Constructing an Organizational Memory for Software Development

Scott Henninger
Department of Computer Science & Engineering
115 Ferguson Hall, CC 0115
University of Nebraska-Lincoln
Lincoln, NE 68588-0115
(402) 472-8394 FAX: (402) 472-7767
scotth@cse.unl.edu

Abstract
Software development organizations are currently under tremendous pressure to refine and streamline the process of developing software. Process maturity models have been created to define a organization-wide process for all software development efforts [Paulk et al. 93]. But these methods advocate a one-size-fits-all approach that can have detrimental effects on organizations that develop diverse products with equally diverse development needs. In conjunction with a large information technology organization, we have undertaken an effort to help software development organizations make informed decisions on development strategies based on knowledge of techniques previously used in the organization. This provides information on previous projects experiences that can serve as the basis for streaming the development process by tailoring methods to the specific needs of a project.

Motivation
Software development organizations are currently under tremendous pressure to refine and streamline the process of developing software. This has resulted in efforts to define process maturity models that define the development process at increasing levels of maturity and the things organizations need to do to reach the next level of maturity [Paulk et al. 93; Humphrey 1989]. But these efforts are largely of a prescriptive nature, relying on analyses and methods that have little empirical foundation and do not take into account the nuances of an organization, the kinds of software products they build, or the development context [Grudin 1991].

In conjunction with a large information technology organization, we have undertaken an effort to help software development organizations make informed decisions on development strategies based on knowledge of techniques previously used in the organization. Decisions on which set of development tools and methods are best suited to a given project is a complex and multi-faceted problem. Different tools, techniques and methods have different strengths and weaknesses, and will therefore have different degrees of success for different kinds of development projects in a given organization. We have been designing an information system that captures information about how past projects were developed in a knowledge base that can be used to help developers choose appropriate development strategies for new and ongoing projects [Henninger Lappala 1994].

Our approach stands in stark contrast with current management techniques for software development, which tend to define a development process for the organization as a whole. Instead of mandating an overly rigid, one-size-fits-all, development process, our technique is designed with the flexibility necessary to meet the diverse needs of the different kinds of software developed by an organization. Management decisions are based on previous project experiences, and the organizational memory embodied in the system can be used as the basis for analyzing and improving the development process by learning from those past experiences.

Our view is not that organizations would define a process model for the company and refine it for individual projects [Paulk et al. 93], but instead that organizations develop an understanding of their development process and evolve toward using standard process models for given types of problems encountered in projects. Our decision support system facilitates this understanding by helping developers choose appropriate development resources for projects with common sets of characteristics.

A Case Study
Typical development organizations lack a formal basis for choosing a proper set of resources for developing software. The complexity of the business domain leads to diverse concerns requiring development efforts with widely different requirements and needs. The proliferation of CASE and other development tools exacerbates the situation with the sheer number of potential solutions. The lack of formal analysis and hype common in the literature make it difficult for developers to begin understanding the issues involved in assessing which tools are most appropriate for a given kind of development project.

We have been working closely with a development organization faced with these kinds of issues. This
organization consists of about 300 developers working on a number of projects for a major railroad corporation. They generally develop in-house information systems to support the corporation, although a few of these systems have made their way to commercial production. The organization is experiencing a general shift from data management on mainframe systems to decision support systems in a client-server environment, causing an increased diversity in product development and a lack of familiarity with the new development environment. These problems have caused confusion in the organization on which tools and techniques can or should be used, creating a lack of consistency between projects.

An earlier analysis of the organization revealed that the number of projects using different tools and methods had reached an unmanageable level. The study identified 12 different functional areas of business addressed by 26 different projects. The kinds of applications included order processing, revenue management, dispatch monitoring, resource planning, and scheduling among others. There were over 90 different development resources used in these projects, amounting to approximately one resource for every three developers in the corporation. The resources included operating systems, databases, languages, technologies such as voice recognition and information retrieval, CASE tools, development methodologies, word processors, spreadsheets and others. The majority of these resources were commercial products, but some were developed in-house to specifically fit the organization’s needs.

This study served to validate management’s intuition that decisions on which resources should be used on a project was becoming complex and motivated more by “technology lust” than an informed analysis. Each project presents special constraints that prevent attempts to mandate one set of resources to fit all projects. Projects tend to use an ad-hoc combination of tools and methods. For example, one project used tools based on information engineering to develop a set of business processes (a data-oriented structured requirements statement), data modeling techniques for design, and used a prototyping strategy to implement the user interface. This combination has been effective for this project, but there is no way of knowing if the same would hold true for other projects. Some means is needed to assess the special characteristics of a project and match them to an appropriate set of resources so this kind of success can be repeated.

This and many other development organizations need tools and methods to help them organize knowledge about tool usage; what has proven to work and what doesn’t for the class of applications they typically develop. Methods are needed to capture project experiences and use that knowledge to streamline the development process by matching effective strategies to projects with similar characteristics. Our approach has been to develop an information repository that organizes project experiences so that subsequent development efforts can use the accumulated knowledge.

Four Types of Resources

Our first step has been to identify the kinds of resources necessary for developing software in the organization. Our discussions revealed four resources having significant impact on development organizations within this organization: process models, development methods, technologies, and development tools. These four categories are by no means orthogonal, affecting each other in many ways. They also represent the specific concerns of our development organization. While some organizations may have a different set of concerns, we feel these categories have a broad applicability and can be used to demonstrate our framework.

Process Models. Process models specify the steps taken in the process of developing and maintaining a software product. Fueled by backlash from the CASE industry’s inability to fulfill their promises, researchers have begun to pay more attention to the process by which software is developed in organizations. Prototyping, information engineering, spiral models, and other approaches have gained acceptance as viable alternatives to traditional waterfall models. Maturity models, which define increasing levels of process maturity and the infrastructure that must be in place to achieve each level [Humphrey 89; Paulk et al 93], seek to find an appropriate process model for an organization as a whole.

Software Methods. Process models guide the software process, while the individual steps, by which software is developed is defined as a software method and they are used to execute one or more of the steps in the process [Yourdon 92]. So while a project may choose to use the waterfall model as the process model, structured analysis methods could be used for the analysis phase and object-oriented design methods for the design phase. The factors involved in choosing the proper mix of methodologies are numerous, including the familiarity of project personnel with the methods, problem characteristics, and the fit with process models and tools, among many others.

Software Technologies. Technology orients the development effort, affecting the techniques and tools used to develop software. An increasing number of software technologies have reached a level of maturity that choices need to be made on what kind of technology and which specific technology should be used in a development effort. For example, a decision support system can be developed with artificial intelligence or database technology, among others. Factors such as the experience base of developers, support for the technology and specific problem characteristics have a substantial impact on how the technology is applied and its effectiveness.

Development Tools. Development tools, such as programming languages and CASE tools, provide support for implementing software. Although many organizations want to stick with one language, the fact that different languages have different strengths and weaknesses dictates that a number of languages will be used in a given develop-
A Framework for an Experience-Based Repository

We initially viewed the process of matching tools to technology as simply one of a matrix with problem characteristics on one axis and tools on the other. But it soon became clear that this simple structure did not adequately capture the complexity we observed at the development organization. Figure 1 shows the basic structure we have been using. We call it a HyperMatrix model to point out it original purpose of the matrix, matching problem types to resources, and how it has evolved to include hyper-links to help manage and capture the structure of issues observed in our development organization. There are three parts to the model; a problem representation, project profiles, and a tool representation.

Problem Representation

The problem representation consists of two parts. The first is a problem hierarchy in which a development organization analyzes the kinds of problems it works on and creates a structure as shown in Figure 1. Each level of the hierarchy corresponds to a problem category. Problem categories consist of a label, references to a number of project profiles and references to the four types of resources. Links to project profiles reference specific projects attempting to solve a problem described by the problem category. These links are established when a project is planned and tools are chosen. Projects can be associated with more than one problem category. Links to the four resource types are derived from a category’s projects so that all of the resources used to solve that kind of problem can be viewed at once.

The second representation is a set of project characteristics associated with each project and its subproblems. Problem characteristics are used to describe a project and can be used to identify similar projects to analyze which tools are used to solve common problem characteristics. To facilitate the identification of projects with similar characteristics, each problem characteristic keeps track of all projects the characteristic is used to describe. References to the four kinds of resources are derived from the project listing.

The two kinds of problem representations define two ways in which problem types can be matched to tools. The first is to follow the hierarchy, choosing increasingly specific problem category until the most appropriate is found. From there links are found to the tools that have been found to be most effective for the problem category, and projects undertaking those kinds of problems. This kind of support is most useful when developers have a good idea of what kind of problem they are dealing with. Other problems are not so well-specified. In these cases a query can be constructed by collecting a set of problem characteristics and retrieving projects with similar characteristics.

Figure 1: The HyperMatrix Model

The HyperMatrix model consists of a problem hierarchy, project profiles, tools, and problem characteristics. Arrows indicate hypertext links between objects, indicating that objects have pointers to other objects. For example, the "project y" entry in the Process Model object points to the Project Profile for "y".

Matching Development Resources to Problem Types

Given our four types of software development resources, the solution to our development organization’s problem becomes one of matching the type of problem undertaken in a project to tools and project experiences. To accomplish this, a mapping between tools and problem types is needed to effectively identify which among the many alternatives is most appropriate. One way to achieve this mapping would be to undertake an exhaustive effort to list characteristics of problems needing a software solution and characteristics of software development tools. This kind of effort would indeed be prohibitive in terms of the resources needed to accomplish such a task. The relationships between variables are complex, interrelated, and it is exceedingly difficult to isolate which variables caused observed effects. Furthermore, since technology and problems are continually evolving, it wouldn’t be long before the mapping would become outdated.

Instead of exhaustively listing tool and problem characteristics based on abstract principles and decontextualized analysis, relationships and problems can be captured in the context of development. Subsequent projects can then concentrate on using and extending this information where needed, instead of repeatedly re-inventing the wheel. Basing this mapping on an experience-based repository has several advantages. It provides a concrete medium to disseminate corporate policies and inform people of other efforts in the organization. A key factor for such a repository to succeed is that the information must evolve with the changing development context (experience base of designers, changing organizational practices, new technologies, etc.).
Resource Representation

The resource representation consists of project descriptions for the four types of tools. Tool representations are derived from one of the problem representations. For example, if a project associated with the “Track capacity” category uses prototyping and waterfall process models, then those models will appear in the process model representation for the “Track capacity” category (see “project y” in Figure 1). Tool representations can also be derived from a set of problem characteristics by matching projects to the given problem characteristics.

We chose to match problems to tools in this manner because there were significant problems with other alternatives. The first would be to derive a set of tool characteristics that can be directly matched to problem characteristics. The problem is that tool characteristics are numerous and difficult to assess. While Turing-complete programming languages can produce any kind of computation, the real question is their effectiveness at solving a given problem. Another alternative would be to associate problem characteristics with tools. The main question then becomes how one knows when a tool is effective at solving a given problem characteristic. Choosing to match problem characteristics of projects with the tools used in the project provides the necessary grounding to assess the effectiveness of the tools.

Project Profiles

Project profiles contain information about specific development projects. The general structure of a project profile is shown in Figure 2. Project profiles can be associated with one or more problem category (for example, the ‘Railyard’ category shown in Figure 2). Project information can include a description, pointers to requirements documents, and any other kinds of information necessary to describe the project. Subproblems, such as ‘Railyard Scheduling’ in Figure 2, contain information on the resources used to develop parts of a project. Addressing the different subproblems may require different sets of resources. For example, while a project may be using a waterfall variant for developing the system, a particular issue might need further exploration through a prototyping strategy [Boehm 88], or an interface to a server machine needs to be developed, requiring a different development tool. The different resources are collected in the project profile to give a summary of the resources used in the project.

Reusing Project Experiences

We are currently in the process of developing a prototype to demonstrate our approach and elicit feedback from our client development organization. We will present a scenario of how developers could use the repository to access project experiences. We will also discuss how the deployment of such a system needs to strike a balance between properly fitting organizational practices and mandating organizational changes to incorporate the system and gain maximum benefit from its use.

Finding Similar Projects

Suppose our development organization has a need for a computerized system to track trains that we'll call “Train Tracker.” The system will take periodic update reports from remote train engineers on the current status of their trains. These status reports would include such information as the time table of the train - on time, late, or early - weather reports, track conditions, maintenance needs of cars or engines, etc. This information will be uploaded to a database system on a mainframe. This database will also be accessed by a command center which monitors all trains. Based on the information from the engineers, decisions can be made on how track conflicts can be resolved, where broken cars should be taken for repairs etc. If a track is down, the command center will need to re-route trains. All information transfer must be handled by fast communications and an effective user interface.

Identifying project characteristics

As opposed to starting a project from a blank sheet of paper and synthesize requirements and a project management strategy from scratch, it would be advantageous for developers to take their initial understanding of the problem...
The project has been described to the developer's satisfaction.

Characteristics can be defined using the "Other" field. Once characteristics or project experiences. Also note that new algorithm described below to find similar projects, values represented by the sliders are used in the retrieval descriptive, and the other somewhat less descriptive. The Status' and 'Train Schedule' characteristics highly user defining the Train Tracker project has made the 'Train the sliders associated with characteristics. In Figure 4, the chosen set of characteristics serves as part of the project's profile and is used to query the system for project experiences matching the problem characteristics.

Matching characteristics to tools

The developer now has a partial description of the Train Tracker project. This description can be used as a query to retrieve characteristics or projects that are related to the chosen characteristics. Retrieval is accomplished by a connectionist-based spreading activation algorithm that will not only retrieve projects and characteristics that match the query, but will also retrieve items that are associatively related. This helps users find related information when they don't know the proper terminology or have an ill-defined notion of what they are looking for [Henninger 94]. This algorithm is described elsewhere [Henninger 91; Henninger 93].

The description can also be used to find project experiences stored in the repository. Many kinds of project experiences can be stored, depending on the needs of the developers. We have been working with a development organization that is interested in knowing what kinds of resources should be used in a project with certain problem characteristics. For example, in Figure 5, the user has queried the repository for resources matching the characteristics in the Train Tracker description. The system retrieves the resources and displays them in the windows depicted at the top of Figure 5. The user can then display detailed information on these resources, such as the PowerBuilder entry shown in the figure. An interesting part of this representation is that experiences with the tool have been recorded through the "Cautions" entry. This can be an invaluable aid for selecting an appropriate resource and avoiding the repetition of mistakes.

Project Tracking

Since we are advocating that the mapping from problems to tools must be dynamically maintained to meet the changing needs of an organization, capturing project experiences is a crucial element of our approach. The information capture process needs to strike a delicate balance between gathering enough knowledge to support decisions about tool to problem mapping, while not becoming overly disruptive to

Figure 4: Interface for Characteristics Selection.

The window on the left side show the characteristic of an existing project. The right-side window shows the project currently being defined. The sliders represent a value depicting the degree to which the characteristic is descriptive of the project.

The developer now has a partial description of the Train Tracker project. This description can be used as a query to retrieve characteristics or projects that are related to the chosen characteristics. Retrieval is accomplished by a connectionist-based spreading activation algorithm that will not only retrieve projects and characteristics that match the query, but will also retrieve items that are associatively related. This helps users find related information when they don't know the proper terminology or have an ill-defined notion of what they are looking for [Henninger 94]. This algorithm is described elsewhere [Henninger 91; Henninger 93].

The description can also be used to find project experiences stored in the repository. Many kinds of project experiences can be stored, depending on the needs of the developers. We have been working with a development organization that is interested in knowing what kinds of resources should be used in a project with certain problem characteristics. For example, in Figure 5, the user has queried the repository for resources matching the characteristics in the Train Tracker description. The system retrieves the resources and displays them in the windows depicted at the top of Figure 5. The user can then display detailed information on these resources, such as the PowerBuilder entry shown in the figure. An interesting part of this representation is that experiences with the tool have been recorded through the "Cautions" entry. This can be an invaluable aid for selecting an appropriate resource and avoiding the repetition of mistakes.

Project Tracking

Since we are advocating that the mapping from problems to tools must be dynamically maintained to meet the changing needs of an organization, capturing project experiences is a crucial element of our approach. The information capture process needs to strike a delicate balance between gathering enough knowledge to support decisions about tool to problem mapping, while not becoming overly disruptive to...
Figure 5: Output of Decision Support System.

This figure shows the four lists of resources that the decision support system has returned for a given set of project characteristics. Each list is ranked, with best choices appearing first. Any of the resources can be selected for viewing a detailed representation, as done for "PowerBuilder" in the figure. Words in bold-italic denote hypertext links to other parts of the repository.

Achieving these goals means that the system must become part of the organization's normal design process [Terveen, Selfridge, Long 93]. The system must quickly reach a critical mass of information to become an invaluable part of the design process and maintain that usefulness by evolving with the changing needs of the organization. We have identified two places in which this process can be deployed in our development organization. The first comes from a standard practice in the organization to have a post-implementation survey for each project. This survey can provide information about the effectiveness of different resources for a given problem type as well as capturing caveats that can be stored in a "Caution" field such as the one in Figure 5. The second is to formalize the process of status reports and project tracking to monitor the issues and subproblems that arise in the course of the project. 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 and useful information [Terveen, Selfridge, Long 93].

Figure 6: A Project Profile.

The interface for project profiles allows one to gain access to information relevant to a project. The buttons on the left control what is viewed. For example, selecting the "requirements" button will display more information on project requirements.

Conclusions and Future Directions

A key technical issue of this project is the process of establishing relationships between the problem characteristics of a project and the resources used to develop the software product. Pattern matching techniques is one way to achieve the matching. The problem characteristics for a new project can be matched against a repository of previous project characteristics and displayed to the user. We are using a connectionist-based spreading activation algorithm [Henninger 91] to retrieve project information based on associative similarity measures. Knowledge-based critics [Fischer et al. 91] that monitor user actions and suggest appropriate development tools and strategies are also being employed. Knowledge-based analysis tools could help identify problematic areas in the development organization so that management can take action to improve the development process.

We are in the beginning stages of building the information system and defining how it can be put into place so that the system is use and knowledge collection becomes a normal part of the development process. The issues we are currently struggling with come from the dual, but integrated, needs for using and collecting knowledge about development projects. Developers need to have access to information on how tools were used, problems the tools were used for, assessments of the tools, and projects and people that used the tools. The knowledge capture process needs to strike a delicate balance between gathering enough knowledge to support decisions about tool to problem mapping, while not becoming overly disruptive to the development process. Our goal is to understand organizational practice so that new practices can fit the organizational framework in a natural manner, yet transform the activity into a knowledge collecting process that can provide the means for improving the development process.
By providing an organizational memory of past development efforts, we are providing the means to support the reconfiguration and reorganization of software development organizations. Not only can the system help streamline organizational processes, but new, more flexible, possibilities begin to emerge. Instead of forcing all development efforts into one development strategy or process, different strategies can be constructed for different kinds of projects, thereby tailoring and streamlining organizational policies.

References

[Boehm 88]

[Fischer et al. 91]

[Grudin 91]

[Henninger 94]

Henninger 91

Henninger 93

Henninger, Lappala 94

[Humphrey 89]

[Paulk et al 93]

[Terveen, Selfridge, Long 93]
L.G. Terveen, P.G. Selfridge, M.D. Long, "From 'Folklore' To 'Living Design Memory'," *Proceedings InterCHI '93*, ACM, 1993, pp. 15-22.

[Yourdon 92]