Knowledge Management in Software Engineering

Ioana Rus and Mikael Lindvall, *Fraunhofer Center for Experimental Software Engineering, Maryland*

Knowledge management is an expanding and promising discipline that has drawn considerable attention. Recent developments in computer and IT technologies enable sharing documented knowledge independently of time and location. The guest editors provide an overview of these developments to introduce the issue.

Software organizations' main assets are not plants, buildings, or expensive machines. A software organization's main asset is its intellectual capital, as it is in sectors such as consulting, law, investment banking, and advertising. The major problem with intellectual capital is that it has legs and walks home every day. At the same rate experience walks out the door, inexperience walks in the door. Whether or not many software organizations admit it, they face the challenge of sustaining the level of competence needed to win contracts and fulfill undertakings.

Knowledge management is an emerging discipline that promises to capitalize on organizations' intellectual capital. The concept of taming knowledge and putting it to work is not new; phrases containing the word *knowledge*, such as *knowledge bases* and *knowledge engineering*, existed before KM became popularized. The artificial intelligence community has long dealt with knowledge representation, storage, and application. Software engineers have engaged in KM-related activities aimed at learning, capturing, and reusing experience, even though they were not using the phrase "knowledge management." KM is unique because it focuses on the individual as an expert and as the bearer of important knowledge that he or she can systematically share with an organization. KM supports not only the know-how of a company, but also the know-where, know-who, know-what, know-when, and know-why.

The KM concept emerged in the mid-1980s from the need to derive knowledge from the "deluge of information" and was mainly used as a "business world" term. In the 1990s, many industries adopted the term KM in connection with commercial computer technologies, facilitated by development in areas such as the Internet, group support systems, search engines, portals, data and knowledge warehouses, and the application of statistical analysis and AI techniques.

KM implementation and use has rapidly increased since the 1990s: 80 percent of the largest global corporations now have KM projects.[1] Over 40 percent of Fortune 1000 companies now have a chief knowledge officer, a senior-level executive who creates an infrastructure and cultural environment for knowledge sharing.[2]

0740-7459/02/$17.00 © 2002 IEEE

But will KM really help organizations address the problems they face while trying to achieve their business objectives? Or is it just one of those hyped concepts that rise quickly, ambitiously claim to cure organizational headaches, and then fail and fall quietly? KM has drawn widespread attention and has promising features, but it is too soon to say whether or not it will stay.

However, we can address other questions: What kind of problems can KM help solve for software organizations? What are its challenges? What are the success factors? How can KM leverage all the knowledge that exists in software organizations? This special issue of *IEEE Software* addresses these types of questions.

Motivation for KM in software engineering

Software development is a quickly changing, knowledge-intensive business involving many people working in different phases and activities. The available resources are not increasing along with the increasing needs; therefore, software organizations expect a rise in productivity. Knowledge in software engineering is diverse and its proportions immense and steadily growing. Organizations have problems identifying the content, location, and use of the knowledge. An improved use of this knowledge is the basic motivation and driver for KM in software engineering and deserves deeper analysis.

Business needs

Organizations can apply KM to provide solutions to pressing business issues.

Decreasing time and cost and increasing quality. Organizations constantly need to decrease software projects' development time and costs. Avoiding mistakes reduces rework; repeating successful processes increases productivity and the likelihood of further success. So, organizations need to apply process knowledge gained in previous projects to future projects. Unfortunately, the reality is that development teams do not benefit from existing experience and they repeat mistakes even though some individuals in the organization know how to avoid them. Project team members acquire valuable individual experience with each project—the organization and individuals could gain much more if they could share this knowledge.

Making better decisions. In software development, every person involved constantly makes technical or managerial decisions. Most of the time, team members make decisions based on personal knowledge and experience or knowledge gained using informal contacts. This is feasible in small organizations, but as organizations grow and handle a larger volume of information, this process becomes inefficient. Large organizations cannot rely on informal sharing of employees' personal knowledge. Individual knowledge must be shared and leveraged at project and organization levels. Organizations need to define processes for sharing knowledge so that employees throughout the organization can make correct decisions.

Knowledge needs

Software organizations have vast amounts of knowledge in various areas, such as knowledge that is critical to achieve business goals. We now look at some of these knowledge areas and organizations' related needs.

Acquiring knowledge about new technologies. The emergence of new technologies makes software development more efficient, but at the same time, they can be a project manager's worst nightmare. It is difficult for developers to become proficient with a new technology and managers to understand its impact and estimate a project's cost when using it. When developers or project managers use a technology that a project's team members are unfamiliar with, engineers often resort to the "learning by doing" approach, which can result in serious delays. So, organizations must quickly acquire knowledge about new technologies and master them.

Accessing domain knowledge. Software development requires access to knowledge not only about its domain and new technologies but also about the domain for which software is being developed. An organization must acquire new domain knowledge either by training or by hiring knowledgeable employees and spreading it throughout the team.

Is KM just one of those hyped concepts that rise quickly, ambitiously claim to cure organizational headaches, and then fail and fall quietly?

Passing knowledge informally is an important aspect of a knowledge-sharing culture that should be encouraged.

Sharing knowledge about local policies and practices. Every organization has its own policies, practices, and culture, which are not only technical but also managerial and administrative. New developers in an organization need knowledge about the existing software base and local programming conventions. Unfortunately, such knowledge typically exists as organizational folklore. Experienced developers often disseminate it to inexperienced developers through ad hoc informal meetings; consequently, not everyone has access to the knowledge they need. Passing knowledge informally is an important aspect of a knowledge-sharing culture that should be encouraged. Nonetheless, formal knowledge capturing and sharing ensures that all employees can access it. So, organizations must formalize knowledge sharing while continuing informal knowledge sharing.

Capturing knowledge and knowing who knows what. Software organizations depend heavily on knowledgeable employees because they are key to the project's success. However, accessing these people can be difficult. Software developers apply just as much effort and attention determining whom to contact in an organization as they do getting the job done.[3] These knowledgeable people are also very mobile. When a person with critical knowledge suddenly leaves an organization, it creates severe knowledge gaps—but probably no one in the organization is even aware of what knowledge they lost. Knowing what employees know is necessary for organizations to create a strategy for preventing valuable knowledge from disappearing. Knowing who has what knowledge is also a requirement for efficiently staffing projects, identifying training needs, and matching employees with training offers.

Collaborating and sharing knowledge. Software development is a group activity. Group members are often geographically scattered and work in different time zones. Nonetheless, they must communicate, collaborate, and coordinate. Communication in software engineering is often related to knowledge transfer. Collaboration is related to mutual sharing of knowledge. Group members can coordinate independently of time and space if they can easily access their work artifacts. So, group members need a way to collaborate and share knowledge independently of time and space.

KM background

Before we go further into how KM can address software engineering needs, let us introduce some key KM concepts.

Knowledge levels

The three levels of refinement to knowledge items are *data*, *information*, and *knowledge*. Data consists of discrete, objective facts about events but nothing about its own importance or relevance; it is raw material for creating information. Information is data that is organized to make it useful for end users who perform tasks and make decisions. Knowledge is broader than data and information and requires understanding of information. It is not only contained in information, but also in the relationships among information items, their classification, and metadata (information about information, such as who has created the information). *Experience* is applied knowledge.

Knowledge characteristics

KM deals not only with *explicit knowledge*, which is generally easier to handle, but also *tacit knowledge*. Explicit knowledge, also known as *codified knowledge*, is expressed knowledge. It corresponds to the information and skills that employees can easily communicate and document, such as processes, templates, and data. Tacit knowledge is personal knowledge that employees gain through experience; this can be hard to express and is largely influenced by their beliefs, perspectives, and values. The knowledge *scope* refers to where the knowledge is applicable, to whom it is accessible, and whose activity it supports. Along this dimension, knowledge can be at an individual, group, organization, multiple organization, or industry-wide level.

The *knowledge evolution cycle* defines the phases of organizational knowledge.[4] Organizations should have a knowledge management strategy in place for implementing these phases systematically. The phases are

1. *Originate/create knowledge*. Members of an organization develop knowledge through learning, problem solving, innovation, creativity, and importation from outside sources.

2. *Capture/acquire knowledge.* Members acquire and capture information about knowledge in explicit forms.
3. *Transform/organize knowledge.* Organizations organize, transform, or include knowledge in written material and knowledge bases.
4. *Deploy/access knowledge.* Organizations distribute knowledge through education, training programs, automated knowledge-based systems, or expert networks.
5. *Apply knowledge.* The organization's ultimate goal is applying the knowledge—this is the most important part of the life cycle. KM aims to make knowledge available whenever it is needed.

Knowledge management

An organization's intellectual capital consists of *tangible* and *intangible* assets. Tangible assets, which correspond to documented, explicit knowledge, can vary for different industries and applications but usually include manuals; directories; correspondence with (and information about) clients, vendors, and subcontractors; competitor intelligence; patents; licenses; and knowledge derived from work processes (such as proposals and project artifacts). Intangible assets, which correspond to tacit and undocumented explicit knowledge, consist of skills, experience, and knowledge of an organization's people.

Learning and KM

Learning is a fundamental part of KM because employees must internalize (learn) shared knowledge before they can use it to perform specific tasks. Individuals primarily learn from each other, by doing, and through self study. Knowledge also spreads from individuals to groups, and throughout organizations and industries. KM aims to elevate individual knowledge to the organizational level by capturing and sharing individual knowledge and turning it into knowledge the organization can access. Individuals eventually perform tasks to achieve organizational-level goals. Therefore, the iterative knowledge processing and learning activities at the individual level are of utmost importance. As Peter M. Senge says, "Organizations learn only through individuals who learn. Individual learning does not guarantee organizational learning. But without it no organizational learning occurs."[5]

KM's role in software engineering

In software development, different approaches have been proposed to reduce project costs, shorten schedules, and increase quality. These approaches address factors such as process improvement, introducing new technologies, and improving people's performance ("peopleware"). KM addresses mainly peopleware. Because software development is a human and knowledge-intensive creative activity, KM acknowledges the importance of individuals having access to the correct information and knowledge when they need to complete a task or make a decision.

KM does not ignore the value or need to address other software development aspects, such as process and technology, nor does it seek to replace them. Instead, it works toward software process improvement by explicitly and systematically addressing the management of organizational knowledge, such as its acquisition, storage, organization, evolution, and effective access. Other software process improvement approaches, such as the Capability Maturity Model, might suggest that knowledge should be managed but do not explicitly state what knowledge needs to be managed and how, when, where, or by and for whom. KM ties together daily production activities, improvement initiatives, and business goals, thereby supporting the establishment of a learning organization.

Organizations can view KM as a risk prevention and mitigation strategy, because it explicitly addresses risks that are too often ignored, such as

- Loss of knowledge due to attrition
- Lack of knowledge and an overly long time to acquire it due to steep learning curves
- People repeating mistakes and performing rework because they forgot what they learned from previous projects
- Individuals who own key knowledge becoming unavailable

Software engineering involves several knowledge types—technical, managerial, domain, corporate, product, and project knowledge. Knowledge can be transferred through formal training or through learning by doing. Formal training is often time con-

Organizations can view KM as a risk prevention and mitigation strategy, because it explicitly addresses risks that are too often ignored.

suming and expensive, and if done externally does not cover local knowledge. Learning by doing can be risky because people continue to make mistakes until they get it right. KM does not replace organized training, but supports it. Documented knowledge can provide the basis for internal training courses based on knowledge packaged as training material. However, KM mainly supports learning by doing. It provides knowledge or pointers to people who have the knowledge, when and where it is needed.

KM does not come for free; it requires effort and resources. In KM systems that organizations have implemented so far (see the Experience Factory sidebar and this issue's feature articles), people other than developers often perform KM activities (such as a chief knowledge officer and his staff, an Experience Factory group, or a software process improvement group). This supports developers in their daily work instead of loading them with extra effort.

Software engineering's core task is developing software. Documents (such as contracts, project plans, and requirements and design specifications) are also produced during software development. These documents capture knowledge that emerged from solving the project's problems. Team members can then reuse this knowledge for subsequent projects, for example, by analyzing accepted solutions to different problems. If individuals own knowledge that is not explicitly captured, the organization can leverage that knowledge only if it can identify and access these individuals.

Organizations wishing to improve a team's software engineering capabilities can conduct the task of ensuring that knowledge gained during the project is not lost. They

The Experience Factory Organization

Victor R. Basili
Carolyn Seaman

The basis for the Experience Factory Organization[1] concept is that software development projects can improve their performance (in terms of cost, quality, and schedule) by leveraging experience from previous projects. The concept also takes into account the reality that managing this experience is not trivial and cannot be left to individual projects. With deadlines, high expectations for quality and productivity, and challenging technical issues, most development projects cannot devote the necessary resources to making their experience available for reuse.

The EFO solves this problem by separating these responsibilities into two distinct organizations. We call these organizations the Project Organization, which uses packaged experience to deliver software products, and the Experience Factory, which supports software development by providing tailored experience. Figure A depicts the EFO, which assumes separate logical or physical organizations with different priorities, work processes, and expertise requirements.

The Experience Factory analyzes and synthesizes all experience types, including lessons learned, project data, and technology reports, and provides repository services for this experience. The Experience Factory employs several methods to package the experience, including designing measures of various software process and product characteristics and then building models of these characteristics that describe their behavior in different contexts. These models' data come from development projects via people, documents, and automated support.

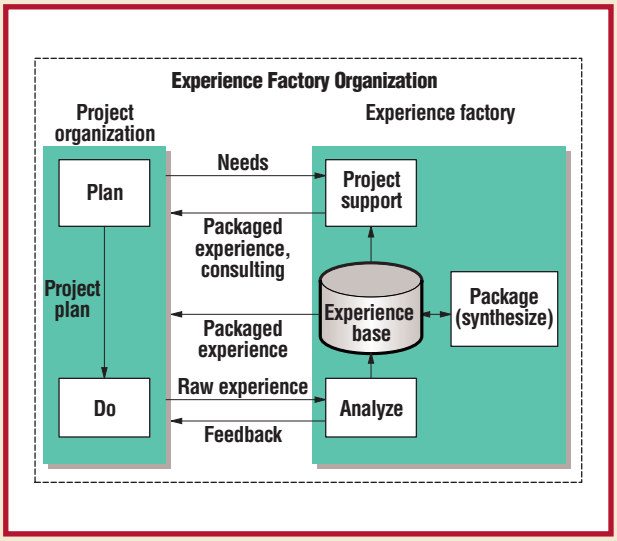

Figure A. Flow of information through the Experience Factory Organization.

When using EFO, not only must the organization add another suborganization for learning, packaging, and storing experience, but it also must change the way it does its work. An organization adopting the EFO approach must believe that exploiting prior experience is the best way to solve problems and ensure that the development process incorporates seeking and using this experience. The EFO also as-

can conduct this task during the project and shortly after they complete it. It addresses both acquiring knowledge that was not documented as part of the core activities and analyzing documents to create new knowledge. Included in this task are all forms of lessons learned and postmortem analyses that identify what went right or wrong regarding both the software product and process.

These activities also include project data analyses, such as comparisons of estimated and actual costs and effort, planned and actual calendar time, or analysis of change history to reflect project events. These tasks collect and create knowledge about a particular project; any organization can perform them. Although these activities' results are useful by themselves, they can also be the basis for further knowledge creation and learning. They can be stored in repositories and experience bases.

At a higher level, organizations and industries must analyze multiple past projects to improve their software developing abilities. This requires extensive knowledge based on many different software development experiences, as well as insights into analysis and synthesis of new knowledge. Patterns, heuristics, best practices, estimation models, and industry-wide standards and recommendations are examples of outcomes from these knowledge-processing activities.

We group KM activities that support software development into three categories: by the purpose of their outputs (supporting core SE activities, project improvement, or organizational improvement), the scope of their inputs (documents or data from one or multiple projects), and the effort level required to process inputs to serve SE needs. We use this classification to describe how both existing and new

sumes that the activities of the Experience Factory and those of the Project Organization are integrated. That is, the activities by which the Experience Factory extracts experience and then provides it to projects are well integrated into the activities by which the Project Organization performs its function. Figure A represents this interaction and exchange of experience.

Making experience available and usable is crucial but is not the essence of an EFO. "Experience" in an Experience Factory is not only the raw information reported directly from projects. It also includes the valuable results of the analysis and synthesis of that local experience, such as "new" knowledge generated from experience. But the new knowledge is based on applying previous experience on real projects, not on analysis in a vacuum.

Thus, an EFO must

- Package experience by analyzing, synthesizing, and evaluating raw experience and build models that represent abstractions of that experience
- Maintain an experience base or repository of data, experience, models, and other forms of knowledge and experience
- Support projects in identifying and using the appropriate experiences for the situation

Victor Basili first presented the EFO concept in a keynote address at COMPSAC in 1989.[2] This was before the term "knowledge management" became popular, but the EFO addresses many of the same concerns. This learning-organization concept evolved from our experiences in the NASA Software Engineering Laboratory, a joint effort of the University of Maryland, Computer Sciences Corporation, and the NASA Goddard Space Flight Center. The SEL's high-level goal was to improve Goddard's software processes and products.

The application of EFO ideas resulted in a continuous improvement in software quality and cost reduction during the SEL's quarter-century lifespan.[3] Measured over three baseline periods in 1987, 1991, and 1995 (each baseline was calculated based on about three years' worth of data), demonstrated improvements included decreases in development defect rates of 75 percent between the 1987 and 1991 baselines and 37 percent between the 1991 and 1995 baselines. We also observed reduced development costs between subsequent baselines of 55 percent and 42 percent, respectively.

References

1. V.R. Basili and G. Caldiera, "Improve Software Quality by Reusing Knowledge and Experience," *Sloan Management Rev.*, vol. 37, no. 1, Fall 1995, pp. 55–64.
2. V.R. Basili, "Software Development: A Paradigm for the Future," *Proc. 13th Int'l Computer Software and Applications Conf.* (COMPSAC 89) IEEE CS Press, Los Alamitos, Calif., 1989, pp. 471–485.
3. V.R. Basili et al., "Special Report: SEL's Software Process-Improvement Program," *IEEE Software*, vol. 12, no. 6, Nov./Dec. 1995, pp. 83–87.

Victor R. Basili's biography appears on page 49.

Carolyn Seaman is an assistant professor of Information Systems at the University of Maryland, Baltimore County and a research scientist at the Fraunhofer Center for Experimental Software Engineering, Maryland. Contact her at cseaman@umbc.edu.

A common problem that knowledge management addresses is expert identification.

software engineering processes and tools can fit into a KM strategy. In the remainder of this section, we also discuss resources external to the organization that are built around the notion of capturing and sharing knowledge.

Supporting core SE activities

Document management, *competence management*, and *software reuse* are knowledge management activities that support software development.

Document management. A software development project involves a variety of document-driven processes and activities. The work frequently focuses on authoring, reviewing, editing, and using these documents, which become the organization's assets in capturing explicit knowledge. Therefore, document management is a basic activity toward supporting an organization's implementation of a knowledge management system. DM systems enable employees throughout the organization to share documented knowledge. Many commercial tools support DM, such as Hyperwave, Microsoft Sharepoint, Lotus Domino, and Xerox DocuShare (see "The Experience Factory Organization" sidebar), and include features such as defining process workflows and finding experts.

Competence management and expert identification. Far from all of an organization's tacit knowledge can be made explicit, and far from all explicit knowledge can be documented. So, an organization must track who knows what to fully utilize undocumented knowledge. An elaborate solution to this problem is competence management, or *skills management*. As we noted earlier, a common problem that knowledge management addresses is expert identification. Competence management systems, such as SkillScape and SkillView, include tools that let experts generate and edit their own profiles. Other tools, such as KnowledgeMail, automatically generate competence profiles by assuming that peoples' emails and documents reflect their expertise. These tools analyze email repositories and documents and build keyword-based profiles that characterize each employee. Software organizations can use them to identify experts in various technical areas, such as specific programming languages, database technologies, or operating systems.

Software reuse. There are endless stories about how programmers continuously implement the same solutions in slightly different ways. Software reuse approaches attempt to reduce this rework by establishing a reuse repository. Programmers submit software they believe would be useful to others. The software development process must change so that instead of developing all software from scratch, a programmer first searches the repository for reusable parts. Only if the programmer found nothing useful would he or she write the software from scratch. This same concept can apply to all software engineering artifacts.

Supporting product and project memory

Learning from experience requires a product and project memory. The environment in which software engineers conduct their daily work often supports creating such a memory. Version control, change management, documenting design decisions, and requirements traceability are software engineering practices that help build such memories as a direct or side effect of software development.

Version control systems, such as the Source Code Control System, represent a class of tools that indirectly create a project memory. Each version of a document has a record attached with information about who made the change, as well as when and why they made it. This "memory" indicates the software's evolution. Software engineers use this information for advanced analysis of software products and processes.[6] They can also use the who-changed-it information to identify experts. Design Rationale is an example of an approach that explicitly captures software design decisions to create a product memory and help avoid repeating mistakes.[7] This is important because during design, engineers test different technical solutions and make decisions on the basis of these test results. Unfortunately, these decisions are rarely captured, making it difficult for anyone else to understand the reasons behind the solutions.

Software requirements drive the development of software systems, but the connection between the final system and its requirements is fuzzy. *Traceability* is an approach that explicitly connects the requirements and the final software system. It indirectly contributes to the product memory and helps answer

questions such as "What requirements led to a particular piece of source code?" and "What code did the engineers develop to satisfy this particular requirement?"

Supporting learning and improvement

Project managers need to make a series of decisions at the beginning of and during projects. Typically, their personal experience and "gut feeling" guide their decisions. But because software development is such a complex and diverse process, gut feelings might not be sufficient, and not all managers have extensive experience. For these reasons, *predictive models* can guide decision making for future projects based on past projects. This requires having a metrics program in place, collecting project data with a well-defined goal in a metrics repository, and then analyzing and processing the data to generate predictive models. Managers or process improvement personnel analyze, synthesize, and process input data using different methods, depending on the model's purpose and the input and output type.

Analytical models have as input data from a large number of projects (either numerical data or qualitative data converted into quantitative levels). Using these formulas for the data that characterize a new project, project managers can estimate cost, effort, defects, reliability, and other product and project parameters. Building, using, and improving these models become a natural part of the KM strategy.

Another class of predictive models that captures the development process structure and the internal-process variable relationships consists of executable behavioral *process models* and simulators based on system dynamics[8] and discrete event modeling. Engineers and managers can execute these models, simulate what-if scenarios, and analyze possible outcomes for multiple decisions. These models capture knowledge that addresses process structure and behavior. The prediction quality these models offer depends on the collected data quality.

Project information can also be in a qualitative form (such as cases and lessons learned, success and failure stories, and problems and corresponding solutions) represented in formats such as rules, indexed cases, or semantic networks. Applying induction, generalization, and abstraction on this knowledge can generate new knowledge (manually, or automatically by applying AI techniques) applicable to similar future problems in similar contexts. This is how patterns, best-practice guidelines, handbooks, and standards are derived. The models transform raw point data into explicit and more applicable knowledge.

For example, *case based-systems* capture project experiences to accommodate software development process diversity while retaining a level of discipline and standards. These experiences provide developers with knowledge of an organization's previous development issues. Deviations from the standard process are opportunities to improve the process itself. Deviations also work as cases in the experience base. As organizations acquire more experience in the form of cases, the development process is iterated and becomes even more refined.

Implementing KM

Implementing KM involves many challenges and obstacles. Three issues are particularly important:[1]

- *Technology issues*. Software technology supports KM, but it is not always possi-

Useful URLs

ACM Special Interest Groups: www.acm.org/sigs/guide98.html
CeBASE: www.cebase.org
Carnegie Mellon's Software Process Improvement Network: www.sei.cmu.edu/collaborating/spins
Expert Exchange: www.experts-exchange.com
Fraunhofer Center for Experimental Software Engineering, Maryland: http://fc-md.umd.edu
Hyperwave: www.hyperwave.com
IBM's Lotus: www.lotus.com/home.nsf/welcome/km
KnowledegeMail: www.knowledgemail.com
Microsoft Product Support Services: http://search.support.microsoft.com/kb
Microsoft Sharepoint Technologies: www.microsoft.com/sharepoint
Oracle Support Services: www.oracle.com/support/index.html?content.html
Perl's FAQ: www.perl.com/pub/q/faqs
Skillscape: www.skillscape.com
Skillview: www.skillview.com
Software Engineering Body of Knowledge: www.swebok.org
Software Program Managers Network: www.spmn.com
Sun's Java community: http://developer.java.sun.com/developer/community
ViSEK at Fraunhofer Center for Experimental Software Engineering, Germany: www.iese.fhg.de/Projects/ViSEK
Xerox Docushare: http://docushare.xerox.com

An analysis of KM failures reveals that many organizations who failed did not determine their goals and strategy before implementing KM systems.

ble to integrate all the different subsystems and tools to achieve the planned level of sharing. Security is a requirement that the available technology does not often provide satisfactorily.

- *Organizational issues*. It is a mistake for organizations to focus only on technology and not on methodology. It is easy to fall into the technology trap and devote all resources to technology development, without planning for KM implementation.
- *Individual issues*. Employees often do not have time to input or search for knowledge, do not want to give away their knowledge, and do not want to reuse someone else's knowledge.

An analysis of KM failures reveals that many organizations who failed did not determine their goals and strategy before implementing KM systems. In fact, 50 to 60 percent of KM deployments failed because organizations did not have a good KM deployment methodology or process, if any.[1] Some organizations ended up managing documents instead of meaningful knowledge. This is an easy mistake to make, because many tools advertised as KM tools address document management rather than knowledge management.

Lightweight KM approaches

A problem with KM is that it might take time before measurable benefits appear. It generally takes a long time before knowledge bases contain a critical mass of knowledge. However, organizations can quickly and easily implement lightweight KM approaches—this can pay off quickly while laying the foundation for long-term goals. One such tool is the Knowledge Dust Collector,[9] which supports peer-to-peer knowledge sharing by capturing knowledge that employees exchange and use every day. The knowledge "dust" evolves over time into knowledge "pearls," a refined knowledge form. The Knowledge Dust Collector captures dialogs regarding technical problems (knowledge dust), analyzes them, and turned them into frequently asked questions (knowledge pearls). These FAQs are then analyzed and turned into best practices (extended knowledge pearls).

Organizational culture

Although new technology makes sharing knowledge easier than ever, organizational cultures might not promote it. Some cultures even overly encourage individualism and ban cooperative work. Lack of a "knowledge culture" has been cited as the number one obstacle to successful KM.[4] If organizations don't foster a sharing culture, employees might feel possessive about their knowledge and won't be forthcoming in sharing it. Employees know that the organization values them because of their knowledge; they might fear that they will be considered redundant and disposable as soon as the employer has captured their knowledge. Employees might not be willing to share negative experiences and lessons learned based on failures because of their negative connotation. So although KM's purpose is to avoid similar mistakes, employees might fear that such information could be used against them. Another hurdle is the "not invented here" syndrome—some believe that software engineers are reluctant to reuse other people's solutions. Although change is hard, such beliefs must be revisited and replaced by a positive attitude that engenders and rewards sharing.

KM champions

Hewlett-Packard advocates using an evangelist or champion for any KM initiative—someone who encourages employees to contribute and use the system and who is always its proponent.[10]

Reward systems

Organizations must not only encourage but also reward employees who share their knowledge, search for knowledge, and use others' knowledge. To encourage sharing and reusing knowledge, Xerox recommends creating a "hall of fame" for those people whose contributions have solved real business problems. Xerox rewards staff that regularly share useful information and identifies them as key contributors to the program. Bruce Karney, evangelist of a Hewlett-Packard KM initiative, gave out free Lotus Notes licenses and free airline miles to prospective users.[10] Infosys rewards employee contribution and use of knowledge with "knowledge currency units," which they can convert into cash. The online expertise provider ExpertExchange rewards experts with points for answering questions and recognizes those with the most points on the front page of their Web site.

Leveraging employees' expertise

Most approaches that support experience reuse and knowledge management assume that all relevant experience can be collected and recorded, but in practice, this does not hold true. A variety of KM solutions address different aspects and tasks of this problem.

For example, Ericsson Software Technology implemented a version of the Experience Factory (see the related sidebar) called the Experience Engine.[11] Instead of relying on experience stored in experience bases, the Experience Engine relies on tacit knowledge. Ericsson created two roles to make the tacit knowledge accessible to a larger group of employees. The *experience communicator* is an employee who has in-depth knowledge on one or more topics. The *experience broker* connects the experience communicator with the employee who owns the problem. The communicator should not solve the problem but instead educate the problem's owner on how to solve it. The German organization sd&m has successfully implemented a similar approach.[12] Relying on tacit rather than explicit knowledge is appealing because it relaxes the requirement to extensively document knowledge. However, although this approach uses the knowledge, it still does not solve the problem of an organization being dependent on its employees and their tacit knowledge.

The benefit of explicit knowledge or experience is that it can be stored, organized, and disseminated to a third party without the originator's involvement. The drawback is that organizations spend considerable effort to produce explicit knowledge. Some development practices, such as pair programming, facilitate knowledge sharing between peers, while job rotation helps knowledge spread throughout the project or organization. Software organizations should encourage these habits to create a knowledge-sharing culture. To achieve the maximum benefit from knowledge sharing, organizations should encourage employees to document and store their knowledge in a KM repository whenever they help another employee. In doing so, they ensure that the information they passed on is recorded and will help other employees, because what is a problem for one can also be a problem for many.

The most used form of knowledge sharing probably occurs in communities in which members can come together and share knowledge and experience. Communities are popular because they are relatively easy to form. Software organizations have formed numerous useful communities. Some examples include the Software Program Managers Network; the Software Experience Consortium, for companies seeking to share experience; Sun's community for Java programmers, the Software Process Improvement Network; and special interest groups of the IEEE and ACM.

Leveraging organizational expertise

Organizations strive to learn more and leverage their expertise through input that comes from outside the organization.

Sharing experience with customers. Organizations learn not only from their own experiences but also from external sources, typically technology vendors. Several software vendors provide online knowledge bases, such as Microsoft's Knowledge Base, Oracle's Support Center, and Perl's FAQ. Such knowledge bases are often open to the public and let software engineers search for knowledge. These knowledge bases resulted from capturing vendor representatives' product knowledge and making it available to the vendors' customers.

Industry-wide knowledge sharing and education. At the software industry level, committees or groups of experts identify patterns (such as software design patterns) and generate handbooks and standards (such as those from the IEEE and ISO) that are generally applicable to software development, to leverage the experience and knowledge of all software development organizations. This is not something any individual or organization can perform, because it takes much effort and requires a considerable amount of SE knowledge and access to project data.

An example of the numerous industry-wide knowledge initiatives is the Software Engineering Body of Knowledge (SWEBOK). It defines the knowledge that a practicing software engineer needs to master on a daily basis. Another example is ISO15504, a comprehensive collection of software engineering knowledge that describes processes related to SE.

The benefit of explicit knowledge or experience is that it can be stored, organized, and disseminated to a third party without the originator's involvement.

There are good reasons to believe that KM for software engineering can succeed if organizations appropriately focus and implement it.

Projects whose goal is to build knowledge bases include the Center for Empirically Based Software Engineering and ViSEK (Virtual SE Competence Center). They accumulate empirical models to provide validated guidelines for selecting techniques and models, recommend areas for research, and support SE education.

KM challenges

As we noted earlier, implementing KM is challenging because many resources and much time and effort are required before benefits become visible. Project managers who feel they need to focus on completing their current project on time, and not on helping the next project manager succeed, often consider this a burden.

Another obstacle is that most SE knowledge is not explicit. Organizations have little time to make knowledge explicit. Additionally, there are few approaches and tools that turn tacit into explicit knowledge. Technology's fast pace often discourages software engineers from analyzing the knowledge they gained during the project, believing that sharing the knowledge in the future will not be useful.

Opportunities

Despite the challenges we presented earlier, there are good reasons to believe that KM for software engineering can succeed if organizations appropriately focus and implement it. A major argument for KM is that software organizations should already have much of the appropriate information technology in place to support KM systems. IT might be intimidating to many people, but not to software engineers. Instead, we expect that engineers will benefit even more from advanced technology. Additionally, all artifacts are already in electronic form and thus can easily be distributed and shared. In fact, software engineers already share knowledge to a large degree in some environments. A good example is Google, whose user community of software engineers share knowledge by answering questions and helping solve problems that other software engineers post, without being compensated. This shows that software engineers are willing to share their knowledge, even outside their company, and that capturing gained knowledge is worth the effort even though technology changes quickly.

State of the practice

This special issue of *IEEE Software* investigates KM's state of the practice in software engineering. The large number of submissions we received (over 40 papers presenting SE applications of KM) shows the interest for this topic throughout the software community. The selected articles report on the needs, implementations, issues, results, success factors, and lessons learned from a variety of KM applications.

These articles cover a diverse ground; they come from the US, Europe (Germany, Norway, and Finland), and Asia (India and Taiwan) and from commercial and government organizations developing either software for diverse domains (satellites, cars, electronics, telecommunications, and the Internet) or integrated circuits. The authors are practitioners, consultants, researchers, and KM officers. The articles address different levels and scopes of KM activities, from project-level knowledge (captured by postmortem analysis or by gathering knowledge on demand), to organization-wide initiatives. They examine various techniques through case studies, from local analysis to traceability, to complex and highly automated knowledge and experience repositories.

Jay Liebowitz's "A Look at NASA Goddard Space Flight Center's Knowledge Management Initiatives" describes the creation of a NASA knowledge management team, a KM officer position, and a series of KM initiatives at NASA Goddard Space Flight Center. They learned that KM should start small and see what works in a specific environment and that knowledge should be collected during projects, not after their completion. They also learned to capture individual knowledge through interviews before people leave the organization and to embed KM processes in daily activities.

Andreas Birk, Torgeir Dingsøyr, and Tor Stålhane's "Postmortem: Never Leave a Project without It" addresses the problem that the individual knowledge that engineers gain during projects is not reused or shared between teams. Birk and his colleagues' solution was to use systematic postmortem analysis for capturing and reusing experience and improvement suggestions. Having an open atmosphere for the project team to discuss issues was a key success factor. As a result, teams experienced increased

understanding and experience sharing, identification of potential improvement needs, and increased job satisfaction.

Software development and acquisition competencies are a scarce resource in many companies, and DaimlerChrysler is no exception. To cope with this, DaimlerChrysler researchers set up a Software Experience Center project. SEC explicitly reuses experience from previous software projects using a customized Experience Factory approach (see related sidebar). Kurt Schneider, Jan-Peter von Hunnius, and Victor R. Basili's "Experience in Implementing a Learning Software Organization" presents the most important challenges they faced: learning implies change—and change is not easy. Learning involves risks that must be mitigated, and experience reuse requires packaging, which is not trivial. Success factors included understanding that change takes time and that experience must be packaged and tailored for the organization. Thanks to the SEC project, learning has become part of developers' daily routine, leading to process improvements.

From studying best practices in 30 system development organizations, Balusubramaniam Ramesh observed the need to link knowledge fragments (which were spread across the organization) to facilitate successful knowledge transfer and reuse. In "Process Knowledge Management with Traceability," he proposes creating, storing, retrieving, transferring, and applying knowledge by using traceability (creating and maintaining relationships between objects and people in software development). This is possible where development of facilities that support KM processes (traceability schemes and tools) exist. This approach's benefits are enhanced collaboration and communication, new knowledge creation, knowledge retention, and increased knowledge availability and access for software developers.

In Shivram Ramasubramanian and Gokulakrishnan Jagadeesan's article, "Knowledge Management at Infosys," Infosys, an Indian consultancy and software services company, realized that locating information and knowledge needed for project activities took too long. Senior management therefore launched a company-wide initiative for building and maintaining a KM infrastructure based on central repositories of documents and expertise maps. They introduced an incentive system that converted "knowledge units" into cash to encourage use of and contributions to these repositories. The initiative resulted in reduced defects, increased productivity, decreased cost (mainly by reducing mistakes and rework), and better teamwork.

The problem of retaining, reusing, and transferring knowledge occurs not only in SE but also in other knowledge-intensive domains. Borrowing solutions and experience from these other domains can benefit SE. Chin-Ping Wei, Paul Jen-Hwa Hu, and Hung-Huang Chen's "Design and Evaluation of a Knowledge Management System" reports on KM implementation in an integrated-circuit assembly and testing organization. The company implemented a system that supports expert identification and collaboration for when engineers need to solve a specific problem. Once a problem is solved, its history and solution are stored in a codification-based knowledge repository for later reuse. This system leads to ease of locating experts and sharing knowledge, increases productivity, reduces production interruptions caused by lack of knowledge and search for knowledge, and reduces delayed responses to customer inquiries or complaints.

Seija Komi-Sirviö, Annukka Mäntyniemi, and Veikko Seppänen in "Toward a Practical Solution for Capturing Knowledge for Software Projects," emphasize the need for addressing local needs, problems, and specific context for KM initiative implementation. At an organization that develops software-intensive electronic products, KM activities were not as efficient as they expected. They replaced their current large scale, technology-centered KM approach with a project need-based approach to knowledge collection and delivery. This proved successful, leading to project staff satisfaction.

Recent developments in IT definitely enable sharing documented knowledge independent of time and space.

Not all KM initiatives presented here (maybe none of them) address all aspects of KM. This is normal because KM has not matured yet, especially in SE. When an organization implements such a program, it has to start small. It needs to identify which specific problems and priorities it should address and what does or does not work for that organization. The results must show benefits relatively quickly and convince

both the employees to use the system and the management to support it (by continuous investment in the necessary resources).

It is too early to say whether knowledge management in software engineering will survive, but it is clear that software organizations struggle to retain valuable knowledge and they still depend on knowledge possessed by their employees. From the enthusiastic response to this special issue, reflected both by the large number of high quality articles submitted (many of them presenting implementation of KM in industry) and by the impressive number of volunteering reviewers, we infer that KM is of interest to many software professionals and researchers and is being tested by many companies as a potential solution to these problems.

Recent developments in IT definitely enable sharing documented knowledge independent of time and space. We foresee that in the future there will also be support for capturing and disseminating knowledge in various formats, enabling organizations and individuals to share knowledge on a worldwide scale.

Acknowledgments

We thank Tom McGibbon and David Nicholls from Data Analysis Center for Software, who supported our work when writing two knowledge management state-of-the-art reports on which we based this tutorial.[14,15] We thank Seija Komi-Sirvio, Patricia Costa, Scott Henninger, Raimund Feldman, Forrest Shull, and Rose Pajerski for reviewing; Kathleen Dangle and Tricia Larsen for useful discussions on KM and software process improvement; and Jennifer Dix for proofreading. Many thanks go to the *IEEE Software* editorial board who anticipated the benefit of this special issue, and to the reviewers of the articles who kindly offered their help for improving the quality of the published material.

References

1. G. Lawton, "Knowledge Management: Ready for Prime Time?" *Computer*, vol. 34, no. 2, Feb. 2001, pp. 12–14.
2. D.E. O'Leary, "Enterprise Knowledge Management," *Computer*, vol. 31, no. 3, Mar. 1998, pp. 54–61.
3. D.E. Perry, N. Staudenmayer, and L. Votta, "People, Organizations, and Process Improvement," *IEEE Software*, vol. 11, no. 4, July/Aug. 1994, pp. 36–45.
4. W. Agresti, "Knowledge Management," *Advances in Computers*, vol. 53, Academic Press, London, 2000, pp. 171–283.
5. P.M. Senge, *The Fifth Discipline: The Art and Practice of the Learning Organization*, Currency Doubleday, New York, 1990, p. 139.
6. S.G. Eick et al., "Does Code Decay? Assessing the Evidence from Change Management Data," *Trans. Software Eng.*, vol. 27, no. 1, Jan. 2000, pp. 1–12.
7. C. Potts and G. Bruns, "Recording the Reasons for Design Decisions," *Proc. 10th Int'l Conf. Software Eng.* (ICSE 88), IEEE CS Press, Los Alamitos, Calif., 1988, pp. 418–427.
8. T. Abdel-Hamid and S.E. Madnick, *Software Project Dynamics: An Integrated Approach*, Prentice-Hall, Englewood Cliffs, N.J., 1991.
9. M. Lindvall et al., "An Analysis of Three Experience Bases," *Proc. Workshop on Learning Software Organizations.* (LSO 01), Springer-Verlag, Hiedelberg, Germany, pp. 106–119.
10. T. Davenport, Knowledge Management Case Study, *Knowledge Management at Hewlett-Packard*, 2002, www.bus.utexas.edu/kman/hpcase.htm.
11. C. Johansson and P.C.M. Hall, "Talk to Paula and Peter—They Are Experienced," *Proc. Workshop on Learning Software Organizations* (LSO 99), Springer-Verlag, Hiedelberg, Germany, 1999, pp. 171–185.
12. P. Brössler, "Knowledge Management at a Software Engineering Company—An Experience Report," *Proc. Workshop Learning Software Organizations* (LSO 99), Springer-Verlag, Hiedelberg, Germany, 1999, pp. 163–170.
13. T. Davenport, Knowledge Management Case Study, *Knowledge Management at Microsoft*, 2002, www.bus.utexas.edu/kman/microsoft.htm.
14. M. Lindvall, *Software Tools for Knowledge Management*, tech. report, DoD Data Analysis Center for Software, Rome, N.Y., 2001.
15. I. Rus, M. Lindvall, and S. Sinha, *Knowledge Management in Software Engineering*, tech. report, DoD Data Analysis Center for Software, Rome, N.Y., 2001.

About the Authors

Ioana Rus is a scientist at Fraunhofer Center for Empirical Software Engineering, Maryland. Her research interests include experience and knowledge management, software process modeling and simulation, process improvement, measurement, and empirical studies in software development. She has a PhD in computer science and Engineering and is a member of the IEEE Computer Society and ACM. Contact her at irus@fc-md.umd.edu.

Mikael Lindvall is a scientist at Fraunhofer Center for Experimental Software Engineering, Maryland. He specializes in work on experience and knowledge management in software engineering. He is currently working on ways of building experience bases to attract users to both contribute and use experience bases. He received his PhD in computer science from Linköpings University, Sweden. His PhD work was based on a commercial development project at Ericsson Radio and focused on the evolution of object-oriented systems. Contact him at mlindvall@fc-md.umd.edu.