

Representing Knowledge in Domain Engineering

David Eichmann*

Repository Based Software Engineering Program
Research Institute for Computing and Information Systems
University of Houston – Clear Lake
2700 Bay Area Blvd.
Houston, TX 77058
(281)283-3875
eichmann@rbse.jsc.nasa.gov

Abstract

The nature of domain analysis revolves not around change, although that is a necessary and desired aspect of the task. Rather, domain analysis has at its core the capture and characterization of instantiation alternatives, the parameterization of architectural and component features in a manner such that they can handle the variations in requirements confronting the system developers working in this domain. Hence, support for change is only the temporal projection of the variation found in analyzing a domain for multiple systems to be concurrently developed. Transforming a system to account for newly arising requirements differs from simultaneously instantiating two closely related systems only in the temporal aspects of the acts. The domain model must be able to support both as natural aspects of domain engineering.

Keywords

domain engineering, patterns, knowledge representation

Workshop Goals

To try to fuse a bunch of ideas that I've got running around in my head with other folks' perceptions.

Working Groups

“Domain Engineering Helps Manage Change – Hype, Myth, Wishful Thinking, or Reality?”

* This work has been supported by NASA Cooperative Agreement NCC-9-30, RICIS research activity RB02A.

Background

I've been exposed to a number of rather diverse ideas in the last year or so, ranging from new work in some traditional reuse areas such as domain engineering to work in some areas not so traditionally associated with software reuse, like design patterns (although we may be closing that conceptual rift) and ontological modeling.

The Repository Based Software Engineering (RBSE) project at the Research Institute for Computing and Information Systems (RICIS) has been working for a number of years in the area of software reuse and more recently, in the reengineering of legacy systems as an aspect of reuse [7]. The purpose of RBSE is the support and adoption of software reuse through repository-based software engineering in targeted sectors of industry, government, and academia.

In applying a reuse-based dual lifecycle process in order to achieve efficiency and consistency across the software products as a whole, we've encountered a number of challenges that have proven to derive as much benefit from knowledge representation as from software reuse.

Position

The software lifecycle is a complex set of interrelated activities. Some in our field claim to understand the nature of the relationships between the phases of the lifecycle and the roles involved in traditional “point-solution” approaches to software development. However, newer and arguably more productive reuse-based process models such as the “dual-lifecycle” approach to software development – where two perspectives (domain engineering and application engineering) are employed – raise new uncertainties about the way the lifecycles (and their individual phases) fit together, and how and where to capture and model knowledge.

Our experience with the ROSE project [8, 12, 17] has allowed us to witness the establishment and implementation of a reuse-based dual-lifecycle software development organization on a non-trivial scale (see Figure 1). The choice and tailoring of initial “theoretical” models (e.g., the CLLCM [14]) into the practical processes that ROSE team members use on a day-to-day basis gave us insight into the structure and interactions within working teams using this approach.

Much of what currently passes for “domain model” in the sense of a characterization of a family of *systems* is more suitably modeled as an ontology within which a properly parameterized architectural framework is defined. The nature of domain analysis revolves not around change,

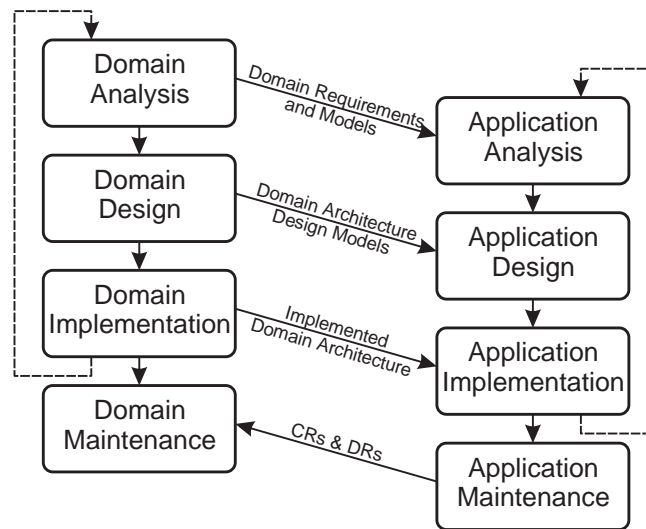


Figure 1: A DE/AE Software Lifecycle.

although that is a necessary and desired aspect of the task. Rather, domain analysis has at its core the capture and characterization of instantiation alternatives, the parameterization of architectural and component features in a manner such that they can handle the variations in requirements confronting the system developers working in this domain. Hence, support for change is only the temporal projection of the variation found in analyzing a domain for multiple systems to be concurrently developed. Transforming a system to account for newly arising requirements differs from simultaneously instantiating two closely related systems only in the temporal aspects of the acts. The domain model must be able to support both as natural aspects of domain engineering.

Comparison

Much of the current work on domain analysis and dual-lifecycle software processes centers around the capture of domain information and making it available to builders of actual applications in the domain [2, 4, 5, 6, 10, 11, 13, 15, 16]. A characteristic of domain products is their support for adaptation (through composition, inheritance or – if necessary – modification). Domain engineering efforts thus strive to employ consistent and effective means to support application-specific adaptation. One of the process designer's choices is to decide where these means are defined: they can be embedded in the process, left to the application engineer's choice or produced explicitly as part of domain engineering. These three views by themselves may be a little simplistic in that processes in the dual lifecycles have more influence on each other than implied by a simple producer-consumer relationship.

Examination of domain engineering products leads to interesting questions:

- What makes a domain product (analysis model, architecture design, etc.) “special” as opposed to the corresponding products in a point-solution lifecycle?
- Is there any distinction between the first iteration of an organization's application engineering life cycle and the n-th iteration?

I believe there is no distinction: although they result from a different process (enactment), domain products tend to be expressed in a manner consistent with habitual products of their kind. This implies that domain engineering can be performed in a manner that does not automatically entail readily apparent domain engineering specificity* in the product set.

The influences that the domain and application engineering lifecycles have upon each other still hold (although with different manifestations) when domain engineering-aware notations and techniques are used. For instance, the layered design architecture presented by Becker and Diaz-Herrera [3] is consistent with the culmination of an informal design feature selection procedure similar to that in the ROSE Domain Design process. Alternative formalisms for the representation of domain architectures (such as the formal architectural components proposed by Allen and Garlan [1]) present opportunities for treating some of the interactions between lifecycles in a disciplined manner, as they represent a common formal language for manipulating the composition of domain products.

References

- [1] Allen, R. and D. Garlan, “Formalizing Architectural Connection,” *Proc. 16th International Conference on Software Engineering*, Sorrento, Italy, May 16-21, 1994.
- [2] Bailin, S., “Difference-Based Engineering,” *Seventh Annual Workshop on Software Reuse*, St. Charles, IL, August 28-30, 1995.

* That is, distinguished characteristics that appear specifically in domain engineering products but are not usual for point-solution products. For example, Becker and Diaz-Herrera [3] demonstrate design guidelines that result in a very specific layered architecture for domain engineering products.

- [3] Becker, M. and J. L. Diaz-Herrera, "Creating Domain Specific Libraries: a methodology and design guidelines," *Proc. Third International Conference on Software Reuse*, Rio de Janeiro, Brazil, November 1-4, 1994.
- [4] Biggerstaff, T. J., "Second Order Reusable Libraries and Meta-Rules for Component Generation," *Seventh Annual Workshop on Software Reuse*, St. Charles, IL, August 28-30, 1995.
- [5] Butler, G., "Design Deltas in Reusable Object-Oriented Design," *Seventh Annual Workshop on Software Reuse*, St. Charles, IL, August 28-30, 1995.
- [6] Edwards, S., "Good Mental Models are Necessary for Understandable Software," *Seventh Annual Workshop on Software Reuse*, St. Charles, IL, August 28-30, 1995.
- [7] Eichmann, D., "The Repository Based Software Engineering Program," *Proc. Fifth Systems Reengineering Technology Workshop*, Monterey, CA, Feb. 7-9, 1995.
- [8] Eichmann, D. and C. Irving, "Life Cycle Interaction in Domain/Application Engineering," *Seventh Annual Workshop on Software Reuse*, St. Charles, IL, August 28-30, 1995.
- [9] Eichmann, D., M. Price, R. Terry and L. Welton, "ELSA and MORE: A Library and Environment for the Web," *Proc. 13th Annual National Conference on Ada Technology*, Valley Forge, PA, March 13-16, 1995.
- [10] Guerrieri, E., "Enhancing the Use of Domain Analysis," *Seventh Annual Workshop on Software Reuse*, St. Charles, IL, August 28-30, 1995.
- [11] Henninger, S., "Accelerating the Successful Reuse of Problem Solving Knowledge Through the Domain Lifecycle," *Fourth Int. Conf on Software Reuse*, Orlando, FL, April 23-26, 1996, p.124-133.
- [12] Irving, C. and D. Eichmann, "Patterns and Design Adaptability," *Third Annual Conference on The Pattern Languages of Programs*, Allerton Park, IL, Sept. 4-6, 1996.
- [13] Legard, S. and B. Karakostas, "The Impact of Technological Change on Domain Specific Software Architectures," *Seventh Annual Workshop on Software Reuse*, St. Charles, IL, August 28-30, 1995.
- [14] Rogers, K., M. Bishop, C. McKay, "An Overview of the Clear Lake Life Cycle Model," *Proc. Ninth Annual National Conference on Ada Technology*, U.S. Department of Commerce, National Technical Information Service, March 1991.
- [15] Simos, M. A., "Domain Modeling Techniques for Representing Commonality and Variability: Towards a Comparative Framework," *Seventh Annual Workshop on Software Reuse*, St. Charles, IL, August 28-30, 1995.
- [16] Ta, A. D. and D. W. Hybertson, "A Domain Framework: A Basis for Enhancing Reuse Among Domains," *Seventh Annual Workshop on Software Reuse*, St. Charles, IL, August 28-30, 1995.
- [17] Weisskopf, M., C. W. Irving, C. W. McKay, C. Atkinson, and D. Eichmann, "Maintenance In a Dual-Lifecycle Software Engineering Process," *Int. Conf on Software Maintenance*, Monterey, CA November 4-8, 1996.

Biography

David Eichmann is an Associate Professor and Chair of Software Engineering at the University of Houston - Clear Lake and Director of Research and Development of the Repository Based Software Engineering Program (RBSE). Besides normal academic duties, his responsibilities include management of a fifteen person research and development group working in the areas of reuse /

reengineering and Internet resource discovery.

Dr. Eichmann joined the UHCL software engineering faculty in August of 1993 after visiting for a year in his role with RBSE. He held a position in computer science at West Virginia University from 1989 to 1992, where he lead the Software Reuse Repository Lab (SoRReL) group. Before that he was a member of the software engineering and computer science faculty at Seattle University and database manager at Weeg Computer Center at the University of Iowa.

Dr. Eichmann is a graduate of the University of Iowa, where he received his Ph.D. in computer science in 1989.