored training can be helpful in introducing new concepts, but apprentice-like learning can support the implicit sharing of skills and knowledge within a team.

Ciborra (1994) shows that reflecting-in-action helps organisational actors to gain insight into the background context of the change thus helping to reshape and restructure the organisational context: that of business policy and software development. Mapping Ciborra's (1994, p.21) findings on to process improvement, we can see that changes that are 'developed close to and serve the grassroots of the organisation' will enable the creation of locally appropriate but significantly beneficial processes. Giddens (1984) shows that human knowledge is developed through actors reflexively monitoring their own and others' actions and the consequences, both intended and unintended. This reflexivity happens naturally through the action, but software management can support this by providing time and resources during and after projects to consider the lessons. Continuous improvement requires the opportunity, and encouragement, for reflection on previous activities.

Whilst learning through reflection occurs as a natural part of action, time is an important factor in the degree of learning. Some MAP developers felt that when they were under pressure to produce the product then they did not have time to fully consider the implications of their actions. This interpretation is supported by Mustonen-Ollila and Lyytinen (2003) who found that innovations occurred when the project members had slack time available. So to create an environment responsive to the development needs, it is necessary to provide sufficient slack time to encourage actors to put their views into practice, but this time needs to be managed. Time on its own is not sufficient, organisations need to encourage a willingness to challenge existing practice and seek innovative solutions to current problems.

### 6.3.4 Improving team competence

From an organisational learning perspective software process management should consider promoting learning and sharing within and between teams more explicitly too. This helps to enable the reuse of innovations across members of a team and across teams.

Knowledge sharing across a team can occur explicitly through the codification of that knowledge or implicitly through socialisation. The case showed that the latter helped to change the norms, and thus the process-in-use as shared meaning was developed through socialisation: shared standards, understanding and experiences. Through interaction a common language emerges across each community-of-practice. Each community needs to find ways of sharing ideas, requiring a level of trust across the organisation. It has been shown that the emergence of the process occurs through negotiated practice. Not all changes are fully accepted, with evidence of resistance

and powerplay. However, attempts to avoid all conflict are mistaken, as small doses of conflict within the team can help to communicate ideas more forthrightly (Cockburn, 2002).

Where project teams remain distinct it is necessary to enable the knowledge transfer through documentation of the lessons or cross training. However, this is not to suggest that such knowledge can be transferred en-masse, rather it means that making the tacit reflections more explicit will reinforce the structures of signification within (and across) communities-of practice. The sharing of understanding gained from experience in projects can be embedded in formal processes, but is more readily shared by the exchange of people, ideas and mutual experiences. One way to promote this is through knowledge management, but perhaps a more active form of sharing would be by establishing networks of professionals across organisational groups. Truex et al (1999) suggest that back channel communications (virtual discussion groups, white boards, etc.) can enhance this sharing. An organisation's ability to manage its knowledge base is an important management skill.

### 6.3.5 SPI Coordination

To support the learning and innovation activity, the SPI initiative can be supported through a suitable infrastructure. The SEI view of setting up process action teams under the remit of a software engineering process group (SEPG) is a model that can be adapted to suit this more emergent view of SPI. Rather than the SEPG being the sole identifiers of process areas that need to be introduced, they could act as a facilitating group. By supporting through training, encouragement, resources, and progress management the SEPG can enable the introduction of significant changes to the defined process. The group can also act as a research and development group,

facilitating boundary scanning, thus helping an organisation to be more aware of and ready to take on external innovations.

Working with the SEPG, process action teams are useful vehicles for supportive development and introduction of innovations. These can be formed from the available body of developers. Yet whilst all developers are shown to be reflexive in nature, and thus change their own practice through that reflection, not all actively support any organisational initiative. It is suggested therefore that any planned activity as part of a SPI project should be centred around those who are intent on participating. The literature suggests that SPI should be an inclusive activity as participation will help to generate shared norms (Mathiassen et al 2002; Conradi and Fuggetta, 2002), but as shown in the case study personal motives are an important factor in the efficacy of the change process. So whilst all can be involved, and drawn into the discussions, forcing everyone to be involved may be counter-productive. Maintaining an open dialogue about the innovations will help to engage those who are less interested, and help to facilitate the negotiation required to bring about the change. So whilst continuous personal development and continuous improvement are key to incremental improvement, focusing on selected process areas and managing improvements in those is more likely to be beneficial.

The SEPG could also act as the control group for process actions, synchronising SPI actions over time. New initiatives can be incorporated into the improvement action plan, thus maintaining legitimacy for the SPI project. Also, any lessons from previous development activities can be incorporated into the action plan. The resources can be prioritised by the group to suit the business needs, as well as personal motives. As at

InfoServ, Jakobsen (1998) suggests seeking early success opportunities as this fuels further innovation. Software practitioners will be motivated by the successful application of their knowledge and seek further opportunity to learn, so early successes need to be seen to be useful by the team.

One way to improve the relevance of the SPI activity is to align it with the emergent business strategy. Both planned and improvised improvements could be coordinated to focus upon the areas that are expected to be important to the business to engender a sense of purpose over the longer term. Thus, changes to the software process would enable the organisation to deliver products suitable to its customer base allowing the process improvement to be judged on product related benefits, as well as the internal process benefits.

It is useful to make an assessment of benefits in the early stages of innovations and to assess the associated risk with this change (Gibson, 1998). So to encourage ongoing innovation the usefulness of the changes should be analysed by identifying and collecting key metrics for feedback on the process. The primary measures for package development organisations are best associated with customer values, which may be collected through surveys or through direct measures such as sales revenue or help desk statistics. Also, to give a more immediate review of the impact, software development metrics can be collected. It is suggested that the SEPG, along with any business strategy group, discuss the key targets for improvement and then actions, measures and success can all be associated with those goals.

### 6.3.6 Summary

The purpose of this discussion was to highlight the need for greater agility and flexibility in the process improvement activity. The ideas presented here relate to aspects that have arisen from the InfoServ case. This model is intended as indicative of the issues that need consideration by IS management, but the ideas are not an exclusive list therefore the views should not be seen as prescriptive. The model highlights the need to learn to improve through situated practice within an organisational framework that supports the needs of the business.

# Chapter 7. Conclusion

## 7.1 Introduction

This final chapter draws together the major contributions of the study from the findings and lessons identified in previous chapters. The critique of the research strategy in Chapter 3 will be extended to evaluate the lessons from the research experience, showing how the approaches have informed the findings and identifying issues with the methods adopted. The thesis will finish by proposing ways of taking the research forward.

## 7.2 Major Contributions

Previously, SPI has been mainly understood as something to be engineered. Yet this literature has been criticised because it fails to understand the microdynamics of software practice. Here the ongoing relationship between software process improvement and product development is seen as a constant and fundamental aspect of software practice. The thesis therefore makes significant contributions to information systems theory and practice by: revealing the nature of SPI activity within a packaged software organisation; developing a theory of SPI as a form of emergent change adding to recent developments in the SPI field; proposing a model of SPI as an agile, flexible activity; and extending existing IS research methods by elaborating on the previous use of a critical interpretive strategy.

In reviewing the organisational change literature in Chapter 2 it was argued that SPI could be better understood as emergent, situated change rather than a rational, predetermined activity that transformed an organisation from one state to another. By building on the continuous process improvement models in the literature, it was

shown that an emergent view provided an alternative way of looking at the improvement activity. The emergence of the software process is shown to be intricately linked with the contextual features that enable and constrain it. The changes to software processes are intertwined with the changes to software products, each informing the other. This view of software process improvement was highlighted in the theoretical framework in Chapter 5.

To suggest that SPI is emergent could be contentious. In the software engineering domain the processes to be changed are often seen to be driven by an external reference point, such as a maturity model, and theorised as occurring consistently across all members of an organisation. Here an alternative theory is developed, drawing on the broader organisational literature, which proposes that the change is not linear through time, nor is it uniform across all actors or all tasks, and that it cannot always be pre-planned or foreseen. The data in Chapter 4 illustrated the irregularity of the changes over time, between communities-of-practice, and in each instantiation of a method in practice. The analysis revealed how the changes occurred through a structuring process, linking action with its context and the context with the actions. The framework was based upon Giddens's Structuration Theory, which has been used by other IS researchers to explain changes to IS practice. Here though the work is developed to draw out the learning and political features of the emergence. These are each understood to occur as part of the structuring process.

This research adds to a limited, but growing, body of work exploring the organisational issues of SPI. The findings reflect the learning intensive nature of SPI recently identified elsewhere. This case extends the understanding in the existing

literature by showing that this learning is not orchestrated but occurs through situated practice and forms part of the integration of the product development and process improvement. Drawing on an active view of learning, where individuals learn through reflection-in-action, process improvement has been understood to occur through ongoing sensemaking. In turn, as this individual learning is shared through communities-of-practice the process-in-use becomes the norm, which is drawn on the next time. In time the espoused theory changes to reflect these norms.

Likewise, the use of the supporting theoretical framework informs our understanding of the political nature of SPI. The improvement is understood to be a negotiated process of change occurring through dynamic relationships between agents. Software practice draws on the norms of the communities-of-practice to sanction the decisions to use, adopt or avoid aspects of the software process. Actors draw on structures of domination to achieve this mobilisation of (or resistance to) the use of personal and organisational resources to enable or constrain an innovation. Ongoing software practice influences individual motives, and challenges the existing norms of the community-of-practice. This interaction forms part of the emergent nature of SPI. To date there has been a paucity of work in this area, so this theoretical development suggests a significant area of research that needs further exploration.

From these insights, lessons are derived to inform future SPI practice. Organisations can support the ongoing improvement through recognition of the learning and political aspects of the change. Issues facing IS managers are outlined in a proposed model of agile SPI, reflecting the understanding of SPI as an emergent activity. The purpose of this model is to highlight the need for greater flexibility in the process

improvement activity, set within an organisational framework that supports the needs of the business and individual needs. The model relates primarily to packaged software organisations, but can be interpreted for other domains. In presenting these suppositions, it is intended to provoke debate and to challenge the existing software engineering hegemony. Further development and testing of these ideas is required; these tasks form part of the planned future research materialising from this thesis.

Finally, this study adds to a growing body of interpretive IS research. Specifically, the research takes a critical hermeneutic perspective, addressing previous criticisms of interpretive studies. Adding to Myers' (1994a; 1994b; 1997) previous work, the thesis shows how the general principles that he outlines can be achieved. The need for the research to be reflexive in nature, to challenge preconceived ideas and bias, and the need to look for contrasts in the data, not simply confirmatory findings, are critical elements of the methodology. The next section further explains the contribution of the adopted research strategy.

## 7.3 Critique of the Research Project

### 7.3.1 Relevance of Research

The question of relevance was raised in Chapter 3 and the methods adopted were discussed in that light. It is worth returning to this question, now that the outcomes of the study have been presented, to discuss whether the research approach helped to develop findings of relevance to practitioners and researchers. The dialectical hermeneutic position enabled the analysis to look beyond the face-value of the case. In so doing the study brought out issues that are relevant to understanding how

software processes are, and are not, improved, and therefore showing how this activity might be managed differently in the future.

Had the output been left as a theoretical discussion its relevance to practice may not have been self-evident. The discussion in Chapter 6 is designed to overcome this difficulty and presents a series of related suggestions for adopting the lessons in SPI practice to encourage their implementation. The theoretical development expands the recent trend to investigate SPI as a form of organisational change, so the outcomes of this study inform this body of knowledge by drawing together the learning and political nature of the situated change. The use of social theory to support the theoretical discussion broadens the study's relevance for the wider group of IS researchers; this aspect is discussed further below.

### 7.3.2 Theoretical Underpinning

The theoretical assumptions used in this thesis have been based upon contextual, process theory. The analysis of the emergence of the software processes and their related context has been supported by considering the intertwining of the actions and social structures of the actors in the case over sufficient time to observe the patterns of this interaction. Structuration Theory supports an emergent perspective of action that is contextually situated. This process theory is integrated with a dialectical hermeneutic approach to support the analysis. Both these approaches emphasise the enabling and constraining nature of the context, and the subjective, socially constructed nature of the actor's actions that recreate the context. They thereby provide underpinning theory that illuminates the essence of the emergent change.

By adopting a dialectical hermeneutic perspective it encouraged the research to look beyond the statements of the individual actors to challenge their position. The established views of the researcher, and those in the literature, were not taken for granted. Established views, such as the Diffusion of Innovation Theory, were discounted in favour of one that corresponded with the evidence in the data. The theoretical position was developed by comparing and contrasting the parts and the whole of the case: contrasting the observed actions with the statements; statements by people at different points in the case history; and the different perceptions of actors within the case. On its own this analysis may have remained relatively local, but through the use of underpinning social theory it is possible to generalise the findings. The use of social theory in the IS field is now well established following the recognition that the IS domain is just another forum for social change. Whilst it is sometimes argued that the technological aspects need to be a separate element of the analysis, here they were subsumed into the contextual features of the case. Even if technology should be separated out for special attention in some cases, here the technical aspects were not the focus of the study and so there was no need to pay heed to this issue.

### 7.3.3 Research Strategy

The purpose of the research was to uncover how the process improvements occurred within the context of a packaged software organisation and the reasons why these changes happened. The interdisciplinary nature of this study helped to understand the organisational aspects of the changes. The focus was on personal accounts and perspectives of the software practice rather than the specifics of the software techniques. The rich data captured as part of this study has enabled an in-depth analysis of the interactions, personal perspectives, and organisational context. This

richness was made possible by the level of access. Whilst other strategies were possible, a single case allowed time for an elongated period of observation, facilitating a processual analysis of the case that revealed the emergent nature of the process improvement. Aspects of the complex interplay between the actions and context would not have been identified through a standard short-term data collection activity. The analysis of the changes drew out the nuances of the emergent nature of the process. The research strategy assisted in this analysis through the inductive approach adopted, where concepts were developed from the case data interspersed with periods enfolding the literature.

Longitudinal studies of this nature are relatively rare, but should be encouraged so as to overcome the problems of only taking a snap-shot of data. However, one area that could have been improved in this study is the long-term follow up. Further visits were made to the case site and the concepts were corroborated by key informants, but minimal further data was collected to establish the outcomes of the SPI action over time. Questions such as whether the benefits were maintained and how the processes continued to change would have been usefully answered. One reason that such data was not collected was that several of the key protagonists left the organisation, making access difficult. So it is recommended that future longitudinal studies build a post-observation period of data collection into their design.

## 7.4 Future Directions for SPI Research

Five proposals are made for future research in SPI. These are organised into those items specifically related to the case data and the findings of the study, and those

topics that are considered to be important for researchers working in the area of SPI generally.

Two suggestions have been made earlier in the thesis for future work on this specific study. The first idea proposed related to the theoretical development. It was shown that the structurational perspective only provided limited support for analysing the role of the individual. Two other examples were suggested that could offer an alternative view of the data. Foucault's work could be used to further analyse the knowledge-power features of the case. Habermas's ideas could be used to analyse the relationship between language and action, drawing out the social and personal aspects of the communicative action. These are the two options that seem most immediately applicable, but others could be considered. The selection of an appropriate alternative social theory, by which to review the analysis of this study, could help to further illuminate how the changes occurred.

The second area already mentioned is the extension and evaluation of the agile SPI concept and model. Following further development of the ideas, it is proposed that these are assessed and extended using action research. Such an approach would add knowledge about the usefulness of the agile perspective and help to develop any supporting mechanisms. This approach would ensure that the concepts developed were pragmatic and usable, not expounded for their own sake.

Looking beyond the specific scope of this study there are three suggestions for researchers in this field: to develop a deeper understanding of the political nature of SPI; to further understand the issues relating to packaged IS organisations; and to

evaluate the appropriateness of existing higher education programme as a means for facilitating the reflexive, improvisational nature of improvement seen in this case study.

The political nature of organisational change has been widely researched, and yet there is a limited body of work that considers this aspect of software quality or SPI. Given the volume of prior work on power and politics in the wider literature, the options for this work are broad. It is suggested that the focus of this work should continue to be on understanding the nature of power relations as part of the way the improvement activity is enacted.

The majority of studies in SPI tend to be based on large software organisations working in technical development in areas such as defence, telecommunications, and safety critical systems, but few in information systems organisations. It has previously been shown that packaged software organisations tend not to adopt SPI based on a maturity model approach. This study has highlighted some of the contextual factors that affect an SPI project in this type of organisation, but these findings need corroborating against, or comparing to, other organisations in a similar situation. So, cross-organisational research should be designed to establish the patterns across the packaged software industry, for example by undertaking surveys or further in-depth case studies. Suitable contrasting case sites would include one of those that had applied a maturity model or industry standard approach, one involving an organisation-wide SPI project, or where the software product was more business critical than a market analysis tool, such as an enterprise resource planning system.

Finally, as SPI is a learning process, professionals need the capability to improvise, to be resourceful and to develop their knowledge. It was asserted earlier that these skills are relatively under developed in software engineering education. The lifelong learning rhetoric has pervaded the higher education sector for some time, but this is often treated as a side effect of any programme of study rather than a central element of it. Research is therefore required to evaluate the current approaches and how they support the graduate practitioner. Academics who are trying to bridge the gap between theory and practice need to review the learning and teaching strategies within their courses. Consideration should be given to developing programmes that have an apprentice-like approach that are developed and delivered in partnership with employers, or for existing employees part-time studies could be developed with a significant element of work-based learning. These approaches are just some options available, so a detailed study of the efficacy of current and innovative pedagogical approaches is required.

## 7.5 Conclusion

In conclusion this thesis has developed a theoretical framework of SPI as a form of emergent change. This work adds to the existing knowledge, showing that SPI is more than just implementing planned change. Much of the previous work does not adequately explain the dynamic relationship between organisational actors and the emergent organisation. The analysis of this case study has demonstrated the complexity of the changes that are involved in the improvement of software processes. The changes were shown to emerge through the reflexive nature of the software developers, shaped by the context and traditions of the organisation. The changes incorporated planned, improvised and adaptive actions of the developers so

the process changes through anticipated and unanticipated outcomes of the reflexive actions of the actors. Secondly, emergence happens at an organisational level. The context shifts through the outcomes of the actions. The situated practice of the individual becomes the process-in-use, which forms the norms that shape the ongoing practice. As these practices become routinised they become established as the espoused process, changing the values and knowledge of the organisation. It is therefore necessary to understand the changing theory-in-use by studying the process changes as they occur.

# Bibliography


Aaen, I., Arent, J., Mathiassen, L. and Ngwenyama, O. (2002), Mapping SPI Ideas and Practices. In: Mathiassen, L. Pries-Heje, J. and Ngwenyama, O. (eds.) *Improving Software Organizations*, 23-46, Upper Saddle River, NJ: Addison-Wesley.

Aitken, J., Harrison, A. and Partington, D. (1996) *Experiences In The Application Of Qualitative Methods For Research Into Organisational Change*. Working paper, Cranfield: Cranfield School of Management.

Allen, D. and Ellis, D. (1999) The Paradigm Debate in Information Systems Research, In: Brooks, L. and Kimble, C. (eds.) *Proceedings of the 4th UKAIS Conference*, 83-96, York 7-9th April, Maidenhead: McGraw-Hill.

Allison, I. (1999) Information Systems Professional Development: A Work-Based Learning Model, *Journal of Continuing Professional Development*, 2(3), 86-92.

Allison, I. and Merali, Y (2003) Software Process Improvement: Towards an Emergent Perspective, In: Levy, M. Martin, A. and Schweighart, C. (eds.), *Proceedings of the 8th UKAIS Conference*, 9 – 11th April, University of Warwick.

Aladwani A.M. (2002) An Empirical Examination of the Role of Social Integration in System Development Projects, *Information Systems Journal*, 12(4), 339-353.

Alvesson, M. and Sköldberg, K. (2000) *Reflexive Methodology: New Vistas For Qualitative Research*, Thousand Oaks, CA: London: Sage Publications Inc.

Argyris, C. and Schön, D.A. (1996) *Organizational Learning II: Theory, Method and Practice*, Reading: MA : Addison-Wesley.

Atkinson, P. and Hammersley, M. (1998) Ethnography and Participant Observation, In: Denzin, N.K. and Lincoln, Y.S. (eds.) *Strategies of Qualitative Inquiry*, 110-136, Thousand Oaks, CA: Sage Publications Inc.

Avison, D, Fitzgerald G, and Powell, P. (2001) Reflections on Information Systems Practice, Education and Research: 10 Years of the Information Systems Journal, *Information Systems Journal*, 11(1), 3-22.

Avison, D. and Fitzgerald G. (2003) Where Now For Development Methodologies?, *Communications of the ACM*, 46 (1), 78-82.

Avison, D.E. Shah, H.U. and Wilson, D.N. (1994) Software Quality Standards In Practice: The Limitation Of Using ISO 9001 To Support Software Development, *Software Quality Journal*, 3(2), 105-111.

Ayas, K. (1996) Professional Project Management: A Shift Towards Learning and a Knowledge Creating Structure, *International Journal of Project Management*, 14(3), 131-136.

Bach, J. (1994) The Immaturity of the CMM, *American Programmer*, 7(9), 13-18.

Bach, J. (1995) Enough About Process: What we need are Heroes, *IEEE Software*, 12(2), 96-98.

Baron, R.S., Kerr, N.L., and Miller, N. (1992) *Group Process, Group Decisions, Group Action*, Buckingham: Open University press.

Baskerville, R. and Wood-Harper, A.T. (1998) Diversity in Information Systems Action Research Methods, *European Journal of Information Systems*, 7(2), 90-107.

Benbasat, I. and Zmud, R.W. (1999) Empirical Research in Information Systems: the Practice of Relevance, *MIS Quarterly*, 23(1), 3-16.

Benbasat, I., Goldstein, D.K. and Mead, M. (1987) The Case Research Strategy in Studies of Information Systems, *MIS Quarterly*, 11(3) 369-386.

Bennetts, P.D.C., Wood-Harper, A.T., and Mills, S. (1998) The Soft Systems Methodology as a Framework for Software Process Improvement, *Journal of End User Computing*, 10(1), 12-19.

Berger, P. and Luckmann, T. (1967) *The Social Construction of Reality: a Treatise in the Sociology of Knowledge*, London: Penguin Publishers.

Bernstein, R.J. (1983) *Beyond Objectivism and Relativism*, Pennsylvania: University of Pennsylvania Press.

Bertilsson, M. (1997) The Theory of Structuration: Prospects and Problems, In: Bryant, C. and Jary, D. (eds.) *Anthony Giddens: Critical Assessment (vol I)* , 44-60, London: Routledge.

Bhandari, I. , Halliday, M. Tarver, E., Brown, D, Chaar, J. and Chillarege, R. (1993) A Case Study of Software Process Improvement During Development, *IEEE transactions on Software Engineering*, 19(12), 1157- 1170.

Birk, A and Rombach, D. (2001) A Practical Approach to Continuous Improvement in Software Engineering, In: Wieczorek, M. and Meyerhoff, D. (eds.) *Software Quality: State of the Art in Management, Testing and Tools*, 34-45, Berlin, Germany: Springer-Verlag.

Boehm, B. (1981) *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice-Hall.

Boland, R.J. and Hirschheim, R.A.(Eds.) (1987) *Critical Issues in Information Systems Research*. Chichester: John Wiley & Sons.

Bollinger, T. B. and McGowan, C. (1991) A Critical Look at Software Capability Evaluations, *IEEE Software*, 8(4), 25-41.

Brancheau, J.C., Janz, B.D. and Wetherbe, J.C. (1996) Key Issues in Information Systems Management: 1994-95 SIM Delphi Results, *MIS Quarterly*, 20(2), 225-242.

Briand, L. El Emam, K. and Melo, W. (1999) An Inductive Method for Software

Process Improvement: Concrete Steps and Guidelines. In: El Emam, K. and Madhvji, N.H. (eds.), *Elements of Software Process Assessment and Improvement*, 113-132, Los Alamitos, CA: IEEE Computer Society Press.

Brown, A.W. and Short, K. (1997) On Components and Objects: The Foundations of Component-Based Development. In: Nahouraii, E (ed.) *Proceedings of Fifth International Symposium on Assessment of Software Tools and Technologies*, 112-121, Los Alamitos, CA: IEEE Computer Society Press.

Brown, S.L. and Eisenhardt, K.M. (1997) The Art of Continuous Change: Linking Complexity Theory and Time-Paced Evolution in Relentlessly Shifting Organizations, *Administrative Science Quarterly*, 42(1), 1-34.

Burrell, G. and Morgan, G. (1979) *Sociological Paradigms and Organisational Analysis*, Portsmouth, NH : Heinemann.

Card, D. (1991) Understanding Process Improvement, *IEEE Software*, 8(4), 102-103.

Carmel, E. and Sawyer, S. (1998) Packaged Software Development Teams: What Makes Them Different? *Information Technology and People*, 11(1), 7-19.

Cassell, P. (1993) *The Giddens Reader*, Basingstoke: Macmillan Press.

Cavaye, A.L.M. (1996) Case Study Research: a Multi-Faceted Research Approach for IS, *Information Systems Journal*, 6(3), 227-242.

Checkland, P. (1981) *Soft Systems Methodology*, Chichester: John Wiley & Sons.

Checkland, P. and Scholes, J (1990) *Soft Systems Methodology in Action*, Chichester: John Wiley & Sons.

Choo, C.W. (1998) *The Knowing Organization: How Organizations Use Information To Construct Meaning, Create Knowledge and Make Decisions*, New York: Oxford University Press.

Christie, A.M., Earl, A.N., Kellner, M.I, Riddle, W.E. (1996), A Reference Model For Process Technology. In: Montangero, C. (ed.) *Proceedings of Software Process Technology: 5th European Workshop (EWSPT'96)*, 3-17. Nancy, France, October 8 – 11th , Berlin, Germany: Springer.

Ciborra, C. (1994) The Grassroots of IT and Strategy, In: Ciborra, C. and Jelassi, T. (eds) *Strategic Information Systems: a European Perspective*, 3-24, Chichester: John Wiley & Sons.

Ciborra, C.U. (1999) A Theory of Information Systems Based on Improvisation, In: Currie, W.L. and Galliers, R. (eds.), *Rethinking Management Information Systems*, 136-155, Oxford: Oxford University Press.

Cockburn, A. (2002) *Agile Software Development*, Boston, MA: Addison-Wesley.

Collins, D. (1998) *Organizational Change: Sociological Perspectives*, London: Routledge.

Collinson, D. (1994) Strategies of Resistance: Power, Knowledge and Subjectivity in the Workplace. In Jemier, J. Knights, D. & Nord, W. (eds) *Resistance and Power in Organizations*, London : New York : Routledge

Conradi, R.and Fugetta, A. (2002) Improving Software Process Improvement, *IEEE Software*, 19(4), 92-99.

Coopey, J., Keegan, O., and Emler, N. (1998) Managers' Innovations and the Structuration of Organizations, *Journal of Management Studies*, 35, 263-284.

Cotterman, W.W. and Senn, J.A. (1992) *Challenges and Strategies for Research in Systems Development*. Chichester: John Wiley & Sons.

Craib, I. (1992) *Anthony Giddens*, London: Routledge.

Crosby, P (1979) *Quality is Free*, New York: McGraw-Hill.

Cross, J., Earl, M.J., and Sampler, J.L. (1997) Transformation of the IT function at British Petroleum, *MIS Quarterly*, 21(4), Dec, 401-423.

Currie, W. and Willcocks, L. (1996) The New Branch Columbus Project at Royal Bank of Scotland: the Implementation of Large-Scale Business Process Re-engineering, *Journal of Strategic Information Systems*, 5(3), 213-236.

Curtis, B (2000) The Global Pursuit of Process Maturity, *IEEE Software*, 17(3), 76-78.

Curtis, B. and Paulk, M (1993) Creating a Software Process Improvement Program, *Information and Software Technology*, 35(6/7), 381-386.

Curtis, B., Kellner, M.I. and Over, J. (1992) Process Modeling, *Communications of the ACM*, 35(9), Sept, 75-90.

Cusumano, M.A. (1997) How Microsoft makes Large Teams work like Small Teams, *Sloan Management Review*, Fall, 9-20.

Cusumano, M.A. and Selby, R.W. (1997) How Microsoft Builds Software, *Communications of the ACM*, 40(6), 53-61.

Daft, R.L. and Weick, K.E. Toward a Model of Organizations as Interpretation Systems, *Academy of Management Review*, 9(2), 284-295.

Dahlbom, B. and Mathiassen, L. (1995) *Computers in Context: the Philosophy and Practice of Systems Design*, Cambridge, MA: Oxford : NCC Blackwell.

Darke, P., Shanks, G. and Broadbent, M. (1998) Successfully Completing Case Study Research: Combining Rigour, Relevance and Pragmatism, *Information Systems Journal*, 8(4), 273-289.

Davenport,T.H. and Markus, M.L. (1999) Rigor Vs Relevance Revisited: Response to Benbasat and Zmud, *MIS Quarterly*, 23(1) 19-23.

Dawson, S.P. (1994) Continuous Improvement in Action, *Information Systems Management*, winter, 31-39

DeMarco, T. and Lister, T. (1987) *Peopleware: Productive Projects and Teams*, NewYork, NY: Dorset House Publishing.

Denzin, N.K. and Lincoln, Y.S. (1998) Entering the Field of Qualitative Research. In: Denzin, N.K. and Lincoln, Y.S.(eds) *The Landscaped of Qualitative Research: Theories and Issues*, 1-34. Thousand Oaks, CA: Sage Publications.

Diaz, M. and Sligo, J. (1997) How Software Process Improvement helped Motorola, *IEEE Software*, 14(5), 75-81.

Dion, R. (1993) Process Improvement and the Corporate Balance Sheet, *IEEE Software*, 10(4) 28-35.

Doherty, N.F. and King, M. (1998) The Importance of Organisational Issues in Systems Development, *Information Technology and People*, 11(2), 104-123.

Doolin, B. (1998) Information Technology as Disciplinary Technology: Being Critical in Interpretive Research on Information Systems, *Journal of Information Technology*, 13(4), 301-311.

Dorling, A. (1993) SPICE: Software Process Improvement and Capability dEtermination, *Software Quality Journal*, 2, 209-224.

Drouin, J. (1999) The SPICE project, In: El Emam, K. and Madhvji, N.H. (eds.), *Elements of Software Process Assessment and Improvement*, 45-55, Los Alamitos, CA: IEEE Computer Society.

Dyer, M. and Kouchakdjian, A. (1990) Correctness Verification Alternative to Structural Software Testing, *Information and Software Technology*, 32(1), 53-59

Edgar-Nevill, V.M.A. (1994) Evaluation of the SEI Software Capability Model within an Information Systems Context: in Pursuit of Software Quality, In: Ross, M., Brebbia, C.A., Staples, G. and Stapleton, J (eds.) *Software Quality Management II vol 1: Managing Quality Systems*, 263-278, Southampton: Computational Mechanics Publications.

Eisenhardt, K.M. (1989) Building Theories from Case Study Research, *Academy of Management Review*, 14(4), 532-550.

Eisenhardt, K.M. and Tabrizi, B.N. (1995) Accelerating Adaptive Processes: Product Innovation in the Global Computing Industry, *Administrative Science Quarterly*, 40(1), 84-110

Elden, M. (1981) Sharing the Research Work: Participative Research and its Role Demands. In: Reason, P. and Rowan, J. (eds.) *Human Inquiry: a Sourcebook of New Paradigm research*, 253-266. Chichester: John Wiley & Sons.

Fagan, M. (1986) Advances in Software Inspections, *IEEE Transactions on Software Engineering*, SE-12 (7), 744-751.

Feeny, D.F. and Ives, B. (1997) IT as a Basis for Sustainable Competitive Advantage, In: Willcocks, L., Feeny, D. and Islei, G. (eds) *Managing IT as a Strategic Resource*, 43-63, Maidenhead: McGraw-Hill.

Feeny, D.F. and Willcocks, L P. (1997) The IT Function: Changing Capabilities and Skills, In: Willcocks, L.P., Feeny, D.F. and Islei, G. (eds.) *Managing IT as a Strategic Resource*, 455-474, Maidenhead: McGraw-Hill.

Fenton, N.E. (1991) *Software Metrics: A Rigorous Approach*, London: Chapman and Hall.

Fielding, N.G. and Fielding, J.L. (1986) *Linking Data*, Beverley Hills, CA: Sage Publications.

Fincham, R. (2002) Narratives of Success and Failure in Systems Development, *British Journal of Management*, 13(1), 1-14.

Fichman, R.G. and Kemerer, C.F. (1997) The Assimilation of Software Process Innovations: an Organizational Learning Perspective, *Management Science*, 43(10), 1345-1363.

Fitzgerald, B. (1996) Formalized Systems Development Methodologies: a Critical Perspective, *Information Systems Journa*l, 6(1), 3-23.

Fitzgerald, B. and Howcroft, D. (1998) Towards Dissolution of the IS Research Debate: from Polarization to Polarity. *Journal of Information Technology*, 13(4), 313-326.

Fitzgerald, B. and O'Kane, T. (1999) A Longitudinal Study of Software Process Improvement, *IEEE Software*, 16(3), 37-45.

Fox, C. and Frakes, W. (1997) The Quality Approach is it Delivering? *Communications of the ACM*, 40(6), 25-29.

Fox, S.(1990) Becoming an Ethnomethodology User: Learning a Perspective in the Field. In: Burgess, R.G. (ed), *Studies in Qualitative Methodology: Reflections on Field Experience*, Vol 2., 1-23, Greenwich, Connecticut: JAI Press Inc.

Frewin, G.D. and Hatton, B.J. (1986) Quality Management – Procedures and Practices, *Software Engineering Journal*, 1(1), 29-38.

Gadamer, H-G. (1976) The Historicity of Understanding. In: Connerton, P. (ed.), *Critical Sociology, Selected Readings*, 117-133. Penguin Books: Harmondsworth.

Galliers, R. (1992) Choosing Information Systems Research Approaches. In: Galliers, R.D. (ed.) *Information Systems Research: Issues, Methods and Practical Guidelines*, 144-162, Blackwell Scientific Publications, Oxford.

Galliers, R. (1995) A Manifesto for Information Management Research, *British Journal of Management*, 6 (special issue), S45-S52.

Galliers, R.D., Merali,Y. and Spearing,L. (1994) Coping with Information Technology? How British Executives Perceive Key Information Systems Management Issues in the Mid-1990s, *Journal of Information Technology*, 9(3), 223-238.

Gasston, J. and Halloran, P. (1999) Continuous Software Process Improvement Requires Organisational Learning: An Australian Case Study, *Software Quality Journal*, 8(1), 37-51.

Gibbs, W.W. (1994) Software's Chronic Crisis, *Scientific American*, Sept, 86-95.

Gibson, R. (1998) Software process improvement: innovation and diffusion. In: Larsen, T.J. and McGuire, E.B. (eds.) *Information Systems Innovation and Diffusion: Issues and Directions*, 71-87, Hershley, PA: Idea Group Publishing,.

Gibson, R. (1999) Software Process Modelling, In: McGuire,E. (ed) *Software Process Improvement : Concepts and Practices*, 1-16, London: IDEA Group Publishing.

Giddens, A. (1979) *Central Problems in Social theory: Action, Structure and Contradiction in Social Analysis*, Basingstoke: Macmillan Press.

Giddens, A. (1984) *The Constitution of Society*, Cambridge: Polity Press.

Giddens, A. (1993) *New Rules of Sociological Method* (2nd ed.), Cambridge: Polity Press.

Giddens, A. (1999) Elements of the Theory of Structuration, In: Elliott, A. (ed.) *The Blackwell Reader in Contemporary Social Theory*, 119-130, Oxford: Blackwell Publishers Ltd.

Glaser, B.G. and Strauss, A.L. (1967) *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago: Aldine.

Goldenson, D. and Herbselb, J. (1995) *After the Appraisal: A Systematic Survey of Process Improvement, its Benefits, and Factors that Influence Success*, Technical Report CMU/SEI-95-TR-009, Pittsburgh, Pennsylvania : Carnegie Mellon University/ Software Engineering Institute.

Gray, E.M. and Smith, W.L. (1998) On the Limitations of Software Process Assessment and the Recognition of a Required Re-orientation for Global Process Improvement, *Software Quality Journal*, 7(1), 21-34.

Hailey, V. (1998) ISO 9001 and SPICE framework, In: El Emam, K. Drouin, J. and Walcélio, M. *SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*, 233-268, Los Alamitos, CA: IEEE Computer Society Press.

Hall, T.J. (1995) *The Quality Systems Manual: The Definitive Guide to the ISO 9000 Family and Tickit*, Chichester: John Wiley & Sons.

Hamel, J. (1993) *Case Study Methods*, Newbury Park, CA: Sage Publications Inc.

Hammersley, M. (1990) Reading *Ethnographic Research: A Critical Guide*. Harlow: Longman.

Harkness, W.L., Kettinger, W.J. and Segers, A.H. (1996) Sustaining Process Improvement And Innovation in the Information Services Function: Lessons Learned at The Bose Corporation, *MIS Quarterly*, 20(3), 349-367.

Harré, R. (1981) The Positivist-Empiricist Approach And Its Alternative. In: Reason, P. and Rowan, J. (eds.) *Human Inquiry: a Sourcebook of New Paradigm Research*, 3-18, Chichester: John Wiley & Sons.

Hartley, J.F. (1994) Case Studies In Organizational Research. In: Cassell, C. and Symon, G. *Qualitative Methods In Organizational Research: A Practical Guide*, 208-229, Newbury Park, CA: Sage Publications Inc.

Herbselb, J., Zubrow, D., Goldenson, D., Hayes, W., and Paulk, M. (1997) Software Quality and the Capability Maturity Model, *Communications of the ACM*, 40(6), 30-40.

Hertzum, M. (2002) The Importance of Trust in Software Engineers' Assessment and Choice of Information Sources, *Information and Organization*, 12(1), 1-18

Hollenbach, C., Young, R., Pflugrad, A. and Smith, D. (1997) Combining Quality and Software Improvement, *Communications of the ACM*, 40(6) 41-45.

Holstein, J.A. and Gubrium, J.F. (1997) Active Interviewing, In: Silverman, D. (ed) *Qualitative Research: Theory Method and Practice*, 113-129, London: Sage Publications.

Hosking, D.M. and Anderson, N. (1992) *Organizational Change and Innovation: Psychological Perspectives and Practices in Europe*. London: Routledge.

Huff, A. S., and Schwenk, C. R. (1990) Bias and Sensemaking in Good Times and Bad. In: Huff, A. S. (Ed.), *Mapping Strategic Thought*, 89-108, Chichester: John Wiley and Sons.

Humphrey, W.S. (1989) *Managing the Software Process*, Reading, MA: Addison-Wesley.

Humphrey, W.S. (1997) *Personal Software Process*, Reading, MA: Addison-Wesley.

Humphrey, W.S., Synder, T.R. and Willis, R.R. (1991) Software Process Improvement at Hughes Aircraft, *IEEE Software*, 8(4), 11-23.

Hutchings, T., Hyde, M.G., Marca, D. and Cohen, L. (1993) Process Improvement That Lasts: An Integrated Training And Consulting Method, *Communications of the ACM*, 36(10), 105-113.

IEEE (1993) *New IEEE Standard Dictionary Of Electrical And Electronic Terms*, Piscataway, NJ: IEEE.

Jacobsen, A.B. (1998) Bottom-up Process Improvement Tricks, *IEEE Software*, 15(1), 64-68

Jansen, P. and Sanders, J. (1998) Guidelines for Process Improvement, In: El Emam, K., Drouin, J. and Waleélio, M. *SPICE: The Theory And Practice Of Software Process Improvement And Capability Determination*, 171-192, Los Alamitos, CA: IEEE Computer Society Press.

Jarvine, J. (1994) On Comparing Process Assessment Results: BOOTSTRAP and CMM, In: Ross, M., Brebbia, C.A., Staples, G. and Stapleton, J (eds.) *Software Quality Management II vol 1: Managing Quality Systems*, 247-262, Southampton: Computational Mechanics Publications.

Jermier, J.M., Knights, D. and Nord, W.R. (eds.) (1994) *Resistance and Power in Organizations*, London: Routledge.

Jones, C. (1996) *Patterns Of Software Systems Failure And Success*, Boston, MA: International Thompson Computer Press.

Jones, C. (1999a) The Economics Of Software Process Improvement. In: El Emam, K. and Madhvji, N.H. (eds.), *Elements Of Software Process Assessment And Improvement*, 133-149, Los Alamitos, CA: IEEE Computer Society.

Jones, G.R. and George, J.M. (1998) The Experience and Evolution of Trust: Implications for Cooperation and Teamwork, *Academy of Management Review*, 23(3), 531-536.

Jones, M. (1999b) Mission Impossible? Pluralism and Multi-Paradigm IS Research. In: Brooks, L. and Kimble, C. (eds.) *Proceedings of the 4th UKAIS Conference (Information Systems – The Next Generation)*, 71-82, York 7-9th April, Maidenhead: McGraw-Hill.

Jones, M. and Karsten, H. (2003) *Review: Structuration Theory and IS Research*, Working Paper WP 11/2003, Cambridge: Judge Management Institute, University of Cambridge, online at www.jims.cam.ac.uk, accessed December 2003.

Jorgensen, D.L. (1989) *Participant Observation*. Sage Publications Inc., Newbury Park, CA.

Jung, H., Hunter, R. Goldenson, D.R. and El-Emam, K. (2001) Findings from Phase 2 of the SPICE Trials, *Software Process Improvement and Practice*, 6, 205-242

Juran, J.M. (1979) *Quality Control Handbook (3rd ed)* The Free Press, New York

Kautz, K., Hansen, H.W., and Thaysen, K. (2001) Understanding and Changing Software Organizations, In: Ardis, M.A. and Marcolin, B.L. (eds.) *Diffusing Software Products and Process Innovations (IFIP TC8 WG8.6 Fourth Conference)*, 87-110, Norwell, MA: Kluwer Academic Publishers.

Kautz, K. and Larsen, E.A (2000) Diffusion Theory and Practice, *Information Technology and People* , 13(1), 11-26.

Kautz, K. and Nielsen, P.A. (2004) Understanding the Implementation of Software Process Improvement Innovations in Software Organisations, *Information Systems Journal*, 14(1), 3-22.

Keeni, G. (2000) The Evolution of Quality Processes at Tata Consultancy Services, *IEEE Software*, 17(3), 79-88.

Kellner, M.I. Briand, L. and Over, J.W., (1996) A Method for Designing, Defining, and Evolving Software Processes, *Proceedings of the 4th International Conference on the Software Process*, 37-48, Los Alamitos, CA: IEEE Computer Society Press.

King, S. (1996) Case Tools And Organizational Action, *Information Systems Journal*, 6, 173-194.

Klein, H.K. and Myers, M.D. (1999) A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly*, 23(1), 67-94.

Knights, D. and Murray, F. (1994) *Managers Divided: Organisation Politics and Information Technology Management*, Chichester: John Wiley & Sons.

Koch, G.R. (1993) Process assessment: the 'BOOTSTRAP' approach, *Information and Software Technology*, 35(6/7), 387-403.

Kolb, D.A. (1984) *Experiential Learning*, Englewood Cliffs, NJ: Prentice Hall.

Krishnan, M.S. (1998) The Role of Team Factors in Software Cost and Quality, *Information Technology and People*, 11(1), 20-35.

Kuvaja, P. (1999) BOOTSTRAP 3.0 – A SPICE1 Conformant Software Process Assessment Methodology, *Software Quality Journal*, 8(1), 7-19.

Kuvaja, P. and Bicego, A. (1994) BOOTSTRAP – A European Assessment Methodology, *Software Quality Journal*, 3(3), 117-128.

Lee, A, Cheng, C, Chadha, GS, (1995) Synergism between information technology and organizational structure: a managerial perspective, *Journal of Information Technology*, 10, 37-43,

Lee, A.S. (1989) A Scientific Methodology for MIS Case Studies, *MIS Quarterly*, 9(1), 33-50.

Lee, A.S. (1999) Researching MIS, In: Currie, W.L. and Galliers, R (eds) *Rethinking Management Information Systems*, 7-27, Oxford: Oxford University Press.

Lee, S., Goldstein, D.K., and Guinan, P.J. (1990). Informant bias in information systems design team research. In: Nissen, H.E. and Klein, H.K. (eds.) *Proceedings of the IFIP TC8/WG8.2 Working conference on the Information systems research arena of the 90's challenges, perceptions and alternative approaches (Information systems research : contemporary approaches and emergent traditions)*. Amsterdam: North-Holland.

Lefebvre, E. Lefebvre, L A, and Roy, M (1995) Technological penetration and organizational learning in SMEs: The cumulative effect, *Technovation*, 15 (8), 511-522.

Lehman, M.M. (1997) Process Modelling - where next? "Most Influential Paper of ICSE 9 Award", In: *Proceedings of ICSE 19*, Boston, 20 - 22 May, 549 - 552. Online at http: www-dse.doc.ic.ac.uk/~mml/feast/papers/postscripts/565.ps, accessed April 1998.

Lewin, K. (1951) *Field Theory in Social Science*. New York: Harper and Row.

Li, G. and Rajagopalan, S. (1998) Process Improvement, Quality and Learning Effects, *Management Science*, 44(11, 1 of 2), 1517-1532

Lyytinen, K. (1999) Empirical Research In Information Systems: On The Relevance Of Practice In Thinking Of IS Research, *MIS Quarterly*, 23(1) 25-28.

Lyytinen, K. and Damsgaard, J. (2001) What's Wrong with the Diffusion of Innovation Theory? In: Ardis , M.A and Marcolin, B.L. (eds.) *Diffusing Software Products and Process Innovations* (IFIP TC8 WG8.6 Fourth Working Conference, 173-204, April 7-10th, Banff, Canada), Norwell, MA: Kluwer Academic Publishers.

Lyytinen, K. and Robey, D. (1999) Learning Failure In Information Systems Development, *Information Systems Journal*, 9, 85-101.

MacDonald, S. (1998) *Information For Innovation: Managing Change Form An Information Perspective*, Oxford University Press: Oxford.

March, J.G. and Olsen, J.P. (1976) *Ambiguity and Choice in Organizations*, Olso, Norway: Scandinavian University Press.

Markus, M.L. (1983) Power, Politics and MIS Implementation, *Communications of the ACM*, 26(6), 430-444.

Mathiassen, L. (1998) *Reflective Systems Development*, Online at http: www.cs.auc.dk ~larsm rsd.html, Accessed on 5/12/2001.

Mathiassen, L., Pries-Heje, J. and Ngwenyama, O. (2002) *Improving Software Organisations: from principles to practice*, Boston, MA: Addison-Wesley.

May, T. (1993) *Social Research: Issues, Methods And Process*. OU Press, Buckingham.

McFeeley,B (1996) *IDEAL: A User's Guide For Software Process Improvement* (CMU/SEI-96-HB-001), Pittsburgh, PA: Software Engineering Institute/ Carnegie Mellon University.

McGarry, F. and Decker, B. (2002) Attaining Level 5 in CMM Process Maturity, *IEEE Software*, 19(6), 87-96.

McGill, S. (2001) Overcoming Resistance to Standard Processes, or, "Herding Cats",

In: Hunter, R.and Thayer, R. (eds) *Software Process Improvement*, 147-154, Los Alamitos, CA: IEEE Computer Society.

McGuire, E.G. (1996) Factors Affecting The Quality Of Software Project Management: An Empirical Study Based On The Capability Maturity Model, *Software Quality Journal*, 5, 305-317.

McGuire, E.G. and Randall, K.A. (1999) IS Change Agents in Software Process Improvement, In: McGuire, E.G.(ed), *Software Process Improvement: Concepts and Practices*, 93-107, London: Idea Group Publishing.

McKeen, J. D. and Smith, H. A. (1996), *Management Challenges in IS: Successful Strategies and Appropriate Action*, Chichester: John Wiley & Sons.

Mellis, W. (2001) Process And Product Orientation In Software Development And Their Effect On Software Quality Management. In: Wieczorek, M. and Meyerhoff, D. (eds.) *Software Quality: State Of The Art In Management, Testing And Tools*, 3-15, Berlin, Germany: Springer-Verlag.

Merali, Y. (2002) The Role of Boundaries in Knowledge Processes, *European Journal of Information Systems*, 11(1), 47-60

Mestrovic, S.G. (1998) *Anthony Giddens: the Last Modernist*, Routledge: London.

Miles, M.B. and Huberman, A.M. (1994) *Qualitative Data Analysis: An Expanded Sourcebook (2nd ed.)*. Thousand Oaks, CA : Sage Publications, Inc..

Miller, J and Glassner, B. (1997) The 'Inside' And 'Outside': Finding Realities In Interviews, In: Silverman, D. (ed) *Qualitative Research: Theory Method and Practice*, 99-112, London: Sage Publications.

Mills, H.D., Dyer, M. and Linger, R.C. (1987) Cleanroom Software Engineering, *IEEE Software*, 4, 19-25

Mingers, J. (2001a) Embodying information systems: the contribution of phenomenology, *Information and Organization*, 11, 103-128

Mingers, J. (2001b) Combining IS Research Methods: Towards a Pluralist Methodology, *Information Systems Research*, 12(3), 240-259.

Mingers, J. (2003) The Paucity of Multimethod Research: a Review of the Information Systems Literature, *Information Systems Journal*, 31, 233-249.

Mingers, J. and Willcocks, L. (2004) *Social Theory and Philosophy for Information Systems*, Chichester: John Wiley & Sons.

Moller, K (1991) Increasing software quality by objectives and residual fault prognosis, In: Ince, D. (ed) . *Software Quality and Reliability: Tools and Methods*, 102-112, Chapman and Hall : London.

Moreton, R. and Chester, M. (1997), *Transforming the Business: the IT Contribution*, Maidenhead: McGraw-Hill.

Morgan, G. (1993) Organizations as Political Systems. In: Mabey, C. and Mayon-White, B. (eds) *Managing Change 2nd ed*, 212-217, Liverpool: Paul Chapman Publishing / The Open University.

Mustonen-Ollila, E. and Lyytinen, K. (2003) Why Organizations adopt Information System Process Innovations: a Longitudinal Study using Diffusion of Innovation Theory, *Information Systems Journal*, 13(3), 275-297.

Myers, M.D. (1994a) Dialectical Hermeneuties: a Theoretical Framework for the Implementation of Information Systems, *Information Systems Journal*, 5(1), 51-70.

Myers, M.D. (1994b) A disaster for everyone to see: an interpretive analysis of a failed IS project, *Accounting, Management, and Information Technology*, 4(4), 185-201.

Myers, M.D. (1997) Interpretive research in information systems. In: Mingers, J. and Stowell, F. (eds.) *Information systems: an emerging discipline?*, 239-266, McGraw-Hill, Maidenhead.

Nandhakumar, J. and Jones, M. (1997) Too close for comfort? Distance and engagement in interpretive information systems research, *Information Systems Journal*, 7(2), 109-131.

Nissen, H.E. and Klein, H.K. (eds.) (1990) *Proceedings of the IFIP TC8 WG8.2 Working conference on the Information systems research arena of the 90's challenges, perceptions and alternative approaches (Information systems research : contemporary approaches and emergent traditions)*. Amsterdam: North-Holland.

Ngwenyama, O.K. (1990) The critical social theory approach to information systems: problems and challenges. In: Nissen, H., Klein, H.K., and Hirschheim, R. (eds.) *Proceedings of the IFIP TC8/WG 8.2 Working Conference on the information systems research arena of the 90's challenges, perceptions and alternative approaches*, 267-280, Copenhagen, Denmark, 14-16th December. Amsterdam: North-Holland.

Nicholson, B. and Sahay, S. (2001) Some Political and Cultural Issues in Globalisation of Software Development: Case Experience From Britain and India, *Information and Organization*, 11, 25-43.

Nonaka, I. (1991) The Knowledge-Creating Company, *Harvard Business Review*, 69, Nov-Dec, 96-104.

Nonaka, I. and Takeuchi, H. (1995) The Knowledge-Creating Company, New York: Oxford University Press.

Olsen, D.R. (1980) Some aspects of meaning in oral and written language. In: Olsen, D.R. (ed.) *The social foundations of language and thought: Essays in honor of Jerome S. Bruner*, 90-110, New York, NY, Norton and Company.

Opolski, K. and Szemborska, E. (1997), Further professional training - an investment by the employee or by the enterprise?, *Higher Education in Europe*, 22(4), 525-529.

Orlikowski, W.J. (1992) The Duality of Technology: Rethinking Concepts of Technology of Organizations, *Organization Science*, 3(3), 398-427.

Orlikowski, W.J. (1993) CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development, *MIS Quarterly*, 17(3) 309-340.

Orlikowski, W.J. (1996) Improvising Organizational Transformation Over Time: A Situated Change Perspective, *Information Systems Research*, 7(1), 63-92.

Orlikowski, W.J. and Barley, S.R. (2001) Technology and Institutions: What Can Research on Information Technology and Research on Organizations Learn From Each Other?, *MIS Quarterly*, 25(2), 145-165.

Orlikowski, W.J. and Baroudi, J.J. (1991) Studying information technology in organizations: research approaches and assumptions. *Information Systems Research*, 2(1), 1-28.

Orlikowski, W.J. and Robey, D. (1991) Information Technology and Structuring of Organizations, *Information Systems Research*, 2(2), 143-169.

Patton, M.Q. (1990) *Qualitative Evaluation and Research Methods*, 2nd Edition, Newbury Park, CA: Sage Publications.

Paulk, M.C. (1993) Comparing ISO 9001 and the Capability Model for Software, *Software Quality Journal*, 2, 245-256.

Paulk, M.C. (1995) How ISO9001 Compares with the CMM, *IEEE Software*, 12(1), 74-83.

Paulk, M.C. (1998) Foreward, In: Zahran, S., *Software Process Improvement: Practical Guidelines For Business Success*, xv-xvii, Harlow: Addison-Wesley.

Paulk, M.C., Weber, C.V., Curtis,B., and Chrissis, M.B. (1995) *The Capability Maturity Model: Guidelines For Improving The Software Process*, Reading, MA: Addison-Wesley

Perry, D.E., Staudenmayer, N.A., and Votta, L.G (1994) People, Organizations, and Process Improvement, *IEEE Software*, 11(4), 36-45.

Pettigrew, A.M. (1979) On Studying Organizational Cultures, *Administrative Science Quarterly*, 24, 570-581.

Pettigrew, A.M. (1987) Context and Action in the Transformation of the Firm, *Journal of Management Studies*, 24(6), 649-670.

Pettigrew, A.M. (1990a) Studying Strategic Choice and Strategic Change. A Comment on Mintzberg and Waters: 'Does Decision Get In The Way?', *Organization Studies*, 11(1),6-11.

Pettigrew, A.M. (1990b) Longitudinal Field Research On Change: Theory And Practice, *Organizational Science*, 1(3) 267-292.

Pettigrew, A.M. (1997) What is Processual Analysis? *Scandinavian Journal of Management*, 13(4), 337-348.

Pettigrew, A.M. (2000) Linking Change Processes To Outcomes, In: Beer, M. and Nohria, N. (eds.) *Breaking the Code of Change*, 243-265, Boston, MA: Harvard Business School Press.

Pfeffer, J. (1993) Understanding Power in Organizations, In: Mabey, C. and Mayon-White, B. (eds) *Managing Change 2nd ed*, 201-206, Liverpool: Paul Chapman Publishing / The Open University.

Pfeffer, J. (1995) Producing Sustainable Competitive Advantage Through Effective Management Of People, *Academy of Management Executive*, 9(1), 55-69.

Pourkomeylian, P. (2001) Knowledge Creation in Improving a Software Organisation, In: Ardis, M.A. and Marcolin, B.L. (eds.) *Diffusing Software Products and Process Innovations (IFIP TC8 WG8.6 Fourth Conference)*, 205-224, Norwell, MA: Kluwer Academic Publishers.

Prahlad, C.K. and Hamel, G. (1990) The Core Competence of the Corporation, *Harvard Business Review*, May/June, 79-91.

Pries-Heje, J., Hass, A.M.J. and Johansen, J. (2001) Taking the Temperature on Danish Software Quality, In: Wieczorek, M. and Meyerhoff, D. (eds.) *Software Quality: State Of The Art In Management, Testing And Tools*, 46-60, Berlin, Germany: Springer-Verlag.

Quintas, P. (1994) A Product-Process Model Of Innovation In Software Development, *Journal of Information Technology*, 9(1), 3-17.

Rainer, A. and Hall, T. (2003) A Quantitative and Qualitative Analysis of Factors Affecting Software Processes, *Journal of Systems and Software*, 66(1), 7-21.

Ravichandran, T. and Rai, A. (2000a) Quality Management in Systems Development: an Organizational System Perspective, *MIS Quarterly*, 24(3), 381-415.

Ravichandran, T. and Rai, A. (2000b) Software Process Management: An Organisational Learning Perspective, In: Hansen, H.R., Bichler, M., and Mahrer, H. (eds.) *Proceedings of the 8th European Conference on Information Systems*, 202-209, 3rd-5th July, Vienna, Austria: Vienna University of Economics and Business Administration.

Reason, P. and Rowan, J. (1981) Issues of Validity In New Paradigm Research In: Reason, P. and Rowan, J. (eds.) *Human Inquiry: A Sourcebook Of New Paradigm Research*, 239-252. Chichester: John Wiley & Sons.

Ricoeur, P. (1991) *From Text to Action: Essays in Hermeneutics II*, Evanston: Northwestern University Press (translated by Blamey, K. and Thompson, J.B.)

Rigby, P.J., Stoddart, A.G., and Norris, M.T. (1990) Assuring Quality In Software Practical Experiences In Attaining ISO 9001, *British Telecom Engineering*, 8, 244-249.

Riley, J. (1990) *Getting The Most From Your Data: A Handbook Of Practical Ideas On How To Analyse Qualitative Data*, Bristol: Technical and Educational Services.

Robey, D. (1996) Research Commentary: diversity in information systems research: threat, promise, and responsibility. *Information Systems Research*, 7(4), 400-408.

Rogers, E.M. (1995) *Diffusion of Innovations (4th ed)*, New York, NY: The Free Press.

Scarbrough, H. (1998) Path(ological) Dependency? Core Competencies from an Organizational Perspective, *British Journal of Management*, 9(3), 219-232.

Scarbrough, H. (1999) The Management of Knowledge Workers, In: Currie, W.L. and Galliers, R. (eds.), *Rethinking Management Information Systems*, 474-496, Oxford : Oxford University Press.

Schön, D.A. (1983) *The Reflective Practitioner: How Professionals Think In Action*, New York: Basic Books.

Schutz, A. (1967) *The Phenomenology of the Social World*, Evanston, Ill: Northwestern University Press.

Seale, C. (2000) Using Computers to Analyse Qualitative Data, In: Silverman, D. (2000) *Doing Qualitative Research: a Practical Handbook*, 154-174, London: Sage Publications.

Senge , P. M., (1990) *The Fifth Discipline: the Art and Practice of the Learning Organization*, New York, NY: Doubleday.

Senge, P.M. (2000) The Puzzles And Paradoxes Of How Living Companies Create Wealth: Why Single-Valued Objective Functions Are Not Quite Enough, In: Beer, M. and Nohria, N. (eds.) *Breaking the Code Of Change*, 59-82, Boston, MA: Harvard Business School Press.

Sharp, H. Robinson, H., Woodman, M. (2000) Software Engineering: Community and Culture, *IEEE Software*, 17(1), 40-47.

Silverman, D. (1998) Qualitative research: Meanings Or Practices? *Information Systems Journal*, 8(1), 3-20.

Silverman, D. (2000) *Doing Qualitative Research: a Practical Handbook*, London: Sage Publications

Software Engineering Institute (2003) *Process Maturity Profile: Software CMM CBA IPI and SPA Appraisal Results 2002 Year End Update*, Online at sei.cmu.edu/sema/pdf/SW_CMM/2003apr.pdf accessed 10 Sep 2003.

Somasundaram, S. and Baduri, A.B. (1992) Project Management For Successful Implementation Of Continuous Quality Improvement, *International Journal of Project Management*, 10(2), 89-101.

Sommerville, I. (1996) *Software Engineering (5th edition)*, Addison-Wesley : Harlow.

Stelzer, D., Mellis, W. and Herzwurm, G. (1998) Technology Diffusion In Software Development Processes: The Contribution Of Organisational Learning To Software Process Improvement. In: Larsen, T.J. and McGuire, E.B. (eds.) *Information Systems Innovation And Diffusion: Issues And Directions*, 297-344, Idea Group Publishing, Hershley, PA.

Stienen, H (1999) Software Process Assessment And Improvement: Five Years Of Experiences with BOOTSTRAP. In: El Emam, K. and Madhvji, N.H. (eds.), *Elements of Software Process Assessment And Improvement*, 57-75, Los Alamitos, CA: IEEE Computer Society.

Suchman, L.A. (1987) *Plans and Situated Actions: the Problem Of Human-Machine Communication*, Cambridge University Press, Cambridge.

Swan, J. A., and Newell, S. (2000). Linking Knowledge Management and Innovation, In: Hansen, H.R., Bichler, M., and Mahrer, H. (eds.) *Proceedings of the 8th European Conference on Information Systems*, 591-598, 3rd-5th July, Vienna, Austria: Vienna University of Economics and Business Administration.

Swanson, E.B. (1994) Information Systems Innovation among Organizations, *Management Science*, 40(9), 1069-1092.

Sweeney, A. and Bustard, D.W. (1997) Software Process Improvement: Making It Happen In Practice, *Software Quality Journal*, 6(4), 265-274.

Thompson, J.B. (1981) *Critical Hermeneutics: a Study in the Thought of Paul Ricoeur and Jürgen Habermas*, Cambridge: Cambridge University Press.

Thompson, J.B. (1997) The Theory of Structuration, In: Bryant, C. and Jary, D. (eds.) *Anthony Giddens: Critical Assessment (vol II)*, 310-330, London: Routledge.

Thompson, K. and McParland, P. (1993) Software process maturity (SPM) and the information systems developer, *Information and Software Technology*, 35(6/7), 331-338.

Tjorvold, D. (1993) *Learning to Manage Conflict: Getting People To Work Together Productively*, New York, NY: Lexington Books.

Trauth, E.M. and O'Connor, B. (1990) A Study Of The Interaction Between Information, Technology And Society: An Illustration Of Combined Qualitative Research Methods. In: Nissen, H.E. and Klein, H.K. (eds.) *Proceedings of the IFIP TC8/WG8.2 Working conference on the Information Systems Research Arena of the 90's Challenges, Perceptions and Alternative Approaches (Information Systems Research : Contemporary approaches and emergent traditions)*, 131-144, Amsterdam, Holland: North-Holland.

Truex, D., Baskerville, R. and Klein, H. (1999) Growing Systems in Emergent Organizations, *Communications of the ACM*, 42(6), 117-123.

Truex, D., Baskerville, R. and Travis, J. (2000) Amethodical Systems Development: The Deferred Meaning Of Systems Development Methods, *Accounting, Management, and Information Technology*, 10, 53-79.

Van Solingen, R. (2004) Measuring the ROI of Software Process Improvement, *IEEE Software*, 21(3), 32-38.

Van Solingen, R., Berghout, E. Kusters, R. and Trienekens, J. (2000) From Process Improvement to People Improvement: Enabling Learning in Software Development, *Information and Software Technology*, 42 (14), 965-971.

Van Solingen, R., Kusters, R.J., Trienekens, J.J.M, Van Uijtregt, A (1999) Product-Focused software Process Improvement (P-SPI): Concepts And Their Application, *Quality and Relieability Engineering International*, 15, 475-483.

Wallmüller, E. (1991) Software Quality Management, *Microprocessing and Microprogramming*, 32, 609-616.

Walsham, G. (1993) *Interpreting Information Systems in Organizations*, Chichester: John Wiley & Sons.

Walsham, G. (1995) Interpretive Case Studies in IS Research: Nature and Method, *European Journal of Information Systems*, 4(2), 74-81.

Watson, T.J. (1994) *In Search of Management*, London: Thomson Business Press.

Weick, K.E. (1995) *Sensemaking in Organizations*, Thousand Oaks, CA: Sage Publications Inc.

Weick, K.E. (2000) Emergent Change as a Universal in organizations, In: Beer, M. and Nohria, N. (eds.) *Breaking the code of change*, 223-242, Boston, MA: Harvard Business School Press.

Weick, K.E. (2001) *Making Sense of the Organization*, Blackwell Publishers: Oxford.

Willcocks, L. Lacity, M. and Fitzgerald, G. (1997) IT Outsourcing in Europe and the US, In: Willcocks, L., Feeny, D. and Islei, G. (eds) *Mangaing IT as a Strategic Resource*, 306-338, Maidenhead: McGraw-Hill.

Yin, R.K. (1994) *Case Study Research: Design and Methods (2nd ed.)* Thousand Oaks, CA: Sage Publications, Inc.

Zahran, S. (1994) The Software Process – What Is It, and How To Improve It, In: Ross, M., Brebbia, C.A., Staples, G. and Stapleton, J (eds.) *Software Quality Management II vol 1: Managing Quality Systems*, 215-231, Southampton: Computational Mechanics Publications.

Zahran,S. (1998) *Software Process Improvement: Practical Guidelines for Business Success*, Harlow : Addison-Wesley.

Zhiying, Z. (2003) CMM in Uncertain Environments, *Communications of the ACM*, 46(8), 115-119.

# Appendix 1 List of Interviews and Reviews

## *List of Interviews*

| Reference Number | Date | Role |
|---|---|---|
| 1 | 19/10/98 | Director |
| 2 | 26/10/98 | Other IS |
| 3 | 26/10/98 | Other non IS |
| 4 | 03/11/98 | Other IS |
| 5 | 05/11/98 | Other IS |
| 6 | 09/11/98 | Director |
| 7 | 16/11/98 | Developer |
| 8 | 02/12/98 | Developer |
| 9 | 12/01/99 | Developer |
| 10 | 14/01/99 | Developer |
| 11 | 20/01/99 | Developer |
| 12 | 20/01/99 | Developer |
| 13 | 27/01/99 | Developer |
| 14 | 10/02/99 | Developer |
| 15 | 16/03/99 | Developer |
| 16 | 17/03/99 | Developer |
| 17 | 19/03/99 | Developer |
| 18 | 23/03/99 | Developer |
| 19 | 28/04/99 | Developer |
| 20 | 28/04/99 | Developer |
| 21 | 15/06/99 | Director |
| 22 | 15/06/99 | Director |
| 23 | 16/06/99 | Non-divisional |
| 24 | 16/06/99 | Non-divisional |
| 25 | 25/06/99 | Developer |
| 26 | 13/07/99 | Director |
| 27 | 23/06/99 | Other non IS |
| 28 | 24/11/98 | Other non IS |
| 29 | 21/06/99 | Other non IS |
| 30 | 23/06/99 | Other non IS |
| 31 | 17/08/99 | Other non-IS |

## *List of Reviews*

| Reference Number | Date | Name | Role |
|---|---|---|---|
| 1 | 30/10/98 | Bridges | Project Manager |
| 2 | 02/11/98 | Tyler | Software Director |
| 3 | 06/11/98 | Tyler | Software Director |
| 4 | 06/11/98 | Bridges | Project Manager |
| 5 | 17/11/99 | Bridges | Project Manager |
| 6 | 20/11/98 | Tyler | Software Director |
| 7 | 20/11/98 | Bridges | Project Manager |
| 8 | 04/12/98 | Bridges | Project Manager |
| 9 | 01/12/98 | Bridges | Project Manager |
| 10 | 15/01/99 | Bridges | Project Manager |
| 11 | 22/01/99 | Bridges | Project Manager |
| 12 | 25/01/99 | Tyler | Software Director |
| 13 | 29/01/99 | Bridges | Project Manager |
| 14 | 01/02/99 | Tyler | Software Director |
| 15 | 05/02/99 | Bridges | Project Manager |
| 16 | 12/02/99 | Tyler | Software Director |
| 17 | 26/02/99 | Bridges | Project Manager |
| 18 | 01/03/99 | Tyler | Software Director |
| 19 | 05/03/99 | Bridges | Project Manager |
| 20 | 15/03/99 | Tyler | Software Director |
| 21 | 26/03/99 | Bridges | Project Manager |
| 22 | 26/03/99 | Tyler | Software Director |
| 23 | 16/04/99 | Bridges | Project Manager |
| 24 | 07/05/99 | Bridges | Project Manager |
| 25 | 27/05/99 | Tyler | Software Director |
| 26 | 27/05/99 | Bridges | Project Manager |
| 27 | 04/06/99 | Bridges | Project Manager |

# Appendix 2 Interview Questions

## *Team Member Interview*

**Please describe your role in the team.**

- Thinking about your role, what helps you to do it well? What problems do you face in doing you job?
- What have been the main lessons you learned from your involvement with the project?

**Company**

- How have the products developed since you arrived?
- How would you to describe the management style of the different people in the hierarchy?
- Tell me what is important for you in the way the Division / team is managed?
- What are the strengths and weaknesses in the current team?
- How do ideas get communicated around the organisation?

**Process Improvement**

- What lessons did you learn from the MAP project?
- How have the processes changed?
- What is the single most significant challenge in improving quality?
- How has the process improvement initiative affected the work of the team?
- How have people changed in their approach / attitude to software development since this initiative?
- What future development actions do you foresee?

## ***Management Interview***

**Tell me about yourself / background**

- What is your role?
- What is your background?
- What is your management style?

**What is the Divisional strategy / business**

- Competitive situation
- Innovation
- Research & development
- Customers
- Business Strategy – process / how it impacts the development
- When do you decide to look externally for a solution / component?
- How do vendors get chosen?

**Explain how the division is managed**

- Explain the thinking behind current organisational structure
- Describe the way the division is managed
- How do managers arrive at important product decisions?
- Explain how the MAP team is managed
- Does there tend to be agreement over key issues between the managers and the team?
- Describe the operation of the team
- What changes are you anticipating for the team?
- How would you describe the skill level of the different teams / people?

**Tell me your views of the MAP project & current development:**

- people & relationships
- product development
- software quality

**Software Quality**

- Think back to the problems occurring after the MAP development are these still there / how have they been resolved? What are the main problems now?
- How has the process changed?
- Why do you think it was necessary to change the processes?

## ***Review checklist***

**Key Events** that have occurred in the last period (internal or external) such as

Key decisions about the process/product,

Meetings with directors/ supplier companies / customers

**Business initiatives**

Changes to the strategy

Initiatives in the work / systems

**Quality issues**

Problems & improvements

Diffusion of the ideas / sharing of ideas between teams

**Staff issues**

Personnel / team issues, e.g. skills, motivation, personal problems, constraints

General environment – this might be anything which has an impact on the team from the world cup to a major snow fall keeps everyone at home - however, only note these if it affects things significantly

**Software / technology**

New releases

Upgrades in technology

# Appendix 3 Summary of Questionnaire Data

The data given below is the mean of the twenty-five responses from across the GeoMarketing development team, and so extended beyond the MAP team. The questionnaire was targeted at those directly involved in the development and maintenance of software products within the GeoMarketing Division. Twenty-five people responded to the questionnaire. A sample of twenty-five is easily skewed by a few responses, so when talking about a single team within the whole group this is all the more likely. However this thesis only makes light use of the quantitative data, and used the information to inform the interviews.

Most people were educated to University level with over a third holding postgraduate qualifications. Two-thirds of people consider that their qualification is relevant to their job. The MAP development team members were more experienced in software development than the other teams. The MAP team felt that the experience within the team had improved over the last year in the areas of reviews and inspections, the programming language, and the methodology. To a lesser extent the project management and application experience has also improved. Despite these improvements, the team still felt that they did not have the necessary knowledge and experience of the methodology.

One of the issues that came out from the second part of the questionnaire was that training was not well planned and was often the casualty of tight schedules. A number of people suggested that what was required was a 'clear development plan' by 'positively identifying who needs training and ensuring it is given where necessary'. It was clearly a concern to some that they were making mistakes or not doing things as a

result of an ignorance of what should be done. On average approximately 3 days of external training was actually provided per person. This average is rather misleading though, as the quantity of training varied from nothing to several weeks.

Another aspect to do with training was ensuring that it was appropriate for the needs of the team. One suggestion was to evaluate the courses. Another idea was for the trained staff to spread any useful knowledge from external courses through a localised form of the training. In a similar vein the idea was made to use customised courses, more appropriate to the needs of the work and staff here. Some suggested a 'refresher course' would be a good idea but a refresher in what was not clear.

This idea of better training within the team came up in several different forms. One suggestion was for better induction training. This should accelerate their learning and therefore their performance on the team. The next stage would be to move into a continuous learning cycle as described above. Suggestions to achieve this included mentoring and 'on the job training' which could fit neatly together, with more experienced members of staff coaching others. Many people felt that they did not know enough about the work of others around them and in particular the Division's own products. It appears that more knowledge of how the products are used by clients and the internal workings of the products would be significant in improving the interest in, and quality of, the work performed. However this was not unanimous by any means as some 'could not become interested in the product [which] makes it difficult to understand the big picture'.

The overall ranking of the factors is shown below. In addition to the motivational factors listed on the questionnaire, satisfaction at doing the job well was mentioned by

several respondents as an important motivator. A strong feeling was that good wages were not provided by the organisation and the company should pay a 'more realistic wage for [the] job'. As well as financial reward, people mentioned that other forms of recognition were a motivator to do the job better and 'some idea of where my career is going' would help to give a stronger sense of direction as well as a form of recognition in its own right.

| Motivational factor (in rank order) | Average Ranking (1=important) | How well organisation achieves this need (1=very well;5 not at all) |
|---|---|---|
| Interesting Work | 2.3 | 2.3 |
| Good Wages | 3.2 | 3.5 |
| Promotion and personal growth | 3.7 | 3.1 |
| Appreciation for work done | 3.9 | 3.1 |
| Good working conditions | 4.0 | 2.2 |
| Job Security | 4.6 | 2.0 |
| Feeling of 'being in on things' | 5.6 | 3.4 |
| Personal loyalty to employees | 6.5 | 3.1 |
| Tactful discipline | 7.8 | 2.8 |
| Sympathetic understanding and help | 8.2 | 2.5 |

Overall the questionnaires were consistently favourable about the team leaders. The political and technical strengths complementing their concern for staff development

and teamwork. The desire to get the work done on time was by far the strongest indicator, but whether this is seen as a 'good' thing is debatable. There appears to be a stronger sense of working as a team in the Bespoke area, perhaps simply as a result of size and organisational factors such as location. Communication may be another factor that is affected by size and other organisational factors. The smaller Bespoke and Testing teams were more positive that communication was clear and frequent, than the MAP development team who felt this had actually become worse in the last year. The reason for this was not clear from the questionnaire. Some felt that being better informed would increase motivation. However the team were more aware of the current plans than they were even though they felt less involved in the planning activity.

Project planning was considered to be more effective in the MAP team than a year ago. Perhaps the reason for this is that the project plans are maintained better than they were and more information is kept on how long is actually spent on tasks. By contrast the estimation of elapse time and staff effort is not considered effective, and indeed the recording of actual effort and maintenance of the plan still require improvement.

Two significant improvements were perceived to have occurred: the introduction of the testing tool and the wider use of reviews and inspections. Whether as a result of the review process or for other reasons, they also felt that requirements were captured more successfully than a year ago. And whilst test status was not well communicated, knowledge of outstanding defects prior to the release of software was better now. Also, a number of minor improvements were reflected throughout the area of tools and metrics, but these areas were still immature. The area that was of most concern

was externally produced component software. It was considered to be of poor quality by the MAP team, and should be written internally.

Overall the changes made by the MAP team in the last year can be said to be a step in the right direction but often counteracted by other factors. The Bespoke team, whilst clearly adopting a less engineered approach, feel satisfied that this is successful. How this view will change in the world of component based development remains to be seen.

| | (1=strong agree; 5=strongly disagree) | 1997 | 1998 | Difference |
|---|---|---|---|---|
| | **Team Background** | | | |
| 1 | The project team has relevant methodology experience | 3.92 | 3.47 | 0.45 |
| 2 | The team has strong programming language experience | 2.64 | 2.00 | 0.64 |
| 3 | The team has appropriate application experience | 2.92 | 2.56 | 0.37 |
| 4 | There is a wealth of project management experience | 3.57 | 3.11 | 0.47 |
| 5 | There is strong agreement between the team and management | 3.08 | 2.78 | 0.30 |
| 6 | There is thorough software engineering training for staff | 3.57 | 3.47 | 0.10 |
| 7 | Team members have experience of reviews and inspections | 3.29 | 2.47 | 0.81 |
| 8 | There is a clear training plan for team members | 4.08 | 3.89 | 0.19 |
| | **Teamwork** | | | |
| 9 | Roles and responsibilities are clearly defined | 2.79 | 2.58 | 0.21 |
| 10 | There is clear and frequent communication in the team | 3.43 | 3.11 | 0.32 |
| 11 | Feedback is regularly sought and accepted | 3.79 | 3.37 | 0.42 |
| 12 | Team performance is emphasised over individual performance | 3.29 | 3.11 | 0.17 |
| 13 | Organisational reward structure supports teamwork | 3.62 | 3.44 | 0.17 |
| 14 | There is a strong sense of a | 3.00 | 2.84 | 0.16 |

| | | | | |
|---|---|---|---|---|
| | common purpose in the team | | | |
| 15 | Teamwork skills training is available | 3.71 | 3.79 | -0.08 |
| | **Management of changes in work practice** | | | |
| 16 | The organisation communicates the goals of any change | 3.54 | 3.22 | 0.32 |
| 17 | Negotiation is used to obtain agreement and co-operation | 3.00 | 3.00 | 0.00 |
| 18 | Staff behaviour which supports the change is rewarded | 3.46 | 3.28 | 0.18 |
| 19 | Resources are provided/redirected to support change | 3.00 | 3.12 | -0.12 |
| 20 | Participation and joint decision making encouraged | 2.82 | 2.88 | -0.06 |
| | **Team leadership (immediate team leaders plus Chris S)** | | | |
| 21 | The team leader(s) set clear goals | 2.83 | 2.47 | 0.36 |
| 22 | The team leader(s) are politically strong | 2.75 | 2.59 | 0.16 |
| 23 | The team leader(s) are technically strong | 2.00 | 1.88 | 0.12 |
| 24 | The team leader(s) are focused on process | 2.82 | 2.50 | 0.32 |
| 25 | The team leader(s) serve as role models | 3.08 | 2.82 | 0.26 |
| 26 | The team leader(s) understand people and build relationships | 2.58 | 2.41 | 0.17 |
| 27 | The team leader(s) are concerned about my development | 2.67 | 2.47 | 0.20 |
| 28 | The team leader(s) organise work according to staff capabilities | 2.67 | 2.47 | 0.20 |
| 29 | The team leader(s) are concerned to get the work done on time | 1.92 | 1.82 | 0.09 |
| 30 | The team leader(s) encourage quality work | 2.73 | 2.38 | 0.35 |
| | **Quality focus** | | | |
| 31 | Quality assurance aspects are planned for each release | 3.00 | 3.00 | 0.00 |
| 32 | Test status is communicated to the whole team | 3.93 | 3.84 | 0.09 |
| 33 | Project post-mortems help to learn lessons for the future | 3.57 | 3.42 | 0.15 |
| 34 | The distribution of defects through the system is known | 3.29 | 3.05 | 0.23 |
| 35 | Operation of the testing function is effective | 3.36 | 3.05 | 0.30 |
| 36 | Quality assurance is effective throughout the process | 3.57 | 3.16 | 0.41 |
| 37 | Status of project quality is reported | 3.75 | 3.53 | 0.22 |

| | | | | |
|---|---|---|---|---|
| 38 | Defects are known prior to each release | 2.71 | 2.47 | 0.24 |
| 39 | The root cause of problems are analysed and tackled | 3.57 | 3.00 | 0.57 |
| 40 | Error prone modules are analysed and tackled | 3.57 | 3.21 | 0.36 |
| 41 | Creativity / improvement initiatives are supported | 3.31 | 3.00 | 0.31 |
| 42 | **Customer focus (these may be internal or external)** | | | |
| 43 | Customers are well supported in their use of the product | 2.00 | 2.06 | -0.06 |
| 44 | Customer defects reported and acted upon | 2.92 | 2.39 | 0.53 |
| 45 | Customer requests for enhancement are documented | 2.50 | 2.44 | 0.06 |
| 46 | Customer satisfaction is reported and acted upon | 3.09 | 2.82 | 0.27 |
| 47 | Customers are involved during acceptance testing | 4.20 | 3.75 | 0.45 |
| 48 | Customers are satisfied with the documentation | 3.67 | 3.47 | 0.20 |
| 49 | Customers are involved during customer documentation | 4.00 | 3.71 | 0.29 |
| 50 | **Other factors external to the team** | | | |
| 52 | Externally produced component software is of high quality | 4.23 | 4.39 | -0.16 |
| 53 | Externally produced component software is value for money | 3.42 | 3.75 | -0.33 |
| 54 | External component software reduces our development time | 3.00 | 3.35 | -0.35 |
| 55 | Standard market software is of high quality (e.g. MS products) | 2.85 | 2.88 | -0.04 |
| 56 | Standard market software suppliers give value for money | 3.08 | 3.06 | 0.01 |
| 57 | Standard market products support our development activities | 3.00 | 3.00 | 0.00 |
| 58 | Changes in underlying technology are planned in the scope | 2.83 | 2.93 | -0.10 |
| 59 | Directors understand the work of the team | 3.15 | 3.17 | -0.01 |
| 60 | Directors support the work of the team | 2.92 | 2.78 | 0.15 |
| 61 | Directors interfere with the project's work | 2.93 | 3.39 | -0.46 |
| 62 | New ideas for the product are planned into the scope | 2.78 | 2.79 | -0.01 |
| | Pressure from customers disrupt the work in the team | 2.93 | 3.11 | -0.18 |

| | | | | |
|---|---|---|---|---|
| | Changes in competitor products | | | |
| 63 | force changes in our plans | 3.64 | 3.50 | 0.14 |
| | Internationalisation of the product | | | |
| 64 | adds complexity | 2.21 | 2.47 | -0.26 |
| | Internationalisation is planned as | | | |
| 65 | part of the development | 2.93 | 2.76 | 0.16 |
| | **Methodology / techniques** | | | |
| | Requirements are successfully | | | |
| 66 | captured in the analysis stage | 3.43 | 2.94 | 0.48 |
| 67 | The software is well designed | 3.21 | 3.17 | 0.05 |
| | Prototyping is used where | | | |
| 68 | appropriate to validate the design | 3.00 | 3.17 | -0.17 |
| | Software development produces | | | |
| 69 | working code | 2.36 | 2.33 | 0.02 |
| | System test planning & testing | | | |
| 70 | removes main bugs | 2.86 | 2.68 | 0.17 |
| 71 | Software release is well controlled | 2.86 | 2.79 | 0.07 |
| | Design / code is reused whenever | | | |
| 72 | possible | 4.00 | 3.69 | 0.31 |
| | Existing design / code is | | | |
| 73 | reengineered for improvement | 3.64 | 3.40 | 0.24 |
| | **Project planning** | | | |
| | Project planning is undertaken | | | |
| 74 | effectively | 3.46 | 2.82 | 0.64 |
| | Actual versus estimated schedule is | | | |
| 75 | maintained | 4.17 | 3.56 | 0.60 |
| | Milestone and progress reviews | | | |
| 76 | occur frequently | 3.62 | 3.67 | -0.05 |
| | Software estimation methods and | | | |
| 77 | tools are used | 4.15 | 4.06 | 0.10 |
| | Schedule estimation process is | | | |
| 78 | effective (elapse time) | 4.17 | 4.00 | 0.17 |
| | Effort estimation process is | | | |
| 79 | effective (staff time) | 4.00 | 3.63 | 0.38 |
| | Past project data is used to plan | | | |
| 80 | new tasks | 4.23 | 3.88 | 0.35 |
| | I feel involved in the planning | | | |
| 81 | activity | 3.38 | 3.47 | -0.09 |
| | I know what the current plans are | | | |
| 82 | for my team | 2.50 | 2.41 | 0.09 |
| | The review/inspections are | | | |
| 83 | scheduled | 3.25 | 2.75 | 0.50 |
| | Project management activities are | | | |
| 84 | audited | 4.11 | 3.77 | 0.34 |
| | The scope for releases is well | | | |
| 85 | managed | 3.56 | 3.31 | 0.25 |
| | The product strategy (future) is | | | |
| 86 | clear | 3.27 | 3.00 | 0.27 |
| | **Process focus** | | | |

| | | | | |
|---|---|---|---|---|
| 87 | Software processes are defined | 2.92 | 2.65 | 0.28 |
| 88 | Software processes are frequently monitored | 3.92 | 3.47 | 0.45 |
| 89 | Software processes are assessed | 3.85 | 3.65 | 0.20 |
| 90 | Software processes reflect organisational objectives | 3.36 | 3.20 | 0.16 |
| 91 | Software process practices reflect state-of-the-art | 3.91 | 3.73 | 0.18 |
| 92 | **Metrics captured** | | | |
| 93 | Actual effort taken is known for each task | 4.00 | 3.44 | 0.56 |
| 94 | Task size measured (e.g. function points/lines of code) | 4.36 | 4.06 | 0.30 |
| 95 | The level of complexity for the software is measured | 4.00 | 3.83 | 0.17 |
| 96 | Review/inspection defect removal efficiency is known | 4.14 | 3.72 | 0.42 |
| 97 | Aspects of the code / design (e.g. levels of inheritance, calls) | 4.27 | 4.21 | 0.06 |
| 98 | Defect quantity / severity | 3.64 | 3.29 | 0.35 |
| | **Use of Tools (for these 1 = excellent use; 5 = poor / no use)** | | | |
| 99 | Software analysis & design tools | 4.18 | 4.07 | 0.11 |
| 100 | Program debugging tools | 2.27 | 2.29 | -0.01 |
| 101 | Data dictionary | 4.29 | 4.20 | 0.09 |
| 102 | Internal documentation tool | 3.91 | 3.71 | 0.19 |
| 103 | Source code library | 2.55 | 2.71 | -0.17 |
| 104 | Testing and QA tools | 3.36 | 2.56 | 0.80 |
| 105 | Project management software | 3.64 | 3.57 | 0.06 |

# Appendix 4 Database of case data

| Category of material | Type of data | Storage |
|---|---|---|
| Case study field notes | Journals | Containing notes and observations from over the year, including meetings attended and general activity. Chronologically in hard bound notebook |
| | Interviews | 29 Interviews and 26 unstructured review discussions with team managers. By date order within category of people, with index to tape. Tapes stored in numeric sequence. |
| | Team SPI questionnaires | By respondent order, with summary |
| Documents | Development documents data | Samples of analysis and design documentation |
| | | Defect ("bug" or "SRF") data on database |

| | | |
|---|---|---|
| | | Helpdesk call data on database |
| | | Defect / helpdesk summary data |
| | | Memos, briefings, finance, etc by date |
| | Quality Management /SPI documents | SPI action teams |
| | | Documents and notes by process improvement action area |
| | | Quality manual |
| | | Corporate SPI project memos and documents |
| | Minutes | Team Leaders' meetings by date |
| | | Bug Prioritisation meetings by date |

| | | |
|---|---|---|
| | Detailed planning and effort data over 2 years | Strategic plans by date<br><br>Development plans by date from version 1.<br><br>Work breakdown structures by sub-product for MAP version 2<br><br>Gantt charts by date for MAP version 2 |
| | Press and organisational publications<br><br>Internal product publications | Pamphlets etc stored in wallet files |
| | Press cuttings | Categorised into:<br><br>Organisational level<br><br>Competition / the business environment for GeoMarketing<br><br>General materials on |

| | | |
|---|---|---|
| | | InfoServ business (not GeoMarketing) |
| Analysis notes | Index action lists<br>Codes<br>Contact summary<br>Event summary<br>Period in diary summary<br>Time based data<br>Concept notes<br>Working papers | Organised by type of item / date of original |

# Appendix 5 Data Summary Sheets

## Contact Summary

1. What were the main issues that struck you with this contact?

2. Summarise information you got
   History of the products / business

   Context – major topics

   Changes in the process

   Factors that influenced quality of software / development of product

   Factors that influenced the process improvement

3. Anything else that struck you as salient /important?

4. What questions / speculations / theories does this raise?

5. Any follow up required?

## Document Form

Name / description of document:

Event associated with :

Significance / importance of the document

Key issues / salient aspects / questions raised

**Period in Log Summary**

Log number / ref
Dates:

What were the main issues that struck you with this period?

Summarise information you got

History of the products / business

Context – major topics

Changes in the process

Factors that influenced quality of software / development of products

Factors that influenced the process improvement

Anything else that struck you as salient /important?

What questions / speculations / theories does this raise?

**Event Summary**

Name / description of event:

Location in documentation

Associated with :

Relevant Documents

Significance / importance of the event

Key issues / salient aspects / questions raised

## Appendix 6 Coding Structure Used for Data Analysis

| Theme | Code |
|---|---|
| **Context** | |
| **Organisation context** | |
| Infrastructure | ORG-struct |
| Hierarchy | ORG-struct-hier |
| Authority relations | ORG-struct-auth |
| Incentive Systems | ORG-struct-pay |
| History | ORG-hist |
| Products (general) | ORG-prod |
| **Company environment** | ORG-env |
| Suppliers | ORG-env-supp |
| Customers | ORG-env-cust |
| Other parts of organisation | ORG-env-co |
| Organisational management (Strategies, plans and leadership) | ORG-mgt |
| Norms and policies | ORG-norm |
| Work culture and relations | ORG-cult |
| **IS function context** | |
| Software Development Norms, Techniques and Standards | IS-norms |
| Social relations | IS-social |
| Knowledge and experience | IS-know |
| Individual people characteristics | IS-people |

| | |
|---|---|
| SPI context | |
| SPI project approach | SPI-project |
| SPI knowledge and experience | SPI-know |
| Commitment | SPI-commit |
| Company SPI context | SPI-co |
| Software Professional Environment | SPI-prof |
| Action and process | |
| **Action of software developers to:** | ACT-dev |
| propose/introduce product innovation | ACT-dev-introprod |
| propose/introduce process innovation | ACT-dev-introproc |
| propose/introduce SPI project | ACT-dev-introSPI |
| realise product innovation | ACT-dev-realprod |
| realise process innovation | ACT-dev-realproc |
| realise SPI project | ACT-dev-realSPI |
| not realise product innovation | ACT-dev-notprod |
| not realise process innovation | ACT-dev-notproc |
| not realise SPI project | ACT-dev-notSPI |
| **Action of IS management to** | ACT-ISmgt |
| propose/introduce product innovation | ACT- ISmgt -introprod |
| propose/introduce process innovation | ACT-ISmgt-introproc |
| propose/introduce SPI project | ACT-ISmgt-introSPI |
| realise product innovation | ACT-ISmgt-realprod |
| realise process innovation | ACT-ISmgt-realproc |
| realise SPI project | ACT-ISmgt-realSPI |
| not realise product innovation | ACT-ISmgt-notprod |

| | |
|---|---|
| not realise process innovation | ACT-ISmgt-notproc |
| not realise SPI project | ACT-ISmgt-notSPI |
| **Action of company management to** | ACT-COmgt |
| propose introduce product innovation | ACT- COmgt -introprod |
| propose introduce process innovation | ACT-COmgt-introproc |
| propose introduce SPI project | ACT-COmgt-introSPI |
| realise product innovation | ACT-COmgt-realprod |
| realise process innovation | ACT-COmgt-realproc |
| realise SPI project | ACT-COmgt-realSPI |
| not realise product innovation | ACT-COmgt-notprod |
| not realise process innovation | ACT-COmgt-notproc |
| not realise SPI project | ACT-COmgt-notSPI |
| **Action of other company specialists:** | ACT-Others |
| propose introduce product innovation | ACT- Others -introprod |
| propose introduce process innovation | ACT-Others-introproc |
| propose introduce SPI project | ACT-Others-introSPI |
| realise product innovation | ACT-Others-realprod |
| realise process innovation | ACT-Others-realproc |
| realise SPI project | ACT-Others-realSPI |
| not realise product innovation | ACT-Others-notprod |
| not realise process innovation | ACT-Others-notproc |
| not realise SPI project | ACT-Others-notSPI |
| Cultures sub-cultures communities of practice | Proc-culture |
| Political process control, authority | Proc-polities |

| Multi-meanings | Proc-meanings |
|---|---|
| **Structural Process** | |
| Unanticipated outcomes from the change to process | Unant-proc |
| Unanticipated outcomes from the change to process | Unant-prod |
| The practices that enacted the changes in the process. | Enact |
| Specific software engineering features appropriated in use | SEng-feature |
| Process models and methods as frameworks for learning | Model-learn |
| Action and structure duality | Duality |
| Reflection-in-action | Reflection |
| Dynamic nature of development | Dynamic |

## Appendix 7 Assessment of Process Capability by Time

| Process Areas | 1995 | 1997 | 1999 |
|---|---|---|---|
| **Level 2 : Managed** | | | |
| Requirements Management | Not implemented | Defined | Implemented |
| Project Planning | Informal | Defined | Implemented |
| Project Monitoring & Control | Not implemented | Informal | Informal |
| Supplier Agreement Management | Not implemented | Not implemented | Informal |
| Measurement & Analysis | Not implemented | Not implemented | Not implemented |
| Process & Product Quality Assurance | Defined | Defined | Implemented |
| Configuration Management | Not implemented | Implemented | Implemented |
| **Level 3 : Defined** | | | |
| Requirements development | Informal | Implemented | Implemented |
| Technical Solution | Defined | Implemented | Implemented |
| Product Integration | Informal | Defined | Implemented |
| Verification | Defined | Defined | Implemented |
| Validation | Informal | Implemented | Implemented |
| Organisation Process Focus | Defined | Implemented | Implemented |
| Organisational Process Definition | Informal | Informal | Informal |
| Organisational Training | Informal | Informal | Informal |
| Integrated project Management | Not implemented | Not implemented | Not implemented |
| Integrated Teaming | Informal | Defined | Implemented |
| Risk Management | Defined | Defined | Defined |
| Decision Analysis & Resolution | Not implemented | Not implemented | Not implemented |
| Organisational Environment for Integration | Not implemented | Not implemented | Not implemented |

The definitions of the judgements are:

| | |
|---|---|
| Not implemented | No consistent attempt to implement this process area |
| Informally implemented | Suitable methods were applied in this process area, but these were implemented in an ad hoc or individual fashion |
| Defined but not always implemented | Methods were documented in the process definition manual, but only partially implemented |
| Implemented | Methods were consistently implemented across the team, even if not fully documented |