Tools Supporting the Creation and Evolution of Software Development Knowledge

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

Abstract

Software development is a knowledge-intensive activity involving the integration of diverse knowledge sources that undergo constant change. The volatility of knowledge in software development requires that knowledge bases are able to support a continuous knowledge acquisition process where tools are available that can make use of partial knowledge. To address these issues, case-based technology is used in combination with an organizational learning process to create an approach that turns Standard Development Methodologies (SDM) into living documents that capture project experiences and emerging requirements as they are encountered in an organization. A rule-based system is used to tailor the SDM to meet the characteristics of individual projects and provide relevant development knowledge throughout the development lifecycle.

The Incompleteness of Knowledge Bases for Software Development

The use of intelligent assistants for software design environments has a long and diverse history. While the focus on automating the design process dominates much of the work, interest in tools that assist the process has risen [7, 17]. The AI focus for these systems is to use "knowledge" in some form to reason about software designs and provide some form of assistance to the developer. For example, the Programmers Apprentice [17] assumes that knowledge about programming techniques and combinations of concepts in the form of *clichés* can be used to synthesize and analyze programs. Other approaches encode rules in the form of *critics* which are then used to notify users of inconsistent, suboptimal, or errant design decisions are made [8]. These and other systems are important because they provide an indication of the *kinds* of knowledge that would be useful for intelligent assistance for software system design.

The systems used to demonstrate these techniques typically focus their discussion on how the system can reason with a particular form of knowledge. Given that good knowledge-based methods can be shown to work, the next, and perhaps more difficult task, is to put the necessary knowledge into a repository. When scaling these techniques to large, complex, systems, the problem of getting knowledge into the system becomes critical. Besides scale, software engineering domains are particularly volatile, evolving as technology changes and as the developers understand and discover new user requirements [6]. Because of this volatility, knowledge bases will always be incomplete, lacking knowledge either because knowledge engineers do not elicit everything the domain expert knows or because the knowledge cannot keep pace with evolving domain knowledge.

The inherent incompleteness of knowledge bases for software engineering lead to two scarcely explored issues. First, techniques are needed that support designers in the absence of complete domain knowledge. Second, knowledge acquisition needs to be supported throughout the design process to reflect the evolving understanding of the system and its domain. This paper explores these issues through an *organizational learning* process coupled with technology that collects knowledge from practitioners in a manner that can serve as the basis for increasingly formal levels of domain knowledge. Case-based techniques are used to evolve the repository and turn the emerging, knowledge into standards representing a software development organization's best practices. In its canonical form, the process is simple: A Standard Development Methodology (SDM) is adopted and defined through a set of process elements. Projects tailor the SDM as necessary, using the resulting set of documents as a central repository for the project. Project experiences are then used to evolve the repository to meet new software development needs. The key issues addressed in this paper are to 1) provide support for tailoring the SDM and 2) modify the SDM to capture the organization's best practices in such a way that valuable resources are provided for software development teams.

An Organizational Learning Approach to Software Development

An organizational learning approach to software development captures project-related information during the creation of individual software products [12]. This information can then be disseminated to subsequent projects to provide experience-based knowledge of development issues encountered at the organization. Given a repository of experiences with tools and methods, new projects can match their characteristics to the characteristics of existing solutions through various retrieval mechanisms [10-12]. Information in the repository not only points to re-usable solutions and code, but also suggest how novel problems can be approached by adapting similar problems and providing "lessons learned" or tips and techniques for accomplishing specific tasks. The development process in general can be refined and streamlined by identifying commonly occurring patterns, thereby reducing the amount of duplicated effort and learning from the mistakes and successes of projects with similar characteristics. The emphasis becomes one of systematically institutionalizing knowledge as it is generated by people in the organization.

As the cases accumulate in the organization-wide repository, the knowledge contained in the repository becomes increasingly tailored to the kinds of design problems that frequently occur in the organization. The repository therefore serves not only as a means to disseminate design knowledge, but also helps an organization *learn* what does and does not work for their development context. This is where we distinguish between organizational *memory* systems that many have advocated [20, 21] and our notion of organizational *learning*, where the emphasis is placed on learning from previous experiences, which has also been applied to the organizational level as "learning organizations [18]. The method also naturally incorporates an evolutionary, continuous, process of improvement that evolves with the ever-changing development context.

Early work with the Information Technology division of Union Pacific Railroad helped identify the organizational learning approach [10-12]. This organization, which develops in-house applications for the railroad, has many medium to small-scale projects developing client-server and legacy systems. The sheer number of projects and their diverse requirements makes it difficult to set organization-wide standards and creates barriers to reusing software development knowledge. While many initiatives within the organization had emerged to address the need to capture and disseminate information, a strategy was needed to tie these efforts together.

This approach has been demonstrated in an exploratory prototype, named BORE (Building an Organizational Repository of Experiences) [11]. In BORE, cases represent project issues and contain information about how issues were resolved and the resources used to resolve the issues [9]. Cases are stored in a repository that can be searched and browsed to find issues with similar characteristics [10-12]. Although refinement and evolutionary processes have always been part of our vision [10], BORE and its predecessors have been concerned with creating a repository of project experiences. This is valuable in and of itself, but emphasis must be placed on establishing continuous improvement processes that improve product quality and developer productivity, while recognizing past experiences as a catalyst for the learning process.

While it is desirable to systematically institutionalize knowledge as it is generated by people in the organization, recent experiences with BORE have indicated that this must be balanced with procedures that control the content, structure and validity of the knowledge.

Evaluation of BORE Prototypes

Early BORE prototypes were evaluated in two separate contexts. In the first, a pilot project at UPRR evaluated BORE by documenting some cases and providing comments of how BORE can be improved to better meet their specific needs. The second evaluation context was a Software Engineering course consisting of seniors and first-year graduate students. Students in this class developed small to medium scale software systems for clients external to the university system (one of the projects worked with UPRR). Project assignments for requirements, design, implementation, and formal testing all required the use of BORE. The scope of the projects were rather small, with less than a semester to develop the software and 4-6 people on the project. Nonetheless, the six projects generated over 150 cases.

In both instances we were disappointed in the detail and amount of information provided by BORE users. Cases were not described adequately, solutions were often left blank, and seldom contained adequate information for subsequent users to re-use the knowledge. There was clearly a need to encourage users to provide higher quality information. Our observations at UPRR indicated that the kind of documentation we desired was indeed collected, albeit in a number of separate repositories in Lotus Notes [12]. It became clear to us that BORE alone was not enough to ensure the success of an organizational learning approach. While it can be argued that more disciplined settings would produce the desired results, procedures are needed that help people take maximum advantage of the organizational learning approach.

Capturing and Using Best Practices in an Organizational Setting

Through these studies, a set of requirements began to emerge that refined the organizational learning approach to software development. People needed guidelines on how to use BORE and what kinds of information they needed to capture. In addition, the information needed to be refined into something that could readily be re-used by subsequent development efforts. Use of the organizational learning system needed to be *formalized* in the sense that its use is mandated in a documented process to ensure that proper information was collected. In addition, it has become clear that for such a process and tool combination to work, the resulting repository must become the central planning and documentation repository for a project. This would both avoid duplication of effort and provide one place for projects to coordinate and document their work.

These requirements point to a dynamic technology for Standard Development Methodology (SDM) documents that can serve as a focal point for an organizational learning process. The complicating problem with using an organization-wide SDM is that diverse application requirements and rapidly changing technologies make it profoundly difficult to mandate a process that is useful and not overly restrictive [5, 15]. A flexible mechanism is needed where the SDM is turned into a living document designed to evolve and improve through use.

Towards A Defined, Extensible, SDM

Most traditional SDMs created by software organizations eventually fall into disuse for two reasons: 1) To accommodate diverse needs, the SDM is stated at a very high level and lacks the detail that can truly guide a project. Stating the need for "detailed design" is insufficient. 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. 2) The SDM quickly falls behind the rapid pace of technology and business needs. Even if SDM curators are appointed, it is difficult for a few people to keep pace with changes in large development organizations.

To reduce these problems and accomplish the goal of creating a formally defined SDM that can be systematically extended to meet diverse project needs, we have combined the organizational learning tools of BORE with a process that uses and controls the evolution of SDM documents. The process begins with a defined on-line SDM. In addition, multiple paths through the SDM are designed to accommodate the different kinds of projects typically encountered in the organization [15]. Figure 1 shows a diagram of the overall process of using and evolving the SDM. In this figure, the SDM (shown in the box marked "On-Line SDM" in the BORE repository) has three different paths. More complex structures are possible, and it is not expected that everyone should understand the SDM in its entirety. Instead, a decision support system is used to reduce the complexity by eliciting project characteristics, such as end-user, data and security requirements, whether an iterative or waterfall methodology should be used, etc.

The decision support system steps through questions designed to elicit project characteristics. These characteristics are matched against requirements in the SDM to propose a customized process for the project. Developers can then modify this process to meet the project's needs. The end result is subjected to an approval review in which the answers provided to the decision support system and any modifications made to the resulting process are carefully examined. The outcome of the review can be one of three actions: 1) the project is required to re-visit some of the questions posed by the decision support system, 2) modifications are rejected and/or new modifications are proposed, and 3) the process is approved.

Upon approval of the process, a BORE project is created and new cases are created for each process element. These cases are associated with SDM process elements and can be used as an information resource for that element. In addition, any exceptions to the SDM, such as adding new process elements, must become part of the SDM. This means modifying the SDM and the decision support system so that projects with similar characteristics are guided toward the SDM modifications made to accommodate the project. The overall process is designed to retain a degree of process rigor while being able to support a dynamic development context.

Over time, the SDM will evolve to meet the needs of issues that recur in the organization [9], positioning the organization to take advantage of previous experiences and improve. But, particularly in the face of diverse applications and interface design requirements, this process can create a large and potentially cumbersome repository.[1] Its complexity will exceed any individual's ability to understand all the issues involved in the guideline repository. Appropriateness is ensured through the conformance and modification, but without proper technology, the organization has no hope of ensuring that the guidelines lead to quality applications.

[1] Note that there is a metaphor with American Case Law which, while sometimes maligned for being cumbersome, has stood the test of time under the constantly evolving forces of human culture and diverse circumstances. While some of these problems are inherent to human activities, which become embodied in the interfaces and inner workings of our systems, we hope to create technology that can lead to effective and timely decision making.

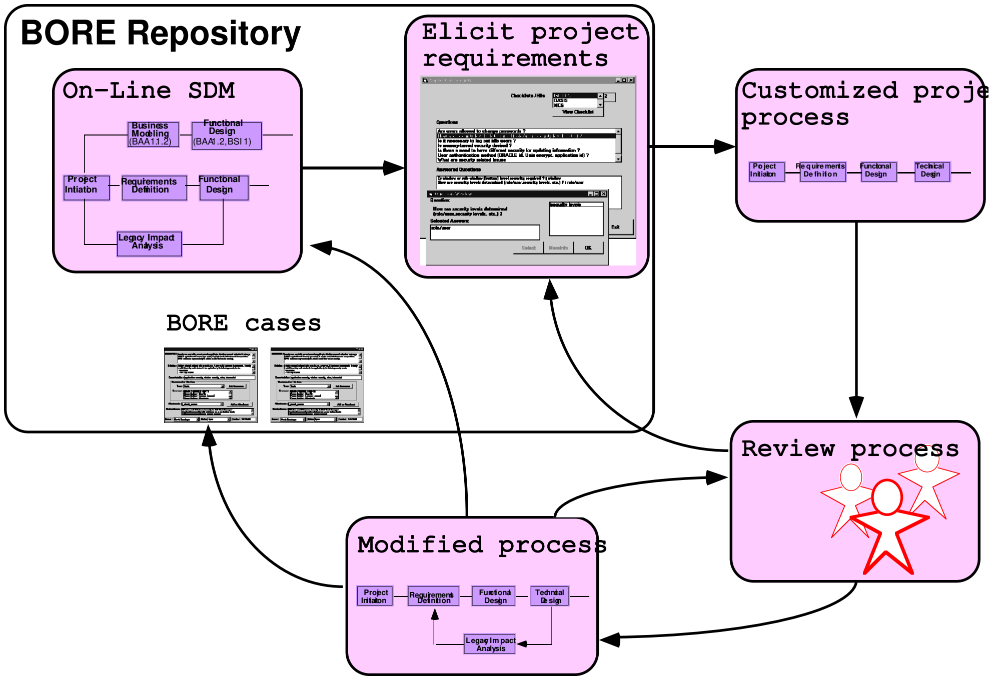

Figure 1: Tailoring and Refining the SDM.

Tools for Supporting SDM Tailoring and Modification

The process outlined in the previous section is designed to turn an organization's SDM into a living document that evolves and improves with use. The following sections use BORE to demonstrate how this is accomplished. There are two major components of this process: 1) The standard SDM is *tailored* to specific project requirements, and 2) if necessary, the SDM is *modified* to meet the emerging needs of development projects.

Tailoring the SDM to Specific Projects

The general structure of a process in our view of an SDM consists of a set of process elements connected by relationships. The general SDM is tailored to specific projects through decision trees such as the one shown in Figure 3 that describes the criteria for choosing a database back-end. These decision trees can be used at an arbitrary level of project detail, from high-level development processes such as whether business modeling or requirements definition is needed (see the 'On-Line SDM' box in Figure 1) to specific corporate standards, such as which login screen should be used.

When creating a new project, BORE assigns a number of cases to a project that represents the tasks that a project is required to perform when initiating a project. An example of the resulting case hierarchy is shown in Figure 2. The actual cases used in this process can easily be changed

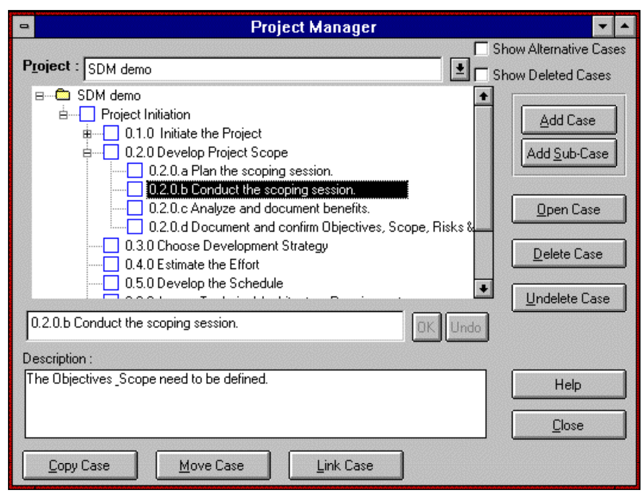

Figure 2: The Project Manager view of the Project Initiation Process

through the modification process (see the next section). Cases are placed in a hierarchy that serves as a checklist of project tasks. BORE supports a color coding of the glyph to the left of the case name to provide a visual means of identifying the status of cases -- whether the case is open, actively being worked on, resolved, and etc.

The purpose of the case is to document how a given task was addressed by the project. The Overview, Resources, Attachments, and Related Cases tabs can be chosen to reveal information about the issues involved in addressing the task. The Question tab, which is expanded in the

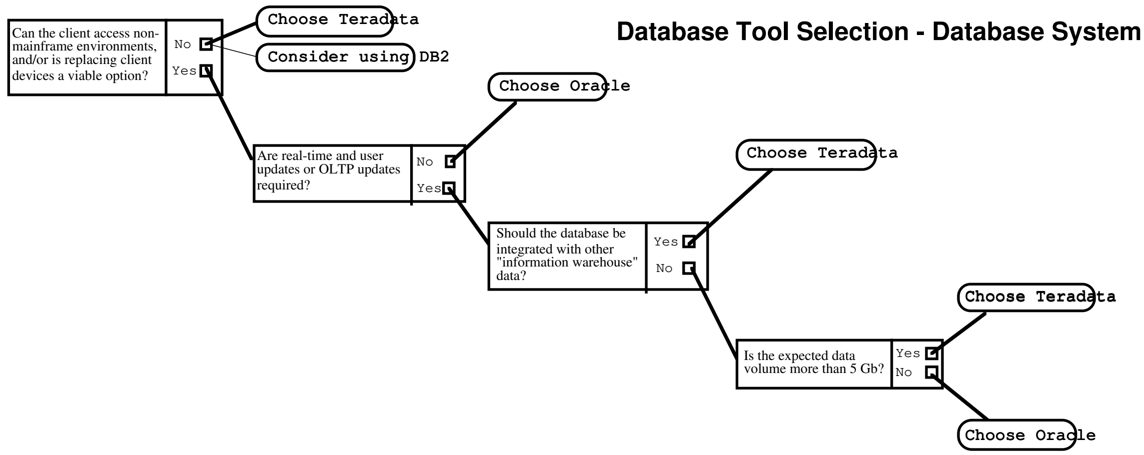

Figure 3: A Decision Tree for Choosing Database Technology.

figure, is used to elicit information about a project's characteristics to determine which SDM process elements should be assigned as project tasks. In other words, the questions step users through a decision tree such as the one in Figure 3, resulting in an context appropriate task to be added to the project's to-do list.

Because it is possible to identify all of a project's characteristics at project initiation, any BORE case can contain questions that may add tasks to a project. Therefore, a multi-step process is used to create a tailored process. For example, suppose the development process is divided into a number of tracks depending on whether the project is a mission critical enterprise-wide application, a local application, or a research prototype. One of the project initiation processes will then step the user through a number of questions designed to determine which of these tracks is appropriate for the project. Suppose that on an initial pass a task named Business Modeling is added to the project. When work begins on this task, it can be broken down into subtasks through another decision tree. This allows to iteratively expand each part of the project's process as they gain the knowledge necessary to address process issues [15]. In this way, the project tailors the overall SDM to its needs as it is able to address the issues involved.

The Questions tab displays questions that can be answered in any order. Questions that must be answered are written in red font, the rest are optional. The 'Decision Resources' button displays another window with ancillary information about the chosen question, such as current SDM policy statements, to help the user decide which answer is right for the project. The 'Cases' button provides a list of other projects that have answered this question. By clicking this button and perusing the cases, users can view how similar projects have answered the question, and what their results were.

Questions are represented in a rule-based production system. Each rule has a set of preconditions consisting of one or more question/answer pairs. When all preconditions evaluate to true, the rule "fires," causing a set of consequents to be executed. Consequents in BORE can take one of four forms: 1) A question can be removed from the question stack (displayed in the Question window of a case), 2) a question can be added to the question stack, 3) a task can be added to a project, and 4) a function call can be evaluated. The production system is implemented using a MS Access database for a back-end.

Modifying the SDM

As shown in Figure 1, the SDM 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 all cases for a given process element. In addition to ensuring a measure of conformance to the SDM, this provides an opportunity to disseminate best practices. Developers assigned a given process element can easily check the repository to see cases documenting how other projects accomplished the process goals and potentially reuse their solutions.

The second modification to the SDM changes to SDM tasks and the production system that assign SDM tasks to individual projects. This involves creating new decision points and process elements in the SDM and integrating questions and/or answers to questions into BORE. Rules are edited through the interface shown in Figure 5 which displays preconditions and consequents for a given rule. Tasks are added to the SDM Schema through the Project Manager (see Figure 2).

Therefore, modifying the SDM consists of either or both adding tasks to the SDM and editing the rules for adding

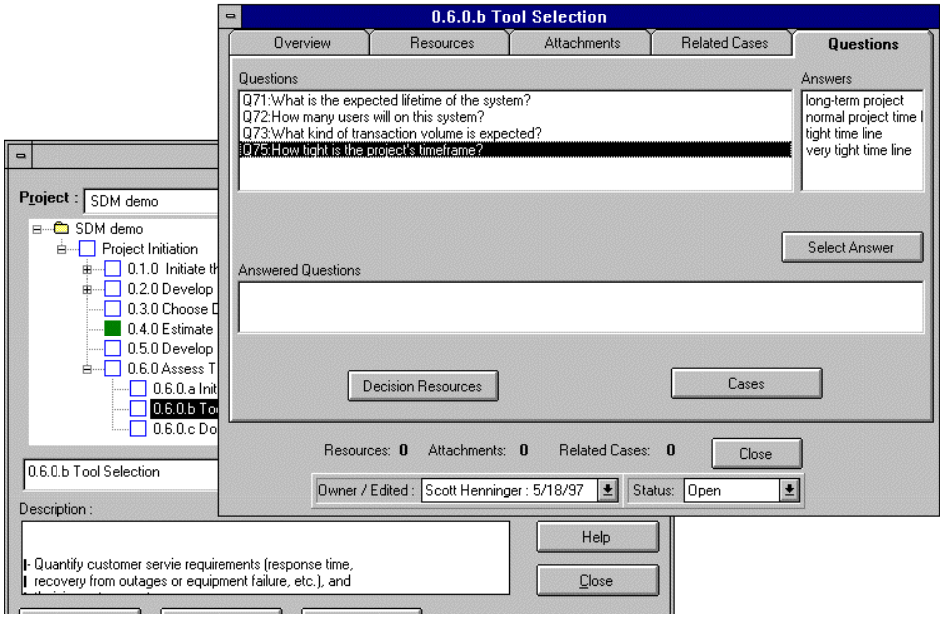

Figure 4: A BORE Case Showing Questions for Tailoring the SDM to an Individual Project.

tasks to a project. For example, suppose a project is amongst the first to create Java applets, and designs a method to use JDBC to access Oracle tables. To do this means deviating from the SDM because certain issues of creating a Web-based database server haven't been addressed before in this organization. The project goes through the review process (see Figure 1), is approved, and engages in the extra documentation mandated to pave a path for this kind of project. The project creates a number to tasks and subtasks that must now be added to the SDM. A process expert is assigned to add these tasks and insert new questions into the repository that ensures that future Web-based project can take advantage of, and improve on, the procedures created by this ground breaking project.

Related Research

The general goal of this approach is to try to split the gap between overly-restrictive development methodologies and ad-hoc software development practices. 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 SDMs into a resource, something 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 using the SDM to refine and improve its procedures. These concepts have their predecessors, particularly in the form of software factories [2-4] and process programming [5, 15], but little in the way has been done to create interfaces and CASE tools to support these concepts. This work attempts to fill this gap with the organizational learning perspective, case-based decision support, and the BORE prototype.

Thus far, most research on development processes have focused on defining the process. Approaches include high-level strategies such as the waterfall, spiral, and prototyping models, methods for combining process elements [15], and universal process models such as the CMM [16] 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. The SDM is seen as a "seed" that evolves as it is used [8]. 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 a form of process programming [5, 15], or knowledge editing [19], and we are looking into how this research can be applied to create tools for integrating new processes into the SDM and ensuring process integrity as the SDM evolves.

This approach fills two important gaps in fulfilling the Level 2 CMM requirement for a defined process [16]. The first is that while the CMM calls for tailoring the defined process to individual projects, no guidance or procedures are provided on how this should be accomplished. Our approach formalizes the tailoring process through BORE question/answer sessions and a review process to look over deviations to the SDM. The second gap is the need for refining, evolving, and improving the development process. No "Level 5" process is assumed. Rather, feedback from actual projects is used to define the best

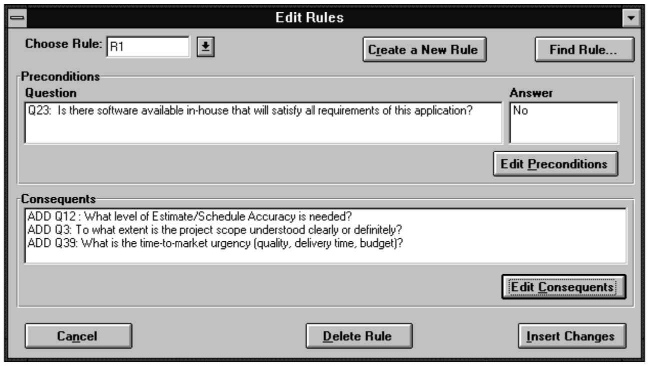

Figure 5: Interface for Editing Rules.

known methods within the development context and application requirements of a specific development organization. In addition, instead of stating that "Configuration Management" is a required process, this approach provides resources in the form of information on how other development efforts have addressed the process.

The approach also incorporates elements of organizational memory and a formalized approach to supporting a learning organization with information technology. The Designer Assistant project at AT&T created an organizational memory system whose use became part of the design review process as a way of ensuring conformance to the usage of a specific piece of complex functionality in a large switching system [20]. 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" [20] has served as a guiding principle for this work.

The production system for ordering process elements is similar to branching networks of multiple-choice questions [1] and decision trees [20]. Checklists have also been used to provide information about step that need to be followed to achieve a certain task [14]. The Project Manager (Figure 2a) can be seen as a kind of checklist that enumerates the different things that must be accomplished at each step in the development process.

Open Issues and Future Directions

The general goal of this approach is to try to split the gap between overly-restrictive procedures and not following a defined process. 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 SDMs into something 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 using the SDM to refine and improve its procedures.

One of the greatest dangers of this approach is that the resulting SDM will become unwieldy and incomprehensible. It remains an empirical question as to whether BORE's production system approach is sufficient to help developers make decisions about which process elements are necessary and how the SDM should be followed. There is a need for higher-level views and models to help people understand the SDM's structure and support a principled evolution of the processes it contains. Nonetheless, the complexity of the resulting SDM reflects the true diversity that lie in the intersection of business needs, development tools, and developer skill sets that defines modern software development. Without this, the process will settle at the ineffective least common denominator that current SDMs suffer from.

More tools are needed that help SDM administrators insert new process and the associated question/answer pairs into the SDM. This is essentially a knowledge acquisition problem, and we are looking into using agents and critics to flag knowledge base inconsistencies and gaps [19].

Some of the research questions we will continue to explore with BORE include designing work practices centered around case-based organizational learning processes. Design tools must reward users for use in the current project as well as leaving information for subsequent development and maintenance efforts. We believe these are compatible goals, as long-term projects need to track their progress and decision making process. Some major research questions include: designing methodologies based on standards set by previous projects yet flexible enough to accommodate changes in business needs and technology; creating a development process that begins by using the resources supplied by the repository and updates the repository as the project progresses; finding the "right" level of documentation that doesn't overwhelm developers yet leaves a trace for subsequent efforts; and organizational changes needed to make such techniques work in the organization (i.e. the technology transfer and tool adoption problems).

Conclusions

Centering the information around a single repository through an ongoing process of capturing project experiences 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 [20]. 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 people are rewarded in the short term for their efforts.

To achieve these goals, we have coupled process and technology to turn SDM documents into dynamic "living" resources that can be extended and improved as new project needs and application requirements emerge. As the repository accumulates through principled evolution of the SDM, it improves to able to handle a wider range of circumstances [13], 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.

Acknowledgments. I gratefully acknowledge the efforts a number of graduate students that have helped develop BORE, including Kurt Baumgarten, Shawn Clymer, Michelle Conway, and Roger Van Andel. This research was funded by the National Science Foundation (CCR-9502461), Union Pacific Railroad, and the Applied Information Management Institute in Omaha.

References

[1] M. S. Ackerman and T. W. Malone, "Answer Garden: A tool for growing organizational memory," *Proc. of Office Information Systems*, 1990, pp. 31-39,

[2] V. R. Basili, G. Caldiera, and G. Cantone, "A Reference Architecture for the Component Factory," *ACM Transactions on Software Engineering and Methodology*, vol. 1, 1992, pp. 53-80

[3] V. R. Basili and H. D. Rombach, "Support for Comprehensive Reuse," *Software Engineering Journal*, 1991, pp. 303-316

[4] V. R. Basili and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, vol. 14, 1988, pp. 758-773

[5] B. Curtis, M. I. Kellner, and J. Over, "Process Modeling," *Communications of the ACM*, vol. 35, 1992, pp. 75-90

[6] B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, vol. 31, 1988, pp. 1268-1287

[7] G. Fischer, A. Girgensohn, K. Nakakoji, and D. Redmiles, "Supporting Software Designers with Integrated, Domain-Oriented Design Environments," *IEEE Transactions on Software Engineering* vol. 18, 1992, pp. 511-522

[8] G. Fischer, R. McCall, J. Ostwald, B. Reeves, and F. Shipman, "Seeding, Evolutionary Growth and Reseeding: Supporting the Incremental Development of Design Environments," *Proc, of Computer-Human Interaction (CHI '94)*, Boston, MA, 1994, pp. 292-298,

[9] S. Henninger, "Accelerating the Successful Reuse of Problem Solving Knowledge Through the Domain Lifecycle," *Fourth International Conference on Software Reuse*, Orlando, FL, 1996, pp. 124-133,

[10] S. Henninger, "Building an Organization-Specific Infrastructure to Support CASE Tools," *Journal of Automated Software Engineering*, vol. 3, 1996, pp. 239-259

[11] S. Henninger, "Case-Based Knowledge Management Tools for Software Development," *Journal of Automated Software Engineering*, 1997, (in press).

[12] S. Henninger, K. Lappala, and A. Raghavendran, "An Organizational Learning Approach to Domain Analysis," *Seventeenth International Conference on Software Engineering*, Seattle, WA, 1995, pp. 95-104,

[13] J. L. Kolodner, *Case-Based Reasoning*: Morgan-Kaufman, San Mateo, CA, 1993.

[14] A. C. Lemke, *Design Environments for High-Functionality Computer Systems*, Department of Computer Science, University of Colorado, Boulder, CO, 1989.

[15] L. Osterweil, "Software Processes are Software Too," *Ninth International Conference on Software Engineering*, Monterey, TX, 1987, pp. 2-13,

[16] M. C. Paulk, B. Curtis, M. Chrissis, and C. V. Weber, "Capability Maturity Model, Version 1.1," *IEEE Software*, vol. 10, 1993, pp. 18-27

[17] C. H. Rich and R. C. Waters, *The Programmer's Apprentice*. Reading, MA: Addison-Wesley, 1990.

[18] P. Senge, *The Fifth Discipline: The Art and Practice of the Learning Organization*. New York: Currency Doubleday, 1990.

[19] 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, 1990, pp. 491-496,

[20] L. G. Terveen, P. G. Selfridge, and M. D. Long, "'Living Design Memory' - Framework, Implementation, Lessons Learned," *Human-Computer Interaction*, 10, 1995, pp. 1-37

[21] J. P. Walsh and G. R. Ungson, "Organizational Memory," *Academy of Management Review*, vol. 16, 1991, pp. 57-91