Appears in *Third Int'l Conf. on Product Focused Software Process Improvement*, Kaiserlautern, Germany, Sept. 2001, pp 182-195.

A Tool for Managing Software Development Knowledge

Scott Henninger, Jason Schlabach

Department of Computer Science & Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588-0115
{scotth, jschlaba}@cse.unl.edu

Abstract. Software development is a knowledge intensive activity involving the integration of diverse knowledge sources that undergo constant change. Most approaches to knowledge management assume that information exists and is readily sought by software developers, usually through a search tool. In this paper, an approach is presented that actively delivers information through a rule-based system that matches system requirements to experience-based knowledge embedded in work breakdown structures. A reuse-based methodology based on an organizational learning process is used to capture and organize knowledge as it is created. The combination of tool and methodology work together to capture characteristics of individual projects and tailor processes to meet diverse and emerging software development needs. The tool and methodology are demonstrated using two examples of how this technique can be applied

1 Knowledge Management For Software Development

Software development is a knowledge-intensive activity involving the integration of diverse knowledge sources that undergo constant change. The knowledge required to successfully develop software systems ranges from programming techniques to the "tool mastery burden" [6] to the critical role of application domain knowledge [10] and the diverse nature of software development processes and techniques [27]. In addition, the knowledge is under constant pressure from fluctuating requirements and technology advances, causing knowledge volatility and further pressures on the knowledge mastery burden.

The management of this knowledge and how it can be brought to bear on software development efforts has received little attention in the software engineering research community. Most software engineering methodologies view the project as the object of study, assuming that the project begins with a blank sheet of paper and leaves a documentation trail only to assist the maintenance phases of the project. This leaves little basis that the organization, or the community as a whole, can use to understand and learn from previous development efforts. Tools, techniques, and methods are needed that capture knowledge in a continuous knowledge acquisition process and disseminated in a form that can be brought to bear on subsequent development efforts.

Traditional methods to capture and organize knowledge use repository technology coupled with a search engine. While this technique addresses a real need for information searching, it tacitly assumes that people know what information exists and can articulate their information need properly. Although this assumption can be questioned in general information retrieval settings [5], it is particularly dubious in software engineering, where studies have shown that functions and methods are regularly re-implemented, often differing only in name [11]. Tools and techniques are needed that not only allow search for information, but also actively alert software developers to the existence of potentially relevant knowledge.

In the following sections, we present a tool and methodology for software development knowledge management that draws on past development experiences and actively presents information of potential relevance to development efforts. First, the overall system infrastructure and methodology are presented, followed by two examples of how this technique can be applied. We finish with a discussion of what has been learned from this research and future directions.

2 Building An Organizational Repository Of Experiences

An organizational learning approach to software development [22] captures project-related information during the creation of individual software products which is then disseminated to subsequent projects to provide experience-based knowledge of development issues encountered at a software development organization. These principles have been demonstrated through an exploratory prototype, named BORE (Building an Organizational Repository of Experiences) [17].

Early BORE research focused on developing tools to support the creation of case-based repositories [17]. Evaluations of these prototypes [16] revealed issues often found when adopting software tools [23]. In particular, we found that BORE activities were regarded as a pure documentation effort with few perceived benefits, and was therefore seen as ancillary to the immediate goal of producing a working software product. Because BORE was not intimately tied to these goals, people regarded using the system as supplemental and not part of the critical path. Despite our efforts to present the organizational learning approach [17, 19], the simple repository tools provided by BORE were insufficient to allow people to draw on past experiences. Tools were needed that actively presented relevant information, guides people through the methodology, and collects and disseminates this knowledge as part of the development process.

The work described here draws on these experiences and extends the BORE work by coupling organizational memory tools with software process tools based on a work breakdown structure. The studies also demonstrated the need to guide the use of the tool with a corresponding methodology that addresses how existing information is brought to bear on new projects, and how new knowledge is captured to meet the emerging needs of software development efforts.

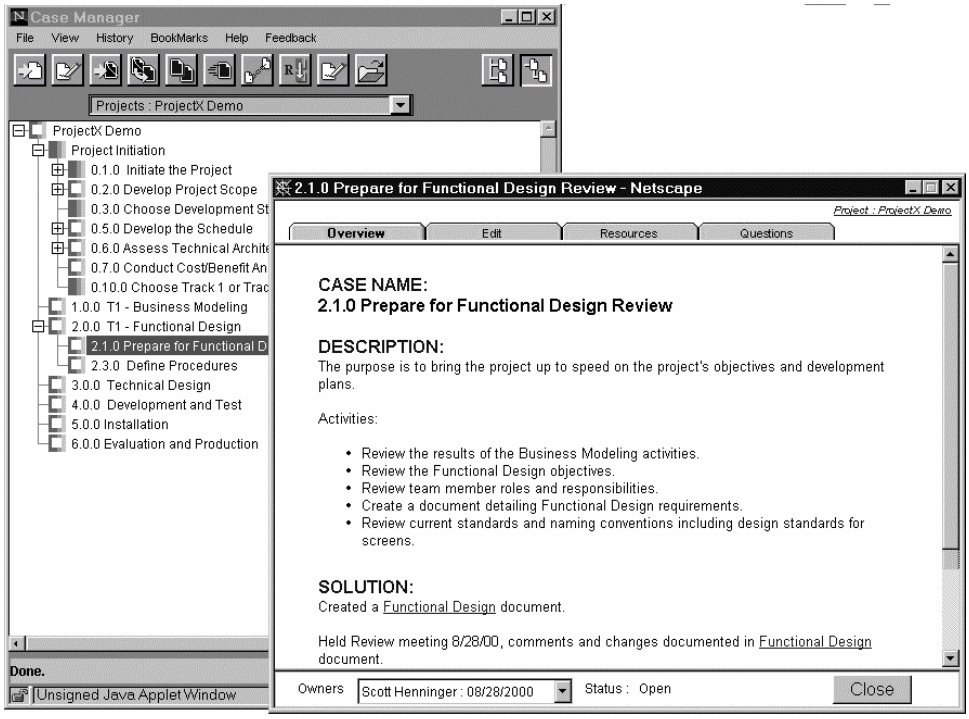

Figure 1: BORE Case Manager and a case.

2.1 The BORE Prototype

The BORE prototype is a Web-enabled application using a three tiered architecture consisting of HTML and Java AWT for rendering the interface, Java for application logic, and relational database back-end. It can be accessed through a Web http://cse-ferg41.unl.edu/bore.html (log in as 'guest')[1].

The main interfaces for BORE are shown in Figure 1. The Case Manager, shown to the left in Figure 1, displays a hierarchical arrangement of project activities. In the figure, a project named "projectX demo" has been chosen from the list of resources that can be displayed in the Case manager's tree view display. Each case contains project-specific information as shown in the window to the right in the figure, which was obtained by double-clicking on the activity named "Prepare for Functional Design Review" in the project hierarchy.

Cases are used in a case-based decision support approach [25], and describe situation-specific solutions to software development activities. Cases consist of a description to a problem, a solution statement, and some ancillary information, such as related cases, etc. For example, in Figure 1, the case describes a specific activity of preparing for a functional design. The description was copied when the case was added to the project. The solution field is then used to document project-specific

[1] Results are best if using Netscape Communicator 3.1 or greater or Internet Explorer 5.0 or greater. Java and Javascript must be enabled.

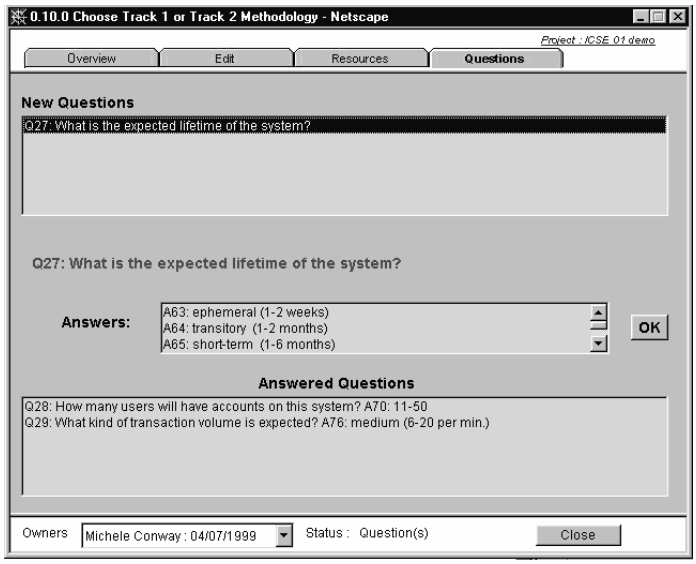

Figure 2: The Options tab.

information accumulated while executing the activity. In addition, a color-coded icon is used in the Case Manager to depict the status of a case (active, resolved, etc.), thus providing a way to visually summarize a project's status.

2.2 BORE Knowledge Domains

Bore divides its repository into domains. Domains are independent knowledge realms that consist of a set of domain *cases* defining activities and domain *rules* that define the context under which a process is applicable to a development effort. Currently, projects belong to a single domain, which is chosen when a project is created in BORE. All subsequent project activities will use the cases and rules defined for that domain. This supports scalability and allows organizations to partition development activities into domains that address the needs of diverse units and/or product lines.

Domain Cases define the space of possible activities for projects within a given domain, structured in a work breakdown hierarchy. They define standard activities that have proved useful for situations encountered by projects within the domain. The cases describe some of the problems or necessary steps that must be taken to ensure that the process is properly followed, and can be updated by a project to reflect specific activities the project follows. For example, an activity may dictate that the corporate standard login screen is used for an application. A project trying to follow this step may find that using the password database required a couple of work-arounds to fit their specific architecture. The project's case would reflect these problems and document how the problems were solved in the "Solution" field of BORE cases (see Figure 1).

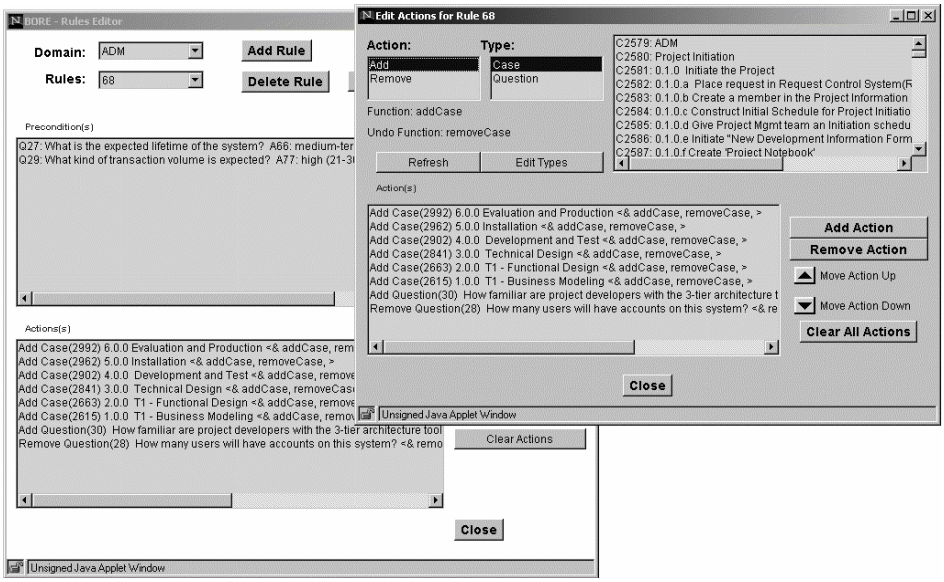

Figure 3: The Rules Manager.

Domain Rules are used to tailor the overall development process to the specific needs of different projects. Internally, BORE uses a simple forward chaining production system implemented using an SQL database to represent rules. Rules consist of a set of preconditions and actions. Preconditions are represented by question/answer pairs. Questions are associated with the Options tab of a case (see Figure 2). Cases that have options assigned to them appear in the Case manager with a '?' in the status icon. Options are chosen by project personnel to tailor the organization's standard process to individual project needs. For example, Figure 2 shows a session in which two questions have already been answered (i.e. the options were chosen) and a third has been chosen, revealing the possible answers. These options address high-level project requirements that may impact which activities are chosen for the project. Options can be designed to address requirements at any level, from business concerns to development tools to reusable components.

When all preconditions of a rule evaluate to true, the rule "fires," causing a set of defined actions to be executed. The core actions in BORE can remove questions from the question stack (the New Questions pane of Figure 2), add a question to the question stack, or add a case to a project. In addition, new action types can be created by attaching a Java method to the action. BORE supports rule backtracking that allows rule actions to be undone when a precondition changes in such a way that rule should no longer be fired. Therefore, all action types supporting backtracking need to develop both an action and an undo method.

2.3 Creating and Modifying Domains

During project execution, knowledge in a BORE domain is refined in two ways. First, a new case is instantiated for each process element assigned to the project. This case will document how the project met the requirements of the given process element. This will also provide a resource for subsequent development efforts because one can look at the cases for all projects assigned a given activity. The second modification to the repository involves creating new domain rules and cases to reflect project activities that should become part of the overall domain. For example, if a project determines that a Java Plug-in is necessary when developing applets that must run both cross-platform and cross-browser, a review team may decide that the project's action are setting a precedent of significant value to other projects. Modifying the domain to reflect this new knowledge involves 1) creating a new rule, and 2) creating new domain cases, and 3) associating new options with domain case(s).

Creating Rules. Rules are edited by selecting a rule and editing the associated preconditions and actions. Figure 3 shows the interface for editing rules. Clicking on the Edit Preconditions button pops up another window (not shown) that lists all questions in the domain. For each precondition clause a question is chosen along with the answer (as stated earlier, all questions in a rule have an implicit AND relationship). In our example, the rule will fire when the question in the preconditions box is answered 'yes'. This will cause two actions to fire, one that adds a new question and another that adds a case to the current project. New questions and associated answers can be added by clicking the Edit Questions button on either the Window in Figure 3 or the Edit Preconditions window (not shown). Cases containing questions are designated in the BORE Case Manager with a green question mark icon.

Actions are added to the rule through the window to the right in Figure 3 that lists the possible action types. Once a type is chosen, the defined actions are listed and can be selected for inclusion in the rule. New actions and action types can be added by creating a link to an existing Java object with methods for executing and undoing the action.

Creating Domain Cases. Domain cases are edited in Case Manager and are treated just like any other project hierarchy in BORE. The domain librarian (someone with permissions to edit domains) selects the domain needing editing and adds, moves, deletes, and edits domain cases as needed. Domain cases are added using the buttons at the top of the Case Manager and edited with the Edit tab on the case.

Attaching Rules to Domain Cases. The questions used to choose options can be attached to domain cases by editing the Options tab in the domain case. Any questions chosen will be displayed in the Options tab when the case is created for a project.

Implementing BORE in an organization will require having personnel trained in rule-based systems. Rule systems can become nearly as complex as programs themselves, and people will need to ensure that the rules perform the appropriate actions and that rules are not inconsistent, contradictory, or incomplete. The benefits arising from these additional costs, as described later, is a formalized definition of an organization’s process that can be tailored to individual project needs, thus reducing

the burden on project personnel to become software process experts before using the organizations defined standard.

3 Examples Using BORE

Domain rules and cases are used to define software processes at any level of detail, from high-level development processes to specific corporate standards, such as which login screen should be used. The level of desired detail is left entirely to the designer of the domain. To demonstrate this versatility, we present two examples of how BORE can be used for software development activities. The first shows how BORE can be used to define a standard development methodology that can be tailored to specific project needs. The second shows how a set of parameterized templates can be used to define Oracle configuration files and SQL scripts.

3.1 A Defined, Extensible SDM

One application of BORE is to create a tailorable Standard Development Methodology (SDM) that can serve as a focal point for an organizational learning process. This approach is similar in focus to software process technology, but is less concerned with automating parts of the process [8, 15], and more concerned with work breakdowns and providing resources when they are needed. In spite of advances in software process technology, most organizations continue to document the majority of their standard development practices in monolithic SDM documents. Hypertext versions of the documents on Web-based Intranets are becoming popular, but solve none of the problems inherent with finding information and applying it to a specific software development effort.

An inherent problem with SDMs is the tendency to try to accommodate diverse development needs with an SDM that is stated at a very high level and lacks the detail that can truly guide a project. Industry standards, such as CMM and ISO do little to address this issue, and may in fact contribute to watering down processes to create universally applicable development methodologies. There also needs to be specific guidelines, examples, and detailed procedures that help people create the design. I.e., to be successful, the SDM must provide valuable resources for development efforts.

To reduce these problems, BORE can create an on-line SDM using rules to create multiple paths through the SDM. The paths can be designed to accommodate the different kinds of projects typically encountered in the organization [30]. The question/answer preconditions of BORE rules serve as a specification of the conditions under which a specific activity should be applied to a project. Options can be added to cases that allow projects to choose amongst alternative processes. The example in Figure 2 is taken from the software development standard of a large transportation company that defined two major tracks for their process, one for developing applications that will be deployed across the enterprise, and another for more localized and prototyping efforts.

One way to design this kind of SDM would be to have each project answer a series of questions at project initiation. We find this approach unappealing for a number of

reasons. First, we want BORE to support a number of different levels of decision making, from choosing whether the project is enterprise-wide or local to what kind of database should be used all the way to choosing reusable code, such as logon screens and associated logic or back-up and recovery schemes. Secondly, if a system spans these levels of support, and indeed it must to be called an organizational memory, it would be impossible to anticipate the answers to all questions at project initiation.

Therefore, we used a more iterative strategy. When a project begins, an initial set of cases for a project can be defined any of which can have options associated with it. Choosing some options may further break down an activity into constituent activities. Since any case can have options, these can implement further detailed breakdowns. For example, an initial pass may break a system into architecture-level elements, such as a user interface. Embedded in the user interface activities can be options to decide which interface widgets should be used, what choice methods are applied and etc. This multi-step process allows projects to incrementally expand each part of the project's process as they gain the knowledge necessary to address process issues [30]. In this way, the domain can be designed to tailor the overall SDM to its needs when it is able to address the issues involved.

3.2 An Extensible Wizard Using Parameterized Templates

Another knowledge intensive activity in the development process is the creation of databases to support application programs. In many development organizations, domain experts, often named Database Administrators, or DBAs for short, are called upon for this often laborious task. In this application of BORE, we designed an extensible wizard to embody DBA knowledge to create network configuration files and SQL scripts for creating tables in an Oracle database. The SQL script is a file of SQL statements. The network configuration files allows for complexities of the network, such as the Net8 network layer used by Oracle, to be hidden from applications and the database. We used BORE to generate an entry in the configuration file that resides on the client (in a client/server architecture) to allow a simple way for the client-side application to communicate with the database server.

Wizards are a convenient mechanism for directing people to relevant information when the needs are well-known. Microsoft Word provides wizards for common tasks such as creating envelopes, Web pages, legal pleadings, etc. These Wizards direct users through a set of choices, and the desired object (such as a MS Word document or a database table) is created automatically by the system.

While traditional wizards have been applied to technical domains, such as component generators for data structures [4], the static nature of Wizards needs to be addressed before the method can be used as a software development tool. For example, while Oracle provides wizards that aid in the creation of database objects, such as tables and constraints on those tables, there is no way to extend this functionality to create other database objects or embody domain-specific knowledge such as local conventions for naming and organizing tables. It would be useful if the wizard could capture knowledge specific to an organization and disseminate that knowledge. For example, a banking or loan organization may frequently develop systems that need a lender table. The wizard can then allow the user to choose the

```
?service_name?.world = (DESCRIPTION =
(ADDRESS_LIST = (ADDRESS = (PROTOCOL
= TCP)(Host = 129.93.33.120)(Port =
1521)))(CONNECT_DATA = (SID = ORCL)))
```

Figure 4: A template for creating Oracle network configuration file entry.

default lender table, rather than go through all of the generic questions about which fields, data types and sizes, and constraints are necessary for the lender table. This approach facilitates enhanced reuse of knowledge and code.

Developing an Extensible Wizard with BORE. To create a wizard for creating Oracle database tables, we designed a BORE domain that constructs working database components for a simple class of address book applications. This application domain applies knowledge from the second author's experiences with a student lending organization.

This was accomplished by creating string templates that contain variables, creating BORE rules that performs text replacement in the templates, placing the resultant strings in an appropriate file (such as network configuration file or SQL script), and using the resulting files to create database objects on an Oracle server. An example template is shown in Figure 4. It is entered through the Rule Manager as a simple text string. The template defines variables by surrounding a variable name by question marks, such as the “service_name” variable in Figure 4. Through the facility for adding new action types, a Java object was created with methods to search the text for variables and replace the string with the values of variables defined by BORE rules.

In addition to string replacement in the templates, methods were developed to create actions that communicate with the database in order to create new schemas and new tables. This application requires that the new database tables are created in an existing Oracle database. Defaults, such as standard tablespaces (logical storage areas), for this database are already established and included in BORE rules. An Oracle database server runs on the same machine as the BORE server and is accessed through system calls.

Creating the Database Tables. The BORE wizard interface is used the same way as with the SDM. That is, the user makes choices within the Options tab of the BORE case window. In addition to the simple single answer selections shown in Figure 2Figure 2 can select multiple answers or type a text answer. The latter is used in conjunction with a rule that associates a rule variable with some text, such as names and other items that cannot be predefined in the domain. In the student lending domain, the service name is example text that needs to be filled in by users.

All variables do not necessarily *have* to be manually typed by the user. For example, at the specific institution we developed this system for, all development databases resided on one machine while all production databases resided on another machine. We used a question that determined whether the new tables were to be created on the development machine or the production machine. BORE rules then determined the IP address and port number.

If the user should go back and change the answer to a question, the appropriate rules are backtracked and new rules fire. With the example of our SQL script file, backtracking a rule that created a SQL statement will comment out the statement with a timestamp. If the rule that created the file is backtracked, then the file is renamed with a timestamp. This process of renaming files and commenting out statements allows for a history of the actions to be kept.

A Feasibility Study. To assess the feasibility of this approach, we asked three test subjects to develop the database component described above for the address book application. Two subjects were experienced Oracle developers from a student loan organization with no BORE experience. The third was a BORE researcher/developer with no Oracle experience and no experience in the student lending problem domain. We asked the subjects to both build the component using whatever traditional development method they would ordinarily use, and to build the component using the BORE system.

The task involved creating a database schema with three tables (LENDER, SCHOOL, PERSON) where each table is stored in one of the standard (already existing) tablespaces, and a way for other system components to communicate with the database component.

All of the test subjects were able to build the database component using the BORE wizards in a matter of minutes. Only one of the test subjects was able to successfully create the database component using their traditional method, and it is significant to note that this database expert needed to be prompted by the experimenter to insert a database directive to complete the component. The BORE researcher/developer, as expected, was not able to create the database component without the use of BORE.

None of the test subjects were able to expand the rule base for the new tablespace requirement mentioned in the above paragraph. The issue here seems to be less on the nature of using a rule-based system, and more on the need for improvements in BORE's rule editing capabilities, a fact that has been acknowledged in the past and needs further work.

4 Related Work

The general goal of this approach is to create knowledge management tools that are more proactive in delivering information to software developers than typical repositories and search engines. One way this can be accomplished is to create a tailorable process that provides context-sensitive information to software development efforts. Another way is to create extensible wizards that anticipate desired features, allowing for extensions when unanticipated features are needed.

The BORE tool and methodology is flexible enough to split the gap between overly-restrictive development methodologies and ad-hoc software development practices to fit the needs of software development organizations as they evolve. Currently, the industry standard is to define a SDM and then ignore it in its entirety or follow it enough to justify it to certification authorities. We wish to turn these procedures and software processes in general into resources that truly supports the development process as it is actually practiced, while adding necessary degrees of

formal procedures to ensure high-quality products. This involves not only defining a process, but also using feedback from projects to refine and improve its procedures.

The concept of experience factories [1-3] has many common themes to this work, although little in the way has been done to create interfaces and CASE tools to support the concept. The QIP approach in TAME [3] framework is designed to develop and package experiences to facilitate reuse within the organization. Basili et al. advocate the use of metrics and quantifiable goals to create an improvement strategy and address controlling the content, structure and validity of the knowledge [1]. While these metrics will become necessary for an organizational learning approach to succeed, our focus and contribution thus far has been to provide a support environment for this kind of approach in a framework that allows for decisions support for choosing appropriate processes and evolution of processes to continuously improve the organization's best practices.

Thus far, most research on software processes has focused on defining the process. Approaches include high-level strategies such as the waterfall, spiral, and prototyping models, methods for combining process elements [30], and universal process models such as the CMM [31] or ISO 9000. A significant contribution of this work is to define not only the process, but how the process evolves with the changing needs of the development organization, a phenomena that process technology is only beginning to explore [8]. BORE domains are seen as a "seed" that evolves as it is used [14]. In addition, the process can be defined at many levels of detail, allowing projects to adopt the process at an appropriate level of detail. In essence, this is more of a form of knowledge editing [32] than process programming [9, 30], which is more concerned with integrating tools and documents to automate development activities, although elements of both issues are present.

This approach also has some roots in the design rationale field [7, 13, 26, 28, 29]. Similar to the organizational learning approach, the motive for capturing design rationale is to avoid repeating common pitfalls or re-discussion of design decisions by describing the alternatives debated and decisions made for a given effort. Many schemes, from the simple of Procedural Hierarchy of Issue structures [7, 13] to more complex structures designed to capture logical relationships for computation [26] have been devised. All have the same basic structure of a set of hierarchically arranged questions posed to flesh out issues. Alternatives and rationale for the alternatives can be attached to the questions to document discussion paths. BORE extends these techniques by incorporating knowledge other than design rationale and creating rule-based techniques for identifying when issues are applicable.

There are also common themes with the Designer Assistant project at AT&T, which created an organizational memory system to ensure conformance to the usage of a specific piece of complex functionality in a large switching system [33]. In this setting, the design process was modified to include a trace of the Designer Assistant session as part of a design document. The appropriateness of the designer's choices and adequacy of the advice given by Designer Assistant are discussed during software design reviews. If the advice is found to be lacking, designers begin a formal change process to update the knowledge. Utilizing a combination of existing and new organizational processes to place use of Designer Assistant into development practices ensures that the knowledge will evolve with the organization. The

observation that “technology and organizational processes are mutual, complementary resources” [33] has served as a guiding principle for this work.

5 Conclusions and Future Work

Knowledge management for software development is more than repositories and search engines. It also requires actively delivering information during the development process and ongoing process of capturing project experiences. Using such "active" knowledge management techniques can prevent the duplication of efforts, avoid repeating common mistakes, and help streamline the development process. Similar projects have shown that such a system will be used by development personnel, provided it contains relevant, useful and up-to-date information [33]. This mandates a strong tie between technology and process in which using the technology must become part of routine work activities. Such an approach will succeed to the extent that developers are rewarded in the short term for their efforts.

The contribution of this research is to develop a combination of tools and techniques that turn current development practices into *living* documents designed to evolve and improve through use, drawing on the collective knowledge of the organization.. As the repository accumulates through principled evolution of the knowledge domain, it improves to able to handle a wider range of circumstances [24], while evolving toward answers to problems that fit the organization’s technical and business context. The real question is not whether the repository is “correct” in some objective sense, but rather whether less mistakes are repeated and better solutions adopted when using the repository.

These principles were demonstrated by applying our approach to SDM processes and wizards for developing database components for address book applications. As demonstrated in our small empirical study, the knowledge embedded in these tools can be invaluable not only for novice users, but experts as well.

Future Directions. Aside from refining the normal quirks of a research prototype, a number of improvements are needed to take the next step in evaluations of BORE. It was clear from our brief study that the rule interface of BORE needs some improvements. In particular, there needs to be ways to find a rule given some behavior (such as a case being added to a project) or other attributes. Currently, users need to page through the rules one-by-one to find a rule with the desired precondition or action.

More tools are needed that support creating domain rules and ensuring that the rules are consistent and no contradictory. Visualizations and aggregate rules will prove useful, and we are looking into using agents and critics to flag knowledge base inconsistencies and gaps [32] and ensure that complete and consistent processes are assigned to projects.

The extensible wizard work was designed as a step towards applying BORE to component-based software engineering environments. We are currently working on using BORE to select and configure components in the JavaBeans platform to create software applications using the extensible wizards described here.

Early BORE evaluations indicate progress has been made on our overall goals of creating a medium for flexible software process definitions and turning SDM documents into a resource that supports the development process as it is actually practiced [20, 21].

The BORE system is beginning to move out of the prototyping phase and become a production-quality system, which will enable evaluations in realistic software development settings. Within the coming months, we hope to deploy it in a few development organizations, including NASA Goddard [18], some local small organizations, and the Software Design Studio at the University of Nebraska-Lincoln [12]. Through these evaluations, we hope to gain an increased understanding in the overall approach and how knowledge-based tools such as BORE can be used to improve software development practices.

Acknowledgements. We gratefully acknowledge the efforts a number of graduate students that have helped develop BORE, particularly Kurt Baumgarten, Michelle Conway, and Roger Van Andel. This research was funded by the National Science Foundation (CCR-9502461 and CCR-9988540).

References

1. V. R. Basili, G. Caldiera, and G. Cantone, "A Reference Architecture for the Component Factory," *ACM Transactions on Software Engineering and Methodology*, vol. 1, pp. 53-80, 1992.
2. V. R. Basili and H. D. Rombach, "Support for Comprehensive Reuse," *Software Engineering Journal*, pp. 303-316, 1991.
3. V. R. Basili and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, vol. 14, pp. 758-773, 1988.
4. D. Batory, G. Chen, E. Robertson, and T. Wang, "Design Wizards and Visual Programming Environments for GenVoca Generators," *Transactions on Software Engineering*, vol. 26, pp. 441-452, 2000.
5. N. Belkin, "Helping People Find What They Don't Know," *Comm. of the ACM*, vol. 43, pp. 58-61, 2000.
6. F. P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, vol. 20, pp. 10-19, 1987.
7. E. J. Conklin and K. Yakemovic, "A Process-Oriented Approach to Design Rationale," *Human-Computer Interaction*, vol. 6, pp. 357-391, 1991.
8. G. Cugola, "Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models," *IEEE Transactions on Software Engineering*, vol. 24, pp. 982-1000, 1998.
9. B. Curtis, M. I. Kellner, and J. Over, "Process Modeling," *Communications of the ACM*, vol. 35, pp. 75-90, 1992.
10. B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, vol. 31, pp. 1268-1287, 1988.
11. P. Devanbu, R. J. Brachman, P. G. Selfridge, and B. W. Ballard, "LaSSIE: A Knowledge-Based Software Information System," *Communications of the ACM*, vol. 34, pp. 34-49, 1991.
12. S. Dunbar, S. Goddard, S. Henninger, and S. Elbaum, "Bootstrapping the Software Design Studio," *Fifth Annual National Collegiate Inventors and Innovators Alliance Naitonal Conference*, Washington, DC, pp. 180-188, 2001.

13. G. Fischer, J. Grudin, A. Lemke, R. McCall, J. Ostwald, B. Reeves, and F. Shipman, "Supporting Indirect Collaborative Design With Integrated Knowledge-Based Design Environments," *Human-Computer Interaction*, vol. 7, pp. 281-314, 1992.
14. G. Fischer, R. McCall, J. Ostwald, B. Reeves, and F. Shipman, "Seeding, Evolutionary Growth and Reseeding: Supporting the Incremental Development of Design Environments," *Proc. Human Factors in Computing Systems (CHI '94)*, Boston, MA, pp. 292-298, 1994.
15. V. Gruhn and J. Urbainczyk, "Software Process Modeling and Enactmant: An Experience Report Related to Problem Tracking in an Industrial Project," *Proc. 20th International Conference on Software Engineering*, pp. 13-21, 1998.
16. S. Henninger, "Capturing and Formalizing Best Practices in a Software Development Organization," *The Ninth International Conference on Software Engineering and Knowledge Engineering (SEKE '97)*, Madrid, Spain, 1997.
17. S. Henninger, "Case-Based Knowledge Management Tools for Software Development," *Journal of Automated Software Engineering*, vol. 4, pp. 319-340, 1997.
18. S. Henninger, "Software Process as a Means to Support Learning Software Organizations," *Twenty-fifth Annual NASA Software Engineering Workshop*, Greenbelt, MD, 2000.
19. S. Henninger, "Supporting Software Development with Organizational Memory Tools," *International Journal of Applied Software Technology*, vol. 2, pp. 61-84, 1996.
20. S. Henninger, "Tools Supporting the Creation and Evolution of Software Development Knowledge," *Proceedings of the Automated Software Engineering Conference*, Lake Tahoe, NV, pp. 46-53, 1997.
21. S. Henninger, "Using Software Process to Support Learning Software Organizations," *1st International Workshop on Learning Software Organizations (LSO 1999)*, Kaiserlautern, FRG, 1999.
22. S. Henninger, K. Lappala, and A. Raghavendran, "An Organizational Learning Approach to Domain Analysis," *17th International Conference on Software Engineering*, Seattle, WA, pp. 95-104, 1995.
23. C. C. Huff, "Elements of a Realistic CASE Tool Adoption Budget," *Communications of the ACM*, vol. 35, pp. 45-54, 1992.
24. J. L. Kolodner, *Case-Based Reasoning*: Morgan-Kaufman, San Mateo, CA, 1993.
25. J. L. Kolodner, "Improving Human Decision Making through Case-Based Decision Aiding," *AI Magazine*, vol. 12, pp. 52-68, 1991.
26. J. Lee, "Design Rationale Capture and Use," *AI Magazine*, vol. 14, pp. 24-26, 1993.
27. M. Lindvall and I. Rus, "Process Diversity in Software Development," *IEEE Software*, vol. 17, pp. 14-18, 2000.
28. A. Maclean, V. Bellotti, R. Young, and T. Moran, "Questions, Options, and Criteria: Elements of Design Space Analysis," *Human-Computer Interaction*, vol. 6, pp. 201-251, 1991.
29. T. Moran and J. Carroll, "Design Rationale: Concepts, Techniques, and Use,". Hillsdale, NJ: Lawrence Erlbaum Associates, 1996.
30. L. Osterweil, "Software Processes are Software Too," *Ninth International Conference on Software Engineering*, Monterey, CA, pp. 2-13, 1987.
31. M. C. Paulk, B. Curtis, M. Chrissis, and C. V. Weber, "Capability Maturity Model, Version 1.1," *IEEE Software*, vol. 10, pp. 18-27, 1993.
32. L. Terveen and D. Wroblewski, "A Collaborative Interface for Browsing and Editing Large Knowledge Bases," *National Conference of the American Association for AI*, Boston, MA, pp. 491-496, 1990.
33. L. G. Terveen, P. G. Selfridge, and M. D. Long, "Living Design Memory' - Framework, Implementation, Lessons Learned," *Human-Computer Interaction*, vol. 10, pp. 1-37, 1995.