

Towards a Taxonomy for Software Product Lines

Charles W. Krueger
BigLever Software, Inc., 10500 Laurel Hill Cove
Austin TX 78730 USA
ckrueger@biglever.com

Abstract. Drawing from the growing number of software product line experiences and case studies, this report describes a taxonomy for characterizing different software product line approaches. The taxonomy helps to illuminate the general principles of software product line engineering and the effectiveness of different software product line solutions in different situations.

1 Introduction

Software product line case studies have reported some of the greatest improvements in software engineering productivity, cost, and quality since high-level languages were introduced in the late 1950's[1]. One would expect a rush by the software industry to exploit the competitive advantages offered by software product lines. However, our experience in marketing commercial software product line technology and services at BigLever Software is that most of the software industry is either unaware of the emerging field of software product lines, or if an organization is aware they don't understand how software product lines might be applied in their situation[2][3].

Part of the problem is that case studies and descriptions of product line approaches often take a narrow view of a specific solution without the perspective of a point solution in a broader solution space[4]. Uninitiated readers have reported difficulty to us in making the leap from these point solutions to both the general and specific principles that might apply to their particular situation.

What is missing is a clear and concise taxonomy describing the space of software product line problems and associated solutions. There is sufficient research and experience in the field to start drawing generalizations and identifying how specific solutions and case studies lay as point solutions within a generalized taxonomy.

2 What Should a Taxonomy Illustrate?

The objective of a software product line is to optimize software engineering effectiveness and efficiency by capitalizing on the commonality and managing the variation that exists within a product line of similar software systems. As such, a taxonomy for software product lines should illuminate how well an approach capitalizes on commonality and manages variation.

In order to *capitalize on commonality*, duplication and divergence must be avoided for software artifacts in a software product line. Common artifacts should be consolidated and maintained in a single source. Similar artifacts should be consolidated and maintained in a single source along with the variations explicitly represented.

In order to effectively and efficiently *manage variation*, the complexity and combinatorics associated with software variation must be constrained. Variations should be explicitly and formally represented so that they can be easily viewed, reasoned about, and mechanized.

Therefore, in the taxonomy proposed here, the primary focus is on how a software product line approach avoids duplication and divergence, consolidates commonality, controls complexity and combinatorics, and formally represents and mechanizes variation.

3 Building Blocks of the Taxonomy

The primary distinction between software product line engineering and conventional software engineering is the presence of *variation* in some of the software artifacts. In the early stages of software product line engineering, software artifacts contain variations that represent unbound choices about how a final software product will behave. At some point during the engineering process, decisions are bound for the variations, after which the behavior of the final software product is full specified. This is illustrated in Figure 1.

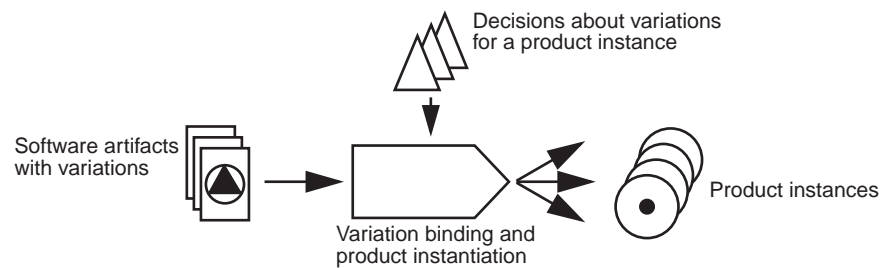


Fig. 1. Binding Decisions about Variation

It is possible to have multiple binding times for variation in a single software product line approach. In this case, some decisions are bound earlier in the process while some decisions are deferred until later in the process. Examples of different binding times and mechanisms include:

- Source reuse time. Specialization, glue, frameworks, binary components
- Development time. Clone-and-own, independent development, design/architecture
- Static code instantiation time. CM branches & labels, directory naming, GEARS
- Build time. Preprocessors, script variants, application generators, templates, aspects
- Package time. Assembling collections of binaries/executables/resources, GEARS
- Customer customizations. Code modifications, glue code, frameworks
- Install time. Install config file. Wizard choices, licensing
- Startup time. Config files, database settings, licensing
- Runtime. Dynamic settings, user preferences

The representation of multiple binding times in the taxonomy is illustrated in Figure 2.

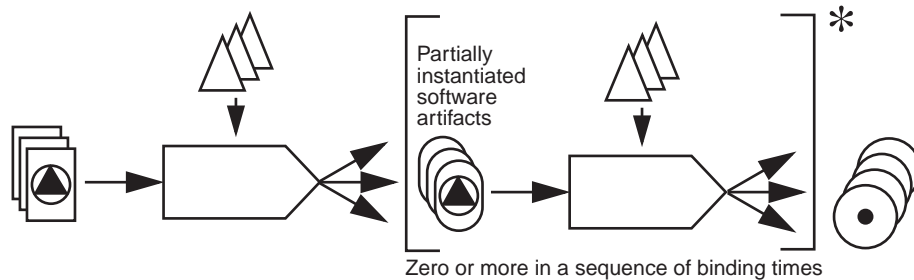


Fig. 2. Multiple Binding Times

As with conventional software engineering, the software artifacts in a software product line are subject to maintenance and evolution. The variation of software artifacts over time – as well as the variation within the space of the product line – is an important factor in the software product line taxonomy[5]. The taxonomy’s representation of variation over time for software product line artifacts is illustrated in Fig. 3.

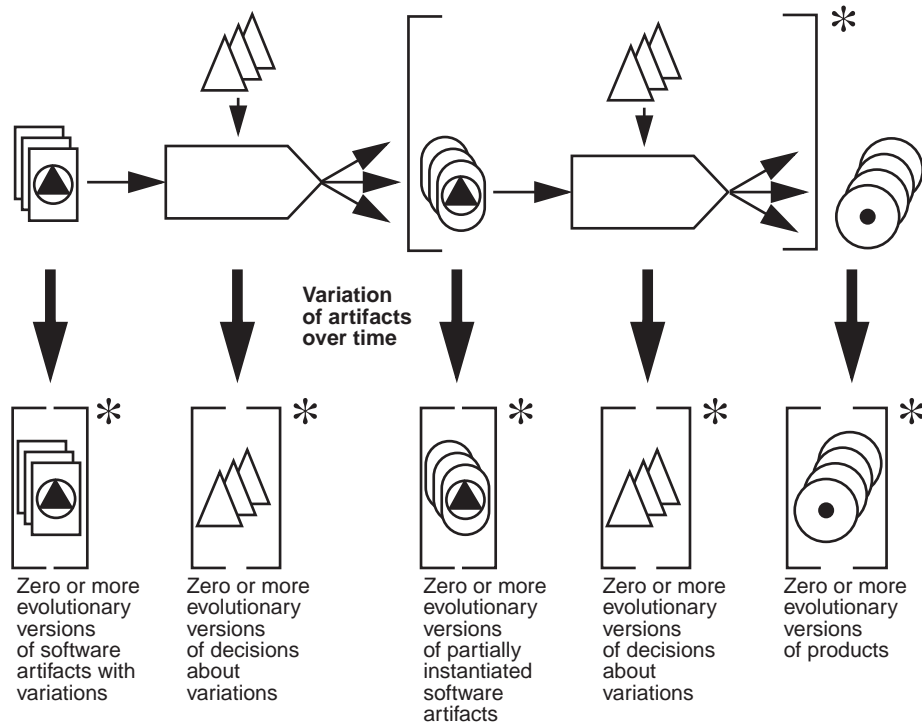


Fig. 3. Evolution of Software Artifacts over Time

Evolutionary changes to an artifact or decision may require that those changes be propagated to the upstream or downstream artifacts in the product line workflow. The possible dependencies that need to be considered include:

- Update paths. A change made to an upstream artifact may need to be reflected in all existing downstream artifacts previously derived from the original artifact.
- Feedback paths. If downstream artifacts can be manually modified, then a change made to a downstream artifact may need to be fed back to the original artifact or artifacts from which the original downstream artifact was derived. When upstream artifacts are modified, update paths may need to be recursively applied (as described in the previous bullet).
- Cross-merge paths. If downstream artifacts can be manually modified, then a change made to a downstream artifact may also need to be reflected in its peers.

Figure 4 illustrates the potential upgrade, feedback, and cross-merge paths in the taxonomy.

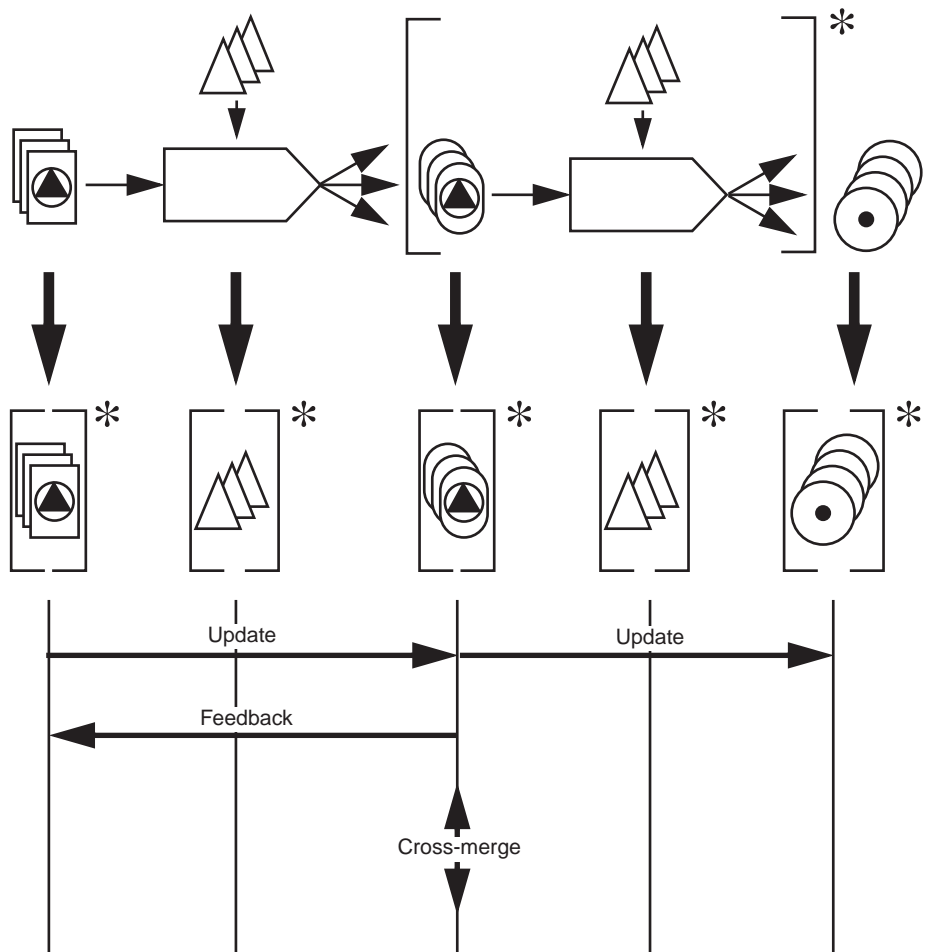


Fig. 4. Taxonomy for Production in Time and Space

3.1 Production

A binding time in the taxonomy represents a step in a software engineering process where fully or partially instantiated products are created from software artifacts that contain variation. This *production* step is a key discriminator between different software product line approaches, so the taxonomy represents information useful in distinguishing among the approaches. Refer to Figure 5 for the following descriptions of information characterizing the production step.

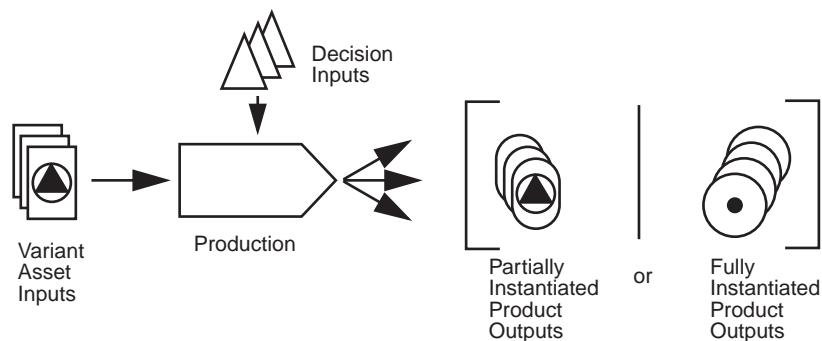


Fig. 5. Production for a Single Binding Time

Production Activity

- Automated versus manual production. The production activity can be fully automated, completely manual, or somewhere in between. Examples of fully automated approaches include application generators, where high level decisions are sufficient to generate product outputs. Examples of completely manual approaches include the textual *production plan*, where software engineers interpret and follow directives in the plan and the decision inputs to tailor, integrate, and provide “glue code” around reusable core variant assets to create product outputs.
- Production roles. Some product line approaches define a separate role for the production activity called *product engineering* (or *application engineering*), distinct from the *domain engineering* role responsible for engineering the variant asset inputs. Other approaches do not distinguish between these two roles. Separate roles are common in approaches with manual production while a single role is more common in approaches with fully automated production.
- Periodic production. As with conventional software engineering, the production step in software product line engineering is typically not a one-shot activity. Products may need to be periodically reproduced to reflect enhancements to the variant asset inputs, decision inputs, and production techniques. The frequency of periodic production may be measured in terms of hours in agile approaches with high degrees of production automation or in terms of year in sequential and highly manual production approaches.

Decision Inputs

- Decision representation. Different formats are utilized to represent decision inputs, ranging from informal language descriptions to formal machine-interpreted languages.
- Decision guidance. Different types of decision making guidance can be provided during the decision process for a particular product, ranging from written heuristics to constraint checking to full-blown expert system guidance. The size and complexity of decision set and the degree of automation desired influence the level of decision guidance.
- Decision role. The decision making role for a particular product may be separate from other engineering roles (e.g., a product marketing role) or it may simply be part of the application engineering or overall production engineering role.
- Replayable decisions. For periodic production of products, the previously made decisions for each product may need to be “replayed” rather than rederived from first principles. Persistent representations and replay mechanisms for decisions vary based on the need for automation and the frequency of periodic production.

Variable Asset Inputs

- Representation. The variable asset inputs (also referred to as *core assets* for the first stage in a multiple binding time sequence) can come in different formats, such as binary executable representations, formats requiring further automated instantiation or translation, or mutable reusable source code that is manually instantiated, modified, tailored, or extended as needed during production.
- Variation mechanisms. Different variation mechanisms are utilized in support of different production approaches. During production, the decision inputs are utilized to instantiate the asset inputs using the variation mechanisms in the assets inputs. A variation mechanism can be as simple as an empty block in source code that an application developer must fill in, or as complex as a translation system that generates source code from high-level requirement specifications.

Product Outputs

- Representation. Similar to the asset inputs, the product outputs from production can take on many different formats: binary or source, mutable or immutable, partially or fully instantiated.
- Partial instantiation. If the product outputs are partially instantiated in one stage of variation binding, the unbound variations become the variable asset inputs for the next stage and can be characterized accordingly.

3.2 Product Line Evolution

The taxonomy characterizes how a product line approach avoids duplication and divergence, consolidates commonality, and controls complexity and combinatorics during the evolution of a product line. This information is illuminated by the character and number of update, feedback, and cross-merge paths required during evolution. Refer to Figure 6 for the following descriptions of information used in the taxonomy to characterize product line evolution.

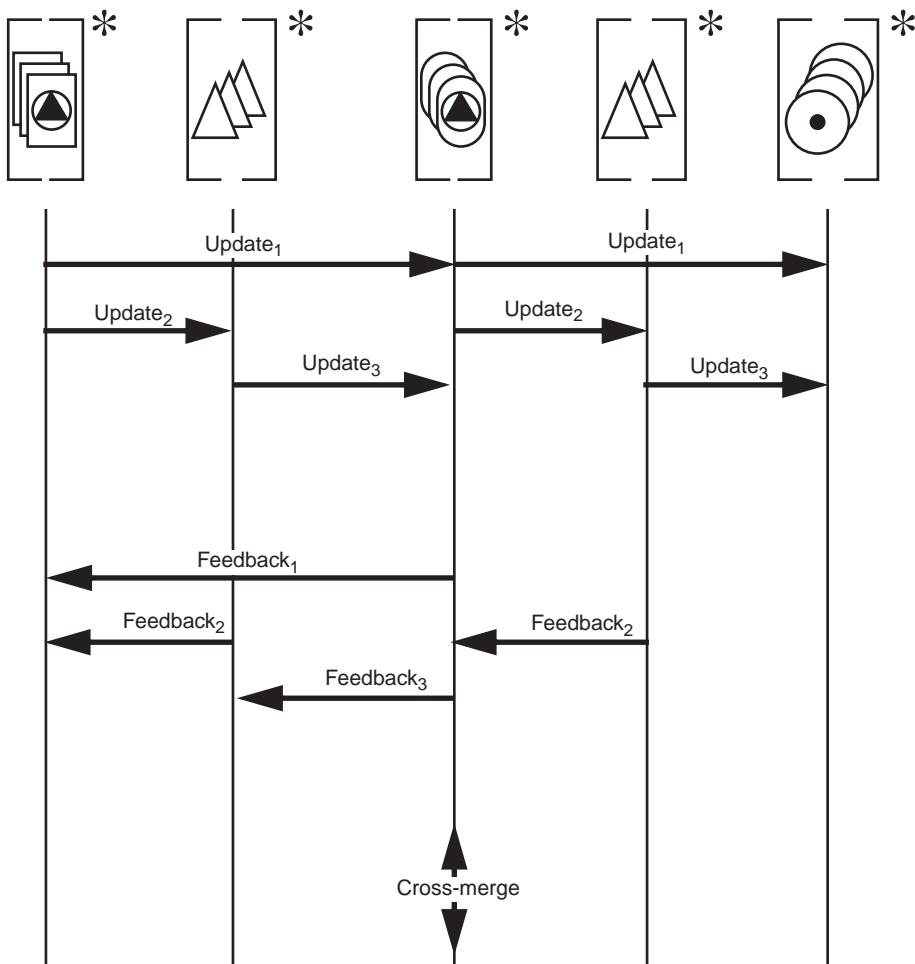


Fig. 6. Update, Feedback and Cross-merge During Evolution

Updates

- Asset-to-product updates (see Update₁ arrows in Figure 6). Updates to products may be required whenever a new version of an asset is created. For every product produced from the original asset, an update operation is required. If the software assets are mutable during production, then the update operation may require a manual merge for each product. If multiple binding times are utilized, updates may need to be applied recursively. If decisions cannot be replayed automatically, decisions and production may have to be manually replayed for each product.
- Asset-to-decision updates (see Update₂ arrows in Figure 6). If modifications to asset inputs result in new or different ways of making decisions, then updates may be required for the decision model, decision mechanisms, and replayable decisions. Some or all products may need to be recursively updated to reflect updates to the decisions.
- Decision-to-product updates (see Update₃ arrows in Figure 6). Any changes to the decisions for a particular product may require an update for the product. Changes to the way that decisions are mapped to products may require all downstream products be updated. If the production process is manual or if manual modifications have been applied to the downstream products, then manual production and merging may be required for each product.

Feedback

- Product-to-asset feedback (see Feedback₁ arrows in Figure 6). If asset inputs are mutable during production, then evolutionary changes to a product may need to be fed back to the originating assets. Resulting changes to asset inputs may require that updates be recursively applied to all other products created from the asset, as described in the previous section.
- Decision-to-asset feedback (see Feedback₂ arrows in Figure 6). Changes to the decision model or mechanism may require feedback to upstream asset inputs to make them consistent with the decisions, usually affecting the variabilities in the asset inputs. This must be done prior to updating the downstream products with the modifications to the decision inputs.
- Product-to-decision feedback (see Feedback₃ arrow in Figure 6). If asset inputs are mutable during production, then evolutionary changes to a product may need to be reflected in the decision model, decision mechanisms, and replayable decisions. Some or all products may need to be recursively updated to reflect updates to the decisions.

Cross-merge

- If asset inputs are mutable during production, then changes to a product may need to also be cross-merged to some or all peer products. This may be an interim solution while a longer term feedback and updates are performed, as described in the previous sections.

4 Scope Management

The taxonomy distinguishes between two methods for managing the scope of a software product line, *proactive* and *reactive*[2][6]. In the case of a pure proactive approach, all of the products needed on the foreseeable horizon are supported by the product line. In the case of a pure reactive approach, only the products needed in the immediate term are supported by the product line. There is, of course, a continuum between the two.

The reactive approach typically requires less upfront effort than the proactive approach since the initial scope coverage is smaller. To implement the same scope coverage, the reactive approach incrementally defers the cost and effort over longer period of time compared to the proactive approach.

It is possible for upstream artifacts in a software product line (e.g., the architecture) to have a more proactive scope than downstream artifacts (e.g., the source code). However, the overall scope of the product line is defined by the scope of final product set that can be produced, tested, and deployed.

5 Transition

There are a variety of approaches for transitioning to, or adopting, a new software product line approach. The taxonomy distinguishes among the following characteristics:

- Source of core asset inputs. The primary distinction is between the *green field* versus *extractive* approach[2]. With the green field approach, the core assets are coded from scratch (i.e., reusing no existing products). With the extractive approach, existing software artifacts from one or more existing products are extracted and reengineered to serve as the core asset inputs for the product line. An interesting special case of the extractive approach is where a single product serves as the core asset baseline, followed by a reactive approach to incrementally extend the scope to include additional products.
- New versus enhancement. Another distinguishing characteristic for the transition to a software product line approach is whether or not a product line already exists. If a product line exists, then the taxonomy described in this document can be used to characterize both the *initial state* of the product line and the *target state* of the product line. The transition can then be characterized in terms of how to move from the initial state characterization to the target state characterization. Note that if the initial state has multiple products managed using even the most primitive, ad hoc, conventional techniques such as clone-and-own or IFDEFs, it is still a product line that can be characterized using the taxonomy.

6 Conclusions

This software product line taxonomy helps to both (1) characterize the space of possible approaches to engineering a software product line and (2) provide a framework for comparing and contrasting different point solutions and the associated trade-offs among them. Ideally this will provide a means to educate novices to the field of software product lines and help identify optimal choices when defining a product line approach.

References

- [1] Clements, P. and Northrop, L., *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2000.
- [2] Krueger, C. *Easing the Transition to Software Mass Customization*. Proceedings of the 4th International Workshop on Product Family Engineering. October 2001. Bilbao, Spain. Springer-Verlag, New York, NY.
- [3] BigLever Software, Inc. Austin, TX. www.biglever.com
- [4] Bosch, J., *Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization*, in Proceedings of Second International Conference, SPLC 2, San Diego, CA, August 2002, pages 257-271
- [5] Krueger, C., *Variation Management for Software Product Lines*, in Proceedings of Second International Conference, SPLC 2, San Diego, CA, August 2002, pages 257-271
- [6] Clements, P. and Krueger, C., *Being Proactive Pays Off/ Eliminating the Adoption Barrier*. IEEE Software, Special Issue of Software Product Lines. July/August 2002, pages 28-31.