Experience in Implementing a Learning Software Organization

Kurt Schneider and Jan-Peter von Hunnius, *DaimlerChrysler Research Center*

Victor R. Basili, *University of Maryland*

DaimlerChrysler created its Software Experience Center to investigate experience reuse and encourage experience-based software process improvement. Here, the authors report on challenges the company faced when creating the SEC.

Competence in software development and acquisition has become essential for the automotive industry. As a manufacturer of premium-class cars, DaimlerChrysler depends on high-quality software for its sophisticated electronic control units. ECUs implement features such as intelligent brake assistants, electronic stability, and engine controls. Unfortunately, software development and acquisition competencies are a scarce resource. Consequently, DaimlerChrysler decided to better use its internal software knowledge in two ways. First, it wanted to improve software development and acquisition processes to increase software quality and repeatability of success. Second, it wanted to explicitly reuse knowledge from previous software projects to enhance future ones. In particular, DaimlerChrysler considered reusing experiences as a key to better project performance and higher software quality.

We can view such experience exploitation as a variant of knowledge management.[1] Unlike factual knowledge, we can't find experience in textbooks. Experiences are related to the environment and context in which they occurred, and when reused in their original context, they can direct software process improvement (SPI). For example, we can calibrate the frequency and intensity of design or code inspections to optimize effort spent and errors detected.

Here, we report on DaimlerChrysler's Software Experience Center project. The SEC aimed to investigate experience reuse (as a variant of knowledge management) and apply insights and collected experiences to SPI.

Experience-based SPI

The SEC's operational goal was to provide business units with the concepts of a learning organization and a prototype of an experience base. To do this, researchers acted as experience and knowledge engineers and coaches for the business units to assist them in their experience exploitation activities and transfer insights into SPIs. DaimlerChrysler expected a learning software organization would better use the scarce resource of available software competency.[2]

The SEC built on the concept of an *experience factory*,[3] which is an organizational unit that supports several software projects. As a

0740-7459/02/$17.00 © 2002 IEEE

separate entity, an experience factory receives plans, status information, and experiences from all participating projects. Incoming data is organized in models, such as defect density models, Pareto charts of defect class baselines, algorithms, and so forth.[4] These models provide projects with immediate feedback—such as "your error density is now 10 percent higher than usual"—and experience-based advice—"when you inspect more than five pages of code at a time, your performance goes down." NASA first implemented this concept.[5] DaimlerChrysler's situation and environment called for some modifications.[6] Its experiences were more qualitative than those of a classical experience factory because it had less quantitative data available. Consequently, it also used process models rather than parametric equations.

DaimlerChrysler's SEC supported all activities, from experience elicitation to making experience available for a software task at hand. Experience elicitation involved interviews, feedback forms, and an optimized version of a lightweight elicitation workshop.[7] To spread consolidated experiences, we used an intranet description of the software process at hand (for example, of an inspection process). Using HTML, we could describe a process and link it to such items as training materials, experiences, checklists, frequently asked questions, and expert emails. We called this collection our *experience base* for any given topic (topics include software inspections, software risk management, and requirements engineering). An experience base is the persistent storage of experiences, organized around a software process.[8] It offers email contact in different contexts and users are encouraged to provide feedback. We had some scripts that could quickly update an experience base, but beyond this, there was no automation. Elicitation and reuse procedures were more critical than tools or automation.

Three major challenges for a learning software organization

Establishing an SEC as a learning organization in the business units participating in this project was surprisingly difficult. We discuss two cultural challenges and one technical challenge that were decisive for turning a department or business unit into a learning software organization at DaimlerChrysler.

Learning implies broad and deep change

Introducing a learning software organization deeply changes how developers and managers should work. Developers must change their mindsets, skills, and behavior; managers must change their expectations about what gets delivered and when; and the organization must reconsider its training approach and any processes that interact with the primary process it is changing. If the effects of change do not occur across the organization, improvements will not happen, and the cost and effort invested will have been wasted.

Introducing a learning software organization deeply changes how developers and managers should work.

Experiences at DaimlerChrysler. We experienced several situations in which management commitment was limited in time and scope. For example, one manager complained about an insufficient effect of change after only two months of SEC operation. In another case, a manager constantly refocused the area for improvement. In both cases, management blocked emerging change by not understanding an important implication of change: it takes time.

On the other hand, when handled with care, change can lead to success. For example, over a three-year period, a quality management support group encouraged a software risk management process by developing pilot projects, discussing the process in user groups, submitting it to management for approval and commitment, offering training in tailored courses, and using a specific, custom-made risk management experience base. The group considered all stakeholders and their concerns, differences, and needs; it anticipated and addressed ripple effects (for example, the need for training).[9] Risk management is now accepted in the entire department and applied to all top strategic software projects as well as to a growing number of other software projects. Risks are now identified earlier and can be mitigated more effectively.

Recommendations. Improvement causes change, and effective change is not local. Because we cannot always control the speed of ripple effects, patience and a long-term vision are indispensable. All stakeholders must understand what is happening, why it is happening, how it affects their jobs, and why it takes so long. Things usually differ from what we first believe, so adaptation and iteration are needed. Spreading best practices in

Apply risk management to your software improvement initiative and learning efforts just as you would to a software development project.

experience bases can support the ripple of change and its speed. Plan activities that help stakeholders adjust to the ripple effects, and advise key people that they must adjust their own behavior—not just provide funding.

Capitalizing on learning involves risk

There is much to gain by making changes based on learning—for example, there are many opportunities in SPI.[2] However, improvement programs and learning initiatives imply that skills must change and acquired expertise might become obsolete, which could be perceived as personal risks. Consequently, employees sometimes try to avoid the change. Ignoring or neglecting personal risks can turn into a serious risk for the entire SPI activity. Unfortunately, risk management is rarely applied to SPI, and an ignored risk is the worst kind.[10]

Experiences at DaimlerChrysler. We introduced a use case technique for requirements engineering in one group, not considering the risk this introduction posed for several group members. As electrical engineers, they were used to model-based development. Consequently, the new technique was neither openly rejected nor actively adopted. Because we couldn't modify the team's background, we decided to cancel the method introduction when we reconsidered it from a risk perspective. We thus immediately stopped wasting money and no longer risked the SPI initiative's reputation. Instead, we refocused and concentrated on a more appropriate improvement activity (improving documentation structure).

Recommendations. Apply risk management to your software improvement initiative and learning efforts just as you would to a software development project. Use experience-based checklists of common risks in your environment (such as personnel and budget risks), thus reusing knowledge and experience gained in earlier projects. Watch out for personal risks that an activity might create for important stakeholders. Stop an activity before you lose too much money and your reputation. Better yet, refocus activities long before risks turn into problems, and you'll avoid failure completely.

Experience value through packaging

The experience base must provide information in a format that is useful for intended users. At DaimlerChrysler, qualitative experiences are most common. *Packaging* refers to the activity of comparing, analyzing, and combining several pieces of raw experiences received from various projects (for example, from developers or project leaders). We organize this material according to the steps of the process models. The result is a document that represents either a consolidated view of the packaged experiences or modifications to an existing (process) model.

Packaging is a technical challenge because it requires identifying and working with several models.[4] The key is to make experience-related material relevant to a real user. This includes tailoring contents and format to a concrete anticipated usage situation. Experience is only valuable when set in context. We must base iteration, evolution, and learning on explicit information to form the seed for the next cycle.[11]

Experiences at DaimlerChrysler. SEC researchers observed *quality circles*: Quality assurance staff gathered in an informal way to exchange experiences. Unfortunately, they captured little information, writing down almost nothing. The few items they did capture were hardly reusable. To assist their experience capturing, the SEC team needed to develop expertise in the subject under discussion (such as inspections and review plans, and quality management issues). Over time, the QA staff adopted this practice and wrote down more information and introduced an experience base for quality assurance. Establishing the base was a lengthy process of describing, reworking, and representing (on paper and on slides) inspection processes, support material, and review plans.[8]

Recommendations. Develop packaging techniques and tailor them for your organization. Domain experts should create packages based on anticipated user needs. It should be made as simple as possible to create and use packages. Be aware of the importance and timeliness of feedback. Iterate often to know what the users really need. Don't fall in love with an idea you might have packaged, and don't try to reach the final solution in one iteration. Let the user see, use, and comment on partial solutions over several iterations.

Each issue we've discussed was more of a challenge than tool support and automation. We have learned to better cope with these challenges, but they remain the most decisive issues in implementing a learning software organization through explicit experience exploitation. We recommend providing advice on dealing with the issues, but each organization must find its own approach.

Once we recognized these challenges and dealt with them directly, the SEC ran more smoothly. It has improved many processes, and learning from experience has become a more natural part of daily life in the business units.

References

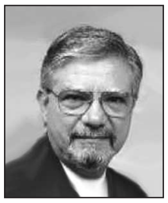
1. T.G.P. Davenport, *Knowledge Management Case Book*, John Wiley & Sons, New York, 2000.
2. R.V. Solingen et al., "No Improvement without Learning: Prerequisites for Learning the Relations between Process and Product Quality in Practice," *Product Focused Software Process Improvement* (PROFES 2000), Springer-Verlag, New York, 2000, pp. 36–47.
3. V. Basili, G. Caldiera, and D.H. Rombach, "The Experience Factory," *Encyclopedia of Software Eng.*, John Wiley & Sons, New York, 1994, pp. 469–476.
4. V. Basili and F. McGarry, "The Experience Factory: How to Build and Run One," *Proc. Int'l Conf. Software Eng.* (ICSE 19), ACM Press, New York, 1997, pp. 643–644.
5. V. Basili et al., "The Software Engineering Laboratory: An Operational Software Experience Factory," *14th Int'l Conf. Software Eng.* (ICSE '92), ACM Press, New York, 1992, pp. 370–381.
6. F. Houdek and K. Schneider, "Software Experience Center: The Evolution of the Experience Factory Concept," *Int'l NASA-SEL Workshop*, *Proc. 24th Ann. Software Eng. Workshop*, NASA Goddard Software Eng. Lab (SEL), Greenbelt, Md., 1999.
7. K. Schneider, "LIDs: A Light-Weight Approach to Experience Elicitation and Reuse," *Product Focused Software Process Improvement* (PROFES 2000), Springer-Verlag, New York, 2000, pp. 407–424.
8. K. Schneider and T. Schwinn, "Maturing Experience Base Concepts at DaimlerChrysler," *Software Process Improvement and Practice*, vol. 6, 2001, pp. 85–96.
9. K. Schneider, "Experience-Based Training and Learning as a Basis for Continuous SPI," *Proc. 6th Ann. European Software Eng. Process Group Conf.* (EuropeanSEPG), 2001.
10. E.M. Hall, *Managing Risk: Methods for Software Systems Development*, Addison-Wesley, Reading, Mass., 1997.
11. G. Fischer, "Seeding, Evolutionary Growth and Reseeding: Constructing, Capturing and Evolving Knowledge in Domain-Oriented Design Environments," *Automated Software Eng.*, vol. 5, no. 4, Oct. 1998, pp. 447–464.

For more information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.

About the Authors

Kurt Schneider is a researcher and project leader at the DaimlerChrysler Research Center, Ulm, Germany. He currently works in software process improvement, software quality, and lightweight approaches to software development. He has led large research projects—in particular the SEC project—with several business units and international partner companies. He studied computer science at the Friedrich-Alexander Universität Erlangen-Nürnberg, Germany, and received his doctoral degree in software engineering from the Universität Stuttgart, Germany. Contact him at DaimlerChrysler Research Center Ulm, P.O. Box 2360, 89013 Ulm, Germany; kurt.schneider@daimlerchrysler.com.

Jan-Peter von Hunnius is a researcher and PhD student at the DaimlerChrysler Research Center, Ulm, Germany. His research interests include experience-based process improvement, software development processes in general, extreme programming, and the rational unified process. He received his Diplom-Informatiker in computer science from the Albert-Einstin Universität Ulm, Germany. Contact him at DaimlerChrysler Research and Technology, Software Process Engineering (RIC/SP), P.O. Box 2360, 89013 Ulm, Germany; jan.hunnius@daimlerchrysler.com.



Victor R. Basili is a professor of computer science at the University of Maryland, College Park, and the executive director of the Fraunhofer Center, Maryland. He is also one of the founders and principals in the Software Engineering Laboratory. He works on measuring, evaluating, and improving the software development process and product and has consulted for many organizations. He is co-editor-in-chief of the *International Journal of Empirical Software Engineering*, published by Kluwer. He is an IEEE and ACM Fellow.