Enabling the Creation of Knowledge about Software Assets

Paul Klint
Centrum voor Wiskunde en Informatica and University of Amsterdam
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands,
e-mail: Paul.Klint@cwi.nl

and

Chris Verhoef
Free University
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands,
e-mail: x@cs.vu.nl

December 4, 2001

Abstract

In most companies two factors play a crucial role: managing the knowledge that is necessary for doing business and managing the hardware and software infrastructure that supports the business processes. Usually, business processes and infrastructure are not optimally aligned.

We investigate how principles from knowledge management can be applied to enable the creation, consolidation, conservation and continuous actualization of knowledge about valuable software systems ("software assets") that are part of the infrastructure.

Our point of departure is a generic framework for knowledge creation proposed by Von Krogh, Ichijo and Nonaka. We investigate the explicit and tacit knowledge about software assets that may exist in an organization and specialize the framework to obtain a strategy for creating new knowledge about these software assets. By applying this strategy, one can optimize the quality and the flexibility of the software assets while reducing costs.

KEY WORDS: Knowledge management, knowledge creation, software asset management, software engineering, software maintenance, software renovation.

1 Managing Knowledge versus Managing Knowledge Creation

It is widely believed that *knowledge management* should be a key factor in the strategy of every modern company. Knowledge management has managerial as well as technical aspects. From the management perspective, it should contribute to the company's global strategy and be included in the standard operating procedures [19, 20]. Of particular importance are measures to promote knowledge management such as incentives to share knowledge and procedures to use shared knowledge. From a technical perspective, information systems or knowledge bases are used to store the shared knowledge.

In practice, knowledge management does not always live up to its promises. There are a variety of reasons for this (also see [10]):

- Much knowledge in an organization is "tacit", i.e., unformalized knowledge that is only known to a small community inside a company. Knowledge management relies too much on easily detectable, quantifiable, information.
- Technical solutions dominate managerial solutions and knowledge management reduces to "filling a database".
- A "knowledge officer" is responsible for the knowledge management process. From a managerial perspective this is a reasonable approach, but in most cases the knowledge officer is a staff member and is too far away from the organizational units where actual knowledge is being generated (manufacturing departments, marketeers, and the like). In addition, knowledge creation is a serendipitous process that can only be *enabled* but it cannot be *controlled* as many knowledge officers have found out the hard way.
- Knowledge is not static but dynamic: it is created, it is used and it becomes obsolete. These dynamics are not well supported by standard knowledge management practices.

Knowledge management is an economically rational theory stating that sharing knowledge leads to better (organizational) intelligence than without sharing. The consequence is that using this shared knowledge, organizations can compete much better than would otherwise be possible. As the saying goes: *Knowledge is power.* Knowledge management is indeed strongly connected to power of expertise. This truism not only applies to competing organizations but also to the social systems within a single organization.

A well-known phenomenon inside organizations is the "knowledge czar": an individual who has large informal power based on the unique knowledge that he or she possesses (power of expertise). The strong relation between power and knowledge thus effectively blocks knowledge sharing and dissemination throughout the organization. Without proper *power management*, successful knowledge management may easily fail. However, it is hard to manage power [13, p. 7]:

> The greater the recourse to power, the stronger the desire for it, just as the use of 'hard drugs' will result in a stronger craving for drugs.

Power is not explained in economically rational terms, but in terms of addiction. Therefore, the quantity of power is a more decisive factor than the quality of it [13, p. 16]. This is the clarification for another truism: *Power Corrupts*. So without proper power management, one could say that *Knowledge Corrupts*. This may be one of the reasons that knowledge management does not live up to its promises.

The general observation is therefore that knowledge cannot be controlled or managed in a rational, top-down fashion like other assets of an organization as is assumed by knowledge management. As a response to this observation, current literature is focusing on the question how the process of knowledge creation can be enabled in such a way that the special properties of knowledge are taken into account.

In this paper, we address the question how general strategies for enabling knowledge creation can be used to increase the knowledge about an organization's *software assets*: all software systems that support the realization of its goals. These software systems provide functions like information management, billing, telecommunications, e-commerce and the like. In this case, there is first business knowledge that leads to requirements for software systems and these requirements are used to build the desired systems. Successful systems generate new requirements thus further obfuscating the knowledge about old and new requirements. As a consequence, business knowledge gets concealed in the program code. Unfortunately, this step cannot easily be reversed: having the code does not mean that one can understand it and can recover the original requirements. To use an analogy: knowing the human genome does not automatically lead to understanding humans.

To complicate matters even further, the software assets are continually evolving over time due to changed business insights (leading to changing requirements) and technical evolution of the platforms on which the software depends. In the case of software assets there are also ample opportunities for knowledge czars having expertise about some subsystems, but in the end no one understands the software assets to their full extent. This is largely due to maintenance. Or more precisely, this is due to the lack of knowledge management during the evolution of software assets which becomes manifest during maintenance. Weinberger calls this the "maintenance masking dynamic". In [22, p. 243], he describes the problems that occur when an organization fails to manage the tacit knowledge of maintenance teams.

> There is another essential component to maintainability: the competence of the crew, which is affected by turnover, training, and management attitude toward maintenance. Because people are naturally learners, the competence of the crew to maintain a particular system will tend to grow over time, possibly masking the deterioration of the code itself. But the crew's competence must also be

> maintained, largely by providing them with tools, training, and resources for the job. If there should be a sudden exodus from the maintenance crew, management will quickly discover how ugly a situation has been allowed to fester in the code, masked by the growing competence of the people.

The central problem addressed in this paper is therefore how to obtain circumstances that enable the creation, consolidation, conservation and continuous actualization of knowledge about software assets. We proceed as follows. In Section 2 we summarize a state-of-the-art framework for knowledge creation and knowledge enabling. A central notion in this framework is the distinction between tacit and explicit knowledge. In Section 3 we give an overview of implicit and explicit knowledge about software assets. In Section 4 we show how the general framework helps to understand the mechanisms to increase the knowledge about software assets. Our conclusions are presented in Section 5.

2 Managing Knowledge Creation

Knowledge creation is an inherently bottom-up process while standard management practices are of a top-down nature. Managing knowledge creation is therefore a balancing act to reconcile bottom-up and top-down processes: the chaos of knowledge creation and the order of management.

2.1 Knowledge Creation Steps

In [15] the following five *knowledge creation steps* are distinguished:

Sharing Tacit Knowledge. As already pointed out above, the basis of organizational knowledge creation is the tacit knowledge held by individuals. This knowledge cannot easily be communicated directly to others since it is implicit, has not yet been verbalized, and depends on the physical and mental circumstances in which it has emerged. Instead, indirect mechanisms are needed such as joint problem solving in small knowledge communities and teacher/apprentice relationships. The result of this is a shared mental model of the tacit knowledge. Knowledge communities are in most cases completely unrelated to the actual organizational structure.

Creating a Concept. Based on the shared mental model, a verbalization of the tacit knowledge is made. Metaphors and figurative speech are useful tools to achieve this.

Justifying a Concept. The concepts that are created in the previous step are screened for relevance and worth for the organization. Note that other parts of the organization will be involved in the justification process.

Building a Prototype. The justified concept is turned into a tangible prototype: either a working prototype of a product or a model of a service. Note that other parts of the organization (e.g., manufacturing, marketing) will be involved in the building of a prototype.

Cross-Leveling Knowledge. The previous two steps already involved other parts of the organization, but once the prototype is completed it is used to propagate the knowledge that it embodies throughout the whole organization.

2.2 Knowledge Enablers

Using the above five knowledge creation steps as point of departure, the question arises what circumstances can be created that enable this knowledge creation process. As already pointed out, hierarchical and technological solutions are not sufficient. Rather mechanisms must be found that increase the awareness of knowledge creation in all "veins" of the organization.

In [10], the following *knowledge enablers* are identified, that support the knowledge creation steps in various ways:

Instill a Vision. Rather than a top-down managerial approach to knowledge creation, a *knowledge vision* should succinctly describe the relevance of knowledge creation for the organization. It should create awareness for knowledge creation at all levels and it should identify knowledge sharing as a company value. It should create trust, care and cooperation rather than suspicion, indifference and internal competition.

Manage Conversations. Conversations over a cup of coffee form the social fabric of each organization [12]. They also form the primary mechanism for sharing tacit knowledge. Personal dialogues are one of the most effective mechanisms for information and knowledge exchange. Managing conversations amounts to creating an environment in which everyone can participate in (semi-formal) conversations and make valuable contributions. A conversation manager may define explicit rules for conversational etiquette, intervene and direct conversations, and introduce innovative language to describe concepts and ideas. Form of and rules for conversation differ for each knowledge creation step.

Mobilize Activists. Knowledge activists are persons that facilitate the knowledge creation process. They may act as *catalyst* and start a new initiative by bringing together the right people. They may act as *coordinator* by creating the right context and by making connections with the global knowledge vision as well as with related local initiatives in the organization. They may act as *merchant* by attracting attention for an initiative in other parts of the organization. As opposed to the traditional knowledge officer who tries to control the knowledge creation process, the knowledge activist aims at enabling it [21].

It is known from sociology that many people find jobs via personal contacts. The majority of the personal connections are not close friends but so-called "weak ties". Weak ties are outside the person's inner circle and have knowledge that differs from the knowledge of the people closer by. Granovetter who discovered this in the 1970s called this: *the strength of weak ties* [6]. The more acquaintances a person has, the more powerful he or she becomes. Knowledge activists have the professional goal to connect the right people and by doing so they can become socially very powerful. Proper power management should be applied to avoid that this unintended potential power concentration compromises the long term goal of knowledge sharing.

Create Right Context. How can one create the right context for knowledge creation? When answering this question several paradoxes become manifest. How to combine central managerial control with flexibility? How to reconcile the top-down formal structure of an organization with spontaneous structures that have emerged in a bottom-up fashion among knowledge workers working in different departments of the same company or even of other companies (e.g., customers)? How to moderate between the need for technological advancement and the need to survive as a company? There is no one-size-fits-all solution for this. It is important to understand that a proper context should support the cycle of sharing individual tacit knowledge, documenting it, and again internalizing it at the group level. From an organization perspective various solutions exist ranging from task forces and empowered divisions to cross-divisional units.

Globalize Local Knowledge. The final enabler aims at transferring locally created knowledge throughout the perhaps globally distributed company. All phenomena known from diffusion theory [21, 8] apply here: this theory explains how innovations diffuse into society and organizations. The knowledge has to be transferred between a creator and a receiver and psychological, sociological as well technological barriers have to taken. In [10] this is formulated as a three-staged process: *triggering* the process of recognizing a business opportunity and relating it to knowledge available in some part of the company (using knowledge activists, workshops, bulletin boards, and the like), *packaging and dispatching* this knowledge, and *re-creating* it at the site of the receiver. The underlying idea is that knowledge transfer is not a verbatim copying operation from sender to receiver but that it has to consider the implicit and explicit knowledge of sender and receiver as well as the local circumstances in which the knowledge has to be re-created and applied.

In Table 1 the relation between knowledge creation steps and knowledge enablers is sketched. An empty field denotes no correlation, a + indicates a moderate correlation, and ++ indicates a strong correlation. We refer again to [10] for an extensive motivation of this table.

		Knowledge Creation Steps				
		Sharing tacit knowledge	Creating a concept	Justifying a concept	Building a prototype	Cross-Leveling Knowledge
Knowledge Enablers	Instill a Vision		+	++	+	++
	Manage Conversations	++	++	++	++	++
	Mobilize Activists		+	+	+	++
	Create Right Context	+	+	++	+	++
	Globalize Local Knowledge					++

Table 1: Relation between knowledge creation steps and knowledge enablers.

3 Tacit and explicit knowledge about software assets

The *software assets* of an organization are formed by all software systems that support the realization of the organization's goals. In principle, software assets are like other tangible assets of an organization: they can be captured by standards and technical procedures can be used to control their development. However, software is also unlike many tangible assets, e.g., a new automobile runs, but new software usually not (cf. [11]). The older an automobile gets, the more chance it stops running, but the older software gets, the more chance it starts doing what it should have done in the first place. Another difference is that software congeals valuable business knowledge as time goes by, while this is not the case for a tangible asset like an automobile.

In practice, however, the ideal situation that software assets are being managed like ordinary tangible assets is far from being achieved and the creation and management of software assets is either non-existent or is done in an *ad hoc* fashion. It certainly does not resemble an engineering discipline. This makes it even more urgent to consider the question how knowledge about software assets is being created, managed and used. As we have seen above, knowledge as such behaves differently from ordinary assets and the same is true for knowledge about software assets.

Knowledge about software assets is clearly important and the question arises what the implicit and explicit knowledge about software assets amounts to. To answer this question, we must first understand that there are differences between organizations in the way they deal with their software assets. The various dimensions in this space are:

- Application software is developed in-house or by third parties (insourcing).
- Applications are operated in a corporate computer center or by a third party (application services provider).
- Maintenance is done in-house or by third parties (outsourcing).

In practice, mixtures of these extremes are quite common. In order to simplify the presentation, we discuss the knowledge about software assets from the perspective: in-house software development, corporate computer center, and in-house maintenance. It is however, straightforward, to adapt the following overview to other perspectives.

In the following paragraphs we summarize the knowledge areas architecture, application area, construction, implementation, operations, maintenance, performance, quality and costs. In each case we list relevant knowledge items and the frequently occurring forms of explicit and implicit knowledge. By default, the implicit knowledge consists of all instances where the explicit knowledge is incomplete, erroneous or out of date. It also covers all undocumented aspects or features. Also note that the division between explicit and tacit knowledge depends on the *maturity* of the organization as we will further discuss in Section 4. The division we present below, applies to the vast majority of organizations.

3.1 Architecture

Architecture concerns the global structure and functionality of the software assets, such as:

- Overall Architecture of all systems and applications.
- Operating systems, databases, networking, user-interfaces.
- Local as well (inter)national standards.
- Software engineering methods and tools (including design, construction, implementation, and maintenance).
- Global inventory of all software assets.

Explicit knowledge. White papers describing architecture, and design principles; architecture diagrams; standards.

Tacit knowledge. Quality, performance and cost aspects of the architecture as a whole as well as of individual applications, engineering techniques and tools.

Illustration. The need for knowledge management for architecture is illustrated by the following example. The United States General Accounting Office (GAO) reviewed the Defense Logistics Agency (DLA). This review comprised the efficiency and effectiveness of meeting customer requirements, application of best practices, and opportunities for improving DLA operations. The importance of architecture is made explicit in this review [16]:

> DLA does not have an enterprise architecture to guide its investment [...] even though Department of Defense policy requires their use. Rather, DLA plans for creating an architecture as a by-product [...] Moreover, DLA's architecture development plans address only one, albeit the largest, of its six primary business areas [...] According to DLA's plans, its architectural products will not be extended to its other business areas until 5 years from now. This nonagencywide approach to developing and implementing an enterprise architecture is not consistent with federal guidance, and it increases the risk that DLA will modernize in a way that optimizes an individual business area but does not optimize agencywide logistics management performance and accountability.

So, the GAO makes clear that enterprisewide knowledge sharing is crucial in order to achieve the desired results.

3.2 Application area

The application area covers the overall goals and techniques relevant in a certain application area, such as:

- Business goals, application concepts, and standard operating procedures.
- Technical concepts, standards and procedures.
- Markets and products.

Explicit knowledge. Handbooks describing the application area; market and tool surveys.

Tacit knowledge. Knowledge about new, immature, application areas that is not available in handbooks; up-to-date knowledge of the market; experience with state-of-the-art tools.

Illustration. It is well-known that software can become so complex that repairing one error leads to another error. In that case, the fault injection rate has approached 100% and the project has entered the so-called *complexity catastrophe* [3]. In reaction to this, organizations tend to discard the past, plan to build a completely new software system, and by doing so they destroy valuable knowledge. This will lead to so many errors in the new system that the new system will not be acceptable.

Implementation of Enterprise Resource Planning (ERP) packages is a form of discarding the past. Seen from the knowledge perspective, it should not come as a surprise that The Standish Group has estimated that over 90% of ERP projects end up behind schedule or over budget. Discarding the past, leads to the so-called error catastrophe [3]. The solution is to exploit the old, while exploring the new, which is in fact using knowledge management of the application effectively.

3.3 Construction

Using software engineering methods and tools as well as knowledge about an application area, software for that application area can be constructed. Relevant topics are:

- Software engineering methods and tools (design, construction, testing, documentation).
- The software development environment (including programming languages, compilers and other construction tools).
- New and existing libraries and utilities.
- Procedures and tools for unit testing.
- Detailed inventory and analysis of the application software (including all programs, databases, and user-interfaces and their interrelationships and relevant metrics.)

Explicit knowledge. Software engineering handbooks; vendor reference manuals (for programming languages, tools, procedures); in-house developed documentation (for existing applications, libraries and utilities); maintenance and testing history of all programs.

Tacit knowledge. Quality, performance and cost aspects of the application; qualitative and quantitative assessment of maintenance and testing history (i.e., which are good and bad programs); quality of individual programmers; up-to-date knowledge about the application that extends or replaces the explicit knowledge (e.g., changes in program interfaces, performance problems in certain library functions, changed algorithms, new program dependencies, the effects of foreseen changes in other applications or libraries).

Illustration. The Weinberg-Schulman experiment [23] is a clear illustration of the role of explicit knowledge during construction. Making the goals for construction *explicit* has the effect that you get what you asked for. However, without explicit goals, you also get what you asked for. In this case construction

		Achievements				
		Minimize effort to complete	Minimize number of statements	Minimize memory required	Optimize program clarity	Optimize program output
Goals	Minimize effort to complete	1	4	4	5	3
	Minimize number of statements	2-3	1	2	3	5
	Minimize memory required	5	2	1	4	4
	Optimize program clarity	4	3	3	2	2
	Optimize program output	2-3	5	5	1	1

Table 2: Results of the Weinberg-Schulman experiment

is optimized according to the goal that has been communicated implicitly, e.g., deliver as quickly as possible. In this experiment, 5 programming teams were given the same job, but each team got a specific explicit goal to do the work. The findings are summarized in Table 2. Each team optimized indeed according to the explicit goal, and none of them performed consistently on the other goals.

Knowledge management helps to make such goals explicit, to recognize that some of them conflict, and to achieve the desired goals.

3.4 Implementation

The word “implementation” is ambiguous. In the computer science literature it means “building software”. In the parlance of software development for business applications it usually means “introducing software in a production environment”, e.g., an enterprise resource planning system from some vendor is implemented in a specific organization. In this paper, we will use the latter meaning. Relevant knowledge topics are:

- The production environment.
- Procedures and tools for testing the integration of a new application in the production environment.

Explicit knowledge. In-house developed documentation (testing procedures); vendor reference manuals (operating systems, databases, networking, user-interfaces); test histories.

Tacit knowledge Quality, performance and cost aspects of transferring the application to the production environment; qualitative and quantitative assessment of test histories.

Illustration. Traditionally, development tools may be replaced from time to time, but the production environment tends to be immutable. This implies that vendor' tools or products that are part of the production environment will be in use over a very long period of time.

It turns out that not all vendors can guarantee this. Some are taken over by competitors who have the explicit goal to kill the competing product, others just go out of business due to lack of profitability. From this perspective, it is not surprising that many organizations have started projects to eliminate "exotic" 4GLs and GUI generators and replace them by main stream solutions. The lesson here is that implicit knowledge about the software and tool market may be crucial for the long term implementation and operations strategy.

3.5 Operations

Operations entail the day-to-day production usage of all software assets. This is typically done in a corporate computer center. Knowledge topics include:

- Scheduling and optimization of jobs in the production environment.
- Monitoring of the production environment.
- Backup procedures.
- Trouble shooting.

Explicit knowledge. In-house developed documentation (operating procedures for the production environment); vendor reference manuals; operations history.

Tacit knowledge. Quality, performance and cost aspects of running applications in the production environment; qualitative and quantitative assessment of operations history (e.g., which applications cause problems, how quickly can problems be resolved).

Illustration. It is known from accident analysis that 60 to 80% of all errors are attributed to *operator errors* [18, p. 9]. This research has been done on complex systems ranging from nuclear plants and dams, to tankers and airplanes. These so-called operator errors, are more a blame of the victim (the operator) than that the cause of the error can be attributed to the operator. Operators are confronted with ultra complex systems, have to deal with incomplete information, or even contradictory data. Simultaneously, they have to decide sometimes rather fast to prevent disaster. Such complex systems are all very software intensive, and the lack of knowledge and knowledge sharing increases the chances that operators make errors.

3.6 Maintenance

Maintenance occurs when an application program fails to perform as required during operations. This may be discovered during operations (program crashes or does not terminate) or after wards (program computes wrong answers). Knowledge topics include:

- The application area.
- The application program.
- Software engineering methods and tools (testing, debugging).
- The software development environment (debugging and testing).

Explicit knowledge. In-house developed documentation (application programs, debugging and testing procedures); vendor reference manuals (tools); maintenance history; test history.

Tacit knowledge. Qualitative and quantitative assessment of maintenance and test history.

Illustration. Lack of knowledge sharing hinders optimal deployment of existing software assets. The maintenance masking dynamic (already explained in Section 1) is a prime example of the relevance of knowledge management for maintenance.

3.7 Performance, quality and costs

For each of the above areas knowledge about the required and achievable performance, quality and costs. More precisely, for each application, knowledge is required about:

- Detailed inventory of the application software.
- Development costs.
- Quality of service during operations.
- Costs during operations (response time, resource usage, trouble shooting, human resources, software, hardware).
- Maintenance costs.
- Economic value from a business perspective.
- Qualitative and quantitative assessment of all cost factors.

The inventory of all software applications combined with the detailed knowledge about each application makes it possible to obtain knowledge bout performance, quality and costs of the complete software portfolio.

Explicit knowledge. Databases with history information gathered during construction, implementation, operations and maintenance; qualitative and quantitative assessment of the information in these databases.

Tacit knowledge. In many cases history and performance information is not gathered in a systematic fashion and the assessment of performance, quality and costs has to be judged by individuals based on incomplete and subjective insights.

Illustration. The Y2K problem has demonstrated the need to share knowledge. In many companies there was no detailed knowledge about the software portfolio: which systems were in use, which systems were no longer functioning, etc. A complete lack of such information effectively blocks a solution to system wide problems like the Y2K problem, the Euro conversion, and others.

At the beginning of most Y2K projects this information had to be collected at high costs in order to start the actual Y2K conversion. It is sobering to observe that this same knowledge had to be collected at the start of many Euro projects as well. Clearly, in some companies there has been no knowledge sharing between these projects.

The lack of inventory information prohibits organizations to have insight in their total IT spending. Proper knowledge management can potentially solve some problems related to IT-spending, and more important to IT-wasting.

4 Increasing the knowledge about software assets

From the analysis in the previous section, it becomes clear that knowledge about software assets can be subdivided in three areas:

- The software development process.
- Operations.
- Maintenance.

The maturity of the *software development process* can be judged by the Capability Maturity Model (CMM) as developed by the Software Engineering Institute of Carnegie Mellon University [17]. CMM distinguishes the following five levels:

1. *Initial level. Ad hoc*, informal management practices are used. Characteristics of the software (quality, performance) and the software process (budget, schedule) are unpredictable.
2. *Repeatable level.* Formal management, quality assurance and version control are in place. The outcome of similar projects becomes predictable.

However, there is still a major dependence on the management quality of individuals.

3. *Defined level.* A formal software development process is in place and there is a basis for qualitative process improvement.

4. *Managed level.* The formal development process is complemented with a formal programme for quantitative data collection. Quantitative process improvement is enabled.

5. *Optimizing level.* Continuous process improvement is budgeted and planned and is an integral part op the organization's process.

As one can see, going from level 1 to level 5, the knowledge about the software process is first made explicit, then it is used for qualitative improvements, then data collection about the process starts and finally these data are used to optimize the process.

From a knowledge engineering perspective, CMM judges the amount of explicit knowledge about the software development *process*. It does not cover operations and maintenance, but proposals to extend the model in those directions exist [14]. It does not cover people management either, but an extension for this is described in [4]. It is clear that CMM takes a top-down managerial view which is at odds with knowledge creation as we have seen in Section 1.

What we need for a better governance of software assets is knowledge about the software development process, about operations and about maintenance. It goes without saying that detailed technical knowledge about the software itself as well as about its history (development, operations, testing, maintenance) is essential to achieve this. However, as we have seen in the previous section, there is usually a lot of missing or tacit knowledge about software assets. To make things worse, this knowledge may change rapidly.

In order to explore how we can increase this knowledge we follow the model for managing knowledge creation developed in [10] and summarized in Section 2. We specialize the model here for creating knowledge about software assets. The creation of knowledge about software assets should be part of the overall knowledge creation strategy of an organization.

Instill a Vision. In many organizations there is only a limited awareness of the crucial role that software assets play to achieve the organization's goals. As we have seen, much of this knowledge is tacit. A *software asset knowledge vision* should therefore succinctly describe the relevance of creating knowledge about software assets for the organization. It should generate awareness for knowledge creation at all levels and it should identify sharing knowledge about software assets as a company value. It should lead to trust, care and cooperation rather than suspicion, indifference and internal competition. Typically, the knowledge vision should stress that

- Software assets are crucial for achieving the organization's goals.

- Creating knowledge about software assets is essential to
  - monitor their business value;
  - enable their continuous evolution;
  - optimize their quality and performance.
- Knowledge about software assets should be made explicit and measurable.
- Everybody wins in the long run by sharing knowledge about software assets. The individual wins, since its expertise becomes valued company-wide rather than only locally. The department and business unit win, since sharing knowledge prevents re-inventing the wheel and potentially improves operational performance.

Software assets are essential production factors for virtually all businesses and company values should strongly encourage creating and sharing knowledge about them. This should also be made clear by making managers at the highest level in the organization responsible for software assets.

Manage Conversations. In the case of software assets there are *several* sources of tacit knowledge that can be tapped:

- The tacit knowledge of system architects, system designers, development programmers, testers, operators and maintenance programmers. This is mostly technical and operational knowledge about software systems.
- The tacit knowledge of experts from marketing, customer relations, and other business departments. This is knowledge how well the software assets behave from a business perspective.
- The source code itself. This includes the text of all programs, test code, test data, database schema's (meta-data), data (database contents), job control scripts, in-house developed tools, compilation and testing scripts as well as history information about revisions, testing, operations, and maintenance. The source code itself is explicit, but the understanding of it is tacit.

In many organizations there are impenetrable walls between the various categories of professionals mentioned in the above summary. Usually, software people don't communicate with business people. But the same is true for development programmers and operators, development programmers and maintenance programmers, or system architects and operators. It is, for instance, not uncommon that development programmers and maintenance programmers only share code but no other knowledge. They may even use different tools suites.

Regarding the source code itself, organizations largely differ in how well they manage the knowledge about their software. In CMM level 1 organizations everything is done in an *ad hoc* fashion: they use no version management,

CMM level	Meaning	Frequency of Occurrence (%)
1 = Initial	Chaotic	75.0
2 = Repeatable	Marginal	15.0
3 = Defined	Adequate	8.0
4 = Managed	Good to Excellent	1.5
5 = Optimizing	State of the art	0.5

Table 3: Distribution of organizations over CMM levels.

configuration control, build management, bug tracking, or test management. In this case, most knowledge is completely implicit. In a CMM level 5 organization, all these aspects are taken care of and in addition detailed performance data is being collected about costs of software development and maintenance and operations.

The amount of implicit knowledge about software assets is staggering: about 75% of all the organizations are still at CMM level 1 [9, p. 30]. Only in a few cases, the explicit knowledge is adequate, see Table 3 for a recent distribution of organizations over CMM levels.

For organizations at levels 1 and 2 a useful knowledge creation scenario is as follows:

- Use automatic tools to extract knowledge from the source code.
- Confront human experts with these results and extend the automatically generated knowledge with expert opinions.
- Create informal groups consisting of software professionals from different disciplines to discuss and extend this knowledge.
- Create informal groups consisting of software professionals as well as professionals from business units to discuss and further extend this knowledge.
- Introduce techniques such as version management, configuration control, build management, bug tracking, or test management.
- Start collecting performance data.

Once the above steps have been implemented, another problem can be addressed: it is very common that decisions about software assets are made by the wrong people at the wrong organizational level. Once managerial support for software asset management exists and the knowledge creation process has been initiated by the above steps, well-informed decisions are enabled at the corporate level.

Mobilize Activists. Knowledge activists are important for all knowledge creation enablers. For managing conversations, they act as catalysts by starting new initiatives and by coordinating and facilitating meetings and workshops. They are also important for creating the right context: finding a compromise between bottom-up knowledge creation versus top-down knowledge management. Finally, they promote globalizing local knowledge by acting as merchant that attracts attention to local initiatives. In the area of software assets, an activist is facing many challenges:

- Disbelief that automatic analysis tools can extract useful information from the source code.
- An even bigger disbelief that automatic tools can transform and improve the source code.
- Resistance to adopt new practices, for instance, for collecting history information.
- Resistance to adopt new tools, for instance, for version management, testing or measuring.
- Reluctance to share locally developed practices or tools with other departments.
- Reluctance to give up the power of expertise (Section 1).
- Disbelief that "the guys from the other department" have something useful to say about the software you are working on.
- Lack of interest in considering the merits of practices and tools used in other departments.

To address these issues, knowledge activists can be appointed according to various strategies. *Process activists* can focus on the overall software development process and its improvement. Typical actions are initiatives to share tacit knowledge, "selling" best practices and appropriate tools to the various departments [2], and initiating measuring and improvement processes. *Architecture activists* aim at collecting knowledge about the global architecture in relation to all applications and using this knowledge for architectural improvement. *Application activists* aim at creating all relevant technical and business knowledge regarding one application and using that for further improvement.

Create Right Context. As already mentioned in Section 2.2, the paradox of knowledge creation is how to reconcile top-down managerial control with bottom-up knowledge creation. We have already seen that there is a strong separation between the various departments involved in software assets, ranging from architecture, development, operations, maintenance, marketing and other business units. We have also seen that this separation is very detrimental to knowledge sharing.

There are various approaches to this problem. A first, lightweight, approach is to create task forces for *process improvement*, *architecture improvement* and *application improvement*, one for each application. A second, heavyweight, approach, is to create cross-divisional units with similar charters. A third approach is to create empowered divisions that are responsible for all aspects of a part of the software assets. Typically, one division per application and a central division for architecture.

Globalize Local Knowledge. Relevant knowledge created locally, has the biggest impact if it used globally. Recall from Section 2.2 that this knowledge transfer can be seen as a three-staged process consisting of *triggering*, *packaging and dispatching*, and *re-creating* knowledge. For software assets the following triggers can be identified:

- A changed business strategy or new commercial opportunities impose new requirements on the existing software assets.
- Costs for development, operations or maintenance exceed the averages in industry benchmarks.
- Long development times result in a too long time-to-market to profit from new commercial opportunities.
- Reliability or performance problems during operations frustrate the business strategy.
- New technological standards or developments require changes to the software assets.
- The vendor-support for a certain tool stops.

For each trigger, the need arises to find or create relevant knowledge. Packaging and dispatching this knowledge implies the following:

- Package in-house developed tools as well as documentation and course material. Install the tools at the appropriate site.
- For software asset knowledge: package the knowledge in appropriate form (e.g., HTML pages, content-management system, database).

In order to re-create the knowledge at the receiving side the following steps are relevant:

- Give tailored courses about tools. Use feedback to adjust course material and tools.
- Create an interdisciplinary working group that applies the software asset knowledge to a particular problem. This may lead to solutions of the problem as well as to the identification of omissions in the software asset knowledge.

5 Conclusions

As we have shown in this paper, the abstract concepts from the field of knowledge management can easily be instantiated for software asset management. The main results from this study are:

- An attempt to construct an inventory of explicit and tacit knowledge about software assets (Section 3).
- An explicit strategy for increasing the knowledge about software assets in an organization (Section 4).

These insights are partly theoretical and they are partly based on our experience in software maintenance and renovation. For instance, the Dutch bank ABN AMRO has taken several steps described in Section 4. As part of their "software logistics" programme they have implemented fully automatic documentation generation (using DocGen [5, 7]) for all their circa 50 Million lines of Cobol code [1]. As a result, the knowledge buried in their software becomes explicitly available and can be accessed with an ordinary web browser. Standard search engines can be used to further query this knowledge. Software is clearly used as a source of knowledge. Other companies are following this example.

However, to further validate our insights and increase their applicability we foresee the following steps:

- Application and qualitative validation of our knowledge creation strategy in other organizations.
- Design of metrics that can be used to measure the impact of knowledge creation strategies for software asset management.
- Quantitative case studies.

Software maintenance and renovation can easily be seen as knowledge creation processes where tacit knowledge is made explicit. The more encompassing view is to consider software and all the tacit knowledge it embodies as a company asset and to properly manage that asset. The knowledge creation perspective described in this paper may help to understand how proper software asset management can be achieved.

References

[1] J. Boef, A. van Deursen, and P. Klint. Goede softwarelogistiek basis voor snelle aanpassingen (in Dutch: Good software logistics basis for fast adjustments). *Automatisering Gids*, page 19, September 7 2001.

[2] M.T. Bosworth. *Solution Selling – Creating Buyers in Difficult Selling Markets.* McGraw-Hill, 1994.

[3] S.L. Brown and K.M. Eisenhardt. *Competing on the Edge – Strategy as Structured Chaos*. Harvard Business School Press, 1998.

[4] B. Curtis, W.E. Hefley, and S. Miller. Overview of the People Capability Maturity Model. Technical Report CMU/SEI-95-MM-002, Software Engineering Institute, 1995.

[5] A. van Deursen and T. Kuipers. Building documentation generators. In H. Yang and L. White, editors, *Proceedings of the International Conference on Software Maintenance*, pages 40–49. IEEE Computer Society Press, 1999.

[6] M. Granovetter. *Getting a Job – A Study of Contacts and Careers*. University of Chicago Press, 2nd edition, 1995.

[7] Software Improvement Group. Automatic Documentation Generation. Software Improvement Group, May 2001. URL: `http://www.software-improvers.com/PDF/DocGenWhitePaper.pdf`.

[8] V.K. Jolly. *Commercializing New Technologies*. Harvard Business School Press, 1997.

[9] C. Jones. *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley, 2000.

[10] G. Von Kroch, K. Ichijo, and I Nonaka. *Enabling Knowledge Creation*. Oxford University Press, 2000.

[11] L. Lamport. How to Tell a Program from an Automobile. In J. Tromp, editor, *A Dynamic and Quick Intellect – Liber Amicorum in honor of Paul Vitanyi's 25-year jubilee*, pages 77–79. CWI, 1996. URL: `http://research.microsoft.com/users/lamport/pubs/automobile.pdf`.

[12] C. Locke, R. Levine, D. Searls, and D. Weinberger. *The Cluetrain Manifesto – The End of Business as Usual*. Perseus Books, 2001.

[13] M. Mulder. *The Daily Power Game*, volume 6. Martinus Nijhoff Social Sciences Division, Leiden, 1977. International series on the quality of working life.

[14] F. Niessink and H. van Vliet. Software maintenance from a service perspective. *Journal of Software Maintenance: Research and Practice*, 12(2):103–120, March/April 2000.

[15] I. Nonaka and H. Takeuchi. *The Knowledge-Creating Company*. Oxford University Press, 1995.

[16] General Accounting Office. DLA Should Strengthen Business Systems Modernization Architecture and Investment Activities, 2001. URL: `http://www.gao.gov/new.items/d01631.pdf`.

[17] M.C. Paulk, C.V. Weber, B. Curtis, and M.B. Chrissis. *The Capability Maturity Model: Guidelines for Improving the Software Process.* Addison-Wesley Publishing Company, Reading, MA, 1995.

[18] C. Perrow. *Normal Accidents – Living with High Risk Technologies.* Princeton University Press, 1984.

[19] M.E. Porter. *Competitive Strategy – Techniques for Analyzing Industries and Competitors.* The Free Press, New York, 1980.

[20] M.E. Porter. *Competitive Advantage – Creating and Sustaining Superior Performance.* The Free Press, New York, 1985.

[21] E.M. Rogers. *Diffusion of Innovations.* The Free Press, Simon & Schuster Inc., 1995. Fourth Edition.

[22] G.M. Weinberg. *Quality Software Management: Volume 1 Systems Thinking.* Dorset House, 1992.

[23] G.M. Weinberg and E.L. Schulman. Goals and performance in computer programming. *Human Factors*, 16(1):70–77, 1974.

About the authors

Paul Klint is head of the software engineering department at Centrum voor Wiskunde en Informatica (CWI, the Dutch national research center for computer science and mathematics) and professor in computer science at the University of Amsterdam. He is also president of the European Association for Programming Languages and Systems (EAPLS) and co-founder of the Software Improvement Group (SIG), a CWI spinoff company. He holds a MSc in Mathematics from the University of Amsterdam and a PhD in Computer Science from the Technical University Eindhoven. He (co)authored three books and has published over hundred scientific articles. He has consulted for companies and governments worldwide. His research interests include generic language technology, domain-specific languages, software renovation, and technology transfer. Contact him at Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands; `Paul.Klint@cwi.nl`; `www.cwi.nl/~paulk`.

Chris Verhoef is a computer science professor at the Free University of Amsterdam and principal external scientific advisor of the Deutsche Bank AG, New York. He is also affiliated with Carnegie Mellon University's Software Engineering Institute and has consulted for hardware companies, telecommunications companies, financial enterprises, software renovation companies, and large service providers. He is an elected Executive Board member and vice chair of conferences of the IEEE Computer Society Technical Council on Software Engineering and a distinguished speaker of the IEEE Computer Society. Contact him at the Free University of Amsterdam , Department of Mathematics and Computer Science, De Boelelaan 1081-A, 1081 HV Amsterdam, Netherlands; `x@cs.vu.nl`; `www.cs.vu.nl/~x`.