

# Case-Based Knowledge Management Tools for Software Development

SCOTT HENNINGER

scotch@cse.unl.edu

*Department of Computer Science & Engineering, 115 Ferguson Hall, CC 0115, University of Nebraska-Lincoln, Lincoln, NE 68588-0115*

**Abstract.** Modern software development is a knowledge-intensive activity. The proliferation of development tools, rapidly changing technology, and increasing complexity and diversity of application domains all increase the cognitive burden placed on software developers. General purpose programming languages and CASE tools offer little relief from these problems. Knowledge management tools are needed that can effectively capture and disseminate software development knowledge that applies to the domain-specific needs of an organization. This knowledge is not static, but evolves with technology and the changing needs of the organization's development practices, customer base, and business milieu.

This paper presents an infrastructure that supports evolving knowledge through case-based techniques and domain analysis methods that capture emerging knowledge and synthesize it into generally applicable forms. The approach is less concerned with the veracity of knowledge in its repository than evolving the knowledge toward answers to problems that fit the organization's technical and business context. Implications of this approach go beyond supporting software development to other knowledge-intensive professions where knowledge management tools can be used to support an organizational memory.

**Keywords:** case-based reasoning, knowledge management, knowledge-based software engineering, organizational memory, organizational learning, domain analysis

## 1. Knowledge requirements for software engineering

Modern software development is a knowledge-intensive activity. The number of resources available to the modern software developer is astounding. Process models, development methods, technologies, and development tools are all part of the modern software designer's toolbox, which also includes various tool kits, configuration management tools, test suites, standards, and smart compilers with sophisticated debugging capabilities, just to name a few. The vision of the software engineer carefully crafting language statements into a working program is obsolete, giving way to the employment of an arsenal of tools and techniques designed to support the coordination of work and the creation of systems that match the conceptual complexity demanded by modern software users. While it is unclear whether these tools have had the great impact predicted by CASE vendors, it is clear is that the "tool mastery burden" (Brooks, 1987) for software developers has increased.

But the software engineering community has for the most part ignored the knowledge burden placed on developers, opting instead to pile new methods, languages, tools, and formal methods onto the requisite knowledge heap. Knowledge of these technical resources, as well as the application domain and the programs being developed, are amongst the software

developer's most valuable assets. Currently, these resources are found scattered throughout development organizations in separate manuals, on-line databases, and documents with routing slips. Above all, software development remains an "oral culture" where knowledge is maintained as informal "folklore" (Terveen et al., 1993) passed between individuals.

Because the field is changing so rapidly, and because user needs are as volatile as today's business milieu, software development knowledge is constantly changing. Knowledge emerges in work practices, often being defined by the first project to address the issues involved. Lack of experience in domains such as client-server computing cause projects to constantly push the boundaries of an organization's development infrastructure. Smart planning and analysis can identify some of this knowledge, but there will always be hidden, obscure, and non-obvious issues that will be missed without some means to detect recurring patterns of problem solving activities. Refined knowledge is difficult to come by, but there is a need to disseminate what is currently known so the organization can build on successes, avoid duplicate efforts, and avoid repeating mistakes.

This paper presents a case-based method for collecting and managing software development knowledge as it evolves in an organizational context. Tools have been created to collect and disseminate project experiences as "cases" representing emerging knowledge of development practices in an organization. Situation-specific cases are augmented with organizing principles that help developers find applicable cases and helps the organization set standards in domains that are well-known or critical to the infrastructure of the organization. A methodology is being explored that complements the tools with a continuous process of knowledge acquisition so that existing knowledge can evolve to improve on previous efforts and meet the changing needs of customers (Henninger, 1995b, 1996b).

The following describes some of our empirical findings about knowledge needs in a software development organization. Our overall approach to using case-based technology for knowledge management is discussed, followed by a description of a set of tools for collecting and disseminating experience cases. Knowledge acquisition issues are illustrated through a knowledge acquisition cycle for continuous update of knowledge in the repository. The relationship between cases and organizing principles is then described and examples are given. The paper closes with a discussion of future directions.

### *1.1. Knowledge in a software development organization: A case study*

We have been working closely with a technically sophisticated Information Technology department at Union Pacific Railroad (UPRR), a major U.S. railroad corporation. This organization employs about 350 software developers and a number of consultants to develop in-house information systems supporting the corporation. The organization has been undergoing a general shift from data management on mainframe systems to decision support systems in a PC-based client-server environment. The shift has caused problems along a number of dimensions, including Unix server technology, communications, PC applications, and decision support systems. Issues of configuration, server location, system downtime and recovery, and others that were stable or did not arise in the mainframe world are becoming critical issues in need of effective organization-wide solutions. These problems