On Knowledge Management in Software Development Life Cycles

Emerging Postgraduate Research Paper

Ernest Cachia
Dept. of Computer Science
University of Malta
ernest.cachia@um.edu.mt

Mark Micallef
Dept. of Computer Science
University of Malta
mark.micallef@um.edu.mt

ABSTRACT

Software engineering is a knowledge-intensive activity. For software organisations, the main assets are not manufacturing plants, buildings, and machines, but the knowledge held by the employees. Studies [13][19] have shown that projects do not tend to fail because of developers' lack of technical knowledge, but rather for reasons such as *requirements failures*, *communication failures* and *estimation failures*. These failures can be traced back to inadequate knowledge management practices as a root cause. Software development processes tend to address knowledge management issues by prescribing documentation, a strategy which to knowledge management practitioners is known as *codification*. Codification is the process of converting a body of knowledge held within a 'knower' (tacit knowledge) to a form which makes the knowledge permanent and thus easier to transfer (explicit knowledge). As with most strategies/techniques, this approach works in some cases but less so in others.

As software systems become increasingly complex and deadlines increasingly tight, this paper argues that the profession will benefit if software development life cycles evolved to explicitly handle knowledge management aspects. The approach taken here is to provide an overview of some core knowledge management concepts whilst giving examples of how they apply (or not) to the software engineering world. The material presented here is related to ongoing Ph.D. research within the department entitled "A knowledge driven software development life cycle".

Keywords

Software Engineering, Knowledge Management, Software Development Processes

1. INTRODUCTION

Ever since Grace Murray Hopper wrote the first computer program on the *Mark 1* computer in 1944 [22], software development has embarked on an evolutionary journey marked by a number of successes and characterised by numerous and persistent failures. The term "software engineering" was brought into common use when NATO organised a Software Engineering conference in 1968. The conference title was deliberately chosen to be provocative, given that by that time, the tendency for commercial and governmental systems to be delivered late, over budget and lacking functionality was already becoming apparent [7]. Two years later, Royce published an article [20] proposing a development life cycle for managing the development of large software systems, a restricted version of which became popularly known as the waterfall model.

Looking back over the past sixty-six years, software development processes have gone through three overlapping phases of evolution (see figure 1). In the first few decades, the cost of computer hardware meant that most software development happened as a result of government or military contracts. During this period, waterfall-like processes were used due to the nature of government contract rules and also because people tended to engineer software like they engineered hardware. By the mid-1980s, declining hardware costs and 3GLs lead to more commercial software development and the birth of new software development processes such as the Spiral Model and V-Model. However, projects were still failing (mainly due to fluctuating requirements) and as a result, the mid 1990s saw the introduction and gradual adaptation of various agile development methodologies. The fundamental mentality change here was that software engineers had accepted that requirements would be subject to continuous change in response to fluctuating markets.

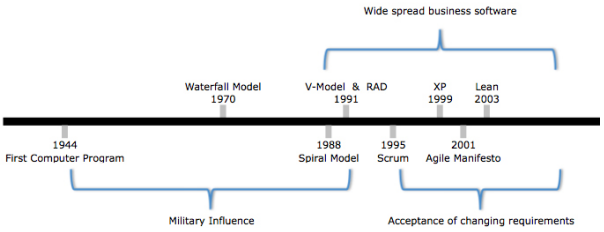

Figure 1: A distilled timeline of software development life cycles

In this paper we propose that the area of software development processes has started veering into a new phase of evolution, one which is heavily influenced by modern software engineering environments. In a market prone to disruptive innovations [21], where geographical boundaries have been torn down and consumers have a wealth of information at their finger tips, software complexity is increasing and deadlines are becoming ever tighter. Also, it is now acknowledged that projects tend to fail for reasons which are not related

to technical incompetence but rather due to factors such as inadequate communication, requirements failures and estimation/scheduling failures [13][19]. These problems are exacerbated by a high rate of staff turnover within the ICT industry since this leads to important knowledge being lost by an organisation. While agile development processes have started addressing some of these issues, we argue that software development processes are entering an era where the focus is less on the product being built and more on the knowledge being created and applied within an organisation.

We also argue that software engineering needs to explicitly adopt concepts from the world of knowledge management where similar problems have been encountered and addressed in the past. It is proposed that the application these concepts to the context of software engineering will result in more effective software development processes and as a result, more successful projects.

In following sections, we present an overview of knowledge management interleaved with references to the world of software engineering.

2. KNOWLEDGE MANAGEMENT

Knowledge management is defined as *the identification, growth and effective application of an organisation's critical knowledge* [5]. The concept of knowledge and its exact definition has been debated by philosophers since the time of the ancient greeks, and to this day is still not totally demystified [2]. Six perspectives of knowledge have been identified in the literature:

1. Knowledge is personalised information [4][14][27]
2. Knowledge is a state of mind [23]
3. Knowledge is an object to be stored and manipulated [3][15][31]
4. Knowledge is a process of applying expertise [1]
5. Knowledge is a condition of access to information [15]
6. Knowledge is a capability - the potential to influence action [3] [29]

A detailed discussion of all knowledge perspectives is out of the scope of this paper but it is worth noting that the way one perceives knowledge within the organisation has a direct impact on how that organisation approaches knowledge management. For example, an organisation that percieves knowledge as an object is likely to focus its knowledge management activities on building and managing knowledge stocks. On the other hand, an organisation which subscribes to the view that knowledge is a process of applying expertise is likely to focus on enabling adequate flow of knowledge between 'knowers' and 'learners' in the company.

2.1 Knowledge Taxonomies

Knowledge taxonomies are important because the simple act of knowing what different taxonomies exist helps one reason more effectively about the knowledge held within one's organisation. The following classifications have been identified in the literature:

Tacit vs Explicit Knowledge - Widely cited, this classification divides knowledge based on whether it resides purely within its 'knower' (tacit knowledge) or whether it has been explicitly articulated, codified or otherwise communicated (explicit knowledge). Szulanski [25] points out that tacit, context-specific and ambiguous knowledge is likely the most difficult to transfer within the firm. Nonaka [16] also alludes to this by stating that tacit knowledge is hard to communicate and is deeply rooted in action, involvement and commitment within a specific context. He also further splits the concept of tacit knowledge into cognitive tacit knowledge and technical tacit knowledge. Cognitive tacit knowledge refers to an individual's mental models, beliefs, paradigms and viewpoints. Technical tacit knowledge on the other hand, is concrete know-how and skills within a specific context [1].

Individual vs Social Knowledge - Individual knowledge is knowledge that is created and homed within an individual. An example of this is the act of a person learning from her mistakes in the course of a project. When a person realizes that a negative event was caused by her own actions, she creates individual knowledge aimed at helping her modify her behaviour in future similar situations. On the other hand, group knowledge is created and inherent in the collective actions of a group. A typical example here might be a team's knowledge of the best way to communicate amongst each other depending on circumstances such as work load, time of day and the criticality of what needs to be communicated. Such knowledge is usually developed over time by the group as a whole and is driven by group experiences and discussions between its members.

Operational Classification - The operational classification splits knowledge types depending on the circumstances in which particular knowledge is likely to be used. In this classification, knowledge is split into five categories: declarative (know-about), procedural (know-how), causal (know-why), conditional (know-when) and relational (know-with) [10][32].

Pragmatic Classification - Finally, a pragmatic approach to knowledge classification attempts to identify knowledge which is useful to a particular organisation [1]. This is more of an ad-hoc view classification which will differ depending on the circumstance in which it is applied. Examples of this classification might be the classification of knowledge within a software development company as consisting of development processes, system architectures, acceptance criteria, project experiences, and so on.

3. KNOWLEDGE WITHIN SOFTWARE DEVELOPMENT PROCESSES

For the most part, the numerous development life cycles in existence today are a re-organisation of a number of core activities: requirements specification, system design, coding, testing and deployment. The focus is on the software product and what is being done with respect to the product at any point in time. However, one must keep in mind that the

development process is implemented by a number of human participants whose knowledge and expertise are depended upon by the organisation. In general, software engineers need to possess two main types of knowledge in order to be effective in their job. The first is *technical knowledge*. This refers to their knowledge of appropriate programming languages, technologies and so on. Secondly, engineers also need to have *domain knowledge* relevant to whatever system they are working on. If one is working on an insurance application, then one needs to be familiar with insurance procedures and regulations. Furthermore, the domain knowledge might need to be refined to the organisational context which the engineer is working in. When hiring software engineers, it is relatively easy to find ones with the right technical knowledge but less so to find people with domain knowledge catered to one's needs. Hence it is important to look at software development organisations from a knowledge management view, yet software development life cycles tend not to do this.

Various knowledge management processes exist but most can be distilled down to a generic process of four stages (see figure 2). These are *knowledge creation*, *knowledge storage and retrieval*, *knowledge distribution* and finally *knowledge application*. The following sections will discuss each of these in turn in the context of software development practices.

Figure 2: A generic knowledge management process

3.1 Knowledge Creation

Creation of organisational knowledge refers to the creation or updating of stocks of knowledge within the organisation. These stocks of knowledge are usually a mixture of tacit and explicit knowledge (refer to section 2.1). Nonaka [16] proposes a comprehensive model of organisational knowledge creation which identifies four modes of knowledge creation - socialisation, externalisation, internalisation, and combination (see figure 3).

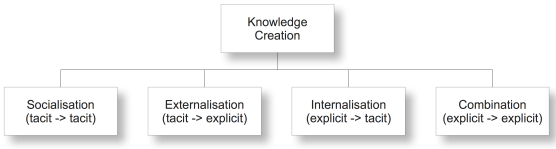

Figure 3: Nonaka's four modes of knowledge creation [16]

Socialisation involves the conversion of tacit knowledge to new tacit knowledge based on social interactions [1] such as internships. Very few software development processes support this form of knowledge creation. However, one good example would be the concept of *pair programming*, part of the *extreme programming (XP)* suite of practices. Pair programming involves two developers working together on the same machine. As they work and observe each other's practices they both learn from each other, thus creating new knowledge.
Externalisation refers to the conversion of tacit knowledge to explicit knowledge. This is usually done by creating documents, communicating via e-mail, communicating knowledge verbally, and so on. *Internalisation* converts explicit knowledge to tacit knowledge by absorbing externalised knowledge from other sources. For most software development life cycles, this is the only way knowledge creation is supported. At the end of each phase, knowledge is externalised into a document (e.g. a business analyst creates a requirements document at the end of the specification phase) and at the beginning of the next phase, the document is internalised by someone else (e.g. an engineer reads the requirements at the start of the design phase). This is effective in a world where knowledge is static but unfortunately, the tendency is for such documents to become irrelevant or unreliable very shortly after being created.

Finally, the *combination* mode of knowledge creation refers to the creation of new explicit by combining multiple explicit knowledge components. For example, one might analyse and merge different statistical results to form new conclusions about a particular subject matter. In a software engineering context this does not seem to have obvious direct implications but one might apply combination when combining different software engineering reports. For example, one might combine test coverage analysis reports with live bug reports to identify relationships between the two.

Researchers claim that providing the mechanisms for all these modes of knowledge creation to occur is central to successful organisational knowledge creation[16][17]. From a software engineering stand point, we argue that we focus too much on externalisation and internalisation and not enough on socialisation. If an organisation relies too much on storing knowledge in documents, it risks loosing critical knowledge unless the explicit knowledge is kept updated. In today's marketplace, keeping documentation updated tends to take a back seat to development work.

3.2 Knowledge Storage and Retrieval

Alavi [1] points out that studies have shown that while organisations create knowledge and learn, they also forget. As the philosopher George Santayana's famous adage goes, "he who does not learn from the past is condemned to repeat it". This brings to the forefront the concept of organisational memory [24][28], i.e. *the means by which past knowledge, experience, and events influence organisational activities in the present time.* Although the concept of maintaining a good organisational memory is widely believed to be beneficial [11][30], some researchers warn against its negative aspects. In particular, Leonard-Barton [12] presents the possible positive and negative effects of organisational memory on a firm's performance through the concepts of *core capabilities* and *core rigidities*. Core capabilities refer to organizational know-how and competencies that lead to a competitive advantage for a firm. However, these same capabilities can turn into *core rigidities* in the face of major change which may be required from time to time in a competitive dynamic market [1]. That is to say that by over relying on the way it has done things in the past, an organ-

isation risks missing opportunities to adapt to a changing market and environment. One can easily relate this risk to software organisations where new technologies and techniques are constantly going in and out of style. If an organisation is too rigid in what it does and how it does it, it risks becoming obsolete.

From a software engineering standpoint, the maintenance of organisational memory has traditionally taken the form to various official documentation formats. However recent years have seen the increased use of corporate intranets, wikis and defect tracking tools. In the knowledge management camp, such approaches form part of the *Organisational* school of knowledge [6]. However, six other schools of thought exist in this regard, all with valid arguments about how to approach knowledge management. It would be worth relating the approaches from all schools to software engineering in an attempt to find alternative ways for maintaining organisational memory.

3.3 Knowledge Distribution

In an interesting keynote at the 2004 International Conference on Software Engineering and Knowledge Engineering entitled "Collecting the Dots" [18], Shari Pfleeger referred to real world incidents whereby failures in knowledge distribution led to catastrophic results. Of particular relevance at the time, she referred to the terrorist attacks of September 11th 2001 and how the intelligence to prevent this event had been available to the US intelligence services in the months leading up to the attack. However, the information was scattered between different agencies and branches around the globe and this led to a situation where no one was able to see all relevant information at one go and realise what was going on. Although this example may be extreme, similar examples from software engineering have led to inefficiencies, reinvention of the wheel, late projects and financial losses. Huber [9] claims that organisations do not really know what they know and have weak systems for locating and retrieving their knowledge. This essentially means that knowledge distribution has a key role to play in knowledge management and organisational learning, and as such should be given adequate attention.

Distribution of knowledge (also referred to as knowledge transfer) is characterised by the source-target paradigm. That is to say, in order for knowledge transfer to occur, three elements must be present: a source of knowledge, a receiver of that knowledge, and one or more channels through which the knowledge transfer occurs. Much research has been carried out into factors which facilitate effective knowledge transfer. Gupta and Govindarajan [8] propose that effective knowledge transfer depends on 5 key elements. These are (1) the perceived value of the source unit's knowledge, (2) the source unit's willingness to share that knowledge, (3) the existence and richness of transmission channels, (4) the receiving unit's willingness to receive the knowledge, and (5) the absorptive capacity of the receiving unit. All these have parallels in software development. For example, an engineer could have a bright new idea but is prevented from pushing forward with her team because management does not think it will be of a high enough value to warrant 'wasting' time on it. Or in a different example, an engineer might have a valuable idea but is reluctant to share it for fear of loosing his competitive advantage. These are real problems with are faced every day, yet development processes will not address them unless a knowledge-based view of software development is adapted.

Despite the risks involved when knowledge is not properly transferred within an organisation, most development processes do not explicitly tackle knowledge distribution. Some agile processes attempt to improve the situation by promoting open work spaces, moving people around on a regular basis, and having daily stand up meetings. These measures do help but they are simply addressing one success factor by increasing the existence and richness of transmission channels. Further work needs to be done in terms of addressing other success factors when if comes to knowledge transfer. For example, researchers in knowledge management have acknowledged that the existence of a strong co-operative and collaborative culture is an important prerequisite for knowledge transfer [26]. Unfortunately current reward structures based on individual appraisals tend to lead to collaboration and sharing what you know being perceived as detrimental to personal career goals.

3.4 Knowledge Application

Arguably, the most important aspect of the knowledge based view of software engineering organisations is the application of the knowledge rather than the maintenance of organisational knowledge itself. The latter tends to exist in order to enable the former, thus creating organisational capability. Knowledge application can be achieved through three primary mechanisms: directives, organisational routines, and self-contained task teams. Directives are a top-down mechanism whereby instructions are issued to employees based on some expert knowledge. In software engineering for example, a directive might instruct developers to follow a particular coding standard. Organisational routines refer to followed processes / protocols within the organisation which are designed to allow individuals to apply and integrate their specialised knowledge. Lastly, self contained task teams can be set in situations which task uncertainty and complexity prevent specification of directives or organisational routines. It is safe to say that all these approaches are utilised in software development, albeit not entirely intentionally. However, we argue that the profession stands to benefit from deliberate evaluation of how knowledge is applied in development teams and whether alternative approaches can be adapted based on research from the knowledge management camp.

4. ONGOING WORK

In this paper, we have attempted to provoke the reader into thinking about software development processes from a knowledge management perspective. The concepts covered here are amongst the most basic in knowledge management literature and there is more to explore. As part of ongoing research, we are currently working on detailed analysis of parallels between knowledge management and software engineering. Our goal is to develop and evaluate the effectiveness of a *knowledge-driven* development life cycle in which knowledge-management takes primary focus over actual development itself. The driving idea is that if participants focus on maintaining the right knowledge flows and organisation memory, quality software would naturally follow.

5. REFERENCES

[1] Maryam Alavi and Dorothy E. Leidner. Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS Quarterly*, 25(1):107–136, 2001.

[2] Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann, May 2004.

[3] S. Carlsson, O. ElSawy, Inger V. Eriksson, and A. Raven. Gaining competitive advantage through shared knowledge creation: In search of a new design theory for strategic information systems. In *Proceedings of the Fourth European Conference on Information Systems*, 1996.

[4] Fred Dretske. *Knowledge and the Flow of Information.* MIT Press, 1981.

[5] N. Duffy. Benchmarking knowledge strategy. *Leveraging Knowledge for Business Performance 1999: Knowledge In Action*, 1999.

[6] Michael Earl. Knowledge management strategies: Toward a taxonomy. *Journal of Management Information Systems*, 18(1):215–233, May 2001.

[7] John Edwards. Managing software engineers and their knowledge. *Managing Software Engineering Knowledge*, pages 5–27, 2003.

[8] Anil K. Gupta and Vijay Govindarajan. Knowledge flows within multinational corporations. *Strategic Management Journal*, 21(4):473–496, 2000.

[9] G. P. Huber. Organizational learning: The contributing processes and the literatures. *Organization Science*, 2(1):88–115, 1991.

[10] Nolan Norton Institute. *"Putting the knowing organization to value" white paper.* Nolan Norton Institute, 1998.

[11] A. M. Kantrow. *The Constraints of Corporate Tradition.* Harper and Row, 1987.

[12] Dorothy Leonard-Barton and Dorothy Leonard. *Wellsprings of Knowledge: Building and Sustaining the Sources of Innovation.* Harvard Business School Press, April 1998.

[13] K Linberg. Software developer perceptions about software project failure: a case study. *Journal of Systems and Software*, 49(2-3):177–192, December 1999.

[14] Fritz Machlup. *Knowledge : its creation, distribution, and economic significance.* Princeton University Press, Princeton, N.J. :, 1983.

[15] Robert J McQueen. Four views of knowledge and knowledge management. In *Proceedings of the Fourth Americas Conference on Information Systems*, pages 609–611, 1998.

[16] I. Nonaka. A dynamic theory of organizational knowledge creation. *Organization Science*, 5(1):14–37, 1994.

[17] I. Nonaka and H. Takeuchi. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation.* Oxford University Press, USA, May 1995.

[18] S.L. Pfleeger. Collecting the dots. In *Proceedings of the 2004 International Conference on Software Engineering and Knowledge Engineering.* Knowledge Systems Institute, 2004.

[19] Roger Pressman. Fear of trying: The plight of rookie project managers. *IEEE Softw.*, 15(1):50–51,54, 1998.

[20] W Royce. Managing the development of large software systems. In *9th International Conference on Software Engineering*, pages 328–338, Monterey, California, 1987. IEEE Computer Society Press.

[21] Reza Samavi, Eric Yu, and Thodoros Topaloglou. Applying strategic business modeling to understand disruptive innovation. In *Proceedings of the 10th international conference on Electronic commerce*, ICEC '08, pages 15:1–15:10, New York, NY, USA, 2008. ACM.

[22] Jean E. Sammet. Introduction of captain grace murray hopper. *History of programming languages I*, pages 5–7, 1981.

[23] P. Schubert, D. Lincke, and Schmid B. A global knowledge medium as a virtual community: The netacademy concept. In *Proceedings of the Fourth Americas Conference on Information Systems*, pages 618–220, 1998.

[24] E. W. Stein and V. Zwass. Actualizing organizational memory with information systems. *Information Systems Research*, 6(2):85–117, June 1995.

[25] Gabriel Szulanski. Exploring internal stickiness: Impediments to the transfer of best practice within the firm. *Strategic Management Journal*, 17:27–43, 1996.

[26] F. Tao and W. Meng. Evaluation model of mnes knowledge flow management. *Knowledge Enterprise: Intelligent Strategies in Product Design, Manufacturing, and Management*, pages 71–78, 2006.

[27] D Vance. Information, knowledge and wisdom: The epistemic hierarchy and computer-based information system. In *Proceedings of the Third Americas Conference on Information Systems*, 1997.

[28] J.P. Walsh and G.R. Ungson. Organizational memory. *The Academy of Management Review*, 16(1):57–91, 1991.

[29] R.T. Watson. *Data Management: Databases and Organizations.* John Wiley, 1999.

[30] A.L.. Wilkins and Brestow N.J. For successful organization culture, honor your past. *Academy of Management Executive*, 1:221–229, 1987.

[31] M. Zack. An architecture for managing explicated knowledge. *Sloan Management Review*, 1998.

[32] M. Zack. What knowledge-problems can information technology help to solve. In *Proceedings of the Fourth Americas Conference on Information Systems*, pages 644–646, 1998.