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.

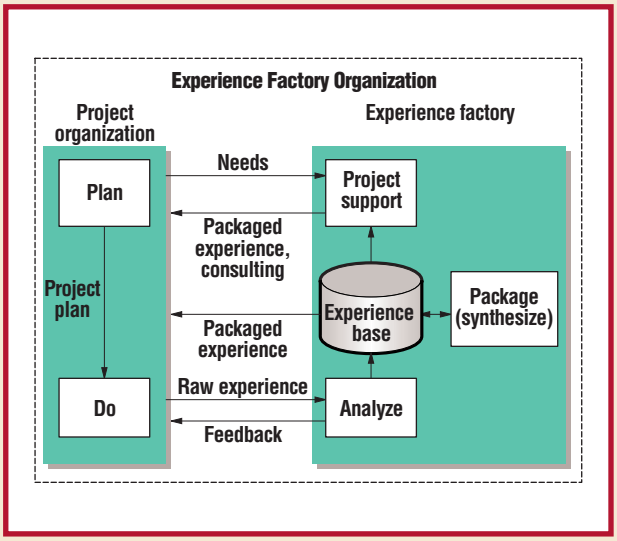

Figure A. Flow of information through the Experience Factory Organization.

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.