The Role of Process Families in Reuse Adoption

Steven Wartik
Software Productivity Consortium
2214 Rock Hill Road
Herndon, Virginia 22070
Tel: 703/742-7176
E-mail: wartik@software.org

Abstract

Organizations often overreach when trying to adopt reuse. They try to institute reuse visions that are too complex, in whole or in part, for the knowledge and skills they possess. This paper proposes that reuse processes be defined as families, with precisely-defined commonalities and variations, and that the variations be chosen such that some family members are suited to organizations just learning reuse, whereas others are for organizations with much reuse experience. This is illustrated using the Synthesis process family. The paper presents some of Synthesis' commonalities and variabilities, then delineates how two members—one for advanced organizations, another for those new to reuse—resolve the variabilities. Comparing the two leads to some insights on how one's ultimate reuse goals might be relaxed to accommodate current practice.

Keywords: Software reuse, software development process, family, process family, domain engineering.

Workshop Goals: Validate problem and solution based on experiences of other researchers; understand domain analysis/engineering in family-oriented terms.

Working Groups: Domain analysis/engineering, reuse process models, reuse maturity models.

1 Background

For the past four years, I have been working at the Software Productivity Consortium, a consortium of aerospace companies. In these companies, projects develop software under contract to an external customer. These projects are generally grouped into business-area organizations. These organizations tend to amass expertise in particular areas, and try to win contracts in those areas. Such organizations lend themselves to reuse approaches that incorporate domain analysis techniques.

2 Position

2.1 Problem Statement

Reuse presents many challenging technical problems. However, a clear theme from last year's workshop on software reuse was that some of the major impediments to its adoption are glaringly non-technical [1, 2, 3, 4]. Companies often lack a comprehensive reuse program because project managers are unwilling to gamble on anything new and unproven. Those willing to take the chance often find their hands tied: contractual obligations and organizational requirements prevent them from acquiring new tools, or using non-traditional software methods, or altering their software development process—in brief, from making changes that would facilitate reuse.

Management at organization-wide levels (above project management) faces similar hurdles. An organization might establish a reuse library, but be unable to populate it with components from previous projects—the contractors who paid for developing those components claim software ownership rights, not being eager to see the components used to develop a competitor's software cheaply. Moreover, the cost of establishing a suitable environment for reuse can be high, and is difficult to predict. An organization should not expect to recoup its investment immediately, which can make management uncomfortable.

This situation is not universal, but it exists in many organizations that the Consortium supports. This poses a problem. The Consortium has been tasked to develop advanced technologies that support reuse. It has done so, and the technology has been well-received by some organizations. Others find the technology interesting, but do not feel able to risk trying an advanced reuse technology. Yet they are under pressure to reuse software. How should the Consortium support them?

The Consortium has devised a reuse adoption process that addresses this question [4]. This paper concentrates on one aspect of reuse adoption: the software development process an organization uses. Specifically, what problems does an organization face as it changes its software development process to accommodate reuse, and what is an appropriate strategy to address those problems?

When an organization adopts a new software development process, it must expect to invest resources in retraining, acquiring appropriate CASE tools, etc. This in itself can frighten away anyone considering making such a change. External factors pose even greater difficulties. The Consortium's member companies are usually contractually obligated to use a specific software development process. Projects cannot expect to use any process that is particularly radical.

Even those organizations wanting to try an advanced reuse process often find that doing so is not quite as easy as they thought. The Consortium has an abstract description of what a reuse-driven software development process should be, but its exact realization depends on both the software domain and the organization using it. Tailoring the process takes both time and experience.

The Consortium must therefore support organizations that wish to adopt reuse slowly as well as those willing to adopt it aggressively.

2.2 Proposed Solution

This paper proposes that an organization seeking to institute a reuse-driven software development process should consider not a single process, but a family of processes. The term "family" is used with a meaning analogous to how it is used to describe software components: a family of processes is a set of processes that are sufficiently similar that it is worthwhile to understand their common properties before considering the special properties of individual instances.[1] They differ in ways that suit particular family members to projects and organizations at particular stages of reuse capability.

The common properties ("commonalities") are what an organization expects to remain constant about its process, independent of its reuse objectives. Examples of such properties include splitting software development into domain engineering and application engineering, or use of particular methods, tools, and technology (whether because of recognition of investment or belief in their efficacy).

The family's variabilities are expectations of how software development will change as the organization improves its ability to reuse software. For example, an organization might want eventually to reuse entire software subsystems or complete chapters of requirements documents; such a capability would obviate, or at least substantially alter, many process activities concerned with creation, integration, and validation of the work products a project produces. However, if an organization does not yet possess the ability to do such large-scale reuse, it might be content to start by reusing smaller entities (individual functions and paragraphs) which would not affect the process nearly so much. Other examples of how reuse influences process are the effect on customer interaction activities (customers do not need as much formal involvement if they are confident that the contractor is using previously-certified work products) and the activities associated with domain engineering. (Domain engineering's importance in an organization correlates to how effectively it assists projects in that organization. As an organization achieves its reuse goals, the process followed by domain engineers encompasses more activities that directly relate to helping all projects in an organization fulfill their contracts.)

The Consortium is developing a process family called Synthesis [6, 7]. The rest of Section 2 is an illustration of our proposed solution using Synthesis. Section 2.2.1 presents some of the commonalities and variabilities of the Synthesis process family. Sections 2.2.2 and 2.2.3 show how the variabilities are resolved in two of the family members—one member being an "advanced" process incorporating reuse, the other a more traditional process. Section 2.3 states what we believe are the benefits of the family-oriented approach.

2.2.1 Characteristics of Synthesis Family Members

The following are some of the characteristics common to all members of the Synthesis process family:

- A Synthesis process involves an interaction of a domain engineering group with one or more application engineering projects. Each application engineering project has a customer and an

[1] This is an adaptation of the definition for a program family given by Dijkstra [5].

obligation to build software satisfying that customer's requirements. The domain engineering group is responsible for assuring that the application engineering project(s) with which it interacts are able (through reuse) to deliver software to their customers in the most effective manner practical.

- Domain engineers formalize a domain as a family of products. A product is composed of a set of work products (requirements, designs, implementations, etc.). Here, "formalize" means that the ways in which the products vary are explicitly and precisely defined.
- Application engineers build work products by resolving the variations identified during domain analysis. These variations are presented to the application engineers as engineering decisions whose resolution requires their knowledge and experience of the problem at hand. The result of this decision-resolution is an "application model."
- Application engineers build work products through mechanical adaptation of components based on information obtained from the application model.

Now consider three variations among family members:

1. The degree to which the work products form an integrated product family. They might be fully integrated, or they might be very loosely integrated.
2. Whether the application engineering process is based on the expectations and requirements of the customer, or on considerations as to how software can most effectively be developed within the domain and within the organization.
3. Whether domain engineering sees its objective as supporting the long-term needs of an entire business-area organization or supporting a particular set of current or planned projects.

2.2.2 "Leveraged" Synthesis

In leveraged Synthesis, the variations are resolved as follows:

- (Variation 1) The product family is highly integrated. Domain engineers understand the relationships between the variations among requirements documents in the domain and the variations among reusable code components. If they can describe their customer's needs in terms of variations at the requirements level, they can also describe variations among reusable code components. A single application model then describes all the work products a project needs to fulfill its contractual obligations.
- (Variation 2) The application engineering process is optimized for the domain. The process might call for the use of specific analytical techniques, for instance. Reuse helps achieve more radical innovation: if enough reusable components are present, then entire process steps can be skipped. Recall that Synthesis processes rely on mechanical adaptation of reusable components. If developers can assemble a design document by selecting and mechanically adapting fragments of existing designs, they do not need to perform the design step.
- (Variation 3) Domain engineering is a strategic component of a business-area organization's ability to develop software within a domain. It determines and defines the optimal application engineering process.

Leveraged Synthesis is an example of an advanced reuse process. It is the vision of a reuse process the Consortium presents to its member companies, many of whom are eager to try it. Unfortunately, most organizations today have trouble institutionalizing leveraged Synthesis. The following are some reasons why:

- Domain engineers must possess a thorough understanding of a domain to integrate the work product families that comprise it. Most organizations do not have such expertise unless they have developed many systems in a domain. Actually, organizations do tend to possess the requisite knowledge, but are not willing to take the time to do a systematic domain analysis (a lengthy task) and therefore cannot formalize a domain in terms of variabilities among family members.
- Customers generally have their own ideas about what makes a software process acceptable. These ideas will be based on processes that have been shown to be historically acceptable (i.e., waterfall models) rather than optimal. The customer may mandate use of such a process as a condition for signing any contracts.
- Domain engineers need time to provide a useful domain implementation. Ongoing projects do not have much opportunity to benefit from reuse. Management, however, often wants immediate results.

This description is somewhat simplified, but it illustrates why an organization might not be able to perform leveraged Synthesis.

2.2.3 "Opportunistic" Synthesis

Leveraged Synthesis was the original Synthesis process. As we began to appreciate the difficulties in introducing it, we conceived of Synthesis as a process family, with the commonalities and variations mentioned above. We defined a new family member, called opportunistic Synthesis (because reuse is based on whatever opportunities arise; domain engineers do not attempt to gain leverage for future projects). In this model, the variations are resolved as follows:

- (Variation 1) Domain engineers formalize variations among members of work product families, rather than complete product families. Work product families are not integrated. A domain is a collection of whatever work product families are necessary.
- (Variation 2) The application engineering process is based on whatever process application engineering uses.
- (Variation 3) Domain engineering is responsible for assuring that a particular application engineering project can benefit from reuse. Domain engineering's goal is always to support upcoming phases of application engineering. Suppose application engineering uses a waterfall process. During system design, domain engineers would be analyzing existing software requirements documents that application engineers might be able to reuse during the software requirements phase. During software requirements, domain engineers would analyze software designs. During software design, they would be analyzing existing code.

Opportunistic Synthesis offers less long-term gain than leveraged Synthesis. It does not encourage rewriting code to make it reusable, or devoting resources to create new reusable components. It does not encourage innovative new software processes. Instead, it allows projects to maintain, more or less, their current software development status quo. This, of course, is an advantage for many projects today. Opportunistic Synthesis gives immediate, tangible results, with minimal investment and low risk. We believe that opportunistic Synthesis would be acceptable to many managers who might find leveraged Synthesis (or most other reuse technologies) interesting but too risky.

2.3 Benefits

The advantage of the family-oriented approach lies in how it facilitates transition from current practice to one's ultimate reuse goals. The family's commonalities help people understand how reuse fits into the software process. Because these properties are invariant, people can expect certain investments in reuse to have long-term rewards. In Synthesis, for example, where performing domain engineering is a commonality, the process and goals of domain engineering may vary between family members, but the domain knowledge that engineers accumulate is always of benefit to current and future projects.

The family's variabilities make reuse amenable to situations from current practice to ultimate reuse goals. This is important for several reasons. First, variabilities help people transition between family members: they describe exactly what is being changed, thus highlighting the only new principles that must be mastered. Second, variabilities help organizations plan a long-term strategy to improve reuse capability: they help organizations evaluate and compare the difficulty, relative to their own environment, of using any two family members. Third, variabilities help organizations manage the differences that individual projects may encounter. Differences in personnel experience, or in customer requirements, may force two projects within the same organization to use different processes. An understanding of the variabilities will help each project choose the process that best fits its needs.

3 Comparison

Synthesis is a reuse-driven software development process. The software development process is only one dimension of the Consortium's process for reuse adoption [8]. Others include management, application development, and asset development. Each of these dimensions has its own variations. An organization adopting reuse must be prepared to resolve many more variations than have been discussed here, many of which are not concerned with software development processes. In other words, the family-oriented approach can be used beyond software development process. However, we still have much work to do in determining the most appropriate variations among family members.

At the 1992 workshop on software reuse, Mark Simos presented work towards an industry-wide consensus reuse process model [9]. The work presented here has been more specific than his in certain aspects. This is because it reflects the concerns of our member companies. He envisioned a domain-independent reuse process model that would be applicable in government, commercial, and public-domain organizations. Our needs are not quite so general; thus we are able to define a reuse process tailored to the commercial and government worlds, and incorporating domain-specific enhancements where helpful.

Our model is also broader than Simos' in some areas. For instance, he proposed that the process should only address aspects of process that relate to reuse. This is true of opportunistic Synthesis, where the application engineers dictate the process; it is the responsibility of domain engineers to determine those parts of application engineering that can benefit from reuse, and to influence no others. In leveraged Synthesis, however, reuse drives what application engineering may do, rather than the other way around. It is therefore hard to see what aspects of the process are not addressed by reuse.

This should not be taken to mean that Synthesis is incompatible with Simos' work. Quite the contrary, a process family is a way of organizing elements of a model such that the right process can be defined for any given situation. Whether families are sufficiently expressive to allow this is a topic we are actively pursuing.

References

[1] B. Joos, "So Much for Motherhood, Apple Pie and Reuse," in *Proceedings of the WISR 5th Annual Workshop on Software Reuse*, (Palo Alto, California), 1992.

[2] R. Erickson, "Software Reuse Adoption: Some Practical Issues," in *Proceedings of the WISR 5th Annual Workshop on Software Reuse*, (Palo Alto, California), 1992.

[3] P. Collins, "Considering Corporate Culture in Institutionalizing Reuse," in *Proceedings of the WISR 5th Annual Workshop on Software Reuse*, (Palo Alto, California), 1992.

[4] T. Davis, "Toward a Reuse Maturity Model," in *Proceedings of the WISR 5th Annual Workshop on Software Reuse*, (Palo Alto, California), 1992.

[5] E. Dijkstra, "Notes on Structured Programming," in *Structured Programming* (O. Dahl, E. Dijkstra, and C. Hoare, eds.), pp. 1–82, Academic Press, 1972.

[6] "Domain Engineering Guidebook," Tech. Rep. SPC-92019-CMC, Software Productivity Consortium, Herndon, Virginia, 1992.

[7] S. Wartik and R. Prieto-Díaz, "Criteria for Comparing Reuse-Oriented Domain Analysis Approaches," *International Journal of Software Engineering and Knowledge Engineering*, vol. 2, no. 3, pp. 403–431, 1992.

[8] "Reuse Adoption Guidebook," Tech. Rep. SPC-92051-CMC, Software Productivity Consortium, Herndon, Virginia, 1992.

[9] M. Simos, "Towards and Industry-Wide Consensus Reuse Process Model," in *Proceedings of the WISR 5th Annual Workshop on Software Reuse*, (Palo Alto, California), 1992.

4 Biography

Steven Wartik is a Senior Member of Technical Staff at the Software Productivity Consortium. His research interests are in domain analysis, software processes, and the relationship between the two. He received his B.S. in 1977 in Computer Science from the Pennsylvania State University, and his Ph.D. in 1984 from the University of California at Santa Barbara. From 1981 to 1984, he worked on the Software Productivity Project at TRW. From 1984 to 1988, he was an assistant professor of Computer Science at the University of Virginia.