February 5, 2005

Difficult problems in Product Line approach that arise in practice

If relevant and possible, address those problems in your presentation (CS6201)

Acronyms and terms:

PL: Product Line

PLA: Product Line Architecture, Product Line baseline [3]

proactive approach to building a PLA: by domain engineering; costly see [2][3]

extractive approach to building a PLA: uses one or more of existing products to build a PLA; "requires lightweight techniques to reuse existing software without much reengineering" [3]

reactive approach to building a PLA: by propagating variant features from products back to PLA; "you analyze, architect, design and implement one or several variant features in each iteration;

product derivation: deriving PL members from PLA during reuse-based application/system engineering

variation point: point at which architecture or code can be changed to address a variant feature --- this is usual definition of 'variation point', e.g., in Jacobson, Griss et al *Software Reuse*. But I am not sure if authors of [1] define it in that way; please help me to figure this out.

1. General problems underlying PL approach: Variant feature combinations, feature dependencies, and complexity

The following problems must be addressed when designing a PLA to facilitate cost-effective derivation of PL members from PLA during reuse-based application/system engineering.

Problem 1. Large number of variant features, feature dependencies and code components (complexity).

Each PL member needs some combination of variant features. The number of possible legal combinations of variant features is huge. Each variant feature is addressed at multiple variation points, and features affect each other. Complexity increases as PLA evolves. Some variants have system-wide impact: to address them, many components must be modified, at times an architecture and component interfaces must be modified. Poor, sub-optimal variability realization techniques contribute to inherent complexity of the problem.

Solution 1: Suppose we try to pre-implement component configurations already customized for reuse (that is, accommodating legal combinations of variant features).

If we try to do so – the number of components grows rapidly. It seems unthinkable to pre-implement components accommodating all legal combinations of variants.

Solution 2: We pre-implement only component configurations accommodating some typical legal combinations of variants. Or we just keep a record of component configurations from product line members implemented and released so far.

According to [1], we may deal with tens of thousands variation points and even larger number of implicit dependencies between variation points.

Problem 2. In view of the above Problem 1 - how to design components for ease of reuse?

The following are more specific questions related to the above problems:

Problem 3. How to achieve traceability (mappings) from variant features to variation points?

Problem 4. How to know all the consequences of choices/changes that occur at multiple variation points? How to document the mutual impact of changes that may occur at multiple variation points, taking into account all the implicit dependencies?

Problem 5. How to achieve "small variant – small change impact" effect?

A common approach is to re-implement and replace any component affected by variant, even if the change is very small. Thousands of component variants may arise over time [1]**Error! Reference source not found.**.

Problem 6. How to limit expert dependency? [1].

Problem 7. How to formally represent variant features and variation points and their dependencies?

How do OO/component-based development approaches and associated techniques such as Software Configuration Management (SCM) address the above problems?

2. A common approach to designing PLA and product derivation from PLA [1]

PLA is given as an architecture shared by PL members, a collection of component versions implemented into some PL members, and configurations of component versions that form PL members (or parts of them). Component versions/configurations are typically stored in a Software Configuration Management (SCM) system.

A typical derivation of a new PL member S from PLA is as follows: developers select configuration of components that "best match" variant features required in S. Having selected such configuration, developers modify its components to fully meet needs of S. Or developers select individual components that "best match" variant features required in S. See [1] for details.

In this case, much customization effort is needed when deriving PL members from PLA "effort is spent on implementing functionality that highly resembles the functionality already implemented in reusable assets or in previous projects" [1]. This quotation sounds like a failure to achieve reuse.

3. Problems arising during product derivation from a PLA

Problem 8. How to design a PLA to provide developers with components that closely match what is needed in a new system being build? How to simplify and make more accurate selecting component configuration for customization during PL member derivation? How to minimize customization effort and, therefore, increase the benefit of reuse? Engineering with reuse should be much easier/cheaper than engineering from scratch. [1].

Problem 9. How to simplify integration of customized components during derivation? Reduce the number of errors that occur in integration testing.

4. Problems arising during evolution of the PLA and evolution of PL members

A PLA evolves in reactive way (in response to new features needed in specific PL members) and in proactive way (independent analysis, domain engineering, generalizations). The problem is that evolution of PLA and PL members may occur independently of each other. Still, it is important to keep (evolving) PL members in sync with (evolving) PLA.

A PL member S-A for customer A is enhanced with new features (or existing features have been modified). It turns out that some of the changes are useful for yet other PL members:

Problem 10. How do we make new features available to the existing PL members that need them (not all of them may need new features)?

Problem 11. How do we bring new features into the PLA so that PL members we build in the future can benefit from them (PLA evolution)?

Problem 12. How do we know which changes (exact locations at which code has been changed) are related to the new features?

Problem 13. How do we trace/propagate changes across PLA and PL members?

Problem 14. How do we maintain visibility of changes related to various new features? variant features?

Suppose we add new feature today (to PLA or to a specific product). How do we know tomorrow which codes have been added/modified because of that new feature? It is important as knowing past changes helps in implementing new changes. Often, new changes are not totally new but follow a pattern of some past changes. The best is not to innovate here but follow the same pattern. But it is difficult if knowledge of changes is not explicit.

Problem 15. Dealing with obsolete variation points (e.g., optional variant is never/always chosen or the same variant is always chosen from some variant alternative)

5. Cost problem

Problem 16. How to minimize the cost and risk of implementing the PL approach?

The initial cost of domain engineering may be prohibitive to adopting a PL approach. To avoid high cost and risk, a PLA is best build incrementally (a reactive approach), by building into a PLA variants that actually arise in PL members [2][3]. On contrary, a proactive approach assumes that we analyze a domain to predict a

possibly wide range of variants and build a PLA based on that. The proactive approach is known to be very expensive.

References

[1] Deelstra, S., Sinnema, M. and Bosch, J. "Experiences in Software Product Families: Problems and Issues during Product Derivation," Proc. Software Product Lines Conference, *SPLC3*, Boston, Aug. 2004, LNCS 3154, Springer-Verlag, pp. 165-182

[2] Krueger, C. "Salion's Experience with a Reactive Software Product Line Approach," *5th Int. Workshop Product Family Engineering PFE5*, 2003, LNCS 3014, Springer-Verlag, pp. 317-322

[3] Krueger, C. "Eliminating the Adoption Barrier," Point-Counter Point Column, in *IEEE Software*, special issue on Initiating Software Product Lines, July/August 2002, pages 28-31 http://www.biglever.com/papers/PointCounterPoint.pdf

[4] Pattersson, Jarzabek "An Industrial Application of a Reuse Technique to a Web Portal Product Line" draft