The Systems and Software Product Line Engineering Lifecycle Framework

Contact Information:

info@biglever.com
www.biglever.com
512-426-2227

Mainstream forces are driving Systems and Software Product Line Engineering (PLE) approaches to take a more holistic perspective that is deeply integrated into the systems and software engineering lifecycle. PLE challenges will not be solved at any one stage in the product engineering lifecycle, nor will they be solved in independent and disparate silos in each of the different stages of the lifecycle. This paper describes BigLever Software's response to these forces – a *PLE Lifecycle Framework*.

The motivation for this technology framework is to ease the integration of tools, assets, and processes across the full systems and software engineering lifecycle. The framework provides product line engineers with a common set of PLE concepts and constructs for all of their tools and assets, at every stage of the lifecycle, and to assure that product line engineering traceability and processes flow cleanly from one stage of the lifecycle to another.

1. Introduction

According to Geoffrey Moore's classic hi-tech marketing book *Crossing the Chasm*, widespread adoption of a new hi-tech offering requires more than just a great concept, technique, technology, or product. The mainstream majority of users require a *complete solution*. Although a small percentage of potential users – *Innovators* and *Early Adopters* according to Moore – are willing to take on the task of extending new ideas and technology into a complete and workable solution, the majority of users that make up the mainstream – the *Early Majority* and *Late Majority* – are simply unwilling or unable to dedicate the resources needed to innovate complete solutions out of promising concepts, techniques, technologies, and tools [1].

The goal is to provide product line engineers with a common set of PLE concepts and constructs for all of their tools and assets, at every stage of the lifecycle, and to assure that product line engineering traceability and processes flow cleanly from one stage of the lifecycle to another.

These mainstream forces from the early majority and late majority contingents require Systems and Software Product Line Engineering (PLE) approaches to provide holistic solutions that are deeply integrated into the full engineering lifecycle. We describe here BigLever Software's response to these forces – the *PLE Lifecycle Framework* for the full systems and software product line engineering lifecycle.

The PLE Lifecycle Framework is a key ingredient to a complete, out-of-the-box PLE solution that enables easy adoption by mainstream organizations. It provides an integration framework for engineering tools, assets, and processes across the systems and software engineering lifecycle. The PLE Lifecycle Framework offers engineers and managers a common set of PLE concepts, constructs, and capabilities for all of their existing tools and assets, at every stage of the lifecycle.

When used in conjunction with BigLever's *3-Tiered PLE Methodology* [2], the PLE Lifecycle Framework offers a straightforward transition path from legacy approaches to a complete PLE solution. The PLE Lifecycle Framework has enabled organizations with some of the largest, most sophisticated, and complex safety-critical systems ever built to adopt the PLE approach [5].

2. Importance of the PLE Lifecycle

A fundamental tenet of product line engineering is that product line variation must be simultaneously managed along two dimensions, time and space [3]:

- *variation in time*, to manage asset evolution, or multi-baseline management;
- *variation in space*, to manage diversity within assets and products in the product line -- that is, multi-product management.

Practical experience with numerous commercial PLE deployments at BigLever Software has illuminated an equally important third dimension that must be addressed to provide a complete PLE solution for mainstream product line engineering organizations:

- *variation in the lifecycle*, to provide consistency and to manage traceability among asset variations in different lifecycle stages (i.e., multi-phase management)

These three dimensions -- multi-product in the domain space, multi-phase across the lifecycle, and multi-baseline in time -- are illustrated in Figure 1.

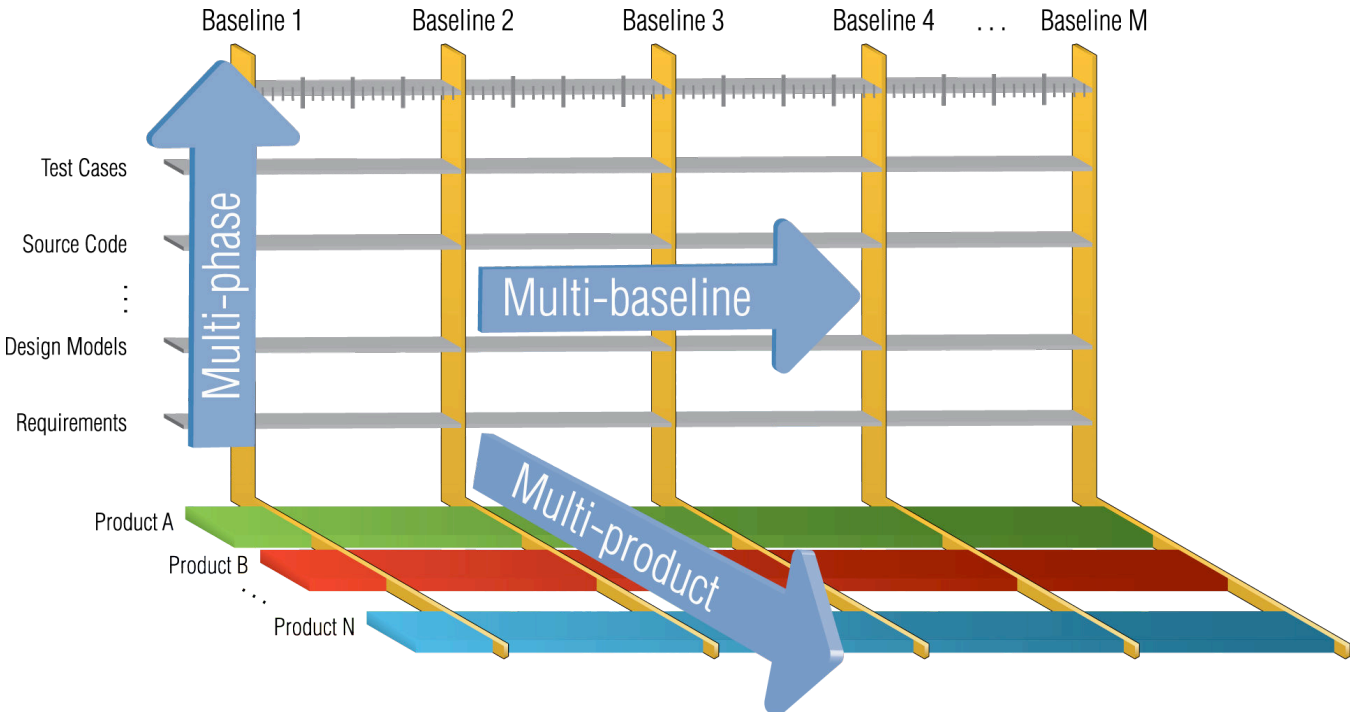

Figure 1. The Three Dimensions of a Complete PLE Solution

The importance of the additional multi-phase lifecycle dimension for providing a complete end-to-end solution can be attributed to the importance of:

- *Engineering process management*. Well defined engineering processes are common in mainstream organizations, often due to large size but also to external contractual or legal governance. As a result, the engineering lifecycle is usually explicitly defined by these organizations, within each lifecycle stage and across the boundaries between each of the stages.
- *Systems and software engineering*. Software – the primary focus in the early history of product line engineering – is often engineered within the larger context of a *system*. Embedded software in mechanical systems is an obvious example, though other common examples include enterprise and web-based software designed in the specific context of server, network, or specialty hardware and IT system architectures. For systems in which software does not play a major role in the end product, the engineering assets supporting those systems' development and sustainment (for example, requirements, or design models, or test cases) are represented or manipulated with software or software tools. As a result, the engineering lifecycle in-

cludes the combined systems and software perspective for requirements, architectures, models, documentation, test plans, and so forth, rather than just a narrow software-only product line perspective.

When transitioning to a PLE approach, a complete solution must retain the holistic lifecycle focus and capabilities. That is, a complete solution must simultaneously manage systems and software product line commonality and variability in all three dimensions of time, space, and lifecycle.

2.1. The Need for a PLE Lifecycle

One of the key capabilities of a good engineering lifecycle solution is *traceability* [4]. For example, every requirement should be traceable to one or more design elements that satisfy that requirement and each design element should be traceable back to one or more requirements that it satisfies. Each design element should be traceable forward to its implementation and vice versa. Each requirement should be traceable to and from one or more test cases that validate whether or not the requirement is satisfied in the final product.

For the lifecycle of an individual product, traceability is represented as a static set of relationships among the assets in the lifecycle (e.g., requirements, architecture, design, source code, test cases, documentation). Of course, as the software assets evolve over time, the traceability relationships also evolve. In fact, traceability relationships are intended to facilitate evolution by indicating which related artifacts need to change whenever any traceable artifact is modified. Thus, traceability for a single product applies to the multi-baseline and multi-phase dimensions of Figure 1.

When traceability is applied to a product line rather than an individual product, the third dimension – multi-product – must now be accommodated as well. Similar to the case of an individual product, traceability among the parts of assets that are common to all products is represented as a static set of relationships. However, for the parts of the assets the vary from product to product, the traceability relationships in and out of these asset "variation points" also vary. This is where most PLE concepts, technology, tools, and techniques fall short.

Imagine that a requirements engineering team has embraced a PLE requirements management technique based on tagging requirements in a requirements database with attributes that differentiate capability variations in requirements. Imagine that the design team has adopted a UML tool and has embraced inheritance as the mechanism for managing PLE design variations. The development team is using a FODA feature model drawn in a graphical editor, plus #ifdefs, build flags, and configuration management (CM) branches to manage PLE implementation variations. Finally, the test team has adopted clone-and-own of test plan sections, stored in appropriately named file system directories for each product to manage their PLE test plan variations.

Now imagine what would be needed to create a complete PLE lifecycle solution that integrates into a larger business engineering model. How do the requirements database attributes and tagged requirements relate and trace to the subtypes and supertypes in the design models? How do these attributes and supertypes relate and trace to the #ifdef flags, CM branches, FODA features, and test case product-specific cloned directories?

Without answers to these questions, it is not possible to define a lifecycle process that flows cleanly from one stage of the lifecycle to another. Trying to translate between the different representations and characterizations of capabilities, features and product variations creates dissonance at the boundaries between stages in the lifecycle.

This example illustrates that a complete PLE solution cannot be built with concepts, technology, tools, and techniques at any one stage of the lifecycle. Nor can it be built with a combination of disparate point solutions at different stages.

3. The PLE Lifecycle Framework

To meet the needs of engineering organizations for a complete systems and software product line engineering solution, BigLever Software created the *Gears PLE Lifecycle Framework*. This framework supports all three of the PLE solution dimensions illustrated in Figure 1 – multi-baseline in time, multi-product in the domain, and multi-phase in the lifecycle. The motivation for this technology framework is to ease the integration of engineering tools, assets, and processes across the full systems and software engineering lifecycle.

The goal is to provide product line engineers with a common set of PLE concepts and constructs for all of their tools and assets, at every stage of the lifecycle. Furthermore, the goal is to apply these concepts and constructs uniformly, so that traceability and engineering processes flow cleanly from one lifecycle stage to another – for systems analysts, requirements engineers, architects, modelers, developers, build engineers, document writers, configuration managers, test engineers, field engineers, project managers, product marketers, and so forth.

The PLE Lifecycle Framework is an integration framework for engineering tools and assets. It is narrowly focused on the PLE issues of feature-based variation management in the product space, and asset variation management in the lifecycle phases. This enables the third dimension (time) to be managed under the PLE methodology with conventional change, configuration and baseline management.

The PLE Lifecycle Framework provides product line engineering organizations the following [5][6]:

- A consolidated feature model to uniformly express product diversity, for assets in any stage of the systems and software engineering lifecycle, including requirements, architecture, models, design, source code, test cases, and documentation.
- Standardized variation point constructs that can be uniformly applied to any engineering tool and its associated assets in any stage of the systems and software engineering lifecycle, including tools and assets for requirements management, model-driven development, mechanical design, electrical design, source code development, test case development, configuration management, build automation, change management, and document development.
- An automated product configurator that uniformly assembles and configures assets from each stage of the engineering lifecycle to automatically produce any of the engineering assets in a product line.

An example product line engineering environment based on BigLever's *Gears PLE Lifecycle Framework* is illustrated in Figure 2. Key elements of the framework are:

- *Feature Catalog.* The Feature Catalog contains all the feature options available for all the products in that product line, and is used by individuals and teams across the lifecycle, and across organizational functions.
- *Bill-of-Features.* The features chosen for each product are specified in the Bill-of-Features for that product.
- *Shared PLE Assets*, such are requirements, design models, source code and test cases. Shared PLE assets contain feature-based variation points that can be configured in different ways to reflect the feature selections in a feature profile.
- *Gears Product Configurator*. The product configurator automatically configures all of the shared PLE assets for a product instance, based on the feature selections in a feature profile.

On the right half of the diagram shown in Figure 2 is the full set of product instances that can be produced by the product configurator in the framework. There is a one-to-one correspondence to the feature profiles and the product instances – each feature profile is used by the product configurator to automatically configure the assets for the corresponding product instance. Note that all the assets from the full engineering lifecycle are produced for each of the products in the product line.

The underlying PLE technology for the framework, such as the configurator, feature profiles and variation points, is described in [5][6]. Similarly, the underlying PLE methodology is described in [2].

The PLE Lifecycle Framework provides the critical lifecycle dimension needed for a complete PLE solution.

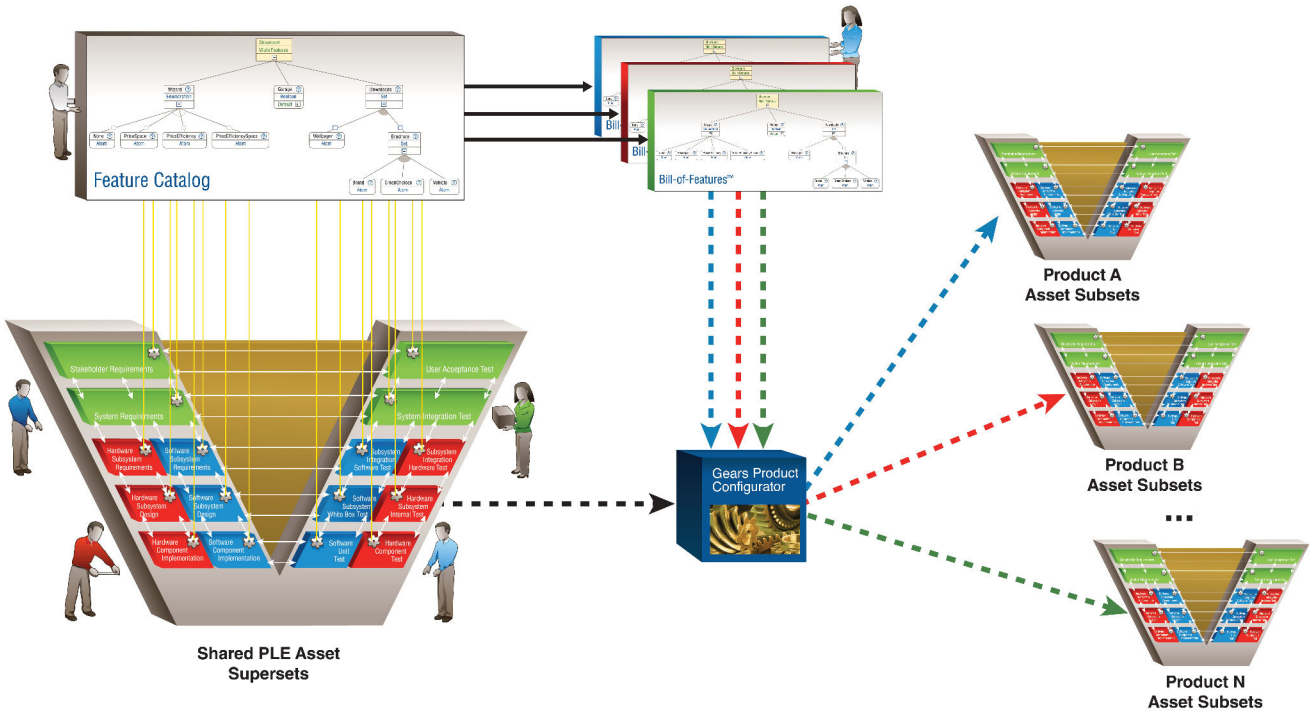

Figure 2. Product Line Engineering with the Gears PLE Lifecycle Framework

4. Common PLE Constructs and Concepts Across the Lifecycle

To overcome the problem of dissonant PLE concepts, technology, tools and techniques imposing disparate PLE silos in the engineering lifecycle, BigLever's PLE Lifecycle Framework provides a complete PLE solution by introducing common constructs and concepts across the full systems and software lifecycle. The common PLE concepts and constructs are provided by way of tool integration and asset integration into the PLE Lifecycle Framework.

4.1. Tool Integration

The purpose of tool integrations into the PLE Lifecycle Framework is simply to make conventional engineering tools "PLE-aware". This means making a tool aware and capable of working on sharable and configurable PLE assets that contain internal PLE variations rather than merely working on conventional assets for one-of-a-kind applications.

The central concept in making any engineering tool PLE-aware is the *variation point*. Variation points are the encapsulated locales within an asset that can be instantiated in different ways by the Gears product configurator, based on feature selections in the feature profiles. For an engineering tool to be PLE-aware, it must be able to:

- identify and display variation points
- create and modify variation points

- aid the PLE framework's product configurator in instantiating variation points during product configuration
- display the instantiated representation of a variation point, as well as all the superset of uninstantiated variability
- aid the PLE framework with product line management operations such as semantic checks on variation points, impact analysis, statistics and queries

Figure 3 illustrates the *PLE Bridge API* on the Gears PLE Lifecycle Framework. The API is used to create *bridges* to existing engineering tools, in order to make them PLE-aware, according to the capabilities outlined above.

Specific characteristics of a tool and its associated assets determine the type of integration bridge that is required. When a tool's assets have a text-based representation in the file system, the Gears framework can often provide all the variation point capabilities, without the need for an explicit tool integration.

On the other hand, when a tool's assets are maintained in a proprietary internal representation, a deeper two-way bridge integration is often required, where the tool is made PLE-aware and the framework is made tool-aware. For example, the IBM Rational DOORS® requirements management tool uses a database for its proprietary internal representation of requirements. A two-way *DOORS/Gears Bridge* was created as a dual plugin between DOORS and the Gears PLE Lifecycle Framework, to make them mutually aware so that operations and data can be exchanged to collaboratively perform PLE engineering operations in DOORS [8].

The integration bridges are independently installable options in the framework. When an organization transitions to the BigLever PLE approach, they identify preexisting bridges to match their existing toolset and install them into the Gears PLE framework, as illustrated in the example in Figure 3. Commercial, open source, and proprietary tool vendors across the full engineering lifecycle have recognized the importance of the PLE Lifecycle Framework and are working with BigLever to provide bridge integrations for their tools.

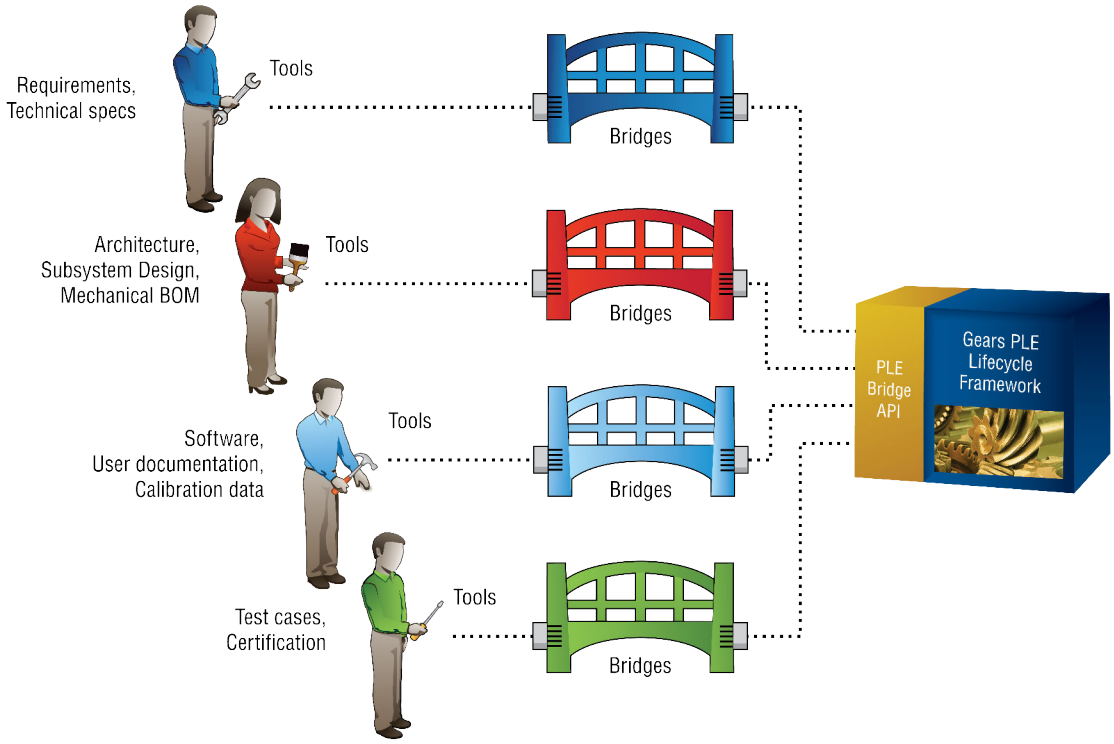

Figure 3. Tool Integrations with the Gears PLE Bridge API

4.2. Asset Integration

The purpose of asset integration into the Gears PLE Lifecycle Framework is to enable conventional one-of-a-kind assets to become shareable and configurable PLE assets. The key to asset integration is adding the variation point concept and constructs into the asset structure [6]:

- identify which asset constructs can become variation points
- define variation point encapsulations to contain the optional and alternative variants for the variation point, as well as the feature-based variation point logic to drive the variation point instantiation
- define the representation for variation point instantiations

To achieve the objective of common PLE concepts and constructs across all stages of the lifecycle, the variation point implementation in all assets must provide the capabilities outlined in the three bullets above.

For example, with file and text based source code assets, the constructs that can become variation points are typically directories, files, text blocks, text patterns, and text tokens. Directories make good variation point encapsulations to hold file variants and variation point logic files. The variation point instantiations can simply be files and directories.

In another asset integration example, variation points in a UML™ model could be any of the model elements. In the IBM Rational Rhapsody® integration – the *Rhapsody/Gears Bridge* – UML "tags" within a model element are used to hold the variation point logic and the optional and alternative variants for the model element. The variation point instantiation for a model element is reflected in the executable code that Rhapsody generates off of a model element variation point as well as visual indicators as to which model elements are excluded in a particular model instantiation [7].

In the *DOORS/Gears Bridge* requirements integration into the framework, a requirement and its descendants can become a variation point. The instantiation logic for a variation point is contained in a DOORS attribute. For requirements alternatives in a variation point, child requirements are specially tagged as *variants* for the parent requirement. The variation point instantiation is displayed in a separate database column [8].

These illustrate different kinds of variation points for different kinds of tool-supported assets, but their adherence to the PLE Lifecycle Framework means that in all cases:

- exercising the variation points – that is, producing an instance of the asset for a product – is driven by a selection of features in the profile for that product
- the logic to express the choice uses the same language
- the variation mechanisms available are the same across all of the assets

5. Complete and Consistent End-to-End PLE Lifecycle

The uniformity of concepts and constructs provided by the PLE Lifecycle Framework allows traceability and processes to flow cleanly among different lifecycle stages, without the dissonance created by different techniques in different lifecycle stages.

Figure 4 illustrates this by showing and example of the classic V-model for systems and software engineering. For PLE-aware tools, each of the lifecycle phases (shown as colored parallelograms), is augmented by the addition of variation points (shown as gear symbols) to the assets native to that tool and phase.

A *Bill-of-Features* in the top-center of the figure corresponds to the feature selections in the feature profile for a product. The name is meant to elicit the analogy to the *Bill-of-Materials*, a term that is used in mechanical design to designate the listing of parts that comprise a product. By this analogy, the Bill-of-Features is the listing of PLE features that comprise a product.

The yellow arrows illustrate that all of the variation points in all of the assets across the full lifecycle are synchronously and consistently configured according to the single consolidated collection of feature selections in the Bill-of-Features.

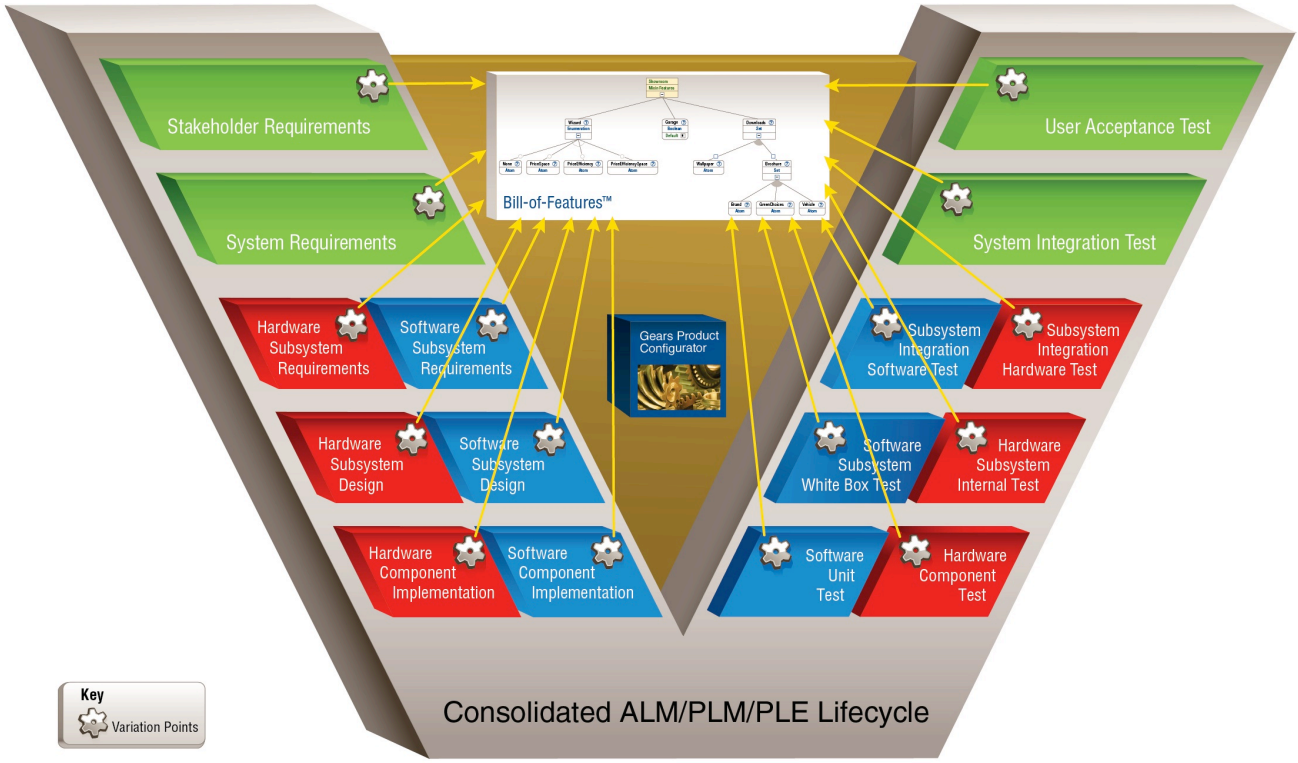

Figure 4. Complete and Consistent Systems and Software PLE V-Model

5.1. Traceability

There are three simple heuristics for defining and managing traceability among the assets in different stages of the lifecycle.

- Traceability among the common parts of the assets is managed identical to the way it is managed with one-of-a-kind products. Common requirements trace to common design elements, which trace to common source code elements, and so forth.
- Traceability among the variation points in the assets is managed similarly. Variation points in requirements trace to variation points in design elements, which in turn trace to variation points in source code, and so forth.
- If there is a traceability relationship between a common asset in one stage and an optional variation point in another stage (or vice versa), it is most likely an error in the traceability relationship. This type of traceability consistency analysis can be automated by tools in the framework.

Traceability with the PLE Lifecycle Framework is remarkably simple compared to the intractable complexity described earlier when trying to define traceability relationships among disparate PLE point solutions in different lifecycle silos.

5.2. Process Flow

The flow of engineering processes among different lifecycle stages closely mirrors traceability. For example, passing a new set of requirements to an architecture and design team leads to new design models, with traceability relationships between the requirements and design. If the intended traceability in the requirements is difficult to extract and express, then there will be dissonance in the process between the requirements and design stages in the lifecycle.

Thus, the shared PLE concepts and constructs in the different lifecycle stages, plus the simplicity of traceability shown in the previous section translates to simplicity in the processes at the lifecycle boundaries.

Furthermore, since the framework feature model and feature profiles are shared across all lifecycle stages, assets in all stages are constructed under the same PLE feature set. This makes it easier to understand assets and their variations in multiple stages when processes depend on it.

6. Conclusions

With the Gears PLE Lifecycle Framework providing a common set of PLE concepts and constructs at every stage in the systems and software engineering lifecycle, the engineering of assets for a product line can be defined, viewed and managed as a *single production system*, rather than a collection of disparate silos for each of the lifecycle stages. This single production system represents a complete PLE solution that is readily adopted by mainstream engineering organizations.

References

1. Geoffrey A. Moore, *Crossing the Chasm*. HarperCollins Publisher, New York, NY, 1991.
2. *The Pragmatic 3-Tiered Product Line Engineering Methodology*. BigLever Software technical report #200709261r3, January 2013.
3. Clements, P.; Northrop, L. *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2002.
4. Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering, ACM, 2005, Long Beach, California November, 2005.
5. Flores, R., Krueger, C., Clements, P. "Mega-Scale Product Line Engineering at General Motors," *Proceedings of the 2012 Software Product Line Conference (SPLC)*, Salvador Brazil, August 2012.
6. Krueger, C; Clements, P. "Systems and Software Product Line Engineering," *Encyclopedia of Software Engineering*. Taylor & Francis Online, 2013.
7. Krueger, C; Bakal, M. *Systems and software product line engineering with SysML, UML and the IBM Rational Rhapsody BigLever Gears Bridge*. IBM Rational technical report, July 2007.
8. Krueger, C; Jackson, K. *Requirements Engineering for Systems and Software Product Lines*. IBM Rational technical report, January 2009.

