1 Managing Software Engineers and their Knowledge

John S. Edwards

Aston Business School, Birmingham, B4 7ET, U.K.
Telephone: +44 121 359 3611 ext 5029 Fax: +44 121 359 5271 E-mail: j.s.edwards@aston.ac.uk

Abstract: This chapter begins by reviewing the history of software engineering as a profession, especially the so-called "software crisis" and responses to it, to help focus on what it is that software engineers do. This leads into a discussion of the areas in software engineering that are problematic, as a basis for considering the knowledge management issues. Some of the previous work on knowledge management in software engineering is then examined, much of it not actually going under a "knowledge management" title, but rather "learning" or "expertise". The chapter goes on to consider the potential for knowledge management in software engineering, and the different types of knowledge management solutions and strategies that might be adopted, and touches on the crucial importance of cultural issues. It concludes with a list of challenges that knowledge management in software engineering needs to address.

Keywords: knowledge management; software engineering; software process improvement; learning; expertise; knowledge management strategy.

1.1 Introduction

Software engineering is one of the most knowledge-intensive professions. Knowledge and its management are relevant to several aspects of software engineering at different levels from the strategic or organizational to the technical. These include:

- Estimation of costs and time-scales
- Project management
- Communicating with clients and users
- "Problem solving" in system development
- Reuse of code
- Training and staff development
- Maintenance and support

It might therefore be expected that software engineers would be well advanced in the practice of knowledge management. However, there are few signs of this being the case. Although the general knowledge management literature contains many examples of knowledge management systems in successful use in Information Technology-related companies, relatively few are specifically for software engineering. Most reported systems in these companies address areas such as overall company performance, sales and marketing, or perhaps trouble-shooting hardware failures. Mouritsen *et al*, for example, give a very detailed account (Mouritsen *et al.*, 2001) of knowledge management in the form of producing an intellectual capital statement for a software engineering firm, Systematic Software Engineering. However, there is virtually nothing in their article that is specific to software engineering.

One reason for the lack of "visibility" of software engineering in the wider knowledge management literature is the tendency for discussion of such topics to take place at conferences for the software engineering community. These include the Learning Software Organizations Workshop, the International Conference on Software Engineering, the International Conference on Software Engineering and Knowledge Engineering, and the European Software Process Improvement Conference. Thus there is an active knowledge management community in software engineering, but it is interesting that much of their work is distanced from the knowledge management mainstream.

In this chapter, we begin by reviewing the history of software engineering as a profession, to provide a background for discussing the issues involved in knowledge management in software engineering. We then look at the aspects of software engineering that may make knowledge management problematic, but equally are often the reasons why it is important. We then go on to consider what has been done so far by way of knowledge management in software engineering, and in particular the question of whether knowledge management has been taking place, but under other names. Finally, we

look at the potential for knowledge management in software engineering by offering a framework for discussing knowledge management, including the cultural issues that most influence this profession. We conclude by identifying the principal challenges for knowledge management in software engineering, and arguing for a "complementary" strategy to address them.

1.2 History of the Profession

In this section, we review some of the key features of the history of software engineering, both as an activity and a profession. This serves to introduce the relevance of knowledge management to software engineering. The topics include: the impression given of perpetual crisis; efforts at software process improvement; what software engineers actually do in technical/functional terms; and whether or not software engineering is knowledge work.

1.2.1 Perpetual crisis?

At one level, the history of software engineering gives the impression of a profession in perpetual crisis. Even before the 1968 NATO conference on Software Engineering which brought the term into common use (Naur and Randell, 1969), back in the days of punched cards and paper tape, the development of software was regarded as being problematic. Indeed, it is asserted (Randell, 1996) that the term software engineering was chosen for this conference title deliberately in order to be provocative. The tendency for commercial and governmental systems to be delivered late, over budget and lacking functionality was already becoming apparent. There was a need for the development of computer systems to be performed with the rigor and discipline associated with branches of engineering.

More than thirty years later, and in another century, not much seems to have changed, as the paper by Bryant indicates (Bryant, 2000). Granted, the majority of software development now takes places in specialized companies rather than in the in-house departments of large organizations, but the problems relating to cost, time and quality still seem to be similar. One might therefore conclude that nothing much has changed in software engineering over this period. Yet the situation is not as simple as this. Recent major successes of software engineering, such as avoiding (for the most part) any major Y2K problems and (in Europe) coping with the introduction of the Euro, have earned the profession little credit either externally or internally. The profession presents itself in a strange light, presumably because this is how it sees itself – a crisis of identity, at least. Indeed, one of the UK's weekly magazines for professionals in this field has a reputation for almost always headlining a negative story. Bryant rightly questions whether software engineering as a profession is part of the solution, or part of the problem.

Towards the middle of the period we have been discussing, Andrew Friedman (with Dominic Cornford) produced an influential account of the history of software engineering (Friedman and Cornford, 1989). One of the frameworks used for this analysis was a model based on three phases, derived from "the story so far" up to the late 1980s. The phases were dominated by hardware constraints, software issues and user needs respectively. Baxter (Baxter, 2000) argues that if Friedman's time-based phasing model had been correct, then "by now software writing would be unproblematic", but that this does not seem to be the case, as we would agree. However, Friedman himself said that phase three (dominated by user needs) would not necessarily give way to a phase four, and that "one possibility…is to revert back to the domination of earlier phase concerns". Programming issues still have a great influence on what software is created, rather than just the requirements of the users. Baxter points out that 'beta versions', 'patches' and 'bugs' are all commonplace in the software world, but, as she puts it "can the reader imagine having a 'beta' set of wheels on their car?"

The view within the software engineering department is no more reassuring. For example, Perlow (Perlow, 1999) refers to the "fast paced, high-pressure, crisis-filled environment in which software engineers work" If a general expectation that software will not work properly, and a crisis-filled environment, are reasonable indications, then software engineering is indeed a profession in a continuing state of crisis.

1.2.2 Software Process Improvement

The comments in the previous sub-section should not be taken as evidence that nothing has been done to improve matters. On the contrary, many systematic attempts have been made to produce software that is more reliable and of higher quality. One way to do this is simply to improve the testing procedures, but we will not consider this further here, for two reasons. Firstly, this approach goes against all the principles of total quality management, since it is far cheaper and easier to avoid errors rather than to find and correct them. Secondly, the ever-increasing complexity of modern software (Glass, 1996) makes it much harder to test than, say, a piece of mechanical equipment. The emphasis has therefore rightly been on producing software that is more reliable and of higher quality by methods that are more predictable and robust. These approaches are generally grouped under the heading of software process improvement. A good review of various different improvement 'technologies' is given by the experienced commentator on the field, Robert Glass (Glass, 1999).

In this section we concentrate on those improvement methods termed 'process models' by Glass, as these have the greatest relevance to the management aspects of the software engineering profession, as opposed to the technical aspects. If improvements are left solely to the technical level, then the best that is likely to be achieved will be isolated "islands of knowledge". This is a widely recognized problem in knowledge management. Among these process models are the Capability Maturity Model (CMM), the Quality Improvement Paradigm (QIP), Software Process Improvement and Capability dEtermination (SPICE), and the ISO9000 series of internationally agreed standards.

The Software Capability Maturity Model

One of the most widely recognized frameworks for looking at the extent of professionalism in a software engineering company or unit is the software Capability Maturity Model (Paulk *et al.*, 1993;Humphrey, 1989;Paulk *et al.*, 1995). This was developed at Carnegie-Mellon University's Software Engineering Institute (SEI). The CMM for software (there are now other related CMMs) is organized into five maturity levels:

1. *Initial*. The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.

2. *Repeatable*. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

3. *Defined*. The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

4. *Managed*. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

5. *Optimizing*. Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Here we see the progression from a "let's run the program and see what happens" approach to the technically rigorous and managerially disciplined approach that an engineering discipline should have. Perlow frequently refers to "individual heroics" in discussing the organization that he studied (Perlow, 1999); clearly it belongs at level 1. Knowledge management is by definition non-existent in a level 1 unit, but becomes increasingly important as the level rises. Indeed, it could be argued that more effective knowledge management is one of the hallmarks distinguishing the higher levels of capability maturity.

Quality Improvement Paradigm

The Quality Improvement Paradigm (QIP) is an approach that draws on the field of Total Quality Management (TQM). One of the pioneers of this approach was the Software Engineering Laboratory (SEL) at NASA's Goddard Space Flight Center (Basili *et al.*, 1992). The phrase coined for the resulting organization is the "experience factory". The relationship between the QIP and the experience factory is well described by Basili and Caldiera (Basili and Caldiera, 1995). They also explain why

manufacturing-based total quality approaches have not worked well in software engineering. Such approaches do not deal well enough with the nature of a software product; for example, the fact that any particular piece of software is only developed once, so that statistical quality control approaches are impossible. Some of the lessons learned at the Goddard Space Flight Center are described in Chapter 12 of this book (Rosenberg, 2003).

Software Process Improvement and Capability Determination

Software Process Improvement and Capability dEtermination (SPICE) is an initiative intended to produce an international standard for Software Process Assessment (http://www.sqi.gu.edu.au/SPICE/). This covers not only software development and operation, but also procurement and support as related to packaged software. Extensive trials have been taking place for some years. Thus far it has reached the status of a technical report (ISO/IEC TR 15504:1998) published by the International Organization for Standardization (ISO), with the intention that this will evolve into a full International Standard. More general international quality standards are covered in the next sub-section.

ISO9000 Series Standards

The ISO9000 series of standards (Hoyle, 2001) relates to quality management systems of all kinds in organizations, but some parts of the software engineering industry have been particularly attracted by the idea of systems designed to deliver products that meet customer needs. In many industries, ISO9000 certification is either a source of competitive advantage or an essential qualifier in order to be considered as a supplier at all. Software consultancies, therefore, have shown great interest in becoming accredited under ISO9000. A particular point of commonality with the other methods mentioned is that the latest version, ISO9000:2000, is constructed around the idea of viewing a business in terms of its processes, and separating those into the "realization processes" that form the core of what the organization does, and support processes. Thus in a software house or consultancy, developing software is a core realization process. However, in an organization whose business is making diesel engines or selling insurance, it would be a support process.

1.2.3 What functions do software engineers carry out?

In this sub-section, we look at the functional or technical activities carried out by software engineers, to complement the "management" perspective of the previous sub-section. Historically, attempts to describe what software engineers do have usually gone hand in hand with attempts to formalize the process by which they do it. Thus the "waterfall" led to the life cycle approaches and then to structured methods, also sometimes called methodologies; see (Friedman and Cornford, 1989). Similarly, prototyping, once the ultimate in "make it up as you go along" approaches, has acquired far more structure and transferability in recent years because of initiatives such as the development of DSDM (Dynamic Systems Development Method).

As an example of a structured method, we shall use the UK government-approved Structured Systems Analysis and Design Method (SSADM). A suitable reference for SSADM is (Weaver, 1993). In its most recent version (4.3), SSADM comprises five modules: Feasibility Study, Requirements Analysis, Requirements Specification, Logical System Specification, and Physical Design.

DSDM by its very nature has a more complex structure than the hierarchical one of SSADM. At the top level, the project process has five phases: Feasibility Study, Business Study, Functional Model Iteration, Design and Build Iteration, Implementation. In addition, there are the Pre-Project and Post-Project phases, making seven in all. The authoritative source for information on DSDM is the web site at http://www.dsdm.org (last accessed November 1, 2002). From this, it is clear that in DSDM, the term project refers to the actual system development, not to its maintenance or support. SSADM, unusually for a structured method, is even more restricted, stopping before even the programming, let alone the implementation or maintenance. Cynics might say that the mindset resulting from this is why many UK government projects run into difficulty…

This is not just an issue of semantics, however. In principle, a software development project may be cancelled at any time before its completion. Often, the method being used includes specific points at which a "stop/go" decision is to be taken. However, Baxter (Baxter, 2000) points out that in fact there is in reality only one gate (as she terms such decision points), at the end of what she terms the feasibility phase. As she puts it, "projects are never cancelled once started". Our own experience supports this view. Thus there is a very specific knowledge management issue in identifying knowledge relevant to this single gate.

There are many other methods for systems development; some of the principal ones are reviewed and compared by (Harry, 2001). Drawing on these together with SSADM and DSDM, we obtain the following list of ten activities involved in systems development and maintenance: Investigation, Determine Feasibility, Systems Analysis, System Design, Programming, Testing, Training, Documentation, Implementation, and Maintenance/Support.

Figure 1 gives an idea of the relationship between these various technical and functional activities of software engineers. It is not intended to be an exact representation, because the time spent on activities varies from one project to the next, and there will be loops back. As well as these, also shown in Figure 1, there are in addition two higher-level activities: Project Management and Control, and People Management (users, clients, project team).

For the remainder of this chapter, we shall keep this list in our minds as our description of "what software engineers do".

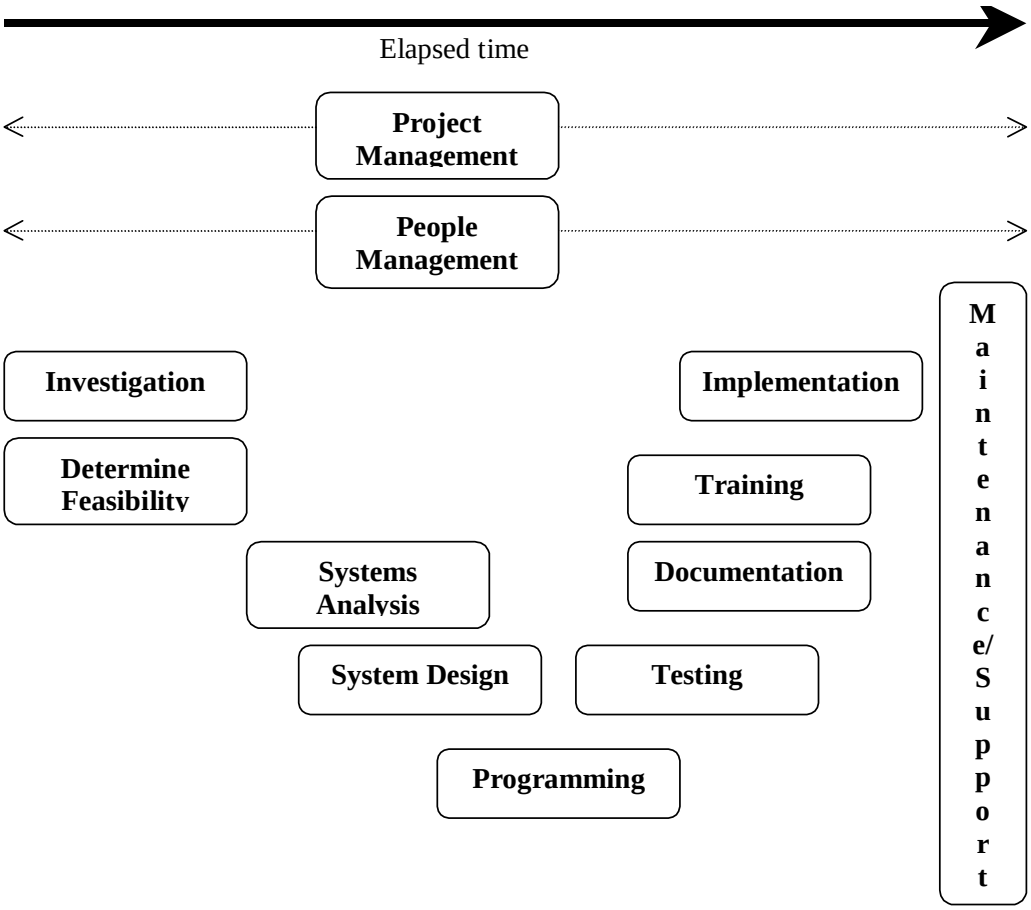

Fig. 1. The activities in software engineering

1.2.4 Is software engineering knowledge work?

Let us now consider whether software engineering qualifies as knowledge work at all. Newell *et al* suggest (Newell *et al.*, 2002) that knowledge work has three particular distinctive characteristics. The first two of these are autonomy and co-location. Autonomy of the workers is a consequence of the creativity and problem-solving aspects of the work. Creativity and problem solving have long been recognized as vital elements of software engineering. Clearly, therefore, this feature is present. Co-location is described by Newell *et al* as "the need to work remote from the employing firm, typically physically located at the client firm". This does not apply to all software engineers, but it is a definite feature of the profession, as seen in the widespread use of contractors and the outsourcing of either or both of development and maintenance work. Newell *et al*'s comment that "The client firm might therefore be in direct competition with the employing firm for the services of knowledge workers" will strike a chord with many in the IT industry.

The third feature identified by Newell *et al* is that knowledge workers are "gold collar" workers, a term coined by Kelley (Kelley, 1990). Such workers need to be "provided with excellent working conditions and generally afforded exceptional, or at least very good, terms and conditions of employment." No doubt many software engineering professionals would challenge the notion that their pay and conditions are excellent as a matter of principle, but by and large they do receive a better remuneration and benefits package than their opposite numbers in many other jobs. For example, the average salary for graduates entering IT jobs in the UK is typically some 10% higher than the average for all graduates.

We can safely conclude, therefore, that software engineering is knowledge work, and hence that knowledge management is of high importance in software engineering – or at least it should be. We now go on to look at what the problematic issues are in software engineering and its management.

1.3 Problematic areas in software engineering

Various authors have studied software engineers and software engineering over many years (Kidder, 1981;Moody, 1990;Zachary, 1994;Baxter, 2000;Perlow, 1999;Basili and Caldiera, 1995;Hohmann, 1997). Combining their views with our own experience, we see that among the problematic features particular to this profession are:

- The tension between systems development and maintenance/support work
- A combination of organizational and technical aspects
- The nature of team working
- A combination of generic skills and extremely specific skills
- Constant change, some of it externally imposed
- The need for a quick response coupled with long system lifetimes

1.3.1 The tension between systems development and maintenance/support work

Fundamentally, the work of software engineering splits into two parts – development and maintenance. These can be characterized (or perhaps caricatured) as the creative, interesting, exciting part and the boring, routine, annoying part respectively. Glass (Glass, 1996) points out that software engineering theory tended to ignore maintenance for many years, perhaps for this reason. Naturally, as with almost all such categorizations, there is a grey area in the middle where the two overlap. An important consequence of this division in the work, however, is that in many cases there is a corresponding split into separate teams. This is a distinct obstacle to successful knowledge management, because it is as important to share knowledge between the two "functions" as within them. The maintenance team needs access to knowledge about how a system was developed, but equally the development team might well benefit from knowledge of maintenance issues relating to a similar system developed previously. Sharing knowledge across teams is bound to be more difficult than within teams.

1.3.2 A combination of organizational and technical aspects

The discussion in the previous section identified that that there are both technical and organizational or managerial aspects to a software engineer's work. It is also important to realize that very few of those involved in software engineering have only technical or only organizational or managerial responsibilities. Table 1 shows a broad characterization of the relationship between these responsibilities and the activities identified earlier. This balance, or indeed tension, between technical and organizational activities is an issue to which we shall return later.

1.3.3 The nature of team working

Another relevant feature is that software engineers – and especially software *developers* – normally work in groups. However, compared to similar groups in other professions, software development groups change very rapidly. For this reason, Baxter (Baxter, 2000) prefers to call them coalitions rather than teams. Perlow (Perlow, 1999) reports that although individuals worked together, success meant doing high-visibility work, and that this was associated with the individual rather than the team. The

knowledge management implications of this are readily apparent. Sharing knowledge is necessary to get the work done, but the rapidly changing membership of the team/coalition means that the basis of the knowledge is often an individual rather than a group. As Perlow found, helping others is often seen as a distraction rather than something that will be rewarded by management.

Table 1. The different aspects of the various activities in software engineering

Activity	Main Responsibilities (in descending order)
Investigation	Organizational
Determine Feasibility	Organizational
Systems Analysis	Organizational, Technical
System Design	Technical, Organizational
Programming	Technical, Managerial
Testing	Technical, Managerial, Organizational
Training	Organizational, Managerial
Documentation	Technical, Managerial
Implementation	Organizational, Technical, Managerial
Maintenance/Support	Technical, Organizational, Managerial
Project Management and Control	Organizational, Managerial
People Management	Managerial

1.3.4 A combination of generic skills and extremely specific skills

Skills possessed by software engineers are a curious combination of the very general and the very specific. A database administrator, for example, needs to have not only generic knowledge about the principles of database design and structure, but also very detailed specific knowledge about the particular software package version, hardware configuration and operating system for which she is responsible (Barrett and Edwards, 1995). This is by no means unique to software engineers; a similar problem applies to automobile mechanics, for example. However, the balance between the general and the specific seems far less clear in software engineering than in many other professions. When does knowledge about a particular facet of database design in Oracle 8I on a Unix platform override more general knowledge of database design principles, for example?

1.3.5 Constant change, some of it externally imposed

Change increases the importance of knowledge management whilst simultaneously making it more difficult to do it effectively. A further degree of control over potential change is lost because most of the changes faced by software engineers are, to a greater or lesser extent, externally imposed. At the highest level, if a government decides to change the way in which a particular tax is calculated, then all systems relating to that tax have to be amended. However, in another sense, most of what software engineers do is externally determined, because it is client driven. Thus there is the need to anticipate change, as well as reacting to it.

1.3.6 The need for a quick response coupled with long system lifetimes

This raises an issue of what knowledge to keep, and what to discard. At one extreme, keep everything, and the response provided to a query or problem is likely to get slower and slower. At the other, keep only what is used daily, and you will soon find yourself in trouble; for example when reports or procedures that are only run annually come along. The tradition that documentation is the poor relation in software development does not help matters here.

1.4 Previous work on knowledge management in software engineering

As we said at the start of the chapter, there are relatively few “mainstream” article about knowledge management in software engineering, for example as defined by the result of a keyword search. However, the situation is beginning to change, including eight articles in a special May/June 2002 issue of *IEEE Software*. The article by the guest editors for that issue, Lindvall and Rus (Rus and Lindvall,

2002), gives a good overview of the present state of the art, as does Chapter 4 of this book, contributed by the same authors (Lindvall and Rus, 2003).

Carter (Carter, 2000) interviews Kathy Schoenherr, a software engineering manager about knowledge management in her organization, an American insurance company. Schoenherr identifies three categories of activity in software engineering where knowledge management can contribute:

- Problem tracking and resolution
- Method documentation
- Human resource issues

She also argues that effective use of knowledge management would allow more sharing of analysis and design from previous applications. (Again, the remainder of the article is about knowledge management more generally, not specifically knowledge management in software engineering.)

Hellstrom *et al* (Hellstrom *et al.*, 2001) use a software engineering firm as an example of what they call the "decentralized management of knowledge work". They argue that top-down approaches to knowledge management are inappropriate in such circumstances, and propose instead that "semi-organized" knowledge exchange, or brokerage, between individuals is what is most effective. This approach resonates with the view sometimes heard expressed that managing software engineers is like herding cats!

Kautz *et al* also look at knowledge management, specifically knowledge creation, in a small Danish software house (Kautz *et al.*, 2002). They look in particular at the role of IT systems in knowledge management, and discuss various tasks as knowledge processes, especially quality assurance for the software. They conclude that the IT systems played "an important, yet subordinate role". Openness, trust and mutual respect were vital in enabling learning to take place.

Doctoral theses (which have an inevitable three- or four-year time lag) are also beginning to appear in the area of knowledge management in software engineering, for example those of Birk (Birk, 2000), Dingsøyr (Dingsøyr, 2002) and van Aalst (van Aalst, 2001). Some of Dingsøyr's work may also be found in Chapter 3 of this book (Dingsøyr and Conradi, 2003).

1.4.1 Knowledge management by another name?

As well as the research outlined above, there is also much work that is relevant to knowledge management in software engineering which does not actually call itself knowledge management, either by choice (especially in the case of some of the conferences referred to earlier), or because the term was not current when the article was written. There are three strands of relevant work, one being that on professional expertise in software engineering, a second on learning and experience in software engineering, and the third on the use of knowledge-based systems in software engineering.

Professional expertise in software engineering

We have already drawn on this literature in our earlier discussions, including (Friedman and Cornford, 1989). The work in this strand stresses that knowledge is socially constructed. Although there must be limits to the extent to which this affects, say, a work-around for a bug in a COBOL compiler, the organizational dimension of software engineering knowledge management is clearly dependent on this. Scarbrough (Scarbrough, 1996b) explains this position well.

Williams and Procter discuss IT expertise in a bank, using an extended case study (Williams and Procter, 1998). They use a typology due to Winstanley to identify four different situations for the software engineer, according to the power that their expertise possesses in internal (within their own organization) and external labour markets. This is shown in Table 2.

Table 2. Winstanley's typology (Winstanley, 1986)

	Undeveloped internal labour market	Developed internal labour market
Positive worker power in external labour market	A. Independent mobile professional	B. Company professional
Negative worker power in external labour market	C. Insecure contract worker	D. Dependent worker

Expertise in this context appears to mean the same as what we have termed knowledge. The external labour market has a strong component of technical knowledge. The internal labour market has a strong element of organizational knowledge. Williams and Procter identified three teams of software engineers (including all roles from programmers up to management) within the bank who fell into three different categories in the typology. The first team was very technically-oriented, and their knowledge

related mainly to programming languages and technology. They thus fell into category A, independent mobile professionals. A second team, although possessing strong programming knowledge, relied even more on its internal reputation – earned by knowledge of the bank's systems. They come into category B, company professionals. The third team had a much broader range of knowledge, but not the same in-depth knowledge of any area as the other two. They came under category D, dependent workers.

Newell *et al* (Newell *et al.*, 2002) continue to draw on this school of work, although nowadays with an explicit knowledge management label. They remark that IT experts are increasingly subject to market pressures, because of developments such as the rise in outsourcing and the use of consultants, and that this tends to dilute the role of the profession in regulating abstract knowledge. In the Williams and Procter/Winstanley terms, software engineers are being pushed from category A to category C, and from category B to category D. This substantially increases the knowledge management problems for user organizations, who are becoming more and more dependent on their "providers" for software knowledge. It will also have adverse effects on the attitude of the software engineers towards sharing their knowledge, especially for those in category C.

Where the outsourcing or consultancy is provided from another country, the problems will be more acute still. Davenport and Prusak (Davenport and Prusak, 1998) explain the need for face-to-face meetings to facilitate knowledge sharing. Edwards and Kidd (Edwards and Kidd, 2003) describe some of the additional problems of cross-border knowledge management.

Learning and experience in software engineering

A central element of this strand is the "experience factory" work referred to earlier (Basili and Caldiera, 1995). More recent papers drawing on the earlier work (Schneider *et al.*, 2002;Houdek *et al.*, 1998) describe DaimlerChrysler's implementation of an Experience Center in software engineering. These ideas have now spread widely; for an Australian example see (Koennecker *et al.*, 1999), and see also (Brössler, 1999;Chatters, 1999). The thrust of this work involves robust processes with a strong emphasis on managing the people as well as the software systems. There are strong connections between this strand of work and the extensive literature on learning organizations, much of which was inspired by the work of Senge (Senge, 1990).

Knowledge-based systems in software engineering and more generally

This strand of work also has a long history, although just as most knowledge management research about software engineering firms is not specifically related to software engineering, so most knowledge-based systems in software engineering firms are not specifically related to software engineering either. One of the themes that carries over into knowledge management work has been that of understanding the nature of what software engineers do. See, for example, all eight of the articles in Part I of the collection edited by Partridge (Partridge, 1991).

The more important lessons from past research or applications in this strand are often not the knowledge-based systems that were created (or even in some cases failed to be created) but the processes of knowledge elicitation and representation that the developers, experts and users went through. For example, the issues of work in teams and the balance between general and specific knowledge were central to the work of Barrett and Edwards (Barrett and Edwards, 1995) on a system for database design and maintenance. No fewer than eight layers of expertise, from the most general to the most specific, were identified. Different experts would propose different solutions to a problem, and some means of "adjudicating" between them was necessary. A "knowledge czar" approach – nominating someone as the senior expert - was chosen.

A great deal of knowledge-based systems work in software engineering has been carried out at the Fraunhofer Institute for Experimental Software Engineering (IESE). Examples of this can be found in (Bomarius *et al.*, 1998) and in some of the papers in (Althoff *et al.*, 2001), and Chapter 11 of this book gives the current position (Althoff and Pfahl, 2003).

More generally in the knowledge-based systems field, one of the most widely-used methods for building knowledge-based systems, CommonKADS, an extension of the earlier KADS (Schreiber *et al.*, 1994;van Heijst *et al.*, 1997) is based on a philosophy of knowledge modelling. CommonKADs incorporates no fewer than six types of model: Organizational, Task, Agent, Expertise, Communication, and Design. There are libraries of common problem solving methods, and extensive ontologies. Knowledge modelling surely is one approach to knowledge management, but the knowledge management literature makes virtually no reference to KADS or CommonKADS at all.

1.5 Potential for Knowledge Management

Let us now attempt gradually to bring these diverse themes together. Picking up the earlier theme from Kautz *et al* (Kautz *et al.*, 2002), there have been many studies over the years of the psychological profiles and personality traits of computer programmers and software engineers. A relatively recent example (Wynekoop and Walz, 1998) is interesting in that it considers programmers, systems analysts and project managers separately. Many previous studies have either considered only one of these groups, or have combined all of them together. Wynekoop and Walz found that the three groups differed both from each other, and from the general population:

"The picture that emerges is that IS personnel are more conventional, conscientious, diligent, dependable, organized, logical, and analytical than the general population. However, systems analysts and managers also possess more leadership characteristics, and are more ambitious hardworking and creative with more self-confidence and a stronger self-image. Programmers, on the other hand, are more inflexible and predictable and less social than the general population."

Assuming that we can equate "IS personnel", as identified by Wynekoop and Walz, with software engineers, a further important point is that their results confirmed earlier findings that software engineers are innovative and creative (Sitton and Chmelir, 1984). Thus both innovative/creative and analytical/technical dimensions of knowledge are present in software engineering, and both may benefit from being managed.

In order to proceed further, we present in Figure 2 a model that we have used before (Edwards, 2000). This model takes an organizational viewpoint regarding what happens to a particular element of knowledge. First, it is Created/Acquired; then it goes through a cycle of Retain, Use and Refine/Update (any of these activities may be temporary, or indeed missing entirely). It may also be Shared with/Transferred to those outside the circle of people who originally Created/Acquired it, in parallel with this Retain-Use-Refine cycle.

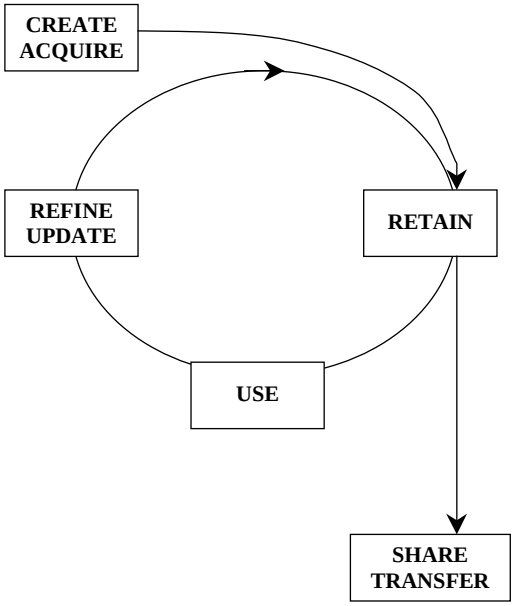

Fig. 2. A view of the knowledge management process

These five "knowledge activities" need to be considered in relation to the list of "software engineering activities" presented in the previous section. In principle, there needs to be a process to carry out each of the knowledge activities effectively for each of the software engineering activities. In general, there can be no rules as to which is more important or easiest to do. Knowledge management must be situated in an organizational context; these priorities must be determined for any given software engineering unit at any given time.

1.5.1 Types of solution

An investigation into the approaches that managers believe should be used in knowledge management (Edwards *et al.*, 2002) identified that, broadly speaking, there are three types of "solution" that can be

applied in knowledge management. These are technological, people and process solutions. Although this research looked at knowledge management in general, we believe that the categories apply to knowledge management in software engineering.

Technological solutions are concerned with installing new technology or making better use of existing technology. Specific technologies in the study included data mining, databases or Intranet access. Activities included standardization of hardware or software, eliminating duplicate systems or data, and in one case trying to discourage the use of privately owned personal organizers and laptops, which were seen as a barrier to sharing information and knowledge.

People solutions are concerned with staff retention and motivation, training, debriefing and networking. One organization identified the need to rely less on "training through osmosis". Significantly for software engineering, another thought the processes should involve removing their previous "culture of confidentiality".

Process solutions are concerned partly with paper-based specifications and process instructions but also with the mix between formal and informal methods of sharing knowledge. The emphasis is on "working smarter". In the study mentioned above, these solutions tended to be favored by the smaller organizations – the ones in which, at least in principle, everyone knew who everyone else was.

1.6 Overall Knowledge Management Strategy

The last element in our framework is that, broadly speaking, there are two overall strategies in knowledge management: codification and personalization, as pointed out by Hansen *et al* (Hansen *et al.*, 1999). These may be applied either separately or, more profitably, in a complementary fashion. Within the overall strategy, any or all of the three types of solution mentioned in the previous section may be deployed. Certain combinations tend to occur naturally. Codification strategies tend to be associated with technological solutions such as Intranets and knowledge repositories. Personalization strategies more often favour people-based solutions such as Communities of Practice (CoPs) and storytelling. A more complementarist approach may favour process-based solutions, especially those that integrate top-down and bottom-up knowledge management concerns: see Edwards and Kidd (Edwards and Kidd, 2001) for further discussion of the latter. We now look at the possibilities for each of these three strategies in software engineering knowledge management.

1.6.1 Codification strategies

Some software engineers might be expected to be more sympathetic to a codification strategy. The work of Wynekoop and Walz mentioned earlier (Wynekoop and Walz, 1998) would suggest that this ought to be especially true of programmers. Codification strategies seem appropriate when the "right answer" from one context is easily transferable to another. Thus sharing knowledge about programming issues should be suited to this strategy. There has indeed been a considerable amount of work on tools to support programming and design work (two of the most technical activities from Table 1). These include so-called CASE tools and designer workbenches. These are most useful for Retain, Share and Use activities in knowledge management; they provide little support for Refining knowledge and none for Creating knowledge.

Problem tracking and resolution, and method documentation, identified earlier as categories of knowledge management activity, also seem to be targets for codification strategies. There is, however, a snag here. Much of this work has concentrated on Retaining and Sharing knowledge within a single project. As was argued by Schoenherr in the interview cited earlier(Carter, 2000), effective Sharing of analysis and design knowledge between applications is a major potential benefit.

The more concrete products of the knowledge-based systems work on software engineering mentioned earlier also correspond to a codification approach to knowledge management.

1.6.2 Personalization strategies

Having identified codification strategies as best suited to the more technical activities within software engineering, personalization strategies by implication are more suited to the managerial and/or organizational activities. Personalization strategies can be very effective for Creating and Refining knowledge, and also effective for Sharing and Retaining it. They provide less direct help in Using it.

Human resource issues in software engineering are clearly candidates for a personalization strategy for knowledge management. Most of the discussion in the paper by Hellstrom *et al* (Hellstrom *et al.*, 2001) concerns successful personalization strategies. The professional expertise, and learning and experience strands of research into software engineering also ally themselves naturally with this viewpoint. We would argue that the managerial activities (i.e. those relating directly to the people involved with the project) are those where a personalization strategy is likely to be most successful, along with higher-level technical activities such as those in analysis and implementation where Creating and Refining knowledge is crucial, i.e. existing 'solutions' aren't good enough.

1.6.3 Complementary strategies

Our view is that, while codification and personalization both have their place, a complementary strategy is the most effective. This must involve process-based 'solutions', often to link technological- and people-based ones. How, for example, does an organization ensure that knowledge Created in a community of practice is then successfully Retained? What elements can be stored in some kind of repository, and what cannot? Post mortems, as advocated by Birk, Dingsøyr and Stålhane (Birk *et al.*, 2002) are useful under all types of strategy. In a personalization strategy, a post mortem aids both individual and group understanding, while in a codification strategy, it assists in determining what documents, databases etc. are worth keeping.

The paper by Kautz *et al* (Kautz *et al.*, 2002) is a good example of a complementary strategy towards knowledge management using IT for codification where it is appropriate , but also employing a range of other approaches. The knowledge-based systems work where the emphasis was on elicitation of the knowledge rather than building a system also fits well into this category.

1.6.4 The importance of cultural issues

Although we come to this heading last, research suggests that in many ways culture is the most important aspect of knowledge management generally (Scarbrough, 1996a;Newell *et al.*, 2002). Software engineering should be no exception, because most of the emphasis in the "process improvement" and "experience" approaches is on understanding and controlling the process and the product. This must be a shared, rather than an individual understanding, or else there is no guarantee that the process will be repeatable. Individuals may excel in Creating or Using knowledge (to use the Figure 2 terminology), but successful knowledge management in software engineering means an emphasis on Retaining and Sharing knowledge, whether the overall strategy is codification, personalization, or both. This can only be achieved with an appropriately supportive knowledge sharing culture (Snowden, 2000;Huber, 2000). Such a culture may not come naturally to all software engineers or their departments, given the findings of Wynekoop and Walz (Wynekoop and Walz, 1998) that programmers are less social than average, and the rewarding of individual heroics found by Perlow (Perlow, 1999).

Crucially, such a culture needs to be generated both top down, from management expectations and leadership, and bottom up, from the community of software engineers within the organization (Edwards and Kidd, 2001).

A final cultural issue is that knowledge management in software engineering may not involve just the software engineers. The culture of the users may be important too. Al-Karaghouli *et al* (Al-Karaghouli *et al.*, 2002) discuss a system to help what they term the system developers and their customers to understand and communicate with each other. However, such a technological solution will be of little help unless the customers also trust the developers, whether they are external consultants, or in-house colleagues.

1.7 Conclusion/Summary

The way in which software engineering is organized has changed substantially over the past 35 years, but many of the knowledge management issues have not. Software engineers face issues connected with technical, managerial and organizational activities, the balance between which depends both on the particular individual's job, and the context they are working in at any given time.

Among the principal challenges to be faced are:

- Software engineering is knowledge work. Effective knowledge management is therefore vital in improving the professionalism of a software engineering department or unit. Analysis and design knowledge particularly needs to be Shared between projects.
- The fact that projects are rarely cancelled except at the end of the feasibility study makes Retaining knowledge about how to make this stop/go decision crucial.
- The division between development and maintenance can easily become a split with dire consequences if knowledge management is not performed well, especially Sharing knowledge between individuals and teams.
- Rapid turnover of staff makes it important to retain continuity of knowledge. However, the high workloads that are in part a consequence of this high turnover mean a lack of time for knowledge Sharing and for reflective activities such as knowledge Refinement.
- Software engineering knowledge contains an unusually complex combination of different layers of expertise, from the very general to the very specific. This is especially problematic when Using knowledge.
- The culture of the department or unit, and indeed the organization it is part of, must encourage a bottom up "buy in" to knowledge management activities that matches the knowledge management strategies employed from the top down.

Despite the many problems, nevertheless effective knowledge management in software engineering is possible. There are technological, people and process-based solutions, and the best approach is surely a combination of all three, within an overall knowledge management strategy that includes both personalization and codification elements. At least any obstacles facing software engineers are not related to technical issues of computer support for knowledge management, since using computer-based tools poses few such problems for software engineers. The most important aspect overall, however, is to develop a culture that encourages both knowledge sharing and reflection.

References

Al-Karaghouli, W., Fitzgerald, G. and Alshawi, S. (2002) Knowledge Requirements Systems (KRS): An Approach to Improving and Understanding Requirements. In *Knowledge Management in the Sociotechnical World: The Graffiti Continues* (Eds, Coakes, E., Willis, D. and Clarke, S.) Springer-Verlag, London, pp. 170-184.

Althoff, K.-D., Feldmann, R. and Müller, W. (Eds.) (2001) *Advances in Learning Software Organizations.* Springer Verlag.

Althoff, K.-D. and Pfahl, D. (2003) Integrating experience-based knowledge management with sustained competence development. In *Managing Software Engineering Knowledge* (Ed, Handzic, M.) Springer-Verlag, Berlin.

Barrett, A. R. and Edwards, J. S. (1995) Knowledge elicitation and knowledge representation in a large domain with multiple experts. *Expert Systems with Applications*, **8**, (1), 169-176.

Basili, V. R. and Caldiera, G. (1995) Improve software quality by reusing knowledge and experience. *Sloan Management Review*, **37**, (1), 55-64.

Basili, V. R., Caldiera, G., McGarry, F., Pajerski, R., Page, G. and Waligora, S. (1992) The Software Engineering Laboratory - An operational software experience factory. Presented at *14th International Conference on Software Engineering, ICSE 14*, 1992.

Baxter, L. F. (2000) Bugged: The software development process. In *Managing knowledge: critical investigations of work and learning* (Eds, Prichard, C., Hull, R., Chumer, M. and Willmott, H.) Macmillan, Basingstoke, pp. 37-48.

Birk, A. (2000) *A Knowledge Management Infrastructure for Systematic Improvement in Software Engineering.* Unpublished Dr. Ing Thesis, University of Kaiserslautern.

Birk, A., Dingsøyr, T. and Stålhane, T. (2002) Postmortem: Never Leave a Project without It. *IEEE Software*, **19**, (3), 43-45.

Bomarius, F., Althoff, K.-D. and Müller, W. (1998) Knowledge Management for Learning Software Organizations. *Software Process - Improvement and Practice*, 89-93.

Brössler, P. (1999) Knowledge Management at a Software House : An Experience Report. In *Learning Software Organizations : methodology and applications; proceedings from the 11th International Conference on Software Engineering and Knowledge Engineering, SEKE '99, Kaiserslautern, Germany, June 16 -19,*

1999, vol. 1756, Lecture Notes in Computer Science (Eds, Ruhe, G. and Bomarius, F.) Springer Verlag, Berlin, pp. 163 - 170.

Bryant, A. (2000) "It's Engineering Jim; but not as we know it" - Software Engineering, solution to the software crisis or part of the problem? In Proceedings of *22nd International Conference on Software Engineering* Limerick, Ireland.

Carter, B. (2000) The expert's opinion: knowledge management. *Journal of Database Management*, **11**, (1), 42-43.

Chatters, B. (1999) Implementing an experience factory: Maintenance and evolution of the software and systems development process. Presented at *IEEE International Conference on Software Maintenance*, 1999.

Davenport, T. H. and Prusak, L. (1998) *Working knowledge : how organizations manage what they know.* Harvard Business School Press, Boston, Mass.

Dingsøyr, T. (2002) *Knowledge Management in Medium-Sized Software Consulting Companies*. Unpublished PhD Thesis, Norwegian University of Science and Technology.

Dingsøyr, T. and Conradi, R. (2003) Usage of Intranet tools for knowledge management in medium-sized software consulting companies. In *Managing Software Engineering Knowledge* (Eds, Aurum, A., Jeffery, R., Wohlin, C. and Handzic, M.) Springer-Verlag, Berlin.

Edwards, J. S. (2000) Artificial Intelligence and Knowledge Management: How Much Difference Can It Really Make? In Proceedings of *KMAC2000. Knowledge Management Beyond The Hype: Looking Towards The New Millennium* (Eds, Edwards, J. S. and Kidd, J. B.) Operational Research Society, Aston University, Birmingham, UK, pp. 136-147.

Edwards, J. S. and Kidd, J. B. (2001) Knowledge management when "the times they are a-changin'". In Proceedings of *Second European Conference on Knowledge Management* (Ed, Remenyi, D.) MCIL, Reading, UK, Bled, Slovenia, pp. 171-183.

Edwards, J. S. and Kidd, J. B. (2003) Knowledge Management sans frontières. *Journal of the Operational Research Society*, **to appear**.

Edwards, J. S., Shaw, D. and Collier, P. M. (2002) Group perceptions of knowledge management. In Proceedings of *Third European Conference on Knowledge Management* MCIL, Reading, UK, Trinity College, Dublin, Ireland, pp. 209-222.

Friedman, A. L. and Cornford, D. S. (1989) *Computer systems development, history, organization and implementation.* Wiley, Chichester.

Glass, R. L. (1996) The relationship between theory and practice in software engineering. *Communications of the ACM*, **39**, (11), 11-13.

Glass, R. L. (1999) The realities of software technology payoffs. *Communications of the ACM*, **42**, (2), 74-79.

Hansen, M. T., Nohria, N. and Tierney, T. (1999) What's your strategy for managing knowledge? *Harvard Business Review*, **77**, (2), 106-116.

Harry, M. J. S. (2001) *Business Information: A Systems Approach.* Financial Times Prentice Hall, Harlow.

Hellstrom, T., Malmquist, U. and Mikaelsson, J. (2001) Decentralizing knowledge: managing knowledge work in a software engineering firm. *Journal of High Technology Management Research*, **12**, (1), 25-38.

Hohmann, L. (1997) *Journey of the software professional : a sociology of software development.* Prentice Hall, New Jersey.

Houdek, F., Schneider, K. and Wieser, E. (1998) Establishing Experience Factories at Daimler-Benz. An Experience Report. Presented at *20th International Conference on Software Engineering, ICSE 20*, Kyoto, Japan, 1998.

Hoyle, D. (2001) *ISO 9000 Quality Systems Handbook.* Butterworth-Heinemann, London.

http://www.sqi.gu.edu.au/SPICE/ The Software Process Improvement and Capability dEtermination Website. (Last accessed November 6, 2002.)

Huber, G. P. (2000) Transferring Sticky Knowledge: Suggested solutions and needed studies. In Proceedings of *Knowledge Management beyond the Hype: Looking towards the new millennium. Proceedings of KMAC 2000* (Eds, Edwards, J. S. and Kidd, J. B.) Operational Research Society, Birmingham, pp. 12-22.

Humphrey, W. S. (1989) *Managing the Software Process.* Addison-Wesley, Reading, Mass.

Kautz, K., Thaysen, K. and Vendelø, M. T. (2002) Knowledge creation and IT systems in a small software firm. *OR Insight*, **15**, (2), 11-17.

Kelley, R. (1990) *The gold collar worker - harnessing the brainpower of the new workforce.* Addison-Wesley, Reading, Mass.

Kidder, T. L. (1981) *The soul of a new machine.* Avon, New York.

Koennecker, A., Jeffery, R. and Low, G. (1999) Implementing an Experience Factory Based on Existing Organisational Knowledge. Presented at *Australian Conference on Software Metrics, ACOSM'99*, 1999.

Lindvall, M. and Rus, I. (2003) Knowledge management in software engineering. In *Managing Software Engineering Knowledge* (Ed, Handzic, M.) Springer-Verlag, Berlin.

Moody, F. (1990) *I sing the body electric: a year with Microsoft on the multimedia frontier.* Viking, New York.
Mouritsen, J., Larsen, H. T., Bukh, P. N. and Johansen, M. R. (2001) Reading an intellectual capital statement: describing and prescribing knowledge management strategies. *Journal of Intellectual Capital*, **2**, (4), 359-383.
Naur, P. and Randell, B. (Eds.) (1969) *Software Engineering: Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968.* Scientific Affairs Division, NATO, Brussels.
Newell, S., Robertson, M., Scarbrough, H. and Swan, J. (2002) *Managing knowledge work.* Palgrave, Basingstoke.
Partridge, D. (Ed.) (1991) *Artificial intelligence and software engineering.* Ablex, Norwood, NJ.
Paulk, M. C., Curtis, B., Chrissis, M. B. and Weber, C. V. (1993) Capability Maturity Model, Version 1.1. *IEEE Software*, **10**, (4), 18-27.
Paulk, M. C., Weber, C. V. and Curtis, B. (1995) *The Capability Maturity Model: Guidelines for Improving the Software Process.* Addison-Wesley, Reading, Mass.
Perlow, L. A. (1999) The time famine: Toward a sociology of work time. *Administrative Science Quarterly*, **44**, (1), 57-81.
Randell, B. (1996) The 1968/69 NATO Software Engineering Reports. Presented at *Dagstuhl-Seminar 9635: "History of Software Engineering"*, Schloss Dagstuhl, Germany, August 26 - 30, 1996.
Rosenberg, L. H. (2003) Lessons learned in sofware quality assurance. In *Managing Software Engineering Knowledge* (Eds, Aurum, A., Jeffery, R., Wohlin, C. and Handzic, M.) Springer-Verlag, Berlin.
Rus, I. and Lindvall, M. (2002) Knowledge management in software engineering. *IEEE Software*, **19**, (3), 26-38.
Scarbrough, H. (1996a) *The management of expertise.* Macmillan Business, Basingstoke.
Scarbrough, H. (1996b) Strategic IT in financial services: the social construction of strategic knowledge. In *The management of expertise* (Ed, Scarbrough, H.) Macmillan, Basingstoke, pp. 150-173.
Schneider, K., von Hunnius, J.-P. and Basili, V. R. (2002) Experience in Implementing a Learning Software Organization. *IEEE Software*, **19**, (3), 46-49.
Schreiber, A. T., Wielinga, B. J., Akkermans, J. M., van de Velde, W. and de Hoog, R. (1994) CommonKADS: a comprehensive methodology for KBS development. *IEEE Expert*, **9**, 28-37.
Senge, P. M. (1990) *The fifth discipline, the art and practice of the learning organization.* Doubleday, New York.
Sitton, S. and Chmelir, G. (1984) The Intuitive Computer Programmer. *Datamation*, **30**, (16), 137-140.
Snowden, D. (2000) Cynefin, a sense of time and place: an ecological approach to sense making and learning in formal and informal communities. In Proceedings of *KMAC2000* (Eds, Edwards, J. S. and Kidd, J. B.) Operational Research Society, Birmingham, UK, pp. 1-11.
van Aalst, J.-W. (2001) *Knowledge Management in Courseware Development*. Unpublished PhD Thesis, Technical University Delft.
van Heijst, G., Schreiber, A. T. and Weilinga, B. J. (1997) Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies*, **46**, 183-292.
Weaver, P. L. (1993) *Practical SSADM 4.* Pitman, London.
Williams, R. and Procter, R. (1998) Trading places: a case study of the formation and deployment of computing expertise. In *Exploring expertise: issues and perspectives* (Eds, Williams, R., Faulkner, W. and Fleck, J.) Macmillan, Basingstoke, pp. 197-222.
Winstanley, D. (1986) Recruitment strategies as a means of managerial control of technical labour. In Proceedings of *Labour Process Conference* Aston University, Birmingham.
Wynekoop, J. L. and Walz, D. B. (1998) Revisiting the perennial question: are IS people different? *Database for Advances in Information Systems*, **29**, (2), 62-72.
Zachary, G. P. (1994) *Showstopper! the breakneck race to create Windows NT and the next generation at Microsoft.* Free Press, New York.

Author Biography

John S. Edwards is Professor of Operational Research and Systems at Aston Business School, Birmingham, UK. His principal research interests are in knowledge management and decision support, especially methods and processes for system development. He has published more than 60 research papers on these topics, and two books, *Building Knowledge-based Systems* and *Decision Making with Computers*. Current work includes the transferability of best practices in knowledge management, linking knowledge-based systems with simulation models to improve organisational learning, and an investigation of knowledge management in organizations using group facilitation techniques. He is also editor of the journal *Knowledge Management Research & Practice.*