Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 49 (2007) 668–681
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Software process improvement as emergent change: A structurational analysis

I. Allison [a,*], Y. Merali [b]

[a] School of Computing and Informatics, Nottingham Trent University, NG11 8NS, UK
[b] Warwick Business School, University of Warwick, CV4 7AL, UK

Available online 11 February 2007

Abstract

This paper presents a framework that draws on Structuration theory and dialectical hermeneutics to explicate the dynamics of software process improvement (SPI) in a packaged software organisation. Adding to the growing body of qualitative research, this approach overcomes some of the criticisms of interpretive studies, especially the need for the research to be reflexive in nature.

Our longitudinal analysis of the case study shows SPI to be an emergent rather than a 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 unfolding/realisation of the process improvement (intent) are enabled and constrained by their context. The work builds on the 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 influence the improvement through facilitating or resisting the changes. 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.

Keywords: Software process improvement; Software quality; Software package development; Structuration theory

1. Introduction

Software process improvement (SPI) facilitates the identification and application of changes to the development and management activities in order to improve the product. Perry et al. [26] show that without understanding the technological, social and organisational aspects of software development we cannot hope to significantly improve processes. This work therefore extends the existing literature by investigating the effect of contextual and social factors on the changes in software processes as they are enacted. An explanatory theory of the SPI change process is developed from the experiences of a specific software package organisation over a 10-year period providing an understanding of how and why software process improvements occur and what the consequences of the change process are within this specific case. This understanding is based upon a structurational perspective that highlights how the process improvements are enabled and constrained by their context. An emergent view of software process change helps to understand the way the actions intertwine to inform each other, and shape and are shaped by the context they are in. The outcomes of software products and processes emerge from this intertwining.

The paper begins by showing that in contrast to the software engineering literature that tends to be restricted to a rational, deterministic view of change, the on-going nature of the software processes needs to be placed at the heart of the analysis: opening up the facets of the change not taking it as a given. To support this analysis a theoretical framework is outlined; the framework is drawn from a combination of the case study data and elements of the existing literature. The qualitative methods adopted in this study and the case study methodology are explained to enable the reader to appreciate the basis of the findings.

* Corresponding author. Tel.: +44 115 8488357.
E-mail address: ian.allison@ntu.ac.uk (I. Allison).

0950-5849/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2007.02.003

The core of the paper is a chronological analysis of the case. The case analysis accentuates 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. This narrative adopts the structure of the theoretical framework for the case narrative, and evaluates the outcomes of the process changes.

Finally, lessons are drawn from the study in three areas. To inform SPI theory we highlight the importance of incorporating an understanding of the complexities of the dynamics that occur as software processes emerge over time. For software engineering practice, lessons are learnt by considering how SPI programmes would benefit from recognising the emergent nature of the improvement, and that more fully acknowledging how the processes can support the business objectives would help to focus the improvements. And then, for other qualitative researchers, we highlight lessons learnt from our approach to this study with respect to improving the relevance of such studies for software practice.

2. Software process improvement research

Much of the current understanding of software process improvement has been derived from the work of the Software Engineering Institute. To support improvement programmes, the software engineering community has developed a set of normative maturity models for organisations to follow and enable the assessment of current capability. Within such norm-based models, improvement in the software process is considered to result in the maturing of the activities undertaken by a software development group [17]. Evidence shows that benefits can be achieved as a result of this adoption [14]. Consequently, the majority of the work to date has concentrated on developing such models [13].

The normative models, though, are criticised for the rigidity of the pre-defined actions and their underlying deterministic assumptions about implementation [6]; and for their inflexibility and the emphasis of technology rather than people [21]. Subsequently, not all companies have found software process improvement to be beneficial with many abandoning SPI programmes [14]. Some leading commercial software producers do not, therefore, follow software process improvement according to the maturity models or quality standards [9]. However, the software engineering literature has tended to assume that an ability to instigate, plan and direct all forms of change can be taken for granted, creating 'the illusion of manageability' [16, p. 6]. Indeed, 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.

A holistic approach is necessary to study all the relevant factors in a given software context [26]. What is required therefore is an integrative theory, building on research that examines the enabling and constraining factors on quality management practice [29]: an understanding of change that reflects a more complex, dynamic and unpredictable world. Yet, SPI research does not pay enough attention to the organisational factors that enable or constrain the process improvements. So, it is appropriate to examine whether organisational issues arise as software development groups move towards a more structured, process-oriented environment. In summary the main questions of this study are:

- How does the software process improvement initiative unfold within the context of a packaged software organisation, and how does this compare to stated intent?
- 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?
- In relation to adopting process innovations, how and why do the behaviours of individuals affect the dynamics of the software practice?

3. SPI as situated change: a structurational perspective

In order to highlight the emergence, interplay and outcomes occurring from the software practice a contextualist and processual perspective is adopted. Taking a processual view enables us to attend to the unfolding interplay between the espoused process improvement logic and the perceived outcomes of the realised process.

A framework was devised to enable case analysis to be more attentive to the emergent nature of the SPI activity. 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. It rests on a combination of the case study data and elements of the existing literature. 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 intended to be 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, but these concepts can sensitise the analysis of a software process improvement initiative.

A central theme of the framework is the way that process improvement interweaves with product development, within their specific historical context (Fig. 1). The data analysis therefore includes this view of practice, 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

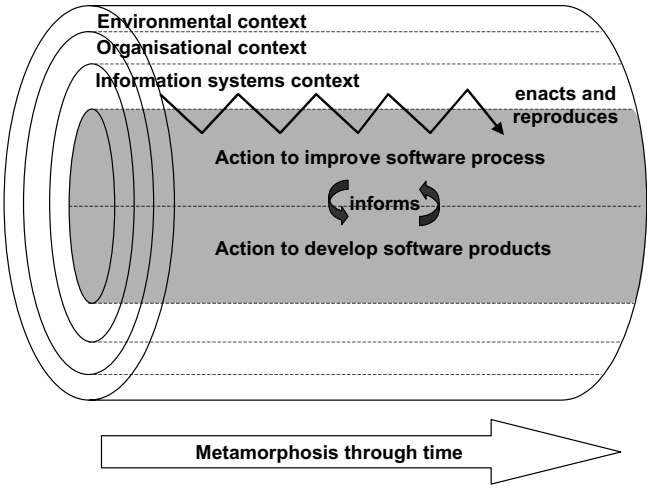

Fig. 1. An emergent view of software process improvement.

within its organisational context. 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 [28]. Each aspect is briefly discussed below and summarised in Table 1 (see [3] for details).

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 [25,31]. Here these layers are not seen as entirely separate entities, as the edges are blurred, rather as a way of illuminating the different aspects of the context. Additionally, the formative context for the changes comprises the organisational and personal history, and the actors' experience relevant to the software process. 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.

The content of change 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 they are normally considered as separate.

Process improvement incorporates the interplay between the two aspects of the software 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 (re)produce 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.

Similar to previous work on process related change [e.g., 25], the process of change is understood by drawing on Structuration theory [12]: a process theory that reconciles agency and structure by asserting 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 both, the 'medium and outcome of the contingently accomplished activities of situated actors' [7, p. 132], and are considered to be enabling as well as constraining. The emergence of the software processes is seen to occur through a structuring process that acts as a linkage between the context, and the content to help to explain the process of change. Giddens' [12] 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.

First, human communication uses interpretive schemes, which are drawn upon to make sense of the actions as they are played out. To undertake the interaction related to systems development and improvement actors draw on stocks of knowledge and shared experiences to communicate about how to undertake, and change, the software practice; and software engineering features and defined processes act as frameworks for learning [20]. So, related knowledge is contested and shifting: knowledge is drawn on and changed through action. The theory-in-use (or process-in-use) is challenged and reinterpreted through this new experience. Espoused theory changes initially through changing the interpretive schemes that individuals hold and then in turn the social structures such as the defined processes and infrastructure. This reflection may be a speculative or deliberate attempt to learn for the future, but it is implicit in Giddens' view of actors being knowledgeable and continuously reflexive.

Next, 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. 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 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. 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.

Third, we sanction our actions by drawing upon norms thereby creating or recreating structures of legitimation.

Table 1
Conceptual model of the theoretical framework

Context	Organisational history (past success/failure; previous quality; conditions for adopting and using new process) Environment (software engineering profession; competition dynamics; influence of customers and vendors) Organisational (resources; strategy; mimetic behaviour) Information systems function (development and SPI infrastructure; group dynamics; individual attributes)
Emergence of process and products	Dialectic of defined process and process-in-use (software development informed by the process improvement and defined processes; processes changed through practice; application of skills and knowledge developed through experience) Planned and unintentional change (adopting and using new processes; reflection-in-action; maturing mental models) Consequences of adopting (outcomes anticipated and unanticipated; efficacy and quality performance of outcomes)
Process of change	Interpretative schemes (stocks of knowledge; software engineering features as frameworks for learning) Facilities (use of personal or organisational resources to change/retain current approaches; trust between managers and practitioners) Norms (the defined process acts as the norm; practised process becomes the norm; narratives of meaning and action; language as active process of legitimisation and institution)

Norms are the rules or standards that govern appropriate conduct, constraining and enabling action. 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. Developers and managers sanction their actions through the narratives of meaning and action [11], by which the defined process acts as the norm and the practised process becomes the norm.

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.

The separation of these modalities is only for analytical purposes, as they occur together in practice. Incorporating these modalities into an analytical framework therefore sensitises the researcher to these unconscious features linking the process of change to the emergent contextual structures. The enactment of, and changes to, the software processes are seen to embody the modalities of the structuring process.

Giddens [12] 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. The framework recognises this on-going dialectic between the defined processes and the processes in use that shape the software practice. How this framework was applied within the research is discussed further below.

4. Research methodology

4.1. Case strategy

The research methodology was developed with the purpose of ensuring the work had relevance and rigour. Case studies are ideal for achieving the research aims as they capture the knowledge and views of practitioners, with longitudinal case studies helping to show how the contextual relationships develop and evolve. A single-case strategy is adopted here with the emphasis on the richness of the data collection over a significant period. Single cases are an effective means for communicating conceptual develop-

ments to practice [5]. Interpretive research is particularly applicable in complex real-life situations and can give a deeper understanding of the underlying processes of organisational change. The case study methodology was based on Pettigrew's [27,28] principles of longitudinal study and processual analysis. Following Eisenhardt [10], the research stages were structured to enfold the literature, rather than being theory led.

A dialectical hermeneutic perspective [23] was adopted in support of Structuration theory to understand the influences, both realised and hidden, and the consequences of any action, both intentional and unintentional. Dialectical hermeneutics builds on hermeneutics (the analysis of the meaning of a text) but also recognises the dialectic between text and interpreter. Hermeneutics is used to explore the socially constructed contexts of organisations by interpreting the underlying sense from the ethnographic data. This is a reflexive, iterative approach to deriving meaning, from the historical context of the phenomena. By looking beyond the actors' own interpretations it helps to understand the influences, both realised and hidden, and the consequences of any action, both intentional and unintentional. 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.

By combining these theories and research approaches the situated action of the SPI programme within the case is 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, this strategy 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.

4.2. Case selection

The case organisation, InfoServ (a pseudonym), is a leading global information services company. In 1980, InfoServ's UK sales were £3 million, rising to over £60 million by 1990. Following a number of acquisitions, it has over 13,000 employees now 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.

This study focuses on the 25 person Market Analysis Package (MAP) software team based in the GeoMarketing (UK) division over a 10-year period. The division is the smallest in the organisation, with approximately 200 personnel and £10 million turnover now. The division combine data and software products for the market analysis purposes. The MAP product is the division's flagship product and with nearly 200 customers contributes half of their income. It is a PC-based geographical information system developed primarily in Visual C++ that supports statistical analysis of customer or prospect data.

Rather than making the unit of analysis a specific organisational group, the continuous software process improvement activity is used as a theoretical boundary to guide the data collection and analysis, and allowing the case to be compared at a later date with another improvement initiative even where the organisational forms are different [22,27].

4.3. Data collection

A mixture of in-flight and historical data was captured through participant-observation for over a year. This engaged research gave wide-ranging access to a variety of people and data sources, allowing contrasts and congruencies between and within social groups to be identified by taking the different views into account. Data have been drawn from across the organisational strata and subgroups, avoiding the tendency to concentrate on a management or development group perspective.

The role was undertaken overtly, also enabling 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. A journal was used to note thoughts, observations, ideas, events, etc. This approach provided access to the insiders' world of meaning and enabled the development of a more holistic response to the events [18,24].

A set of 29 formal semi-structured interviews of all the software development team and their line management were undertaken. Standard interview questions ensured a systematic coverage of material, whilst the openness of the interviews allowed the interviewer to respond to any gaps, misunderstandings or follow up required. This interaction makes the interview more 'interpretively active' [15, p. 114] allowing the researcher to pursue tracks as they emerge, and to challenge the perceptions and values of the respondents as is consistent with a critical interpretive study.

Also, a set of 27 informal review meetings were held with the software managers to discover their intentions for, and reactions to, product and process developments. These meetings were based on a generic structure but the questions emerged from the context of the current events,

so there was no pre-determined set wording or detailed topics.

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. These provided information about the activities of the software team, their stated intent, and the outcomes. Software metrics collected by the software group were also acquired for use in the analysis of the perceived efficacy of the SPI activities.

4.4. Data analysis and interpretation

The initial conceptualisation began with making notes in the margins. During this stage the material was read to identify nuances and to reflect on what the significant patterns were through the study. The analysis stage developed through an iterative coding and pattern generation process. So the initial process analysed the interview, journal, meeting and quantitative data for patterns and themes.

Time-based analysis was conducted to show the temporal relationship between different parts of the data [22]. The analysis included identifying patterns between problems identified with the processes, actions planned and taken (or not taken), and outcomes, both intended and unintended. 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 was developed.

The reflexive nature of the analysis enabled the theoretical framework to mature. Our engagement with the meaning of the case encourages the dialectic between the researchers' own frames of meanings and those found in the data, thus challenging the positions that are taken for granted, the perceptions and values of the participants and our own pre-constructed perspectives of software process improvement through pre-determined change. The rejection of early frameworks ensured an original rather than a pre-conceived framework was created. This approach was akin to Pettigrew's [28] cycles of deduction and induction, which he claims is where the real creative process takes place. The iterative nature of the methods is key to the success of case research [19]. The analysis identified patterns of data from different sources that pointed to common themes and facets, and to contrasts.

Further analysis of the literature was undertaken to review how the case supported and challenged the existing materials, thus augmenting the theoretical framework with social theory. The selection of the literature followed from the data and had resonance with the tentative theoretical concepts. For instance, Structuration theory gave a richer understanding of how intended and unintended outcomes related to stated motivation, and developed a deeper understanding of how the actors' norms were changed in relation to the processes and how these were shared through action. This interpretation of the facts occurred through reflection about the data, as part of the construction of the theory. Once the framework had matured and stabilised, detailed codes were defined based on the model in Table 1 and used to fully re-analyse the data to ensure consistency.

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, 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. To ensure that the account is plausible two forms of feedback have been sought: on the case accuracy and interpretations, and on the framework and analysis. Informants have reviewed the accuracy of the case study data and agreed the interpretation of the data.

5. SPI at InfoServ: a chronological analysis

5.1. Chronlogical analysis

A chronological approach is adopted to analyse the emergent nature of the processes and their interplay with the context and software product development. In this section, a narrative of events is used to show how changes emerged through situated practice bringing about a metamorphosis in the context and the software processes. The account highlights factors in the company and external contexts that impacted on the progression of process improvement and product development. The nuances of how and why the processes changed are discussed in detail below by drawing on structurational concepts. The richness of the original case data means that the data presented here is only a sub-set of the full case, but it reflects the analysis of the whole. The following section then discusses the findings in terms of the theoretical perspective.

The case study is split into three periods, coinciding with three distinctive product and process development episodes occurring within the case: the early years of development forms the historical context (period 1); and then the action – context linkage is discussed over the process formalisation (period 2) and the SPI initiative (period 3) using the framework as a structure for the discussion. Each of the latter sections highlights the contextual shaping of actions, the emergence of products and processes through action, and how the changes occurred by drawing on features from Structuration theory.

Table 2 summarises the key actions and the context in which they occurred across these periods. 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 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.

Table 2
Emergence of processes and products

		Period 1 (5+ years)	Period 2 (3 years)	Period 3 (2 years)	Post period 3
Context	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	OO methods and CASE tools refined
	Organisational	SPECTRUM classification; centralised IS	Move to become ISO 9000 accredited	Merger with USI; lean sales period; pressure for new functionality	Move towards internet products
	IS function	Decentralised PC development; project overrun	Recruitment of new IS staff; Lack of experience of OO/C++ in team	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, new standards; process–action teams	New processes defined
Structure/action linkage					
Actions	Product development	DOS system becomes amorphous/distended	New Windows product using layered architecture	MAP product stabilised & enhanced; develop new products via suppliers	Major release of MAP2 with data mining tool is 'most stable yet'
	Process improvement	Individual ad-hoc processes	Develop QM but application variable; OO approach partially adopted	Redefinition of process; SPI formalised across the team; process refined	Progress with formal SPI actions slow; new process changes

5.2. Period 1: historical context

The historical context exhibited strong financial growth from an innovative approach to product development. The development of a PC-based DOS software package had helped to expand the market share for the company. The market strength of the product formed structures of domination, allowing the divisional Board of Directors to draw on internal resources to persuade the organisational management to allow them to set up their own development team. As the software team grew, the division's expertise in programming was enhanced during this period, but typical of many PC developments of this era the methods used remained ad hoc.

A reactive business strategy was a key feature of the dynamic between the organisational context and the ongoing software development. The software development was characterised by changes to the products in response to opportunities and client requests. Traditions from previous developments were used to legitimise the informal and reactive approach during development.

5.3. Case history period 2: formalisation of the software process

5.3.1. Contextual shaping of software change

In addition to the formative context from the previous development there were three external contextual factors that influenced 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 following the early industry growth in the use of object-oriented (OO) methods, there was a lack of expertise and supporting tools in the profession. These external factors interacted and blurred with organisational factors and with the customers' demands for new products.

The software product and processes were also shaped by the process capability of the team, which can be understood as an element of that social structure that changes through learning in action. The OO capabilities within the IS function varied; the new ideas were understood to different degrees depending on prior skills and experience. 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 seen in the vagaries of the MAP product. However, during the development of the product the developers' knowledge of OO grew through observation, training, and their learning-in-action.

5.3.2. Emergence of process and products

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.

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 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 shaped the attempt to change the development approach.

The enacted processes did not always follow the defined version. The process-in-use changed during the development and the organisational understanding of the espoused process had shifted through the on-going organisational inquiry, negotiated practice, and shared learning. Through this on-going, changing software practice, the structures of software development and business context emerged. The actions reinforced, or altered, the context at all levels.

5.3.3. Process of change

These changes in the process can be understood by reference to the duality of context–action interaction as analysed through the modalities in Structuration theory.

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. They changed the software processes through their own capability, and related social structures. The developers' willingness to support the changes increased as they began to trust the project manager. The technical architect too was able to legitimate the new processes by appealing to the successes in his previous work. These knowledgeable actors, in Giddens' terms, drew on relevant interpretive schemes to communicate their experiences.

An example of how the Board drew on facilities for power structure was seen as a 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. The organisational move towards ISO 9000 accreditation supported these changes. 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'.[1] 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 the project manager's agenda for change. He was able to communicate the intent to 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 perceived industry norms of development methodology use to deliver quality systems.

An interesting example of the dialectic of control; and the interplay of resource, norms and interpretive schemas was evidenced in the way the organisational context affected the software processes through the reaction to the pressure of the competitive environment. In this project, once their main competitor launched the Windows version of their product, the pressure on the organisation dramatically increased. The pressure sanctioned some developers to dispense with aspects of the defined process, who saw them as a burden. So, some developers partially reverted to their previous approaches with which they were confident drawing on their shared knowledge of the previous approaches as norms to legitimate this reversion. The unanticipated outcome of this action is evident in a subsequent lack of modularity, in problems in implementing abstraction in a tiered architecture, and in product defects. One team, however, pursued the defined process as their team leader's experience contrasted with this team norm. So whilst the espoused process had changed during this period, the processes-in-use varied by developers and through time.

5.4. Case history period 3: software process improvement

5.4.1. Contextual shaping of software change

The historical context of the software function had been shaped by the outcomes of the intentional and adaptive actions within the previous development. 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.

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, with the side-affect of web-based systems being established for a variety of business uses. Whilst these changes did not affect the MAP product directly, a resultant growth of the division's software product portfolio reflected these opportunities and thereby influenced actions in the team.

5.4.2. Emergence of process and products

The software management felt that it was important to continue to refine the processes for developing and managing the development. So, the MAP team initiated a set of improvement actions and, in due course, this became a software process improvement programme. These actions, whilst partly planned can also be seen to incorporate

[1] Developer Interview.

improvised and unplanned activities. InfoServ did not use maturity models in the definition of the process changes. To undertake the SPI project, the division devised and followed an approach which closely resembled the SEI's IDEAL model [21]. Specific needs were explicitly identified and solutions proposed as part of the organisational improvement activity; these solutions were then designed, disseminated and evaluated. Workshops and process action teams were used to identify changes. Actions were planned based on perceived needs. Actions were planned related to project planning, software development processes, and quality assurance and control.

By allowing team members to do the actions according to their own priorities meant that progress was variable. Some of these actions produced intended outcomes such as the definition of new process areas, enhancements in the software inspection process, and the introduction of automated testing and post-release review meetings. However, other stated actions were altered through practice or abandoned in favour of other priorities. The planned actions and outcomes from the formal SPI project were therefore only part of the story. Improvements occurred through the on-going practice and improvisations of the practitioners as they identified and sought to solve perceived problems, or found and shaped an external solution to solve a problem previously identified.

An example, of the process-in-use changing through practice was the introduction of a beta test process to address the need for client involvement. 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 response caused an unanticipated impact upon the development team, as they had to react to suggestions and defects. In due course, the concept of beta releases was gradually incorporated and then formalised, but this had not been the intention of the initial action.

In addition, some of the new ideas were drawn from outside of the organisation but they were only brought into the development practice when the idea complemented the perceived need, such as the introduction of Critical Chain for project planning and control as a way to reduce project overruns.

Two years after the initial instigation of the SPI project the software management stated that the SPI initiative was at an end, but the study followed the development group for two further years and observed the unfolding of the existing initiatives, and interlinked new developments in the processes and the product portfolio.

5.4.3. Process of change

Again, we can see these emergent changes as having occurred through a structuring process as defined by Giddens. First, interpretive schemes are drawn upon to make sense of the actions as they are played out, and in so doing change the social structures of the process as defined through language. In the example of the beta test from above, in an attempt to respond to the unanticipated effects of releasing prototypes of the software early to key clients and to justify the resource required by the developers, the software management began to change the language, describing it as a beta testing process. In so doing, this change in interpretive scheme reformed the structures of signification thereby redefining the software life cycle. In due course, the espoused process changed, but only after the process-in-use had become the norm.

Individual motives were seen to be significant in shaping the activity as they related to how people were willing to use personal or organisational resource, but these motives did not always equate to the publicly stated intent. The project manager 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'.[1] This stated reason was perhaps sufficient to legitimate the change, but only shows part of the underlying motive, which can be identified through the manager's actions and statements. As partly indicated by the nature of the above statement, his motives were related to strengthening 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 management needed to react to this developing political situation. To legitimate these actions, they drew on the professional norms that valued structured approaches in developing quality software.

Similarly, as improvisations arose from needs identified during the development, as individuals reflexively monitored their own action the champion of the idea would legitimate it through a personal success or external norms. In each case the communication drew on structures of signification – language that others could relate to. A common pattern with such changes was that the person championing the introduction considered them relevant and valuable. 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 practised process was then recreated as the norm, and thus the interpretive schemes shifted through time.

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 an initiative to introduce component-based design that eventually failed. In this example, developers and managers alike acted to resist changes – sometimes in contrast to their own stated purpose – by withholding resource and drawing on structures of domination to maintain the current norm. So processes changed through an

on-going negotiation between individuals that reinforced or changed the existing structures.

5.5. Evaluation of the software process improvement project outcomes

At the end of the process improvement programme its progress was reviewed to establish the future direction. A summary of the review is provided here to evaluate the efficacy of the changes.

Despite slow progress on the official SPI actions, significant progress was achieved through changes that emerged through practice and individual reflection. The project had a significant impact on the processes, the resultant products, and therefore the business. There were a number of unplanned, improvised changes. Some of these changes were to address problems identified at the beginning of the SPI initiative, others were to address new issues that had arisen since that meeting, but many simply arose from the on-going activity of developing the software and were formalised. 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.

It is, of course, difficult to tell the impact of SPI to the exclusion of other factors, but it is possible to identify benefits that occurred over the course of the project:

- Reduction in defects in releases
- Improved perceived product quality
- 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

Based on the cost of defect rework and time spent on SPI activities, a conservative estimate of the return on investment was 6.5, which is in the middle of the range of examples from other cases [35]. More importantly, it was the indirect perception of such metrics around the organisation that was most important. The view of all the business directors was very positive, recognising the business impact that the process improvement had.

In terms of the division's process capability, an informal assessment was made of their position at the beginning of each of the three periods. This assessment takes each process area at levels 2 and 3 of the CMM and makes a simple assessment of the capability. The judgements are: not implemented; informally implemented; defined but not always implemented; implemented. It can be seen from the summary provided in Table 3 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 areas across levels 2 and 3 of the CMM.

6. Discussion: lessons for theory, practice and research

This section draws out the lessons from the case and our research methods in three areas. First, we reflect on the theoretical developments from the analysis. We show how the use of Structuration theory augments the current SPI literature. Consequently, we highlight the way in which the case supports the view of SPI as situated change with the software processes emerging through the on-going software product development. From these theoretical developments we propose lessons for software engineering practice that acknowledge the adaptive, reflective perspective by moving towards an agile view of SPI. We then reflect on our research approach to enable other qualitative software engineering researchers to exploit and build on our approach.

6.1. Lessons for software process theory

6.1.1. Insights from using the structurational perspective

It has been shown that to look back at SPI projects as a simple planned change tends to give an overly neat, ordered view. Such views of software activity imply a picture of fake rationality [34]. By employing Structuration theory as the basis for the analytical lens, the case explicates the dynamics of emergence in process improvement. By imposing the temporal dimension the structurational perspective enables us to articulate the currents and eddies of the micro-level space-time dynamics of the process with the macro-level flow of events over time. The structuration-

Table 3
Summary of process capability by time

	Perceived level of competence	Period 1	Period 2	Period 3
Level 2: managed 7 process areas	Not implemented	5	2	1
	Informal	1	1	2
	Defined	1	3	0
	Implemented	0	1	4
Level 3: defined 13 process areas	Not implemented	3	3	3
	Informal	6	2	2
	Defined	4	4	1
	Implemented	0	4	7

al frame makes explicit the complex interplay of the agency-structure inter-relationship as it co-evolves with the exercise of agent autonomy.

The actions of agents can be understood as purposive and intentional as situated in a given time-space locality. However the intended and unintended consequences of multiple agents become conditions of new actions: structures change constantly but in unpredictable ways. Improvement in software processes can be seen to be 'simultaneously rational and unpredictable; planned but emergent; purposeful but opaque' [8, p. 137].

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. The agent's power can be seen to be instantiated in their action, for example: using or not using the process; giving or withholding support through resources; the use of language to promote or counter the existing process norm.

As the narrative demonstrates, the metamorphosis of interpretive schemes is confluent with transitions between legitimisation structures and dialectics of control. The legitimisation structures draw on ''norms'' selectively, recruiting different internal and external facilities over time (demonstrating the co-evolution of interpretive schemes, agency and structure). The norms shaped the retention of existing ideas or the introduction of new ideas, with the process-in-use informing, guiding, and organising future practice. Such norms thereby sustained existing approaches.

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 [12] view that agency takes place with knowledge and practical consciousness. Agents' capacity to choose (and their ''bounded rationality'') can result in divergent co-evolutionary paths as is illustrated in Section 5.3.3 where there is apparently a ''bifurcation'' under pressure – one set of agents revert to past practice using that as the ''reference norm'' whilst the other set continue with the new practice.

Such resistance has been noted as a feature of SPI programmes [32]; here it was seen that both managers and developers drew on their resources and structures of domination to resist changes in line with their motives. The norms changed as they were challenged through experience, negotiation within the group, and through the introduction of new ideas from other sources. The norms of the practitioners become the traditions for future practice.

So overall, 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 theoretical framework helped to highlight how the enacted processes form the norms for future practice and recreate the context for future developments. The improvement was understood as a negotiated process of change, occurring through a structuring process. The reasons for undertaking the SPI project and related actions were more complex than initial rhetoric suggests.

6.1.2. Process metamorphosis: change through situated action

Whilst the literature recognises the emergent nature of software development practice [20,33], the dynamics of emergence is under-explored. In this paper, we have used the structurational lens to elucidate the dynamics of emergence in the case example.

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' [1, p. 34]. There is a dynamic relationship between our beliefs and our actions: knowledge is acquired through action by practice and habit [36]. Actors made sense of their action by imposing their own worldview, interpreting the application of specific methods according to their perspective. As such, organisations are continuously changing through active implementation and reflection on their theory-in-use, rather than simply implementing the espoused theory (or defined processes) [30]. As small, often unplanned, improvisations continue through time then significant changes occur [26].

The emergent properties of actions and outcomes mean that they cannot be known a priori, but it does not mean that there is no intended design for an action. Rather, design and emergence coexist. Emergence accounts for divergence between intended and realised design [30].

6.1.3. Emergence of the process: the product–process dynamic

The case narrative demonstrates the on-going dialectic between the defined processes and the processes in use that shape the software practice. Throughout the study, the processes were seen to change through an on-going dynamic with the product development. Software development was informed by the process improvement activity. The means developers used to create software artefacts are were drawn from the context, and through the application of skills and knowledge developed through previous experience. The context of software development included the defined and routine processes, which were (re)created through the actions and experiential learning in the development activity.

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 emergence is not 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. An

actor's commitment helps to focus their sensemaking for sustained action [36].

6.2. Lessons for software engineering practice

6.2.1. Improvement through planned and adaptive change

The case data show 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.

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.

Within the case the processes can be seen to have changed through a combination of planned 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 on-going changes emerging from the product development activity.

The process of improvement needs to account for reactive, reflective changes if the processes are to be improved not just extemporised. 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. Rather it was when a need was clearly answered, often serendipitously, within a training event that it was incorporated into the practice. Changes in the process-in-use at InfoServ were seen to occur through different forms of innovation. 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.

6.2.2. Linking improvement in software products and process to business objectives

The project at InfoServ 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 [4] 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; 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.

To support this more situated, market-oriented perspective, we need to develop an agile approach to SPI so that the process improvement reflects the needs of the given context [2]. 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.

6.3. Lessons for qualitative software engineering research

The research 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 researchers' own biases are recognised and challenged.

Relevance can be achieved by selecting appropriate topics that are of interest to and develop outputs that can influence practitioners [5]. The topic was identified directly from the case scenario but also through the recognition of the growing importance placed on process capability by software organisations. SPI has been well researched but the problems have not been resolved. Here the case is used to identify a fresh understanding of process improvement.

To develop research that is useful to the software engineering community the research was designed to be theory-based and context-rich. Here we build 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 the information systems literature. 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.

7. Conclusion

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. Yet this literature has been criticised because it fails to understand the micro-dynamics of software practice. This paper therefore adds to a limited, but growing, body of work exploring the organisational issues of SPI.

Here the on-going relationship between software process improvement and product development is seen as a constant and fundamental aspect of software practice. The longitudinal, processual case research helps to disentangle facets involved in the emergence of the processes over time. It has been shown that systems development and process improvement is the outcome of a complex process of interaction and communication shaped by the context of the actors. This interaction has been seen as occurring through a structuring process. In contrast to the dominant deterministic views in the SPI literature, the problems with implementing SPI are understandable if we consider it to be organisationally situated.

A theoretical model is used, 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 analytical framework provided a useful lens through which the process of change has been reviewed as occurring through a structuring process. The analysis revealed how the changes occurred through a structuring process, linking action with its context and the context with the actions.

The dialectical hermeneutic analysis of this case 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. Second, 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 on-going 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.

The paper therefore makes significant contributions to software engineering 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; and extending existing qualitative research methods by elaborating on the previous use of a critical interpretive strategy. This perspective has been incorporated into a theoretical framework that highlights the intertwining between software development and process improvement. The framework provides a lens through which other cases can be analysed. It is, however, necessary to further evaluate the framework. From this theoretical perspective it is anticipated that a more agile view of SPI is required if organisations are to leverage the emergent nature of the process improvement activity.

References

[1] I. Aaen, J. Arent, L. Mathiassen, O. Ngwenyam, Mapping SPI ideas and practices, in: L. Mathiassen, J. Pries-Heje, O. Ngwenyama (Eds.), Improving Software Organizations, Addison-Wesley, Upper Saddle River, NJ, 2002, pp. 23–46.
[2] I. Allison, Towards an agile approach to Software Process Improvement: addressing the changing needs of software products, Communications of the IIMA 5 (1) (2005) 67–76.
[3] I. Allison, Y. Merali, Software process improvement: towards an emergent perspective, in: M. Levi, A. Martin, C. Schweighart (Eds.), Proceedings of 8th UKAIS Conference, University of Warwick, 9–11 April, 2003.
[4] J. Bach, The Immaturity of the CMM, American Programmer 7 (9) (1994) 13–18.
[5] I. Benbasat, R.W. Zmud, Empirical research in information systems: the practice of relevance, MIS Quarterly 23 (1) (1999) 3–16.
[6] T.B. Bollinger, C. McGowan, A critical look at software capability evaluations, IEEE Software 8 (4) (1991) 25–41.
[7] P. Cassell, The Giddens Reader, Macmillan Press, Basingstoke, 1993.
[8] C.U. Ciborra, A theory of information systems based on improvisation, in: W.L. Currie, R. Galliers (Eds.), Rethinking Management Information Systems, Oxford University Press, Oxford, 1999, pp. 136–155.
[9] R. Conradi, A. Fugetta, Improving software process improvement, IEEE Software 19 (4) (2002) 92–99.
[10] K.M. Eisenhardt, Building theories from case study research, Academy of Management Review 14 (4) (1989) 532–550.
[11] R. Fincham, Narratives of success and failure in systems development, British Journal of Management 13 (2002) 1–14.
[12] A. Giddens, The Constitution of Society, Polity Press, Cambridge, 1984.
[13] B.H. Hansen, J. Rose, G. Tjornehoj, Prescription, description, reflection: the shape of the software process improvement field, International Journal of Information Management 24 (2004) 457–472.
[14] J. Herbselb, D. Zubrow, D. Goldenson, W. Hayes, M. Paulk, Software quality and the capability maturity model, Communications of the ACM 40 (6) (1997) 30–40.
[15] J.A. Holstein, J.F. Gubrium, Active interviewing, in: D. Silverman (Ed.), Qualitative Research: Theory Method and Practice, Sage Publications, London, 1997, pp. 113–129.
[16] D.M. Hosking, N. Anderson, Organizational Change and Innovation: Psychological Perspectives and Practices in Europe, Routledge, London, 1992.
[17] W.S. Humphrey, Managing the Software Process, Addison-Wesley, Reading, MA, 1989.
[18] D.L. Jorgensen, Participant Observation, Sage Publications Inc, Newbury Park, CA, 1989.
[19] H.K. Klein, M.D. Myers, A set of principles for conducting and evaluating interpretive field studies in information systems, MIS Quarterly 23 (1) (1999) 67–94.

[20] L.Mathiassen, Reflective Systems Development (1998), Online at http://www.cs.auc.dk/~larsm/rsd.html, 2006 (accessed 05.03.06).
[21] B. McFeeley, IDEAL: A User's Guide For Software Process Improvement (CMU/SEI-96-HB-001), Software Engineering Institute/ Carnegie Mellon University, Pittsburgh, PA, 1996.
[22] M.B. Miles, A.M. Huberman, Qualitative Data Analysis: An Expanded Sourcebook, 2nd ed., Sage Publications, Inc., Thousand Oaks, CA, 1994.
[23] M.D. Myers, Dialectical hermeneutics: a theoretical framework for the implementation of information systems, Information Systems Journal 5 (1) (1994) 51–70.
[24] J. Nandhakumar, M. Jones, Too close for comfort? Distance and engagement in interpretive information systems research, Information Systems Journal 7 (2) (1997) 109–131.
[25] W.J. Orlikowski, CASE tools as organizational change: investigating incremental and radical changes in systems development, MIS Quarterly 17 (3) (1993) 309–340.
[26] D.E. Perry, N.A. Staudenmayer, L.G. Votta, People, organizations, and process improvement, IEEE Software 11 (4) (1994) 36–45.
[27] A.M. Pettigrew, Longitudinal field research on change: theory and practice, Organizational Science 1 (3) (1990) 267–292.
[28] A.M. Pettigrew, What is processual analysis? Scandinavian Journal of Management 13 (4) (1997) 337–348.
[29] T. Ravichandran, A. Rai, Quality management in systems development: an organizational system perspective, MIS Quarterly 24 (3) (2000) 381–415.
[30] P.M. Senge, The puzzles and paradoxes of how living companies create wealth: why single-valued objective functions are not quite enough, in: M. Beer, N. Nohria (Eds.), Breaking the Code of Change, Harvard Business School Press, Boston, MA, 2000, pp. 59–82.
[31] E.B. Swanson, Information systems innovation among organizations, Management Science 40 (9) (1994) 1069–1092.
[32] K. Thompson, P. McParland, Software process maturity (SPM) and the information systems developer, Information and Software Technology 35 (6/7) (1993) 331–338.
[33] D. Truex, R. Baskerville, H. Klein, Growing systems in emergent organizations, Communications of the ACM 42 (6) (1999) 117–123.
[34] D. Truex, R. Baskerville, J. Travis, Amethodical systems development: the deferred meaning of systems development methods, Accounting, Management, and Information Technology 10 (2000) 53–79.
[35] R. Van Solingen, Measuring the ROI of software process improvement, IEEE Software 21 (2004) 32–38.
[36] K.E. Weick, Sensemaking in Organizations, Sage Publications Inc, Thousand Oaks, CA, 1995.