Nottingham University Business School

Software Development as Organizational Learning

Allan Kelly

2003

A dissertation presented in part consideration for a degree in Master of Business Administration.

Abstract

Most literature on software development addresses the subject as an engineering discipline. As such the discipline is a little over 30 years old, while technology has changed the dominant paradigm of organization has been one of process and methodology. While "people" are often cited as the determining factor in success or failure much less attention has been paid to this side of the discipline.

Since the early days of the discipline there has been an alternative text which places people centre stage, and rather than taking an engineering perspective takes an anthropological or sociological view of the discipline. However, this text has lacked a framework in which to analyse development activities.

This paper proposes *Organizational Learning* as a framework with which to analyse software development. Through literature review and qualitative research the organizational learning perspective is contrasted with the classical engineering perspective. We find that while the classical view forms part of the software developers' identity and can be a powerful force for change it fails to accurately describe the complexity of the field and may even support a number of social defences which inhibit software development. The organizational learning view provides for much richer description and analysis of the field.

While the classical view sees software development as largely separate from the wider organizational environment the organizational learning perspective embeds software development within this environment. From this point of view the act of software development has a role to play in higher order organizational learning by the company.

Finally, as a predominantly knowledge based activity software development can be seen as a metaphor for twenty-first century business. In this context we see that the fetish of methodology can hinder the emergence of knowledge and capabilities.

"In many ways, managing a large computer programming project is like managing any other large undertaking - in more ways than most programmers believe. But in many other ways it is different - in more ways than most professional managers expect."

Frederick P. Brooks, *The Mythical Man Month* (1975, p.vii)

"Software entities are more complex for their size than perhaps any other human construct, because no two parts are alike"

Frederick P. Brooks, *No Silver Bullet* (1986)

"Some readers have found it curious that *The Mythical Man Month* devotes most of the essays to the managerial aspects of software engineering, rather than the many technical issues. This bias ... sprang from [my] conviction that the quality of the people on a project, and their organization and management, are much more important factors in the success than are the tools they use or the technical approaches they take."

Frederick P. Brooks, *The Mythical Man Month*

- Anniversary Edition (1995a, p.276)

Table of Contents

Table of Figures

Acknowledgements

There are many people who have helped, directly and indirectly, in the creation of this paper and to whom I am most grateful. Firstly I would like to thank my supervisor, Professor Ken Starkey for his guidance, suggestions and encouragement.

Secondly, I would like to thank the interviewees who helped with my research for their time and confidence. Although only identified by pseudonyms here I am sure they will recognise themselves, thank you.

I must also thank everyone at EuroPLoP 2003 - the European conference on Pattern Languages at Kloster Irsee in June 2003. Many of the participants present, and some of the papers presented, where wrestling with the issues considered here. Knowingly or unknowingly, many of the participants served as sounding boards for my ideas and helped in their formation. I am convinced that the fields of knowledge management and organizational learning have yet to realise the potential of pattern languages.

Thanks too, to my sister-in-law, Nicki Kelly for proof reading and pointing out numerous errors in my English, all mistakes that remain are my own.

Finally, I must thank Nottingham University Business School, the school staff, and the MBA class of 2002-2003 for an incredible year.

Allan Kelly

September, 2003

http://www.allankelly.net

Words and tables by Word and Excel ’97, bibliography by Endnote 7.0, pictures by Inspiration 7.0; Mozilla for mail and web and CuteWriter for PDFs all on top of my trusty Compaq Presario laptop.

1 Introduction

1.1 Software business

Modern businesses depend on computer software. Whether it be a commodity shrink-wrapped package like Microsoft Excel or a bespoke application which supports a unique competency of the organization, there can be few companies which would adopt their current structure today without software support.

All software must, in the first instance, be created through the writing of program code. This is fundamentally a human task. Over the years different tools have been proposed and created to reduce this task, however, these tools have met with mixed success. Even when a tool is successfully used all it really does is allow the human software developer to work at a higher level of abstraction and, hopefully, more productively. The creation process is still fundamentally a human activity.

It is easy to forget about the development process when considering the use of IT in the business strategy. However all IT ultimately depends on software development somewhere. Some may consider development a *simple matter of programming* (SMOP) however to do so is to overlook the role software development, and developers themselves, can play in IT deployment and wider organizational change.

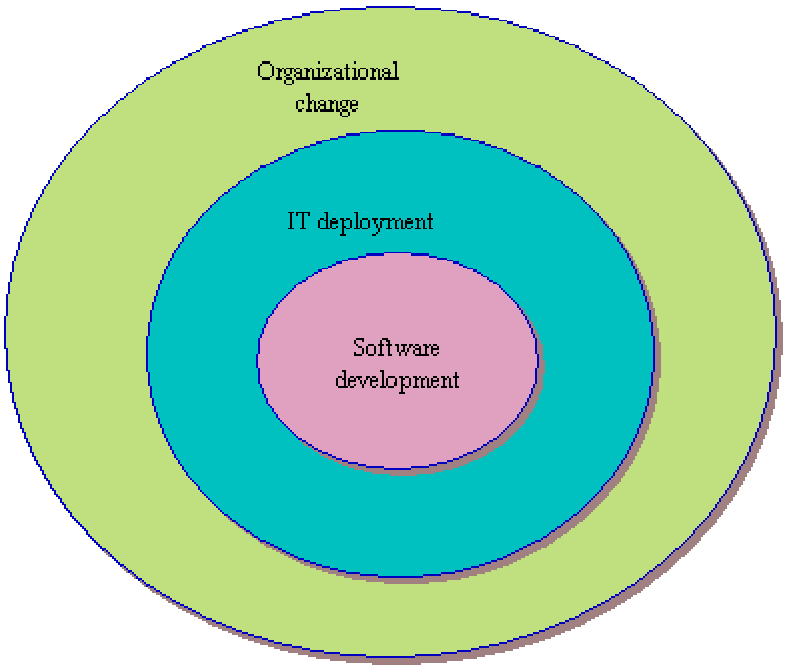


Figure 1 Software development is part of organizational change

Obviously, organizational change can happen without IT, but the reverse is not true. Whether intended or not, organizational change occurs when IT systems are deployed and software is a key factor in any such deployment.

Software may be developed in-house, in which case it is a business necessity for those involved to understand their role in the change process. Alternatively, software may be bought in, perhaps *common-off-the-self* (COTS) in which case, the external developers need to understand how their software may be used.

1.2 Key thesis

The key thesis of this paper is:

> The tools, techniques, and literature of the organizational learning community can usefully be applied to the software development process. To date, analysis of this process has primarily viewed it as an engineering activity. This has lead to an emphasis on technical solutions that may obscure or inhibit learning processes both within the development process, and within the wider organization.

There are three dimensions embedded within this thesis:

- Modern business is dependent on software: therefore it is essential that managers understand the role of IT in organizational change.
- Software developers can benefit from better understanding of change and learning processes, and their role in both.
- As a knowledge based activity, software development is a metaphor for the modern business which can provide valuable insights for the entire organization.

By describing software development as a learning activity it is hoped to show that two advantages accrue:

- This view provides for a better understanding of the development process, allowing for process improvement and thus improving the firm's competitive position.
- Insights from software development may inform the debate on organizational learning and knowledge management, and the wider domain of business administration.

In addition, there is a need to recognise that learning does not only occur during the development process but continues once software has been delivered to customers.

Indeed, rather than viewing the customer/end-user as a static entity, we may view them as an active participant in the learning process that is software development.

1.3 A brief history of software development

The theoretical foundations of the programmable computer were laid by Alan Turning in his 1937 paper, *On Computable Numbers* (Singh, 1999, p.168-169). In this paper he described the *Universal Turing Machine* that could be programmed to decide any computable question. It was another six years before such a machine, *Colossus,* was built by Tommy Flowers (Singh, 1999,p.244, Smith, 1998, p.9). With the creation of the first programmable computer the age of computer programming was born.

The 1960's saw the arrival of the first large computer systems, most notably OS/360 for the IBM/360 mainframe. It quickly became apparent that large systems were difficult to develop and were prone to cost and time over runs. Yet it was becoming more apparent that software was key to new products and, for NATO weapons systems.

So concerned was NATO that it convened two conferences in 1968 and 1969 on the subject. The 1968 conference at Garmisch in Germany coined the term "Software Engineering" and declared there to be a "software crisis" with the demand for software development running far beyond the ability to develop it.

The "software crisis" never really went away, nothing ever happened to resolve the crisis, but, after 35 years it is difficult to use the word "crisis" about what seems to be a fact of life.

One response was the emergence of methodologies. These described the steps that needed to be taken in order to develop a piece of software in an engineering fashion. Arguments raged about which methodology was best, which order to do things in, how to maintain flexibility and how detailed the methodology should be, but no methodology ever succeeded in providing the *silver bullet* (Brooks, 1995b) to produce software on time, and to budget.

The only thing the competing authors seemed to agree on was that if we could only find the correct ceremony to produce software, and follow it rigorously all would be well. Such methodologies have been characterised as "high ceremony."

Describing software development as engineering or as a methodical process fails to explain much of what happens when developing software. An alternative view has grown up which regards software development as a social activity. These authors

(e.g. Brooks, 1995a, Weinberg, 1998, DeMarco, 1987, Kidder, 1981, Constantine, 1995, Argyris, 1977) tend to express a minority view which is neglected in much of the subject teachings. The majority of writings on software development come from technological and methodological point of view.

During the late 1990's this cycle of beggar-thy-neighbour methodologies started to break down. The rate of business change had accelerated and the high ceremony methodologies could not keep up. Researches such as Fitzgerald (1994, 1995, 1997) and Truex (2000) started to question the logic of the methodologies. In parallel a new generation of writers (e.g. Beck, 2000, 2001, Cockburn, 2002) from the "code-face" of programming started to advocate *Agile Software Development* using "low ceremony" methodologies taking their inspiration from the *Lean Manufacturing* movement.

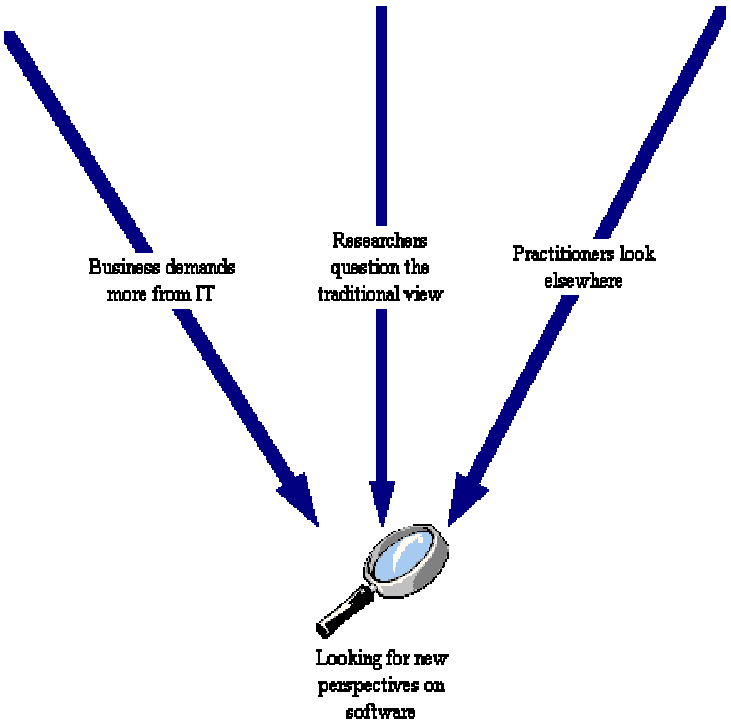


Figure 2 - Factors driving towards a re-assessment of software development

The turn of the century IT spending binge and the dot-com crash have left many wondering about the true value of IT. Such a climate demands a new view of IT and software development itself.

1.4 Audience

This dissertation is written as part of a Master's in Business Administration degree at the University of Nottingham Business School. The primary audience for this report is the scholarly community. In the first instance this audience consists of examiners from the University, but in the longer term, it is hoped this community will include researchers from both the fields of organizational learning and software development.

It is also hoped that this dissertation will find a second audience amongst those in the software development community who are looking for new approaches to development.

1.5 A note on terminology

The term *software development*, sometimes abbreviated to just *development,* is used extensively throughout this paper. Some authors prefer to use the term *system development*, and some prefer the term *information system development* (ISD). For the purposes of this paper these terms are all treated synonymously.

Traditionally the term *computer programming*, or just *programming*, has also been used in this context and some of the older texts referenced use these terms. While strictly speaking programming is but one part of the software development activity when used here the term it is also considered to be synonymous with software development.

Many current texts prefer to use the term *software engineering;* while this may be a further synonym, use of this term implicitly accepts the definition of software development as a engineering discipline supporting the dominant paradigm. Therefore use of this term has been restricted wherever possible.

Software development falls within the wider domain of *Information Technology* - or IT - this term is also used widely. The term is used in this paper when the arguments advanced can be generalised to the wider domain.

In addition the terms *Information Systems* (IS), *Information Communication Technology* (ICT) and *Management Information Systems* (MIS) are taken to be synonymous with *Information Technology* for our purposes.

Finally, the term "hacker" is used in its original sense:

> "A person who enjoys exploring the details of programmable systems and how to stretch their capabilities" (Raymond, 2003)

The term is also extended colloquially as a derogatory term to describe on who writes software without any design or forward thinking. Unfortunately the popular press use the term to refer to what Raymond calls a cracker:

> "One who breaks security on a system." (Raymond, 2003)

Appendix C provides a short glossary of other terms and abbreviations.

2 Literature Review

Literature Review

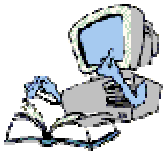
Emergence and dominance of the waterfall and methodologies

Failure of methodologies and the questioning of rational development

Is software going the right way?

The need for a new model

Apply Organisational Learning to software development



What is, organisational learning?

A framework for reconsidering software development

Figure 3 - Overview of Literature Review

2.1 A historical view of software development and the emergence of methodology

Initially software systems were small enough for one person to comprehend. Although they may have been complex in their day they were easily comprehended by the advanced mathematicians and engineers who worked with the systems. Even for many small systems today a single developer, or *lone hacker*, is quite capable of writing a system, but for many systems development is a team activity.

As the size and complexity of systems increased the obvious answer was to add more people:

> "... the worst way to do a programming project is to hire a horde of trainees and put them to work under pressure and without supervision - although this is the most common practice today [1971]" (Weinberg, 1998, p.69)

Despite Weinberg's early warning this is a lesson the software industry continues to make:

> "The earliest software lifecycle, one still in use today, is the Mongolian Hordes approach. This works from the assumption that finished code is always full of bugs, so the sooner you can produce the finished code, the quicker you can get down to removing the bugs." (Ince, 1990)

The "don't plan, just code" approach has been characterised as "hacking". McConnell expresses a common view that a preparation phase, with extensive planning, is vital. He suggests failure to do sufficient planning is due to managers pressuring programmers to start coding - this he names WISCA syndrome, "Why isn't Sam coding anything?" (McConnell, 1993, p.23)

This "just add more people" approach gave birth to Brooks' Law:

> "Adding more manpower to a late software project makes it later." (Brooks, 1995a, p.25)

Brooks has been called "the farther of the IBM System/360" and his seminal 1975 work, *The Mythical Man Month*, drew on his experiences as project manager for the IBM/360 and later the machine operating system - OS/360. Brooks described the failings of contemporary software development and suggested new ways of working.

Eleven years later Brooks suggested there were *No Silver Bullets*:

> "There is no single development, in either technology or management technique, which by itself promises even on order of magnitude improvement within a decade [of 1986] in productivity, in reliability, in simplicity." (Brooks, 1995b)

Over the years many silver bullets have been proposed, some technological, some managerial. This paper is concerned with the latter. These managerial silver bullets have taken the form of methodologies.

2.1.1 Methodologies

Brooks' OS/360 project may have been the biggest software project in its day but it was the shape of things to come. By the late 1960's it was clear that relying on a *lone programmers* or *Mongolian Hordes* was not the way to proceed.

At first the response was to describe the development process, Royce (1970) describe the "waterfall model" - according to Brooks (1995a, p.265), Royce was simply describing what he saw on the Gantt chart. This was a sequence of steps that occur to produce any software:

- Requirements gathering
- Specification writing
- Architecture/Design
- Coding
- Testing
- Operation
- Maintenance

No software engineering text is without its description of this model (e.g. Somerville, 2001, Pressman, 1997). Despite being widely acknowledged as fundamentally flawed it is embedded in culture.

Using this model it is quite easy to apply the scientific management principles of F.W.Taylor (Mullins, 2002, p.55). We see the emergence of methodologies claiming to deliver software on time, and on budget. Workers are grouped by task: *business analysts* gathering requirements and writing specifications, *software architects* and *designers* constructing designs, *programmers* writing code for *testers* to test before the product is put handed over to *operations staff* and a different group of *maintenance programmers* to fix bugs.

Over the years various methodologies have been developed which, to a greater or lesser degree are based on the waterfall model, or, attempt to overcome the limitations of the model. For example Yourdon's Structured Project Life Cycle (Yourdon, 1989), "Jackson" System Development (Jackson, 1983), Object-Oriented Analysis and Design (Booch, 1994), Object-oriented Software Construction (Meyer, 1988). One of the most widely documented and studied methodologies is *Structured System Analysis and Development Method* known as SSADM (Eva, 1991, Duncan, 1995, Downs, 1988) and also known *as Business System Development*, British Standard BS7738. This was mandated for UK Government Projects during the 1980's and 1990's but now appears to be only "best practice."

Essentially, each methodology describes the steps required in the application of technology to a (business) application context. Advocates of methodologies

attribute a number of benefits both to methodologies in general, and usually, to the methodology they advocate. Typical of these would be:

> "There are three primary objectives [for having a project life cycle]:
>
> 1. To define the activities to be carried out in a system development cycle.
> 2. To introduce consistency among many system development projects in the same organization.
> 3. To provide checkpoints for management control for go/no-go decisions." (Yourdon, 1989, p.79)

Despite the multitude of methodologies (over 300 were identified by Fitzgerald in 1994) and tools to choose from, software developments continue to fail. For some the response is to improve the methodologies:

> "Many researchers see the solution to the software crisis in terms of increased control and the more widespread adoption of rigorous and formalised system development methodologies" (Fitzgerald, 1994)

Yet, adopting a more rigorous methodology has problems too. Wastell studied the use of SSADM, one of the most prescriptive "high ceremony" methodologies:

> "Far from facilitating the development process, SSADM encouraged a rigid and mechanical approach in which the methodology was applied in a ritualistic way which inhibited creative thinking. The argument is thus, that methodology, although its influence may be benign, has the potential to operate as a 'social defence', i.e. as a set of organizational rituals with the primary function of containing anxiety." (Wastell, 1996, p.25)

DeMarco and Lister pre-empted Wastell's findings by nine years:

> "You encourage this defensiveness when you try to systemize the process, when you impose rigid methodologies so that staff members are not allowed to make any of the key strategic decisions lest they make them incorrectly." (DeMarco, 1987, p.8)

Behind all methodologies is the assumption that there is a rational, repeatable set of steps for writing software. That we only need to apply these steps correctly and we can produce any piece of software we like. DeMarco and Lister make a useful distinction:

> "There is a big difference between Methodology and methodology. Small *m* methodology is a basic approach one takes to getting the job done. It doesn't

> reside in a big fat book, but rather inside the heads of the people carrying out the work. ...
>
> Big-M methodology is an attempt to centralize things. All meaningful decisions are made by the Methodology builders, not by the staff assigned to the work." (DeMarco, 1987, p.114)

They go on to attribute the claimed benefits for Methodologies not to the method itself but to the benefits of convergence (i.e. developers know what to expect from one another) and *Hawthorne effect* - the well documented tendency of people to perform better when trying something new.

2.1.2 The call of rationality

> "... the 'rational design process' of hierarchical top-down design described by Parnas and Clements (...) is definitely not the way we humans design real systems. Instead, we hop around from level to level, getting good solution insights (...) at seemingly random time." (Boehm, 2001)

In their 1986 paper Parnas and Clements describe a rational development process; as they point out, it seems reasonable that if one is to specify or study a process it should be rational. Importantly however, they recognise that it is difficult to follow any rational process. Of the reasons they give for this, several are of particular interest:

> "1. In most cases the people who commission ... [the] system do not know exactly what they want and are unable to tell us what they know.
>
> 2. ... Many of the details only become known to us as we progress in the implementation. Some of the things we learn invalidate our design and we must backtrack.
>
> 3. ... human beings are unable to comprehend fully the plethora of details that must be taken into account in order to design and build a correct system.
>
> ...
>
> 6. We are often burdened by preconceived design ideas." (Parnas, 2001, p.356)

Here we see several dimensions of learning described:

- Learning by customers who learn *exactly what they want* and learn to communicate it.
- Learning by doing, and the use of feedback loops.

- Gradual learning of details as they become important, that is, developers learn to understand the complexity over time.
- Existing preconceptions inhibiting design and failure to unlearning as the development progresses.

While recognising that a hierarchical rational process will have difficulty with these learning activities Parnas and Clements suggest that a base line, rational, process is still useful. Such a process helps to guide developers in their work and facilitates communication with others, in particular, those new to the project.

In effect we have a dilemma: the rational process is desirable, but is unrealistic. The solution offered by Parnas and Clements is novel: fake it.

> "The process is 'faked' by producing the documents we would have produced if we had done things the ideal way. One attempts to produce the documents in the order we have described here." (Parnas, 2001, p.366)

In effect, Parnas and Clements are saying: "We recognise there is irrationality in the development process, we recognise the appeal of rationality, we can only reconcile the two by faking the rationality."

While Parnas and Clements freely acknowledge they would like to run software development as a rational process they also recognise that this "Philosopher's Stone" is unachievable. They recognise that learning must occur during the development process.

However, the authors also recognise that some degree of control is necessary. In explaining why the "Philosopher's Stone" is desirable (Parnas, 2001, p.357) they echo Yourdon's thinking (section 2.1.1). These points can be summarised as:

- Designers need guidance if they are not to be overwhelmed by complexity.
- Accepting an approximation of the ideal process will bring us closer to the ideal process than a random ad hoc process.
- It enables transfer of people, resources and knowledge between projects within an organization.
- It is easier to conduct project review and measurement.

Within certain parameters, Parnas and Clements are giving developers freedom to customise, or create their own processes. By laying down some restrictions they are creating tight-loose (Peters, 1991, p.318) control mechanism which may be conceived as a form of creative tension (Senge, 1990, p.150). However, Parnas and

Clements do not intend to create a tight-loose or creative-tension environment, their motivation is a pragmatic response to what they see happening.

Parnas' argument is a vividly illustration of the thinking of Mintzberg:

> "Mintzberg argues strongly that we want to be rational but that is difficult to deal with our complicated world in a rational fashion." (Starkey, 1996, p.261)

However, Parnas' solution, to fake the rational, is somewhat different to Mintzberg:

> "we need a management process that is sensitive to both the need for emergent learning and to the practical possibilities, and limitations, of deliberate planning." (Starkey, 1996, p.262)

2.2 Domains of development

Before continuing the discussion it is useful to introduce some terminology. So far we have only considered the development process, or methodology. However we should not forget that the objective of the process is to bring technology in the form of software, and maybe hardware, to bear on a business problem area. Coplien has used the term "domain" to differentiate these fields:

> "A *domain* is an area of specialization or interest. We talk about the *application domain* - the body of knowledge that is of interest to users. ... We walk about the *solution domain*, which is of central interest to the implementors but of only superficial interest to system users." (Coplien, 1999, p.7) (italics in original)

To these terms we will add *process domain* to talk of the body of knowledge concerning the software development process. Most of the literature reviewed up to this point has concerned itself with process domain.

(Some authors have used the term *problem domain* as an alternative to Coplien's *application domain*.)

2.3 Critique of the classical view

2.3.1 An amethodical perspective

The views of software development documented so far have conformed to the prevailing paradigm of software development as a methodological processes. Yet even esteemed authors like Parnas and Boehm recognise the difficulty operating such a practice. Authors such as Fitzgerald and Wastell question the controlling nature of methodologies.

Truex, Baskerville and Travis go further and question the dominant position of methodological paradigms in the software development field (Truex, 2000). They point out that the terms "information system development" and "information system development method" have, in effect, been merged, giving *method* a privileged position in the literature. Consequently the literature has neglected much of what actually happens in the development process:

> "The marginalized [amethodical] text suggests that information systems development unfolds differently [to that] previously believed and that developers adapt methods to particular situations. Developers are successfully mixing and matching elements from seemingly contradictory systems methods" (Truex, 2000)

If we accept the need for methodologies, and that they are necessary for successful software development, then the methodology-less developers observed by Truex should not succeed in their work. Surely methodologies exist to organise development and save managers from *Mongolian Hordes*? Fitzgerald offers an explanation:

> "Non-use of a methodology is not a licence to conduct development in a sloppy or careless manner. Those who suggest that the failure of practitioners to use a formalised methodology is due to ignorance or a lack of awareness on their part may not be presenting a totally-accurate picture. An appropriate analogy might be that of Picasso dispensing with conventional artistic perspective, but from a position of superior knowledge. ... In practice, situations will inevitable arise where the developer needs to step outside the methodology, but formalised methodologies often serve to impose a considerable inertia on the development process. Indeed, the degree of inertia is proportional to the degree of formality of the methodology." (Fitzgerald, 1994)

While there is a good case why formal, rational, methodology should be used to bring order to the development process it seems that development does not occur in a methodological manner. Indeed, Howcroft and Wilson (2003) suggest that trying to understand it as a rational process obscures other views, specifically, the political view - a view also considered by Robey and Markus (1984).

Given that the classical software development literature is centred around the rational, and specifically, methodology, how are we to characterise the software development if the classical understanding is so flawed?

Such is the situation that Fitzgerald has likened methodology to a lamppost:

> "an analogy could be drawn with that of the drunk losing his watch in the street and moving to look for it under the light of a lamppost because the light is best there, even though it had been lost somewhere else. Likewise, it is perhaps easier to conduct research on existing methodologies as the light is best there, rather than to investigate the real complexity of systems development" (Fitzgerald, 1995)

2.3.2 Towards a new understanding: why apply organizational learning to software development?

So far we have seen that the classical, rationalist, approach to software development has centred on prescriptive, even ritualistic, adoption of a methodology. We have also seen that there is good reason to question this approach, however, the language and tools used to evaluate the development process are themselves tainted by association with the methodologies. By confining the language of the debate to the rational the political context is marginalised, and by choosing process as the tool of analysis non-conforming actions are viewed as deviant.

What is a required is a set of tools, a framework, through which we can examine the process domain without using the language of the domain. Yet, the results of this analysis must be applicable to the domain. For all their faults, methodologies contain important ideas, working practices and knowledge - these have often been distilled from developer culture and working practices. We should not seek to destroy them en masse but to extract what is worthwhile.

As we have already seen there is a strong undercurrent of learning in the IT community. Indeed, given the rapid pace of change and introduction of new technologies there is a constant need for software developers to learn. For example, in 1993, the internet as we know it did not exist. To support its growth over the last 10 years the software development industry has developed, and widely adopted a myriad of technologies from languages such as Java, Perl and Python, to communication protocols such as HTTP, SMTP, IP v6.0, etc. Clearly there is learning, innovation and change occurring in the IT community at a rapid pace.

It therefore seems logical that the application of theories of learning to this field may yield some interesting findings. In particular, in the context of group and organization learning it seems the organizational learning theories advocated by the likes of Argyris, Senge, Brown and others may be useful.

While the *People Capability Maturity Model* (PCMM) (Curtis, 2001) from the Carnegie Mellon *Software Engineering Institute* could provide a useful framework for this investigation the model is rooted in the quality-through-process movement and is still prescriptive in nature:

> "The People CMM is a process-based model which assumes that workforce practices are standard organizational processes that can be continuously improved through the same methods that have been used to improve other business processes." (Curtis, 2001, p.15)

Although PCMM acknowledges the importance of learning, and knowledge management in software development it makes little direct reference to organizational learning, or to the theories of Argyris and Senge, and therefore is not suitable for our purposes.

2.4 Application of an organizational learning paradigm

2.4.1 Can organizational learning be applied to software development?

The lens of organizational learning has been applied to the field of software development before (Ang, 1997, Cusumano, 1995, Argyris, 1977, Huysman, 2000, Robey, 2000, Edberg, 2001, Stein, 1996) and as long ago as 1971 Weinberg described software development as learning:

> "Specifications evolve together with programs and programmers. Writing a program is a process of learning - both for the programmer and the person who commissions the program." (Weinberg, 1998, p.12)

As has already been suggested, there is a learning process at work in the development of software. Coplien uses the term *knowledge* (1999) to describe the contents of the solution and application domain, this implies a learning process has occurred to create the knowledge. Elsewhere Coplien and Harrison (2003) draw parallels between their work on organizational patterns in software development and organizational learning and suggest they have observed triple loop learning during software development.

The Edberg and Olfman study looks at the relationship between organizational learning and software maintenance ("maintenance work performed to change an existing software system after that system has been transferred to its intended recipient" (Edberg, 2001, p.1)). This is particularly interesting because

> "it merges the technical literature on software maintenance with the managerially oriented research into organizational learning" (Edberg, 2001, p.9)

Edberg and Olfman observe that:

> "Existing research and practice views software as an expense incurred after development that should be contained through better development methodologies, better evaluation of system characteristics, better enforcement of programming standards; and more participation of users during initial systems development" (Edberg, 2001, p.1)

Their study finds that 40% of software enhancements had learning as the primary motivation - although other motivations could be also be important. When refusing enhancements the IS departments acted as a learning inhibitor. Software changes which were made could be seen as example of individual learning benefiting the group.

Ang et al (1997) considered the role of learning within the whole organization as an insurance company attempted to develop and deploy an IT system. Taking the view that IT deployment took the form of a change episode within the organization the researchers identified instances (p.332) of single and double loop learning.

Again, it was pointed out that IT can act not only as an enabler of organizational learning but as an inhibitor (p.331) because of the ability to freeze practices and prevent further change. The role of IT as a hindrance to organizational learning has also been considered by Gill (1995) who emphasises that IT is often used thin the ranks of middle managers - exactly the people who Nonaka (1995, p.127) identifies as knowledge creators.

Similarly, Stein (1996) identified opportunities and obstacles for higher order learning in organizations through the development and implementation of new systems. The role of developer as knowledge engineer, and their sensitivity to organizational issues were identified as critical success factors.

Although Ang et al were primarily concerned with the higher order learning of the organization as a whole they do, briefly, discuss the role of IT staff in the change process (p. 333). They identify four ways in which software designers and implementors can facilitate higher order learning within the organization:

- Developers have a legitimate reason to study and enquire into the operation of the business.

- Through technology the developers engage in the discovery of assumptions and mental models, and (because of their legitimacy) can question these assumptions.
- Involving end-users in the design process provides an opportunity to enroll users in the change and commit them to the changed environment.
- Introducing the new system freezes the change process in the new model.

Potentially this process may lead to the perceived "failure" of IT implementations. In the first instance developers have initiated a learning process, potentially this continues even when the developers withdraw, either because they advance to the next stage and finished user consultations, or because a "finished" system is delivered. However, the users to whom the system is delivered do not hold the same mental models which they held when the system was designed, and it is likely their learning continued after final consultations.

The developers discussed by Ang (p.333) are acting as enablers of learning, agents of change and even knowledge engineers. In introducing successful change good social skills are desirable, however, there is a general perception, if only anecdotal, that IT staff frequently lack good social skills. If true, IT staff acting as change agents are taking on roles for which they lack suitable skills.

The Ang paper demonstrates that the organizational learning perspective can be usefully applied to the IT environment. However, it sheds little light on how the software development process itself engages in learning.

Robey et al (2000) have suggested that researchers are only beginning to investigate the relationship between information technology and organizational learning. Robby identifies two streams of research. The first is concerned with the application of organizational learning in the IT environment, while the second is concerned with the use of IT to assist organizational learning.

This paper is concerned with the first of these streams of research which Robey suggests started with Agryris' 1977 paper - "Organizational learning and management information systems" (Argyris, 1977). Argyris argued that organizational learning theory could usefully contribute to the debate on the "software crisis." The 1977 piece appears to be the earliest attempt to directly link organizational learning and software development, although Argyris spends most of his time discussing MIS system design rather than the development process specifically. It appears this avenue of research has been ignored by the software engineering community.

2.4.2 How can we define organizational learning?

If we wish to study organizational learning it is necessary to define what we mean by *organizational learning*. Mullins provides a good starting point:

> "**Learning**: a change of a relatively permanent kind which may result in new behaviours and actions or new understanding and knowledge gained through a formal process or spontaneously and incidentally through life experiences." (Mullins, 2002, p.904)

> "**Learning organization**: An organization which encourages and facilitates the learning and development of people at all levels of the organization, values the learning and simultaneously transforms itself." (Mullins, 2002, p.905)

However, Mullins' brief definitions hide a lot of detail and a lot of debate, as noted by Cusumano and Selby:

> "Organizational learning is a very broad subject that appears frequently in recent management literature." (Cusumano, 1995, p.327)

To be sure, for their own study Cusumano and Selby take a pragmatic position:

> "We chose to interpret this concept in practical terms. Organizations have many opportunities to improve what they do: They can reflect on their operations, study their products, listen to their customers, and encourage difference parts of the organization to share knowledge..." (Cusumano, 1995, p.327-328)

The pragmatism of Cusumano and Selby is worth emulating, however, as with Mullins they associate the term "knowledge" with "organizational learning." This raises questions about the learning-knowledge relationship, something that has troubled Nonaka when considering knowledge creation:

> "In the accumulation of over 20 years of studies, they [organizational learning writers] have not developed a comprehensive view on what constitutes 'organizational learning'." (Nonaka, 1995, p.45)

Nonaka's criticism does not mean the field is barren only that it is difficult to define the edges of the field. One reason for this may be the division in literature identified by Argyris and Schön:

> "One branch of the literature - prescriptive, practice-oriented, value-committed, sometimes messianic, and largely uncritical - treats the phrase 'learning organization' as a catchword for whatever it is the ... front running organization are doing. The second branch ... treats organizational learning as a research topic

> for scholars, mainly in schools of management and business." (Argyris, 1996, p.xix)

The two branches are not totally disparate:

> "Both branches tend to pick up on ... recognzing, surfacing, critizing and restructuring organizational theories of action (... "mental models") ... [and] between single and double loop learning." (Argyris, 1996, p.xix)

Both branches of organizational learning, and the literature on knowledge management, is rooted in Penrose's resource based view of the firm (Pitelis, 1998). The belief being that knowledge created through learning constitutes one of the resources available to the organization.

As the above descriptions testify, the difficulty in defining organizational learning illustrates the multi-facetted nature of the subject. Therefore, any attempt to analyse a domain through this lens must also take a mutli-facetted approach.

2.4.2.1 Senge's view of organizational learning

Despite the nebulous nature of organizational learning the field has spawned many writers and researchers. Foremost amongst the writers is Peter Senge who has done much to further the understanding and practice of organizational learning through his book, *The Fifth Discipline* (Senge, 1990). Here Senge defines "five disciplines" which contribute towards the *art and practice of organizational learning*. Shown graphically in Figure 4 - the five disciplines are linked through the practice of reflection:

- Personal Mastery - individual learning and exploration.
- Mental models - i.e. recognising and overcoming.
- Shared vision - the creation and importance of.
- Team learning - beyond personal mastery team learning is the building block of organizational learning.
- Systems thinking - a call to think about the whole picture. (It is worth noting that for Senge the term *system* has no technological implications.)

Taken together Senge claims these disciplines will enhance learning and reduce learning inhibitors. Although not stated explicitly Senge's discussion of mental models includes the necessity of unlearning as described by Hamel and Prahalad (1996).

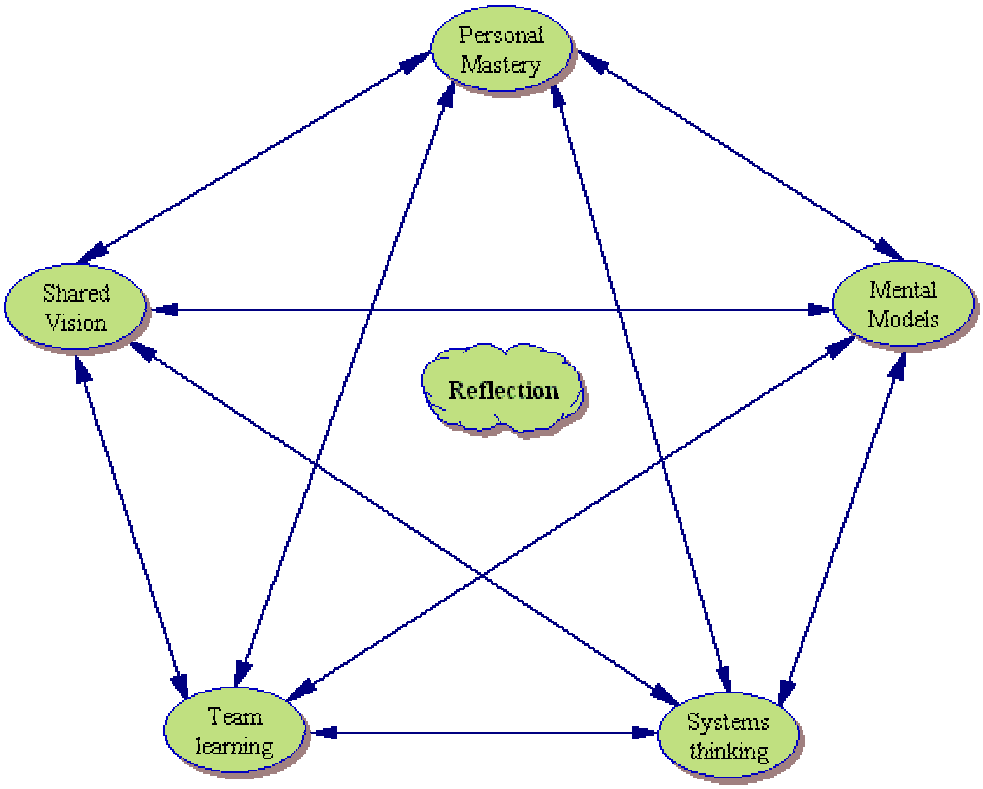


Figure 4 - Senge's five disciplines are closely related

Importantly for Senge, these "five disciplines" do not exist in isolation but should be seen in a holistic context. Although not cited as a discipline in its own right he advocates "reflection" as key to enabling these concepts. Reflection can be seen as a direct application of the inquiry principle advocated by Argyris.

In fact, Senge's work maps closely to that of Argyris:

- Personal mastery and team learning parallels Argyris' suggestion that for organizations to learn people must learn.
- Mental models and shared vision help highlight the values that Argyris talks about.
- System thinking and reflection are the tools of inquiry.

Beyond Senge's five disciplines much of his book is aimed at overcoming learning inhibitors and enacting the disciplines.

In the context of IT there is an interesting overlap here with Willcocks' (1997, p. 460) *Nine core IS [Information Systems] capabilities*:

- Leadership and creation of shared vision is at the centre of Willcocks' core capabilities
- Business Systems Thinking is Willcocks' second core capability, reporting on case study research Willcocks notes:

  > "The second front office requirement is to make an IS contribution to the top level business dialogue, ... The necessary skill was universally described as 'systems thinking' ... The CIOs in the study believed that the IS function was both the natural home and the breeding ground for systems thinking skills." (Willcocks, 1997, p. 486)

- Relationship building between IT/IS and business, between "techies" and "users" is an example of Senge's *Team Learning*, in this case the team is both the technical staff and their business customers.

Despite predating Senge by some 20 years Weinberg (1998, first published 1971) vividly illustrates the same principles within the IT context. The same themes of team working/learning, mental models, unlearning, learning inhibitors and shared vision are clear, albeit, often, in a different language.

2.4.2.2 Practise alone is not enough

Senge's disciplines clearly describe what we may characterise as the *practice based view*. However, it is important to recognise that merely practising these disciplines alone cannot define learning, there must be some result. The presence of such practices is not sufficient alone to characterise organizational learning. Argyris and Schön make the point:

> "the attribution of organizational learning is contingent on the presence of an observable change in behaviour"(Argyris, 1996, p.33)

Argyris and Schön point out that a change in behaviour does not necessarily imply that learning has occurred, but an observable change in behaviour is an a prior requirement for identifying learning in action. We may characterise the results of organizational learning as the *results based view*.

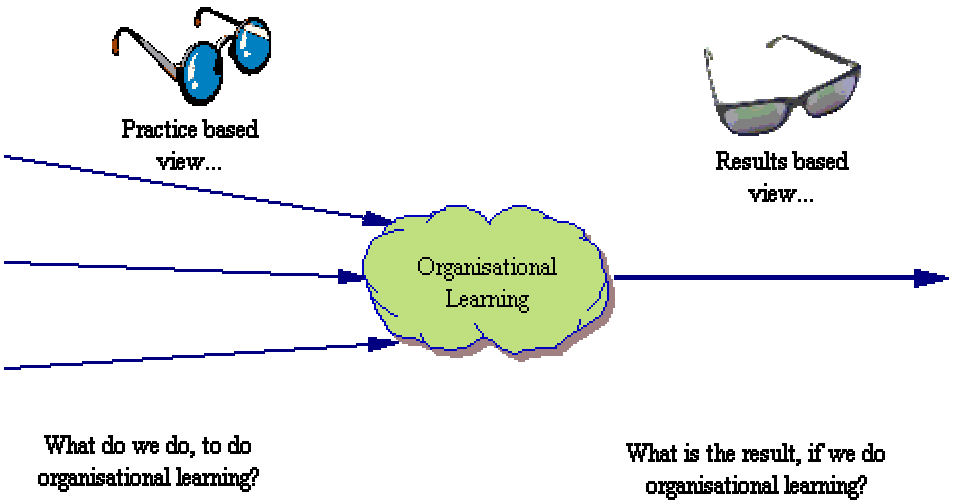


Figure 5 - Two views of organizational learning

The failure to act on information has been characterised as the *Knowing Doing Gap* by Pfeffer and Sutton (2000) who suggest that company culture and support are necessary is companies are to be able to "put knowledge into action."

2.4.2.3 Reconciling learning and knowledge

By making organizational learning contingent on behaviour change Argyris and Schön are attributing a sense of action to learning. This has parallels in the writings Nonaka who suggests:

> "knowledge, unlike information, is about *action*. It is always knowledge 'to some end'." (Nonaka, 1995, p. 42)

To be sure, Nonaka accepts that knowledge creation rests on learning:

> "From our viewpoint, the creation of knowledge certainly involves interaction between these two kinds of learning, which forms a kind of dynamic spiral." (Nonaka, 1995, p.44)

The "two kinds of learning" described by Nonaka are Argyris' single and double loop learning (Argyris, 1996). It seems that the gap between Nonaka's *knowledge creating company* and proponents of *organizational learning* is actually a difference of emphasis.

For example Nonaka's description of the development of the Mitsushita Home Baker bread making machine (Nonaka, 1995, p.95-123) is described as a knowledge creation exercise. The same example could be read as a case study of a *community of practice* (Brown, 1991) and the application of single and double loop learning.

We therefore suggest that in looking for the results of organizational learning we are in fact looking for the creation of knowledge. That is, an organization that is actively learning, is creating knowledge. Importantly, we place an emphasis on action resulting from the learning process and from the knowledge creation.

This is driven by a process of active inquiry which Argyris and Schön (1996, p.11) suggest underpins organizational learning, it is this inquiry process that generates knowledge. They go on to identify three type of inquiry which for them constitute organizational learning (p.20):

1. Organizational inquiry: learning aimed at improving the performance of the organization.
2. Inquiry aimed at redefining what it means to succeed and improve performance.
3. Inquiry aimed at enhancing the ability to practice the first two forms of enquiry.

Clearly, Argyris sees item 1 as an instance of single loop learning whereby an individual or organization improves its process through learning what works, and what doesn't. Items 2 and 3 meanwhile are example of double loop learning in which an individual or organization attempts to improve its learning process, this can lead to questioning values and assumptions which underlay the learning cycle.

Edberg and Olfman (2001) use the terms *exploitation* and *exploration* to describe single and double loop learning respectively. Single loop exploitation allows the organization to leverage what it already knows, while through double loop exploration the organization is able to generate more knowledge.

We can now reconcile the knowledge view of Nonaka with the learning view of Argyris. Figure 6 shows graphically how single and double loop learning, driven by inquiry produce a learning process, the result of which is knowledge creation.

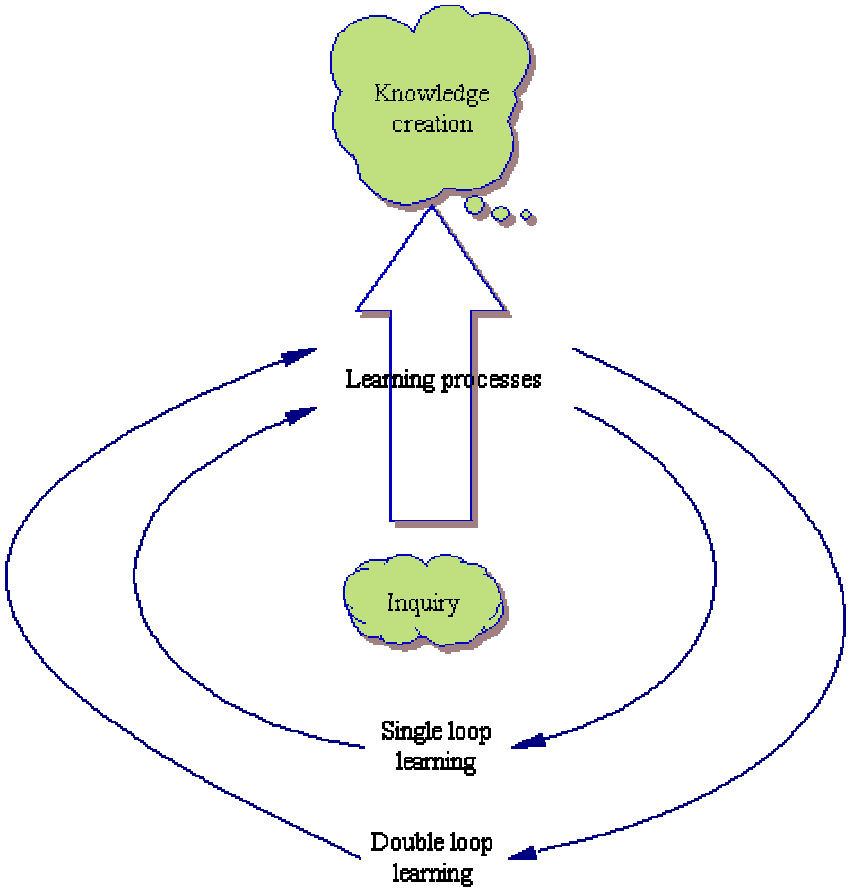


Figure 6 - Reconciling organizational learning and knowledge creation

However, even using this model, and taking Cusumano's pragmatic view there is still a need to identify the practices and results of organizational learning. Figure 6 is therefore incomplete, there is a need to include the practices and inhibitors of organizational learning.

2.4.2.4 Learning inhibitors

So far we have taken an active view of organizational learning, that is, how it occurs. Yet there is a second aspect, identifying why it fails to occur. Both Senge (1990) and Argyris (1994) discuss why organizations fail to learn - and by fail to change and adapt. For, Argyris these are defences to learning, while for Senge these are "learning disabilities". In either case it is clear that in many instances organizations fail to diagnose true problems or, even when the problem is know, fail to act on the information.

We may see two kinds of learning failure. Firstly organizations may simply fail to learn, that is, fail to recognise and act on a issue, or fail to see how a process may be improved. Often this is because there is no feedback and no opportunity to create a learning loop. Secondly, individuals or teams within an organization may know how

to rectify a failure, or how to improve a process but do not act on this information, i.e., the behaviour change noted by Argyris does not occur, creating Pfeffer and Suttons *Knowing-Doing Gap.*

Argyris defined organizational defences as:

> "a policy, practice, or action that prevents the participants (at any level of any organization) from experiencing embarrassment or threat, and at the same time, prevents them from discovering the causes of the embarrassment or threat." (Argyris, 1994, p.2)

Argyris goes on to discuss various forms of defences, including managers as inhibitors. Many of these defences can be traced to individual identity defence described in section 2.4.4.

The role of managers, in facilitating learning, and in inhibiting learning is a common one in the literature of organizational learning and knowledge management. In Nonaka's model middle managers are "at the very centre of knowledge management" (1995, p.124) and goes on to redefine middle managers as "knowledge engineers" (1995, p.151).

Senge is equally forthright on the role of managers:

> "Learning organizations demand a new view of leadership" (1990, p.339).

This new view sees managers as designers, and as vision creators.

> "managers must redefine their job. They must give up the 'old dogma of planning, organizing and controlling,' ... managers fundamental task ... is 'providing the enabling conditions for people to lead the most enriching lives they can." (Senge, 1990, p.140)

Failure of managers to see their new role may result in the destruction of knowledge and a failure to learn. In documenting learning inhibitors, and describing how "companies turn knowledge into action", Pfeffer and Sutton (2000) repeatedly emphasise the role of managers and their attitudes towards learning.

As we observed earlier, some have argued that the problem with methodologies is that they are not applied strictly enough, or are not detailed enough. Part of the classical role of the IT manager has been to police the methodology. Yet it is exactly this command and control mentality which can inhibit learning. In discussing Senge, Starkey says:

> "The key constrain, therefore, on the development of learning organizations is management skill ... we have to overcome our obsession with control and the notion that people are rewarded only for conforming to the rules of others rather than developing better rules. As external locus of control is a recipe for stasis and, in the long run, mediocrity." (Starkey, 1996, p.263)

However, the failure of management to recognise and encourage learning does not prevent it from happening. Brown and Duguid (1991), building on the work of Orr (1990), describe how the failure of managers to recognise learning by a workforce opened a gap between managers and workers. Managers believed that the company training programmes and "dircetives" were sufficient for reps (Orr's term) to perform their jobs. They failed to value the skills developed by the reps, instead they came to view some of the reps' practices as deviant. Meanwhile, the reps felt undervalued by managers and found company directives and instructions made their work more difficult.

The important point is that learning will occur whether it is managed or not. Good managers will work with the process rather inhibit.

2.4.3 The results of organizational learning

Identifying the results, the value added, of organizational learning is more problematic than identifying the core practices. One the one hand, for authors like Senge, the benefits are the practises themselves - these are so evidently beneficial there is no need to elaborate. On the other hand, it is difficult to distinguish between benefits stemming from organizational learning and benefits coming from other business initiatives.

To be fair, organizational learning does not exist in isolation. The practises of organizational learning, and the learning inhibitors, are embedded in the operations of the organization.

Notwithstanding these problems, authors do attribute some outcomes to a positive learning environment:

- Creativity and innovation are frequently linked to learning, for example:

  > "What characterizes innovative organizations? The answer is: they are highly effective at learning, self-critical and committed to continuous improvement." (Starkey, 1996, p.126)

- Experimentation and continuos improvement: learning is often seen as the fundamental element in process improvement, for example the *Kaizen* approach at Toyota (Delbridge, 2002, Spear, 1999). For such improvements to occur there must be experimentation.
- Problem solving is both means of learning, and the result of learning. Brown and Duguid (1991) describe this in the context of communities of practice.

To be sure, these results are closely entwined, one may even see them as a single result. The common factor is: change, each of these outcomes is the result of some change. As noted above (2.4.2.1) Argyris attributes organizational learning only when behaviour is seen to change.

2.4.4 Considerations of identity

Individuals, groups and organizations can all be said to posses *identity*. Since identity can be the basis of values, and since double loop learning may cause values to be questioned and changed there is a need to consider the role of identity in learning by individuals, groups and organizations. Brown and Starkey have considered identity change resulting from organizational learning:

> "the sort of organizational learning we are primarily interested in is that which constitutes a form of identity change. Our argument is that for an organization to learn, there must be an alteration in its participants' organizationally derived self-images. Organizational learning evolves through modifications, additions, and deletions of existing routines (Albert, 1992). These routines are, at least in part, constitutive of members' collective definitions of the organization's identity (organizational self-images) so that variation in one necessarily implies variation in the other (Dutton & Dukerich, 1991; Gioia & Thomas, 1996; Sproull, 1981)." (Brown, 2000, p.28)

Brown and Starkey also consider the role played by psychodynamic defences in defending the identity through resistance to learning. Again, these arguments may be considered at a multitude of levels from the individual, through the group or team and up to the organization.

Rotherman and Friedman (2001) also consider the role of identity in organizational learning but their emphasis is on conflict:

> "Rather than being an obstacle to learning, conflict offer opportunities for engaging in learning. Double loop learning is a form of conflict resolution in

> which organizational members inquire into the reasoning behind positions they take and the meaning of these positions for them." (Rothman, 2001p, 582)

For Rotherman and Friedman identity conflict can drive learning:

> "The analysis of these frames of conflict suggests that the identity frame may be more relevant to organizational learning than are the resource and interest framings because it promotes inquiry into the concerns and motivations of organizational members and learning." (Rothman, 2001, p.582)

In highlighting the role of conflict Rotherman and Friedman are, as they openly state, extending the work of Senge, Argyris and others.

Although the two pairs of authors may highlight different aspects of identity their works are complementary in nature:

> "Individual and organizational concepts of self are maintained by a variety of defenses that are engaged in order to avoid psychic pain and discomfort, allay or prevent anxiety, resolve conflicts, and generally support and increase selfesteem." (Brown, 2000, p.28)

The defences which interest Brown and Starkey function to resolve conflicts - advocated by Rothman and Friedman - at the expense of learning. In order to facilitate double loop learning it is necessary to overcome these defences and allow change to occur.

2.4.5 Software development as planning

It is possible to consider the successive stages of software development as process of planning, each stage represents a refinement on the previous stage. Each stage results in a more detailed plan than the previous stage.

We can reconsider the classical development methodology as a sequence of more detailed plans. First the project is defined, the feasibility study will add enough additional detail to decide if the project is doable. Assuming it is doable, a detailed analysis and system design will add extra layers of planning, so that the overall architecture of the system is laid out. The program code represents the most detailed plan possible as it is designed to execute the plan. Any omissions or errors in the plan will be revealed when the program runs.

The value of up-front planning is made strongly, and repeatedly, in the classical literature such as Pressman:

> "Myth: project requirements change constantly, but change can easily be accommodated because software is flexible.
>
> Reality: It is true that software requirements do change, but the impact of change varies with the time it is introduced. If serious attention is given to up front definition, early requests for change can be accommodated easily." (Pressman, 1997)

It is widely recognised that requirements change:

> "Stable requirements are the holy grail of software development. ... On a typical project, however, the customer can't reliably describe what is needed before the code is written." (McConnell, 1993, p.30)

But authors advocate a view that it can be done right first time

> "Studies over the last 15 years have proved conclusively that it pays to do things right first time. Unnecessary changes are expensive.
>
> Data from TRW shows that a change in early stages of a project, in requirements or architecture, costs 50 to 200 times less than the same change later in the construction or maintenance. (Boehm, 1988)
>
> Studies at IBM have shown the same thing." (McConnell, 1993, p.25)

For Pressman and McConnell the point of planning is to minimise change even though they accept that change will occur. The underlying assumption is that if change can be identified and limited all will be well. This relies on accurate planning and restricts experimentation and opportunities for learning.

A different view is taken by de Geus (1996) who suggests that planning exists in order to understand the future and incorporate change. Under this understanding change and the unforeseen are accepted, the purpose of plans is to explore how we may handle a range of events or requests. The objective is for the people in the process to learn.

The difference of opinion is highlighted when we look the stated objectives of planning:

> "The objective of planning is to provide a framework that enables the manager to make reasonable estimates of resources, costs and schedule." (Pressman, 1997, p.112)

However, for de Geus:

> "So the real purpose of effective planning is not to make plans but to change the microcosm, the mental models these decision makers carry in their heads." (de Geus, 1996, p.94)

Building on de Geus' work, Schwartz describes scenario planning:

> "Scenarios are not predictions. It is simply not possible to predict the future with any certainty." (Schwartz, 1991, p. 6)

While hard project planning may, or may not, be applicable when dealing with well known domains of activity, we can see that all three domains of activity (process, application and solution) in software development are constantly evolving. This suggests that the somewhat softer approach of planning as learning may be more applicable. Even so, Schwartz suggests why managers prefer the hard approach advocated by Pressman and McConnell:

> "Often, managers prefer the illusion of certainty to understanding of risks and realities. If the forecaster fails in his task, how can the manager be blamed?" (Schwartz, 1991, p.6)

This is uncannily like Middleton's conclusion when examining the highly planned world of SSADM methodology:

> "SSADM offers political protection to Civil Servants, should projects go wrong." (Middleton, 2000, p.97).

2.4.6 What organizational learning practices are important in software development?

Building on the work documented above and processes advocated from the software engineering community it is possible to identify several practices which may be performed in a software development setting to enhance learning. These are described below, while Figure 7 adds these practices and inhibitors to the earlier model of combining knowledge creation with learning.

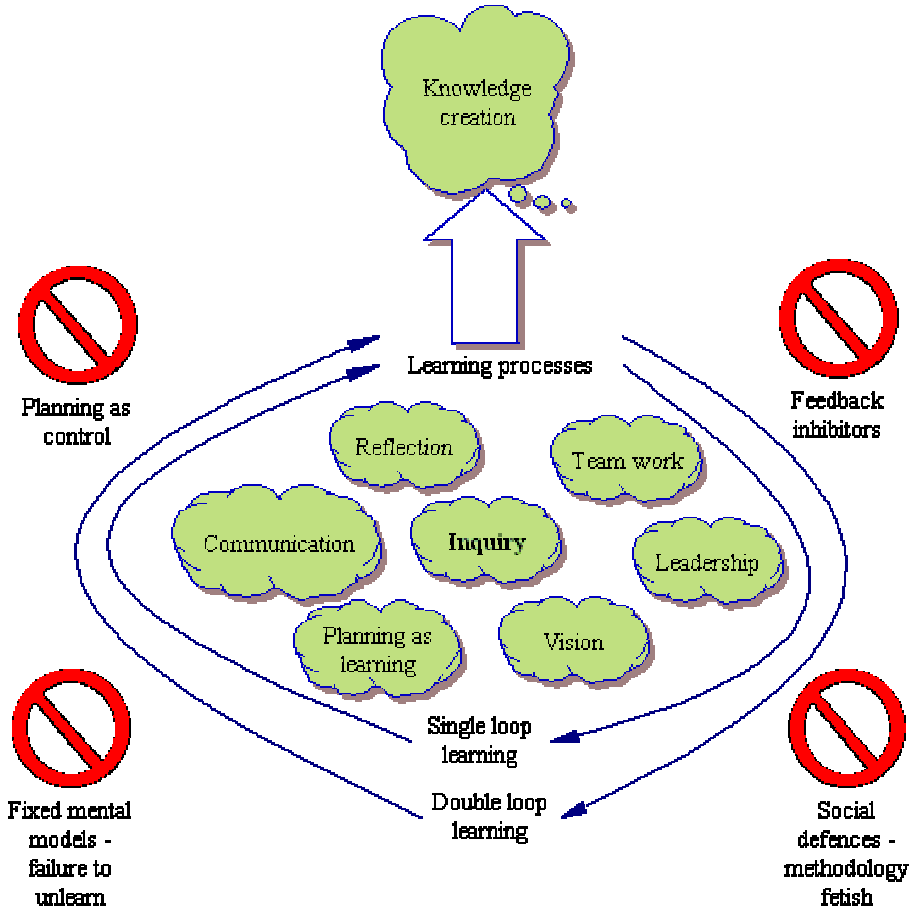


Figure 7 - Practices and inhibitors to learning and knowledge creation

2.4.6.1 Inquiry

The concept of inquiry is central to Argyris' definition of organizational learning, while all the practices outlined by Argyris, Senge and others may be defined in broad terms to constitute *inquiry*. Each one of the practices illustrated in Figure 7 contributes towards the practice of inquiry. It is possible to define each practice as an example of inquiry, although it is more illuminating to define each in its own right.

2.4.6.2 Reflection

The practice of reflection, as advocated by Senge, is the most direct explicit example of inquiry. Senge builds his model of reflection on Argyris "action science" principles:

> "Skills of reflection concern slowing down our own thinking processes so that we can become aware of how we form our mental models and the ways they influence our actions. Inquiry skills concern how we operate in face-to-face

> interactions with others, especially in dealing with complex and conflicting issues." (Senge, 1990, p.191)

There are several well-known practices in software development which may be seen as opportunities for reflection. For example, walkthroughs and code reviews (McConnell, 1993), project post-mortems/retrospectives/reviews (Cockburn, 2002) and the recent, albeit controversial practice of pair-programming (Beck, 2000, Coplien, 2003). However, most of these practices seem to be aimed at single loop learning.

Except for these specific examples the practice of reflection is notably absent in most of the literature examined. Notably, some of the more recent works (e.g. Cockburn, 2002, Eckstein, 2003) are starting to discuss reflection in its own right, while Louridas and Loucoploulas have suggest reflection can complement traditional design processes (Louridas, 2000).

2.4.6.3 Communication

When more than one person works on any project there is a need for communication. Where projects are technically complex, as they frequently are in software development, then the need for clear communication is greater; this is explicitly recognised by Pressman (1997, p.861). However, while classical literature such as Pressman focus on the means of communication and how technology can help resolve the problem, Senge would place the emphasis on communication between individuals as an enabler of team inquiry to take place.

The practice of inquiry and team working demands open communication. It would be impossible to practise team work, vision sharing and dialogue as advocated by Senge without open communication. Willcocks (1997) too has identified communication and dialogue as important in the realm of IT, but it is only recently that writers on software development (e.g. Cockburn, 2002, Eckstein, 2003) have placed greater emphasis on communication:

> "It seems to be a fact, that nearly no project fails because of the usage of a specific technology, tool or the like. The main reason for project failure is almost always the missing or not functioning communication." (Eckstein, 2003, p.22)

Similar comments are made by DeMarco and Lister:

> "The major problems of our work are no much technical as sociological in nature." (DeMarco, 1987, p.4)

And later:

> "The [software] business we're in is more sociological than technological, more dependent on workers ability to communicate with each other than communicate with machines." (DeMarco, 1987, p.103)

There are at least three audiences for communication during software development, in each case the communication is aimed at inquiry:

- Intra-team communication: between members of the development team. Here the inquiry should facilitate greater understanding and learning about the solution domain and the process domain.
- Extra-team communication: between members of the development team and their customers (i.e. end users and project sponsors). At a minimalist level this will serve to enhance the developers understanding of the application domain as they inquire into the problem.

  Ang (1997) noted this type of inquiry may also result in learning by the user; this may be an occasion for the user to reflect on their own practices and surface mental models and assumptions not previously expressed.

  In addition, extra-team communication will also occur were team members need to communicate with project sponsors, e.g. senior managers from outside the team. These different groups may demand different things from the developer, e.g. the customer may want more features while a manager wants a lower cost. This paradox has been explored by Howcroft and Wilson (2003) who consider the "two headed Janus" developer.
- Inter-team communication: on large projects were more than one development team is engaged there will be cause for inter-team inquiry to co-ordinate the teams. This can also be a source of conflicting demands as different teams have different priorities, requirements and processes.

2.4.6.4 Vision

The role of vision setting is explored by Senge who values its motivational attributes:

> "Where there is a genuine vision (as oppose to the all-too-familiar 'vision statement'), people excel and learn, not because they are told to, but because they want to." (Senge, 1990, p.9)

According to Conklin (1996), building of a shared vision was a key element of *Enrollment Management.* Using this model Digital Equipment Corporation (usually

referred to as "DEC" or "Digital") successfully developed and delivered on schedule, the Alpha AXP project - a combined hardware and software project involving over two thousand engineers for several years:

> "The program office uses vision to enroll the related groups in the goals of the program. ... [this] is most effective when it expresses the program's vision in the terms and language of the group being enrolled." (Conklin, 1996, p.55)

In an echo of Senge's words "because they want to", Conklin describes the power of vision:

> "given the group's commitment to the larger result, we found more aggressive behaviour. For example, the OpenVMS AXP group publicly committed to their target schedule and stated, 'We don't know how to achieve this, but we commit to finding a way'." (Conklin, 1996, p.59)

The vision is the motivation for the inquiry activities. However, the vision itself is also subject to inquiry but it serves to focus the inquiry towards some end.

2.4.6.5 Team work

Within the software development literature there is an keen awareness of the need for team work. Indeed, Ince and Andrews (1990) have written:

> "In 1968 the power of computers had increased to such an extent that programming was no longer a solo exercise. In the early 1980s the problem repeated itself but now it was microcomputers ... The evidence points to two main problems in software development: organizing a team of people to build a system, and actually knowing what that system is." (Ince, 1990, p.2)

Having identified team work as a problem the rest of the book concentrates on "technical fixes" with no further reference to team work. Similarly, other standard texts pay scant attention to the subject. Pressman's 800 page 1994 edition hardly mentions group working, the term "team" fails to be included in the index. The 1997 edition devotes a handful of pages to the subject but describes team organization in terms of management decisions based on quantifiable options. Somerville - another standard text - is slightly better, recognising:

> "Most professional software is developed by project teams ranging in size from two to several hundred people." (Somerville, 2001, p.497)

In fact Somerville devote a whole chapter (20 pages) to "managing people" including six pages to the subject of group working - albeit six pages in a 700 page book which is mostly concerned with technical issues.

Conversely, teams and communities take a central place in the literature on organizational learning. Not only does Senge talk about team learning but Brown and Duguid (1991) describe communities of practice, likewise McDermott (1999) advocates the development of knowledge communities.

Weinberg has discussed the need for software teams to develop themselves as well as products:

> "I now interpret 'work requirement' more broadly, to include the development of capability in the team and team members. Over the years I've observed that the requirement to develop capability cannot be adequately met by a single person. We learn much faster and much better with the active co-operation of others." (Weinberg, 1998, p.5.i)

Team working is a catalyst to inquiry. It allows problems beyond one individual's abilities to be tackled, and allows other problems to be tackled more efficiently. By advocating that teams inquire into their own capabilities Weinberg is calling for double loop learning.

2.4.6.6 Leadership

We have already discussed the role of leader as learning facilitator or inhibitor. Leaders also have a key role to play in the inquiry process by creation of a shared vision, establishment of a communicating, teambuilding and facilitating reflection. Senge's new view of leadership (section 2.4.2.4) calls for leaders to be teachers, designers and stewards, while Kolb highlights the need for managers to learn too:

> "Today's highly successful manager or administrator is distinguished not so much by any single set of knowledge or skills but by the ability to adapt to and master the changing demand of his or her job and career - by the ability to learn." (Kolb, 1996, p.270)

Frequently software development teams have distributed leadership. It is not uncommon to find the project manager, team leader and system architect are three different people. This provides an opportunity for conflict of values and vision.

2.4.6.7 Inhibitors

The review to date has identified a number of potential inhibitors to learning. It is useful to summarise these for clarity:

- Compartmentalisation of the development process blocking feedback.
- Existence of mental models which portray an unrealistic view of the development process.
- Defences, possibly linked to identity, by individuals and teams which prevent change and learning.
- Distributed leadership.
- Failure to unlearn existing mental models.
- Failure to act on known best practice.
- Poor communication.

Each of these inhibitors has the power to block the inquiry process and halt the learning processes, or restrict its full potential.

There also arises a process Argyris describes as camouflage (1977, p.114) were single loop learning produces a solution to a problem, however, the solution addresses the symptoms rather than the underlying problem, because the solution is now in place the actual problem is more difficult to diagnose. Senge describes the same issue as "shifting the burden" (1990, p.104).

2.4.7 Where to look for learning

Section 2.2 identified three different domains of considerations in software development. Although we would expect learning to occur across all domains it is worth considering the different forms learning will take in each.

2.4.7.1 Learning within the solution domain

The solution domain is concerned with the technology applied by developers to create a system. In order to be able to use this technology it is necessary for developers to learn the technology. This may occur through formalised training courses, books and self study or through socialisation and use of the technology - Brown and Duguid (1991) would call former *canonical learning* and the latter *non-canonical learning*.

As has already been noted, the pace of change and introduction of new technologies makes learning essential in this domain.

It is also necessary for developers to learn about the technology being developed for any particular application domain. In the first case there is a need for innovation as developers create the solution, secondly, as new developers join the team they need to learn what the original developers have created. There is a need for a shared understanding of what the development is. This has led Holt (2001) to suggest that software architecture is an example of a shared mental model.

2.4.7.2 Learning within the application domain

There are multiple examples of learning within the application domain. In the first instance software developers need to learn about the business need for software. Classically, this is done through a process of performing system analysis and writing a system specification. However, as we have already observed, as later stages of the system are developed this understanding will change as developers gain new insights. Fitzgerald points out:

> "methodologies [do not] allow for the learning experience and greater problem [application] domain knowledge that developers gain over time ... an idealised approach to system development as portrayed in a methodology may be seriously flawed since it omits the fact that failure is essential to human learning" (Fitzgerald, 1994)

As noted, it is not only developers who learn about the application domain. Ang et al (1997), suggested that the process may produce learning by the end-user/customers who are being studied for the specification.

Once a system is deployed there is occasion for users to learn how to use the system. Again, Ang et al have looked at this process and suggested there is a need for the organization as a whole to learn how to deploy the system to obtain the full benefit. Organizations which do not appreciate these diverse aspects may result in the system being considered a failure.

2.4.7.3 Learning within the process domain

As we have already seen, the classical view holds that without a strong process, methodology, software development may descend into simply "hacking" or "Mongolian Hordes".

However, we have also seen that by following a methodology developers can suffer from goal displacement. The methodology acts as a social defence which inhibits learning and becomes an end in itself, rather than a means to the end.

Rigorous use of the methodology may appear to solve problems, however, it is possible for methodology to act as camouflage, hiding the real issues. For example, change requests may be forced through a defined accept/reject process that may appear to reduce the number of requests but is actually stifling further learning by users.

2.5 Summary of literature review

The classical software development literature, based on methodological explanation offers a view of the process which while useful does not accurately reflect what actually happens during software development.

An alternative view of software development is offered in the literature of organizational learning. Indeed, writings about software development from the early 1970's (e.g. Brooks and Weinberg) can today be interpreted within the context of organizational learning.

This raises the prospect that organizational learning offers a better lens through which to describe software development than the classical literature. The question becomes: can we understand the development process through this lens? In effect, we are contrasting the technology based description against a social based description.

In order to explore this question more fully a framework of understanding is needed which will (a) offer improved understanding of the process, (b) highlight differences between the literature and practice.

2.6 A framework for exploring learning in software development

The work of Argyris can be grouped into three broad areas: single loop learning, double loop learning and learning inhibitors (Argyris, 1977, Argyris, 1994, Argyris, 1996). It is possible to view the practices advocated by Senge as examples of how to engage in single and double loop learning, and over coming defences against learning.

Building on Coplien (1999) we have been able to divide the software development environment into three domains: application domain, solution domain and process domain. In each of these domains there is a role for learning.

It is possible to combine these two models by asking what learning (or defences) occur in each domain? For example, we may consider:

- How writing a program specification helps a software developer understand the requirements.

  i.e. the role of single loop learning in the application domain.

- How the writing a program causes developers to question the required specification.

  i.e. the role of double loop learning in the solution domain.

- How adherence to methodology can inhibit an effective development process.

  i.e. how a mental model is used as a defence against learning and improvement.

Continuing this process, it is possible to produce the grid framework shown in Table 1.

	Single loop learning	**Double loop learning**	**Learning inhibitors**
Application domain			
Solution domain			
Process domain			

Table 1 - Framework for identifying learning in domains of software development

While useful for analysis this table risks falling into the same trap as the methodologies discussed before, that is, a belief that the field may be subdivided into smaller elements of analysis. To offset this it is necessary to add several questions to this table. These questions (Table 2) are designed to investigate the over-arching themes of vision, systems thinking, and team based learning/communities of practice that may be present.

4. Is there a clearly articulated, coherent, vision?
5. To what degree are team members aware of the "bigger picture" of the development?
6. Is team learning present?
7. Is there evidence of reflective inquiry?
8. What is the role of managers? Methodology police or learning facilitators?
9. What is the role of planning?

Table 2 - Framework questions for identifying learning in the overall software development process

3 Objectives and research methodology

3.1 Objectives

The objective of this research is to answer the research question:

"What insights can organizational learning offer into software development?"

This question contains two sub-objectives:

- To illuminate the software development process as a learning centred activity.
- To suggest ways in which software development may be improved through the application of organizational learning principles.

In answering the questions we need to create a new view of software development to contrast with the classical description.

3.2 Methodology

The literature review has described classical approaches to software development, the principles underlying organizational learning and shown that these principals may be, indeed have been, applied to software development in contrast to the classical approach. Finally, the review also presented a framework for considering these issues.

Through a cross sectional study comprising interviews with software developers and managers it is hoped to show how organizational learning occurs during software development. It is hoped that analysis of these case studies through the lens of organizational learning theory will (a) prove the applicability of these theories to the field, (b) identify opportunities to improve the process and (c) highlight areas for further research.

Since organizational learning is not the dominant paradigm within software development it is felt necessary to use qualitative studies to explore the domain rather than quantitative studies. Inevitably, such studies will represent cross-sections of practice at a moment in time.

3.2.1 Data collection and sources

Collection of data has been undertaken using a site visits and face to face interviews with members of a software development group within the organizations concerned. Interviews were semi-structured, firstly interviewees were asked basic

administrative questions to ascertain the scope of the development activity within the group. Next a set of open questions were asked to determine the degree to which a name methodology was in use. Finally, a set of open-ended questions were used to encourage interviewees to tell stories about the development process. Where appropriate additional questions were asked to provide deeper understanding of issues raised.

In developing and deploying software systems there are three generic models used by organizations: in house development of systems, outsourced developed by a third party and the purchase of existing *common off the shelf (COTS)* software. Were software is developed specifically for an individual business it is known as *bespoke* regardless of whether it is developed in-house or externally.

Therefore, there are typically three environments, in which a software developer may find themselves working:

- In house bespoke: Creating software for the business that employs them, e.g. Royal Bank of Scotland, Unilever, etc.
- Out-sourced bespoke: Creating software for a single, external customers, e.g. Accenture. (Often called a "Software house" or "Consultancy")
- Generic development: Creating software for sale to many customers, e.g. Microsoft. (Again may be called a "Software House").

Data collection has drawn on a mix of these sources. While developers frequently switch between these different types of companies the demands placed on them can be quite different.

Interviews typically lasted less than an hour. Once complete the interviews were transcribed by the researcher before analysis. On some occasions post-interview conversations added further insights into the subject matter after the tape was stopped, where appropriate these comments were noted at the end of the transcript.

Once the transcript was complete it was analysed for themes consistent with the framework outlined in the literature review. As noted in the literature review, organizational learning is a multi-facetted discipline, it was felt appropriate to note as many concepts as possible as these may provide avenues of further research.

The analysis was continued by way of a written paper. This was structured into three sections: how, within our framework, the subject matter could be explained, how classical software engineering texts may explain the projects, and a discussion section.

The transcripts and analysis are omitted from this paper but may be available from the author or his web-site in the near future.

3.2.2 Limitations of research

All interviews are drawn from the commercial business environment, while this environment represents the bulk of software development activity it does not represent the sole environment. It is hoped that the findings will generalise to all types of software development.

A second limitation on the case studies is that of commercial confidentiality. Some avenues were *off limits* to this research and it has been necessary to obscure the identities of the organizations concerned. Every care has been taken to ensure no substantive material has been omitted by these restrictions.

By their very nature the interviews were retrospective and personal. Where details were not offered there was limited opportunity to gather them from alternative sources, or observe how the events actually played out.

Since these were personal accounts interviewees inevitably offered their interpretation of events. The interviewees had already filtered the information received and used their own sense making process to understand and communicate the events. Where alternative sources were available some of the events recounted may have been interpreted differently.

Three of the five interviews were with former employees of the organizations described. In part this was a deliberate decision since it was felt former employees may discuss their projects more freely. In the case of Supply Chain Systems this was an act of historical necessity in order to gain an insight into an organization operating during the "dot.com" boom in Silicon Valley. While interviewing former employees could bias the research none of the subjects exhibited particularly bad feelings towards their employer.

Due to time constraints it was not possible to perform longitudinal studies with those interviewees involved in on going projects. This limits the opportunity to test some of the points made by advocates of process methodology and organizational learning.

These limitations were partially offset by interviewing a number of different individuals from different backgrounds. While each case exhibited its own collection of events, stories and interpretations common themes could still be identified.

However, one possible source of bias in reporting should be noted. Several of the interviewees, although not all, were contacted through a professional software developers group. Since these interviewees subscribe to the same publications and electronic mailing lists and may attend the same conferences it is possible that a group bias could be reflected in the research.

On the other hand, involvement with such a body may also indicate that the interview subjects are more aware of the issues involved in software development. In particular, membership of the said body could also be interpreted as indicating that these individuals already have a bias towards learning.

On balance, it is felt that participation in the organization concerned would not unduly bias the exploratory nature of the interviews.

Another similarity in the interviews concerns the application domains. Of the five subjects three are involved with supply chain software developments. Again this could introduce bias into the study although given other differences (e.g. location and company size) between the three this is felt unlikely.

3.2.3 Framework for exploring

The questions were based on the framework suggested by the literature review - section 2.6. Appendix A shows the questionnaire guide used for the interviews.

4 Research

4.1 Overview

A summary of the individuals and organizations investigated is contained in Table 3. All interviews occurred between July 2003 and August 2003 inclusive.

The semi-structured nature of the interviews allowed each interviewee to tell their own story of a software development effort. For the three interviewees who were no longer employed by the organizations concerned the story view was evident in their telling. Each story had its own, very clear, theme.

Interviewee	Company	Description
Jenny	Warehouse Software	A dedicated software company developing systems for warehouses and logistics. **A story of chaos**, and the failure of senior management to understand or engage with software development.
Tom	Hedge Fund Inc	A software development group supporting an Anglo-American hedge fund. **A story of identity**, an organization asserting its independence from the parent and in doing so bringing its Chicago and London offices into destructive conflict.
David	Bulk Mailing	A software development group developing applications for a mass marketing company. **Another story of identity**, software developers within a printing business assert their identity as IT people resulting in constructive conflict between technology and business.
Alistair	Supply Chain Systems	A Silicon Valley start-up developing online supply chain market tools. **A story of vision and identity**, a high-performing team arrive to save a floundering company and

		change the world.
Jack	Transport Corp.	The UK software application development group of a well know global transport and logistics company. **A story of classical software**, a chaotic development group is saved by the application of methodology and process.

Table 3 - Summary of individuals and companies investigated

By the very nature of qualitative research it is not possible to foresee the issues which will be raised by research subjects. This was undoubtedly true in this case, while some subjects chose to focus closely on the development process (e.g. David at Bulk Mailing) others chose to discuss the wider business environment (e.g. Alistair at Supply Chain Systems.) Both are equally legitimate points of view and highlight the large diverse nature of *software development.*

4.2 Interview thumbnails

4.2.1 Warehouse Software

Warehouse Software develops bespoke warehouse and logistics systems for organizations throughout the UK. This small to mid-sized company employs its own sales force, business analysts and software developers to develop specialist warehouse and logistics systems designed for customers specific requirements.

Although the company has a history of developing systems over a number of years the project discussed with Jenny suffered a number of major problems best characterised as a complete failure of management. The project was subject to frequent personnel changes - a result of the senior management's decision to operate a resource pool from which staff could be assigned to the project for a few weeks or a few months and returned to the pool, or another project, at any time.

The individuals responsible for management and leadership of the project failed in their roles. Although senior management was alerted to this fact on several occasions they failed to correct the situation. One rear intervention by management was to stop a senior developer when he started to fill the void and assume the leadership role.

Although the project team agreed a development process before the project commenced, this was never documented and remained a cultural entity. As personnel were rotated on and off the project the process became less well defined.

The project experienced what has been characterised as a "requirements explosion" with frequently changing requirements coming from the customer and generated by the development process itself. This made efforts to create a stable technical design and perform planning difficult or impossible.

Customers and users were separated from the development team physically and organizationally. Communication of requirements was handled through business analysts who were responsible for passing requirements and changed from users to developers as documents. Again, these analysts were subject to personnel changes.

Jenny, a software designer, clearly identified a failure to capture the requirements up front as a key failing on the project that resulted in this explosion. She also identified unrealistic deadlines as another cause of failure, deadlines that the project usually missed.

The project seems to have lacked leadership and a clear sense of direction, staff were demotivated and knew they would shortly be moved off the project. This situation seems to have been accepted and the project allowed to continue floundering, and failing, without any serious efforts by management to improve the situation.

4.2.2 Hedge Fund Inc

Tom was the deputy head of IT in London at a Hedge Fund subsidiary of a major investment bank. Software development was split between London and Chicago and was primary concerned with supporting traders. Most of the development function was concerned with rapidly producing "prototypes" for the traders to use. Developers would work closely with traders on short projects. These projects might be initiated by traders themselves or through senior management.

The development of these small projects was quite successful but when the fund decided to develop a new equity risk management system problems set in. Although the fund already had such a system this had been inherited from the parent company which still controlled elements of the system. The original designer of the system was now employed by the fund and it was he who led the development of the new system.

The new system was developed in Chicago, drawing on the existing system and the traders' knowledge. The system was seen a success and it was decided to roll it out in London, this is where problem started.

Although the traders in London were engaged in the same role as their counterparts in Chicago their methods of working were quite different. The developers in Chicago had made assumptions based on the US market and US ways of working which were not applicable to the London market, e.g. the system only supported dollar pricing.

The IT department in London made a series of change requests to the developers but these were either not enacted or enacted slowly. Meanwhile, the developers continued to add features and functionality for the Chicago operation.

This situation persisted, the more the system was seen as a success in Chicago the more the senior management wanted the system in London, but in London the system was seen as a failure and the traders wanted their old system back. Each time the Chicago developers added a new feature, or failed to enact a change request from London the view of failure was compounded.

Finally, the deadlock was broken by the cutbacks in the banking sector after September 2001. The Chicago traders kept the new system while London was allowed to use the old system.

4.2.3 Bulk Mailing

David is a developer with Bulk Mailing, an established firm specialising in the printing and posting of mass mailings. David leads a team of three, who are developing a new data processing system that allows the firm to receive data from customers and produce the mailings. Although the system is still being development it is slowly replacing an existing system based on older technology.

In general Bulk Mailing has little interest in IT. David's project is unusual and reports to a manager who is responsible for two production centres. Even so, David is keen to define himself as a software developer and looks forward to working for a real software company, one dedicated to producing software alone.

The project is vaguely defined, although its objective is to replace the existing system this is far from a systematic process. New developments come from three sources: suggestions, ideas and insights provided by the manager in charge of the process, end user comments and the developer themselves. While the first two

sources are driven by the business the developers seek to apply new technologies to the project, although, only when there is a justifiable case.

There is some resistance in the company from users of the old system who feel they are being deskilled. However, part of the reason the company wishes to replace this system is the difficulty it has in recruiting these skills.

The manager of the project is not familiar with the IT being applied by David's team, this is a source of tension for David who feels the manager is constantly setting unattainable deadlines. Although, the manager's unfamiliarity also means he leaves David and his developers much latitude in how they work and design their technical solutions.

Design and development is centred on a democratic dialogue process whereby David and his team discuss solutions and make ad hoc notes of the decisions rather than formal documents and design diagrams. There is much reworking ("refactoring") of existing work as better solutions become clear, or time permits "kludged" code to be improved.

David is keen to learn new technologies and development techniques, not just for himself but for his team too. All three developers seem to a similar skill level to him. When required (and finances allow) they engage outside consultants to assist them with new and more complex technologies.

At the time of the interview the team were completing a major piece of work which has improved the company's competitive position relative to its competitors. While the team now want to spend some time improving the existing system (refactoring code, removing bugs, improving stability) there is tension with the manager who wants the team to add more new features as soon as possible.

4.2.4 Supply Chain Systems

Supply Chain System was a Silicon Valley based start-up during the 1998-2001 "dot.com" boom period. The company hoped to provide an online mechanism for tracking capital flows as goods moved through the supply chain. This was part of a bigger objective to create a tradable market in working capital debt.

The company first developed a large Oracle based system for its purposes. However, it was found difficult to adapt this system as new customers were added; the system was unable to cope with the massive variance in supply chains used by different organizations.

Alistair's line manager had already moved from his current company, Browser Corp, to Supply Chain systems and encouraged Alistair to move. Subsequently, the whole of Alistair's team from Browser Corp was reassembled working for Supply Chain where they developed a second, more flexible system that implemented a domain specific language. Interestingly, prior to working for Supply Chain the team members had limited knowledge of the supply chain domain.

In developing the new system Alistair encountered resistance from existing developers. This centred on a number of technical issues that Alistair regarded as irrelevant or even counter productive, to the work and design of his second-generation system.

The Browser Corp team knew each other well and worked closely together. The team had a disciplined and an unwritten process, although Alistair is clear, there was a definite process to the development he felt that to write it down would destroy it. The teamwork and process relied on extensive and constant communication, usually via e-mail or instant messenger. However, the team found it difficult to integrate new individuals into the team and to work with other teams. Eventually the existing developers at Supply Chain left the company.

Development on the new system went well - Alistair describes it as pretty much a perfect project - and the team saw themselves as "saving the company" but the company had other problems beyond the new development.

The founders and original management had spent much of the initial venture capital funding on doubtful schemes. A large part of this spending had been on outside consultants to make the original system work. The system had been through several revisions already and much money spent on licensing and integrating third party solutions in an effort to make the system generic.

Eventually the company simply ran out of money; with the post-dot-com climate in Silicon Valley new money was not forthcoming.

4.2.5 Transport Corp.

During the 1990's the UK operations of Transport Corp won several national awards for quality and customer service. They are the subject of several positive case studies on continuous improvement and employee training practices.

The UK applications group develops small applications to support many internal groups within the company and applications for external customers. Bespoke

software is often provided to customers as part of the product offering by the sales team.

Until two years ago the applications group was seen to be failing. Internal customers had lost confidence with the group and would develop their own ad hoc solutions, the group had no reliable process and could not delivery software on time, the process was considered "chaos."

A new manager assumed responsibility for the group and tasked the lead developer, Jack, with rectifying the situation. He instituted a number of small-scale modifications himself, secured funding for external training to help the group and eventually brought in an outside consultant to assist with the change process.

The outside consultant adapted a commercial, off the self, big-M, Methodology known as RUP to the group's requirements. This has been in full operation for a few months now and is yielding considerable benefits. Jack credits the process with a turn around in the group's working, it can now deliver software on time and within budget.

The *big-M Methodology* adopted by the group contains many of the characteristics of classical methodologies with an emphasis on process and documentation. However, the methodology also adopts many evolutionary practices suggested by *Agile methodologies*. Consequence the process provides for considerable dialogue between customer/users, business analysts and developers. The use of iterative development provides for early testing and customer feedback.

Jack demonstrates Senge's *Personal Mastery* - without knowing it by name - although he finds it difficult, and possibly frustrating, that other team members remain locked in out dated mental models of the process and work. He is successfully using new technology and software design to challenge some of their assumptions and practices.

The group faces a number of other challenges. Their workload is set to increase, although some work is outsourced Jack is finding it difficult to identify third party development companies that are willing and able to work to the group standards.

While the new process has succeeded in bringing order to the previous chaos it is too soon to tell whether the process itself is responsible for this improvement or whether, as DeMacro and Lister would suggest, the improvement is due to Hawthorne effect. The biggest challenge facing the group is therefore to retain the benefits of the process.

4.3 Use of the framework

The framework described in section 2.6 proved useful in identifying learning practices and inhibitors. A composite summary of the major themes is given in Table 4 and Table 5. However, this strict framework does not completely describe the full insight offered by organizational learning.

	Single loop learning	**Double loop learning**	**Learning inhibitors**
Application domain			
Jenny:	Scope of project was defined. BA document customer requirements. Detailed information gathered on specific areas, e.g. goods out. Customer expanded the domain of project[1].		Development staff are separated from customers. Failure to recognise tacit knowledge. Goal displacement as staff produce specifications. BA's communicate by e-mail.
Tom:	Business requirements are written. Company enhances view of risk management beyond that of parent (a potential source of competitive advantage.) Features added to system in Chicago. London traders learn "the system is broken."	Managing directors and traders, discuss business opportunities and IT applications with IT staff[1]. Prototyping solutions[2]. Engaging business staff to review/reflect on prototypes.	Chicago developers unaware of London practices. Chicago developers assume mental models apply to London markets. Chicago developers erect defences against criticisms from London, do not revise their mental models. Goal displacement.

			London traders defend themselves against failure of new system.
David:	Company learns AIX is expensive, COBOL programmers are rare. Developers learn about bulk mail processing. Test bed highlights errors for fixing.	Generalisation of existing COBOL system. Manager envisages new ways of using technology to improve process. Users make change requests.	COBOL teams are outside the loop.
Alistair:	Developers learn about supply chains.	New system is capable of modelling any chain.	Management fail to understand their business problems.
Jack:	BA's document customer requirements.	BA's produce remit and specification with lead developer. Customers shown work in progress.	Some customers not interested in progress,
Solution domain			
Jenny:	Document review process.		Review process inoperable.
Tom:	Chicago developers learn "London is unhappy" Chicago developers learn "Chicago traders are happy"		
David:	Developers teach themselves new	Developers seek ways to exploit new	Time pressures.

	technologies. Developers learn from consultants. Test bed highlights errors for fixing.	technologies. Developers re-work existing solutions.	
Alistair:	Existing system is adapted for each new customer. Consultants integrate each customer as one off exercise.	Developers generalise about supply chains. Developers add inheritance and templates to new system to make it more flexible.	Existing developers erect defences to new developments.
Jack:	Skills training. Group learns to improve work estimates.	Architecture & technology questions mental models. Development manager applies outhousing[3] practices to internal team.	Developers resist technology change.
Process domain			
Jenny:	Process is agreed before project start. Staff learn to avoid the project.		Managers believe people are "moaning". Senior managers refuse to become involved.
Tom:	Team members are encultured into the process.		Mental model "lean, agile" image for company. Architect controls system, fails to empower staff; "second system effect"

David:	Creation of formalised system for raising queries. Developers learn manager always wants things "now" so deadlines become meaningless.		Manager has mental model of project management, does not allow for more exploration. David believes formal processes are better.
Alistair:		Rapid communication. Team members know each other, can introduce conflict to seed discussion.	Team has difficulty integrating new members and teams.
Jack:	Customer do not trust development department. Customer develop own software. Continuing to iron "blimps and kinks in the process."	Customers learn about others' priorities. Code reviews are form of reflection. Refactoring.	Mental models of developers' roles. Past management allowed "chaos." May come to see process as "complete." Process handbook could be a block to change and defence.

Table 4 - Composite summary of major loop learning activities and inhibitors found during research

[1]There is a negotiation and discovery process here, as business ideas are proposed, the business and IT staff then discuss possible uses of IT to address the business opportunity. Neither side has enough knowledge of the other domain to propose a complete solution.

[2]Prototypes are used as transient objects to expose underlying mental models, tacit requirements and cultural norms.

[3] "Outhousing" is the term used by Willcocks (1997) to described practices used by outsourcing companies when used internally with in-house teams.

1.	**Is there a clearly articulated, coherent, vision?**
Jenny:	No - Now attempt was made to create a shared vision, instead a vision of "unrealistic deadline" filled the void. Individuals goals and visions were not linked to the overall objective.
Tom:	Yes - "develop new equities risk management system"
David:	Yes - "replace the COBOL"
Alistair:	Yes - Alistair has a personal vision (architect his first large system); there is a team vision ("save the company") and a wider company vision (revoutionaise the supply funding), all three visions are complementary.
Jack:	There is a business vision which developers can relate to; there is a personal vision of "quality"; but some developers resist vision.
2.	**To what degree are team members aware of the "bigger picture" of the development?**
Jenny:	Very limited - Staff only join project after sale has been made, staff are rotated frequently and tend to focus on specific, limited areas.
Tom:	For local projects the picture is clearly visible. For the equity system the picture in London is not as clear.
David:	Very clear - system is to replace COBOL application, team can see clear benefits to the organization.
Alistair:	Completely; Alistair is able to articulate, passionately, the broad vision of the enterprise.
Jack:	Analysts and developers have clear understanding of how their work fits into the business, how software effects the business.
3.	**Is team learning present?**
Jenny:	No - staff are changed frequently.

Tom:	Yes - both single and double loop; some is positive, some is negative.
David:	Yes - development team is learning the application domain from their manager and users, and are also engaged in enhancing their own solution domain knowledge.
Alistair:	Yes - within the development team.
Jack:	Team have learned a new process, they are continuing to learn the new process and challenge existing models. Jack and the business analysts seem to have more "team spirit" then the development team as a whole.
4.	**Is there evidence of reflective inquiry?**
Jenny:	No - all time is spent dealing with immediate problems.
Tom:	No
David:	Manager sees software being developed and this acts as a catalyst to create new ideas the system and surface tacit knowledge. The developers reflect on the work they have done, consider better solutions and how new solution domain technologies may improve their application domain products. There is some evidence that some users are reflecting on the software delivered and requesting improvements and modifications.
Alistair:	Yes - in analysing the solution domain. However, this is limited within the wider company.
Jack:	Yes - Jack is engaged in a process of reflection and inquiry, however this is not true of all team members.
5.	**What is the role of managers? Methodology police or learning facilitators?**
Jenny:	Neither - management fail to manage.
Tom:	Manager do not seek to enforce any methodology. While they do not seek to facilitate learning, they are partners in the learning process.
David:	Application domain expert, project sponsor, customer.

	The manager seeks to *manage* the project like any other, driving his developers to produce a working system in as short a time as possible.
Alistair:	Visionary.
Jack:	Senior managers are learning facilitators.
	Jack is policing the new methodology and facilitating learning.
6.	**What is the role of planning?**
Jenny:	Goal displacement for manager.
Tom:	Limited and conducted on an ad hoc basis.
David:	Minimal formal planning through project plans and documented system design techniques.
	However, the team are actively engaged in planning, building new mental models of how the software will operate. These are built verbally, on a white board and using unstructured "book" documents.
Alistair:	An information channel and means of exploration.
Jack:	Planning, through the writing of documents is extensive. It is used as a control mechanism to determine what to implement and create schedules.
	The documents explore the requirements of customers and apply technical knowledge to the proposal of solutions. As such they are a learning vehicle which allows the application and solution domains to be charted.
	Documents continue to evolve as the project progresses.

Table 5 - Composite summary answers to framework questions observed in research

5 Discussion

In the cases of Bulk Mailing and Hedge Fund Inc the business lead is clear, the developers work for the business and are engaged to address business problems. The case of Warehouse Software is less clear cut. The company is engaged by a second firm to undertake work. Developers are removed from direct contact with the end-users.

Warehouse Software are a pure software company, in effect a pure knowledge based company. They exist to sell expertise in developing software for a specific part of the supply chain. One would therefore expect them to have core competencies in both the software development field and the logistics field yet they fail on both counts.

This problem appears again, although from the other point of view at Transport Corp where the development group is having difficulty in identifying third party developers with whom they can work.

Like Warehouse Software, Supply Chain Systems should, and does, possess a core competency in software development. Although their sales models differ the two companies have much in common. Neither company would exist without software, they both aim to sell software, albeit packaged very differently.

Each one of these companies demonstrates a conflict between business and technology. Management of the business, management of the technology and management of the conflict are three distinct but interlocked domains of management. Success in one is no guarantee of success and even the definitions of "success" and "failure" are open to question.

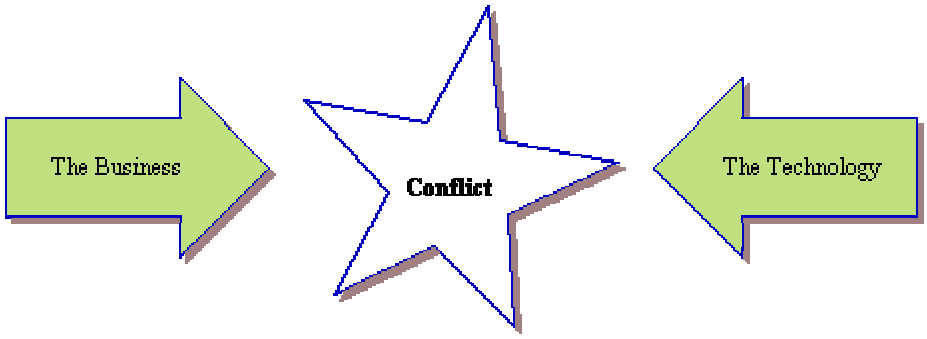


Figure 8 - Business and technology are in conflict

As a consequently of this conflict it becomes difficult to consider technology, in this case software development, without also considering the business environment. To

consider one side alone would not present a complete picture. Therefore, the original intention of this research, to remain strictly focused on the development process, was compromised.

This compromise was implicitly hinted at in the literature review. Many of the reviewed papers connecting software development with organizational learning considered learning by the organization rather than within the development team and process.

Conversely, the literature on software development reviewed mainly concerned itself with software development in the abstract, without a business context. And thus ignored the conflict between business and technology.

While the remainder of this discussion section concentrates on the software development side of this conflict - thereby offering a symmetry to the classical development literature - the central conflict is recognised and discussed.

The conflict itself of great interest. The management of this conflict seems to be a key differentiator between success and failure - however we may choose to define these terms.

5.1 How does the classic view emerge?

Of the companies examined only one, Transport Corp, had a formalised methodology as advocated by the classical literature. Indeed, this company had recently employed a consultant to advise the development process and write a formal methodology. The consultant had customised a big-M, branded, Methodology - RUP from IBM. The Lead Developer credited the methodology with an improving the development environment.

At Warehouse Software, the development team had agreed on a process, a small-m methodology, at the start of the development. However, unlike Transport Corp, this was not written down and was passed culturally from developer to developer. A management policy of rotating staff on the project, and the stress of continually changing requirements meant that this process was not rigorously followed.

None of the other three companies had a formalised methodology although they all had a process that was repeated and passed on culturally. The development team at Supply Chain Systems actually brought their process with them from their previous company but the team architect felt that to codify the process would destroy it.

Clearly, all the teams had a process, which produced software - with the possible exception of Warehouse Software. However, the practice of formalised methodologies as described in the classical texts was absent in all bar one case. This leads us to conclude that a formalised methodology is not a prerequisite for developing complex software.

Despite its absence the "pull of methodology" exerted a strong attraction on many of the developers. When faced with a failing project at Hedge Fund Inc, Tom believed the answer lay in a more rigorous, defined process. David at Bulk Mailing stated that he hoped in future to work for a "software company" which would follow a defined methodology. Of course, it is hardly surprising that developers educated with textbooks which advocate methodology should believe that possession of a methodology would improve their work.

There is also a clear analogy between a computer program and the claims made for some Methodologies. A program contains a series of unambiguous steps which when followed produce a definable results. A Methodology seems to offer a similar set of unambiguous steps, which if followed programmatically should produce a successful computer system.

Combined these, highly rational, models offer an idealised view of the development process. This can act as a motivator for software developers and their managers to aspire to. However it is questionable how successful such models are when implemented in real life, certainly DeMarco and Lister attribute most of the benefits observed to Hawthorne effect, although they acknowledge a marginal benefit from convergence provided by the methodology (DeMarco, 1987, p.113-120).

Transport Corp seems to be realising a third benefit from its adoption of a Methodology, namely that of change motivator. In adopting their Methodology Transport Corp set out a clear vision of what they wished to achieve, since the vision seemed rational, and appealed to developers sense of "what should be" thereby easing the transition from a troubled process to a more defined process.

At the time of the study Transport Corp has successfully made the change and development seemed to be better. The question now arises whether the team will see a gradual loss of benefits as Hawthorne effect wears off, or whether the benefits are real.

The loss of productivity would most likely take one of two forms. First, the discipline needed to follow the Methodology may decline, individuals may start to

miss elements of the process or take "short-cuts." Over time the group may return to their former state.

The second form may come from the reverse phenomenon, that over rigid adherence to the Methodology. Developers may come to use the Methodology rules as a form of defence to hide behind. This has been observed in other situations by DeMarco and Lister - who label it "malicious compliance" - and by Watsell (1996) who labels it "fetish of technique."

If we assume that Transport Corp's Methodology is a success, and the organization can navigate its way between the two opposing hazards it is worth asking if any of the other groups studied here could benefit from a classical approach to process.

Given that Alistair described Supply Chain Systems as a "near perfect project" there seems little that a revised process could improve. Similarly, although David of Bulk Mailing is drawn to a formal development process in reality his team is successful. Indeed, imposition of a methodology on either of these teams may well reduce their productivity and quality.

Hedge Fund Inc is a more complicated situation. The company successfully develops a myriad of small systems already and considers itself a "lean and agile" organization, the problem only lies with the large Equity Risk System. Since the Chicago office considers this project a success it would be somewhat difficult to persuade the developers to change from their current methods of working. Adoption of a new Methodology could be counter productive, although some means of convergence between London and Chicago would be useful.

At Warehouse Software the situation is more clear-cut. It would seem any kind of process would be better than their current way of working. As at Transport Corp it may be possible to leverage developers existing assumptions of how things should be done to install a new way of working. Coping with Hawthorne effect would be a minor problem compared to their existing problems.

It seems that the classical view of software development has some merits. While some of these merits may be a self-fulfilling prophecy they can be leveraged for benefit. However, inappropriate application of classical approaches may also be damaging.

Unfortunately, classical literature does not provide the tools of analysis needed to determine when, where and how, to apply the process dominant view. To be sure, a strict interpretation of the literature almost condemns any project to failure: process

may not exist, process may not be followed in a disciplined fashion, documentation is not produced (for both process and product) or adequate planning is not performed.

Therefore, there is a need for a framework that can be used to analyse software development teams and processes to determine when to apply the classical prescription, and when to leave well alone.

5.2 What aspects of organizational learning do we see?

This section draws directly from the Table 4 and Table 5 described in the research section. Rather than consider all nine categories individually this section focuses on the learning aspects across all three domains of development.

5.2.1 Single loop learning

5.2.1.1 Training and documentation

Single loop learning is much in evidence in these case studies. In its simplest form this takes the form of technology training courses at Transport Corp. Elsewhere, little mention was made of training courses, indeed, according to David, Bulk Mailing were reluctant to send him and his co-workers on training courses which he felt were required.

Where training was discussed it was focused on the solution domain - the technology, or development process. No mention was made concerning training about the application domain - the business. In each case developers seemed to be expected to learn the application domain "on the job" through socialisation and enculturement.

Three of the firms (Supply Chain Systems, Warehouse Software and Hedge Fund Inc) essentially exist to exploit specialist knowledge. That they rely on casual training processes and have little by way of formal knowledge passing seems surprising at first.

Several explanations present themselves. Firstly, this may simply be oversight by management, however, since all the companies had a similar, perhaps implicit, policies simple oversight looks unlikely.

A second explanation seems to a belief that a project should, and will, document what it finds in the business environment. For a rational business this should be unnecessary, in a business operating in a textbook fashion one would expect to find

operating manuals, documented methods of working, etc. Indeed, one would also expect to find training programmes for new recruits.

The absence of these materials in many businesses poses problems to the software development group. Without these basic materials to train developers, and on which to base system designs, it is difficult to begin any development. This explanation may also explain why classical development literature places such emphasis on documentation.

In classical literature the answer to these problems is system analysis, requirements analysis and documentation. In effect, the software development process provides its own materials to boot-strap the development effort. This process provides an interface from the supposed rational world of software development to the apparently irrational world of real business. Consequently, a software development project addressing a business problem could come to represent the most authoritative codified source of information on a business.

(One could see business process reengineering (Hammer, 1994) - BPR - as a extension of this activity. The rational world of technology attempts to push back and rationalise, through process reengineering, the irrational world.)

Another reason for the missing process manuals and induction training may be difficulty in producing these materials. Since much of the knowledge of an organization is tacit and difficult to codify, businesses simply don't bother to. It is only when a software development project starts that there an attempt to codify this knowledge. However, tacit knowledge is still difficult to codify and this may explain the problem of producing stable requirements described by several of the interviewees. Failure to recognise that much knowledge is tacit may impair efforts to produce requirements documents.

A more radical explanation for the missing materials may be that they don't matter, or at least, the advantages gained by the documentation process are outweighed by the disadvantages.

The contrast here is between Warehouse Software and Supply Chain Systems. The former organization was experienced in developing within the application domain and emphasised documentation. Yet neither of these two experience or documentation seemed to provide an advantage, and indeed, may actually have hindered the project. There is some evidence that Warehouse Software's customer in this case was unusual, yet developers continued to operate their existing mental models of the application domain thereby failing to appreciate differences.

Meanwhile, an emphasis on documents to communicate between customer, business analyst and software developer made the transfer of tacit knowledge difficult.

Conversely, the development team at Supply Chain Systems were highly successful despite the fact than none of the developers had a supply chain background. These developers were unencumbered by mental models and set out to learn the new application domain. Rather than use documentation as communication the organization provided "domain experts" within the company with whom developers could discuss issues with, thus the flow of tacit knowledge was unimpeded.

Finally, the very newness of the application domain to the Supply Chain Systems developers may have acted as an additional motivator - the desire to learn about something new by a high performing team contrasts starkly with the Warehouse team who were repeating a previous exercise. Undoubtedly both teams were facing new problems which required new solutions, the Supply Chain team were constantly operating in learning mode - everything was new to them - while the Warehouse team switched between learning/solving and repetition.

5.2.1.2 Software embeds knowledge

Single loop learning is also present when the organizations studied sought to embed knowledge in software. Bulk Mailing understood how to remove duplicates and "dead names" from mailing lists but this was time consuming and expensive. By embedding this knowledge in a software program the process was brought in-house, speeding it up and reducing costs.

Similarly, Hedge Fund Inc sought to embed their understanding of equity risk in a computer program so the knowledge could be more widely used and potentially form the basis of competitive advantage.

These findings support the suggestion of Edberg and Olfman that software enhancement allows the dissemination of learning:

> "The case studies in this research demonstrate practical examples of individual learning transferred to the group through software enhancement. A contribution from this demonstration is the concept that the learning of an individual can be disseminated to a group through enhancements to software thus making the process of software enhancement an act of organizational learning." (Edberg, 2001)

5.2.1.3 Model reinforcement - deadlines

In several instances here single loop learning was reinforcing, or even creating negative mental models. One recurring example was that of deadlines.

Several of the interviewees reported how deadlines were frequently missed. However, the missing of a deadline had become so routine as to be uneventful. In the most extreme case, David at Bulk Mailing describes a 12-month project as a success even though a few moments earlier he had stated that it had taken nine months longer than the original three-month estimate.

> "**David**: We've been bogged down with the D-ROT project now for 12 months, it was meant to be a three month project and now it is nearly 12 months. It is coming to an end so we will see what happens next.
>
> ...
>
> ***Researcher****: That was quite interesting about the project. You said it took 12 months instead of 3, but everything you said a moment ago was in terms of success. Most people who define a project that over ran by 9 months on a 3 month schedule as not successful.*
>
> **David**: I hadn't thought about it like that before. That's quite interesting.
>
> ***Researcher****: So, nobody in the company has raised this as an issue?*
>
> **David**: See, I always judge something as success based on whether it works or not. I don't suppose I'd really thought about the time scales.
>
> I had thought about time scales but we're always being told that it is taking too long."

David was not alone in describing how managers set unrealistic deadlines, nor was he alone in tuning out deadline dates and manager's exhortations to meet the date. It appears that both developers and managers are stuck in a mental model that is reinforced by single loop learning - shown in Figure 9.

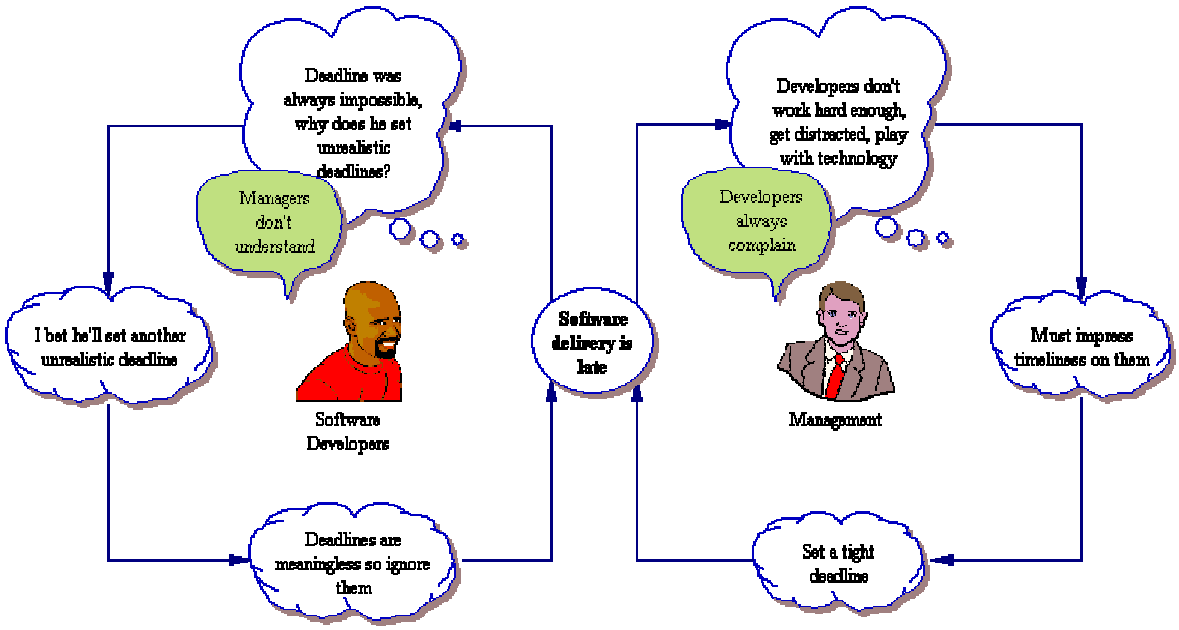


Figure 9 - Single loop learning reinforced mental model of deadlines

Nor is this view confined to the subjects of this study. Glass reports similar attitudes elsewhere:

> "on this consulting job ... I made a heavy, pointed pitch for schedule relief. I said things like, 'The approach you are taking to accelerate schedule is actually costing you long term, in that enhancements and system testing are taking such unpredictable long periods of time that it is not possible to achieve any anticipated schedule'." (Glass, 1998)

Senge would identify this as an example of *Shifting the burden* (Senge, 1990, p. 104). If we believe Glass this is *the* problem in the software industry:

> "So what's the bottom line? That schedule problem is the most overwhelming problem of today's software organization, overriding almost any other problem you might think you have; the problem is culturally ingrained ...
>
> let me add one more bottom line: Schedule pressure is the most serious problem facing software projects today. There is no silver bullet answer to the question (...) The only approach working within the problem is old-fashioned communication." (Glass, 1998)

5.2.1.4 Technology as goal displacement

Another example of single loop learning reinforcing a faulty mental model was observed at Supply Chain Systems. In this case the model centred on goal

displacement where technology was seen as a solution to the problem - show in Figure 10.

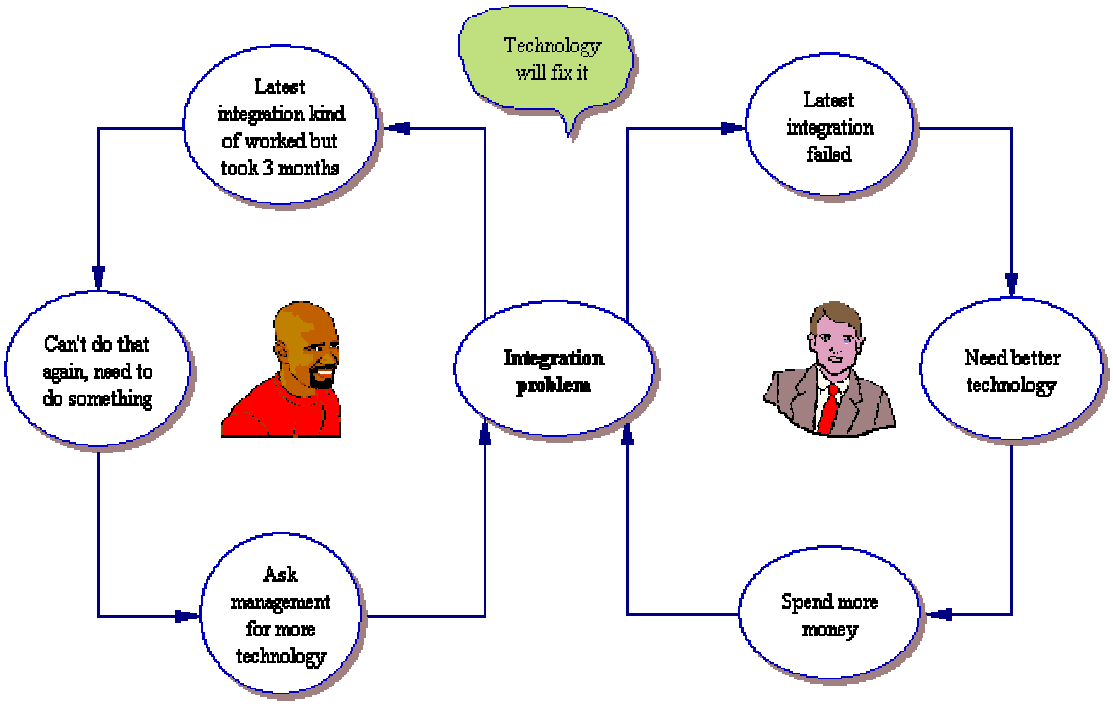


Figure 10 - Single loop learning reinforced goal displacement

Although the developers single loop was eventually broken managers continued to see the solution as technological in nature. Management belief in a technical solution to their problems blinded them of the need to reform other aspects of the company.

For Supply Chain, success in breaking the mental model of developers, albeit largely by replacing them, led to success for the software project, while failure to break this loop for management led to the failure of the company.

5.2.2 Double loop learning

5.2.2.1 Involve all customers, spread the knowledge

Transport Corp has an innovative prioritisation procedure. Faced with far more work than the development group can handle the lead developer, his manager and their internal customers hold fortnightly prioritisation meetings. Each customer makes a case for their project to be priority number one. The IT staff stand back and allow the customers to decide the prioritisation order. The customers debate their relative business needs and potential revenue generation from their projects before finalising a priority order from which the development group will work.

In the interests of fairness customers have an additional option. If they feel their project has been unfairly treated they may draw up a business case and seek funding from the company for external development - the development group will manage such development.

In prioritising the projects diverse parts of the organization learn about other parts of the organization. They become aware of other groups requirements, problems and opportunities. Thus, not only does the development group resolve its scheduling problems but the company as a whole increases in self-awareness and engages in boundary spanning learning.

5.2.2.2 Second systems

Perhaps surprisingly, three of the companies studied were in the process of developing computer systems to replace existing systems. Although motivations for doing so were varied no organization was writing a direct replacement. Each replacement embedded new ideas and features that could only be derived from seeing an existing system, or, through the actual process of developing afresh.

Bulk Mailing operations were already automated by a system written in COBOL and running on an IBM operating system and hardware. The company had noted that COBOL programmers were becoming more difficult to hire, and more expensive so set about replacing the system, this time writing in C++ on Microsoft Windows with commodity hardware. Rather than write a direct functional replacement they also set about generalising the system so that tasks which previously required programmers could be performed by operators.

Even here the process was not a direct feature translation. There was no list of existing features to be rewritten and generalised. Most work originated from the project manager. He was familiar with the existing system and process, his insights led to requirements. Many of his insights came not from the existing COBOL system but by seeing how the new system developed. For this manager the existing system, and the evolving system served as learning tools.

The developers too contributed to this process. While some of their insights and suggestions came from the two systems they were also motivated by the technology. The emergence of new technology served as a trigger for them to reflect on their current activities. New technologies led them to question the best way of encoding and presenting a solution.

While the developers at Bulk Mailing were replacing a system that had been running well for some years the developers at Supply Chain Systems were replacing a system which was only months old. Again, the existing system served a learning tool. Through its development the organization had been able to explore the application domain. In doing so, the system seems to have created more questions than answers; these questions mapped out the space that the new system would need to fill.

The third case of system replacement occurred at Hedge Fund Inc. Unlike the two previous examples where new developers developed a new system, the new Equity Risk System at Hedge Fund was designed by the same person as the original.

This case is difficult to explain because it is perceived as a success in Chicago and a failure in London. It appears that while the developers did learn from the past and did produce an improved system for Chicago they were also blind to the requirements of London. In part this is explained by what Brooks (1995a) calls "second system effect", the tendency to design the second system with embellishments withheld from the first system.

The knowledge gained from developing the first system, and from seeing it in use, was undoubtedly useful to the designer in developing the second system. There seems a clear case of learning in action here - probably both single and double loop. However, the designer's education had not encompassed the London office, he had failed to learn lessons here and what he had learned in Chicago blinded him to variations.

5.2.2.3 Refactoring and rework

> "Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure. It is a disciplined way to clean up code that minimizes the chances of introducing bugs. In essence when you refactor you are improving the design of the code after it has been written." (Fowler, 2000, p.xvi)

At least two of the organizations studied practised refactoring of code. The motivation for refactoring comes from several sources: code reviews conducted by other team members, personal reflection on the code written and changes in wider system and technology which cause the developers to reflect on their existing code base.

Refactoring seems to be a case of reflection and double loop learning. In refactoring developers acknowledge that their first attempts may not be the best possible solution. Even it was once the best possible solution other elements of the system may have changed making this part deficient.

However, refactoring also seems to be the source of tension with managers. In this study David echoes Fowler's descriptions of tensions which arise when management do not accept the case for reworking. Again we see a business-technology conflict.

Jack at Transport Corp is using code reviews and refactoring to directly challenge his developers' mental models about how they write code and approach business problems. Here the tension described by David and Fowler is reversed but is used to good effect.

5.2.2.4 Monkey see, monkey think, monkey learn

Several of the interviewees describe how customer describe their requirements to software developers or business analysts. The developers then engage in discussion with the customer as to the technical possibilities, this may lead to the construction of some software or further discussion. Tom at Hedge Fund Inc described the process:

> "Normally the traders would approach us with something, we'd go away and think about it, then come back to the business with what we thought we could do and how long it would take. Normally the things we were doing we could prototype within a few days. So we would do that and show it to them, and they would say 'Yes this works in some respects, not in others, it's not showing me what I need, etc.'. Then we would have an open discussion on what could be done."

This itself represents a form of double loop learning and knowledge combination. The business knowledge of the traders and the technical knowledge of the developers is combined, through dialogue, to explore the application and solution domain. Both sides question what is required, how it may be undertaken and what resources are needed.

At Transport Corp. the same process operates but in a more structured way. Initially the business analysts explore the application and solution space with the customers and developers. The findings are codified in an options document. As the process

proceeds more documents are written exploring the space in more detail as both sides learn what is required and what can be produced.

Eventually the requirements reach the development of program code. By this stage the requirements have been elaborated into a number of stories known as "use cases" (Jacobson, 1992). Developers implement a number of use cases in each development iteration. At the end of each iteration there is the opportunity for reflection, Jack reviews the developers code, the business analysts tests the program and the developer reworks the necessary sections. When this is complete the customers are shown the software developed so far and given the opportunity to reflect, command and request changes.

5.2.2.5 Developers seek to use new technologies - Personal Mastery

One problem which are largely absent from all the case studies was technical difficulties. This may be surprising to the casual observer who would expect these "high-tech" projects to suffer difficulty with technologies. Even when prompted the interviewees could cite few technical problems effecting their work. These seems to support the views of Eckstein (2003) and DeMarco and Lister (1987) noted in the section 2.4.6.3 that project failures are sociological rather than technological in nature.

This however does not mean that software developers do not learn and use new technologies. Indeed, all the interviewees demonstrated a considerable degree of what Senge would call *Personal Mastery*, in fact two talked directly of the desire to learn:

> "So on a personal level I want to learn all the time. If I'm not picking up new things then I'm not happy, so I'm always thinking up new things to make generic." David, Bulk Mailing

It appears that technological change was a driver for these people to learn. As new technologies and techniques emerged these people wanted to know about them. Further, they reflected on the technologies and considered how they could be used in their work. In many cases this led to new insights into how to overcome business problems. New technologies and the learning of them, acted as a trigger for reflection and inquiry.

Here again we see the conflict between business and technology. When managed well business can harness technological change to bring about business change and improvement. When mismanaged there is the capacity for firms to become

outdated, or IT staff to use technological change as a defensive mechanism to resist organizational change.

Such technology as social defence was evident at Supply Chain Systems. When Alistair proposed his new solution to the problems afflicting the company and development group he was met with a barrage of technical queries. The existing developers attempted to undermine his solution by demanding that particular technologies were used even where there was no particular need for them.

Jack of Transport Corp also spoke of the difficulties in persuading some developers to embrace new technologies and accept change. While those interviewed here may have demonstrated personal mastery and a desire to learn this is clearly not universal among development staff. Several of those interviewed spoke with despair about development staff who refuse to learn new technologies and techniques.

5.2.3 Learning Inhibitors

Several learning inhibitors have already been noted above: faulty mental models, single loop learning reinforcing such models, the failure to recognise the role of tacit knowledge and the erection of social defences. However, other learning inhibitors were noted which are worth of comment.

5.2.3.1 Separation

Separation appeared in three forms: separation from other teams in the organization, separation from customer/users and separation from managers. Some of these separations were physical while others were purely psychological, taking the form of ingrained mental models. Where separation occurred dialogue became difficult and problems usually followed. The separation also meant that the organizations were not able to leverage their full capacity to address issues.

Every one of the five organizations considered had more than one software development team. However relations with the other teams always seemed to be problematic. At Warehouse Software and Bulk Mailing the second teams had erected barriers between themselves and the teams documented here. Thus, the teams under consideration were deprived from the knowledge and insights of the other teams.

At Hedge Fund Inc problems occur when the London team needed to work closely with the Chicago team. The separation between the two offices extended to the users, the more Chicago perceived the project as a success the greater the separation

from the London team who viewed it as a failure. The inability to bridge this divide finally led to the cancellation of the project in London.

The failure of the old and new teams to integrate at Supply Chain Systems also ended unhappily. The older development team gradually left the company until the new team dominated.

Separation also occurs between developers and customer/users. Again Warehouse Software represents the worst case scenario, the development team find it increasingly difficult to meet the customer requirements. Likewise, it is the failure of Hedge Fund Inc's Chicago developers to consider the requirements of the London traders which creates problems.

Supply Chain Systems offers an interesting solution to this problem. Here the team were developing a generic system which would be sold to many customers. The team did not have access to potential customers because they had yet to be sold the system. The company resolves this problem by employing *domain experts* who act as proxy customers.

The proxy customer even has two advantages over a real customer. Firstly, the proxy customer and developer work for the same organization, so there is no need to hide information from one another and their goals are closely aligned. Secondly, the proxy customer is always available for consultation over a problem, unlike a real customer were access is limited to defined meetings.

There is another problem introduced by separation of developers from customer, that of fairness. Kim and Renee (2003) have suggested that there must be fairness in a process if managers are to secure commitment from employees in knowledge based companies. For example, one of the central points of contention at Hedge Fund Inc is the perception in London that the Chicago developers are not acting fairly.

It may be possible to extend this argument to customers - particularly internal customers. The prioritisation process at Transport Corp is seen to be fair by all participants. Supply Chain Systems side-step this problem by using proxy customers when deciding requirements and features.

Finally, there is a separation between management and the development teams. This seems to be the most difficult conflict to manage. Senior managers at Warehouse Software were repeatedly told by development staff that the project was in trouble, they were told the project lacked leadership and stability yet they refused

to act. Similarly, the senior manager at Supply Chain Systems continued to believe the technology team alone would save the company and failed to address the wider issues in the company.

In both cases it appears that senior manager erected their own defences. At Warehouse their mental model may be labelled "Developers complain" while at Supply Chain the mental model could be labelled "Technology will save us."

5.2.3.2 Keep teams together

One of the biggest challengers faced by the development team at Warehouse Software concerned the worker pool. In an effort to increase the utilisation of developers the senior managers enacted a worker pool system. All staff were managed by the Programme Manager who assembled teams dynamically as and where staff were needed.

When not required by the project staff were returned to the pool and could be assigned to other projects. When the project hit trouble staff would be quickly reassigned to the project.

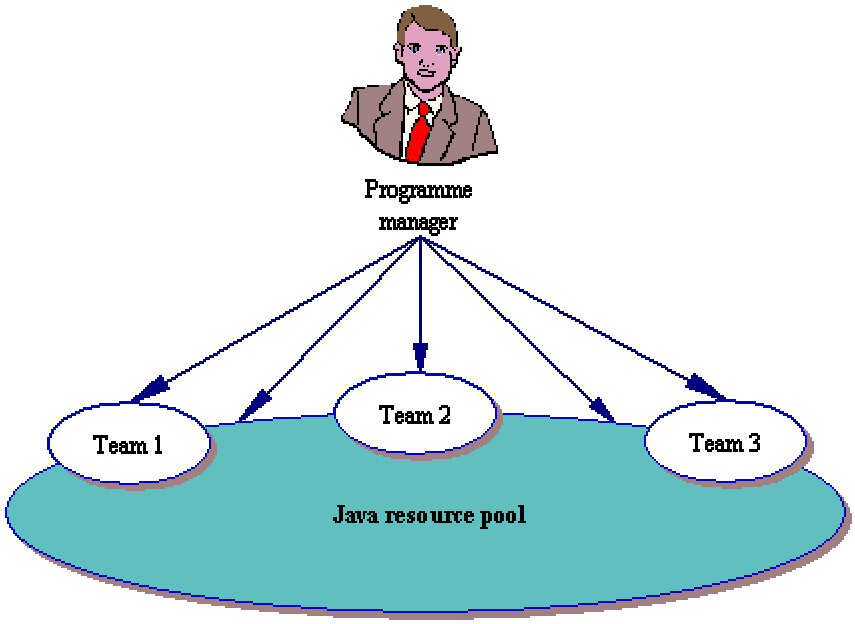


Figure 11 - The Resource pool at Warehouse Software

Consequently projects had few permanent staff to maintain the overall vision and design. Staff assigned to the floundering project knew they could be transferred off the project at any moment. Not only did this make it difficult for the project to learn but it also removed the incentive.

The pool method could hardly be more different to that of Alistair's team at Supply Chain Systems. Initially Alistair and his team worked for Browser Corp in Mountain View. Alistair's line manager, moved from Browser Corp to Supply Chain Systems to become Vice President of Software Development. She in turn recruited Alistair who proceeded to reassemble the whole team working for the new company in Mountain View. Alistair was quite clear about this:

> "I basically recruited only people I knew. That's how teams really work well, is if you have established personal relationships."

Problems occurred for Alistair when it was necessary for the team to expand or work with other teams. New recruits to the team found it difficult, though not impossible, to integrate. While at Browser Corp the team had been required to work with a team based in France acquired when Browser Corp bough a French company. Problems integrating the too teams, aggravated by time differences were eventually resolved when the French developers resigned. The situation which was to repeat itself at Supply Chain Systems.

Alistair's team has all the hallmarks of a *high performing system* (Vaill, 1996), however, such teams present their own challenges.

5.2.4 The role of identity

At the start of the research identity was not considered to be a major force in the field. However, identity issues and assertion of identity appeared again and again. Identity was apparent in discussions concerning teams, vision and leadership. Teams which hold shared vision also share an identity since the vision forms part of the team identity. The stronger the vision the stronger the identity. This is articulated by Alistair:

> "We went in to save the company."

And later:

> "all of the executives saw us as total saviours that would get them out of the shit."

The team identity and vision are clear, not only to the team themselves but to the executives at Supply Chain Systems. Whether these executives were right to invest so much faith in the team is another question but the vision is clear.

There is an important role for leadership in this context. Although the team already existed they were re-purposed by the management of Supply Chain Systems,

primarily the Vice President. The very act of bringing the team from Browser Corp will have enhanced identity, while presenting the team with a "mission" allowed a new vision to complement the identity. Alistair too shows leadership in the way he relates to his team, allowing them to share the vision and build the identity.

Elsewhere, at Bulk Mailing, we see creative conflict through identity assertion. David asserts his identity as a software engineer by learning new technologies and rethinking solutions within the engineering frame of reference. This brings him into conflict with his manager who asserts his identity as a manager by pushing for new functionality, new features, faster delivery and questions the need to reengineering solutions which work.

Again we see the business-technology conflict. This time the conflict is identity-based and mediated through communication, inquiry and reflection. Indeed, we can see identity as central to teams, vision and leadership, while the questioning process works to modify these themes - shown in Figure 12.

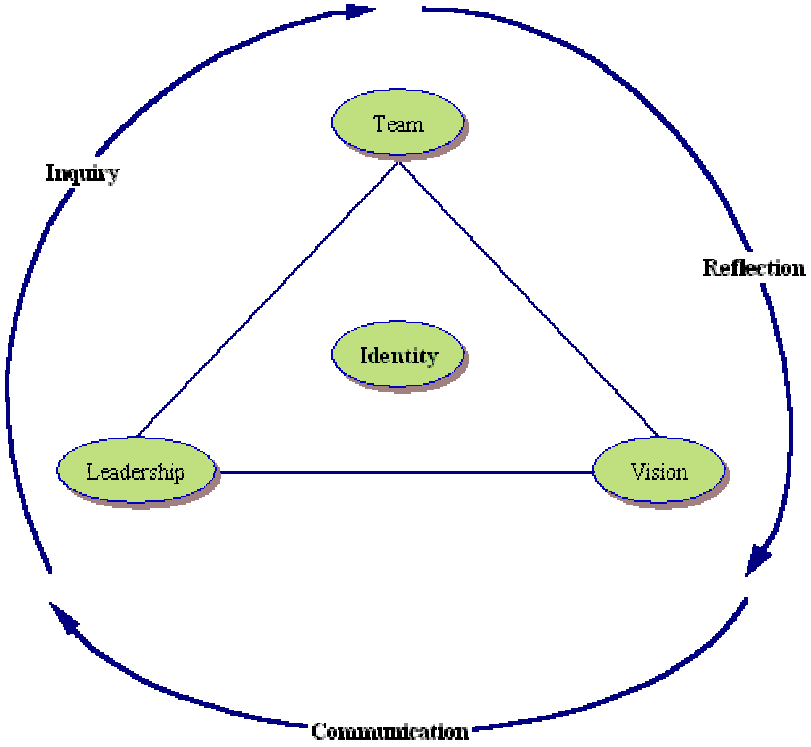


Figure 12 - Identity is central to teams, vision and leadership while inquiry, relfection and communication revolve around these themes

Such conflicts are not always managed well. Jenny at Warehouse Software commented:

> "I know for a fact that Brian tried to manage the project and was told not to do it. People were restrained from stepping in and managing the project, so it didn't have leadership."

Management's reaction to Brian can be conceived as an ego defence. To have allowed Brian, or any other software developer, to manage the project in the default, would represent an acceptance of management failure and a threat to management identity. Had Brian succeeded and the project environment improved this would only further have questioned management and developer identities.

Identity assertion is also apparent at organizational level. Tom recalls how Hedge Fund Inc came to develop a new system:

> "**Tom:** ... One particular project which was to replace the equity risk management system. The company had formed by being spun out from a big organization and had taken many of the IT systems with them, including a new equity risk management system.
>
> At some point people in Chicago decided they could write one which was better. Our IT team in Chicago decided they could do a better job.
>
> ***Researcher:*** *This requirement came from the IT side?*
>
> **Tom:** I think it came from the business side, they wanted to be free from influence from the original firm. They didn't want to be dependent on the parent organization.
>
> A second motivating force was they wanted enhancements and they wanted them quickly. If they remained dependent on the parent they would never have been able to get them. They wanted the application to go in a different direction to the one it was going in from the parent organization."

Although the decision to develop the new system is stated rationally there seems to be an underlying need for the subsidiary to assert its identity as distinct from the parent company.

When problems emerged with the system the company is unable to resolve them until external events force them to do so. Rationally it should be possible to resolve such problems without external stimulus. However, in this case the company is not simply dealing with technical problems but is dealing with issues of business strategy and the very identity of the company; combined, these make resolution more difficult.

5.2.5 Practices of organizational learning

5.2.5.1 Inquiry

There is clear evidence of inquiry occurring in these studies. This occurs in the technology, business and personal domains. As already observed, many of the interviewees were personally motivated to inquire into new technologies and techniques. However, this is not a universally held attribute of software developers, the absence of inquiry in some developers is a source of concern and tension with those who do possess the attribute.

In the business domain there is both a basic, single loop, inquiry when developers simply "learn about the business" but there is also evidence of a more complex, double loop, form of inquiry where the values and reasons for development are questioned. This was very clear at Bulk Mailing and Supply Chain Systems.

Inquiry seems to operate well where both developers and non-developers, specifically managers, are involved with the inquiry. For example, we see development and managerial staff combining to question current processes and combine their knowledge in the search for improvement. However, where one side alone practises inquiry tensions arise, for example, at Warehouse Software developers questioned the values behind management decisions (or lack of decisions) but when these insights were offered to management they were ignored. This probably fed developer's disillusionment with management.

The business analysis stage of system development is by nature an inquiry process and can be quite systematic in nature. This form of single loop inquiry can itself become a defence mechanism:

> "For about two or three months [the business analysts] were doing the requirements for goods out. I was trying to get them to give a high level view of how this would work. They spent about two months going to the low level. ... Eventually I got involved and we got together with the customer and the people doing the design and requirements, and we sat in a room and put together diagrams. That needed doing three months earlier." Jenny of Warehouse Software

5.2.5.2 Reflection

Although no interviewees specifically mentioned the practice of reflection, or indeed described any attempt to openly engage in reflective practices, there was evidence that some reflection was occurring. Since three of the interviewees were

no longer involved with the projects concerned they had had time to reflect on the events, how much reflection occurred inside the project is impossible to say.

Several processes were observed which constituted a form of reflection, or a trigger for reflection. For example, code reviews could allow developers to reflect on one another's code. Where iterative development processes employed developers and customers-users had an opportunity to reflect on the work so far, whether it met their requirements and reconsider the requirements themselves.

5.2.5.3 Vision

Small and large visions feature frequently in the interviews, usually expressed in terms of "buy-in". There is an awareness from most of the interviewees of the importance of achieving "buy-in" for any change, be it a new development, a re-development or organizational change. There is an implicit acceptance that creating a vision, achieving buy-in, is an essential part of the development process.

What makes this particularly interesting is that vision is almost completely absent from the classical, process-centric, view of software development (e.g. Ince, 1990, McConnell, 1993, Pressman, 1997, Somerville, 2001). Even recent advocates of *Agile software development* (e.g. Cockburn, 2002, Beck, 2000, Eckstein, 2003) fail to explicitly recognise the role of vision although they do seem to implicitly acknowledge the role of shared vision.

While all interviewees could articulate the business objective of their development group there was wide difference in the degree to which they had bought into the vision. The degree of "buy in" seemed to be closely related to the degree to which, as individuals, they could associate the greater vision with their own vision.

Personal vision seemed to be closely related not only to financial rewards but to individuals values and goals. Indeed, these three aspects seem to be woven into the identity of the individuals: Alistair openly, if somewhat jovially, cites money and fame as part of his objective, his identity is, in part, created from the twin themes of wealth and technology prevalent in Silicon Valley.

In contrast David, who wants to work for a dedicated software company rather than a business which happens to write software, defines his personal objective in terms of learning new skills and producing quality work thus supporting his identity as a software engineer. Strong visions seem to support individual and team identity, as described in section 5.2.4.

Where a strong, shared, vision is in place it exerts a force for convergence. All team members, and the work they are going, are working in the same direction. In contrast, where the vision is absent individual visions (and defences) pull in different directions. As such, vision seems to fulfil the role claimed by methodology in the classical literature. Both give order and guidance in a complex environment.

There seem to be three interwoven themes at work in discussing vision:

- An individuals understanding and belief in the big vision
- The relationship between the individual's vision of their own identity
- How, or even whether, the vision is broken into small chunks, which place the individual within the big vision.

For example, Alistair enthusiastically describes the big vision of Supply Chain Systems, he can break this down into smaller chunks that narrate his role in changing the world. These chunks of vision represent work for him that is compatible with his sense of identity. Taken as a whole, by asserting his role as a software designer he can save the company and change the world.

Interestingly, each of the interviewees also expresses an identity of a software developer, this identity seems to provide what we may call a "default vision." We may summarise this vision: "Using computer technology in an orderly fashion to satisfy a business need." This vision is rooted in the classical literature and provides the developers with a starting point for their work.

The "default vision" is in effect a mental model. It is good in so much as it allows developers like David to drop into a non-technical environment and operate, and it can be leveraged, as at Transport Corp as a force for change. However, it may also be the source of social defences and resistance to change.

Paradoxically the "default vision" substitutes "process" for "vision." This is a mechanistic approach that allows a process to be repeated on another project, even though each project will have a unique vision. While there are virtues in a repeatable process it has come to dominate the literature on software development at the expense of vision.

Again, it is possible to see this in terms of business-technology conflict where business vision meets technology process. Alistair's high performing team at Supply Chain Systems had adopted the business vision as their vision, this resulted in a strong, tacit process beyond codification:

> "It was kind of deliberately not written down. But it was a process that really did deliberately exist. And that is very important, and that is what I'd say to people. Even though you can't point at a process you know that we have [one]."

Alistair's team have the most repeatable process studied but, in a second paradox, have the process furthest removed from classical literature.

5.2.5.4 Team learning

Although teams have already been discussed in context of learning inhibitors (section 5.2.3.2) and shared vision (section 5.2.5.3) it is worth considering the role of team learning specifically.

It is often assumed somewhat casually that learning detracts from performance. This may be true if we consider purely canonical learning such as classroom sessions and reading text books but this may not be the case were learning takes the form of problem solving and innovation.

If learning does detract from performance than we would expect those teams with least to learn to be the most productive. Certainly in the case of Warehouse Software this is not true. Here we have a group of business analysts and developers, who are skilled in the field of warehouse logistics, yet one of the most difficult problems the team faced was defining the project requirements. Conversely, the new development team at Supply Chain Systems knew very little about the supply chain but encountered no such problems even though they had to learn "on the job."

It seems that learning and problem solving need not detract from team performance, indeed, the explorative nature of the team at Supply Chain may have added to the teams overall performance.

In a changing environment this may actually help, as noted in (section 5.2.1.1) developers may be burdened by existing mental models of how things work. David at Bulk Mailing notes:

> "When you start coding a project, things change, things we thought were true one day turn out to be wrong the next, we're constantly learning how things should be"

David's team is also interesting because the development team is relatively new to the business, and contains no solution domain experts:

> "We frequently find ourselves in over our heads in terms of technology."

Overall David gives an impression of a fairly junior team that is still learning the technology and business yet is productive and works well together. This seems to support Guinan's (1998) findings that teams with similar skill levels have more effective team processes.

5.2.5.5 Leadership

While each of the interviewees were in their own way leaders each of these studies also contain an offstage leader who was absent both from the interview and much of the technical work undertaken by the teams.

Many studies of leadership take the top-down view, being written for, or about leaders, as such they consider what a leader does, what the leader should do, how a leader may motivate their team. In this study we take the bottom-up view of leadership, that is, how the leader(s) appears to the interviewees, what the leader was doing that worked, or didn't work.

In several of the studies the absent leader takes on the role of enabler. The Vice President at Supply Chain Systems hired an existing team and empowered them to come up with a solution. This done her role is secondary to the technical development. This model is repeated at Transport Corp where a new leader gave Jack the authority to bring about change and improve the environment.

A contrast to this is the abdicated leader. This is clear at Warehouse Software where the senior management decline to accept responsibility for the project. Indeed, rather than enabling and empowering the project team, in the one example of the senior management taking action they instruct the emergent leader of the project to stop. There are also signs of abdicated leadership at Hedge Fund Inc where senior management, confused by the contrasting claims of success and failure, allow the project to continue until external events force action.

In outward appearance the two paradigms of leadership share a lot in common, i.e. the leader is absent and decision making is left to the technical staff. However, in the first the leader trusts the developers, who are empowered to draw on their own experience and move the project forward. In the second, there is a lack of trust, staff are hindered and their experience discounted.

5.2.5.6 Planning as learning

Across the studies plans and planning was quite varied and largely ad hoc. A "plan" was inevitably associated with a document whether this was a written English document, a project plan or a technical design using a notation such as UML.

One common theme that does emerge about the planning process is the failure of schedule plans as constructed with Microsoft Project. This type of planning seems to consume time without adding much benefit so is abandoned, or is used as goal displacement by individuals to avoid underlying problems.

Certainly, the use of plans, and the planning process, in actual software development contrasts sharply with that described in the classical literature. Of the organizations considered only Transport Corp had documents and plans that would be recognisable to one schooled in the classical process thinking. Analysts create *use-cases* (Jacobson, 1992) to describe requirements in a story like format. This is an example of Coplien and Harrison's *Scarnarios Define Problem* pattern (2003) with the business problem to be solved emerging through the creation of use-cases which allow exploration of the problem space.

Warehouse Software created requirements documents, indeed these were necessary as part of their contractual arrangements. However the failure of these documents to remain stable is cited as one of the failings of the project.

The use of documents and plans as instruments of control, as formalised agreements between groups, arises on several occasions. On other occasions there is a clear use of documentation as a planning and learning tool. Alistair talks of writing speculative documents that are discussed and refined into plans, writing documents to better understand a problem or resolve conflict.

David discusses a document-less planning process where the developers explore the problem space:

> "When we start a new project, we'll sit down either the three of us, or just me and one of the guys, and we'll draw diagrams, write lists, throw in some technology ideas, that sort of thing."

The document-less planning practised by David and Alistair seems to work well for their teams. However, where documents are written they frequently become contentious because they become part of a formalised contract. (In fact, David and Alistair both document their plans through informal mechanisms of personal notebooks and e-mail archives respectively.)

In so much as documents are synonymous with planning, the process of creation represents an exploration of a domain and a learning activity. Once complete they form part of the contract and consequently change becomes difficult. These document plans are expected to fulfil two conflicting roles.

At Transport Corp this conflict was overcome through extensive dialogue during development process, in effect a formalised conflict resolution process. David and Alistair too surrounded documentation, formal and informal, with dialogue. These mechanisms seem to allow the plan to fulfil a role as a learning instrument rather than one of control.

5.2.5.7 Communication

Of the five studies, the two which experienced the significant problems both had communication difficulties. In particular communication between end users and developers were difficult. Hedge Fund Inc shows this particularly clearly, in Chicago where developers and traders were co-located there seemed to be few problems with the system. The London office followed the same practice for small projects but with the Equity Risk Management system the Chicago developers did not have clear communication with the London traders.

Although the geographical separation was less extreme Warehouse Software also separated their users/customers from the developers. This time the separation was between two businesses were communications, in the form of documentation, formed part of the contract. After the start of the project all communication was channelled through business analysts who had difficulties communicating with both the customers and developers.

An interesting difference of opinion emerges concerning e-mail, Jenny saw e-mail use as part of the problem at Warehouse Software:

> "There came a point where the [business analyst] was working by e-mail with the customer, which was OK when they were firming up on particular points, but they never got back into meeting the customer on a regular basis and talking to them face-to-face."

But Alistair saw e-mail as an integral part of the process:

> "What really made the team work was communication, we had a team mailing list, and we are all on instant messaging, and we would continually spit information out onto the mailing list, and it was like a continuum of thought."

A full discussion of the role of e-mail is beyond this paper and is an active research arena in its own right (e.g. Wilson, 2002, Johnson, 2002). However, it seems fair to say that in these cases it was not so much the medium, but the process of communication that was the issue.

Jenny's team was experiencing problems communicating across companies, and with frequently changing personnel. Alistair's team in contrast already knew each other well and were communicating freely. This would appear to support Coplien and Harrison's W*ork Face to Face Before Working Remotely* pattern (2003).

Particularly interesting is Alistair's tendency to initiate conflict in communication. He describes this process:

> "They [the French team] would see me send a message to the guy who sat next to me complaining about his design and code, needling about stuff, and they would be like "Why didn't he just go to the cube next door? And he's doing this in public, like he wants to point him out."
>
> I deliberately did that because I wanted everyone else to seed discussion. And I deliberately put humour into what I was doing to show that I was having a matey conversation with him."

This indicates a high degree of sophistication in the communication process; such sophistication is only possible because the team actually know one another very well - this also helps explain the teams difficulties in accepting new members as described in section 5.2.3.2.

In the more structured process environment of Transport Corp communication also plays an important role. The process used by Transport Corp offers multiple opportunities for developers, business analysts and customers to discuss issues. However, this is a very structured process when compared to the communication process at Supply Chain Systems. In part this is a result of function; the business analysts and developers at Transport deal with a multitude of frequently changing customers while the staff at Supply Chain mainly talk amongst themselves.

5.3 Success and Failure

Although not strictly part of the framework of this study no discussion of the study would be complete with reference to the ambiguous nature of *success* and *failure* in the cases examined.

Fincham (2002) has previously noted how the computing industry "seems perversely captivated by its own failures" and how a failure narrative could be used to motivate change and later success. As noted before - section 5.1 - the classical view seems to find failure in all environments, this seems to provide motivation for more rigorous processes.

At a casual level success and failure appear to be binary, exclusive labels. In these terms, a project is either a success, or it is a failure, it cannot be both. Linberg (1999) described a software development project which, for the company management was a failure, however, the software developers considered the project a success. A similar scenario plays out at Hedge Fund Inc where the new Equity Risk Management project is a success in Chicago but a failure in London. Each office uses its own narrative of success to support it's own identity - shown in Figure 13.

In both the Linberg and Hedge Fund cases a single project is simultaneously a success and a failure.

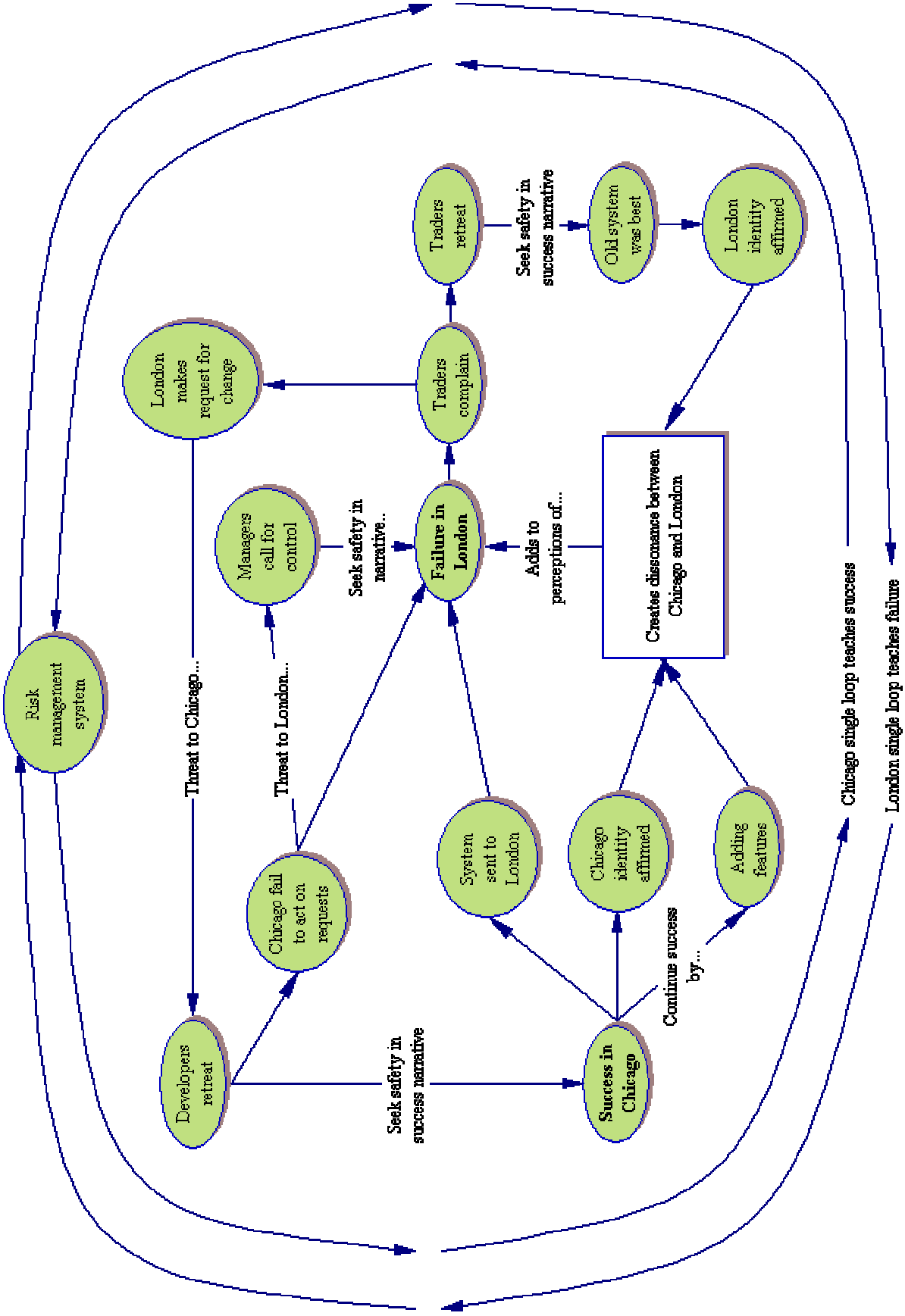


Figure 13 - Single loop learning reinforces identity at Hedge Fund Inc

At Bulk Mailing, and Supply Chain Systems we also see a disconnect between the definitions of success and failure. In the former a project which runs nine months longer than its original three month schedule is seen as a success, while at the later a development project is described as "near perfect" while the company fails.

In fact, the situation at Bulk Mailing is even more complicated. David describes verbally, and emotionally, his environment as failing because he cannot follow classical processes, yet in business terms the development group is a success.

5.4 The musical metaphor

The five case studies considered here are all very different. In fact, the single most common factor among them is how poorly the classical view of development describes them. Of the five companies, three, Warehouse Software, Transport Corp and Supply Chain Systems all operate within supply chain sector yet are widely different.

At Warehouse Software there is a complete failure of management. It is difficult to find any positive lessons in the story.

Transport Corp presents a stark contrast to this. Here we see the lead developer centre stage in co-ordinating activities, helping business analysts, software developers and customers continue with their work. Using Drucker's (1985) symphonic metaphor we can characterise Jack as the conductor. The Methodology used by the team parallels the musical score, telling analysts and developers when to play their parts. While no two projects will be the same they are recognisably performed to the same score.

This metaphor does not fit Supply Chain Systems, they reject any attempt to transcribe their performance. Each team member knows their part and where the team is heading but there is no score to describe the route. Here Weick's (1997, 1999) jazz metaphor seems to provide a better analogy. By not transcribing their score the team allows itself to constantly engage in innovation and renewal. The team are constantly monitoring the movements of each other, sometime responding, sometimes not; division of labour is minimal and the team are tightly bound by social ties.

Neither metaphor is necessarily superior, what is important is that there is a form of organization. An organization may aim for a symphonic structure but if it arrives at a jazz structure this should be valued equally. (An organization aiming for a jazz like structure seems unlikely to end up with a symphonic structure though.)

5.5 Does learning view add value?

Section 3 (Objectives and research methodology) set out two objectives:

- To illuminate the software development process as a learning centred activity.
- To suggest ways in which software development may be improved through the application of organizational learning principals.

These objectives echo the call by Willcocks (1997, p.xxv) for a "paradigm shift to be made if we are to transform the track record of systems development" and by Fitzgerald (1995) for software developers to move away from the lamppost of methodology. It is therefore worthwhile to consider whether the organizational learning view expressed here furthers any of these ideas.

From the discussion above, it is clear that the organizational learning view can provide a lot of insights into the software development arena. Indeed, the nebulous nature of organizational learning means the discipline provides a wide range of tools with which to analyse the arena. This is in stark contrast to the classical literature on software development that relies on the single tool of process.

Viewing software development through the lens of process and methodology seems to be flawed in two important ways. Firstly, almost all development exercises are condemned to failure, either for failure to provide a rigorous process framework, or failure to work to the framework. In reality developers follow process which are as influenced by their business environment as they are by the software textbooks.

Secondly, where rigorous processes are defined they are no guarantee of success. The processes themselves offer numerous opportunities for failure, for example, through goal displacement.

In contrast, the learning view offers a plentiful supply of analysis tools but little by way of pre-packaged solutions to apply. Instead this view provides a number of general practices centred on inquiry and continual learning.

For software developers who work daily with logical computers and programs, and who are literally schooled in logic, the rational, process centric view of the development process presents a strong pull. The nebulous, at times seemingly illogical, learning view is somewhat more difficult to accept.

In truth, the development process is far more complex than that documented in the classical literature. In order to fully understand software development as a rational process we must delve far deeper than current literature does. The process diagrams

in Appendix B have been drawn from interviewees' own accounts and highlight the great complexity in processes.

Rationality exists in the process but it resides at the individual level, not at the group level. None of the interviewees could be said to be irrational but their development processes contained elements which are difficult to explain in rational terms.

This is not to say that the software development process is irrational, only that to understand the process as a rational entity requires such a wealth of detail that such a study would be overwhelmed with minutiae. Consequently, the rational view is limited in its value.

As noted in the literature review (section 2.4.1) this is not the first study to look beyond rationality and methodology of the development process. Indeed, Brook's *Mythical Man Month* (1995a), Weinberg's *Psychology of Programming* (1998)and DeMacro and Lister's *Peopleware* (1987) all move outside the methodological view and remain. While the technical books come and go, and classical software engineering books like Somerville and Pressman are updated every few years these three best sellers merely issue anniversary editions every 20 years or so. In short, the methodological view dates but the amethodological is timeless.

However, what the amethodological view has lacked is a framework into which observations can be placed, explorative narratives created and insights gained. By adopting an organizational learning perspective such a framework can be constructed. What emerges from this framework are two core findings which do differ from the classical understanding.

First, software is developed in an emergent fashion. The *Waterfall* model, and "big bang deliveries" frequently used to describe software development are wrong. The emergent view has been gaining ground for some years, by placing this in a wider framework we can understand how and why the emergent view provides a better understanding.

Second, there is a deep-seated conflict between information technology and business. Successful management of this conflict offers significant opportunities for competitive advantage. However, managing this conflict is difficult, the first step towards leveraging this conflict for advantage is to acknowledge that the conflict not only exists but can be managed to bring about benefits. The view that this conflict must be resolved once and for all, and that there is a definable solution to it is not only wrong but deprives us of the opportunity to leverage the conflict for advantage.

Ultimately, these two insights show that the organizational learning paradigm does add value to our understanding of software development.

6 Conclusion

There appears to be a mismatch between the software engineering as described in the classical texts and that which is actually practised by developers. While there is evidence that software developers do adapt advocated methodologies to their needs the classical view seem to exert a greater influence on their identity than on their activities.

While the classical view is lacking in analysis tools to explain development activities, consequently analysis tends to concentrate on process and adherence to process. In contrast the organizational learning view has a surfeit tools which provide for a rich explanation of software development.

Both views offer a number of practices that may be used to improve the development cycle. The classical practices tend to be prescriptive, based on a rational approach, while the organizational learning practices tend to be social in nature, centred in individual and group interactions.

The primary source of problems in developing software appears to be conflict between business, represented by managers, and on the other hand, the technology, here represented by developers. Conflict occurs both in the mental models held by each side, and in the identities asserted by managers and software developers.

Failure to manage this conflict lies at the heart of many of the issues observed in this study. However, not only is the conflict manageable but can be leveraged as a powerful learning tool resulting in high performance. Key to managing this conflict are vision setting, team work and communication. This agenda needs enlightened managers who have moved beyond scientific management concepts.

For those in the field of organizational learning, this study shows learning in action and validates the work of many authors. Importantly, we show that the study of organizational learning and IT should not confine itself to the realm of computer assisted learning. Study of IT under development within organization should represent a fruitful field of research.

For those in the field of software engineering, this study shows that there is a need to look beyond the process and examine other dimensions of development. Organizational learning can provide a framework for this study and curriculum for education of developers.

6.1 Implications for managers

The most obvious implication for practising managers is: don't get hung up on methodology. While development process is important, alone it is not enough, it is not a magic bullet.

A corollary to this is the need for managers to look beyond the process and tools in use and recognise the role of soft skills in the development process. Highly integrated teams can use conflict to their benefit. Such teams need to be aware of the business and buy into visions which are aligned with their own goals and objectives.

This means overcoming a preconception that "conflict is bad." Conflict is highly useful if the resolution process leads to learning. However, if not managed well the conflict may linger. Alternatively a partial resolution may occur which actually hides the problem, or postpones a conflict - what Senge would call a "shifting the burden" resolution.

Although managing this conflict is difficult, it is the very difficulty in managing the conflict that offers such opportunities. Where conflict resolution easy there would be few chances to learning. Those who are able to resolve conflicts and learn have an advantage over many others.

While the IT industry sometimes gives the impression of being obsessed with "skills" and experience within certain domains and sectors this study shows that experience can in fact be a hindrance to learning. Past experience can hinder individuals, and groups, by creating mental models that do not apply in a new environment. An emphasis on past experience also forgoes the opportunity to exploit Hawthorne effect, individuals merely repeating a task they have done before may show less enthusiasm than those doing it for the first time.

An over emphasis on technical skills may also overlook the importance of soft skills. While some soft skills are individual in nature others are the result of group interactions. Since no two groups are the same it is necessary to develop, and redevelop, team skills for each group.

Finally, managers should be aware of the role of IT as a change agent and tool for learning. The days of using IT to automate activities appear to be gone, the agenda of IT is change. Failure to appreciate this agenda may lead to incorrect diagnosis and actions. The new role links IT in a two way exchange to issues of corporate strategy and identity.

6.2 Further research

A large number of avenues for further research emerge from this paper. Perhaps the most pressing given its privileged position in the literature is research into the true benefits of methodology, and indeed Methodologies. Beyond their role as an abstract teaching tool and agent for change there is the real possibility that they may prove damaging to the long-term success of a project or team.

Such research should also seek to examine the role of corporate structures on development teams and their working processes. In classical texts the engineering process is removed from the corporate environment, yet many authors (e,g, Galbraith, 1996) have written extensively on organising for innovation. Potentially software developers are missing out on a rich source of material.

On the other hand, there is also need to investigate how software developers can change the corporate environment through their role as agents of change and of organizational learning. While there is some research in this field (e.g. Ang, 1997, Edberg, 2001) there is a need for more.

The role of IT personnel as change agents should highlight the role of social skills and emotional intelligence (Goldman, 1996). Managing and implementing change well can be demanding on such skills yet IT personnel are commonly reputed to lack these skills. If true, then equipping these people with such skills should enhance the change process. Alternatively, it is possible that IT staff do not lack these skills, however, they are constantly called on to play the role of change agent thereby taxing what skills they do have. Again, improving social skills should help the change process.

Two more tangential research avenues are also opened up. Firstly, the role of success and failure; how these labels are applied within narratives, and used to support or resist change. This study supports research (Linberg, 1999, Fincham, 2002) that questions the definition and role of these labels.

Finally, this paper has referenced the study of Organizational Patterns by Coplien and Harrison and their forthcoming manuscript (2003). These authors have also noted a relationship between the pattern movement and organizational learning:

> "More broadly, the pattern philosophy of piecemeal growth is a broadening of the popular notion (particularly during the late 1980s) of *organizational learning.* ... There are strong parallels between the organizational learning field and patterns. For example, each believes in building on a small number of

> principles that generate rich emergent behaviour; complex systems of rules don't work [SwieringaWierdsma1992, 9]." (Coplien, 2003)

Many of the Coplien and Harrison's patterns are examples of organizational learning, as such pattern languages, drawing from work by architect Christopher Alexander (1977), may prove to be useful in the field of organizational learning, while the principles of organizational learning may be useful in developing pattern languages.

Further research should both explore the parallels between the two fields, and how pattern languages can be applied to further organizational learning.

6.3 Brooks reprised

This paper opened with three quotes from Fredrick Brooks, father of the IBM/360. We have shown that computer programming projects can learn a lot from modern management thinking, but, since such projects tackle issues of business strategy and identity, together with technology there is good reason to believe they are different.

Indeed, this heady cocktail of topics may well prove Brooks right in asserting that software is the most complex construct built by humans.

Finally, the problems in developing software appear to be dominated by social and managerial issues. Brooks was right to focus the *Mythical Man Month* on people and organization rather than technical approaches.

Appendix A Research questionnaire

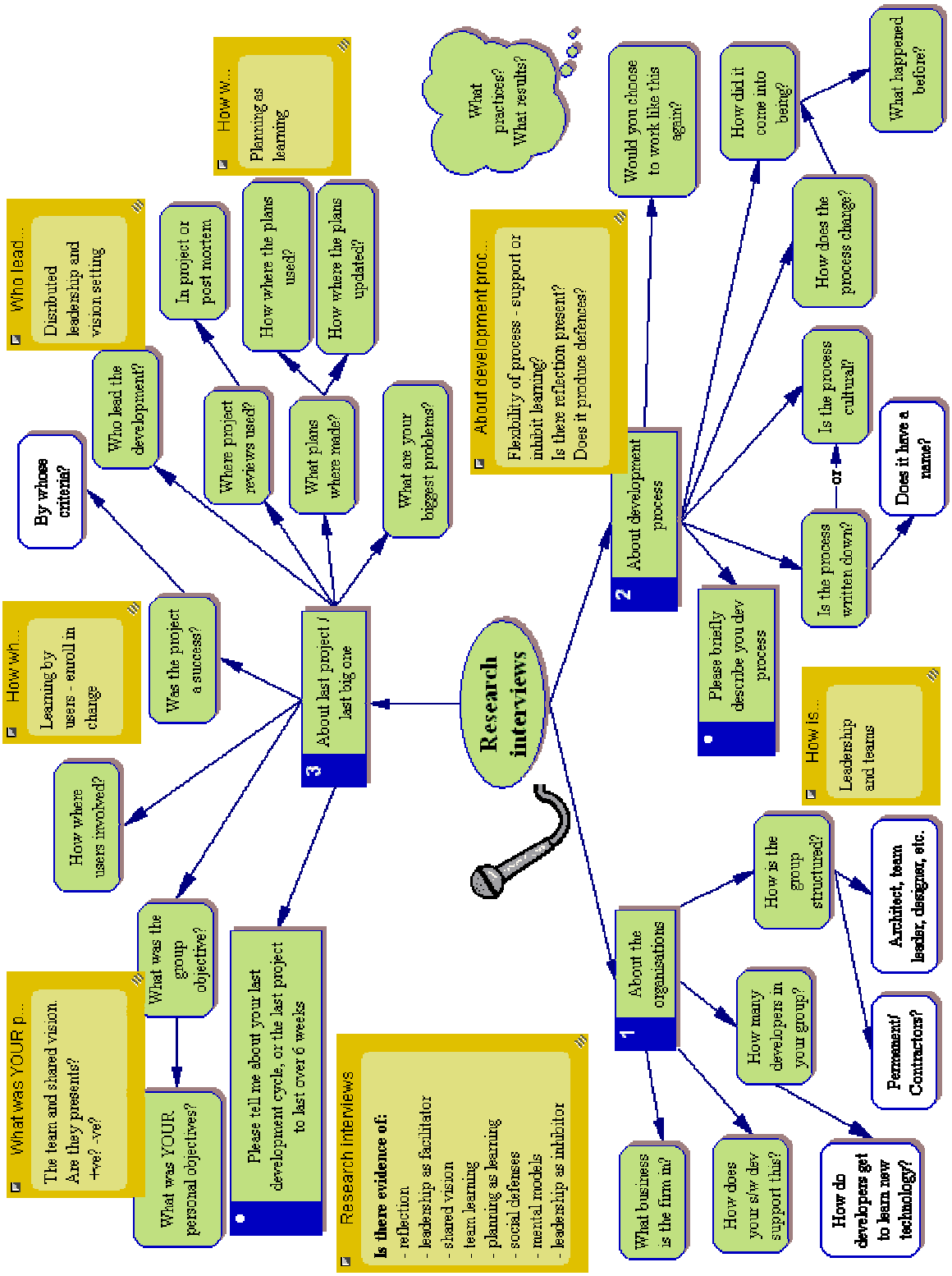


Figure 14 - Research questionnaire

Appendix B Process diagrams

B.1 Warehouse Software

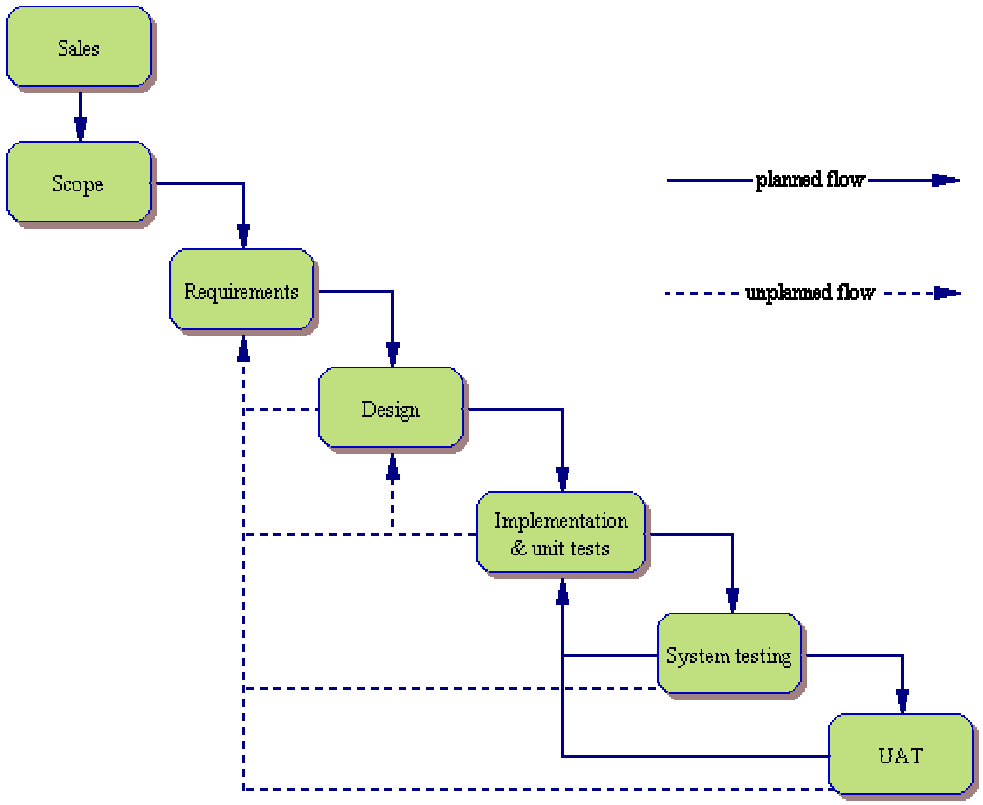


Figure 15 - Overview of development process at Warehouse Software

B.2 Bulk Mailing

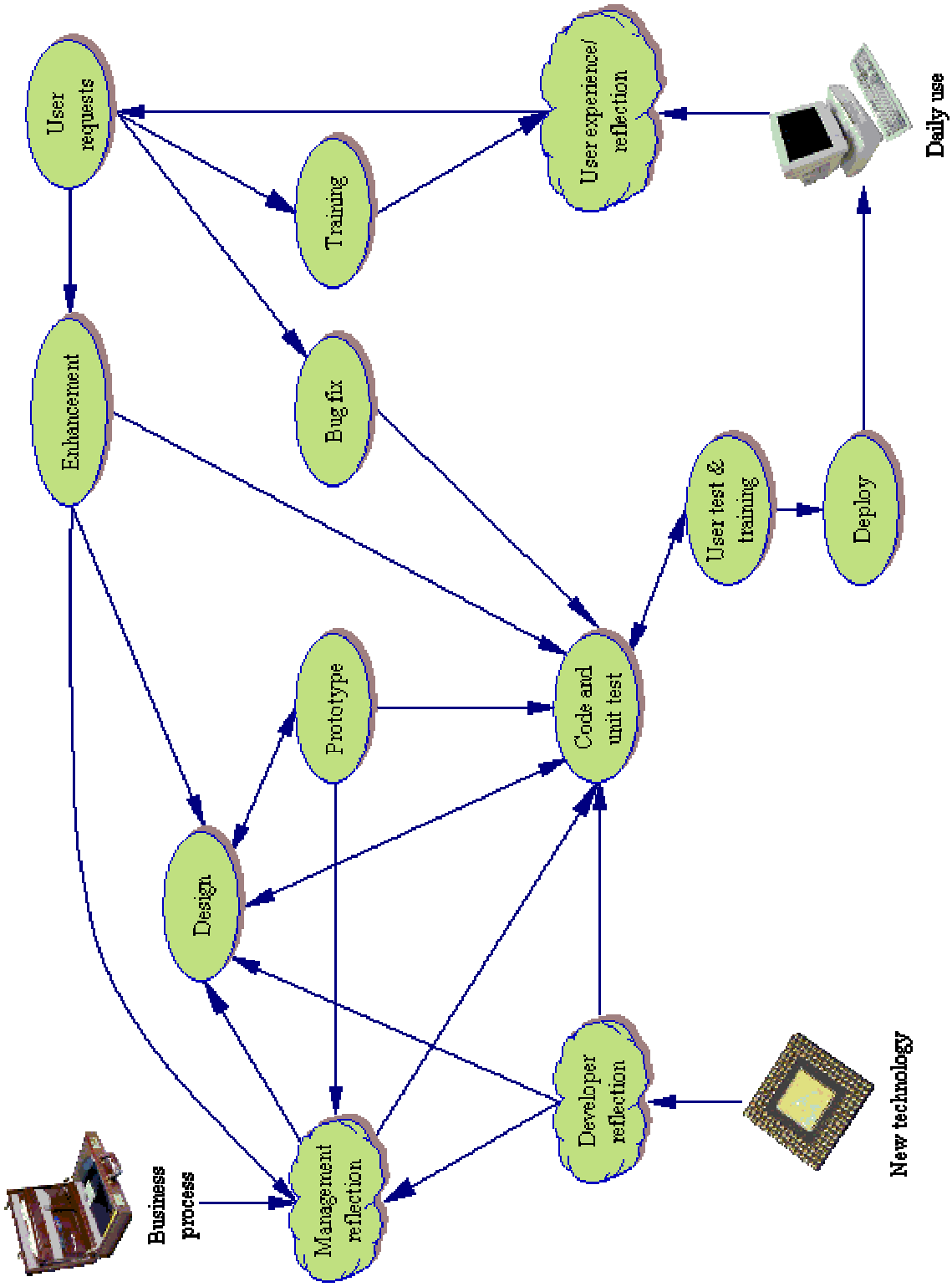


Figure 16 - Overview of the development process at Bulk Mailing

B.3 Transport Corp

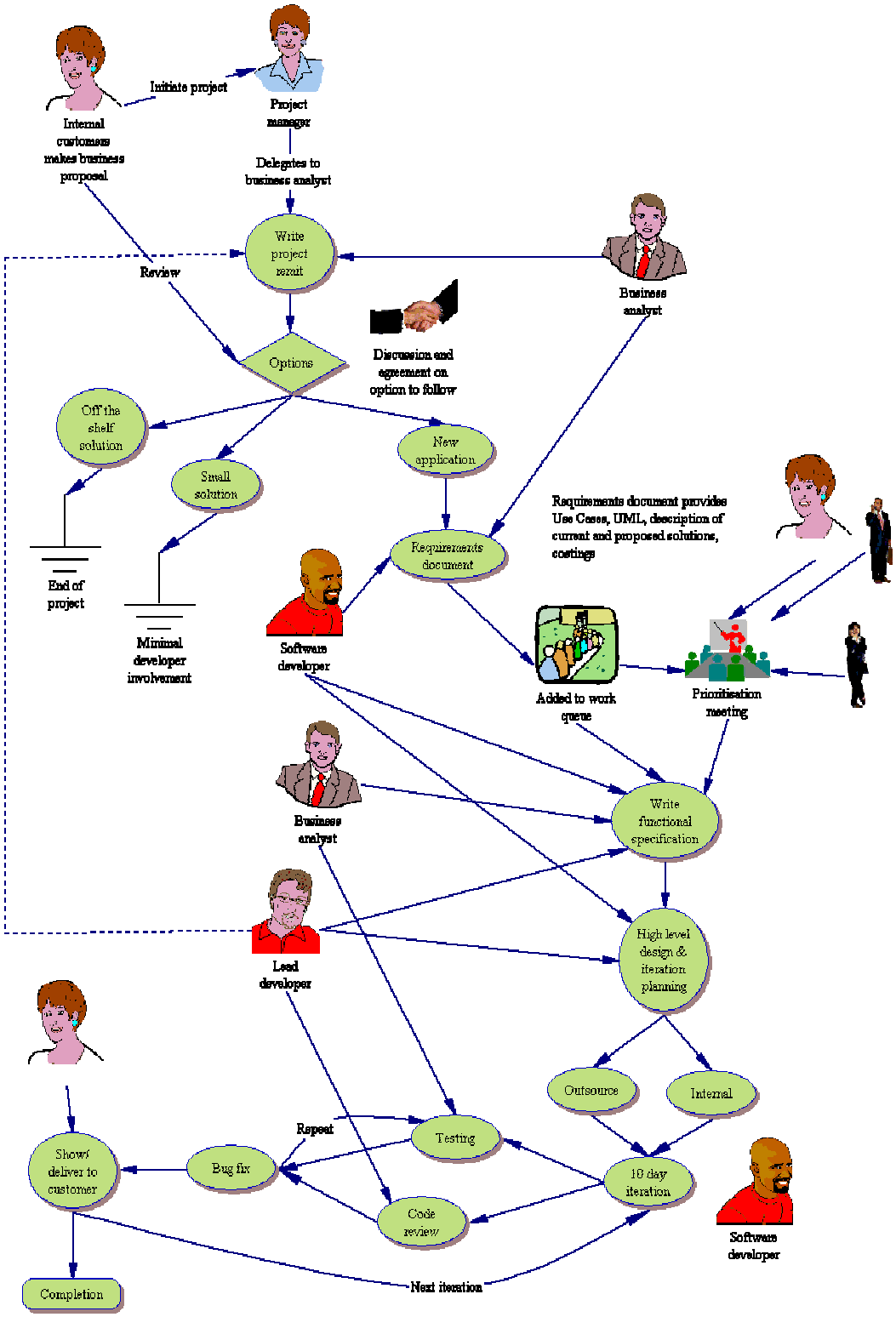


Figure 17 - Overview of the development process at Transport Corp

Appendix C Glossary

Term	Meaning	Source
Agile methodologies	A new set of software development methodologies which shun much of the "high ceremony" practices present in classical methodologies. Authors such as Beck, Cockburn and Eckstein take inspiration from the Agile manufacturing literature.	Cockburn, 2002
AIX	A version of UNIX sold by IBM for their own range of server and workstation machines.	
Application domain	"The body of knowledge that is of interest to the users", e.g. business issues	Coplien, 1998, p.7
BA	Business analyst	
Back-end	Part of software without a user interface and dealing with actual processing.	
COTS	Common off the shelf software	
Dirty Hack	A piece of code which a developer does not consider to be of professional quality, but is written because time, technology or experience does not allow for a professional solution.	
Front-end	Part of sofware handling user interface and little else.	
High ceremony process	A development methodology (e.g. SSADM) which demands rigid adherence to a set of prescribed practices and techniques.	
IS	Information Systems	
IT	Information Technology	
Low ceremony process	A development methodology (e.g. Extreme Programming) which has relatively few prescribed practices and techniques, instead being based on values and principles.	
Methodology (big-M)	A branded process model which is sold commercially, or advocated specifically, e.g. RUP, SSADM.	DeMarco, 1987, p.114
methodology	A way of working either codified or uncodified.	DeMarco, 1987, p.114

(small-m)		p.114
Plug compatible programmer	Derogatory term used by programmers, to describe management who regard all programmers as replaceable. (From hardware sales terminology.)	
Process Domain	The body of knowledge that described the development process.	
RUP	Rational Unified Process	http://www.ibm.com/rational
SMOP	Small Matter of Programming	Coplien, 1998, p.178
Solution domain	"is of central interest to the implementors but of only superficial interest to the system users" e.g. the tools and techniques used to build the system	Coplien, 1998, p.7
SSADM	Structured System Analysis and Design Methodology	
UML	Unified Modelling Language: a style of drawing diagrams of software	http://www.ibm.com/rational
Unit test	Developer testing of program code	
Use case	A form of story which describes how software, or a software feature will be *used.* Can be used as an analysis tool, or as a specification.	Jacobson, 1992 Booch, 1994
User acceptance testing	Testing of program by user aimed at accepting delivery	
User testing	Testing of program by user	
Warm bodies	Derogatory term for programmers who are assigned to projects on an "as needed" basis.	Weinberg, 1998, p.68
XML	Extensible markup language - a language for describing data.	

Appendix D Supplementary sources

Agile Manifesto (Beck, 2001)

The vision statement underlying the *Agile Methodologies*, e.g. Beck's Extreme Programming and Cockburn's Crystal Methodology. The manifesto states four guiding principles:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Designing Engineers (Bucciarelli, 1994)

Through three case studies of engineering design project, Bucciarello shows how engineering innovation and creativity comes from a multitude of sources but is embedded in a social process.

Enabling Software Development Team Performance (Guinan, 1998)

A study of software development teams found that management and team skills to be more important that tools or processes. Somewhat surprisingly Guinan suggests that teams comprised of individuals of a similar skill level are more effective at enabling team processes.

Facts and Fallacies of Software Engineering (Glass, 2003)

Accomplished software engineer and academic Robert Glass presents his 65 facts and fallacies concerning the development process. People, management and "no silver bullet" are reoccurring themes. Glass also argues that there is a disjoint between the problems and issues faced by practising programmers and those which are the focus of academic research.

How buildings learn (Brand, 1994)

Using the metaphor of learning, Brand describes how buildings change after they are built in response to human demands and innovation elsewhere. For Brand, successful buildings are those which evolve and allow change, where architecture has created buildings that are highly designed, and prioritise art over people then evolution is restricted.

How do committees invent? (Conway, 1968)

> "The basic thesis of this article is that organizations which design systems (in the broad sense used here) are constrained to produce designs which are copies of the communication structures of these organizations." (Conway, 1968, p.31)

This is restated by Coplien and Harrison in pattern form:

> "The structure of an organization, and its architecture, are isomorphic. ... therefore: Make sure the organization is compatible with the product architecture." (Coplien, 2003, p.173)

Is Software work routinized? (Ilavarasan, 2003)

A study of software development in Indian companies refutes the idea that software work is routinised and compartmentalised. The authors find that software work is carried on a project basis where teamwork and knowledge sharing are key elements of success.

Learning Company, The (Pedler, 1997)

Similar to Senge, this easy read extols the virtues of organizational learning and describes how organizations can understand their own position relative to an ideal learning organization and steps they can take towards enhancing their ability to learn. The authors lack the exuberance of Senge but provide more by way of concrete steps and organization can enact. Fortunately, these steps do not need to be followed in sequence, probably all organizations could pick up several ideas for improving their ability to learn from this book.

Research Methods for Business Students (Saunders, 2000)

General guidance and insights into the research process and in particular qualitative research.

Bibliography

Alexander, C., et al. (1977) *A pattern language,* Oxford University Press.

Ang, K., Thong, J.Y. L. and Yap, C., 1997, *IT implementation through the lens of organizational learning: a case study of insuror*, International Conference on Information Systems, http://portal.acm.org/toc.cfm?id=353071&coll=portal&dl=ACM&type=proceeding.

Argyris, C. (1977) Organizational learning and management information systems, *Accounting, Organizations and Society,* 2, 113-123.

Argyris, C. (1994) *On organizational learning,* Blackwell Publishers, Oxford.

Argyris, C., and Schön, D.A. (1996) *Organisational Learning II,* Addison-Wesley.

Beck, K. (2000) *Extreme programming explained,* Addison-Wesley.

Beck, K., et al 2001 *The Agile Manifesto*, http://agilemanifesto.org/,

Boehm, B. (2001) *Introduction to A rational design process* In *Software Fundamentals: collected papers of David L. Parnas*(Ed, Hoffman, D. M. a. W., D.M.) Addison-Wesley.

Boehm, B., and Pappacio, P.N. (1988) Understanding and controlling software costs, *IEEE Transactions on software engineering,* 14, 1462-77.

Booch, G. (1994) *Object-Oriented analysis and design with application,* Benjamin/Cummings Publishing Company, Redwood City.

Brand, S. (1994) *How Buildings Learn: What happens after they're built,* Penguin.

Brooks, F. (1975) *The mythical man month: essays on software engineering,* Addison-Wesley.

Brooks, F. (1986) No Silver Bullet - Essence and Accident in Software Engineering, *Information Processing*.

Brooks, F. (1995a) *The mythical man month: essays on software engineering,* Addison-Wesley.

Brooks, F. (1995b) *No Silver Bullet - Essence and Accident in Software Engineering* In *The mythical man month: essays on software engineering* Addison-Wesley, pp. 179-203.

Brown, A., and Starkey, K. (2000) Organizational identity and learning: A psychodynamic perspective, *The Academy of Management Review,* 25, 102-120.

Brown, J. S., and Duguid, P. (1991) Organizational Learning and Communities-of-Practice, *Organization Science,* 2**,**http://www2.parc.com/ops/members/brown/papers/orglearning.html.

Bucciarelli, L. L. (1994) *Designing Engineers,* MIT Press.

Cockburn, A. (2002) *Agile Software Development,* Addison-Wesley.

Conklin, P. F. (1996) Enrollment Management: Managing the Alpha AXP Program, *IEEE Software,* 13, 53-64.

Constantine, L. L. (1995) *Constantine on Peopleware,* Prentice Hall.

Conway, M. E. (1968) How do committees invent?, *Datamation*.

Coplien, J. (1999) *Multi-paradigm design in C++,* Addison-Wesley, Reading, MA.

Coplien, J., and Harrison, N. 2003 *Organizational Process Patterns (forthcoming)*, http://www.easycomp.org/cgi-bin/OrgPatterns, Wiki web site for book

Curtis, B., Hefley, W.E., and Miller, S.A. 2001 *People Capability Maturity Model*, http://www.sei.cmu.edu/cmm-p/,

Cusumano, M. A., and Selby, R.W. (1995) *Microsoft Secrets,* Harper Collins, London.

de Geus, A. P. (1996) *Planning as learning* In *How organizations learn*(Ed, Starkey, K.) Thompson Business Press, pp. 92-99.

Delbridge, R., and Barton, H. (2002) Organizing for continuous improvement, *International Journal of Operations and Production Management,* 22, 680-692.

DeMarco, T., and Lister, T. (1987) *Peopleware,* Dorset House, New York.

Downs, E., Clare, P., and Coe, I. (1988) *SSADM, application and context,* Prentice Hall.

Wall Street Journal 1985: Playing in the Information-Based 'Orchestra', Drucker, P. F., 4 June 1985

Duncan, J., Rackley L., and Walker, A. (1995) *SSADM in Practice,* MacMillan.

Eckstein, J. (2003) *Scaling Agile Processes (forthcoming),* Dorset House, New York.

Edberg, D., and Olfman, L., 2001, *Organizational Learning Through the Process of Enhancing Information Systems*, 34th Hawaii International Conference on System Sciences, IEEE, http://dlib2.computer.org/conferen/hicss/0981/pdf/09814025.pdf?

Eva, M. (1991) *SSADM version 4: a users guide,* McGraw-Hill International.

Fincham, R. (2002) Narratives of Success and Failure in Systems Development, *British Journal of Management,* 13, 1-14.

Fitzgerald, B., 1994, *The Systems Development Dilemma: Whether to Adopt Formalised Systems Development Methodologies or Not?*, Second European Conference on Information Systems, Nijenrode University Press, http://www.csis.ul.ie/staff/bf/.

Fitzgerald, B., 1995, *Beyond Systems Development Methodologies: Time To Leave The Lamppost?*, Information Technology and Changes in Organizational Work, http://www.csis.ul.ie/staff/bf/.

Fitzgerald, B., 1997, *Systems Development Methods for the Next Century*, Proceedings of the Sixth International Conference on IS Development methods, Plenum Press,

Fowler, M. (2000) *Refactoring,* Addison-Wesley.

Galbraith, J. R. (1996) *Designing the innovating organization* In *How Organizations Learn*(Ed, Starkey, K.) Thompson Business Press.

Gill, T. G. (1995) High-tech hidebound, *Accounting, Management and Information Technologies,* 5, 41-60.

Glass, R. L. (1998) How not to prepare for a consulting assignment, and other ugly consultancy truths, *Communications of the ACM,* 41, 11-14.

Glass, R. L. (2003) *Facts and Fallacies of Software Engineering,* Addison-Wesley.

Goldman, D. (1996) *Emotional Intelligence,* Bloomsbury.

Guinan, P. J., Cooprider, J.G., and Faraj,S. (1998) Enabling Software Development Team Performance During Requirements Definition: A Behavioral Versus Technical Approach, *Information Systems Research,* 9, 101-125.

Hamel, G., and Parahalad, C.K. (1996) *Competing for the Future,* Harvard Business School Press.

Hammer, M., and Champy, J. (1994) *Reengineering the corporation : a manifesto for business revolution,* Harper Collins.

Holt, R. 2001 *Software Architecture as a Shared Mental Model*, http://plg.uwaterloo.ca/~holt/papers/sw-arch-mental-model-010823.html, Position paper to ASERC Workshop on Software Architecture

Howcroft, D., and Wilson, M. (2003) Paradoxes of participatory practices: the Janus role of the system developer, *Information and Organization,* 13, 1-24.

Huysman, M. (2000) Rethinking organizational learning: analyzing learning processes of information system designers, *Accounting, Management and Information Technologies,* 10, 81-99.

Ilavarasan, P. V., and Sharma, A.K. (2003) Is software work routinized? Some empirical observations from Indian software industry, *The Journal of Systems and Software,* 66, 1-6.

Ince, D., and Andrews, D. (1990) *The Software Life Cycle*.

Jackson, M. (1983) *System Development,* Prentice Hall, London.

Jacobson, I. (1992) *Object-Oriented Software Engineering: a use case driven approach,* Addison-Wedley.

Johnson, L. K. (2002) Does e-mail escalate conflict?, *MIT Sloan Management Review,* 44, 14.

Kidder, T. (1981) *The Soul of the New Machine,* Avon Books, New York.

Kim, W. C., and Mauborgne, R. (2003) Fair Process: Managing in the Knowledge Economy, *Harvard Business Review,* 81, 127-137.

Kolb, D. A. (1996) *Management and the learning process* In *How organizations learn*(Ed, Starkey, K.) Thompson Business Press.

Linberg, K. R. (1999) Software Developer perceptions about software project failure: a case study, *The Journal of Systems and Software,* 49, 177-192.

Louridas, P., and Loucopoulos, P. (2000) A Generic Model for Reflective Design, *Transactions on Software Engineering and Methodology,* 9, 199-237.

McConnell, S. (1993) *Code Complete,* Microsoft Press, Redmond, WA.

McDermott, R. (1999) Why information technology inspired but cannot deliver knowledge management, *California Management Review,* 41.

Meyer, B. (1988) *Object-oriented software construction,* Prentice Hall, Hemel Hempstead.

Middleton, P. (2000) Barriers to the efficient and effective use of information technology, *The International Journal of Public Sector Management,* 13, 85-99.

Mullins, L. J. (2002) *Management and organisational behaviour,* Prentice Hall.

Nonaka, I., and Takeuchi, H. (1995) *The Knowledge Creating Company,* Oxford University Press, Oxford.

Orr, J., 1990 *Talking about machine: an ethnography of a modern job*. PhD. thesis, Cornell University

Parnas, D. L., and Clements P.C. (2001) *A rational design process: How and why to fake it* In *Software Fundamentals: collected papers of David L. Parnas*(Ed, Hoffman, D. M. a. W., D.M.) Addison-Wesley.

Pedler, M., Burgoyne, J., and Boydell, T. (1997) *The learning company,* McGraw-Hill.

Peters, T. J., and Waterman, R.H. (1991) *In search of excellence,* Harper Collins Publishers.

Pfeffer, J., and Sutton, R. (2000) *The Knowing-Doing Gap,* Harvard Business School Press.

Pitelis, C. N., and Whal, M.W. (1998) Edith Penrose: Pioneer of stakeholder theory, *Long Rang Planning,* 31, 252-261.

Pressman, R. S. (1997) *Software Engineering: a practioner's approach (European adaptation),* McGraw-Hill.

Raymond, E. S., and Steele, G.L. 2003 *The Jargon File*, http://catb.org/~esr/jargon/, 1996 version (4.0.0) published as "The New Hackers Dictionary", 3rd edition, MIT Press, ISBN 0-262-68092-0, editted by Raymond

Robey, D., and Markus, M.L. (1984) Ritual in Information System Design, *MIS Quarterly*, 5-15.

Robey, D., Boundreau, M., and Rose, G. (2000) Information technology and organizational learning: a review and assessment of research, *Accounting, Management and Information Technologies,* 10, 125-155.

Rothman, J., and Friedman, V.J. (2001) *Identity, Conflict and Organisational Learning* In *Handbook of Organizational Learning*(Ed, Dierkes, M., Merthoin Antal, B., Child, J., and Nonaka, I.) Oxford University Press, Oxford, pp. 582-597.

Royce, W. W. (1970) *Managing the development of large software systems: concepts and techniques*.

Saunders, M., Lewis, P., and Thornhill, A. (2000) *Research Methods for Business Students,* Prentice-Hall.

Schwartz, P. (1991) *The art of the long view,* Bantam Doubleday Dell, New York.

Senge, P. (1990) *The Fifth Discipline,* Random House Books.

Singh, S. (1999) *The Code Book*.

Smith, M. (1998) *Station X,* TV Books (Boxtree), New York.

Somerville, I. (2001) *Software Engineering,* Pearson Education, Harlow.

Spear, S., and Bowen, H.K. (1999) Decoding the DNA of the Toyota Production System, *Harvard Business Review,* 77, 96-107.

Starkey, K. (1996) *How organizations learn,* Thompson Business Press.

Stein, E. W., and Vandenbosch, B. (1996) Organizational learning during advanced system development: Opportunities and obstacles, *Journal of Management Information Systems,* 13, 115-137.

Truex, D., Baskerville, R., and Travis, J. (2000) Amethodical systems development: the deferred meaning of systems development methods, *Accounting, Management and Information Technologies,* 10, 53-79.

Vaill, P. B. (1996) *The purposing of high-performing systems* In *How organisations learn*(Ed, Starkey, K.) Thompson Business Press, pp. 60-81.

Wastell, D. G. (1996) The fetish of technique: a methodology as a social defence, *Information Systems Journal,* 6, 25-40.

Weick, K. E. (1997) Improvisation as a mindset for organizational analysis, *Organization Science,* 9.

Weick, K. E. (1999) The aethetic of imperfection in orchestras and organizations, *Comportamento Organizacional E Gesto,* 5, 5-22.

Weinberg, G. M. (1998) *The psychology of computer programming,* Dorset House Publishing.

Willcocks, L., Feeny, D., and Islei, G. (1997) *Managing IT as a strategic resource,* McGraw-Hill.

Wilson, E. V. (2002) Email winners and losers, *Communications of the ACM,* 44, 121.

Yourdon, E. (1989) *Modern Structured Analysis,* Prentice Hall.