Building an Experience-Base for Product-line Software Development Process

Amr Kamel, Mark Chandra, Paul G. Sorenson
University of Alberta, Edmonton, Alberta, Canada,
{amr,chandra,sorenson}@cs.ualberta.ca

Abstract

Accumulating and managing development experiences plays a key role in improving software quality and process. The experience factory paradigm provides an organizational approach for extracting development experiences from current software product-line projects and supplying the experiences to future projects. The paradigm relies upon accumulating experiences and storing them in a repository, the experience base. Due to the complexity of the software process, it is difficult to establish and effectively provide operational support for the experience base.

This paper presents an approach to the implementation of the experience base along with an automated decision support system using Case-Based Reasoning. The paper covers the activities of case acquisition and representation. We also describe how this experience base can be used as a decision support system for software project managers in day-to-day development activities. The paper concludes with an evaluation of our approach to date and a description of future research directions.

Introduction

The concept of quality improvement has permeated the software industry since the 1990s. Improving software performance, reliability, robustness and time-to-market presented a growing need to develop and adopt quality improvement approaches to the software business. The continuous accumulation of evaluated experiences and associated learning presented a promising way to manage and improve the software quality as well as the development process (Basili 1985).

Many researchers (Basili, Caldiera, & Rombach 1994; Experience Base 1996) recognized the need for and proposed solutions to implement process feedback loops where information from previous projects is used to improve development activities of similar projects. They also identified the need to build business competencies by packaging successful experience for future reuse.

The Experience Factory Paradigm

The software Experience Factory (EF) paradigm (Basili, Caldiera, & Rombach 1994) was presented to institutionalize collective learning of the organization. The EF supports three different organizational units that interact with an *experience base* (EB): Support, Analysis and Packaging. The EB contains an integrated set of packaged experiences that capture past development competencies.

The ultimate goal of the EF is to support the continual improvement of the software quality by capturing, packaging and evolving sound development experiences as well as methods for providing packaged experience to the EF potential users.

Building and Running an EF

Although technology supporting the EF concept has been studied by many researchers (Althoff *et al.* 1999a; Henninger 1997), successful realization of the EF concepts for the software process is still a challenge (Tautz, Althoff, & Nick 2000). The challenges vary from defining exactly what constitutes a process experience and, how can it be captured, documented and stored, to institutionalizing effective mechanisms to select the most relevant experience from the knowledge base.

The EF can be started following two possible approaches top-down and bottom-up (Basili & McGarry 1996). That is proceeding either from a well defined ontology, to a schema for the EB, then collecting concrete experience data or else collecting concrete experiences and proceeding towards abstracted knowledge.

The rest of the paper is organized as follows. In the next section we start with an overview of existing approaches to implement an EF, and an introduction to our approach. Then, the nature of the software process knowledge and types of experience packages are discussed. This is followed by a section discussing our methodology for experience representation and acquisition. After that, requirements for the support system are discussed and an overview of the tool is given. Finally we conclude with evaluation of our approach and model to date, conclusions and future directions.

State of the Art

The EF paradigm (Basili, Caldiera, & Rombach 1994) has inspired a lot of research efforts aiming at building new EF(ies). Meanwhile, the ESPRIT III project (Experience Base 1996) researched tailored approaches to introduce EF

concept into organizations. Since the first reported EF at the Software Engineering Lab (SEL) (Basili *et al.* 1992), many industrial organizations, (e.g. Daimler Benz (Houdek, Schneider, & Wieser 1998), the Information Technology division of Union Pacific Railroad (UPRR) (Henninger 1997), and an Australian communication firm (Koennecker, Jeffery, & Low 2000)) reported advances in building their own EF(ies). Furthermore, different research groups reported their efforts in building EF(ies) that covered many aspects of software development. The reported EF(ies) covered the development process (Althoff *et al.* 1999b), experimental software engineering (Althoff *et al.* 1998) as well as domain-specific EF(ies) (e.g software development cost estimation (Finnie, Wittig, & Desharnais 1997), data mining applications (Bartlmae 1999) developing CBR applications (Althoff, Nick, & Tautz 1999; Bergmann *et al.* 1999), and ontology deployment (Kalfoglou & Robertson 2000)).

Research efforts in EF(ies) focused mainly on software knowledge representation and technologies to support creation and evolution of the EB. Research focusing on software knowledge representation aims at finding better ways to document and characterize experiences. Research focusing on technology aims at developing better ways to provide the packaged experience to its potential users.

Six types of software experience packages were identified in (Basili & Caldiera 1991): product, process, relationship, tool, management and data. Several approaches have been proposed in the literature to document and characterize these packages. Informal reports (Basili *et al.* 1992; Basili & McGarry 1996), structured text (Houdek & Kempter 1997; Birk & Tautz 1998), and formal languages (Ostertag 1992) have been used to document experiences. Experience package contents focus also varied from mainly hands-on experiences (Henninger 1997) to lessons learned (Tautz, Althoff, & Nick 2000).

Technologies suggested to support the EF were based on the expected size of the EB. Lists and indexed catalogues have been used to document experiences in a small to medium size EB; the experience was made available either by general purpose browsers (Griss, J., & Walton 1994) or through a human consultant (Houdek & Bunse 1999). For a large EB, Case Based Reasoning (CBR) technology is recommended (Tautz & Althoff 1997) and used (Henninger 1997) to document and disseminate experience packages.

For reasons of simplicity and ease of use, we studied software development experiences, how can they be acquired, documented, packaged and retrieved when needed. Our long term goal is to cover all experiences related product-line development based on object oriented frameworks, however, we started by focusing on the product-line process experiences. A process experience package has a life-cycle process as its center element, providing information on how to enact it and lessons learned from previous enactions, e.g. process models, methods (Basili & Caldiera 1991).

Our Approach

Our goal is to build an EB to support developing many software products using a common Object Oriented Framework. The idea is to develop an EB around the framework that can be shipped with the framework as a development support tool. For developing the EB we favored the bottom-up approach over the top-down for many reasons:

- Top-down approach assumes a relatively stable environment (Althoff *et al.* 1999a). This assumption does not hold true in general, as many organizations favor a dynamic development environments to cope with short technology cycles.
- Building a complete schema for the EB implies making assumptions about how the organization functions. These are assumptions that we can't make because, in general, we don't have much information about the environment of the target organization that will deploy the OO framework.
- For simplicity, we started with the simplest solution that would work as recommended by newer software development methodologies (Beck 1998).
- Our approach focuses on data collected from hands-on experiences. Most of documented EF implementations following the top-down approach are either based on long-term application (Basili *et al.* 1992), or experimental data (Althoff, Nick, & Tautz 1999).

To further support knowledge abstraction we distinguished between two types of knowledge: concrete experiences and best practice. We built an EB, using CBR technology to support experience documentation and EF functioning. The EB we developed aimed at supporting an OO Framework called CSF (Client Server Framework) (Froehlich 1999). The EB was developed in part from fifteen projects completed in three consecutive offerings of a senior software engineering class at the Department of Computing Science, University of Alberta from January 1999 to April 2000. The developed projects were required to use the CSF as basis for their self-chosen application.

Types of Experience Packages

In the context of our work, we view the experience as *"practical knowledge or skill abstracted or directly observed from participation in particular activity."* We are interested in the (abstracted knowledge, direct participation) tuple; for simplicity we will refer to the tuple as (knowledge, participation). This tuple implies that we are especially interested in knowledge accumulated during everyday's work in the organization. However, knowledge and participation need not be reported in the same package. In fact, participation reports are viewed as the concrete knowledge from which abstract knowledge may be deducted.

The (knowledge, participation) tuple implies that development knowledge can be divided into: *concrete* and *abstract*. Concrete knowledge captures hands-on experiences. Abstract knowledge supports the decision making process of a project by offering packaged solutions to its problems. This knowledge may come from in-house experiences, experimental results or the software industry at large. Despite the origin of the abstract knowledge, it should be continuously refined using related concrete packages.

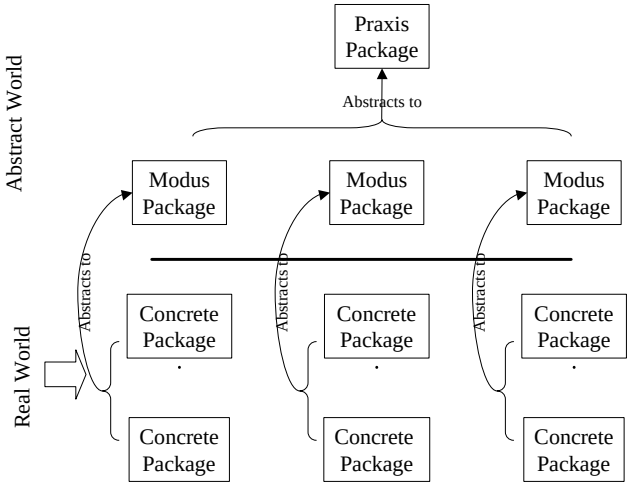


Figure 1: Different Levels of Experience Packages

Abstract knowledge is essential for the functioning of the EF Support unit. Concrete knowledge is essential for the proper functioning of the Analysis unit. Furthermore, disseminating concrete knowledge to the project organization guides experience consumers (e.g. project manager) through the usage of the relevant abstract knowledge, especially when some of the "abstracted out" details are needed.

Process experiences need to be packaged in a variety of ways to fulfil different interests of its users. For example, during project planning, experience base users are more interested in exploring options to decide on the set of processes to use. At this stage, they are interested in process merits and major risks, inter-process interactions and trade-offs, rather than how to enact it. When a particular process is chosen, users' interests shift to issues like comparing the different methodologies to enact the process, and how to measure its success or manage its risks.

While concrete knowledge can be packaged as one type (*concrete* type), abstract knowledge needs to be packaged differently to fulfill different users' interests. To emphasize these differences abstract knowledge is packaged as either: *praxis* and *modus* types. The three experience package types can be described as follows:

Praxis. Praxis packages document industry best practices. Praxis packages are general in nature, documenting for example, the efficacy of a process, the merits of a tool, with enaction details abstracted out.

Modus. Modus packages focus on the details of a particular process or best practice. A modus package may document a particular methodology for enacting the process and, as necessary, clarify how to perform its sub-processes.

Concrete. Concrete packages are tightly related to the real world; they document hands-on experiences. A concrete package reports on how an abstract package is enacted in a given organizational context, and whether the practice was a success or a failure. The package may, but not necessarily, come with a recommendation of "what to do and/or avoid".

Generally, praxis packages capture the merits of the industry best practices; modus packages represent methodologies of enacting these practices and concrete packages describe the experience gained by participation on process enactions.

The three package types can be viewed as representing development experiences at different levels of abstraction, see figure 1. Each enaction of the process is acquired as a concrete package. By analyzing a set of similar concrete packages, environment particulars are abstracted out and the knowledge is represented as one modus package. Details of the enaction methodology are further abstracted out to be documented as one praxis package. For example, various methodologies of performing technical reviews (e.g. Fagen inspection and IEEE standard review) are represented as different modus packages. However, the merits and risks of technical reviews (despite the particulars of the methodology) are represented as one praxis package.

Experience Representation and Acquisition

Several problems are presented while determining and representing experience packages. At the macro level we must address the question: "What is the proper level of granularity for an experience package?." At the micro level, the main question is: "how to characterize different experiences".

Closely related to experience representation is experience acquisition. Where to acquire experiences? When to say that available experience packages are enough for the organization's needs?

Experience Representation

An experience package may represent a process-step (e.g. inspection kick-off meeting), a process (e.g. technical review) or a complete development methodology (e.g. eXtreme Programming (Marcheli & Succi 2000)). At the concrete level, process methodologies are too general to package. On the other hand, we anticipate that process-steps are too specific for our purpose. We set the granularity level of our experience packages at the process level. A process is defined as (Kamel *et al.* 1997): *a set of process-steps, with well defined roles, inputs, outputs to serve a common objective*.

The core of a process package is the *(objective, description)* tuple. Process objectives are the key characteristic that set processes apart; in a sense they represent the "problem(s)" addressed by the process. Each process package must have at least one objective or goal to achieve. For practical considerations, no other constraints are imposed on process packages; it is acceptable to include a process package without well defined input/output or missing a clear definition of roles, etc.

It is also evident that successful process experiences are not globally optimum (Shirey 1992).They are optimal with respect to their enaction environment. Hence, reporting a process alone is not enough, the *context* of the knowledge contained in a process package is also important to report, specially for concrete packages.

The goal of the EB is to categorize packaged experiences based on certain *Features*. A typical feature may be, chance of success at first enaction, or type of training required to perform the process successfully. Process features may not be clear at first, they are determined following an iterative process. Selection of new features to represent a package, is determined by studying the set of available experiences, studying the discriminatory power of the selected features, modifying them if necessary, then starting the next iteration. For example, by analyzing inspection's concrete packages, we might find out that formal inspection training increases the chances of success of the inspection process. Hence, "requires formal training" would be added as a feature for the inspection modus package.

Other information about the process (e.g. references, comments) is also required in the package. The information in a process package is reported in a structured text format following an *experience template*. An experience template consists of:

Name: a unique identifier for the experience.

Type: Praxis, Modus or Concrete.

Objective: describes the objectives of the process described in the package.

Description: is a detailed description of the actual experience.

Context: characterization of the environment from which the experience was acquired.

Features: features of the experience that make it distinctive from other experiences in the EB.

Related Experiences: listing of experience packages semantically linked to current experience (e.g. uses, contains), as well as information for navigation among experiences (e.g. linking inspection Modus package with corresponding inspection concrete packages)

References: Additional material discussing the experience (books, articles, manuals, etc.).

Comments: any additional information important for using the experience.

Administration: listing of administrative information.

Experience Acquisition

There are three basic ways to gain abstract experiences (Birk & Tautz 1998).

- Use available technical knowledge sources.
- Use goal-oriented knowledge acquisition.
- Accumulate knowledge during everyday work.

Ideally praxis and modus packages should come as a result of an abstraction process of concrete experiences. However, restricting them to this path is ineffective, as it neglects the available knowledge accumulated in the industry. Abstract packages may be acquired from experiences internal or external to the organization. External experience are acquired primarily from software engineering publications. Internal experiences are the result of analyzing and abstracting different concrete packages. Concrete packages are acquired directly from development projects.

Candidates for a praxis package are either processes fundamental to software development (e.g. configuration management) or processes which have positive effect on development schedule, process visibility and product characteristics (e.g. maintainability or usability). Apart from fundamental processes, best practices are usually associated with a development methodology.

Current Status of the EB

To manage the different levels of knowledge abstraction in our model, we implemented the EB as a collection of knowledge bases (KB). A KB is dedicated to best practices (praxis package type). Moving down a level of abstraction, a KB is dedicated to knowledge about how to enact a particular process (modus packages). Finally, concrete packages were collected in yet another KB. We found that partitioning the knowledge in this manner helps produce manageable size experience bases, and eases further analysis of the information. Currently, we have three KBs: *Rapid Development Best Practices (RDBP)*, *Technical Reviews (TR)* and *Information Swapping Concrete experiences (ISCE)*.

Packages in the (RDBP) (27 packages) were acquired from the set of best practices in rapid development methodology identified by McConnell (McConnell 1996). These practices are directly associated with development speed and process visibility.

To date, all our modus packages are focused on technical reviews and inspections. A survey of the software engineering literature resulted in acquiring 18 modus packages representing different inspection, technical review and walkthrough mechanisms proposed by researchers and industry experts.

The 15 packages in the ISCE were acquired from an experimental study (Froehlich, Kamel, & Sorenson 2000). The study was conducted over three consecutive offerings of senior software engineering classes at the Department of Computing Science, University of Alberta. As part of the course requirements, students were grouped into small teams to develop small software systems as part of a term projects. Each project group used a common client-server object-oriented framework (CSF) (Froehlich 1999) as a basis for their software project. The study tracked projects performance using surveys, questionnaires, weekly meetings and examining projects documentation. The questionnaires, forms and project documentation are the tools we used to acquire the concrete packages.

Automated Support for the EB

To support the EF goals a strategy and supporting tools to capture and disseminate development experiences are needed. In our approach, experiences are captured using WWW technology and either browsed or retrieved via a specialized decision support system. To retrieve an experience package, the user starts with a problem description, then interactively converges towards the target package.

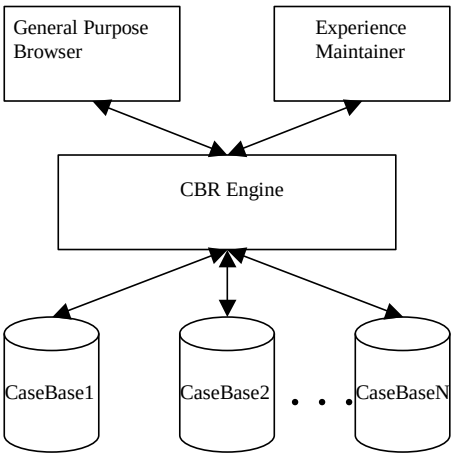


Figure 2: Tool Architecture

System Requirements

In addition to the general requirements to support an EB reported in (Althoff, Bomarius, & Tautz 1998)[1], we identified these requirements to match our target environment. A proposed system should:

- interactively guide the user through package retrieval process. EB users may not know the built-in knowledge classification structure. The system must guide the package selection process by asking specific questions about the problem context to retrieve the best solution(s) that matches the problem's environment characteristics.
- support flexible strategies to acquire information. The lack of knowledge about the target development environment assumes that standard acquisition forms may create barriers to acquiring new experiences. The acquisition forms should be flexible enough to integrate easily into the several development environments without affecting the system's internal knowledge representation.
- rely on familiar technologies (e.g. WWW) to acquire and disseminate information. Learning curve for software tools is a major reason for rejecting them in practice (McConnell 1996; Beck 1998). The tool must be easy to use with familiar look and feel.
- support different execution models (e.g. stand alone, client server). Due to the lack of knowledge about the target environment, the tool must integrate easily with different development setups.

In view of the above requirements, we evaluated candidate approaches to support the EB (Kamel, Chandra, & Sorenson 2001) and found that CBR and WWW technologies are most suitable for our purposes. We used CASEADVISOR^{TM} (Yang, Kim, & Racine 1997) - a commercially available CBR tool - as a CBR engine. Other parts of the system were realized using WWW technology and scripting using CGI and perl.

[1]Our system is targeting process artifacts rather than all kinds of software knowledge artifacts

System Architecture and Implementation

The system consists of three components: *CBR engine, experience-maintainer, and general purpose browser* interacting with a set of case bases (as seen in Fig 2). To date, three case bases have been built for the RDBP, TR and ISCE knowledge bases discussed earlier.

CBR Engine

CASEADVISOR© is an intelligent problem diagnosis and resolution system for applications in enterprise knowledge management. The core technology of the system is CBR, however, it contains more features, e.g. decision trees and constraint satisfaction algorithms. CASEADVISOR is developed by the CBR group at Simon Fraser University using case based reasoning technology. Stand-alone and client-server versions of the tool are available to work on either a PC or via an internet connection. The system consists of two main modules: CASE AUTHORING module to build a case-base and PROBLEM RESOLUTION to use it. In addition to the case name, a case is described by *problem description* and *problem solution*. The nearest neighbor formula is used to retrieve cases. Case similarity is assessed by ranking cases based on keyword matching. Feature-value pairs may be used to increase the ability to distinguish cases in form of question-answer pairs.

Experience Maintainer Module

The effect of EB maintenance activities can be local (affects a particular experience package) or global (affects all packages in the EB). Typical EB maintenance activities for the EB are (Nick & Althoff 2000):

- Add newly acquired knowledge.
- Remove obsolete packages.
- Add/modify a context parameter or a feature to all packaged experiences.

The Experience Maintainer module is implemented using perl and CGI scripts. Maintenance processes starts with extracting experience packages currently stored in the EB. A summary of the EB information along with maintenance activities options are presented to the maintainer as in Fig 3. The summery lists all package names, package features along with the values that feature may take. Depending on the nature of the required update, the maintainer will be presented with forms to either update the information in a single experience package or add a new feature to the EB. If a new feature is added, the next step is to associate a value for that feature for the experience packages. Finally, the updated information is posted back to the EB.

Browser Module

The hierarchical nature of the EB required browsing along two dimensions: (1) within the same case base or (2) from one case base to another. Browsing within the same case

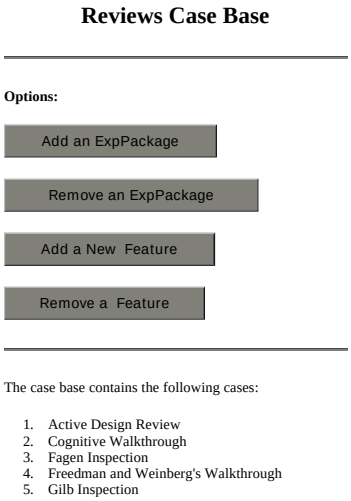


Figure 3: EB Maintenance Screen

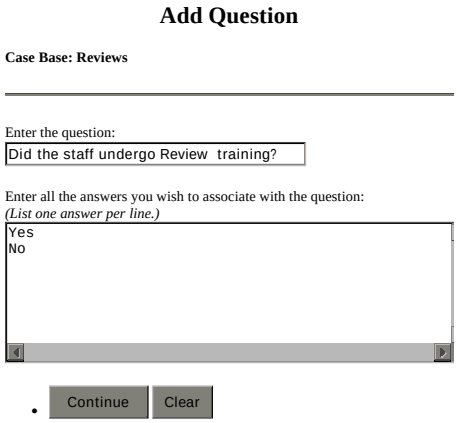


Figure 4: Add a new question

base is required for the EB support activities. While selecting an experience package, the need for mechanisms to navigate through different levels of the EB was evident.

The browser module is realized using full advantage of CASEADVISOR features. Browsing the same case base is provided by default in the PROBLEM RESOLUTION module. For inter case base browsing we used the ”invoking files” feature in CASEADVISOR case description. In the solution description, experience packages are set to invoke and run a new PROBLEM RESOLUTION module using the target case base. At present, browsing is limited to one step either up or down the EB hierarchy.

Current Status of the Tool

Currently, a prototype of the tool has been implemented. The EB has been populated with a total of fifty experience packages (distributed over 3 case bases). Fourteen features have been identified and included in the EB. Our immediate efforts are focused on abstracting more features to increase the ability of the CBR engine to distinguish the cases.

Usage Scenarios

These scenarios demonstrate how the tool is used in practice. The first scenario describes how a typical EF customer would use the tool to find the most similar experience package. The second scenario describes how the tool is used to modify the EB.

Selecting an Experience Package

Assume the project manager of a particular project decides to use some new technique to enhance the product quality. Querying the RDBP experience base, he concludes that technical reviews seem like a promising technique.

To find out what technical review process best suites the project needs, the project manager invokes the TR experience base and compare different review methodologies to find the methodology that best suites the project context and needs. Let us assume he chose code inspection.

Next, the project manager needs to invoke the concrete Inspection experience base to review hands-on experiences within the organization. By retrieving these experiences, information about the effect on the project schedule, and code quality are retrieved and assessed.

At this point, the project manager might decide that the inspection process is very human intensive and stretching beyond the project’s available resources. He can invoke the RDBP experience base and query it to find out other practices that can positively affect the product quality and adheres to the project’s constraints.

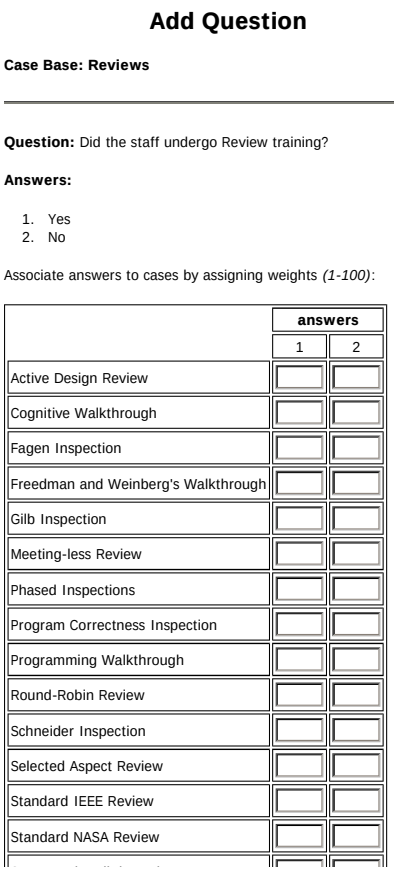
Add a New Feature

Assume that analyzing available concrete packages for technical reviews reveals that the chance of successfully enacting certain technical review methodologies highly correlates with formal staff training. To incorporate this new feature ”requires formal training” in the TR experience base, he begins by adding the question “Did the staff undergo review training?” and all its plausible answers: “Yes” and “No” - see screen shot in Figure 4). After hitting continue, the script will present the domain expert with another form (see screen shot in Figure 5) that contains all technical review methodologies stored in the case base with the option of associating the new question with all stored cases and individually setting the weights of each answer. Finally, the question and answer’ weights for all the associations are posted to the experience base by hitting ‘continue’ button.

Conclusions and Future Directions

Continuous accumulation and reuse of process experiences is a suitable way to manage and improve the software quality as well as the development process. In this paper we presented an approach to capture, structure, store and retrieve these experiences.

The main concept behind the knowledge structure we adopted is to simultaneously capture and maintain abstract



Add Question

Case Base: Reviews

Question: Did the staff undergo Review training?

Answers:

1. Yes
2. No

Associate answers to cases by assigning weights *(1-100)*:

	answers	
	1	2
Active Design Review		
Cognitive Walkthrough		
Fagen Inspection		
Freedman and Weinberg's Walkthrough		
Gilb Inspection		
Meeting-less Review		
Phased Inspections		
Program Correctness Inspection		
Programming Walkthrough		
Round-Robin Review		
Schneider Inspection		
Selected Aspect Review		
Standard IEEE Review		
Standard NASA Review		

Figure 5: Associate questions with cases

and concrete process knowledge. The advantages of this structure is:

- Both the knowledge and its roots are captured and stored in the knowledge base.
- Experience from internal and external sources can be easily fitted into the structure. However, only the internal experiences are supported by concrete knowledge.
- The KB is refined as soon as a concrete experience is captured, rather than waiting till the experience is abstracted and packaged into the EB.
- Abstracted knowledge can be continuously reevaluated based on the accumulation of concrete experiences, supporting the main concept of the EF paradigm.
- Abstract knowledge is packaged in a variety of ways to fulfil different needs of its users. Abstracted knowledge may be packaged as, modus or praxis type experience depending on its intended use.

To support the management and use of captured experiences, we implemented an automated decision support system using CBR technology to capture reason about and disseminate the concrete experiences. CBR proves to be a powerful tool for reasoning about the software process as KB can propose solutions without a full understanding of all the factors affecting the process. Furthermore, encoding experiences as cases is simple and communicating them back and forth with domain experts is straightforward.

The tool needs more work. A richer KB is need on both the abstract and concrete levels. We need to include recommended practices from other development methodologies (e.g. eXtreme Programming). Robust navigation mechanisms and, the ability to integrate the tool with more than one CBR engine are also required. In addition we plan to integrate it into the development environment of the software engineering course (http://peerless.cs.ualberta.ca/cafe401) to test it in its intended environment.

A more challenging problem that we are addressing right now is: Given a set of "concrete packages", what abstractions can be made to add or modify a modus package using automated or semi-automated techniques. There is a need to develop more productive approaches for knowledge abstraction. Currently we are experimenting with statistical techniques and qualitative research methodologies to address this problem.

References

Althoff, K.-D.; Birk, A.; von Wangenheim, C.; and Tautz, C. 1998. CBR for experimental software engineering. In *Case Based Reasoning Technology*. Springer. chapter 9, 235–254.

Althoff, K.-D.; Birk, A.; Hartkopf, S.; Müller, W.; Nick, M.; Surmann, D.; and C., T. 1999a. Managing software engineering experience for comprehensive reuse. In *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering (SEKE99)*., 10–19.

Althoff, K.-D.; Bomarius, F.; Mller, W.; and Nick, M. 1999b. Using case based reasoning for supporting continuous improvement processes. In Perner, P., ed., *Proceedings of the 12th German Workshop on Machine Learning*.

Althoff, K.-D.; Bomarius, F.; and Tautz, C. 1998. Using case-based reasoning technology to build learning software organizations. In *ECAI98, workshop on Building, Maintaining and Using Organizational Memory (OM-98)*.

Althoff, K.-D.; Nick, M.; and Tautz, C. 1999. CBR-PER: A tool for implementing reuse concepts of the experience factory for cbr systems. In *Proceedings of the 7th German Conference on Knowledge Based Systems (XPS99) Workshop on Case-Based Reasoning*.

Bartlmae, K. 1999. An experience factory approach for data mining. In *Proceedings of the 2nd Workshop in Data Mining and Data Warehousing as Basis of Modern Decision Support Systems*.

Basili, V., and Caldiera, G. 1991. Methodological and architectural issues in the experience factory. In *Proceedings of the 16th Annual Software Engineering Workshop, NASA/GSF*, Software Engineering Laboratory series.

Basili, V., and McGarry, F. 1996. The experience factory: How to build and run one. In *Tutorial at the 18th International Conference on Software Engineering.*

Basili, V.; Caldiera, G.; McGarry, F.; Pajerskiand, R.; Page, G.; and Waligora, S. 1992. The software engineering laboratory – An operational software experience factory. In *Proceedings of the 14th International Conference on Software Engineering*, 370–378.

Basili, V.; Caldiera, G.; and Rombach, H. 1994. The experience factory. In Marciniak, J., ed., *Encyclopedia of Software Engineering.* John Wiley & Sons. 468–476.

Basili, V. 1985. Quantitative evaluation of software engineering methodology. In *Proceedings of the 1st Pan Pacific Computer Conference*.

Beck, K. 1998. Extreme programming: A humanistic discipline of software development. In *Fundamental Approaches to Software Engineering: Proceedings of the 1st International Conference, FASE98*, 1–6.

Bergmann, R.; Breen, S.; Gker, M.; Manago, M.; and Wess, S. 1999. *Developing Industrial Case Based Reasoning Applications - The INRECA Methodology*, volume 1612 of *Lecture Notes in Artificial Intelligence (LNAI)*. Springer-Verlag.

Birk, A., and Tautz, C. 1998. Knowledge management of software engineering lessons learned. In *Proceedings of the 10th conference on Software Engineering and Knowledge Engineering (SEKE98)*.

1996. Experience base. A booklet from the PERFECT ESPRIT project 9090 handbook edition.

Finnie, G.; Wittig, G.; and Desharnais, J.-M. 1997. Estimating software development effort with case based reasoning. In *Proceedings of the 2nd International Conference on Case Base Reasoning*, 13–22. Springer-Verlag.

Froehlich, G.; Kamel, A.; and Sorenson, P. 2000. Exploring O-O framework usage. In *22nd International Conference on Software Engineering, Research Poster*.

Froehlich, G. 1999. Client-Server Framework. http://www.cs.ualberta.ca/∼garry/framework.

Griss, M.; J., F.; and Walton, P. 1994. Managerial and organizational issues - starting and running a software reuse program. In Schfer, W.; Prieto-Diaz, R.; and Matsumoto, M., eds., *Software Reusability*. Ellis Horwood Ltd. chapter 3, 51–78.

Henninger, S. 1997. Capturing and formalizing best practices in a software development organization. In *Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering (SEKE97).*

Houdek, F., and Bunse, C. 1999. Transferring experience: A practical approach and its application on software inspection. In *Proceedings of the Workshop on Learning Software Organizations at the 11th International Conference on Software Engineering and Knowledge Engineering (SEKE99)*, 59–68.

Houdek, F., and Kempter, H. 1997. Quality patterns - An approach to packaging software engineering experience. *Software Engineering Notes* 22(3):81–88.

Houdek, F.; Schneider, K.; and Wieser, E. 1998. Establishing experience factories at daimler-benz: An experience report. In *Proceedings of the 20th International Conference on Software Engineering*, 443–447.

Kalfoglou, Y., and Robertson, D. 2000. Applying experienceware to support ontology deployment. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE00)*.

Kamel, A.; Voruganti, S.; Hoover, H. J.; and Sorenson, P. 1997. Software process improvement model for small organization: An experience report. In *Proceedings of the Annual Oregon Workshop on Software Metrics (AOWEM97)*.

Kamel, A.; Chandra, M.; and Sorenson, P. 2001. Supporting knowledge management of product-line software development. Technical Report UA-SERL-01.001, Software Engineering Research Labs, University of Alberta.

Koennecker, A.; Jeffery, R.; and Low, G. 2000. Implementing an experience factory based on existing organisational knowledge. In *Proceedings of the Australian Software Engineering Conference (ASWEC00)*.

Marcheli, M., and Succi, G., eds. 2000. *eXtreme Programming and Flexible Processes in Software Engineering - XP2000*.

McConnell, S. 1996. *Rapid Development*. Microsoft Press.

Nick, M., and Althoff, K.-D. 2000. The challenge of supporting repository-based continuous learning with systematic evaluation and maintenance. In *22nd International Conference on Software Engineering. Workshop on Intelligent Software Engineering (WISE3).*

Ostertag, E. 1992. *A Classification System for software Reuse.* Ph.D. Dissertation, University of Maryland.

Shirey, G. 1992. How inspections fail. In *Proceedings of the 9th International Conference on Testing Computer Software.*, 151–159.

Tautz, C., and Althoff, K.-D. 1997. Using case-based reasoning for reusing software knowledge. In *Proceedings of the 2ndth International Conference on Case Based Reasoning (ICCBR97)*, 156–165.

Tautz, C.; Althoff, K.-D.; and Nick, M. 2000. A case-based reasoning approach for managing qualitative experience. In *Intelligent Lessons Learned Systems: Papers from the Workshop at 17th National Conference on AI (AAAI00).*, 54–59. The AAAI Press.

Yang, Q.; Kim, E.; and Racine, K. 1997. CASEADVISOR: Supporting interactive problem solving and case base maintenance for help desk applications. In *IJCAI'97, Workshop on Practical Use of CBR.*