Session 5: Key Techniques and Process Aspects for Product Line Development

Nancy S. Staudenmayer

Sloan School*
Massachusetts Institute of Technology
Boston MA 02114

Dewayne E. Perry

Software Production Research
Bell Laboratories
Murray Hill, NJ 07974

Elisabetta Di Nitto began the session as the keynote speaker by posing the questions 'What are the key techniques and process aspects that we specifically need to develop product lines?' and 'What is a product line/family?' She noted that for the latter we do not get any help from ISO 9000-3, CMM, Bootstrap, the Software Engineering Reference Book, or the IEEE Software Engineering Standards.

The following definitions were offered to begin the discussions:

- *Product Family*: a set of products offering the same basic functionalities with some significant variations.

- *Product Line*: a set of products each of them offering complementary features, and that are conceived, designed, and implemented to jointly operate in supporting users' activities.

With these definitions, Microsoft would be considered a product line, whereas Eudora would be considered a product family.

From various position and other papers, Di Nitto extracted the following definitions of product lines/families:

- It is a set of assets which support a family of software products and/or processes within a given domain [Boehm].

- A software product line (family) can be viewed as a collection of products that are similar in some important respect yet systematically different in others (for example, successive revisions of a single application, versions of an application for different host platforms, versions with varying features) [Sutton & Osterweil].

- A collection of software products that address a common set of system requirements, and are organized around a specific business activity [CMU/SEI-95-SR-024].

- Programs inevitably exist in many versions (products). A rational design process should address the design of all versions (product family) [Parnas]

The following techniques are often associated with product line/family development:

- software reuse
- domain analysis
- software architecture
- platforms
- generation
- customization
- process reuse
- process federations and cooperation
- reverse engineering and rearchitecturing
- configuration management

Which ones have direct effects on product line/family development? Which ones can be reasonably considered orthogonal? Do product lines have different requirements from product families?

Di Nitto raised a number of questions to consider in the relationship between product lines and software architectures.

- What is the relationship among design for reuse, software architecture development and product line/family development?

- What is the role of domain analysis? What activities are involve?

- What is the role of domain specific software architecture with respect to product lines?

- What are the advantages of using domain specific languages and when can they be used successfully?

Perry raised the issue of doing a generic architecture versus doing a specific architecture. For example, there were two different products doing virtually the same thing but with different implementations. Can we come up with a generic architecture that covers both products? An interesting issue in the context is what to do with distribution — given that the products may be either centralized or distributed, can we define an architecture that is distribution free and hence applicable to the entire range of implementations? Software architecture for a product line should involve/enable ways of developing the architectures for different products.

Balzer pointed out that we need to identify what is different between the products. Architecture is one example of what might be different. What are the other differences? We move from reasoning about an instance to reasoning about types of instances and thus characterize the space of the product line. The intellectual problem is the characterizing of that domain space and how then to fill it easily.

Is this just good engineering? This question was raised by Alex Wolf. We identify the points of likely changes and design and develop accordingly. How do we distinguish between this and product line development? Is it just a matter of knowing the points of variability a priori? Tully continued this point by asking the question: because of the harder problems with product lines, do we just need harder methods, or do we need different methods and processes? Perry suggested that it is both harder than what we already do, there are more drivers on the process (for example: customer requirements, domain requirements, business constraints, project constraints), and there are new kinds of problems. Di Nitto, on the other hand, agreed with Wolf: there is not much difference between doing a rational process and making a product line.

Boehm raised another issue about the critical set of process differences that exists because the development of a product line is primarily a business decision. The greater the domain, the greater is the opportunity for leverage, but the harder it is to develop commonality.

Good design at the product line level requires more than just good engineering, according to Bandinelli, because it involves coordination of multiple processes (for example, various product processes plus the product line process). the domain can be a business decision or it can be defined at the solution level — for example, solving several problems with a common solution. With multiple business needs we need to anticipate or define their commonality and synchronize these concerns over time.

Balzer asked what difference does it make if you are operating in a domain space or an instance? The domain analysis has to do with the definition of the general space you are operating in. The solution space consists of the architectures, designs and components. We have been thinking about product lines as if there were new development. In practice they are defined ad hoc — there is a legacy factor and you try to bring unity to a bunch of disparate products and processes.

Di Nitto resumed her questions about product lines and various other aspects of the development process. Specifically, what is the relationship of configuration management to product lines/families?

- What are the requirements for product lines/families on configuration management?

- Do they need to be specific for product lines/families?

Configuration management as defined by Tichy is the management of different versions of the same product. Estublier claimed that it is very hard to forecast the evolution of a product family because the necessity of evolving and adapting to technological possibilities and market pressures is very unpredictable. It is not possible to forecast an architecture for a product line lasting 40 years. Conradi pointed out that we can design for the future, but that this involves tradeoffs that chose between various possibilities.

Madhavji noted that in looking at 39 projects at a consulting company they often were not able to do a top-down approach and often needed to adapt both the products and processes as they evolved. With respect to generic architectures: one often needed to experiment and try out several different approaches in order to decide on what level was most appropriate. This was a fundamental different situation that simply producing a product.

The issue of whether there is any difference between a product line and a very large, very complex product that exists for a long time was raised again by Tully. Hollenbach pointed out a study by Galerno that consisted of a domain analysis of domain analysis techniques that addressed the question of this difference from good engineering.

Balzer found similarities among the 12 techniques, but asked whether they were all good engineering? Hollenbach also noted their similarity but that some of these techniques only work when you have complete knowledge of a domain. For example, Kellner noted you need a lot of understanding before you can define and use a domain specific language. To understand the implications for the associated processes you need a very rich model of the products.

With respect to domains, Perry noted that we tend to think of a system as addressing one domain when in fact we have multiple domains. Moreover, we ought to be thinking in terms of the problem domain for our product lines, because it is primarily a business venture. The systems we build are built in the solution domain.

Votta asked whether the sources of variability have different implications for process? We are better at forecasting technological needs than user needs. Derniame offered an object model in which we have structural knowledge, behavioral knowledge and genetic knowledge to help understand the variations in the products and processes. Kellner asked whether we needed special capabilities because of the special characteristics of product lines? The process clearly differs from that for a single product. What does this imply for capabilities?

Di Nitto summarized the discussion up to this point.

- There is a difference between a software architecture for a generic architecture and a product line
- Domain analysis is useful for product development but we must know or have experience with the domain
- We cannot forecast all changes to be made without good domain knowledge
- We may start with one domain and then work towards the unknown

Balzer proposed a spectrum from weak product lines in which you have reuse to strong product lines that go further than reuse. Boehm noted that sometimes technology comes along that creates solutions that did not exist before.

As the keynoter, Di Nitto then redirected the discussion to consider the problems of product lines in relationship to reverse engineering and rearchitecturing.

- Can a product line be developed a posteriori?
- What is the relationship between product lines/families and reverse engineering and reachitecturing?
- When are reverse engineering and rearchitecturing cost effect in product line/family development?
- Can a re-engineered component be reused without reusing the architecture from where it was derived?

Re-engineering is fundamental to the issue of product lines. According to Balzer, you must have an experience base to draw upon. You make business decisions to leverage off synergy. Which way you do it (from scratch or by re-engineering or reverse engineering) is irrelevant and depends on the particulars of the situation. But either way, product lines are experience based.

Heineman noted that once you have made the decision, you need two levels of processes: one at the product level and one between the products.

Martinez noted that in her experience at Motorola they did not so much plan a product line as needed to merge and disentangle existing products. Motorola spends about 10% of resources on constant restructuring. Osterweil claimed that it is a set of consistency constraints (that relate the artifacts together) that is the basis for the constant shaping of the experience base. We need to operationalize and make explicit these relationships (which now is implicit in peoples heads).

Estublier claimed that it is almost all a posteriori. We re-architect and redesign as a key part of product improvement. Di Nitto pointed out that this time and effort for a generic and flexible architecture is often at odds with the need to get a product out in a timely manner. Boehm noted the importance of shared assumptions, particularly with respect to interfaces.

The keynote questions then continued concerning product lines and generation and product lines and software reuse.

- How can generation be used to support the development of product lines/families?
- Is reuse of existing components useful in product line/family development?
- Is it specific to this kind of development or is it an orthogonal issue?
- What is a software asset?
- When can a software asset be reused across a product line?
- To what extent does the granularity of software assets influence the product line development process?
- how should we use design for reuse techniques in product line development?

- What is the relationship between customization and product line/family development?
- Is customization related with reuse?

Martinez provided an overview of software development with reuse as practiced at Motorola illustrating both the development cycle and the management cycle.

- Mkt reqs $\rightsquigarrow$ System reqs $\rightsquigarrow$ System specs $\rightsquigarrow$ Dev* $\rightsquigarrow$ Test* $\rightsquigarrow$ Deploy
- Product line/product release planning $\rightsquigarrow$ Project mgmt $\rightsquigarrow$ Field mgmt

Elements with an * indicate where some teams own common software and other teams own specific software.

Wolf asked if the people who owned common assets were dedicated to that job? Martinez answered 'yes' and the listed additional process steps and considerations.

- define product line strategy requirements
- how do new product requirements impact the product line now and in the future?
- time of product releases and restructuring
- how to manage organizational accountabilities?
- impact on system resources
- change control

All this has resulted in a drive towards commonality and modularity. There had been too many experiences of just reusing code. That saved upfront but did not save in the long term. The common part is about 60-70%. There is additionally a 10% investment on maintenance, update and restructuring on the base (this is viewed as preventive maintenance).

The successful cases of product lines within Motorola were those that started as single products and then grew an architecture for a product line out of those initial products. Moreover, since everything we do is in the wireless communications domain, we do not talk about the domain.

Balzer reported on their experience using generators for product lines and how it affected the development process. Using four different domains and their associated generators, the development process is very much a prototype oriented process: for each new prototype you basically write a new specification. Further, they started with the normal case to get a running version of it and then added in the special cases incrementally. This approach works especially well when there exists a technique that automates the building of the system. This immediacy provided by the prototype also provides valuable feedback in the development process. In addition it is important to add visualization techniques to show what is happening in the product. An important lesson here is that generative languages usually do not handle a large amount of variability. Thus we need the technology and tools for people specifying separate specifications of different aspects of the same product. We also then need technology to compose and analyze and a model manager to check for inconsistencies and incompletenesses.

Conradi reported on the Reboot Project which resulted in identifying 4 reuse processes and divided the various issues into technical issues, project level issues and organizational issues. There is a considerable lag in the payoff but the payoff when it comes is considerable. A book is due out - Reuse: a holistic approach from Wiley. The Renaissance project is a reverse engineering of legacy systems project. The method consists of various techniques: scrap and bury, maintain and synthesize, rewrite, re-engineer. 25% of the effort is to be spent on reengineering (5 people) continuously refining assets in a back room activity where reusable assets are discovered.

Estublier reported on their product family that provides software configuration management. It is a family because there are a set of versions for the product. The management part of the product is concerned with change control. We found that the change control process was not well related to architectures or families of products, but was directly related to requirements, reliability, the size of teams, the time to market and further maintenance constraints. The quality constraints included: strong boundaries between activities, different people for different activities (for example developers were different from testers), tool encapsulation, isolation of workspaces, reviews, and clear decision processes. Perry pointed out that there is a growing need for dynamic reconfiguration that requires different kinds of CM support.

Boehm suggested additional product line processes:

- Asset Base Investment
  - domain scoping and modeling
  - domain architecting (interface specifications and shared assumptions)
  - asset generalization (multiple use, certifications, legacy, COTS, adaptation)
- Asset Evolution
  - domain architecture evolution
  - generalized component evolution

    - synchronization of multiple stakeholder processes

- Asset Base Management
    - catalog library management
    - workflow management
    - accountability process
- Asset Composition Evolution
    - evaluation
    - tailoring and generalization
    - problem feedback
    - assimilation in products
    - product line integration

Klinger described Lockheed/Martin's approach to developing domain engineering processes which is a set of activities, products, roles and models. Domain engineers define the domain model which consists of understanding the commonality and variability of the features for the product line, the product model, the process model and the environment model. They are also responsible for the domain architecture. In terms of the general processes, they have not found anything different from general development. The primary difference in the domain modeling approach was in looking at the commonality and variability. This is also where they felt they needed the most help. The applications engineers then used the asset base to build the various products.

An added dimension to this approach comes from the fact that stakeholders often have different processes. Boehm noted that we need support for looking at variability across these processes as well as the products.

Lehman asked what you had to do upfront in producing a product line. He noted that domain analysis and bounding the domain were incremental and evolutionary. One first develops a preliminary definition of the application domain. Then you identify the potentially common functionality and behavior and identify that common functionality with reuse potential. Next you look at the areas of functional and behavioral conflicts (these especially need to be recognized upfront and not in the field). Given this understanding, you are then at the point where you can define an architecture that takes into account both the commonalities and the conflicts and that provides the base of the implementations. This is obviously an idealized vision and in reality the system must be under constant review (especially in rechecking basic assumptions).

Heineman noted that you have existing processes for building products that are fairly autonomous. At some point you decide to make them into a product line. At this point you need to consolidate and synchronize your processes and consider their interdependencies and constraints. Just as in the product you try to identify the core elements that are shared, you need to identify the core process elements across the various processes.

The ROADS project was reported on by Bandinelli as a ESSI project in collaboration with Thompson CSF. There were four experiments in introducing reuse practices. The technologies used were SPC's RSP (reuse-driven software processes) and Reboot. The domains of the experiments were air traffic control (improve time to market), command and control (improve reliability), training systems (reducing costs) and traffic management (improve flexibility and robustness). Given these different objectives for introducing reuse, how do you transition these processes into use and what are the priorities in the transitions? Can we achieve all the objectives within these approaches or are there tradoffs (for example lower costs at the expense of time to market)?

There were three problems encountered. The first was the problem of whether to be organized as technical business units or strategic business units. The second was how to compose the product line engineering teams. The third was how to synchronize product line engineering with product engineering.

Osterweil noted that if one of the aims is to improve quality and increase time to market and that if we know that 50% of development is testing, etc, then why have none of the discussion addressed the problem of testing in product lines. Everyone talks about reusing architecture but not test suites. The latter may represent a greater point of leverage to save time.