Report on the Domain Analysis Working Group Session

James Neighbors, moderator

1988 Workshop on Software Reuse
Rocky Mountain Institute for Software Engineering (RMISE)
1988 Boulder, Colorado
(unpublished)

Why Do Domain Analysis?

Constructing systems from *reusable software components* has been identified as a method of providing significant productivity improvements in the construction of systems. The applicability of different software components ranges from very general (can be used in all systems) to very domain specific (can be used only in vertical applications specific to the domain). *Domain Analysis* is concerned with the process and results of identifying the domain specific components across the entire spectrum of applicable problem domains. This working group was concerned with the process and results of domain analysis.

In particular, given a domain analysis, an organization should be able to:

1. use the domain model to check the specifications and requirements for a new required system in the domain.
2. educate new people in the organization providing them with the general structure and operation of systems in the domain.
3. derive operational systems directly from the statement of the system in domain specific terms.

The Relationship between Domains

Since problem domains span a range from general to domain specific, it is not surprising that they are best arranged as a *domain hierarchy* as shown in figure 1. In this figure the *application domains* are the most problem domain specific while the *execution domains* are the most general. To understand and perform the domain analysis of a very specific but complex domain such as EFT systems, we might enlist the previously performed analyses of banking and communication.

**** insert figure 1 here *****

Once a version of the analysis of EFT systems were done we would expect to provide implementations of it using specializations of the schemes which provide implementations for the constituent domains of the analysis, namely banking and communication.

How is Domain Analysis Done?

The working group decided to approach the rather fuzzy idea of domain analysis with a practical problem. First we assumed that we were a company interested in cornering the market on the vertical application of "library management systems". In this case we were both the *domain experts* and *domains analysts* in the domain analysis. Second, with our domain analysis in hand we would see if we could handle the specification for a specific library system. The specification was not made available to the group during domain analysis.

We attempted to identify the *objects*, *operations* and *relationships* between what we (as domain experts) perceived to be important about the domain. In particular, different members of the group used many different analysis techniques including:

- data flow diagrams
- entity-relationship diagrams
- semantic nets
- object diagrams
- class hierarchies w/inheritance

Since most of the above techniques lend themselves to "leveling", that is formation into hierarchies of abstraction, the group quickly had to deal with the two basic problems of domain analysis. The first problem was one of *depth of analysis*. Since the "library management domain" analysis was performed as a gedanken experiment there were no supporting domains; but usually there is the question of how much support should be put into the library domain itself and how much should be required of the supporting domains. The second problem was one of *width of analysis*. Since doing a domain analysis is basically a trade-off between specialization and generality, there is always the question "is this function required by most of the systems built in this problem domain?" These two sets of trade-offs supported much spirited discussion in the group.

The discussion did not degenerate into the usual nit-picking differences between all of the analysis representations. Quite the contrary. The members of the group seemed fluent in all the representations and it was generally accepted that there is no one favored representation for domain analysis. Each analysis technique can capture the "flavor" of the particular domain under analysis from different overlapping perspectives. Usually only one each from the object hierarchy, data flow and control flow representations need be used to analyze a domain.

The most interesting result of performing the domain analysis was the realization that the *domain analysis rationalization* is one of the most important results of the process. The moderator of the discussion started from a hard-boiled "if it doesn't produce code then what good is it" point of view; but this provides only for the *construction* of systems. A "good" system is one which:

1. works reliably according to specification
2. can be understood
3. can be communicated to others.

The constructive point of view only strives to produce point 1; but, points 2 and 3 applied to users may inform us that a quality system conforming to spec does not suffice to solve the problem. In other words, the specification of the system does not solve the problem.

Since no one analysis technique seemed to be appropriate for all domain analysis and much of the information being communicated about the process of analyzing a particular domain has to do with issues and trade-offs, it was suggested that a hypertext-oriented deliberation system such as MCC's gIbis system is probably the appropriate scheme for capturing the domain analysis rationalization. The nodes in such a system should contain analysis models as design artifacts and trade-offs as issues and positions.

Applying the Domain Analysis

After hammering out an analysis of the domain of "library management systems" we attempted to apply it to the following short specification (This specification was taken from a recent conference on software specification.):

Required Functions

1. Checkout/Return a copy of a book.
2. Add/Remove a copy of the book.
3. Get the list of books either by author or by subject area.
4. Find the list of books checked out by a borrower.
5. Find out what borrower last checked out a particular book copy.

Required Constraints

1. All copies in the library must be available for checkout or be checked out.
2. No copy of the book may be both available and checked out.
3. Ordinary users may not have more than a predefined number of books checked out at any one time.

Our domain analysis easily covered all of the functional requirements and most of the constraints. Constraint 1 however violated our domain analysis. In fact our domain analysis rationalization had led us to believe that constraint 1 is *never* achievable since it assumes no administration or transit time on the part of the library staff. Our domain analysis went to great lengths to characterize this time.

Even in this little example this is very exciting. *The domain rationalization pointed out an error in a system specification in the domain far before any system design or construction was attempted.* Since the cost of correcting an error grows very rapidly as a system's lifecycle progresses, the ability to detect specification problems early may be a primary benefit of domain analysis.

Basic Domain Analysis Process

We found the following general process to apply to domain analysis.

1. Establish the domain subject area.
2. Collect the domain experts.
3. Establish the *depth of analysis* (see above).
4. Establish the *width of analysis* (see above).
5. Define the specific domain objects, relationships, and constraints.
6. Hand test the domain by attempting a description of a specific system in the domain.
7. Package the domain for constructive reusability by expressing it in a form for a transformational refinement tool such as Draco.

I wish to thank the working group for an interesting session. I learned the importance of domain rationalization.