Presented at Twenty-fifth Annual NASA
Software Engineering Workshop

Software Process as a Means to Support Learning Software Organizations

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

ABSTRACT

Software process improvement efforts are faced with a number of difficulties. The first is defining a uniform process in the face of diverse software development needs. Solutions to this problem often involve specifying the process at an abstract level, causing secondary problems, such as methodologies that are too watered down to be useful and the difficulty of obtaining feedback from development efforts. In this paper, an approach is presented that couples rule-based technology and a case-based architecture to capture detailed information and support diverse process definitions. Emphasis is placed not only on tailoring software processes to meet the needs of specific development efforts, but to also deliver information on tools, techniques, and existing artifacts throughout the process. Efforts are currently underway to investigate applying this approach to the ISC Software Methodology at NASA Goddard.

1 Diversity in Software Development Processes

Software development is a complex, knowledge intensive activity involving many types of knowledge [Henninger et al. 1995]. Contrary to the traditional emphasis on applying universally applicable techniques, much of this knowledge is locally defined and practiced. Industry standards, such as ISO 9000 and the Capability Maturity Model are necessary to begin creating standard processes that can be measured and evaluated. But the universal nature of these standards naturally migrate toward general and highly abstract procedures and techniques. Specifying, for example, that configuration management practices must be used covers a very wide range of techniques, and do little to help people understand how configuration management should be practiced for creating client-server vs. single machine applications or how it should be applied to distributed development teams.

The general problem facing these standards is the diversity of modern software development, which must accommodate increasingly diverse application domains reaching nearly every aspect of human life. Even at the most abstract levels of software process, diverse practices have evolved to address the different concerns of industries and organizations involved in creating various kinds of software [Lindvall, Rus 2000].

Most standards, such as the CMM, restrict themselves to defining the kinds of activities that should be practiced, leaving the actual definition of the activities to the organization. This allows local knowledge and practices to have influence over the standards activities. But diversity within organizations [Johnson, Brodman 2000; Schultz et al. 2000] soon becomes complex, causing the procedures to be defined at an abstract level that once again impedes the ability to specify actual practices. Development efforts are left to "tailor" the process to project needs, usually with little or no guidelines on the parameters for acceptable variations or how the tailoring should be accomplished.

Methodologies, tools, and techniques are needed that can accommodate this natural diversity while retaining a level of discipline and standard. In addition, process standards should be used to disseminate best practices that can be adopted and used to guide individual development projects.

1.1 An Extensible Software Process

The approach described in this paper begins with the following guiding principles:

1) Software processes must be flexible enough to not interfere with adept local practices while incorporating enough discipline to ensure acceptable project performance.
2) High-level process standards must be supported with specific activities and examples representing local development practices.
3) Software development efforts must be provided with activities that are as specific to the project's needs as possible.
4) Practices must be allowed to evolve in a disciplined manner to reflect changing software needs and technology advances.

These principles are addressed through a methodology and tool support that captures project experiences within a software process framework. This information is then disseminated to subsequent development efforts to provide experience-based knowledge of development issues encountered within the organization. Deviations from the standard process are used to identify where the process needs refinement or additional options based on project requirements or new technologies.

The overall approach is referred to as an "organizational learning" [Levitt, March 1988; Senge 1990] process because a formally defined process is used as a baseline to assess the adequacy of the process. As feedback from project experiences is encountered, new procedures are created to address issues inadequately handled by the current process.

As cases accumulate and deviations refine the process, the knowledge contained in the repository becomes increasingly tailored to the kinds of design problems that frequently occur in the organization. This information can be analyzed and generalized in a domain analysis [Arango 1989] or software factory [Basili et al. 1992] setting to create generalized knowledge with broader applicability than the context-specific cases. This further refines the process while assuring that it evolves to fit the needs of the organization.

2 Building an Organizational Repository of Experiences

Over the past few years, we have been investigating how tools can be used to support an organizational learning approach for software development. Through evaluation of these efforts [Henninger 1996, 1997; Henninger et al. 1995], we have come to the fairly obvious conclusion that the problem is not one of providing better search engines to disseminate static text or placing standards on the Web, but one of designing work practices to fit the organizational learning approach.

This led us to begin investigating how software process can be used as a driving force for collecting and disseminating software development knowledge. In this regard, we have combined an organizational learning meta-process with a rule-based software process engine to create a tool that provides decision support for tailoring software processes to the needs of individual development efforts. The approach is demonstrated through an exploratory prototype named BORE (Building an Organizational Repository of Experiences) [Henninger 1997], coupled with a methodology to support tailoring the software process to the characteristics of individual development efforts and refining the process when projects need to deviate from current practices.

2.1 Defining a Flexible Software Process

The methodology, depicted in Figure 1, uses a rule-based system to apply current knowledge to software development efforts and a review process to learn and extend from project experiences. Software development efforts tailor an existing development process, depicted by the "Standard Process" in Figure 1, to their current needs through choices on project options. The result is a process that is tailored to the individual needs of the project.

This process is then subjected to a review in which choices are evaluated and discussed. Where the current process fails to provide sufficient guidance, alternatives are discussed and adopted by the project. Once the process is enacted, the repository is modified in two ways. First, each process is assigned to a project as a case, meaning that the project is free to adopt the process to their own needs and document their progress. For example, a process stating that the selection criteria for choosing COTS software needs to be determined would have cases for each project stating the specific criteria used by the project. In this way, projects can easily adopt an existing process by choosing the case most closely matching their needs and refining it appropriately.

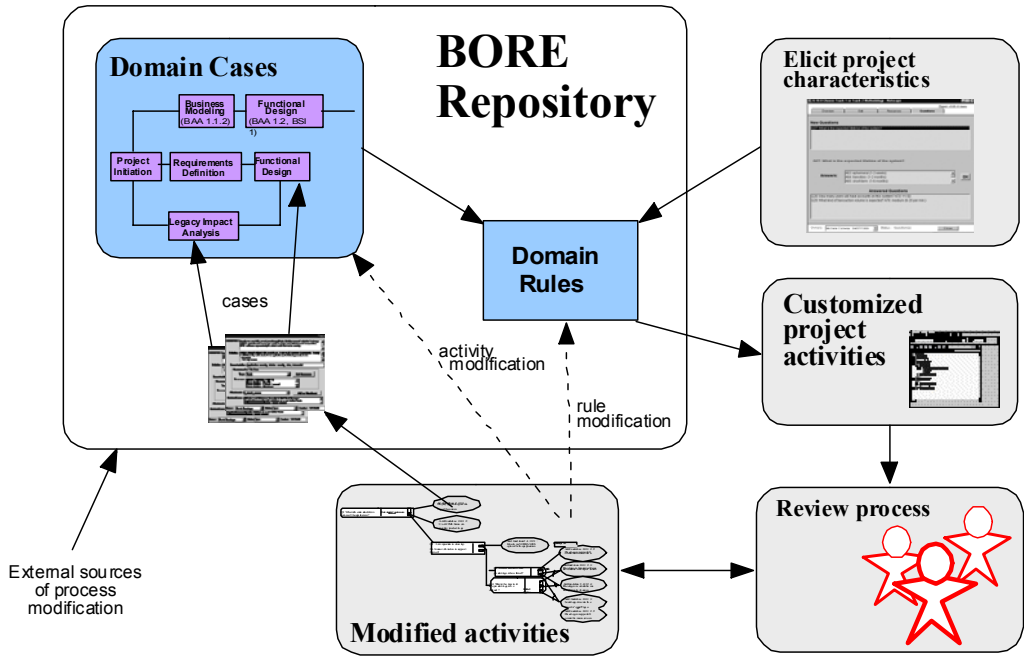

Figure 1: A methodology for capturing and disseminating software development knowledge.

Second, deviations to the software process can be formally expressed by creating a new set of activities and rules for matching the activities to project requirements. For example, suppose a project using Java applets determines that a new step is needed to address cross-platform and cross-browser needs. A new activity would be created along with a rule stating that when Java applet technology is used the activity is required. Subsequent projects will then benefit from this new knowledge, thus preventing problems with re-discovering that this is indeed a risk for systems using this technology.

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 [Terveen et al. 1995; Walsh, Ungson 1991] and our notion of organizational *learning*, where the emphasis is placed on using the standard as a means to learn from previous experiences and improve where needed.

2.2 Tools Supporting Flexible Processes

Because knowledge of software processes and when they should be used are being created dynamically through project experiences, tool support is necessary to disseminate process information. We have been investigating these issues through the BORE prototype, a Web-enabled application using a three tiered architecture consisting of HTML, Javascript and Java AWT for rendering the interface, Java for application logic and JDBC access, and relational database back-end. It can be accessed through a Web browser (must use Navigator 3.1 or greater and both Java and Javascript enabled) at http://cse-ferg41.unl.edu/bore.html (log in as 'guest').[1]

The general objective of BORE is to provide a single point of reference for documenting a project's software process. Instead of documenting the process in static text documents, often referred to as the Standard Development Methodology (SDM), process activities are maintained in a database and assigned to projects when applicable. Each project therefore sees, and is responsible for, that portion of the standard that applies to the project. BORE supports this through an interface that allows projects to tailor the process to their needs and view the resulting customized process.

The main interfaces for BORE are shown in Figure 2. The Case Manager, shown to the left in Figure 2, displays a hierarchical arrangement of cases that define the activities in a project's development process. In the figure, a project named "Goddard ISC demo" has been chosen from the list of resources in the drop-down menu that displays projects. Each activity contains project-specific

[1] BORE does not currently run on Internet Explorer.

Figure 2: BORE Case Manager and a case.

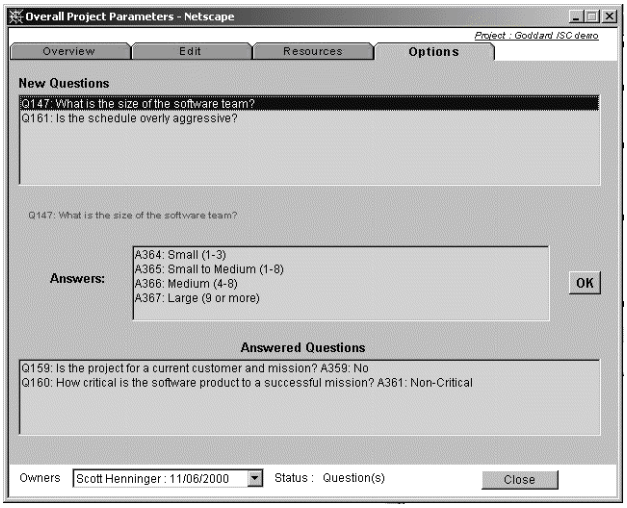

Figure 3: Project Options.

information as shown in the window to the right in the figure, which was obtained by double-clicking on the activity named "Conduct a Systems Requirement Review (SRR)" in the project hierarchy.

Bore Domains. Bore divides its repository into domains. Domains are independent knowledge realms that consist of a set of domain *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 mirror diverse environments or product lines.

Domain activities define the individual activities and relationships to other activities for the entire domain. When an activity is assigned to a project, its contents are copied into the project's project hierarchy. In case-based terminology, it becomes a case, meaning it holds context-specific information related to the general principle. For our purposes, the project has been assigned a specific activity, so the case is often referred to as the activity and the general principle corresponds to the domain activity.

Examples of activities include the standard process elements seen in process improvement standards, such as conducting reviews, planning and documenting test procedures, planning activities, etc. In addition, the BORE methodology allows these procedures to be defined at any desired level. An organization that wants a specific look-and-feel to their product could, for example, specify that a standard login screen be used for their applications. The tailoring process described here make this kind of detailed process possible because each project need only be concerned with the activities that match their specific needs.

Domain Rules are used to tailor the overall development process to the specific needs of different projects by matching domain activities to a specific project. As shown in Figure 3, this is accomplished through a question-answer session in the "Options" tab of an activity. The figure shows a set of questions that define some project tailoring factors. The answers define project requirements that will determine which activities will be assigned to the project.

For example, a recent effort to define an ISO 9000 compliant software process at NASA Goddard found that a System Requirements Review (SRR) is needed when the software is mission critical and the schedule is not aggressive. But if the schedule is aggressive, the SRR should be combined with a System Concept Review. These tailoring factors are easily programmed in BORE by creating rules stating the conditions for the different reviews. After answering the questions shown in Figure 3, BORE will add the proper review (and other activities) that have been designed into the process and encoded as rules.

Options can also be created that help break a process down into constituent activities. For example, a process specifying how COTS components should be evaluated may begin with activities for evaluating current COTS systems. Depending on whether candidates are found, more activities may be assigned to the project. This could be addressed in BORE by having one activity with options that specify further activities depending on the nature of the evaluated COTS. BORE is designed so that any activity can have options associated with it, which allows projects to answer the questions as they are addressed instead of at the beginning of a project.

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. When all preconditions 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 (displayed in the Options window of a case, see Figure 3), 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. A rule editor has been implemented to facilitate the creation of new rules as part of the process shown in Figure 1.

3 Related Work

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 [Osterweil 1987], and universal process models such as the CMM [Paulk et al. 1993] 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 [Cugola 1998]. BORE domains are seen as a "seed" that evolves as it is used [Fischer et al. 1994]. 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 [Terveen, Wroblewski 1990] than process programming [Curtis et al. 1992; Osterweil 1987], which is more concerned with integrating tools and documents to automate development activities, although elements of both issues are present.

These concepts have their predecessors, particularly in the form of software factories [Basili et al. 1992; Basili, Rombach 1991, 1988] and process programming [Curtis et al. 1992; Osterweil 1987], but little in the way has been done to create interfaces and CASE tools to support these concepts. The QIP approach in TAME [Basili, Rombach 1988] is another maturity-based framework that 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 [Basili et al. 1992]. 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.

This approach also has some roots in the design rationale field [Conklin, Yakemovic 1991; Fischer et al. 1992; Lee 1993; Maclean et al. 1991; Moran, Carroll 1996]. 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 [Conklin, Yakemovic 1991; Fischer et al. 1992] to more complex structures designed to capture logical relationships for computation [Lee 1993] 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.

The organizational learning 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 [Terveen et al. 1995]. 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" [Terveen et al. 1995] has served as a guiding principle for this work.

4 Conclusions and Future Work

The general goal of this approach is to create a software process standard that is flexible enough to split the gap between overly-restrictive development methodologies and ad-hoc software development practices and incorporate feedback to refine the process 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. These concepts have their predecessors, particularly in the form of software factories [Basili et al. 1992; Basili, Rombach 1991, 1988] and process programming [Curtis et al. 1992; Osterweil 1987], 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 case-based decision support for defining complex and specific processes that are managed by allowing projects to tailor the process to their individual needs.

Early BORE work, which largely focused on repository technology and search engines, was evaluated in pilot projects and using student projects. These studies demonstrated that repository technology alone was insufficient to successfully institute an organizational learning process. Work activities need to be designed that integrate both use of the tool and its continuous updating and improvement. The work described here attempts to address these issues through a process that supports learning from project experiences. Similar research has shown that such a system will be used by development personnel, provided it contains relevant, useful and up-to-date information [Terveen et al. 1995]. 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.

We are currently investigating the feasibility of using BORE to support the ISC Software Methodology currently being developed at NASA Goddard. Current efforts at defining this process found that there was a strong need to tailor the process depending on a number of high-level factors, such as mission criticality, the size of the software team, aggressiveness of the schedule, and whether the development effort is for new software or maintaining existing software. These factors are currently being documented in a matrix with if-then statements to describe other factors, such as whether high-level requirements have been provided or if COTS/GOTS are being used [Schultz et al. 2000]. Feedback from walkthrough meetings have indicated that project leads understand the need for tailoring the process, but have some problems with the matrix representation. BORE is seen as a possible solution to this problem and we are planning on creating a BORE-based prototype if the process to explore how this can be accomplished.

REFERENCES

[Arango 1989] Arango, G., "Domain Analysis: From Art Form to Engineering Discipline." in *Fifth International Workshop on Software Specification and Design*, (Pittsburgh, PA, 1989), ACM, New York, 152-159.

[Basili et al. 1992] Basili, V.R., Caldiera, G., Cantone, G., "A Reference Architecture for the

Component Factory." *ACM Transactions on Software Engineering and Methodology*, *1* (1). 53-80.

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

[Basili, Rombach 1988] Basili, V.R., Rombach, H.D., "The TAME Project: Towards Improvement-Oriented Software Environments." *IEEE Transactions on Software Engineering*, *14* (6). 758-773.

[Conklin, Yakemovic 1991] Conklin, E.J., Yakemovic, K., "A Process-Oriented Approach to Design Rationale." *Human-Computer Interaction*, *6* (3-4). 357-391.

[Cugola 1998] Cugola, G., "Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models." *IEEE Transactions on Software Engineering*, *24* (11). 982-1000.

[Curtis et al. 1992] Curtis, B., Kellner, M.I., Over, J., "Process Modeling." *Communications of the ACM*, *35* (9). 75-90.

[Fischer et al. 1992] Fischer, G., Grudin, J., Lemke, A., McCall, R., Ostwald, J., Reeves, B., Shipman, F., "Supporting Indirect Collaborative Design With Integrated Knowledge-Based Design Environments." *Human-Computer Interaction*, *7*. 281-314.

[Fischer et al. 1994] Fischer, G., McCall, R., Ostwald, J., Reeves, B., Shipman, F., "Seeding, Evolutionary Growth and Reseeding: Supporting the Incremental Development of Design Environments." in *Proc. Human Factors in Computing Systems (CHI '94)*, (Boston, MA, 1994), ACM, New York, 292-298.

[Henninger 1996] Henninger, S., "Building an Organization-Specific Infrastructure to Support CASE Tools." *Journal of Automated Software Engineering*, *3* (3/4). 239-259.

[Henninger 1997] Henninger, S., "Case-Based Knowledge Management Tools for Software Development." *Journal of Automated Software Engineering*, *4* (1). 319-340.

[Henninger et al. 1995] Henninger, S., Lappala, K., Raghavendran, A., "An Organizational Learning Approach to Domain Analysis." in *Seventeenth International Conference on Software Engineering*, (Seattle, WA, 1995), ACM Press, New York, 95-104.

[Johnson, Brodman 2000] Johnson, D.L., Brodman, J.G., "Applying CMM Project Planning Practices to Diverse Environments." *IEEE Software*, *17* (4). 40-47.

[Lee 1993] Lee, J., "Design Rationale Capture and Use." *AI Magazine*, *14* (2). 24-26.

[Levitt, March 1988] Levitt, B., March, J.G., "Organizational Learning." *Annual Review of Sociology*, *14*. 319-340.

[Lindvall, Rus 2000] Lindvall, M., Rus, I., "Process Diversity in Software Development." *IEEE Software*, *17* (4). 14-18.

[Maclean et al. 1991] Maclean, A., Bellotti, V., Young, R., Moran, T., "Questions, Options, and Criteria: Elements of Design Space Analysis." *Human-Computer Interaction*, *6* (3-4). 201-251.

[Moran, Carroll 1996] Moran, T., Carroll, J. (eds.). *Design Rationale: Concepts, Techniques, and Use*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1996.

[Osterweil 1987] Osterweil, L., "Software Processes are Software Too." in *Ninth International Conference on Software Engineering*, (Monterey, CA, 1987), ACM, IEEE, Los Alamitos, CA, 2-13.

[Paulk et al. 1993] Paulk, M.C., Curtis, B., Chrissis, M., Weber, C.V., "Capability Maturity Model, Version 1.1." *IEEE Software*, *10* (4). 18-27.

[Schultz et al. 2000] Schultz, D., Bachman, J., Landis, L., Stark, M., Meyers, G., Godfrey, S., Tilley, M., "A Matrix Approach to Software Process Definition." in *25th Annual Software Engineering Workshop*, (2000).

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

[Terveen, Wroblewski 1990] Terveen, L., Wroblewski, D., "A Collaborative Interface for Browsing and Editing Large Knowledge Bases." in *National Conference of the American Association for AI*, (Boston, MA, 1990), AAAI, 491-496.

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

[Walsh, Ungson 1991] Walsh, J.P., Ungson, G.R., "Organizational Memory." *Academy of Management Review*, *16* (1). 57-91.