

The 3-Tiered Methodology: Pragmatic Insights from New Generation Software Product Lines

Charles W. Krueger
BigLever Software, Austin, TX
ckrueger@biglever.com

Abstract

Early generation software product line (SPL) methodologies tended to be large, complex, and offer many options and choices, making adoption in practice difficult to comprehend, justify and achieve. We have identified a simpler pattern for SPL methodology, referred to as the 3-Tiered Methodology, based on observations during deployments and operation of new generation SPL practices. This methodology is useful not only for the operation of full-scale SPL practice, but also in understanding, explaining and justifying SPLs, as well as for planning and making transitions to SPL practice.

1. Introduction

Historically, a fundamental problem that prospective SPL adopters have faced is that early generation SPL methodologies were large and complex, offering many options and choices[1,2,3]. The myriad of interrelated technical, process and organizational issues for these methodologies made it difficult to “connect the dots” between the practice changes prescribed by an SPL approach and the resulting benefits predicted in the literature. This made it difficult to clearly articulate the precise relationship between the new capabilities and the resulting benefits delivered by SPL practice. The lack of detail, precision and clarity meant that business managers had great difficulty putting together a rational business case for moving to SPL practice. Ultimately, this has slowed adoption of SPL practice in the mainstream.

Direct involvement in more than ten different SPL deployments over the last 12 years, including Software Product Line Hall of Fame inductees LSI Logic / Engenio[4] and Salion[5], has illuminated a pattern that helps to better articulate the relationship between capabilities and benefits. This emerging pattern reveals a view into a progression of SPL capabilities and a resulting progression of benefits, providing a more insightful characterization than traditional explanations. Based on this pattern, we have captured a pragmatic and simple 3-tiered SPL methodology at the

root of the new generation of software product line successes, simply referred to as the *3-Tiered Methodology*.

2. The 3-Tiered Methodology

As companies shift from conventional product-centric software development to SPL development, three tiers of capabilities and benefits are established, sometimes in sequence and sometimes in parallel. Each tier builds upon and is enabled by the capabilities and benefits of the previous tier. That is, the capabilities at each tier provide direct benefits, but also enable increasingly more strategic capabilities and benefits in the higher tiers. The base tier provides a very tactical set of developer capabilities and benefits, which enables a middle tier of engineering management capabilities and benefits, which ultimately enables the top tier of highly strategic capabilities and benefits in the executive and business operations.

This partitioning of capabilities and benefits into three distinct tiers provides a modular methodology that is easier to understand and explain. The well defined relationships between the tiers reduce the number of options and clarifies choices when defining and adopting an SPL approach.

The *3-Tiered Methodology* provides a holistic view that incorporates many of the successful new generation SPL methods and techniques described individually in previous papers and case studies, including [6,7,8,9,10,11,12].

Figure 1 illustrates the 3-Tiered Methodology. The base tier, *Variation Management and Automated Production*, provides the foundation for a software product line practice. The focus is on the basic infrastructure for first-class variation management in the product line, including feature modeling capabilities, a uniform mechanism that supports variation points in the software assets, and an automated production mechanism to instantiate products from the feature models, software assets and implementation-level variation points.

At the base tier, conventional software assets, such as legacy source code, and conventional organizational

structures, such as teams organized around products, are sufficient. That is, in the base tier of the methodology there is no need for specially engineered *software product line architectures* and *core assets*, or a dichotomy of engineering roles for *domain engineering* and *application engineering*.

First-class variation management and automated production serve to:

- eliminate duplication, cloning, divergence and merging
- consolidate the multiple ad hoc variation management mechanisms typically found in legacy software
- eliminate the manual and parallel production efforts found in conventional approaches

The primary benefit gained from these capabilities in the base tier is lower development overhead compared to conventional approaches and therefore higher developer productivity and lower per-product development cost.

The capabilities and benefits of the base tier enable the middle tier of the methodology, *Core Asset Focused Development*. The focus of the middle tier is on organizing the assets and the development teams around the reusable components and subsystems – which are referred to as the *core assets*. The shift from product-centric development to core asset focused

development enables engineering management to manage the development of the portfolio as a single system rather than managing a multitude of products.

In contrast to early generation SPL methodologies, where engineering management created a dichotomy of engineering efforts, one for *domain engineering* and one for *application engineering*, the 3-Tiered Methodology does not require the dichotomy nor the *application engineering* teams. This has proven to be much more effective in practice[4,5,13].

Being able to organize development teams around core assets rather than products eliminates the need for many or all developers to understand the entire product. Rather, core asset focused development teams can establish deep and narrow expertise. From an engineering management perspective, this organizational structure is very stable. In contrast with *application engineering* approaches that have to scale the organization with each and every product added to the portfolio, the core asset focused capabilities created in the middle tier of the methodology mean that the organizational structure around core assets is stable – it is very similar for 2, 20, 200, or 2000 products.

The primary benefit gained from these capabilities in the middle tier is higher product quality compared to conventional product centric development. This results from high levels of software reuse, deep core asset expertise and stable organizational structures.

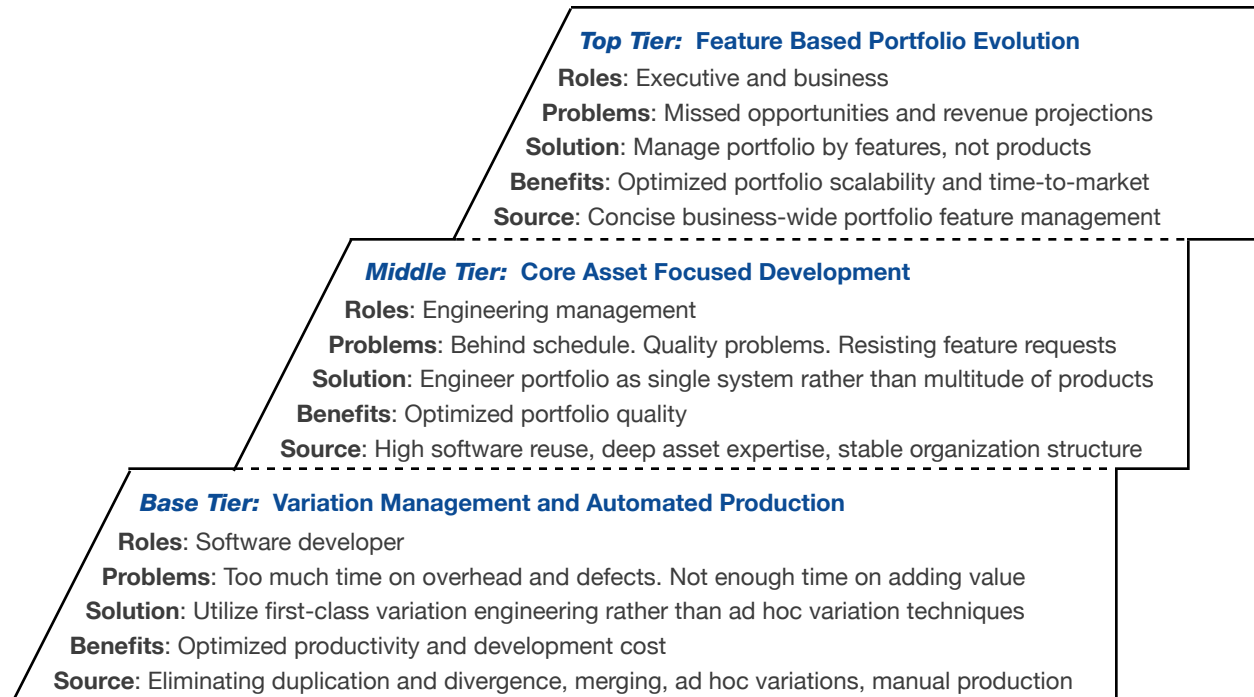


Figure 1. The 3-Tiered Software Product Line Methodology

The capabilities and benefits of the base and middle tiers enable the top tier of the methodology, *Feature Based Portfolio Evolution*. The focus of the top tier is on business-wide management of the entire software product line portfolio using concepts and terminology based on portfolio features – that is, the feature model.

As the business transitions from product based to feature based portfolio evolution – where the entire portfolio evolves by adding or modifying feature requirements for common, optional, and varying features – the result is extremely efficient and concise communication between the business and engineering teams, leading to optimized time-to-market and product line scalability.

An operational view of the 3-Tiered Methodology is illustrated in Figure 2 (which will be explained in greater detail in later sections). In the top tier, the Product Marketing or other business teams define the features to be added or modified for the product line

portfolio to satisfy a business opportunity. These features are mapped into core asset implementation requirements by the product line architecture team in the middle tier. Core asset developers then extend the core assets as needed to implement the new feature requirements, utilizing the SPL infrastructure in the base tier to automatically configure, build and test the products in the portfolio during development, and ultimately to build and deploy the final products for customer release.

In the next three sections, we explore the three tiers of the methodology in more detail. Following the outline used in Figure 1, each of the tiers is examined in terms of problems, solutions, benefits and the source of benefits. The final section in this report describes how the 3-Tiered Methodology simplifies the transition from conventional software development practice into SPL development practice.

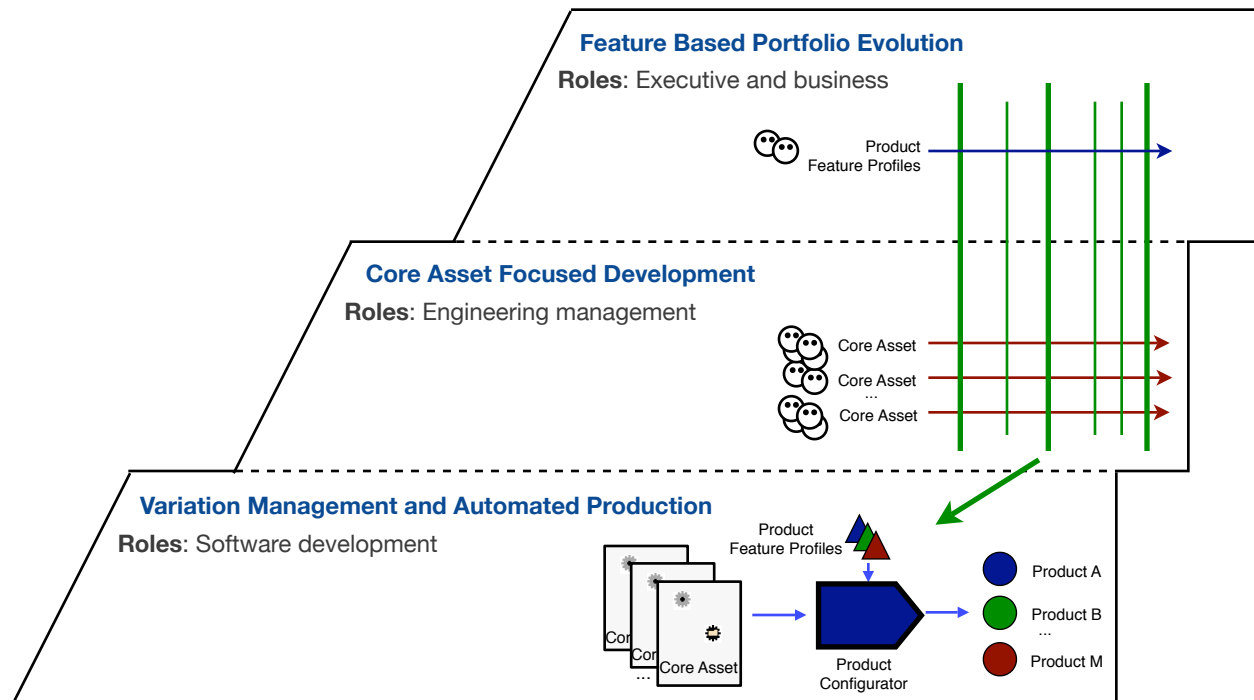


Figure 2. Operational View of the 3-Tiered Methodology

3. The Base Tier – Variation Management and Automated Production

Problem. Conventional software development tools and methods are product-centric, focusing on the lifecycle of one product at a time. In a product line setting, this leads to:

- duplication and divergence in the software assets throughout the software lifecycle
- multiple ad hoc and inefficient code-level techniques used to manage commonality and variation
- complex and manual merging and coordination among products, growing in Order- N^2 combinatoric complexity with the number of products

All of this labor intensive overhead reduces developer productivity. Less time is spent adding new features and value to products. The nonproductive overhead makes it harder to meet delivery deadlines and customer commitments. Lower morale within the development team can result.

Solution. As the base tier name implies, first-class variation management and automated production form the baseline solution for the methodology, including feature modeling capabilities, a uniform mechanism that supports variation points in the software assets,

and an automated production mechanism to instantiate products from the feature models, software assets and implementation-level variation points. For example, *software product line configurators* provide these capabilities and are at the foundation of some of the new generation software product line success stories [6,7,8].

As shown in Figures 2 and 3, software product line configurators take two types of inputs – *core assets* and product *feature profiles* – in order to automatically create product instances. The core assets can include requirements, architecture and design, source code, test cases, product documentation, and so forth. The product feature profiles are concise abstractions that characterize the different product instances in the product line portfolio, typically expressed in terms of a *feature model*[14].

Consolidation is the key to solving the problems addressed at this level. The first-class variation points in the software core assets allow the duplication and divergence to be consolidated into software assets with combined commonality and variation points. The feature modeling and variation point mechanism consolidate the multiple ad hoc variation techniques into one first-class approach. The feature model expresses the portfolio variation, which is normally implicit and diffusely defined in the implementation of the portfolio, into a single consolidated representation.

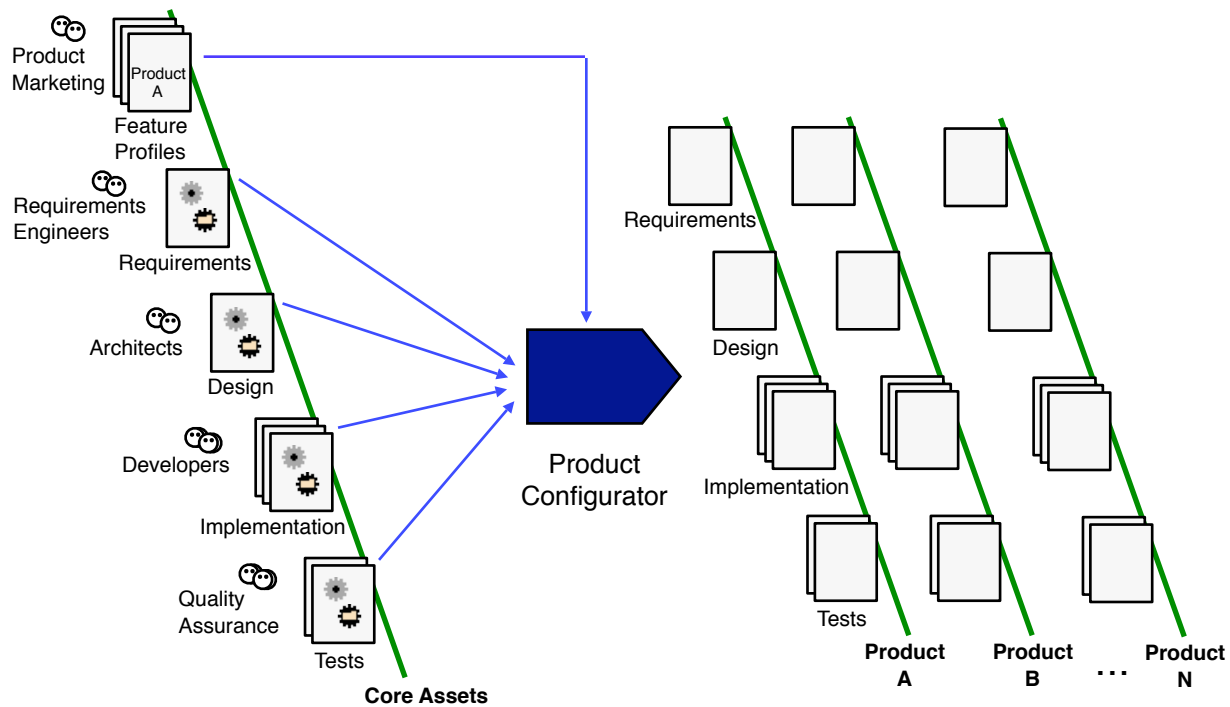


Figure 3. Software Product Line Configurator

Automated production of products from feature profiles and core assets allows a single consolidated change to a software asset to be automatically reconfigured into all products in the product line portfolio. This eliminates the need for manual merging of core asset modifications into multiple products.

Note that the definition of “core asset” within the base tier of the methodology is very liberal. Essentially, any legacy source code or other software asset can serve as a core software asset at this level so long as it consolidates commonality (i.e., eliminates duplication, branching, and other forms of divergence), contains zero or more variation points, and can be used by the product configurator to instantiate products.

Benefits. The variation management and automated production capabilities established in the base tier offer significant increases in developer productivity. Less time is spent dealing with duplication, divergence, merging, less effort creating and dealing with multiple, complex, ad hoc, home-grown variation techniques, and less overhead on Order-N² combinatoric coordination among products. Consolidation with first-class variation points means the software is easier to comprehend, structure and maintain.

Because labor costs often dominate the cost of building software, productivity and cost benefits go hand-in-hand. Productivity increases mean lower development cost per product and per feature.

Compared with early generation SPL methodologies, the base tier of the 3-Tiered Methodology provides a much simpler entry point into SPL practice. Variation management and automated production can be adopted incrementally and with little disruption to ongoing production schedules, particularly when using commercially available product configurators that provide these capabilities out of the box[4]. Legacy software assets can be heavily leveraged with little or no re-engineering. Existing team structures can be maintained with little or no reorganization. The processes and day-to-day activities remain virtually unchanged.

In a recent example, HomeAway experienced the following timeline[13]:

- 30 days from starting to explore SPLs as an option to an executive consensus on the business case to adopt an SPL approach.
- 14 days for 3 engineers to establish the base tier capabilities of the methodology and put the SPL approach into production use by the development team.

Source of Benefits. The root source of all the benefits in the base tier of the 3-Tiered Methodology is *predictive and strategic software reuse*[9]. SPL development tools provide first-class support for SPL-specific development tasks, such as impact analysis on variation points for combinatoric statistics on feature model options and selections. Explicit support for

feature models and product feature profiles and variation points provides a higher level of concise abstractions from which developers can work.

Following are specific examples of the benefits achieved in typical scenarios we have seen in practice when moving organizations from conventional product-centric development to the base tier of the 3-Tiered Methodology.

3.1. Example: Branches and Clone-and-Own

Problem. Duplication and divergence of the software assets due to the use of configuration management branches and/or cloned copies.

Solution. Use first-class variation management to consolidate variant branches and clones into a single collection of software assets.

Benefits. Developers spend less time on the overhead duplicates, branching, or merging and spend more time enhancing and adding new features. The primary benefit from this stage is increased developer productivity that comes from reduced complexity, from eliminating duplication/divergence/merging, and from automated production.

- First-class variation management with variation points and feature model reduces complexity and makes variation concepts and constructs explicit.
- Incremental transition gradually eliminates duplication, divergence and merging.
- Automated production allows one change to be automatically be built into every product.

Source of Benefits:

- Fewer “components” and fewer lines of code
- Potentially higher code coverage by eliminating redundant and “dead” code
- Less duplicate work in maintenance and bug fixing.
- Less coordination and merging. Eliminate N-squared combinatorics among products.

3.2. Example: Multiple Ad Hoc Home-grown Variation Management Techniques

Problem. Most organizations have adopted a combination of ad hoc variation management techniques – we have seen as many as 29 different home-grown variation management mechanisms used in combination in one portfolio.

- Compile time conditionals in the source assets – such as #ifdefs and templates – controlled by compiler flags and directives spread across build scripts, header files, and so forth. Mixing of application logic with portfolio variation logic in the source code. It is complex, error prone and labor intensive to comprehend, learn and maintain. Impact analysis hard.

- Runtime conditionals controlled by config files, nonvolatile RAM, database settings and so forth. Footprint. Testing combinatorics. Mixing of application logic with portfolio variation logic in the source code. It is complex, error prone and labor intensive to comprehend, learn and maintain. Impact analysis hard.
- Build and Installer scripting. File naming conventions. CM sparse branch conventions. Hard to comprehend and learn and maintain. Impact analysis hard.

Solution. A single, first-class SPL variation management solution with a software product line configurator. First-class feature model expresses feature variations in the portfolio. Explicit encapsulated variation points isolate variations the asset implementation level, including variation point logic that expresses the mapping between feature variation and implementation level variation. Feature profiles and composition profiles, with assertions to constrain valid combinations.

Benefits:

- Spend time building portfolio features rather than variation management tools and techniques
- Simpler to learn and use
- More efficient development due to clarity, first-class treatment of variation management, and better impact analysis
- Smaller executable footprint and higher performance when eliminating runtime conditionals

Source of Benefits. First-class feature model expresses feature variations in the portfolio. Explicit encapsulated variation points to isolate variations the asset implementation level, including variation point logic that expresses the mapping between feature variation and implementation level variation.

4. The Middle Tier – Core Asset Focused Development

Variation management and automated production at the base tier eliminate the need for manual product-centric development. The capabilities and benefits in the base tier enable the middle tier of the 3-Tiered Methodology, *Core Asset Focused Development*.

Problem. Organization of teams around product-focused development is ineffective for portfolio development. For engineering management, it introduces significant time, effort, cost and overhead to staff and deploy a development team for each and every product added to the portfolio. Isolated product development contexts degrade the potential for software reuse and lead to potential duplication of effort across product teams. All product teams have to establish expertise for the entire product, resulting in a

steep learning curve and long productivity ramp for new developers. Having to have a broad understanding of the entire implementation of the product necessarily results in a more shallow knowledge in any particular area.

For engineering management, there is a divergent focus on the lifecycle, project schedules and release events for multiple products in the portfolio. This leads to high management overhead and a complex management job doing resource allocation and coordination across product teams.

Solution. The middle tier of the 3-Tiered Methodology focuses on establishing *core asset focused development*. Engineering management organizes teams around *core software assets* rather than products. As illustrated in Figures 2 and 4, core assets evolve over time (left to right arrows) and well-defined baselines of core assets (vertical lines) are used by the product configurator to automatically instantiate products in the portfolio from that baseline.

As noted in the description of the base tier, the definition of what constitutes a “core software asset” is very liberal. It can be a source code component or subsystem, it can be a subsection in a requirements document, it can be a set of unit test cases and so forth. A core asset in this methodology simply consolidates commonality within some asset within the software lifecycle, contains zero or more variation points, and can be used by the product configurator to instantiate lifecycle assets for products.

The core asset focused development teams are similar to the *domain engineering* teams in early generation SPL methodologies – they both create reusable core assets for the product line portfolio. However, as illustrated in Figure 2 and 3, in the 3-Tiered Methodology there are no *application engineering* teams doing product-specific development since the product configurator in the base tier can automatically produce all of the products from core software assets[6,7,8]. In the middle tier, all development activity occurs on the core assets, whether it is for a variation point that applies to only one product or for common software that applies to all products. There is no product-specific glue code.

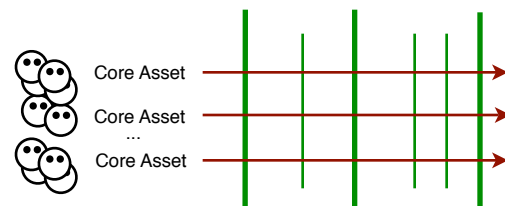


Figure 4. Core Asset Focused Development

The result is a simpler and more elegant solution for engineering management. The portfolio is now

developed as a single system of core assets rather than a multitude of products.

A common means of initiating a software product line approach is to start with legacy software assets from products developed using conventional techniques. In the base tier of the methodology, little or no modification is required to utilize the legacy assets. However, in the middle tier, refinements may be beneficial in order to optimize the effectiveness of core asset focused development.

The key to effective core asset focused development is modularity in the core assets. The modularity can be informal, such as simply partitioning subsystems into subdirectories in the file system structure. As long as core asset teams can work with relative independence, then the modularity is sufficient for the middle tier.

In cases where the legacy assets are tightly coupled in a monolithic asset base, core asset teams can still be assigned based. We have seen that the “gravity” provided by the core asset teams can be leveraged to incrementally refactor the monolithic system into an implementation architecture with more well defined modular boundaries.

Benefits. The core asset focused development capabilities established in the middle tier offer significant increases in quality. Fewer defects are introduced by core asset focused teams and as a result the entire portfolio displays higher degrees of quality. Higher quality means less time fixing defects during development, faster and more efficient test cycles, and less time and cost dealing with defects in the field. All of this leads to higher customer satisfaction.

Core asset focus provides just that – better focus for developers and development teams. There is no need for all teams to understand all aspects of a product implementation. The narrow focus of a core asset means that new staff members can get up to speed and become productive much quicker. The portfolio is developed as a single system rather than a multitude of products, reducing the labor-intensive combinatorics for coordination and knowledge sharing.

The organizational structure is much simpler for engineering managers to establish and maintain, compared to a multitude of product teams with shifting resource and deadline requirements. Because the core asset structure within a software product line architecture tends to be much more stable during portfolio evolution than the number of products in the portfolio, the core asset focused organizational structure tends to be much more stable as well.

Core asset structure is also suitable for geographically distributed development teams. Core assets with well defined modular boundaries can be effectively allocated and developed at different development sites.

Source of Benefits. The quality benefits in the middle tier of the methodology come from three sources.

- high levels of software reuse
- deep asset expertise
- simple and stable organization structure

The degree of software reuse in core asset focused development extremely high. All development effort on core assets in this methodology is candidate for reuse – 100% reusable – for any or all products in the product line. Even when core asset development is done in a variation point that is specific to one product, it is possible to reuse that software on any product at any time in the future.

Because core asset development teams are narrowly focused, they tend to gain deep expertise. That expertise can be leveraged to create a portfolio of feature rich, high quality, and highly competitive products. Core asset teams gain pride of ownership that leads to higher morale and higher quality.

Developing the portfolio as a single system rather than a multitude of products offers a simpler and more stable organizational structure. This makes the job of engineering management easier and more predictable. The number of products in the portfolio is no longer relevant – the organizational structure remains stable for 20 or 200 or 2000 products.

Compared with early generation SPL methodologies, the middle tier of the 3-Tiered Methodology provides a simpler and more effective approach by eliminating the need for *application engineering* (AE) teams and focusing solely on *domain engineering* (DE) in core asset focused development teams. Avoiding the dichotomy of having two types of development teams – AE and DE – eliminates many negative implications, including the “us-versus-them” cultural dissonance between the AE and DE teams [6,7,8].

5. The Top Tier – Feature-Based Portfolio Evolution

Core asset focused development at the middle tier plus variation management and automated production at the base tier eliminate the need for manual and product focused development. The capabilities and benefits in the base and middle tiers enable the top tier of the 3-Tiered Methodology, *Feature Based Portfolio Evolution*.

Problem. With conventional product-based portfolio evolution, the business side of an organization – typically the Product Marketing team – provides a set of requirements for each product needed in the portfolio. The requirements for each new product are often created by cloning the requirements document for the last product and then making

appropriate changes. Product Marketing and Engineering will then review and negotiate the content and schedule for the new product release roadmap.

This approach has several drawbacks. Because the full requirements set is used for each product, there is a high overhead for creating and negotiating the requirements, schedules and roadmaps for new products. This results in slow time-to-market for the products, which ultimately limits the number of products that can be effectively deployed and maintained in a portfolio.

From a business perspective, this has a negative impact on strategic business operations. There is a lack of agility to quickly react to market opportunities, changing market conditions and market turbulence. The limited number of products means that the business has to pass up some market opportunities.

Solution. The top tier of the 3-Tiered Methodology focuses on *feature based portfolio evolution*. Rather than manage the portfolio evolution on a product-by-product basis with full requirements specifications on each product, the portfolio is managed by feature specifications, plus feature profiles for the products. Features become the primary concept for managing the portfolio across the entire enterprise. Products are secondary.

The software product line approach now extends outside of the development organization to include business areas such as Product Marketing. Using domain-level feature concepts, non-technical business roles can use a concise feature profile rather than full-scale requirements document to define a new product within the current scope and to specify where the scope needs to be extended to accommodate new feature requirements.

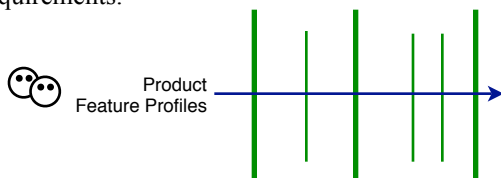


Figure 5. Feature Based Portfolio Evolution

As illustrated in Figures 2 and 5, product feature profiles evolve over time (left to right arrow) and well-defined baselines of the feature profiles (vertical lines) are aligned with the core asset baselines for use by the product configurator to automatically instantiate products in the portfolio.

Benefits. The strategic business benefits of the SPL approach are fully realized in the top tier of the 3-Tiered Methodology. The benefits come both in the form of reductions in time-to-market for new products and features, as well as increases in the scalability of the portfolio.

Product feature profiles are conceptually simple descriptions for the Product Marketing team to create and to negotiate about with the Engineering team. New and extended feature requests can be simply expressed in terms of “deltas” with the feature model and product feature profiles.

Evolution of the portfolio is based on units of features, which is driven by product marketing and motivated by the features have the most business value for the portfolio. Product marketing and engineering negotiate future work and product line scope according to the cost/benefit of adding features to the portfolio.

The size of portfolio can scale as large a needed to meet business demands and opportunities. The number of products in the portfolio loses relevance – this number can scale to the size needed to meet market opportunities rather than being limited by development cost and resources[4].

The business now has enhanced its competitive advantage. It has the agility to rapidly and precisely evolve the portfolio and to expand into new markets. Products can be brought to market very quickly. The business can survive and even capitalize on turbulent market conditions.

Source of Benefits. The capabilities and benefits available at the top tier of the 3-Tiered Methodology are a nearly optimal form of software reuse. We have reduced the *cognitive distance* between new portfolio feature concepts and the products deployed based on those concepts[15]. By reducing the cognitive distance between concept and deployment, the decreases in time-to-market and increases in portfolio scalability directly follow.

- new product feature profiles supported by existing feature models and core assets can be used to immediately and automatically configure the new products
- new product feature profiles supported by existing feature models but not yet fully supported by the core assets can be easily mapped into requirements for suitably extending the core assets
- new product feature profiles not supported by existing feature models and core assets can be easily mapped into requirements for suitably extending the feature models and core assets

The engineering effort required for any new product or feature is pure *delta engineering*, where the only new development required for a new product instance is precisely what is lacking in the current assets. Everything else across the portfolio development lifecycle is fully reused.

Compared to early generation SPL methodologies, where product line scoping by features was often an engineering activity, the business drives the feature based evolution and scoping. This is enabled by

making the feature language suitable for use by non-technical users.

6. Transition to SPL Practice Using the 3-Tiered Methodology

The obvious tendency with a transition to SPL practice using the 3-Tiered Methodology is to think bottom up. That is, begin with the base tier and then start on the middle tier, in sequence, after the base is fully established.

While this style of transition is simple and effective, we have found that it is also effective to address the tiers incrementally and in parallel. In Figure 2, imagine the incremental transition effort growing from left-to-right rather than bottom up. Initially the base tier is incrementally addressed. Once sufficient capability is established in the base tier, then activities can begin in the middle tier of the methodology. Similarly, the top tier activities can commence once there is sufficient capability established in the middle tier.

Other transitions are likely possible, as well. For example, an organization may have a well defined business practice that utilizes *portfolio management* in terms of *features*. Although we have not seen this type of transition occur in practice, there is nothing in the 3-Tiered Methodology to suggest that it would not be possible to “backfill” the middle and base tiers, starting with an established top tier.

7. Related Work

The *early generation* software product line development methodologies we refer to in this paper were often born from efforts to analyze and characterize successful, pre-existing, home-grown software product line case studies, in hopes that these characterizations would provide insight and guidance for a new generation of success stories. Measured by the sparsity of new published case studies, these hopes have not been realized. We have postulated here and elsewhere that the early generation methodologies have been difficult to adopt in practice because they are either overly complex, overly specific and/or overly general.

The 3-Tiered Methodology is enabling a *new generation* of software product line success stories – new successes from applying an SPL methodology rather than SPL methodologies being derived from home-grown approaches[4,5,13]. We have therefore characterized this as a *new generation* methodology.

The 3-Tiered Methodology was born from a doctoral thesis on configurable software architectures (now known as *software product line architectures*), in conjunction with more than a decade of hands-on experiences successfully applying, refining and

validating these ideas in commercial software product line development practice[16].

The most commonly cited early generation SPL methodology is the SEI SPL Framework[1]. It is often referred to as the SPL approach, rather than one of several methodologies that have been published. It comprises a complete book of 29 *key practices*, divided into three *essential activities*, domain engineering, application engineering, and management. The domain engineering versus application engineering dichotomy is where domain engineering teams create reusable assets and application engineering teams create product using the reusable assets. We have argued here and elsewhere that this dichotomy is a major impediment to adoption of the SEI framework and other methodologies[8]. The 3-Tiered Methodology specifically avoids application-focused processes and organizational structures.

BAPO (Business, Architecture, Process, Organization) is another early generation SPL framework and methodology[17]. The B (Business) concern of the framework characterizes very general business practices, whereas the 3-Tiered Methodology has specific focus on a well defined method of portfolio management at the business-engineering interface. The A (Architecture) concern of the BAPO framework prescribes creating a *software platform* and *software components* using application engineering roles to create product instances, whereas the 3-Tiered Methodology utilizes product configurators, is agnostic to architectural method, and avoids the use of the application engineering. The P (Process) concern of BAPO defers to CMMI and therefore provides little guidance. The O (Organization) concern of BAPO describes specific examples of different organizational patterns, based on the other concerns such as platform development. The 3-Tiered Methodology focuses on organizational issues in the middle tier, based on the configurator infrastructure model of the base tier and the business-engineering interface of the top tier.

The FAST methodology of Weiss and Lai was the subject of the first book published about an SPL methodology[2]. The book provides an extensively detailed process description about an SPL methodology developed and used at Lucent. Similar to other early generation SPL methodologies, FAST relies on the dichotomy of domain engineering and application engineering.

Czarnecki and Eisenecker describe a generative programming methodology based on feature models and C++ templates. The focus of this methodology is limited to source code assets that can be reengineered into sophisticated C++ templates for the core assets [14].

Both the Kobra approach and the approach prescribed by Gomma provide SPL methodologies based on UML and component-based development

(CBD)[3,18]. The 3-Tiered Methodology can accommodate, but is not limited to, UML and CBD.

Batory defines a generative SPL methodology based on the definition and application of functional feature compositions and reusable software components that implement the features[19]. This approach requires up reengineering of software assets into a form suitable for functional composition.

Pure::systems is another example of a commercial SPL configurator[20]. In contrast to the 3-Tiered Methodology that focuses the overwhelming majority of development effort on domain engineering, the methodology advocated by pure::systems utilizes a four quadrant approach for modeling and derivation in both the *problem space* (domain engineering) and the *solution space* (application engineering).

8. Conclusions

Mainstream software practitioners have found that early generation SPL methodologies tend to be complex and to offer too many options and choices. This has made the prospect of SPL adoption in practice difficult to justify and achieve. We have identified a simpler pattern for SPL methodology, referred to as the 3-Tiered Methodology, based on observations during deployments and operation of new generation SPL practices. This methodology is useful not only for the operation of SPL practice, but also in understanding, explaining and justifying SPLs, as well as for planning and making transitions into SPL practice.

9. References

- [1] Paul Clements and Linda Northrop. 2001. *Software Product Lines: Practice and Patterns*, AddisonWesley, Reading, MA.
- [2] Davis Weiss and Chi Tau Robert Lai. 1999. *Software Product-line Engineering*. Addison-Wesley, Reading, MA, p 73-74.
- [3] Atkinson, C., et.al. 2002. *Component-based Product Line Engineering with UML*. Pearson Education, London, UK.
- [4] William A. Hetrick, Charles W. Krueger and Joseph G. Moore, *Incremental Return on Incremental Investment: Engento's Transition to Software Product Line Practice*, October 2006, OOPSLA 2006, Portland, Oregon, ACM.
- [5] Ross Buhrdorf, Dale Churchett, Charles Krueger. *Salion's Experience with a Reactive Software Product Line Approach*. 5th International Workshop on Product Family Engineering. Nov 2003. Siena, Italy. Springer-Verlag LNCS 3014, p 315.

[6] *New Methods Behind a New Generation of Software Product Line Successes*, Technical Report 200602261, BigLever Software.

[7] Charles W. Krueger. *New Methods in Software Product Line Practice*, in Communications of the ACM, December 2006, pages 37-40.

[8] Charles W. Krueger. *New Methods in Software Product Line Development*, in Proceedings of the 10th International Software Product Line Conference, Baltimore, MD. August 2006, pages 95-99.

[9] 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.

[10] Charles Krueger. *Variation Management for Software Production Lines*, in Proceedings of the 2nd International Software Product Line Conference, San Diego, California. August 2002, pages 37-48.

[11] Charles W. Krueger and Dale Churchett. *Eliciting Abstractions from a Software Product Line*, in Proceedings of the OOPSLA 2002 PLEES International Workshop on Product Line Engineering. Seattle, Washington. November 2002, pages 43-48.

[12] 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.

[13] HomeAway: A Software Product Line Case Study, Technical Report 20070630, June 2007, BigLever Software, <http://www.biglever.com/>.

[14] Czarnecki, K. & Eisenecker, U. 2000. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Boston, MA.

[15] Krueger, C. *Software Reuse*, in ACM Computing Surveys, June 1992, pages 131-183.

[16] Krueger, C. *Modeling and Simulating a Software Architecture Design Space*, PhD Thesis, Carnegie Mellon University, Computer Science Department, CMU-CS-97-158, December 1997.

[17] Frank van der Linden, Jan Bosch, Erik Kamsties, Kari Käsälä, Henk Obbink. *Software Product Family Evaluation*. 5th International Workshop on Product Family Engineering. Nov 2003. Siena, Italy. Springer-Verlag LNCS 3014, p 352.

[18] H.Gomma, *Designing Software Product Lines with UML*, Addison-Wesley, 2004.

[19] D. Batory. *Feature Models, Grammars, and Propositional Formulas*. In Proc. Int'l Software Product Line Conference, 2005.

[20] www.pure-systems.com