1995

An Organizational Learning Approach to Domain Analysis

Scott Henninger

University of Nebraska-Lincoln, scotth@cse.unl.edu

Kris Lappala

University of Nebraska-Lincoln

Anand Raghavendran

University of Nebraska-Lincoln

Follow this and additional works at: <http://digitalcommons.unl.edu/cseconfwork>

Part of the [Computer and Systems Architecture Commons](#)

Henninger, Scott; Lappala, Kris; and Raghavendran, Anand, "An Organizational Learning Approach to Domain Analysis" (1995). *CSE Conference and Workshop Papers*. 235.

<http://digitalcommons.unl.edu/cseconfwork/235>

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Conference and Workshop Papers by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

An Organizational Learning Approach to Domain Analysis

(Research Paper)

Scott Henninger, Kris Lappala, Anand Raghavendran

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

Abstract

As the application of computer technology continues to proliferate and diversify, the identification and understanding of application domains is becoming increasingly important to software development methodologies. Domain analysis techniques have been developed to accumulate and formalize the knowledge necessary for successful software reuse. These techniques have been shown to be useful, but suffer from defining the domain too restrictively, burying important relationships deep in domain taxonomies, and prohibiting flexible identification of domains with common issues. Techniques are needed that dynamically detect recurring patterns of activities in development projects. This paper presents a method for developing and refining the knowledge and experience accumulated by a development organization so it can learn from previous efforts. A case-based repository of project experiences supports the re-use and refinement of domain knowledge to reduce duplicate effort, build on successful efforts, and avoid repeating mistakes in the process of building quality software systems.

Motivation

The reuse of software components has received considerable attention as a method to improve software quality and development productivity. But there is a growing consensus that providing a library of source code is insufficient to support reuse [8, 16, 27]. The key to successful reuse lies more in understanding and defining the application domain for a collection of components [8, 31]. Defining and understanding domains is a difficult and time-consuming process that takes years to acquire sufficient expertise [11, 47]. The difficulty is exacerbated by the proliferation of technology, development tools, and the application of computing technology to increasingly diverse application fields. This diversity makes it difficult for development teams to integrate all the knowledge sources needed to design and develop software.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association of Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

ICSE '95, Seattle, Washington USA
© 1995 ACM 0-89791-708-1/95/0004...\$3.50

Recognizing the importance of domain knowledge to the development process, domain analysis techniques have been designed to systematically identify objects and relationships of a class of systems [32, 3, 39]. The central issue is identifying what constitutes the domain -- what should and should not be included in the domain. Most methods advocate creating a formal model of the domain [37, 47], making domain analysis most useful for established domains with well-known parameters [8, 38, 41]. But in the fast-paced world of changing business needs and technological advances, well-established domains in the computer industry are an increasingly rare commodity. Static and labor-intensive domain analysis methods are ill-suited for these dynamic and evolving domains, making it difficult to have a reasonably complete domain model that reflects the current state of affairs.

Ironically, the emphasis on defining "the" domain model often obscures the real issue; to find commonalities among systems to facilitate reusing software or other design artifacts. From this perspective, domain analysis is a process of identifying commonly occurring patterns across a number of development efforts. The "domain" does not necessarily need to be a family of applications or a formal model, but a set of problems within applications with recurring activities and/or work products.

Defining the domain in this manner has implications that go far beyond questions of which components can be reused to implement a system. Common project needs can be identified and their experiences can be used to choose development tools that meet the special needs of individual projects. The objective becomes more than software reuse, focusing on using previous efforts to learn how to create better products. As patterns emerge, top-down domain analysis methods can be used to formalize the patterns, facilitating domain evolution from the identification of isolated patterns to formally defined domain knowledge. Identifying established patterns of effort reduces the risk of costly domain analysis efforts by ensuring that the cost of analysis can be amortized over many uses.

We have been using an "industry-as-laboratory" [36] approach to investigate an organizational learning approach to domain analysis by studying an in-house software development organization for a major railroad corporation in the US, Union Pacific Railroad (UPRR). Our analysis revealed a need to collect and disseminate project experiences to reduce duplicate effort, build on successful efforts, and avoid repeating mistakes. Based on these obser-

Knowledge Domains	Examples
Application Domain	train scheduling, capacity planning, car management, train tracking, intermodal transportation, rail yard management, maintenance scheduling, disaster recovery, business process, etc.
Technical Domain	graphical user interfaces (GUI), communications, database systems, operations research, artificial intelligence, decision support, business modeling, etc.
Systems	Unix, PC (DOS, Windows, OS/2), mainframe, teradata
Architectures	mainframe, client/server
Development Methodology	Information Engineering, Prototyping, Custom SDM, Object-Oriented (Rumbaugh), Rapid Application Development, Spiral
Applications	network servers, databases, GUI builders, e-mail communications, CASE tools, project planning & scheduling, word processing, graphics
Languages	C, C++, Cobol, custom (RAIL), Basic, knowledge-based systems
Other Topics	re-engineering, metrics (function points), security, software change management

Table 1: Knowledge Domains at UPRR.

ventions, we have developed a methodology in which project experiences are captured and stored in a case-based repository with hypertext linking capabilities. Developers can identify common activities and artifacts by matching the characteristics of their problem to others stored in the repository. In this way, problems with similar concerns and issues are found, dynamically defining the "domain" to be problems with similar characteristics.

In what follows we begin by describing the organization, what domains of knowledge are required to develop software in the organization, and how it can benefit from the capture and dissemination of domain knowledge. We then describe how an organizational learning approach using a case-based repository can be used to facilitate the reuse and refinement of domain knowledge. A second-generation prototype is used to demonstrate how this method can be accomplished. We close with a brief description of relevant research and future directions for this project.

Distributed Knowledge in a Software Development Organization

An organizational learning approach to domain analysis is fundamentally dependent on the structure of the organization in question. To validate and refine our organizational learning approach, we have been studying an organization consisting of about 300 people that develop in-house information systems to support a major railroad corporation. This organization is experiencing a general shift from data management on mainframe systems to decision support systems in a PC-based client-server environment. The shift has caused a crisis in expertise along a number of dimensions, including Unix server technology, communications, PC applications, and decision support systems. Issues of configuration, server location, system downtime and recovery, and others that did not arise in the mainframe world are becoming critical issues in need of effective organization-wide solutions.

We have systematically studied software development at UPRR at both macro and micro levels. The macro level was revealed through a series of interviews with developers and key management personnel all the way up to the Assistant Vice President of Information Systems. We coalesced ideas from the interviews into a prototype that led to more feedback and a series of meetings that identified similar efforts and resources that we followed up on through interviews and detailed discussions with involved personnel. The micro level was explored through six contextual inquiries [22] in which we followed developers and project managers for half a day and interviewed them about their work while they performed their daily activities. This technique helps uncover details and reveal aspects of one's job that they tend to abstract out in a formal interview setting. Through these studies we have collected extensive notes and hours of video recorded information with an even mixture of developers and project managers.

Our foremost conclusion from these studies is that a combination of diverse development concerns, complexity and novelty in the development environment, and many relatively small-scale individual projects are working together to exacerbate the thin spread of application domain knowledge [11]. There are currently 26 separate projects in 12 different functional areas of the business, ranging from order processing and revenue management to dispatch monitoring, resource planning, and scheduling. In addition to intra-project communication needs, there is a need for communication between these projects, as they share concerns of the application domain (aspects of the railroad business) as well as common development platforms. The organizational lines tend to create barriers for this kind of communication, creating a lack of consistency across products and duplication of effort.

Not only is knowledge distributed, it falls along a number of interdependent domains (see Table 1) that were present in varying degrees in all projects we studied. Many domain analysis techniques focus on algorithms, analytical models

[4], and software components [37, 9]. But our analysis indicated that expertise is needed in a number of knowledge domains to effectively develop software at UPRR. Algorithm design played a minor role in day-to-day activities of developers and managers, and the amount of design work we witnessed by far outweighed the amount of programming work. People we studied spent their time in activities such as learning application domains, coordinating systems, communication between peers, analyzing customer needs and interface design, data structuring, and analysis for decision making, among others.

Because client-server computing is a fairly new field that UPRR has little experience with, many projects are pushing the boundaries of the infrastructure. In one design meeting we attended, a team was building an application that moved a dataset from the mainframe to the more accessible medium of Lotus Notes on PC workstations. The program was designed to be manually invoked from a workstation. This aspect of the design caused a great deal of discussion about the merits of automatically triggering the program from the workstation:

S1: "So then, my question becomes: Are we not far enough in our infrastructure where we can automatically trigger this job and move it over to Lotus Notes?"

Automatic job triggering is clearly an issue with broad applicability in UPRR's client-server architecture, but for which no known infrastructure exists:

S2: "Currently we have jobs that automatically run and populate... be it a Oracle database or whatever."

S3: "On the server!"

S2: "Right, going through server. But I don't know about Lotus Notes, so I can't speak of that. But I know there are things out there that could potentially do that. They can do that through client-server 10-33 machines. So it should be able to do that in Lotus Notes database. I know there are other systems out there that are working to get information directly from Oracle, which is only 10-33 machines, and again leading it into Lotus Notes. So there is a potential option and that is to move it straight to Oracle and then port it out of Oracle."

S3: "Yeah, I think, in fact, there are lot of tools available today and working to automatically initiate jobs on the server. But this one is kind of unique in that it has to be initiated from the workstation. But I don't know whether there are any to remotely initiate jobs on the workstation."

Here we have a potentially recurring problem for the organization that needs to be identified disseminated to projects needing solutions in the future. Its utility may seem obvious in hindsight, but it is an emergent need to people in the organization. Smart planning and analysis can identify some of the issues, but there will always be hidden, obscure, and non-obvious issues that will be missed without some means to detect the recurring patterns.

The issues and structure of the organization produce many barriers to formal domain analysis methods. The sheer number of separate efforts at UPRR lends itself to an organizational learning approach in which project

experiences are disseminated across project boundaries. It is difficult to define a domain because it is difficult to define boundaries. The domains are dynamic and constantly changing, often being defined by the first project to address the issues involved. Lack of experience in client-server computing caused domains to emerge as projects encountered issues fundamental to the development infrastructure.

Yet there is a need to disseminate what is currently known in a timely manner so the organization can build on successes, avoid duplicate efforts, and avoid repeating mistakes. These issues are not an artifact of poor management (in fact the organization has a sophisticated software development operation with many people devoted to development infrastructure issues such as the development process) but is a consequence of the complexity and dynamics of the transportation industry and advances in technology. Techniques are needed that can evolve as quickly as the domains are explored by projects.

An Organizational Learning Approach to Domain Analysis

From the beginning of our study, it has been clear to us that the size, complexity, and dynamic nature of software development at UPRR would prevent the construction of a comprehensive and up-to-date model. Even if a reasonably complete model were possible, understanding it would be a significant barrier. Therefore, our approach has been to create the infrastructure by which UPRR can derive its own models to address their most pressing issues. While some have characterized such an approach as "organizational memory" [6, 44], we have chosen "organizational learning" to emphasize that the real purpose is to learn and improve from previous efforts.

The next issue is how the experiences in the repository should be organized. Creating a classification structure would essentially involve the kind of labor-intensive domain analysis effort we are trying to minimize. We have therefore chosen to use case-based technology. Case-based methods do not require extensive classification to find information, and are often touted as a technique that works best in the kinds of ill-defined problem solving situations we are interested in [24, 42].

A Case-Based Approach to Organizational Learning

Case-based reasoning is an artificial intelligence method based on cognitive models postulating that much of human problem solving involves applying past experiences to analogically related situations. While early case-based systems attempted to provide autonomous problem solving by adapting existing solutions to new situations, recent systems have emphasized providing an external memory for users through an interactive process of decision support [25, 34]. A case-based repository for decision support can suggest how new problems can be approached, suggest the means for adapting a solution that does not quite fit, warn of possible failures, and help designers interpret and understand a situation [24].

Case-based decision aid technology is a perfect fit with an organizational memory approach to domain analysis because we are interested in situations in which there is no formalized or algorithmic solution available, but problem solving examples exist. Human interpretation and analysis of the situation is necessary, but people need help finding relevant cases because they either forgot or did not know about existing solutions and approaches to a given problem. This is particularly valuable to reduce the effect of distributed knowledge at UPRR. Case-based methods can also support the abstraction process that is so important to domain analysis by detecting patterns, such as when several cases suggest the same solution and/or are indexed with similar terms.

Retrieving Similar Cases

Any case-based approach relies heavily on the case retrieval mechanism, often referred to as the *indexing problem* [24], which is responsible for finding appropriate cases for a given problem description. Indexing, the process of representing cases with key terms and phrases, is only half the problem. The other half is the method of matching queries to case representations. Simple matching techniques have been shown to inadequately support the process of satisfying an information need, especially when the query is ill-defined [7, 20]. Methods are needed that can retrieve noisy and inexact patterns with a soft matching retrieval algorithm.

The indexing architecture we have adopted consists of three types of objects; terms, characteristics, and experience cases (see Figure 1). People searching for experience cases specify a query with characteristics. Characteristics are structured objects with a description, a list of cases that use the characteristic, and a list of terms that index the characteristic. They define a controlled vocabulary to index cases. People indexing cases are encouraged to reuse existing characteristics when they apply, although new characteristics can easily be defined. A controlled vocabulary approach was adopted for three reasons. First, for describing objects, such as source code, that do not follow the linguistic regularities of text documents, controlled vocabulary approaches may be superior to other indexing methods [37]. Secondly, this approach fits many organizations where standard terminology and acronyms are used to communicate common issues. We often heard statements like "That's a track capacity issue" at UPRR. Key phrases such as "track capacity" can be used as characteristics to help establish a carefully designed vocabulary that best describes domains within the organization. Third, defining a standard set of terminology is a first step toward formalizing domain knowledge [37].

The problem with a standard or controlled vocabulary is that it must be learned. This is a barrier not only to novices, but experienced people that are exposed to new projects with their own set of terminology. We therefore allow an uncontrolled vocabulary of terms to help find characteristics. People need to use characteristics to look for cases, but if they are unsure of which characteristics to

Figure 1: Indexing architecture for the repository.

The indexing architecture for case retrieval consists of three layers, terms, characteristics, and experience cases. Links represent indexing relationships. For example, characteristic *w* is indexed by terms *A* and *B*.

use, or want to find an exhaustive list of characteristics for a given issue, they can construct a query of terms to find characteristics. We do not allow terms to retrieve cases as this would reduce the benefits of using a controlled vocabulary.

We have chosen a spreading activation retrieval method that uses a connectionist relaxation algorithm to support finding partially matching patterns. The algorithm is explained formally elsewhere [19, 21], but the basic process is as follows. Let's say a user specifies term *A* in a query (see Figure 1). The *A* node is given an *activation value* of 1.0 that is passed to all characteristics it indexes, *w* and *x* in this case. The activation value passed to the characteristic nodes will be reduced by the strength of the link weight (which measures the degree of association between a term and characteristic), and is adjusted by other factors such as fan-in and decay [19]. On the next cycle, *w* and *x* will have a non-zero activation value that will be passed to all term nodes they are connected to. This process repeats until activation values stabilize or a user defined number of cycles is reached. The same process is used to find cases with characteristics defining the query.

The strength of this method is that it is able to find partial patterns in the repository. For example, when *x* passes its activation value to *C* and *D* on the second cycle, these two nodes work together to reinforce *x*'s activation value and activate *z*. Further cycles reinforce *x* and *z* because of the feedback loop between these nodes and *C* and *D*. In the end, *x* and *z* will have similar activation values. The structure of the repository detects that characteristics *x* and *z* are similar because they have similar representations. The spreading activation process has detected the pattern through a partial match. Notice also that *z* would not have been retrieved if we were using a straightforward matching algorithm. Spreading activation found *z* because it is similar to *x*, which directly matched the query of *A*. While other partial match paradigms, such as Latent Semantic

Figure 2: Querying for GUI builders and Displaying the Case Representation of a Development Tool.

Indexing [12] and Lexical Affinity [28], can also find partial matches, the spreading activation method was chosen because it is particularly suited to retrieving non-text objects such as source code [20].

Reusing Project Experiences

Through our studies at UPRR, we have identified some of the kinds of project experiences that need to be disseminated in an organizational learning repository. In this section, we present some of these issues through a second generation experimental prototype that we are using to demonstrate how an organizational learning approach to domain analysis would work. One of the fundamental design issues in case-based technology is defining what constitutes a case. Our answer has been to allow any idea or artifact that may be useful to others to be a case. This leads to eclectic and non-uniform case representations, but it satisfies the requirement to collect project experiences that the organization can build on. The following scenarios, adopted from transcripts of videotapes and notes from interviews at UPRR, explore some of the issues encountered by the various stakeholders in the development process.

Choosing an Appropriate Tool

With over 90 different development tools in use at UPRR, choosing an appropriate set of tools for a project is becoming a significant problem. The sheer number is a formidable barrier, but the complexity and overlapping nature of these tools, ranging from operating systems, databases, and languages to CASE tools, development methodologies and word processors, makes it difficult to know which tool should be used for what kinds of problems. Exacerbating the problem is the fact that vendor claims are often overstated, making it difficult to assess tradeoffs. An organizational learning approach can support the decision making process by providing access to a repository of cases with information on how different tools have fared in the development context of UPRR.

One project at UPRR is developing a system that monitors trains coming in and out of the switching yard. The users require a graphical user interface (GUI) on PC's that communicates with a Unix server to retrieve and store data

on an Oracle database. An informed decision on which of a handful of GUI builders to use needs to be addressed before the project can continue. While the decision can be made based on the usual hearsay or by launching a six-month domain analysis effort, an organizational learning approach uses a repository of project experiences to analyze which tools would be most appropriate. The decision maker can begin by entering queries to find relevant project characteristics (alternatively, the user could have entered the characteristics directly, as shown in Figure 1). For example, Figure 2 shows a query using the term "GUI" that finds the characteristics shown in the *Matches* window. The user can choose any of these characteristics for inclusion in the problem description, which is accumulated in the *Characteristics* window. Each time a new characteristic is added to the problem description, the *Matches* window is updated to display the cases retrieved by the current set of characteristics. Note that the *Matches* window can be changed to view projects, tasks, development methods, etc., as shown in the *View* window. Each case has one or more types associated with it. Choosing different views displays different types of cases found by the same characteristic query.

For the query shown in Figure 2, PowerBuilder is retrieved as the top match (cases are displayed in rank order of matching according to spreading activation values). The decision maker can view the case representation of the tool, which includes a description, the characteristics that index the case, information on how to achieve different tasks with the tool, and some problems encountered when using the tool (Cautions). The How-To, Cautions, and Projects fields provide hypertext links to detailed descriptions and cases in the repository. Perusing this description, the user discovers that PowerBuilder takes total control of the operating system when it sends messages to the system, which would not be compatible with the current design. Another complicating factor is that previous contractor had developed some of the communication software in C++. Therefore, not only must the GUI builder allow for sub-processes, but it must interface with C. These facts can also be entered in the system to find characteristics and projects that meet these constraints.

This scenario mirrors a project at UPRR that went through a lengthy process in which the shortcomings of different GUI builders were gradually discovered at considerable cost in manpower. The issues of controlling the operating system and C compatibility were not issues directly addressed in documentation. Developers experimented until they found it was not possible. Eventually it was discovered that Zink met all the constraints of the project. This tool had been used before, so the expertise and knowledge existed in the organization, but people in this project were simply not aware of it. This episode outlines the difficulty of choosing a standard tool suite. PowerBuilder is the GUI builder of choice for UPRR, but there are situations in which it will not work. An organizational learning approach allows people to explore the specific context in which a tool will be used, resulting in a streamlined tool selection process.

There are a number of other software development resources at UPRR that can benefit from this approach. For example, a standard development methodology (SDM) has been used in the past by the organization, but it is generally considered to be more appropriate for development in the mainframe environment. The diversity of projects and unfamiliarity with client-server tools and techniques have thwarted efforts to come up with one best methodology. An organizational learning approach to this problem, in which experiences with different methods, such as prototyping, joint-application development, information engineering and others, can be stored in the repository. New projects can then explore what has worked best for a problem with similar characteristics. This kind of flexibility is essential for effective management of projects with diverse application needs.

Finding an Expert

Studies of development organizations have revealed that considerable attention and effort is applied to finding people in the organization that are needed to get one's work done [35]. A network of people with expertise in specific problem areas is usually formed by individuals in an informal manner that can cause gaps such as the GUI builder problem in the previous section. Because the repository will never have a full list of problems and solutions, it is important to have the means to find pointers to sources that may be able to help. For example, the Unix server infrastructure at UPRR is still in its infancy. One project needed to communicate with a Unix server. Querying the repository, they find a number of characteristics and issues involving socket communications, but none which tried to use sockets to connect PCs and Unix machines, nor did any of them mention trying to write communication calls of any kind between Unix machines and PCs. So the next best thing is to contact some of the people that have worked on similar problems to see if they can offer any advice. This can be performed by choosing the "people" view as shown in Figure 3.

The repository dynamically constructs a "knowledge profile" [46] through the cases in the repository their name is associated with. This indirectly associates people with

Figure 3: Finding experts.

characteristics, allowing users to query for expertise. To the extent they are connected with characteristics in the query, they will get a stronger retrieval value, and can be considered to possess more expertise on the subject area than others. Notice that people do not have to create and maintain a personal profile for this feature. Instead, expertise is dynamically defined by the cases a person has been involved with.

Capturing Application Domain Knowledge

Another key issue is entering new cases and refining existing cases in the repository. For example, one project at UPRR is concerned with periodically migrating a TSO (Time Sharing Option) dataset on the mainframe onto a Lotus Notes database on a PC workstation. Let's say the developers begin by using the repository to find similar cases. Although several cases in repository have similar characteristics, none match the particular problem of migrating the databases. As the developers find relevant characteristics and explore the issues found by querying the repository with those characteristics, they are in essence accumulating a list of characteristics describing their problem. Once they are finished looking for ideas, the developer can use this description to create a new case. This can be accomplished by creating a new case description (a project description in this case) as shown in Figure 4.

The user needs to fill in some of the fields such as the problem statement and issues and barriers, but the system will keep track of the characteristics used and the projects viewed by the user. These are automatically entered as part of the case representation. The user is free to edit and add to the fields, but the system has taken a significant time-saving step by automatically entering some of the information. The case will also be placed on an open problems list for projects with unresolved issues. This list can be used to identify trends in some of the current problems faced by the organization. Later, when progress is made the case can be modified by updating and filling in fields such as "solution statement" and "problems encountered".

Figure 4: New case interface.

Support for Formal Domain Analysis

An organizational learning repository can be used to support a formal domain analysis effort. By querying the repository for patterns of common activities, the domain analyst can get a comprehensive picture of the issues and approaches used to solve problems in the domain. For example, upper management at UPRR have noticed that a number of projects have begun to struggle with issues of backup and recovery in the client-server world. A domain analysis is needed to create a uniform approach to the problem. The analyst begins by querying the system with terms such as "backup" and "recovery", finding characteristics such as "Automatic backup", "File backup", "Disaster recovery", "Backup scheduling", "Update frequency" and others. The analyst constructs some 'characteristic' queries and follows links to cases describing problems and projects to understand some of the different ways backup and recovery have been addressed. From these, the analyst begins to construct some of the facets such as:

- mode: automatic or manual
- data type: database, files
- architecture: mainframe, server, workstation
- scheduling: volume size, loading

From here the analyst can begin to organize the software artifacts that have accumulated about backup and recovery issues. The repository provides a comprehensive and convenient mechanism for performing the analysis.

Knowledge Collection

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. Similar efforts have shown that design repositories will be used by development personnel, provided it contains relevant and useful information [44]. The information capture process needs to strike a delicate balance between gathering enough knowledge to support decision making, while not becoming overly disruptive to the development process. Deployment of the system outlined by our prototype 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.

Our analysis revealed that there's enough information collection activities going on at UP that we should be careful not to add to the burden. Achieving these goals means that the system must become part of the organization's normal design process [44]. Using the repository as a *basis* for design as outlined in the scenario on capturing domain knowledge ensures that significant issues are collected. We are also working with UPRR to transform status reports from stand-alone documents to a knowledge collection activity that involves developers in the continuous refinement of the repository. Status reports would become a hyperdocument historical tracking medium where issues in the report are linked to specific cases in the repository. Another source is the post-implementation survey for projects that is already a standard practice in the organization. This can give valuable informal feedback on design and technique effectiveness. Some have also mentioned the possibility of doing the feedback during the project. By collecting information already available in case form, we can increase the utility of the system while adding little overhead to the development process.

Assessment and Future Directions

An organizational learning approach to domain analysis and software development is best suited to development contexts in which common customer needs are being addressed in similar application domains by multiple projects. This is closest in scope to in-house development organizations, but can also be adopted in organizations that do contract or commercial off-the-shelf development [17]. The scope of our approach naturally incorporates all of the stakeholders involved in software development. Management, developers, marketing, customers and others can share project experiences. The nature of these experiences will differ across the various disciplines, but the general infrastructure of organizational memory systems remains intact.

Our long-term goal is to use the repository as an empirical testbed to show which techniques work best for a given domain. Although data may be spotty, it will be real, and we feel confident that clear trends will emerge that can add an empirical basis to vendor, methodology, and researcher claims [13]. Our joint charter thus far has been to explore and characterize the organization to understand what kind of infrastructure is needed to more effectively develop software at this organization. Incorporating the technique into the complex fabric of any large development organization is a lengthy and tenuous process. We are currently working with UPRR to integrate these methods into their development methodology and begin the process of setting up the case-based repository. The prototype has thus far been used as a communication medium to disseminate our results and conclusions. The next phase of the project will center around the additional complexities of technology transfer.

In addition to the ongoing work on knowledge collection outlined above, we are working closely with a project at

UPRR that will serve as a pilot project for this approach. We are currently “shadowing” the development project, using videotape [46], project-related electronic mail, and frequent site visits to “seed” our prototype with data relevant to the project. Toward the end of the year, we plan to have project personnel take over the repository we have seeded. We will then analyze the effort and lessons learned from adopting the technology to design an organization-wide set of tools supporting an organizational learning approach to software development.

Related Work

The domain-oriented perspective is beginning to gain momentum in the software engineering community. Researchers in software reuse have long acknowledged that reuse within small, well-defined, domains works better than trying to solve general reusability problems [8, 39]. Recent effort in Domain-Specific Software Architectures (DSSA) [18] and research on software architecture [2, 40] also recognize the importance of domain-specific solutions.

In many respects the approach outlined here follows the domain analysis prescription to *identify* reusable information in the problem domain, *capture* relevant information, and *evolve* the information to fit or evolve it to meet current needs [3]. Specifically, we address issues of domain identification, evolution, and reuse, and are currently in the process addressing acquisition issues. Domain representation will also become an issue for us as we begin to analyze information that we have collected.

But it would not be too unfair to characterize most domain analysis approaches as a form of top-down analysis in which a formal or semi-formal process is applied to turn existing information in well-known domains and artifacts into reusable abstractions with broader applicability than what existed before [38]. Our approach complements these top-down approaches by providing key bottom-up information about projects that is captured as the knowledge begins to emerge. This information can be used to identify common patterns and flag them as candidates for formal domain analysis efforts.

In addition to working with UPRR, our approach has drawn from a number of sources. Design rationale is designed to capture the rationale behind the designs of systems [30]. This provides information about systems that reaches beyond source code, often concentrating on questions of ‘why’ certain design decisions were made. This information can help projects and organizations avoid repeating mistakes or re-hashing decision that have already been addressed. Systems supporting design rationale have largely concentrated on issues of organizing the information into a variety of similar structures [26, 14, 29], although some studies of knowledge capture have been performed [10]. While capturing information about design decisions is certainly an important part of our approach, we have broadened the perspective of design rationale to include issues of tool usage, development methods, project issues, and any other kind of information that may be useful to other development projects.

There are also a number of similar efforts that focus on domain analysis at the component and algorithm level. The experience factory [9] defines an organizational framework that separates component design from application development, but largely focuses on “experience” at the level of source code components. Technology books formalize knowledge about algorithms and formal models for classes of problems [4]. Case-based reasoning techniques have been employed to adapt and compose reusable components [15]. This method also focuses on source code, which we have found to be a small part of the information needs at UPRR. Our approach broadens these perspectives and focuses on providing case-based repository technology that can help an organization accumulate and use expertise to streamline the entire development process.

We have also been influenced by the various process improvement efforts [5, 33, 23]. Our approach shares the common goal of enhancing productivity and quality of software development as a continuous improvement process. The development process is only one of the many issues facing development organizations, and it is often unclear how organizations will accumulate the information necessary to perform process improvement. We support this process by providing the means through which knowledge can be re-used, refined, and accumulated as an organization matures in different domains.

Our approach is most closely related to some approaches to constructing organizational memory systems [45]. While the organizational *learning* approach outlined in this paper emphasizes the process of learning from and improving on previous efforts, these efforts focus on the first step in our domain lifecycle, collecting and disseminating design information. TeamInfo focused primarily on querying and browsing issues for a organizational memory of loosely organized e-mail messages [6]. Answer Garden was built to turn knowledge into an organizational asset in a network of multiple-choice questions and answers [1]. Their bottom-up process evolves the repository in response to user questions, and would be most useful for collecting experiences about development tools. Our framework goes further to support the process of analyzing domains and turning the individual cases into assets that can streamline the development process.

Our approach is closest in scope to an organizational memory effort at AT&T [44]. This research has created a *Designer Assistant* that provides access to a repository of issues such as real-time performance constraints, local programming conventions, properties of an implementation, and others. Their repository approach uses traditional knowledge-based technology, but accomplishes many of the goals we have set out to address. The STARS framework also shares some of our concerns with developing and maintaining domain-specific assets for the continual improvement of reuse-oriented activities [43]. In many ways we have instantiated their framework through our case-based organizational learning approach.

Conclusions

By providing methods that are best suited to well-defined software development domains, most domain analysis techniques fail just when they are needed the most. We have developed an approach that supports developers when they are faced with tasks that are less well-understood. By identifying common patterns among problems and projects, the process of domain understanding is supported, not just formalizing that which is already well-established.

Our approach simultaneously answers two lingering questions in domain analysis: what constitutes the domain and where does the information for a domain analysis come from? In our case, the "domain" is any set of problems with similar characteristics that can be used as a basis for a design decision. Patterns that emerge from the process of finding problems with similar characteristics represent areas within an organization that may benefit from a formal domain analysis process. The case-based approach coupled with the spreading activation retrieval method can find domains with similar characteristics that would escape one's attention in the statically defined taxonomies and models of many domain analysis techniques.

The accumulated knowledge of application domains benefits software development activities by providing a baseline to judge which techniques work best under a given set of circumstances, resulting in quality software developed with streamlined development methods. Our approach begins this process by placing reuse as an integral part of the entire development process. From the very onset of a project, development team members use the repository to design and support decision making.

We are developing the organizational learning approach to software development in a user-centered or participatory design method in which we deploy prototypes to collect feedback and refine our model to fit the organization's needs. Successful deployment of this system would not only help the software development process in our client organization, but will provide a crucial first step toward better understanding the software development process and how it can be improved.

Acknowledgments. We gratefully acknowledge the efforts of Lynden Tennison and Jim Montequin at Union Pacific Railroad. Charles Danile was involved in early phases of the project. This research was funded by Union Pacific Railroad and the Applied Management Institute in Omaha, NE.

References

- [1] M.S. Ackerman, T.W. Malone, "Answer Garden: A tool for growing organizational memory", *Proceedings of the Conference on Office Information Systems*, ACM, New York, 1990, pp. 31-39.
- [2] R. Allen, D. Garlan, "Formalizing Architectural Connection," *Sixteenth International Conference on Software Engineering* (Sorrento, Italy), ACM, IEEE, Los Alamitos, CA, 1994, pp. 71-80.
- [3] G. Arango, *Domain Analysis: From Art Form to Engineering Discipline*, Proceedings Fifth International Workshop on Software Specification and Design, (Pittsburgh, PA), ACM, pp. 152-159, 1989.
- [4] G. Arango, E. Schoen, R. Pettengill, "A Process for Consolidating and Reusing Design Knowledge," *15th International Conference on Software Engineering (ICSE '93)*, Baltimore, MD, IEEE, pp. 231-252, 1993.
- [5] V.R. Basili, H.D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, 14(6), June 1988, pp. 758-773.
- [6] L.M. Berlin, R. Jeffries, V.L. O'day, A. Paepcke, C. Wharton, "Where Did You Put It? Issues in the Design And Use of a Group Memory," *Proceedings InterCHI '93*, ACM, 1993, pp. 23-30.
- [7] N.J. Belkin, W.B. Croft, "Retrieval Techniques", *Annual Review of Information Science and Technology (ARIST)*, M.E. Williams (Ed.), 22, 1987, pp. 109-145.
- [8] T.J. Biggerstaff, "An Assessment and Analysis of Software Reuse", *Advances in Computers*, 34, pp. 1-57, 1992.
- [9] G. Caldiera, V.R. Basili, "Identifying and Qualifying Reusable Software Components", *Computer*, 24(2), Feb. 1991, pp. 61-70.
- [10] E.J. Conklin, KC Yakemovic, *A Process-Oriented Approach to Design Rationale*, Human-Computer Interaction, 6, pp. 357-391, 1991.
- [11] B. Curtis, H. Krasner, N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, 31(11), November 1988, pp. 1268-1287.
- [12] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, R. Harshman, "Indexing by Latent Semantic Analysis", *Journal of the American Society for Information Science*, 41(6), 1990, pp. 391-407.
- [13] N. Fenton, S.L. Pfleeger, R.L. Glass, "Science and Substance: A Challenge to Software Engineers," *IEEE Software*, 11(4), January 1994, pp. 86-95.
- [14] G. Fischer, A.C. Lemke, R. McCall, A. Morch, "Making Argumentation Serve Design", *Human-Computer Interaction*, 6(3-4), 1991, pp. 393-419.
- [15] G. Fouque, S. Matwin, "A Case-Based Approach to Software Reuse," *Journal of Intelligent Information Systems*, 1, 1993, pp. 165-197.
- [16] M.L. Griss, *Software reuse: From library to factory*, IBM systems journal, Vol. 32, No. 4, 1993 pp. 548-565.24(4)
- [17] J. Grudin, "Interactive Systems: Bridging the Gaps Between Developers and Users," *Computer*, 24(4), April 1991, pp. 59-69.

- [18] F.Hayes-Roth, *Architecture-Based Acquisition and Development of Software: Guidelines and Recommendations from the ARPA Domain-Specific Software Architecture (DSSA) Program*, Teknowledge Federal Systems Technical Report, Feb. 4, 1994.
- [19] S. Henninger, "Information Access Tools for Software Reuse," *Journal of Systems and Software*, 1995 (in press).
- [20] S. Henninger, "Using Iterative Refinement to Find Reusable Software", *IEEE Software*, 11(5), September 1994, pp. 48-59.
- [21] S.R. Henninger, *Locating Relevant Examples for Example-Based Software Design*, Ph.D. Dissertation, Department of Computer Science, University of Colorado, University Microfilms International #9320432, 1993.
- [22] K. Holtzblatt, S. Jones, "Contextual Inquiry: A participatory technique for system design", *Participatory Design: Principles and Practice*, A. Namioka, D. Schuler (Eds.), Erlbaum, Hillsdale, NJ, 1993.
- [23] W.S. Humphrey, *Managing the Software Process*, Addison Wesley, Reading, MA, 1989.
- [24] J.L. Kolodner, *Case-Based Reasoning*, Morgan-Kaufman, San Mateo, CA, 1993.
- [25] J.L. Kolodner, "Improving Human Decision Making through Case-Based Decision Aiding," *AI Magazine*, 12(2), Summer 1991, pp. 52-68.
- [26] J. Lee, "Design Rationale Capture and Use," *AI Magazine*, 14(2), Summer 1993, pp. 24-26.
- [27] M.D. Lubars, N. Iscoe, "Frameworks versus Libraries: A Dichotomy of Reuse Strategies," *Proceedings of Sixth Annual Workshop on Software Reuse*, Owego, NY, IEEE Computer Society, 1993.
- [28] Y.S. Maarek, D.M. Berry, G.E. Kaiser, "An Information Retrieval Approach for Automatically Constructing Software Libraries," *IEEE Transactions on Software Engineering*, 17(8), pp. 800-813, 1991.
- [29] Maclean et al., "Questions, Options, and Criteria: Elements of Design Space Analysis", *Human-Computer Interaction*, 6(3-4), 1991, pp. 201-251.
- [30] T. Moran, J. Carroll, *Design Rationale: Concepts, Techniques, and Use*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1994.
- [31] W. Myers, "Workshop Explores Large-Grained Reuse," *IEEE Software*, 11(1), pp. 108-109, 1994.
- [32] J. Neighbors, "The Draco Approach to Constructing Software from Reusable Components," *IEEE Transactions on Software Engineering*, 10, September 1984, pp. 564-573.
- [33] M.C. Paulk, B. Curtis, M. Chrissis, C.V. Weber, *Capability Maturity Model, Version 1.1*, IEEE Software, 10(4), pp. 18-27, July 1993.
- [34] M. Pearce, A.K. Goel, J.L. Kolodner, C. Zimring, L. Sentosa, R. Billington, "Case-Based Design Support: A Case Study in Architectural Design," *IEEE Expert*, 7(5), October 1992, pp. 14-20.
- [35] D.E. Perry, N.A. Staudenmayer, L.G. Votta, "People, Organizations, and Process Improvement", *IEEE Software*, 11(4), July 1994, pp. 36-45.
- [36] C. Potts, "Software-Engineering Research Revisited," *IEEE Software*, 10(5), pp. 19-28, Sept., 1993.
- [37] R. Prieto-Diaz, "Implementing Faceted Classification for Software Reuse," *Communications of the ACM*, 35(5), May 1991.
- [38] G. Arango, R. Prieto-Diaz, *Domain Analysis software system modeling*, IEEE Computer Society Press, Los Alamos, CA, 1991.
- [39] R. Prieto-Diaz, G. Arango, "Domain Analysis Concepts and Research Directions," in *Domain Analysis and Software Systems Modeling*, IEEE Computer Society Press, Los Alamos, CA, 1991, pp. 9-33.
- [40] J. Purtilo, *The Polyolith Software Bus*, Tech. Report CSD 2469, Computer Science Dept., Institute for Computer Studies, Univ. of Maryland, College Park, MD, 1990.
- [41] M.A. Simos, "The Growing of an *Organon*: A Hybrid Knowledge-Based Technology for Software Reuse," in *Domain Analysis and Software Systems Modeling*, IEEE Computer Society Press, Los Alamos, CA, 1991, pp. 204-221.
- [42] S. Slade, "Case-Based Reasoning: A Research Paradigm," *AI Magazine*, 12(1), Spring 1991, pp. 42-55.
- [43] *Informal Technical Report for Software Technology for Adaptable, Reliable Systems (STARS)*, Tech Report No. STARS-UC-05159/001/00, Electronic Systems Division, USAF, November 13, 1992.
- [44] L.G. Terveen, P.G. Selfridge, M.D. Long, "From 'Folklore' To 'Living Design Memory'", *Proceedings InterCHI '93*, ACM, 1993, pp. 15-22.
- [45] J.P. Walsh, G. Rivera, "Organizational Memory", *Academy of Management Review*, 16(1), 1991, pp. 57-91.
- [46] D.B. Walz, J.J. Elam, B. Curtis, "Inside A Software Design Team: Knowledge Acquisition, Sharing, and Integration," *Communications of the ACM*, 36(10), pp. 62-77, October 1993.
- [47] P.T. Ward, L.G. William, *Domain Modeling*, Tech Report SERM-012-90, Software Engineering Research, Boulder, CO, 1990.