

# Variation Management for Software Production Lines<sup>1</sup>

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

**Abstract.** Variation management in a software product line is a multi-dimensional configuration management problem. In addition to the conventional configuration management problem of managing variation over time, software product lines also have the problem of managing variation among the individual products in the domain space. In this paper, we illustrate how to “divide and conquer” the variation management problem into a collection of nine smaller problems and solutions. We also show how to address the nine problems with lightweight solutions that can reduce the risks, costs, and time for establishing and maintaining a software product line.

## 1 Introduction

*Variation management* is the key discriminator between conventional software engineering and software product line engineering. In conventional software engineering, variation management deals with software variation over time and is commonly known as *configuration management*. In software product line engineering, variation management is multi-dimensional. It deals with variation in both time and space. In this context, managing *variation in time* refers to configuration management of the product line software as it varies over time, while managing *variation in space* refers to managing differences among the individual products in the domain space of a product line at any fixed point in time.[1][2][3][4][5][6]

In the same way that configuration management is an essential part of conventional software engineering, variation management is at the core of software product line engineering. However, the potentially complex interactions between variation in time and variation in space can make the design, implementation, and deployment of a variation management solution a daunting task. Practitioners have historically relied on intuition, ad hoc approaches, and pioneering spirit to establish variation management strategies and techniques for their product lines. As a result, the associated risks, costs, and time of establishing and maintaining a software product line approach can be high.[6][7][8]

This paper illustrates how to “divide and conquer” the variation management problem into a collection of nine small, manageable, and interrelated problems. We also show how to address these nine problems with lightweight solutions that can significantly reduce the risks, costs, and time for establishing and maintaining a software product line.

---

1. © Springer-Verlag

Even more interesting is that a complete and consistent variation management solution as defined herein can be used as the fundamental basis for software product line engineering. Elevating variation management to a central role in software product line engineering offers several advantages:

- Back to single-system engineering. Collectively, the common, variant, and infrastructure artifacts in a software product line can be treated as a single *software production line* that evolves over time.
- Lower adoption barrier. Incremental adoption strategies, reuse of existing technology and assets, and lightweight extensions to one-of-a-kind techniques reduce the time, cost, and effort required to adopt a product line approach.[11]
- Flexible product line methodology. Different methods for different business conditions include reuse of legacy assets versus reengineering, proactive versus reactive variation engineering, and refactoring of variation/commonality.[12]
- Formality options. Engineers can choose appropriate levels of formality, including no formal domain analysis, no formal architecture, and no componentization.

## 2 Divide and Conquer Variation Management

The issues of variation management in software product lines can be viewed as an extension to the issues of configuration management in one-of-a-kind systems. This view of variation management is useful because we can reuse the proven technologies and techniques of configuration management, then incrementally extend as needed with technologies and techniques to cover the remaining issues of variation management.

### 2.1 Two Dimensions of Variation Management

Variation management can be “divided” into nine smaller issues, and then “conquered” by addressing each of these sub-problems. Because we have defined variation management as an extension to configuration management, some of the sub-problems will be addressed by conventional configuration management technology and techniques. The division of variation management issues is represented in two dimensions by the nine-cell grid in Table 1.

The columns in the table represent the dimension of *variation types*, while the rows represent the dimension of *granularity of software artifacts*. Each cell in the table represents a variation management issue for a given type of variation (sequential, parallel, spatial) and for a given granularity of artifact (file, component, product).

**Table 1** Two Dimensional View of Variation Management

	<b>Sequential Time</b>	<b>Parallel Time</b>	<b>Domain Space</b>
<b>Files</b>	–	–	–
<b>Components</b>	–	–	–
<b>Products</b>	–	–	–

The first two columns in Table 1, *Sequential Time* and *Parallel Time*, are the types of time-base variation supported by conventional configuration management systems. Sequential time variation is the evolution of an artifact along a single development branch. Parallel time variation is parallel evolution along multiple development branches, which is typically associated with parallel maintenance branches. The third column, *Domain Space*, represents the type of variations that arise to support multiple products in the domain space of a software product line. Combined, these three columns represent the complete set of variation that exists in a software product line.

The rows in Table 1 are the different granularity of artifacts that are supported by conventional configuration management and likewise must be supported by variation management in software product lines. Variations of *Files* are composed to create variations of *Components*, and variations of *Components* are composed to create variations of *Products*.

Modeling variation at the granularity of files may seem counterintuitive at first. However, by using files, components, and products for the artifact granularity in variation management, we have preserved an important property of conventional configuration management – neutrality to architecture, design, and language. That is, neither configuration management nor variation management are concerned with the content of files or the semantics of the file collections used to create a component or product.

## 2.2 Solution Clusters in the Variation Management Grid

Solutions to the nine sub-problems in the variation management grid come from three different technology areas. These three solution technologies and the sub-problems that they address are shown as three clusters in Table 2: *Basic Configuration Management*, *Component Composition*, and *Software Mass Customization*.

*Basic Configuration Management* technology, available in commercial configuration management systems, addresses four of the time-based variation management cells. *Component Composition* technology, either home grown or available in a few commercial configuration management systems, addresses the other two time-based variation issues. *Software Mass Customization* technology, either home grown or commercially available, addresses the three issues of variation in the domain space column.

**Table 2** Variation Management Clusters

	<b>Sequential Time</b>	<b>Parallel Time</b>	<b>Domain Space</b>
<b>Files</b>	<b>Basic Configuration Management</b>		<b>Software Mass Customization</b>
<b>Components</b>			
<b>Products</b>	<b>Component Composition</b>		

### 2.3 Nine Sub-problems of Variation Management

Table 3 illustrates the nine sub-problems of variation management. The clusters from Table 2 are also drawn in the table and are used to organize the follow descriptions of the sub-problems. Details on these problems and their solutions are provided in later sections.

#### Basic Configuration Management

- version management. Versions of files as they change over time.
- branch management. Independent branches of file evolution.
- baseline management. Snapshots of consistent collections of file versions.
- branched baseline management. Independent branches of baseline evolution.

#### Component Composition

- composition management. Snapshots of consistent compositions of component versions.
- branched composition management. Independent branches of component compositions.

#### Software Mass Customization

- variation point management. Variants of files in the domain space.
- customization management. Consistent compositions of common and variant files.
- customization composition management. Compositions of customized components.

**Table 3** Nine Sub-problems of Variation Management

	<b>Sequential Time</b>	<b>Parallel Time</b>	<b>Domain Space</b>
<b>Files</b>	version management	branch management	variation point management
<b>Components</b>	baseline management	branched baseline management	customization management
<b>Products</b>	composition management	branched composition management	customization composition management

### 3 Production Line versus Product Line

The effectiveness of variation management in a software product line depends on the way that software artifacts are structured in the engineering process. We offer the following rule of thumb:

“Variation manage the *production line* rather than the *product line*.”

That is, the artifacts under variation management should include all of the common artifacts, variant artifacts, and product instantiation infrastructure (collectively the production line), but not the individual products that are instantiated (collectively the product line). The individual products should be treated as transient outputs of the production line that can be discarded and re-instantiated as needed.

Figure 1 and Figure 2 illustrate why we emphasize this distinction between product line and production line. Figure 1 illustrates a general approach to variation management, such as the one described in [6], that allows for each product to be refined and independently configuration managed after it is instantiated from the core assets. Although some organizational and business constraints demand this generality (for example, see [9]), there are some undesirable implications of structuring variation management around a product line that should be avoided if possible.

The bold box in Figure 1 encloses all of the artifacts under variation management, including the common and variant artifacts created during domain engineering (i.e., the core assets), the instantiation infrastructure used to assemble common and variant artifacts into a product instance, and all of the product instances that can be further refined after instantiation via independent product development.

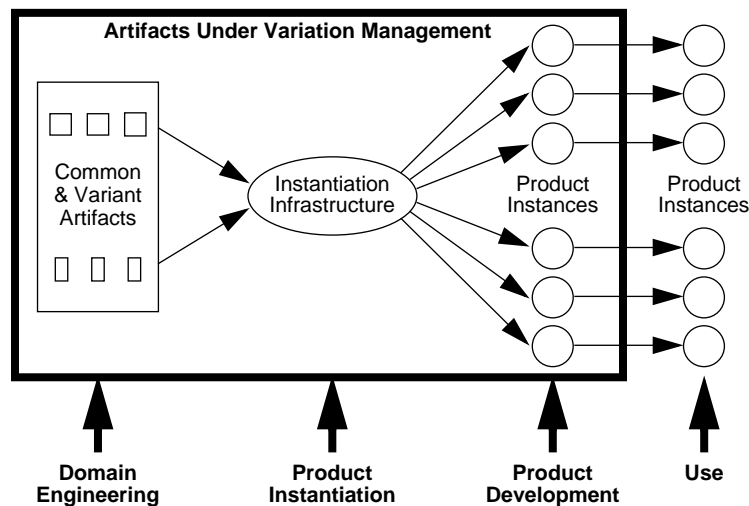
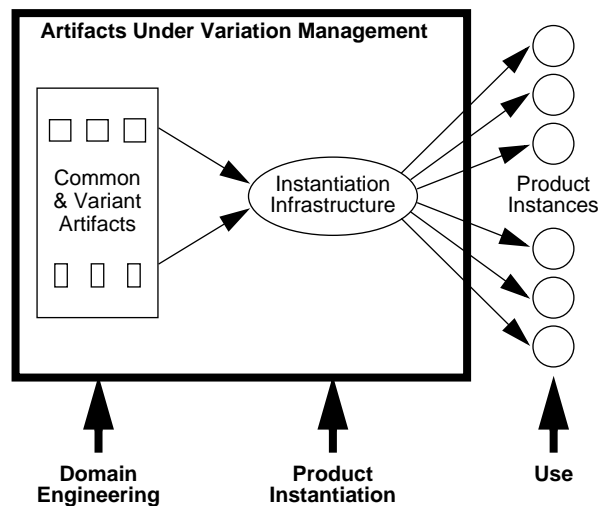


Figure 1 Variation Management of a Product Line

From a variation management perspective, the structure and process in Figure 1 has several negative properties.

- Each product instance becomes a clone that takes on a life of its own after instantiation as it undergoes further refinement via product development. For example, if there are 100 product instances, then there are 100 independent, divergent product evolutions in variation management.
- Modifications made to a product instance during product development are not reflected back into the common and variant artifacts (core assets), nor are they reflected into other product instances. A separate engineering activity is required to refactor changes made to a product instance back into the core assets and other products.
- Enhancements and fixes applied to the common and variant artifacts cannot be directly applied to the product instances since product development may have introduced incompatible changes to some of the product instances.

In contrast, Figure 2 shows variation management structured around a production line. In this case, only the common and variant artifacts and the instantiation infrastructure are variation managed. The product instances are outside of variation management and are used directly without any further modification via application engineering.



**Figure 2** Variation Management of Production Line

From a variation management perspective, the structure and process in Figure 2 offers several distinct advantages over that of Figure 1.

- There is only a single “product” under variation management – the production line. This eliminates the potential for large numbers of independent product evolution branches that have to be maintained under variation management.
- All modifications are made directly to the common and variant artifacts. This eliminates the need to make duplicate changes to product instances and core assets.
- Enhancements and fixes applied to the common and variant artifacts are immediately and directly available to all product instances.

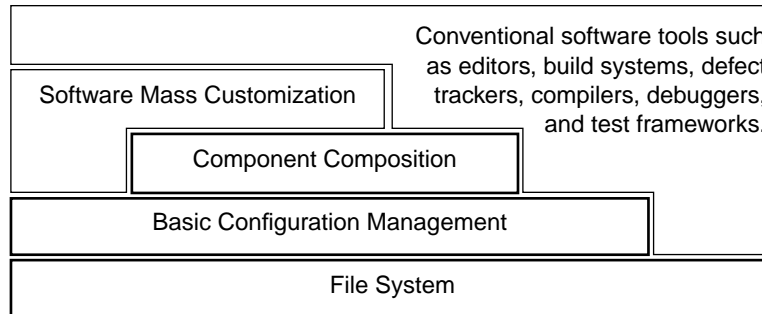
Automotive production lines offer an insightful analogy to the production line approach to software variation management. Mass customization of automobiles – that is, individualized automobiles produced with the efficiency of mass production – is made possible by focusing all of the engineering efforts on the common and variant components plus the automotive “instantiation infrastructure”, not on engineering the individual automobile instances. Similarly, we can achieve true software mass customization efficiencies by taking an analogous production line approach to variation management.

In order to take advantage of variation management benefits of the production line approach illustrated in Figure 2, some of the generality implied in the product line approach of Figure 1 must be sacrificed. Most importantly, the production line approach in Figure 2 implies a single, tightly coordinated engineering effort in order to engineer the common and variant artifacts and the instantiation infrastructure. This does not support the possibility of one organization developing the core assets and a separate organization developing the products based on core assets, such as described in [6]. Nor does it support advanced recursive product line approaches such as described in [9], where multiple product lines can be derived from a single platform of core assets. In these cases, the automobile production analogy resembles going to an automotive parts supplier to get all of the parts for a particular automobile and then performing “product development” to complete the construction.

## 4 Technology Layers for Variation Management

Section 2.2 described three different solutions technologies used for variation management: basic configuration management, component composition, and software mass customization. Figure 3 illustrates how these technologies relate to each other and to other software engineering technology. In the diagram, when two layers are directly adjacent to each other, the layer above is dependent on the services of the layer below.

Basic configuration management technology relies on the underlying operating system file system. Keeping the entire production line under variation management means that the component composition and software mass customization layers always depend on basic configuration management to supply versioned artifacts for a complete and consistent collection of versions at a fixed point in time. Software mass customization operates on variations at the file and component level to produce customized products.



**Figure 3** Technology Layers for Variation Management

## 5 Basic Configuration Management

As illustrated in Section 2.2 and Section 2.3, publicly or commercially available configuration management technology is suitable for solving four of the nine sub-problems in variation management. These are the four sub-problems associated with managing sequential and parallel time variation for files and components.

In Section 3 we demonstrated that all artifacts in the production line should be under variation management. This is essential for *reproducibility*, where any product in the product line can be reproduced, exactly as it was at some point in the past, using only artifacts under variation management. At the basic configuration management level, that means all common files, all variant files, and all instantiation infrastructure files must be placed under basic configuration management.

Following are descriptions of the four sub-problems and the way that they can be solved by basic configuration management technology.

Version management. Sequential versions of files must be managed over time.

**Solution:** *Check-out* and *check-in* operations create sequential file versions on the main trunk.

Branch management. Independent branches of file evolution must be supported.

**Solution:** Parallel *file branches* created off of the main trunk or other file branches.

Baseline management. It must be possible to create a baseline of file versions for all files in a component. This provides a reproducible snapshot of the state of a component at a given point in time.

**Solution:** *Labels* create a persistent named “slice” of file versions.

Branched baseline management. Independent branches of component baselines must be supported.

**Solution:** Naming conventions for *labels* used to create baselines can indicate branching in a baseline. This is typically done in conjunction with naming conventions for *branch* names to indicate which branch the label is associated with.

## 6 Component Composition

Component composition technology as described in Section 2.2 and Section 2.3 will address two of the nine sub-problems in variation management. These are the two sub-problems associated with managing sequential and parallel time variation for the composition of components into products. Some commercial configuration management systems provide this capability, though straightforward home grown approaches can easily be implemented.

Component composition issues arise when products in a product line are assembled out of independent components and when the components can evolve independently of each other. That is, if components evolve together as part of a product baseline rather than evolve with independent component baselines, then component composition support is not necessary. Components in the latter case are simply modular partitions, or subsystems, within a whole software product.

There are two general approaches to assembling component versions for the construction of a product. The first approach, termed “the shelf”, maintains all the versions of all the components on a reusable parts “shelf”. To construct a product you go to the shelf and select one version of each component. With this approach, guidance is needed in determining consistent compositions of component versions since many of the version combinations will be incompatible.

The second approach, termed “the context”, maintains an evolving “context” of component versions. In contrast to the “shelf” approach, where all component versions are maintained as peers on the shelf, a “context” represents only one possible composition of component versions. A context typically evolves incrementally as new component versions are developed and integrated into the context.

Following are descriptions of the two sub-problems and the way that they can be solved by “shelf” or “context” component composition technology.

Composition management. Sequential versioning of the valid component compositions, from component versions into a product version, must be managed as the valid compositions change over time.

**Solution:** The combinatorics of component composition can be a challenge. In a product with 10 components, where each component has 4 baselines, there are over one million possible component compositions, most of which may be invalid combinations. The “shelf” approach may require an expert system, with significant overhead associated with maintaining the composition rules. To support sequential versioning of composition, the composition engine and rules should be maintained in configuration management.

The “context” approach is much simpler and should be used when possible. The “context” can be implemented as a simple list in a single file that tracks the “latest and greatest” composition of components for a product. This list can evolve as a new sequential file version each time a new component baseline is released, validated, and entered into the context list for the product.

Branched composition management. Independent branches of component compositions must be supported.

**Solution:** With the “shelf” approach, the composition engine and rules can be branched using the underlying configuration management branching mechanism. Similarly, with the “context” approach, the context file can be branched using configuration management branches.

## 7 Software Mass Customization

The remaining three variation management sub-problems described in Section 2.2 and Section 2.3 deal with variations that come from the domain space of the product line. These are variations of files, components, and products that exist at a fixed point in time rather than variations over time. They are addressed by *software mass customization* technology and techniques. One commercial system (from the company of the author of this paper) provides this capability, and home grown approaches are also possible.[10]

Software mass customization introduces some new abstractions to variation management.[12][13][14] First is a means of expressing the dimensions of variation that exist in the domain. Second is the ability to identify points of variation in the software artifacts for the production line and how they depend on the dimensions of variation. Third is a mechanism to assemble common and variant artifacts into customized components based on the dimensions of variation and the variation points within the components. Fourth is a mechanism to compose customized components into customized products.

All of the software mass customization infrastructure should be maintained under configuration management, along with the common and variant artifacts used to compose components and products. This assures reproducibility of the customized components and customized products in the domain space, from any baseline in time.

Following are descriptions of the three sub-problems and the way that they can be solved by software mass customization technology. “Standard” software mass customization conventions do not exist in the same way that configuration management or component composition conventions do. Therefore we describe solutions based on the conventions we have adopted as a result of our research on software mass customization.

Variation point management. Points of variation in the software artifacts of the production line must be managed, including the different ways that they can be instantiated and how the instantiation depends on the dimensions of variation in the domain space.

**Solution:** Variation points are implemented to include a collection of file variants and logic for selecting among the variants based on dimensions of variation in the domain space. Given a point in the domain space, a file variant can be identified for each variation point.

**Customization management.** A customized component is created by assembling the component's common and variant artifacts and then instantiating any variation points. Reproducible assembly and instantiation component customizations must be supported by software mass customization.

**Solution:** Logical names, corresponding to a point in the domain space, can be assigned to a component customization, similar to a label in configuration management. The logical name can then be used to identify the associated point in the domain space in order to instantiate each variation point in the component, thereby instantiating the customized component.

**Customization composition management.** Customized products are composed out of customized and common components. Reproducible product composition from customized components must be supported.

**Solution:** A customized product is a composition of customized components. Associating a logic name for a customized product with a list customized components is sufficient to look up or instantiate the customized components for a product.

## 8 Putting It All Together

We have shown that variation management in software product lines is a multi-dimensional configuration management problem, where variation in time and space must be effectively managed. In order to “conquer” the variation management problem, we “divided” it into nine smaller sub-problems with simpler solutions. We covered this solution space with three technology clusters, each of which provides the solutions to several of the nine sub-problems.

To illustrate how all of these solutions work in concert to provide variation management for a software product line, we conclude with a simple scenario for instantiating a custom product instance.

1. Get a baseline for the production line from which to build a product instance. The baseline might be from the latest development activity on the main trunk, from a baseline on a release branch for a production line release two months ago, or from a branch being used to do quality assurance testing prior to the next release. Items of interest in the production line baseline are the component composition “context” (see Section 6) and the customization composition lists for the product instances (see Section 7).
2. Use the component context from step 1 to get the baseline version for each component (see Section 6). The baselines of the components that can be customized will include all of its common artifacts and variant artifacts in the uninstantiated form.
3. Using the logical name for the product instance to be built, get the customization composition list from step 1. This list provides the logical name for each customized component instance needed for the product (see Section 7).

4. For each component, use the logical name from step 3 to identify the desired point in domain space in order to instantiate all of the variation points in the component (see Section 7). The source for each component is fully instantiated after this step.
5. Apply conventional build tools as needed to the component source in step 4 to create an installable and executable product instance.
6. A custom test suite matching the custom product instance should be a part of what is produced in steps 1-5. Use this test suite to test the product instance.

## 9 Conclusions

An interesting observation that we have made while “dividing and conquering” the variation management problem is that it can serve as a complete and consistent basis for creating a product line. For example, we have taken a one-of-a-kind software system built out of components, added software mass customization capability as described in Section 7, and created a software product line without doing any domain re-engineering, re-architecting, re-design, and so forth. We have even seen this approach applied to a legacy software with no formal or documented domain analysis, architecture, or design.

This observation is not a recommendation to omit analysis, architecture, and design. Rather, it demonstrates that software product line approaches can be adopted very quickly and incrementally, starting with whatever legacy artifacts might serve as a suitable starting point and then adding formality as needed during the evolution of the product line. In contrast to some of the more heavyweight methodologies that require significant up front investments to adopt a product line approach, an incremental approach based on variation management may offer a more palatable adoption strategy to “in production” software projects.

## References

- [1] Software Engineering Institute. *The Product Line Practice (PLP) Initiative*, Carnegie Mellon University, [www.sei.cmu.edu/activities/plp/plp\\_init.html](http://www.sei.cmu.edu/activities/plp/plp_init.html)
- [2] Weiss, D., Lai, R. 1999. *Software Product-line Engineering*. Addison-Wesley, Reading, MA.
- [3] Bass, L., Clements, P., and Kazman, R. 1998. *Software Architecture in Practice*. Addison-Wesley, Reading, MA.
- [4] Jacobson, I., Gris, M., Jonsson, P. 1997. *Software Reuse: Architecture, Process and Organization for Business Success*, ACM Press / Addison-Wesley, New York, NY.
- [5] *Software Product Lines. Experience and Research Directions. Proceeding of the First Software Product Lines Conference (SPLC1)*. August 2000. Denver, Colorado. Kluwer Academic Publishers, Boston, MA.
- [6] Clements, P., Northrop, L. 2001. *Software Product Lines: Practice and Patterns*, Addison-Wesley, Reading, MA.
- [7] *Dagstuhl Seminar No. 01161: Product Family Development*. April 2001. Wadern, Germany.

- [8] Postema, H., Obbink, J.H. Platform Based Product Development. *Proceedings of the 4th International Workshop on Product Family Engineering*. October 2001. Bilbao, Spain. Springer-Verlag, New York, NY.
- [9] van der Linden, F., Wijnstra, J.G. Platform Engineering for the Medical Domain. *Proceedings of the 4th International Workshop on Product Family Engineering*. October 2001. Bilbao, Spain. Springer-Verlag, New York, NY.
- [10] BigLever Software, Inc. Austin, TX. [www.biglever.com](http://www.biglever.com)
- [11] Krueger, C. Using Separation of Concerns to Simplify Software Product Family Engineering. *Proceedings of the Dagstuhl Seminar No. 01161: Product Family Development*. April 2001. Wadern, Germany.
- [12] 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.
- [13] Krueger, C. Software Reuse. 1992. *ACM Computing Surveys*. 24, 2 (June), 131-183.
- [14] Krueger, C. 1997. *Modeling and Simulating a Software Architecture Design Space*. Ph.D. thesis. CMU-CS-97-158, Carnegie Mellon University, Pittsburgh, PA.