Creating And Using An Industrial Domain Model *

Michael F. Dunn

Industrial Software Technology
Earlysville, VA 22936
Tel: (804) 978-7475
Email: mfd3k@virginia.edu

John C. Knight

University of Virginia
Charlottesville, VA 22903
Tel: (804) 982-2216
Email: knight@virginia.edu

Abstract

A successful reuse process requires a set of well-understood domain models. Creating these models, however, can be an extremely difficult task, requiring a great deal of information-gathering and formatting. In this paper, we discuss two aspects of a method we have been developing to cope with these complexities. First, we discuss our work in devising a method by which unstructured domain information can be gathered. Second, we discuss the format in which we capture domain knowledge, and how this knowledge can then be used. The challenges we have encountered while using this method in an industrial environment are also discussed.

Keywords: Reuse, domain analysis, domain modeling, practice.

Workshop Goals: Learning; networking; sharing domain modeling experience.

Working Groups: Domain analysis/engineering, Reusable component certification, Tools and environments.

*We thank Motorola, Inc. for their generous financial and technical support for this research.

1 Background

Domain analysis and modeling are generally agreed to be crucial first steps to developing an organized software reuse process. The abstract nature of these two processes, however, has been a major source of difficulty for organizations considering introducing reuse.

The authors' involvement in domain analysis and modeling stems not only from an interest in facilitating component-based reuse processes, but also from a desire to create a framework by which components can be certified. Once a cohesive model is created for a family of systems built by an organization, the quality criteria to which each family member must adhere can be identified.

2 Position

2.1 Introduction

Performing a domain analysis can require an arduous amount of work; it usually requires sifting through often-incomplete system documentation, examining large sections of application code, and interviewing busy technical experts. This diverse, unstructured information must then be tied together into a cohesive model. A complicating factor is the issue of who should perform the analysis. If the analysis is performed by one who is unfamiliar with the domain, then it is often difficult for that person to distinguish what domain information is important from a reuse standpoint. Consequently, the analyst is often required to become a domain expert in his own right in order to do an adequate job. However, a similar situation can exist when a domain expert performs the analysis. The expert can have such detailed knowledge of the domain that he might have a tendency to add too much information to the analysis, clouding the resulting model. Also, the expert will often have a tendency to carry a great deal of conceptual baggage from previous system implementations, constraining the analysis to information on how systems have been built in the past, rather than on how they should be built under a reuse-based process.

Creating a domain model based on the information gathered during the domain analysis is also a major hurdle. The first problem is a carry-over from the domain analysis: namely, determining what domain knowledge is important enough to include in the model, and what information should be discarded. The second issue is that of structuring the model for maximum usefulness. A domain model which includes useful information but which is not formatted carefully will be difficult to use and end up on the engineer's shelf.

The authors have recently been involved with a domain modeling project with a large industrial organization, where these issues came to the forefront. A major part of this project was to devise a "domain modeling cookbook", that is, a generic process for performing a domain analysis, and capturing that information in a standardized format. The goal was for this process to be flexible enough to be used by any industrial organization, but rigorous enough that it was clear what information is important, and how that information should be organized. The next several sections describe the results of this work, and the insights gained.

2.2 Performing A Domain Analysis

The organization with which we collaborated was responsible for developing test harnesses for the software that drives telephone switches manufactured by the company. The organization is staffed by about twenty-five software engineers, and is responsible for about ten large test tools.

Much of the up-front analysis work was based on the results of a questionnaire completed by the organzation's technical staff. This questionnaire was intended to introduce the analysts to the organization's mission, product line, current development process, current tools and methods to support their process, and current status of their reuse activities. The questionnaire was a necessary first step in large part because of the physical distance separating the analysts and the programmer community. The two groups were several hundred miles away from each other, making it necessary to coordinate the initial analysis effort by telephone, email, and occasional site visits.

The answers provided to the questionnaire laid the foundation for the work performed during the on-site visits. During these visits, meetings were held with members of the technical staff, management, process control, and quality assurance areas. This gave a balanced view of the overall sofware production environment. Although the organization's customers were not included in the discussions, it also would have been useful to get insights into the customer's current needs, how these needs are likely to change over time, and how these needs would impact the set of reusable assets the organization should have available.

Nevertheless, the content of the meetings with the technical staff was most revealing. The discussions focused on the following areas:

- The technical requirements of the product line.
- The decomposition of the domain into its subdomains.
- Common terminology used within the subdomains.
- The generic requirements of systems used within the subdomains.
- The set of components required to support systems built within these subdomains.

As a result of these discussions, a number of issues came to light:

Terminology: Members of the technical community often disagree on the meaning of common terms used within a particular domain, even those used on a daily basis. The term "test case", for example, a central concept in this domain, changes its meaning depending on the level and type of testing being performed. Thus, two engineers are likely to have a completely different notion of what constitutes a test case depending on the types of systems with which they are familiar. This highlights the importance of establishing working definitions for basic domain concepts before attempting to gather detailed domain knowledge.

Focus: Different participants often have a different perspective on what is important to the domain. Their perspectives are dictated, in large part, by the projects in which they have participated most recently. Thus, information-gathering sessions can easily turn into debates over technical side-issues unless the moderator maintains a clear focus. However, if the moderator is not familiar with the domain in question, it can be hard for him to realize when the discussion is going astray.

Cohesive Viewpoint: This issue is closely affiliated with the previous issue. While staff members have typically worked with several different system implementations, they often will not have a cohesive view of the similarities and differences between these systems, and the major functional aspects of these systems. This makes it difficult to identify broad categories of useful components. This tendency to get lost in the details makes a case for using an analyst who is not a domain expert, since he will only have a high level view, free of extraneous details.

In addition to questionnaires and interviews, strategic planning documents, system user guides, and a limited amount of source code analysis also proved useful in providing a picture of the domain's structure.

2.3 Purposes of a Domain Model

The main purpose of a domain model is not simply to determine the set of reusable parts needed to support a model, it is also to create a conceptual framework in which these components can be used. This means creating a generic model that captures the salient features of systems developed within the domain. An example of such a generic model is the one often used to explain the five phases of program compilation: lexing, parsing, semantic analysis, code generation, and optimization. Most compilers are not literally implemented with these phases in sequence, but the conceptual model gives implementers an idea of the basic features that need to be included in any compiler.

Other purposes of a domain model include:

- To serve as a basis from which component certification criteria can be derived.
- To help the organization determine which aspects of the domain are best served by using software components, by using domain-specific languages, or by using application generators.
- To help the organization determine which aspects of its development process are best served by reuse, and which are best served by other tools and process improvements.
- To highlight to the organization the specific factors that will influence the set of parts needed for future development. This can include changes to the product line, changes in available technology, and changes in the organization's mission.

2.4 Structure And Use Of The Domain Model Document

We structured the domain model document so that the organizational factors, process and tool factors, and domain-specific factors of the domain were made explicit. The goal was to provide not only a model of the systems supported and developed by the organization, but also to provide insight into how they could be developed using components, and what cost savings could be expected. Consequently, the document includes the following main sections:

Terminology: This section defines the basic words and phrases used within the domain.

Mission and Strategy: This section describes the customer goals which the software organization is commissioned to fulfill. It also highlights the potential changes in the customer community that might affect these needs. A change in product line, for example, would necessitate changes in software, which would necessitate the development of new software components. For each specified goal, either a set of reusable assets is listed that can satisfy the goal, or a set of process improvements is listed that can better serve the goal.

Process: This section describes the development process the organization can use to facilitate creating its target systems. The description includes the stages to be used in the development process, and the roles of the participants in the process.

Technological Infrastructure: This section describes the set of tools needed to support the development process, and maps tools to desired lifecycle work products.

Domain Model: This section, which is the central part of the document, describes the generic models of systems developed within the domain. Each model is broken into subdomains, such as user interface or report generation. Each subdomain is then broken into different implementation models. For example, the subdomain of "user interfaces" would include such implementation models as graphical user interfaces and command-line interfaces. Each implementation model is then broken into a description of the model, the trade-offs of using the model within this domain, the possible failures that can occur as a result of using this model, systems that have already been implemented by the organization that use this model, and a list of potential components that have been or can be developed to support this model.

Certification Criteria: Identifies the criteria to be used in certifying the components identified for the implementation models. The format used here is described by Dunn and Knight [1].

Financial Justification: Identifies the financial costs and benefits that can accrue from the development of the identified reusable assets, and includes a schedule for implementing the assets prioritized according to immediacy of need and highest financial benefit.

3 Comparison

Our method currently relies heavily on manual effort and discussions with experts. By contrast, Iscoe describes a process developed by EDS that relies heavily on automated information gathering techniques [2]. This process takes as input source code, formal models, and other development assets, and produces sets of "specification models". Human expertise is used only to resolve ambiguity.

Tracz provides a comparison of a number of different approaches to domain analysis and modeling [3]. Our experience bears out most of the observations noted about the similarities between these different approaches. Specifically,

- One of our end results was a design model for the domain,
- We assumed the existence of previously developed systems within the domain,
- The resulting model did imply the existence of a layered underlying implementation architecture,
- The process we used was highly iterative.

A significant part of the Software Productivity Consortium's (SPC) Synthesis process involves specifying tailorable requirements documentation, and creating specific source code components to fit those requirements [4]. The SPC's domain analysis process has three parts - development of a conceptual framework for the domain, development of a reuse architecture, and development of product composition mappings. In spite of the differences between these phases and the ones described in this paper, the desired end products of the two processes are quite similar - identification of specific software assets, and development of generic system models and specifications.

References

[1] M. Dunn and J. Knight, "Certification Of Reusable Software Parts," Tech. Rep. CS-93-41, University of Virginia, 1992.

[2] N. Iscoe, "Domain Modeling - Overview and Ongoing Research at EDS," in *Fifteenth International Conference on Software Engineering*, 1993.

[3] W. Tracz, "Domain Analysis Working Group Report - First International Workshop on Software Reusability," *ACM SIGSOFT Software Engineering Notes*, vol. 17, pp. 27–34, July 1992.

[4] G. C. J. S. Faulk and D. Weiss, "Introduction to Synthesis," Tech. Rep. INTRO_SYNTHESIS.90010-N, Software Productivity Consortium, June 1990.

4 Biography

Michael F. Dunn is an independent consultant specializing in software reuse, especially domain analysis and asset certification issues. He has worked as both a practitioner and researcher in reuse-related topics for the past several years, and has been associated with companies as diverse as IBM, Sperry-Marine, and Motorola. He received his B.S. in Computer Science in 1985 from Duke University, and his M.S. in Computer Science in 1990 from the University of Virginia.

John C. Knight is a professor of Computer Science at the University of Virginia. He joined the University of Virginia in 1981 and prior to that was with NASA's Langley Research Center. ¿From 1987 to 1989 he was on leave at the Software Productivity Consortium. His research interests are in dependable computing and software reuse.