INSIGHT

This Issue's Feature: Product Line Engineering in Context

APRIL 2021
VOLUME 24 / ISSUE 1

INCOSE®

A PUBLICATION OF THE INTERNATIONAL COUNCIL ON SYSTEMS ENGINEERING

Systems Engineering: The Journal of The International Council on Systems Engineering

Call for Papers

The *Systems Engineering* journal is intended to be a primary source of multidisciplinary information for the systems engineering and management of products and services, and processes of all types. Systems engineering activities involve the technologies and system management approaches needed for

- definition of systems, including identification of user requirements and technological specifications;
- development of systems, including conceptual architectures, tradeoff of design concepts, configuration management during system development, integration of new systems with legacy systems, integrated product and process development; and
- deployment of systems, including operational test and evaluation, maintenance over an extended life cycle, and re-engineering.

Systems Engineering is the archival journal of, and exists to serve the following objectives of, the International Council on Systems Engineering (INCOSE):

- To provide a focal point for dissemination of systems engineering knowledge
- To promote collaboration in systems engineering education and research
- To encourage and assure establishment of professional standards for integrity in the practice of systems engineering
- To improve the professional status of all those engaged in the practice of systems engineering
- To encourage governmental and industrial support for research and educational programs that will improve the systems engineering process and its practice

The journal supports these goals by providing a continuing, respected publication of peer-reviewed results from research and development in the area of systems engineering. Systems engineering is defined broadly in this context as an interdisciplinary approach and means to enable the realization of successful systems that are of high quality, cost-effective, and trustworthy in meeting customer requirements.

The *Systems Engineering* journal is dedicated to all aspects of the engineering of systems: technical, management, economic, and social. It focuses on the life cycle processes needed to create trustworthy and high-quality systems. It will also emphasize the systems management efforts needed to define, develop, and deploy trustworthy and high quality processes for the production of systems. Within this, *Systems Engineering* is especially concerned with evaluation of the efficiency and effectiveness of systems management, technical direction, and integration of systems. *Systems Engineering* is also very concerned with the engineering of systems that support sustainable development. Modern systems, including both products and services, are often very knowledge-intensive, and are found in both the public and private sectors. The journal emphasizes strategic and program management of these, and the information and knowledge base for knowledge principles, knowledge practices, and knowledge perspectives for the engineering of systems. Definitive case studies involving systems engineering practice are especially welcome.

The journal is a primary source of information for the systems engineering of products and services that are generally large in scale, scope, and complexity. *Systems Engineering* will be especially concerned with process- or product-line–related efforts needed to produce products that are trustworthy and of high quality, and that are cost effective in meeting user needs. A major component of this is system cost and operational effectiveness determination, and the development of processes that ensure that products are cost effective. This requires the integration of a number of engineering disciplines necessary for the definition, development, and deployment of complex systems. It also requires attention to the lifecycle process used to produce systems, and the integration of systems, including legacy systems, at various architectural levels. In addition, appropriate systems management of information and knowledge across technologies, organizations, and environments is also needed to insure a sustainable world.

The journal will accept and review submissions in English from any author, in any global locality, whether or not the author is an INCOSE member. A body of international peers will review all submissions, and the reviewers will suggest potential revisions to the author, with the intent to achieve published papers that

- relate to the field of systems engineering;
- represent new, previously unpublished work;
- advance the state of knowledge of the field; and
- conform to a high standard of scholarly presentation.

Editorial selection of works for publication will be made based on content, without regard to the stature of the authors. Selections will include a wide variety of international works, recognizing and supporting the essential breadth and universality of the field. Final selection of papers for publication, and the form of publication, shall rest with the editor.

Submission of quality papers for review is strongly encouraged. The review process is estimated to take three months, occasionally longer for hard-copy manuscript.

Systems Engineering operates an online submission and peer review system that allows authors to submit articles online and track their progress, throughout the peer-review process, via a web interface. All papers submitted to *Systems Engineering*, including revisions or resubmissions of prior manuscripts, must be made through the online system. Contributions sent through regular mail on paper or emails with attachments will not be reviewed or acknowledged.

All manuscripts must be submitted online to *Systems Engineering* at ScholarOne Manuscripts, located at:

http://mc.manuscriptcentral.com/SYS

Full instructions and support are available on the site, and a user ID and password can be obtained on the first visit.

INSIGHT

A PUBLICATION OF THE INTERNATIONAL COUNCIL ON SYSTEMS ENGINEERING

APRIL 2021 VOLUME 24 / ISSUE 1

Inside this issue

About This Publication

INFORMATION ABOUT INCOSE

INCOSE's membership extends to over 18,000 individual members and more than 100 corporations, government entities, and academic institutions. Its mission is to share, promote, and advance the best of systems engineering from across the globe for the benefit of humanity and the planet. INCOSE charters chapters worldwide, includes a corporate advisory board, and is led by elected officers and directors.

For more information, click here:
The International Council on Systems Engineering
(www.incose.org)

OVERVIEW

INSIGHT is the magazine of the International Council on Systems Engineering. It is published four times per year and features informative articles dedicated to advancing the state of practice in systems engineering and to close the gap with the state of the art. ***INSIGHT*** delivers practical information on current hot topics, implementations, and best practices, written in applications-driven style. There is an emphasis on practical applications, tutorials, guides, and case studies that result in successful outcomes. Explicitly identified opinion pieces, book reviews, and technology roadmapping complement articles to stimulate advancing the state of practice. ***INSIGHT*** is dedicated to advancing the INCOSE objectives of impactful products and accelerating the transformation of systems engineering to a model-based discipline.
Topics to be covered include resilient systems, model-based systems engineering, commercial-driven transformational systems engineering, natural systems, agile security, systems of systems, and cyber-physical systems across disciplines and domains of interest to the constituent groups in the systems engineering community: industry, government, and academia. Advances in practice often come from lateral connections of information dissemination across disciplines and domains. ***INSIGHT*** will track advances in the state of the art with follow-up, practically written articles to more rapidly disseminate knowledge to stimulate practice throughout the community.

ARTICLE SUBMISSION

insight@incose.net

Publication Schedule. ***INSIGHT*** is published four times per year. Issue and article submission deadlines are as follows:
- March 2021 issue – 2 January
- June 2021 issue – 2 April
- September 2021 issue – 1 July
- December 2021 issue – 1 October

For further information on submissions and issue themes, visit the INCOSE website: www.incose.org

ISSN 2156-485X; (print) ISSN 2156-4868 (online)

ADVERTISE

Readership
INSIGHT reaches over 18,000 individual members and uncounted employees and students of more than 100 CAB organizations worldwide. Readership includes engineers, manufacturers/purchasers, scientists, research & development processionals, presidents and CEOs, students and other professionals in systems engineering.

Issuance	Circulation
2021, Vol 24, 4 Issues	100% Paid

Contact us for Advertising and Corporate Sales Services
We have a complete range of advertising and publishing solutions professionally managed within our global team. From traditional print-based solutions to cutting-edge online technology the Wiley-Blackwell corporate sales service is your connection to minds that matter. For an overview of all our services please browse our site which is located under the Resources section. Contact our corporate sales team today to discuss the range of services available:
- Print advertising for non-US journals
- Email Table of Contents Sponsorship
- Reprints
- Supplement and sponsorship opportunities
- Books
- Custom Projects
- Online advertising

Click on the option below to email your enquiry to your nearest office:
- Asia & Australia corporatesalesaustralia@wiley.com
- Europe, Middle East & Africa (EMEA) corporatesaleseurope@wiley.com
- Japan corporatesalesjapan@wiley.com
- Korea corporatesaleskorea@wiley.com

USA (also Canada, and South/Central America):
- Healthcare Advertising corporatesalesusa@wiley.com
- Science Advertising Ads_sciences@wiley.com
- Reprints Commercialreprints@wiley.com
- Supplements, Sponsorship, Books and Custom Projects busdev@wiley.com

Or please contact:
Susan Blessing, Senior Account Manager Sciences
Sciences, Corporate Sales
MOBILE: 24/7 201-723-3129
E-MAIL: sblessin@wiley.com

CONTACT

Questions or comments concerning:

Submissions, Editorial Policy, or Publication Management
Please contact: William Miller, Editor-in-Chief
insight@incose.net

Advertising—*please contact:*
Susan Blessing, Senior Account Manager Sciences
Sciences, Corporate Sales
MOBILE: 24/7 201-723-3129
E-MAIL: sblessin@wiley.com

Member Services – ***please contact:*** info@incose.net

ADVERTISER INDEX Apr volume 24-1

***Systems Engineering* Call for Papers**	inside front cover
16th Annual Systems of Systems Engineering Conference	back inside cover
31st Annual INCOSE International Symposium	back cover

APRIL 2021
VOLUME 24/ ISSUE 1

FROM THE EDITOR-IN-CHIEF

William Miller, insight@incose.net

It is our pleasure to announce the April 2021 *INSIGHT* issue published cooperatively with John Wiley & Sons as the systems engineering practitioners' magazine. The *INSIGHT* mission is providing informative articles on advancing the systems engineering practice and to close the gap between practice and the state of the art as advanced by *Systems Engineering*, the Journal of INCOSE also published by Wiley. The issue theme is product line engineering in context. We thank theme editor Drew Stovall, the Product Line Engineering International Working Group, and the authors for their contributions.

Your editor appreciates the author contributions from professional experience in the early 1980s transformation from analog to digital telephone systems. I had a rotational assignment in the early 1980s from Bell Labs to be the product manager for on-premises digital telephone system adjunct (feature) processors. This was the first commercial Unix operating system use. The growth of the installed base, each unique configuration, with churn in circuit pack upgrades, software versioning, enabled features, along with customer-driven feature enhancements and new features over the adjunct processor platform architecture lifecycle was truly mind-boggling.

Paul Clements begins with "Temporal Management in Feature-Based Product Line Engineering." Clements presents an approach for handling the *temporal dimension* of product line engineering (PLE)— managing artifacts as they change and evolve. The approach relies on a proven traditional change control technique foundation but shows how they apply in the feature-based product line engineering context.

John Wood and Glenn Tolentino address "Product Line Re-Engineering for Modularity in a US Department of Defense Project." Their case study details the options evaluated and the path chosen by a software development organization to re-engineer four existing products with common features into a single product-line resulting in product sponsors taking advantage of cost savings, developers shortening implementation and testing timeframes, and users obtaining product features faster while sharing a common experience across product variants. Although they originally envisioned the software-intensive products to be a product line operating from a common code repository, they diverged due to different product sponsors having differing priorities and schedule commitments. The evaluated re-engineering options included merging common code and maintaining it in a single repository, re-using software code while keeping it in separate repositories for each product variant, and pursuing a modular open systems approach (MOSA) to create common modules that could insert, update, replace, and so forth within any product variant without disrupting the remaining product.

Clements next addresses "Funding the PLE Factory in a Multi-Customer Contract-Based PLE Organization." Feature-Based product line engineering employs the *PLE factory* concept, in which all development occurs for any products in a product line. Automatically configuring shared assets based on the feature choices for a product produces individual products. A product line organization's personnel need to carry out tasks associated with creating, developing, delivering, and evolving products in its product line. Any organization employing this paradigm in a contract-based (as opposed to a mass market) context must answer the question: who pays for the work going on inside the factory that may benefit multiple contracts? The answer can be surprisingly complex, involving security, regulatory compliance, and intellectual property protection issues of both the PLE organization and its customers. This report offers a method for answering "Who pays for the activities in the PLE Factory?" Answering this question means establishing processes culminating with creating charge numbers to which everyone working in the PLE Factory can charge their effort. These processes must connect the funding supply to the funding consumption in a fair and equitable way that complies with applicable rules and regulations.

Evan Helmeid addresses "Development of a Hybrid Product Breakdown Structure and Variability Model." Transforming from a project-based engineering approach to a product line engineering approach requires supporting the engineering teams throughout the transition with evolving tools and methodologies. However, a traditional product breakdown structure (PBS) format provides insufficient detail and structure. The author developed a hybrid PBS-variability model (VM) using standard desktop software, combining the familiar PBS structure with variability modeling aspects based on feature modeling and decision modeling approaches, resulting in an engineering artifact recognizable as a PBS and easy to adapt to design evolution, yet sufficiently expansive to characterize initial variability.

Thomas Froment and Guillaume Angier de Loheac take on "The Convergence of Struggles! Reusability Assessment of Inner-Source Components for Product Lines." Inner source establishes open source-like collaborations within an organization. Product line engineering is the approach for engineering a related product portfolio in an efficient manner, taking advantage of products' similarities while managing their

> continued on page 11

APRIL 2021
VOLUME 24/ ISSUE 1

It's About Time: Temporal Management in Feature-Based Product Line Engineering

Paul Clements, pclements@biglever.com

■ **ABSTRACT**
Feature-based product line engineering employs the PLE factory concept in which all development occurs for any product line product. Any organization employing this paradigm in a contract-based (as opposed to a mass market) context must answer the question: Who pays for the work inside the factory that may benefit multiple contracts? The answer can be surprisingly complex, involving security, regulatory compliance, and intellectual property protection issues of both the PLE organization and its customers. This article offers a method for answering this question by establishing processes to create charge numbers to which everyone working in the PLE factory can charge their effort. These processes must connect the funding supply to the funding consumption in a way that is fair, equitable, and compliant with applicable rules and regulations.

INTRODUCTION

Feature-Based Product Line Engineering (PLE) is a well-known approach for efficiently engineering product lines, which numerous case studies have shown to yield substantial benefits in cost, quality, and time to market. The technical approach centers around the factory concept, configuring shared assets to support any product line member based on a product description in terms of its features. However, we need another ingredient to this narrative before we can apply it in day-to-day operational practice: managing change and evolution. This article presents an approach for handling the product line engineering *temporal dimension*—managing artifacts as they change and evolve. The approach relies on a foundation of proven traditional change control techniques but shows how they apply in the context of Feature-Based PLE.

Feature-Based PLE Overview

A product line includes various engineering assets, such as system or software requirements, design documentation, software source code, test cases and procedures, and more, that play a role in product creation, deployment, and sustainment.

The product line shares these engineering assets. These shared assets are *supersets*, meaning they contain any content needed to support any product. The configurator is

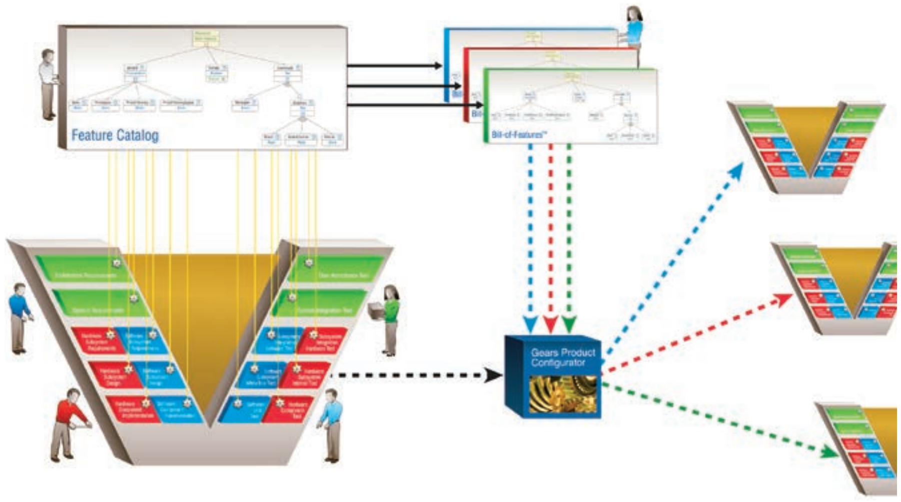


Figure 1. The Feature-Based PLE factory configuring PLE shared asset supersets into product-specific instances according to a product's bill-of-features

a commercial software tool, such as Gears, producing product-specific instances by *actuating* a product—exercising variation points in the supersets according to the feature choices for that product.

Figure 1 shows the shared asset supersets in the bottom left "V.". Gear symbols denote variation points defined in feature terms in the product line's feature catalog. A bill-of-features describes the particular product's feature choices, which the configurator uses to produce product-specific shared asset instances. In Figure 1, the PLE factory comprises everything left of the product subsets.

In the Feature-Based PLE world, the factory naturally evolves over time—the shared asset supersets change, the feature catalog changes, the bills-of-features change all in response to changing needs of the customers or market. To change the product instances, a development team first changes the factory and then uses the configurator to produce the new version of the affected product or products.

This article focuses on managing this factory evolution.

Hereafter, the term *production line* will refer to the physical tool-and-technology-based realization of a PLE factory's feature catalog and bills-of-features. These comprise the physical artifacts (data files) that need managing over time, along with the shared asset supersets.

EVOLUTION OF SHARED ASSETS AND PRODUCTION LINE FILES

Production line files and shared asset supersets are both critical components of a valid and operational factory. Variation points within shared asset supersets refer to features defined in production line files; a variation point denotes content that applies to a product for which certain features have been selected. Conversely, production line files contain references to shared asset supersets. This coupling requires coordinating production line file baselines and shared asset baselines as the product line evolves.

One or more configuration management (CM) repositories should control the product line's shared asset supersets *and* production line files to provide the production line history, version control, and reproducibility at specific points in time.

- **Shared Asset Evolution:** All shared asset supersets naturally evolve to support new functionality, provide improvements, and/or address defects. Development typically occurs on shared asset supersets using various tools, selected by the PLE organization for each asset type: requirements tools, modeling tools, software development environments, documentation tools, and so forth.
- **Production Line File Evolution:** Production line files include files created by the configurator. In Gears, these are plain-text files representing the production line's architecture, feature catalog, bills-of-features, and shared asset locations. As these artifacts change over time, the production line files will also evolve. This change occurs, for instance, in conjunction with adding new capability to the product line, which may involve creating a new feature or a new "flavor" of an existing feature. Change can also occur when a customer wants a capability that is already part of the product line's repertoire leading to a new bill-of-features to support the customer's product.

Of course, a single product line change can cause a change to both the production line files and shared asset supersets. A new feature, for example, will cause a change to the feature catalog, one or more bills-of-features (for each product selecting the new feature), and one or more shared asset supersets to implement the new capability.

KEY PRINCIPLES FOR FEATURE-BASED PLE TEMPORAL MANAGEMENT

The temporal management approach for Feature-Based PLE builds on two key principles:

(1) Perform all development and maintenance modifications on PLE shared asset supersets and/or production line files.

(2) Never use the read-only product-specific asset subsets (actuation results, the right side of Figure 1) for development and maintenance.

These principles are independent of the CM tool, repository, or shared asset type under CM control.

Single-system temporal management (and older copy-based PLE approaches) maintain and manage the engineering artifacts for each product separately. The Feature-Based PLE temporal management strategy focuses on the shared asset supersets, not product-specific instances, and is a major cost avoidance source enjoyed by PLE organizations.

TEMPORAL BASELINES: THE KEY CONCEPT FOR TEMPORAL MANAGEMENT

The Feature-Based PLE temporal management is based on the *temporal baseline* concept, which is essentially a product-line-level baseline comprising the baseline set of each shared asset superset and the production line files themselves.

Thus, a temporal baseline, as Figure 2 illustrates, includes:

- An individual shared asset baseline set;
- A baselined production line file set corresponding to the shared asset baselines.

The temporal baseline strategy uses whatever baseline strategy the organization has already chosen to put in place for each different CM repository. For example:

- Assets using a conventional commercial configuration management tool including plain text files (text files containing computer source code or build scripts), documents in Microsoft® Word, spreadsheets in Microsoft® Excel, and the like. These assets generally do not require a specialized viewing tool.
- Assets residing in a database or otherwise managed by a special-purpose tool. Those can use the built-in host database or tool CM capabilities.

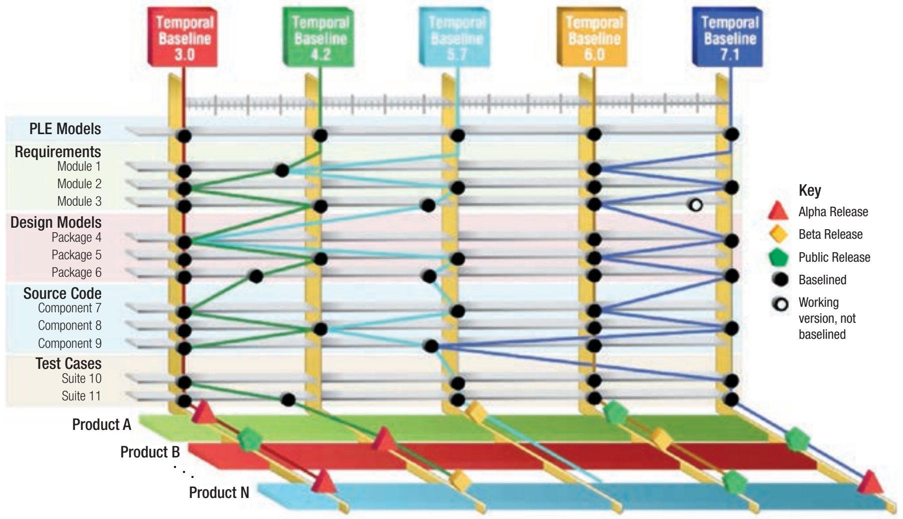


Figure 2. PLE Temporal Management with Temporal Baselines

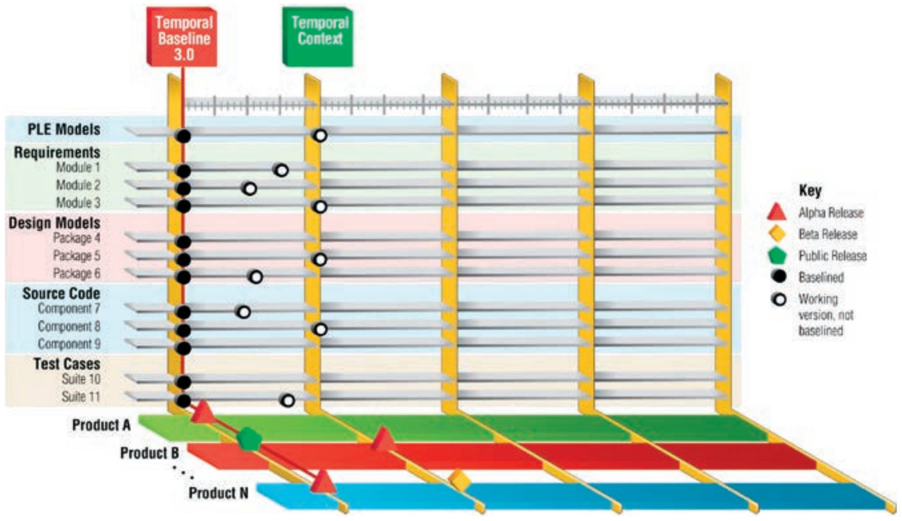


Figure 3. Temporal Baseline 3.0 plus working versions

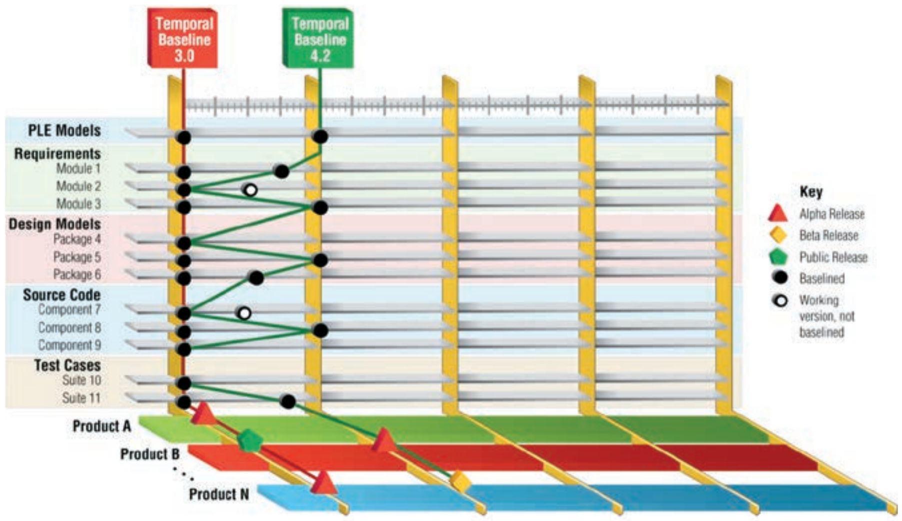


Figure 4. Temporal Baseline 4.2

- Assets effectively using content and naming convention copies to represent versions.

Figure 2 illustrates a product line temporal baseline series. A temporal baseline comprises the colored box across the top of the figure (such as Temporal Baseline 3.0), along with the "zigzag" line of the corresponding color running through shared asset supersets indicating a version of that shared asset contributing to the temporal baseline. The solid black dots represent the baselined versions of each shared asset of PLE models.

Each temporal baseline supports releasing one or more products; represented by the colored bands at the bottom of the figure labeled Product A through Product N.

The light blue band, labeled "PLE Models," shown at the top of the asset type list in Figure 2, represents the production line files. These files, like the shared asset supersets, need CM repository revision-control.

Temporal Baselines and Product Line Evolution

Figure 2 shows a product line after it has undergone several evolutionary steps. To delve deeper, let us return to the starting point.

Figure 3 shows Temporal Baseline 3.0, depicted as the dark red line connecting each asset's baseline. Temporal Baseline 3.0 supports Product A's alpha, Product B's public release, and Product N's alpha release.

Figure 3 also illustrates the production line file evolution and at least some shared asset supersets as development occurs. The white dots represent the current working versions and are the "tips" in the various CM repositories. These "working versions" may themselves baseline many times, even many times daily, as they evolve, depending upon an organization's development practices. When we refer to baselines, we usually mean baselines that are part of a product release. Not everything has evolved; notice Design Package 4, Source Code Component 9, and Test Case Suite 10 have not changed.

Now another release cycle coming due will include a Product B alpha version and a Product N beta version. We baseline the production line files and those shared asset supersets ready to support the product releases; in our figures, those white dots become black dots. Then we define the versions supporting and defining a temporal baseline, which we call Temporal Baseline 4.2. Figure 4 shows this. In Figure 4, Requirements Module 2 and Source Code Component 7 both evolved since their status in Temporal Baseline 3.0, but they are not ready to support the next releases. So, we leave them in their working state and incorporate the previously baselined versions for Temporal Baseline 4.2.

As development continues, the production line files and the shared asset supersets are periodically baselined. These baselines combine as needed to form the temporal baselines supporting the specific versions of any or all products throughout the product line lifecycle. The process continues, resulting in the full picture shown in Figure 2.

Using a Temporal Baseline

Temporal baselines define and re-create any product version at any time. For example, in Figure 2, using Temporal Baseline 5.7 can re-create Product A's public release (the green pentagon in the "Product A" line) at any time. To do so:

- use the shared asset versions specified in Temporal Baseline 5.7.
- use the production line file versions specified in Temporal Baseline 5.7.
- use the configurator to actuate those shared asset supersets according to the bill-of-features for Product A.

A temporal baseline should track all product line shared asset supersets, even those not containing variation points.

Storing a Temporal Baseline

A temporal baseline is essentially a list of shared asset supersets and version numbers of those shared asset supersets, plus a version number for the production line files. Here are two recommended approaches for storing a temporal baseline:

- **Option 1: Uniform version numbers for everything.** Under this approach, the production line files, all shared asset

Table 1: Temporal Baseline representation

Temporal baseline label:	3.0	4.2	5.7	6.0	7.1
Production line files	1.6	1.10	1.14	2.0	2.3
Requirements module 1	6.2.5	6.3.6	6.3.6	6.5.0	6.5.0
Requirements module 2	5.5.1	5.5.1	5.7.2	5.8.0	5.8.5
Requirements module 3	4.4.3	4.4.7	4.4.12	4.5.0	4.5.0
Design package 4	2.1	2.1	2.1	3.0	3.2
Design package 5	1.2	1.3	1.4	2.0	2.0
Design package 6	1.4	1.4.2	1.4.4	1.5.0	1.5.5
Source code component 7	3.6.3	3.6.3	3.6.14	4.0	4.0
Source code component 8	4.2.1	4.2.4	4.2.4	4.3.0	4.3.5
Source code component 9	4.9.1	4.9.1	4.9.17	5.0	4.9.17
Test case suite 10	4.1	4.1	4.2	4.3.0	4.3.9
Test case suite 11	4.5	4.7	4.8	5.0	5.1
Product A	V1 Alpha	-	V1 Beta	V1 Public	-
Product B	V1 Public	V2 Alpha	-	V2 Beta	V2 Public
…	…	…	…	…	…
Product N	V1 Alpha	V1 Beta	-	V1 Public	V2 Alpha

supersets, and the product releases have the same version number—3.0, 4.2, and 5.7—in the CM system(s) in which they reside. Thus, in Figure 2, every Temporal Baseline 5.7 component has version number 5.7 in its CM system, and the Product A through N releases also have the release number 5.7.

- This approach removes the need to store the temporal baseline—everything associated with a release has the same number. This approach has two caveats:
  1. Even if a shared asset has not evolved since the previous temporal baseline, it can (and should) simply receive a new label (version number) as is, to keep its numbering in sync with other artifacts in the new temporal baseline.
  2. Some tools do not support assigning user-defined labels to content versions, and so their assets will not participate in this scheme.

- **Option 2:** Table. A simple table (maintained, for example, in a spreadsheet) can represent and store a temporal baseline. Table 1 uses the Figure 2 temporal baselines as an example. This table becomes a product line asset in its own right and should remain under configuration control.

BRANCHING

The fundamental approach is to maintain versions of all asset supersets plus the production line files, as shown in Figure 2. Taken together, this constitutes a temporal baseline. Each can evolve as it needs to, and we have an elegant, simplified picture of change management for a PLE Factory that eliminates all of the branching that comes with clone-and-own approaches and product-centric-CM approaches. Everybody works on the "tip" (most recent versions); re-producing an earlier version of a product release simply means finding the temporal baseline of that version, checking out all of the associated versions of all of the shared asset supersets and production line files, and using the configurator to reproduce the product.

It would be tempting to conclude at this point but in the interest of practical application we will point out that there are cases where we still need branches — at least short-lived ones.

Branching means to make a copy of an artifact so that special-purpose development can continue in parallel with "main" development activity. (Some CM systems use the word "stream" to refer to a branch.) The "main" branch is called the trunk. A branch generally ends when it is merged back into the branch that spawned it. Merging means reconciling changes in the branch with the branch that spawned it; the reconciliation happens in the latter.

A direct corollary of PLE Temporal Management principles (1) and (2) described earlier having to do with branching is:

- Never use product-specific PLE subsets (actuated artifacts) to create a branch. Branch using only PLE Shared Asset Supersets and the production line files themselves.

Put another way, make sure that branching only occurs inside the PLE factory.

Because copying a shared asset and making changes to the copy is generally antithetical to PLE principles, take care when branching to mitigate the effects of having multiple copies in existence that could require duplicative work to update and keep in sync. Nevertheless, there are cases when branching is appropriate in Feature-based PLE and, in fact, a normal aspect of Feature-based PLE development, and these are described in Table 2. Branching strategies apply to production line files as well as to shared asset supersets, although in the remainder of this section we focus on shared asset supersets to simplify the exposition.

CHANGE MANAGEMENT AND GOVERNANCE

In a disciplined temporal management environment, changes to shared asset supersets and the production line files only occur under a well-defined process including a documented change artifact such as a change request or a problem report and an appointed body or board authorized to approve changes. The policies associated with creating, submitting, and handling these change artifacts are critical, but beyond this document's scope. The article "Key Issues of Organizational Structure and Processes with Feature-Based Product Line Engineering" elsewhere in this special edition of *INSIGHT* addresses this issue.

SUMMARY

The distinguishing characteristic between Feature-Based PLE and product-centric development (and development under earlier copy-based PLE forms) is its greatly simplified strategy for managing change. It eliminates the need for performing change control on the generated products, instead focusing the change management efforts on the much smaller material in the shared asset supersets and production line files.

This eliminates the bottomless branching-and-merging morass many organizations find themselves having to manage, and that is for organizations that do not simply give up the merging part and accept separately-managed clones spiraling off on their own separate evolutionary trajectories, resulting in a situation in which it is almost impossible to economically carry out a portfolio-wide upgrade.

It also provides an environment in which engineers can spend the maximum time on high-value new development, and not low-value activities such as branching, merging, and re-developing already-built capabilities.

Just as Feature-Based PLE calls for all development to occur in the shared asset supersets and not the product instances—that is, inside the PLE Factory—its change management method takes the same approach. Both maximize the extent to which work applies to the entire product line, and not just an individual product, and contribute to the cost avoidance for which Feature-Based PLE is known. ■

ABOUT THE AUTHOR

Dr. Paul Clements is the Vice President of Customer Success at BigLever Software, Inc., where he works to spread the adoption of systems and software product line engineering. Prior to this, he was a senior member of the technical staff at Carnegie Mellon University's Software Engineering Institute, where for 17 years he worked leading or co-leading projects in software product line engineering and software architecture design, documentation, and analysis. Prior to the SEI, he was a computer scientist with the U.S. Naval Research Laboratory in Washington, D. C., where his work involved applying advanced software engineering principles to real-time embedded systems. Clements is the co-author of eight practitioner-oriented books about software architecture and product line engineering. In addition, he has also authored over one hundred of papers in software engineering reflecting his long-standing interest in the design and specification of challenging software systems.

From the Editor-In-Chief cont. from page 6

differences. These two approaches propose smart techniques for reuse but use different terminology to refer to equivalent concepts, which can badly affect project performance when evolving in a multi-domain context. This paper shows it is possible to build a common way to assess the components (also called building blocks) contributing to a product line, thanks to a process to determine the component maturity level using the *similarity* approach. The authors introduce the inner sourcing process maturity level (ISPML) as a simple engineering practice for multi-domain organizations to better determine whether sharing an engineering asset is favorable or not.

Guillermo Chalé Góngora, Pierre-Olivier Robic, and Danilo Beuche address the topic of "Product Line Engineering for Digital Product-Services." Digitizing the value chain brings along new business opportunities to organizations wishing to adopt a service-oriented approach but incurring implementation challenges. The authors present a conceptual framework to define a high-level strategy to implement a product-service offer in an organization. The distinctive framework aspects include the product-service product line (PSPL) concept, that is a product line of product-services, the elements to define the PSPL business model, a product-service typology, and a product line engineering method extension for architecting the PSPL, notably, a specific service building block type supporting a composable design approach and a feature model including service-related, socio-technical features.

The final article by William Bolander and Paul Clements, "Key Issues of Organizational Structure and Processes with Feature-Based Product Line Engineering," describes the transformation organizations should undertake to standup feature-based PLE based on the factory concept. The authors introduce the few roles without analog in other development disciplines but that are new to feature-based PLE. They also describe how traditional systems engineering roles doing traditional systems engineering tasks but with slight PLE-inspired extensions carry out other factory roles.

We hope you find *INSIGHT*, the practitioners' magazine for systems engineers, informative and relevant. Feedback from readers is critical to *INSIGHT*'s quality. We encourage letters to the editor at insight@incose.net. Please include "letter to the editor" in the subject line. *INSIGHT* also continues to solicit special features, standalone articles, book reviews, and op-eds. For information about *INSIGHT*, including upcoming issues, see https://www.incose.org/products-and-publications/periodicals#INSIGHT. For information about sponsoring *INSIGHT*, please contact the INCOSE marketing and communications director at marcom@incose.net. ■

APRIL 2021
VOLUME 24/ ISSUE 1

Product Line Re-Engineering for Modularity in a US Department of Defense Project

John Wood, john.n.wood@navy.mil, and **Glenn Tolentino**, glenn.tolentino@navy.mil

■ ABSTRACT
This case study details the options evaluated and path chosen by a United States (US) Department of Defense (DoD) software development organization to re-engineer four existing products with common features into a single product-line resulting in product sponsors taking advantage of cost savings, developers shortening implementation and testing timeframes, and users obtaining product features faster while sharing a common experience across product variants. Although the software-intensive products were originally a product line operating from a common code repository, they diverged due to different product sponsors having differing priorities and schedule commitments. The mission scope of each variant differs leading to a commonality range of approximately 20% to 70% based on the quantity of common features. The re-engineering options evaluated included merging common code and maintaining it in a single repository; re-using software code while keeping it in separate repositories for each product variant; and pursuing a Modular Open Systems Approach (MOSA) to create common modules for insertion, updating, and replacement within any product variant without disrupting the rest of that product. With product sponsor support, the DoD project decided to pursue a hybrid approach of immediate code re-use complemented with an agile approach to MOSA implementation. This solution allowed the project to re-engineer the four existing product variants while still meeting sponsor, DoD, and end-user operational needs.

■ KEYWORDS: product line engineering; re-engineering; modular; modular open systems approach; MOSA

BACKGROUND

This case study focuses on a US DoD software development organization that supports naval aviation (Maley, Lofber, and Lasiter 2008, Maley et al. 2009, Schmidley 2011, Tolentino and Wood 2018). The project's flagship product began as an informal community development model with the intent of supporting multiple aircraft types (Maley et al. 2009). Over time, this community model devolved, resulting in four unique, largely independently managed product variants. This community development model devolved due to several factors: unique funding lines required of DoD programs based on US Congressional appropriations, introducing new and replacement sponsors not part of the original informal agreements, differing fielding environments (DoD enterprise networks), and differing user priorities. Now, 10 years later, there are four product variants, three sponsors, two support contracts, and one development organization trying to manage and navigate differences in features, requirements, funding levels, risk profiles, technical refresh priorities, and fielding schedules. This scenario creates pain points common to other projects attempting to satisfy multiple sponsors and stakeholders including those related to leadership, authority, requirements, capabilities, integration, and testing (Dahmann 2014).

Due to the issues discussed above, two sponsors jointly requested the development organization merge common feature source code into a single repository. The sponsors' desire is to experience the benefits associated with a product line such as shorter development timeframes, lower total ownership costs, and higher quality (Pohl, Böckle, and Der Linden 2005). Upon receiving that guidance, the development organization explored the effort required to perform the code merge.

EXPLORING MERGED CODE

Wanting to see more economies of scale, the sponsors of two of the four variants asked the development organization to explore combining common features into a single code repository for use by both variants. This would involve merging separate code branches dedicated to common features, resolving any design conflicts, testing the resultant code, fixing any newly introduced bugs, and retesting the updated code to ensure successful bug fixes. As necessary, developers introduce build-time configuration changes to tailor the common features for each

APRIL 2021 VOLUME 24/ ISSUE 1

Table 1: *Code merge level of effort in Full Time Equivalent (FTE)-weeks*

Activity	Level of Effort -Product A	Level of Effort – Product B
Updates and Integration	12 weeks x 8 FTEs	4 weeks x 7 FTEs
Integration Testing	3 weeks x 9 FTEs	Included in estimate above
Follow-on Development/Bug Fix	4 weeks x 8 FTEs	Included in estimate above
System Testing	3 weeks x 9 FTEs	Included in estimate above
Product Rollout	25 weeks x 2 FTEs	1 week x 1 FTE
Total Estimated Duration	47 weeks	5 weeks
Total Estimated Level of Effort	232 FTE-weeks	29 FTE-weeks

variant. Also, at build-time, the developers compile variant-unique code; maintained in an additional, separate repository.

Building and maintaining merged code has happened before. In fact, as recently as a few months prior, there was a shared code repository for common features; however, the two variants diverged because the sponsors had differing priorities, risk tolerances, and schedule commitments.

The development organization explored the costs of re-merging the common features into a single repository. The development organization realized that effort was more extensive than the sponsors assumed and the burden appeared to be uneven; one sponsor would need to invest significantly more labor than the other due to the product scope differences. See Table 1 above.

Daunted by the numbers, the development organization decided to explore alternative options that could help the two sponsors achieve their desires. The development organization explored and ultimately proposed two additional options: code re-use and MOSA.

EXPLORING CODE RE-USE

Seemingly providing nearly the same benefits of merged code is the code re-use concept. In this scenario, developers from each product team meet regularly to share recent accomplishments, planned activities, challenges, and lessons learned. Beyond sharing approaches to solve common challenges, when practical, the development teams can also share full code sections that they can manually insert into the other variant's code base as-is or with some tailoring. The implementation phase realizes the code re-use benefits, yet they still require extensive testing by the receiving organization. Figure 1 below is a systems engineering vee model. Figure 2 below is the vee model with an overlay showing the required effort for code re-use.

While the initial effort level for code re-use is the same as for code merge (See Table 1 above), the sponsor conflict level related to the code base would be less, because code re-use has the added benefit of decoupling the product variants' schedules. This provides greater flexibility to each sponsor as it lets each focus on different priorities at any given time. It also allows each sponsor to take advantage of previous development efforts at the opportune time for that product variant.

EXPLORING MODULAR OPEN SYSTEMS APPROACH (MOSA)

The third option explored was re-architecting all products in a modular fashion following the DoD's MOSA (DoD 2016). After re-architecting, product teams can develop and test common features once but use them many times. Additionally, by defining and controlling the interfaces, product teams can remove, replace, or upgrade modules independently. Similar to code re-use, this approach decouples the variants' schedules. Further, it significantly reduces the receiving team's testing burden. Figure 3 shows the impact of this approach.

To execute the MOSA option, the development organization expects to invest in an initial effort increase followed by the potential effort decrease during product sustainment, achieving a positive return on investment typical to a product line approach (Walden 2015). More specifically, the MOSA option requires significant preparations (developing operating plans, high-level architecture, and prioritization schema) followed by re-developing functionality in a modular manner. Once

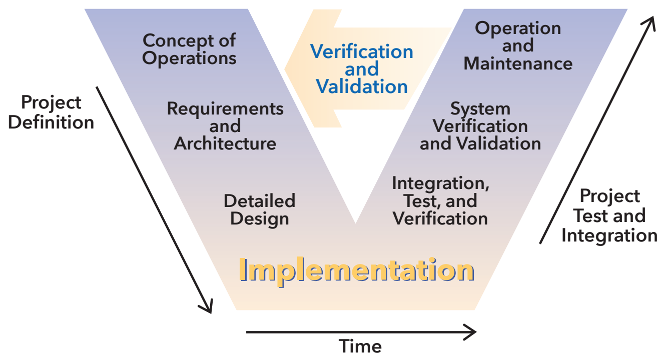
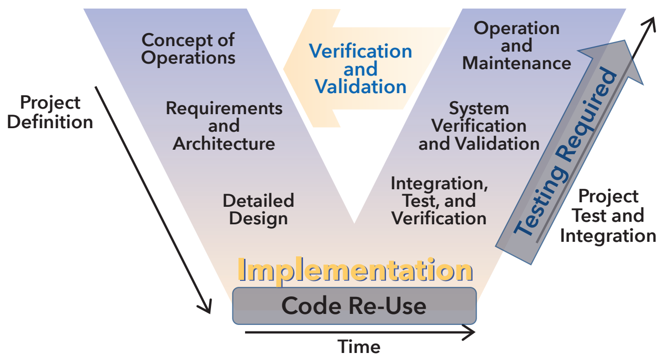

Project Definition
Concept of Operations
Requirements and Architecture
Detailed Design
Verification and Validation
Operation and Maintenance
System Verification and Validation
Integration, Test, and Verification
Project Test and Integration
Implementation
Time

Figure 1. Systems engineering vee model (Osborne et al. 2005)


Project Definition
Concept of Operations
Requirements and Architecture
Detailed Design
Verification and Validation
Operation and Maintenance
System Verification and Validation
Integration, Test, and Verification
Testing Required
Project Test and Integration
Implementation
Code Re-Use
Time

Code re-use results in some savings during the implementation phase, but it still requires significant verification and validation testing prior to fielding.

Figure 2. Code re-use effort on vee model

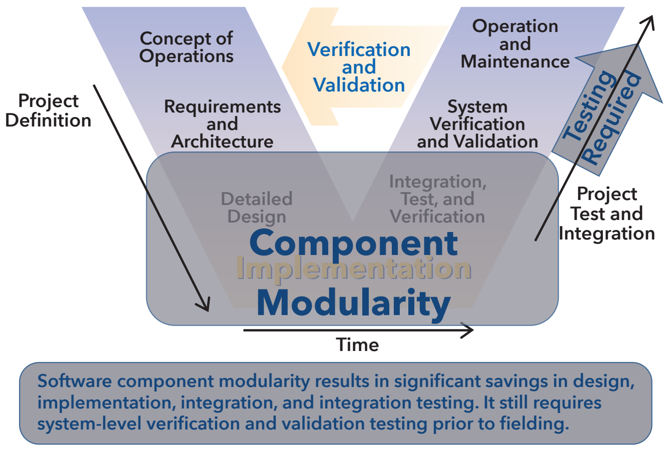

Concept of Operations
Verification and Validation
Operation and Maintenance
Project Definition
Requirements and Architecture
System Verification and Validation
Testing Required
Detailed Design
Integration, Test, and Verification
Project Test and Integration
Component
Implementation
Modularity
Time
Software component modularity results in significant savings in design, implementation, integration, and integration testing. It still requires system-level verification and validation testing prior to fielding.

Figure 3. MOSA effort on vee model

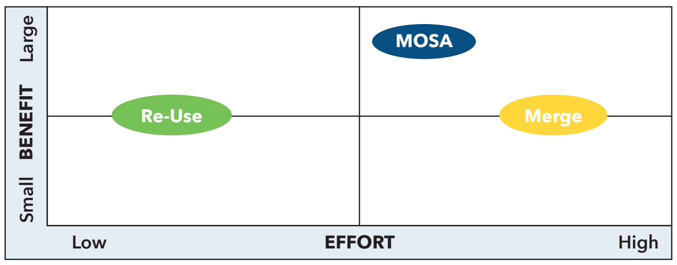


Figure 4. Effort Level versus benefit for code merge, code re-use, and MOSA

the preparatory effort is complete, the product managers envision operating at their current staffing levels of systems engineers, developers, and testers until they implement, certify, and make all modular components available for use. At that point, the product managers envision either a decrease in developers and testers, since the four variants share sustainment efforts, or the ability to leverage the existing developers and testers to provide additional value to the end-users through creating new features.

DECISION SUPPORT

While the development organization was willing to execute any of the three options described above, they believed relaying each option's effort level and benefit to the sponsors was important. To accomplish this, the team created a presentation detailing each option, a diagram depicting the effort level to develop and sustain the approach versus the expected benefit for the three options (Figure 4), and the organization's recommended approach.

After learning about the three options, the sponsors made their decision. First, they decided to abandon the merged code idea. Second, they decided to immediately implement developer exchanges to support maximum code re-use as well as sharing lessons learned. Third, they requested an implementation plan for MOSA as they felt that approach would provide the greatest long-term stability and economies of scale via reduced development timeframe terms, reduced total lifecycle costs, and increased quality.

IMPLEMENTATION

To formalize the interim code re-use, the product managers for the different variants created a charter and agreed to facilitate three meetings per month. The first meeting is the "Developers' Sync" where the development leads discuss recent development efforts, planned development efforts, and any associated challenges. At the end of this meeting, the development teams will choose two efforts or challenges about which they would like more information. Then, the program managers will set up a separate "Deep Dive" meeting for each topic so the developers can explore that area in detail.

Next, the development organization set out to explore what MOSA implementation would require. A literature review provided valuable insights, such as the MOSA framework provided in the *Program Manager's Guide to Open Systems* (DoD 2004), the success factors identified in "Managing Software Productivity and Reuse" (Boehm 1999), and the heuristics-based approach to software re-engineering provided in "An Intelligent Tool for Re-Engineering Software Modularity" (Schwanke 1991). Armed with this information, the development team created a five-step plan.

In Step 1, the development organization will set up an Integrated Product Team (IPT) that simultaneously focuses on both product and process development to execute the MOSA implementation (Magrab et al. 2009). Additionally, to help ensure long-term sustainability, the development organization will designate a governing board and supporting boards that will create and execute the necessary operating plans. In Step 2, the development organization will create a high-level architecture identifying the four variant's functionality overlaps, differences, and gaps. In Step 3, the development team, following an agile approach, will set the priority for converting existing code into the re-usable modular components (Fowler and Highsmith 2001, Tolentino and Wood 2018). Note: Since this is a re-engineering effort, and the development organization is not starting with a clean slate, they will need to balance MOSA efforts with existing requirements, new feature development, technical refreshes, and bug fixes. In Step 4, the development team will create a detailed architecture and design for each modular component. During this step, the development team must ensure the modular component's architecture and design contain sufficient detail to support implementation, verification, and validation efforts. Step 5 will create (implement with software code), verify, and validate the modular components. Completing these efforts for a given modular component certifies that component as compliant with the high-level architecture and makes it available for use by the development team of any product variant. Steps 4 and 5 will repeat for each modular component, following the prioritized order developed in Step 3.

Figure 5 provides a high-level overview of this approach while the following subsections provide further details on the five steps.

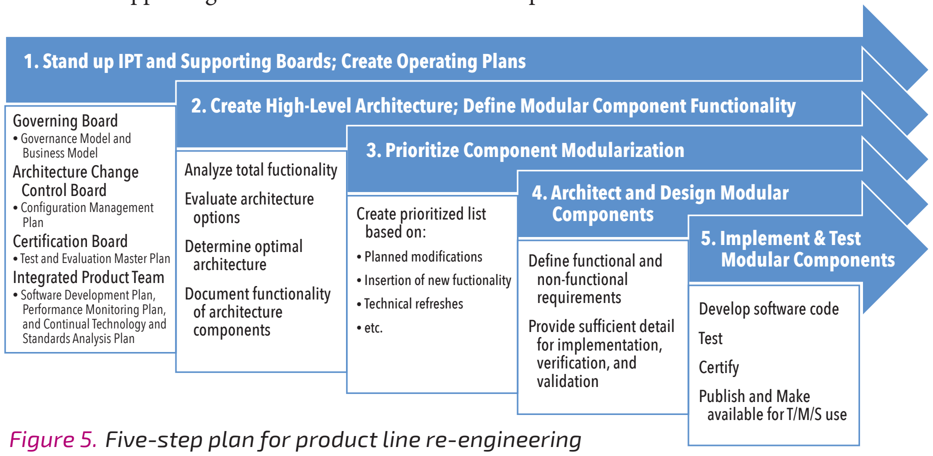

1. Stand up IPT and Supporting Boards; Create Operating Plans
Governing Board
• Governance Model and Business Model
Architecture Change Control Board
• Configuration Management Plan
Certification Board
• Test and Evaluation Master Plan
Integrated Product Team
• Software Development Plan, Performance Monitoring Plan, and Continual Technology and Standards Analysis Plan
2. Create High-Level Architecture; Define Modular Component Functionality
Analyze total fuctionality
Evaluate architecture options
Determine optimal architecture
Document functionality of architecture components
3. Prioritize Component Modularization
Create prioritized list based on:
• Planned modifications
• Insertion of new fuctionality
• Technical refreshes
• etc.
4. Architect and Design Modular Components
Define functional and non-functional requirements
Provide sufficient detail for implementation, verification, and validation
5. Implement & Test Modular Components
Develop software code
Test
Certify
Publish and Make available for T/M/S use

Figure 5. Five-step plan for product line re-engineering

Table 2: *MOSA Integrated Product Team and Supporting Boards*

Team/Board	Responsibilities	Operating Plans
MOSA Integrated Product Team	Responsible for executing the technical portions of the implementation plan.	Software Development Plan, Performance Monitoring Plan, and Continual Technology and Standards Analysis Plan
Governing Board	Responsible for establishing and executing the architecture's governance model and business model as well as providing direction to the IPT and other boards.	Governance Model and Business Model
Architecture Change Control Board	Responsible for controlling technical changes to the architecture.	Configuration Management Plan
Certifications Board	Responsible for performing independent testing on completed software in order to verify whether or not that software meets the standards set forth by the architecture.	Test and Evaluation Master Plan

Table 3: *MOSA Metrics*

MOSA Goal	Metric(s)
Be interoperable with and available for use by all aircraft platforms supported by the project	• Number of MOSA sponsors • Number of programs actively using MOSA-certified modular components • Number of MOSA-certified modular components in use per program
Enable the rapid incorporation and fielding of new capabilities	• Time from MOSA-certification of a modular component to fielding of that modular component by a program
Reduce costs and timeframes associated with development and test	• Development time per modular component • Development level of effort per modular component • Test time per modular component • Test level of effort per modular component • Test time for program integration test/system test • Test level of effort for program integration test/system test
Be financially sustainable	• Total MOSA-related costs • MOSA costs per sponsor • MOSA costs per modular component
Evolve with technology and standards	• Number of standards adopted
Have an assigned test team that will verify whether or not solutions comply with the architecture's standards	• Number of MOSA-certified modular components • Pass rate of MOSA certification testing
Track MOSA implementation progress	• Number of modular components identified • Number of modular components with detailed design documentation • Number of modular components in development • Number of modular components completed • Number of modular components certified

Step 1 — Set up IPT and Supporting Boards; Create Operating Plans

During Step 1, the project will set up an IPT, the governing board, architecture change control board, and certifications board. Table 2 summarizes the IPT and boards' responsibilities and products (operating plans) while the following paragraphs provide additional details.

During Step 1, the project will set up the MOSA IPT. This team is responsible for executing the technical portions of the implementation plan. They will operate under the direction of the governing board (described below) and will include the four product managers (one for each product variant), the four systems engineers, and the project's chief systems architect. During Step 1, this team will create a software development plan, a performance monitoring plan, and a continual technology and standards analysis plan. The software development plan will follow the DoD specifications (DoD 2017) and will detail how the development teams for the four variants will implement the requirements associated with each modular component. The performance monitoring plan will detail MOSA-related metric tracking and reporting in accordance with ISO/IEC/IEEE Standard 15288:2015 Clause 6.3.7 (ISO/IEC/IEEE 2015) to help ensure the project meets its MOSA-related goals. Table 3 below summarizes these metrics. The continual technology and standards analysis plan will detail how the IPT will stay current in their knowledge of emerging technologies and standards in accordance with DoD Architecture Framework (DoDAF) Systems Viewpoint (SV)-9 (DoD 2010) and how the IPT will evaluate whether or not the architecture should incorporate those technologies and standards in accordance with ISO/IEC/IEEE Standard 15288:2015 Clause 6.3.3 (ISO/IEC/IEEE 2015).

During Step 1, the project will also establish the governing board, architecture change control board, and certifications board. The governing board will be responsible for establishing and executing the architecture's governance model and business model as well as providing direction to the IPT and other boards. The governing board membership includes the project manager, two deputy project managers, and the project's chief systems architect. The architecture's governance model will detail the roles and responsibilities of sponsors, product managers, the MOSA IPT, the Governing Board, and supporting boards in accordance with ISO/IEC/IEEE Standard 15288:2015 Clause 6.3.1 (ISO/IEC/IEEE 2015). The model will also include the governing board reporting responsibilities; conflict resolution methods the governing board will use when sponsors have differing priorities, such as modular component development priorities; and procedures for adding and removing governing board and supporting board members. Additionally, the governing board will develop the architecture's business model. This model will define communal cost assessments and allocations to the sponsors in accordance with ISO/IEC/IEEE Standard 15288:2015 Clause 6.3.2 (ISO/IEC/IEEE 2015).

While it was impossible to accurately quantify the expected cost savings within the business model, the sponsors agreed to divide the costs to sustain a MOSA component among the sponsors who use it. For example, if two variants use modular component A, each sponsor would be responsible for funding half of the sustainment costs. If three variants use modular component B, each sponsor would be responsible for funding a third of the sustainment costs. With the variants having a 30 to 40-year life expectancy, the sponsors expect the total lifecycle cost savings to be significant. Additionally, if introducing a new variant, that sponsor would have a modular feature "menu" from which to choose. That sponsor would then be responsible for his/her fair share of sustainment costs while the sustainment costs of the existing sponsors using that modular component would be proportionally reduced.

The architecture change control board will operate under the direction of the governing board and will be responsible for controlling technical changes to the architecture. The board membership includes the project's configuration manager plus the four product variant product managers. The board will review the project's existing configuration management plan for applicability and recommend any necessary changes or additions while still adhering to the DoD specifications (DoD 2013). Once the governing board approves the modifications, the architecture change control board will ensure proper configuration management plan execution as it relates to the modular architecture.

The certifications board will operate under the direction of the governing board and will perform independent testing on completed software to verify whether or not the software meets the architecture standards. The certifications board includes the project's test lead, the two deputy project managers, and the project's chief systems architect. The certifications board will develop and execute the architecture's test and evaluation master plan which will detail the architecture's testing strategy and the resources (hardware, software, and personnel) necessary to execute independent testing in accordance with the DoD's *Test and Evaluation Management Guide* (DoD 2012). Once approved by the governing board, the certifications board will ensure proper test and evaluation master plan execution.

Step 2 — Create High-Level Architecture; Define Modular Component Functionality

During Step 2, the IPT will develop a high-level architecture in accordance with ISO/IEC/IEEE Standard 15228 Clause 6.4.4 (ISO/IEC/IEEE 2015) and document it in accordance with the DoDAF v2.02 (DoD 2010). To do this, the IPT must analyze the total functionality of all four variants, evaluate architecture options, determine optimal architecture, and document major component functionality. Lower-level architecture and design efforts for each modular component will occur in Step 4.

Step 3 — Prioritize Component Modularization

During Step 3, the IPT will develop a prioritization schema and then, using that schema, create a prioritized development backlog for modular components following agile values and principles (Fowler and Highsmith 2001). The IPT will prioritize components based on various factors, such as those listed below.

- Planned component upgrades
- New component functionality insertion
- Technical refresh of any major component
- Component feature commonality among product variants
- Number of product variants adopting the component
- Modularizing component complexity

Step 4 — Architect and Design Modular Components

During Step 4, the IPT will architect and design modular components in accordance with ISO/IEC/IEEE Standard 15228 Clauses 6.4.4 and 6.4.5 (ISO/IEC/IEEE 2015) and document them in accordance with DoDAF v2.02 (DoD 2010). The IPT must ensure the details in the architecture and design can sufficiently support future implementation, verification, and validation activities. The architecture and design work sequence and timing will be in accordance with the prioritization defined in Step 3. This step is expected to further define the stakeholder and component requirements in an iterative fashion in accordance with ISO/IEC/IEEE Standard 15288:2015 Clauses 6.4.2 and 6.4.3 (ISO/IEC/IEEE 2015). The configuration control board will incorporate all related DoDAF models into the architecture with each modular component architecture creation in accordance with the project's configuration management plan.

Step 5 — Implement and Test Modular Components

During Step 5, the product development teams will implement software to satisfy the modular component requirements defined in Step 4 according to the software development plan developed in Step 1. Once implemented, the software test will occur under the certifications board's cognizance according to the test and evaluation master plan developed in Step 1. When the certifications board certifies a modular component, the board will publish that component to a shared repository and notify all product managers that the component is available for their use.

CONCLUSIONS

The development organization of four operational product variants evaluated three options to regain the efficiencies associated with a product line approach: code merge, code re-use, and MOSA. They found code re-use was the quickest to implement while MOSA appeared to provide the most long-term benefits. As such, the sponsors directed the development organization to use code re-use as an intermediate steppingstone as they pursued the long-term product line re-engineering goal via a modular architecture.

The development organization created a detailed five-step plan to implement a modular architecture. Making this plan unique is the fact the development organization is not starting with a clean slate. In fact, it has four operational product variants, each with its own sponsors and end-users expecting continued product support plus new feature development concurrent with the technical refreshes necessary to meet evolving cybersecurity requirements. To balance this reality, the development organization incorporated agile values and principles into their five-step plan (Fowler and Highsmith 2001). This approach allows the designing and building of modular components in a prioritized order. Each modular component built provides value in feature sets as well as in reduced sustainment burden due to cost-sharing among the sponsors. Each modular component aligns to the high-level architecture vision, ensuring previously built and subsequent modular components will interact seamlessly based on well-defined interfaces adhering to industry standards. Since the components are modular, the variant sponsor may choose when to implement a component. This decouples the variants' schedules and permits sponsors to pursue different priorities at any given time.

This product line re-engineering effort is underway, so the final results are unknown. However, the project and sponsors are seeing the immediate results related to code re-use, the end-users continue to have all products meet current operational needs, and the groundwork is being established so future sustainment of the four product variants will be more efficient and effective by eliminating the code merge schedule dependencies and avoiding the manual code re-use testing inefficiencies. Overall, the sponsors and development organization are optimistic the approach chosen will deliver the modular product line benefits and be enacted with the concurrent commitments associated with supporting operational products. ■

REFERENCES

- Boehm, B. 1999. "Managing Software Productivity and Reuse." *Computer* 32 (9): 111-113.
- Dahmann, J. 2014. "System of Systems Pain Points." *INCOSE International Symposium* 24 (1): 108-121.
- Department of Defense. 2004. *Program Manager's Guide to Open Systems—A Modular Open Systems Approach to Acquisition*. Washington, US-DC.
- ———. 2010. *DoD Architecture Framework Version* 2.02. Washington, US-DC.
- ———. 2012. *DoD Test and Evaluation Management Guide*. Washington, US-DC.
- ———. 2013. *DI-SESS-81875, Data Item Description: Configuration Management (CM) Plan*. Washington, US-DC.
- ———. 2016. "Chapter 3—Systems Engineering." In *Defense Acquisition Guidebook*. Washington, US-DC.
- ———. 2017. DI-IPSC-81427B, *Data Item Description: Software Development Plan (SDP)*. Washington, US-DC.
- Fowler, M., and J. Highsmith. 2001. "The Agile Manifesto." *Software Development 9* (8): 28-35.
- ISO (International Organization for Standardization)/IEC (International Electrotechnical Commission)/IEEE (Institute of Electrical and Electronics Engineers). 2015. ISO/IEC/IEEE 15288:2015. Systems and Software Engineering-Content of Systems and Software Life Cycle Process Information Products (Documentation), International Organization for Standardization/International Electrotechnical Commission: Geneva, Switzerland.
- Magrab, E. B., S. K. Gupta, F. P. McCluskey, and P. Sandborn.. 2009. *Integrated Product and Process Design and Development: The Product Realization Process*. Boca Raton, US-FL: CRC Press.
- Maley, S., G. Lofberg, and M. Lasiter. 2008. "Overview of the Comprehensive Automated Maintenance Environment Optimized (CAMEO) System." *Annual Forum Proceedings-American Helicopter Society* 64 (3): 2069.
- Maley, S., M. Stonebraker, J. Schmidley, and M. Lasiter. 2009. "Open-Source Development of an Automated Maintenance Environment (AME) for Lower Cost, Collaborative Implementation of CBM." Paper presented at Sixth DSTO International Conference on Health & Usage Monitoring, Melbourne, AU, 09-12 March.
- Osborne, L., J. Brummond, R. D. Hart, M. Zarean, and S. M. Conger . 2005. *Clarus Concept of Operations. Publication No*. FHWA-JPO-05-072, Federal Highway Administration (FHWA). Washington, US-DC.
- Pohl, K., G. Böckle, and F. J. van Der Linden. 2005. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Berlin, DE: Springer Science & Business Media.
- Schmidley, J. 2011. "Enabling Maintenance in a Net-centric Environment." Paper presented at Department of Defense Maintenance Symposium & Exhibition, Fort Worth, US-TX, 14-17 November.
- Schwanke, R. W. 1991. "An Intelligent Tool for Re-Engineering Software Modularity." In *Proceedings of the 13th International Conference on Software Engineering*: 83-92. IEEE Computer Society Press.
- Tolentino, G., and J. Wood. 2018. "Balancing Systems Engineering Rigor with Agile Software Development Flexibility." *INSIGHT* 21 (2): 25-28.
- Walden, D. D., G. J. Roedler, K. J. Forsberg, R. D. Hamelin, and T. M. Shortell. eds. 2015. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. Hoboken, US-NJ: John Wiley & Sons.

ABOUT THE AUTHORS

John Wood, PhD has spent his career pursuing perfection in areas where less-than-perfect performance can be deadly. During more than two decades in military service, civilian sector innovation, and academia, he has applied his systems engineering expertise to advance high-profile programs in healthcare, aviation, and defense. John earned a Bachelor of Science in electrical engineering from the US Naval Academy and a PhD in systems engineering from the George Washington University.

Glenn Tolentino, PhD has been an engineering practitioner, researcher, and innovator in the Command and Control (C2) and Enterprise Engineering competency for over 25 years. He has been a major contributor in designing and deploying C2 and Intelligence Systems. Glenn earned a B.S. in Applied Mathematics from San Diego State University. He also holds an MS in Software Engineering and a PhD in Computer Science from Southern Methodist University.

APRIL 2021
VOLUME 24/ ISSUE 1

Funding the PLE Factory in a Multi-Customer Contract-Based PLE Organization

Paul Clements, pclements@biglever.com

ABSTRACT
Feature-based Product Line Engineering is a well-defined, repeatable, automation-centric PLE method that is delivering even improvements in time, cost, and quality. An organization intent on adopting it so they can reap the benefits for their product line or product lines needs a viewpoint focusing on the people involved and what they do to keep the PLE factory operational on a day-to-day basis. This article describes an organizational structure for Feature-Based PLE based on the factory concept. It introduces the few roles that have no analog in other development disciplines; they are new to Feature-Based PLE. It also describes how traditional systems engineering roles carry out traditional systems engineering tasks, but with slight PLE-inspired extensions. Finally, we will explain why these changes are necessary.

INTRODUCTION

Feature-based product line engineering (INCOSE 2019) employs the PLE factory concept in which all development occurs for any product line product. Automatically configuring shared assets based on the feature choices for a product produces individual products. A product line organization's personnel need to carry out numerous tasks associated with the product creation, development, delivery, and evolution in its product line. Any organization employing this paradigm in a contract-based (as opposed to a mass market) context must answer the question: Who pays for the work inside the factory that may benefit multiple contracts? The answer can be surprisingly complex, involving security, regulatory compliance, and intellectual property protection issues of both the PLE organization and its customers. This article offers a method for answering this question. Answering

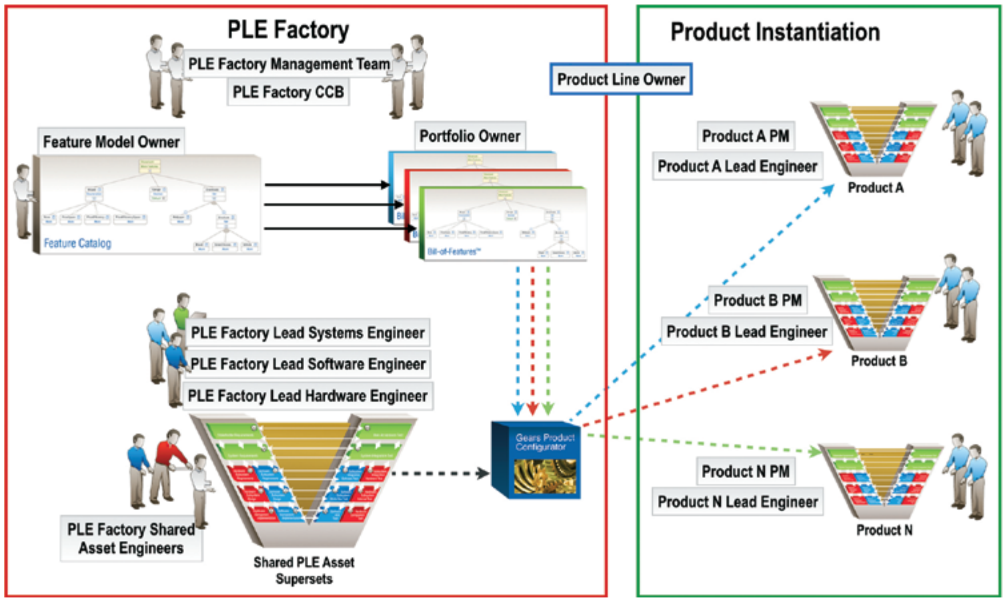


Figure 1: A PLE organizational structure

APRIL 2021 VOLUME 24/ ISSUE 1

this question means establishing processes to create charge numbers to which everyone working in the PLE factory can charge their effort. These processes must connect the funding supply to the funding consumption in a way that is fair, equitable, and compliant with applicable rules and regulations.

In this article, we assume the PLE organization has adopted a structure similar to Figure 1. The PLE factory is on the left; the systems engineering "V" model at the bottom represents the shared assets. The configurator receives a feature-based product descriptions (Bill-of-Features) and produces "V" subsets corresponding to the product. Product teams, who receive outputs from the PLE factory and deliver products and interface with customers are on the right. Ideally, all development happens inside the PLE factory.

We also assume the products will deliver to specific customers under a contract, as opposed to mass-market product line products with anonymous customers. In a PLE Factory, all development and engineering work occurs once inside the factory and applies through automated configuration to each product to which the work applies.

SIX STEPS TO CREATE A PLE FACTORY FUNDING MODEL

We describe the steps in our method for building a funding model below.

Step 1: Identify the Funding Model Goals and Requirements

This step lists goals can and should be the basis for reviewing the resulting funding model. Typical goals include the ones described below.

- **Goal: Compliance with standards and regulations.** A multi-customer, multi-contract environment acquires the systems under one or more contracts, possibly with contract clauses specifying what the customer will, and will not, pay for. In the case where the customer is the United States (US) Government, if the contract does not specify this then federal regulations do. The US embodies these in a regulation body as the Federal Acquisition Regulations (FAR). Cost accounting standards also apply. Individuals familiar with the applicable standards and statutes should review the funding model to ensure this compliance.
- **Goal: Fair and equitable customer treatment.** The FAR considers the US Government, including different US Government agencies, may purchase more than one system from the same supplier. It contains stipulations to ensure each procuring agency receives "fair and equitable" treatment from the supplier. Fair and equitable customer treatment is also a key PLE funding model goal and would be even if not enshrined in the law.
- **Goal: Insulate PLE Factory staff from obligation to a specific customer.** A third goal is to insulate PLE factory staff from funding flowing from a specific customer or customers. One way PLE can fail is to let the PLE factory turn into a dedicated job shop for one or two customers who happen to have the largest budgets. It is human nature to tailor solutions to those paying the bills. But it is vital every solution aims towards the overall product line health and robustness. Rather than have an engineer charge an activity to a product's account, it would be better if there were a charge number associated simply with the work type. Ideally, the contract funds for all contracts in the product line pool together to fund product line development. Of course, the PLE organization itself must track the individual contract contributions so they can bill each customer appropriately and calculate profit from each contract.
- **Goal: Capture truth.** Activities should charge to accounts set up to pay for *those* activities. This is true to aid internal cost understanding of various activities, but also true because (again) of contracting realities. Each contract will have a specific effort scope, meaning the activities charged to that contract need accurate recording. Proper accounting for the effort to service each contract is essential to support an audit. In addition, contracting companies bid on future contracts using past performance on similar contracts as a guide, or "basis of estimates." If the activity levels, as reflected by the amount charged to various charge numbers, are not accurate then the basis of estimates is valueless.
- **Goal: Be neutral concerning contract types.** Rules must make it clear which contract pays for which activities and in what proportion to the activity cost. This allays fears of allocating charges to cost-plus contracts instead of fixed-price contracts, in effect, shifting work from one contract to another to maximize company profits.
- **Goal: Protect the PLE organization's own intellectual property (IP).** Every business needs to protect IP improving its competitive position. Under contracts with the US Government, the government typically manages the associated IP through data rights assignment. The government uses the FAR to prescribe policies and procedures for acquiring data rights, which give the government nonexclusive license rights. Certain data rights types can compromise a company's competitive position because the government can simply hand over any material with these types to another bidder or producer.
  - For this reason, many organizations choose to treat the critical product line IP as protected corporate IP and specially call out activities associated with creating that IP. They will construct their funding model in a way to protect IP ownership.
  - In a PLE setting, the feature catalog is a prime example. If the feature catalog goes to a single customer, then that customer could conceivably produce not only its own product but any product for any customer. To best protect against this scenario, companies can treat the feature catalog as a trade secret and not deliver it to any customer. To prevent delivering the feature catalog, activities related to it must use internal funding sources exclusively—never contract funds.

Step 2: Enumerate the Activities to Fund

The ultimate funding model purpose is to ensure activities charge to the right funding source. Thus, it is necessary to enumerate the activates that need funding resolution. Figure 1 gives a good start at such a list. From it, we can identify activities related to:

- PLE factory management
- Change control
- Feature catalog creation and evolution
- Portfolio (Bills-of-Features) creation and evolution
- Lead engineers for the share PLE asset supersets
- Shared asset engineering

Not shown in Figure 1 are the activities associated with starting up, operating, and optimizing the IT environment—the tools and technologies—the PLE factory uses, as well as training and capturing processes and best practices.

A typical funding model will be much more fine-grained than what we discussed here when it comes to mapping activities to funding sources. The company should review the list to ensure it has the granularity necessary to provide the basis of estimate data for the next contract if that is a funding model goal.

APRIL 2021 VOLUME 24/ ISSUE 1

Step 3: Identify and Implement Protected Intellectual Property Policy

Call out any tasks identified in Step 2 involving company IP creation or evolution that need protection.

Step 4: Identify Available Funding Sources

Funding can be usefully divided between internal and external sources. External sources typically originate from customers or customer organizations via contracts. Internal funding includes sources originating within the product line organization, such as research and development or overhead.

The internal/external distinction is important because regulations often stipulate what the company may or may not use customer funds for, as well as the data rights, a company forfeits when it uses customer-derived funding for a particular activity. Internal funding usually comes with more discretionary privilege about how to spend it.

Step 5: Map from Activities to Funding Sources

This step associates a charging method with each activity. Under a contract, contract documents describe the work scope for each contract. It is important to charge the appropriate contract according to the scope defined in the contract. Some overriding principles include:

- If multiple contracts define the same development scope, they should share the cost. Charging to one contract when multiple contracts have the same development scope may not comply with regulations.
- Level-of-effort tasks involving managing work across the entire product line portfolio can share costs proportionally.
- It is acceptable to charge a single contract for a capability reused in the future by other contracts assuming the future contracts either are not yet awarded or the proposal and contract both document the reuse assumption.

The goal is charging the contracts benefiting from an activity proportionally according to the benefit they derive from the activity. A company should base any distribution algorithm calculating the proportion applicable to a specific contract on quantifiable data relevant to the work performed.

Some activities (PLE factory management) will benefit every product. Other activities (such as updating a shared asset or adding a new feature only some products will use) will only benefit some products. In some cases, an activity will benefit a single product (such as reviewing or delivering that product from the PLE factory). In all cases, proportionally charging the benefiting contract(s) still applies.

How do we calculate the proportion? We need to create a distribution algorithm defining the proportional benefit of each contract, such as basing it on the contract size or the product complexity.

Lockheed Martin describes a cost-sharing approach they use for the AEGIS weapon system product line, for which they are the prime contractor (Gregg et al. 2014):

The government, representing all AEGIS "consumers," has also instituted a cost-sharing approach to equitably allocate the cost of fixing a defect… If a program introduces an upgrade or new capability, it pays for it. Other programs are free to pick it up, or not, as they wish, but they pay for any required testing unique to their context. After a development is complete and time has elapsed, newly found defects become difficult to associate with any one program. In these cases, all programs pitch in to correct the defect. Lockheed Martin has a special funding account to fix all defects, across the entire product line, not related to unique capability content in development. Any program receiving special development funding pays for defects in that development, up to its demonstration milestone, at which point the cost-sharing approach starts.

This step results in a three-column table:

- Column 1 contains all tasks identified in Step 2.
- Column 2 names the funding source for the task, taking care to choose a source appropriately to protect IP.
- Column 3 identifies any information known upfront affecting the proportional cost allocation described above. Examples:
  - A planning activity for a specific contract might have the annotation "Charge to the (single) relevant contract." If this applies, replace the generic row with a set of rows; one for each contract.
  - PLE factory management activities might have the annotation "Charge to all contracts equally."
  - An activity to change a source code shared asset might have the annotation "Charge to all affected contracts proportionally based on each affected product's SLOC count."

Step 6: Establish Charge Numbers

This step begins with the activity-to-funding-source table produced in the previous step. To this table, add a column for charge numbers. Fill it in as follows:

- For those activities that will always charge to a specific source in a specific, known proportion, add a charge number for the activity funded by that source.
  - For any activity touching protected contractor IP, provide a charge number funded by internal funding.
  - For any activity benefiting (by definition) a single contract, provide a charge number funded by that contract. You will need one charge number for each contract.
  - For any activity benefiting (by definition) all contracts equally, provide a charge number funded by all contracts, either equally or in proportion to some metric, such as contract size.

- All other activities will receive funding on a case-by-case basis by some but not all contracts, or by all contracts but in a proportion specific to the exact task. For example, "Update a shared asset" is an activity. We cannot charge for it until a specific task (Update *this* shared asset in *this* way for *these* specific products) is at hand. For this category, choose from these strategies:
  - Produce a charge number set up front covering all possible contract combinations. This can quickly become extremely unwieldy. For an organization managing 20 contracts, this scheme would produce 2^{20} or about a million charge numbers. This scheme might be useful, however, to handle a small number of specific contract-sharing scenarios the organization expects to occur frequently.
  - Enter "TBD" in the table and create a charge number when needed. In this way, an activity such as "Update a shared asset" is a placeholder for specific tasks. When the PLE factory's change authority approves a change, it will know which products that change will affect. Then, either the change authority or an intermediary accounting function can create a charge number for the task and attach it to the change order the PLE factory staff will implement. The correct contracts can proportionally fund that charge number.
  - Enter "Spread-charge" in the table. In some organizations, creating a new charge number is a burdensome task, and so should only occur at project launch. This case may use a spread-charging approach. Spread-charging is allocating hours across multiple charge numbers according to some algorithm. This has the

Table 1. *A skeleton PLE factory funding model*

Activity	Funding source	What do we know upfront?	Charge #
PLE Factory management and CCB	Contract funds	We will charge each contract equally	CF-xxx-yyyyy (staff charges one charge number, funded by all contracts
Feature catalog work	Internal funding source, to protect IP	We will never use customer funds for this.	INT-xxx-yyyyy
Meeting to plan work for Contract #4	Contract #4's funds	We will charge solely to Contract #4	CF-aaa-bbbbb (funded solely by Contract #4)
Any activity updating a shared asset	Contract funds	Any activity here receives funding proportionally by contracts that benefit, where "proportionally" means [fill in allocation algorithm]	TBD. (When CCB approves a change, they list the affected programs. Assign a task-specific charge number then, funded proportionally by those contracts.)

same effect as establishing new charge numbers based on a fixed percentage allocation using multiple charge numbers. However, spread-charging skips creating new charge numbers but has the risk of relying on executing the algorithm reliably by each employee. To avoid this risk, automation should handle the allocation to multiple contracts and charge numbers automatically.

SUMMARY

Table 1 shows a very cursory example of the table resulting from the method outlined above. A real table would have many more activities listed, and specific information about each one, but Table 1 illustrates the general ideas outlined above. The various rows reflect the different needs discussed: Charging all contracts equally, charging to internal funding to protect IP, charging to a single contract, and charging proportionally to benefitting contracts.

Every product line organization must answer the question: "Who pays for the product line activities?" Organizations building products under contract for specific customers need a funding model that provides a mapping between PLE activities and, essentially, charge numbers product line engineers use to pay for their work. The funding model needs to satisfy the organization's overall goals, such as protecting intellectual property and ensure compliance with applicable statutes and contractual requirements. The result should be a charging model complying with contractual requirements and applicable regulations, protects the organization's IP, keeps PLE factory staff from working directly for individual customers, captures truth, and provides a fair and equitable way to charge customers for work. We based the method presented in this paper on our work with PLE practitioners who operate in a government contract environment. ■

ACKNOWLEDGMENTS

We adapted this paper from an internal BigLever technical report, to which several people contributed. Our thanks go to those people for their helpful insights and contributions.

REFERENCES

- Gregg, S., R. Scharadin, E. LeGore, and P. Clements. 2014. "Lessons from AEGIS: Organizational and Governance Aspects of a Major Product Line in a Multi-Program Environment." Paper presented at the 18th Software Product Line Conference, Florence, IT, 15-19 September.
- International Council on Systems Engineering (INCOSE). 2019. *Feature-based Systems and Software Product Line Engineering: A Primer.* San Diego, US-CA: INCOSE.

ABOUT THE AUTHOR

Dr. Paul Clements is the vice president of Customer Success at BigLever Software, Inc., where he works to help organizations adopt feature-based systems and software product line engineering. Prior to this, he was a senior member of the technical staff at Carnegie Mellon University's Software Engineering Institute, where for 17 years he worked leading or co-leading projects in software product line engineering and software architecture documentation and analysis. Prior to the SEI, Paul was a computer scientist with the US Naval Research Laboratory in Washington, DC. Paul has both a BS and MS in computer science from the University of North Carolina at Chapel Hill and a PhD in computer science from the University of Texas at Austin.

Development of a Hybrid Product Breakdown Structure and Variability Model

Evan R. Helmeid, evan.helmeid@safrangroup.com

ABSTRACT
In transforming from a project-based engineering approach to a product line engineering (PLE) approach, the engineering teams must have support throughout the transition from evolving tools and methodologies. As an example, the Product Breakdown Structure (PBS) is traditionally a construction-based decomposition of a complex system, where subsystems reflect a breakdown of the engineering elements with appropriate technical interfaces, subassemblies, and team responsibility delineations. However, when used for a product line with myriad variants, a traditional PBS format provides insufficient detail and structure. As such, using standard desktop software, the author developed a hybrid PBS-Variability Model (VM), combining the familiar PBS structure with variability modeling aspects based on feature modeling and decision modeling approaches. This resulted in an engineering artifact recognizable as a PBS and easy to adapt to design evolution, yet sufficiently expansive to initially characterize variability. In this way, the traditional PBS evolves to the hybrid PBS-VM before transitioning to a complete variability model, thereby supporting the engineering teams transitioning from a project-based engineering approach to a PLE approach. In this paper, the author describes the traditional PBS limitations, the hybrid model development process with a custom-developed syntax description, the resulting hybrid model, and conclusions on appropriate product line usage.

INTRODUCTION

Within high-tech industry, transforming an engineering team from project-based[1] development into product line engineering (PLE) involves transitioning concepts, working methods, project management tools, and engineering tools and methods. A large, geographically distributed organization must make this transition gradually and deliberately, allowing the team to continue supporting current projects while preparing for and implementing future product lines. Along with this evolution in the "way of thinking," the tools and methodology changes must also support an evolution, Figure 1 (INCOSE 2019).

1. Project-based teams working in a "silo" environment may have little visibility into other engineering artifact use and little incentive, interest, or opportunity to promote reusing designs, whether opportunistic or intentional.

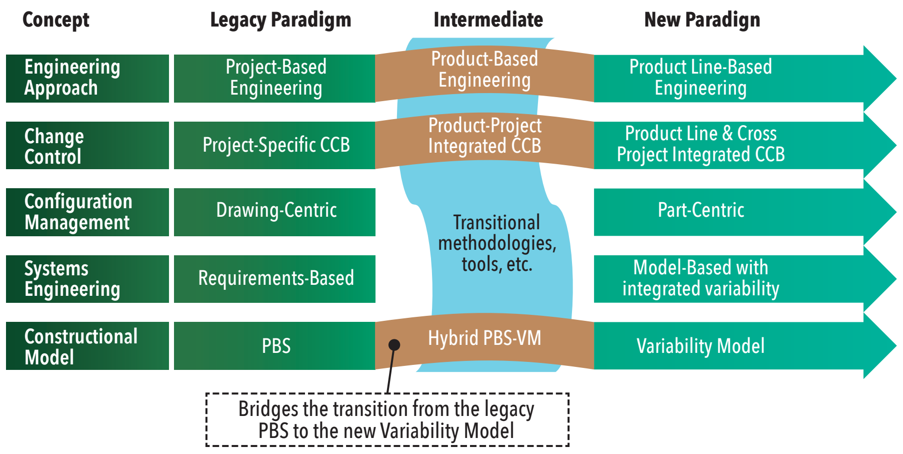
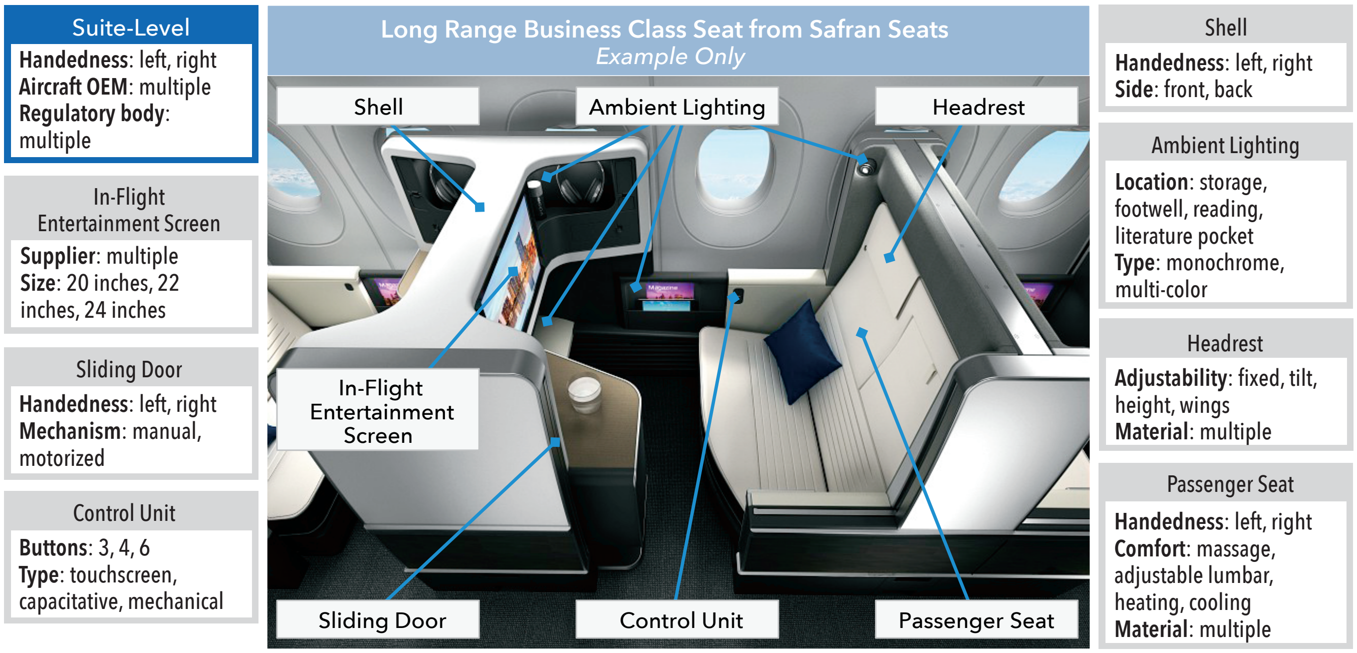

Concept
Legacy Paradigm
Intermediate
New Paradigm
Engineering Approach
Project-Based Engineering
Product-Based Engineering
Product Line-Based Engineering
Change Control
Project-Specific CCB
Product-Project Integrated CCB
Product Line & Cross Project Integrated CCB
Configuration Management
Drawing-Centric
Part-Centric
Systems Engineering
Requirements-Based
Model-Based with integrated variability
Constructional Model
PBS
Hybrid PBS-VM
Variability Model
Transitional methodologies, tools, etc.
Bridges the transition from the legacy PBS to the new Variability Model

Figure 1: All concepts of an organization must transition from a legacy project-based engineering approach to a new product line-based approach; the hybrid PBS-VM bridges this transition for the constructional model representation. CCB is configuration control board



Suite-Level
Handedness: left, right
Aircraft OEM: multiple
Regulatory body: multiple

In-Flight Entertainment Screen
Supplier: multiple
Size: 20 inches, 22 inches, 24 inches

Sliding Door
Handedness: left, right
Mechanism: manual, motorized

Control Unit
Buttons: 3, 4, 6
Type: touchscreen, capacitative, mechanical

Shell
Handedness: left, right
Side: front, back

Ambient Lighting
Location: storage, footwell, reading, literature pocket
Type: monochrome, multi-color

Headrest
Adjustability: fixed, tilt, height, wings
Material: multiple

Passenger Seat
Handedness: left, right
Comfort: massage, adjustable lumbar, heating, cooling
Material: multiple

Figure 2: A business class aircraft seat contains many variation points—including passenger-facing and non-passenger-facing features—with multiple drivers, inductors, and constraints (passenger, airline, regulatory body, manufacturer, supplier).

In designing and manufacturing aircraft seats, the engineering teams are distributed across multiple countries, research and technology groups, industrialization teams, and manufacturing facilities. A new product line development includes the technical architecture definition. A Product Breakdown Structure (PBS) traditionally describes this architecture for the constructional view. However, for a product line approach, the PBS shows insufficient detail—namely, variants. As such, models can add variation points to identify the product variants as perceived by the customer and the impacts of those variations on the technical solutions. Designing and managing an effective product line architecture requires understanding planned variants to promote a viable lifecycle, as well as the ability to convey this information to the sales team and other stakeholders.

Accomplishing this requires accurate variation characterization and conveyance to the engineering, management, and sales teams, as well as to the customer (via a "product catalog").

In evolving the tools to support the product line approach, the author developed a hybrid PBS-variability model (VM) using standard desktop software to bridge the gap between the traditional PBS and the capabilities of a complete VM.

ARRIVING AT A NEED FOR A HYBRID PBS-VM

Context: Product Line Complexity

When considering aircraft seats, the PLE problem is not as simple as considering the different features a passenger or an airline may request. Regulatory and airframe requirements highly constrain the design space, the airlines have wide-ranging needs to meet the specific flight route needs as well as the airline values and branding, and the passengers have broad expectation and demand ranges for high-quality products. Every aircraft type, lavatory and galley configuration, in-flight entertainment (IFE) solution, and regulatory revision results in a new product variant—whether major or minor. Indeed, the features a customer interacts with are only the beginning of the challenge (Figure 2).

Initial Problem Statement

The need for a hybrid PBS-VM solution appeared after receiving direction to develop a PBS for a new business class aircraft seat product line. Specifically, the intent was to create a PBS providing insight into the reusability and commonality level between customer program product line instantiations.

Initially perceived as a straightforward task, a conventional PBS quickly showed inherent limitations and lack of adaptability to provide the requested insight into product line architecture efficiency or effectiveness.

The initial PBS request came from a traditional systems engineering approach context, which is insufficient for a product line approach. However, even adding the concept of a "150% PBS," where "the maximal product breakdown structure is generated by identifying the components necessary to implement all of the functions" (Krob and Le Sauce 2015), is still insufficient—a 150% PBS is useful as a way to perceive the PLE breadth, but it remains based on the basic PBS principles and, therefore, has inherent limitations when trying to provide insight into a complete product line (CESAM 2017, and Krob and Le Sauce 2015), as discussed in the following sections.

Discovering Traditional PBS Limitations

The first attempt began with a traditional PBS expanded to achieve a 150% PBS model, representing the product line scope, but easily recognizable as a PBS from the traditional systems engineering tools and methods. If using a small number of levels, this is a conceivable solution. Major subsystems have variations—such as ambient lighting options, IFE providers and peripherals, and left- and right-handed suites (Figure 2).

Figure 3 shows a simple, single-solution PBS example for an IFE system. Figure 4 shows an expanded 150% PBS for the same IFE system; note the 150% PBS is similar to a feature model with variation points (Czarnecki et al. 2012).

Within a business class cabin context, the IFE system is only a small part. But even this small part is highly customizable with implications to other parts of the suite (mechanical integration and power supply) and with dependencies on other systems (ability to support a suite-level active acoustic control system).

In pursuing reusing technical solutions across the product line and in understanding major variations can occur at different PBS levels with or without affectations at

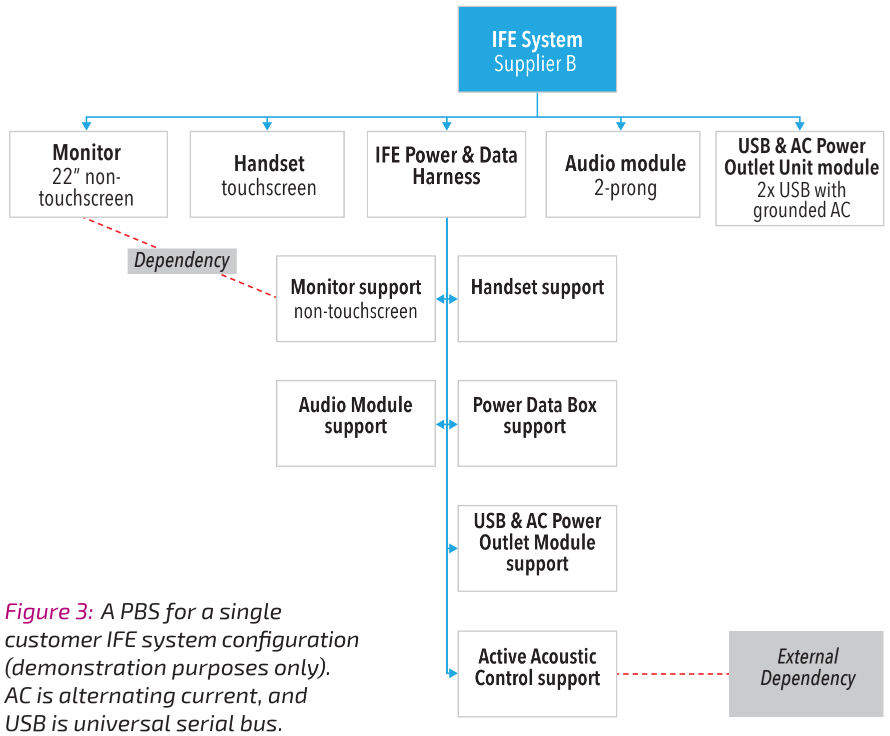


Figure 3: A PBS for a single customer IFE system configuration (demonstration purposes only). AC is alternating current, and USB is universal serial bus.

sub-tier levels, the low-fidelity of the 150% PBS is therefore insufficient.

For example, one chooses seat handedness at a high PBS level, but elements lower in the PBS are independent of handedness and can actually be common. Looking closely at the IFE system shown in Figure 3 and Figure 4, we can already see even this simple example cannot capture the supplier options or dependencies on other PBS elements (the power and data harness being able to support an active acoustic control system). As such, indicating two completely new variants at such a high level in the assembly (duplicating the IFE system to reflect multiple suppliers) does not reflect the actual reusable amount of engineering (such as common mechanical interfaces) nor the common elements (such as screen and power outlet module). Additionally, designs can be reusable—such as for seat covers or cushions—but customers can select different materials.

In essence, different components or subsystems can be reusable depending on different inductor sets—aircraft type, airline branding, seat position in the cabin, and seat handedness. These decisions may not be mutually exclusive, resulting in complex relationships between allowed variations and, therefore, complex relationships to understand the product line's efficiency with reusability and commonality.

Table 1 (next page) provides more detailed variability descriptions of each major PBS element in the IFE system example.

As is evident, both PBS versions, shown in Figure 3 and Figure 4, lose this information. As such, understanding our product line requires a more advanced model.

Creation of the hybrid PBS-VM

After reviewing further PLE documentation and recommended practices and tools, the feature modeling and decision modeling concepts provided a potential solution, combined with standard desktop tools (Microsoft® Excel® and Visio® with import capabilities, as described in the "Tools" section) (Czarnecki et al. 2012). However, incorporating these ideas into the constructional model required developing a hybrid solution merging the 150% PBS with the tree notation of feature modeling and leveraging the syntax complexity and flexibility of decision modeling.

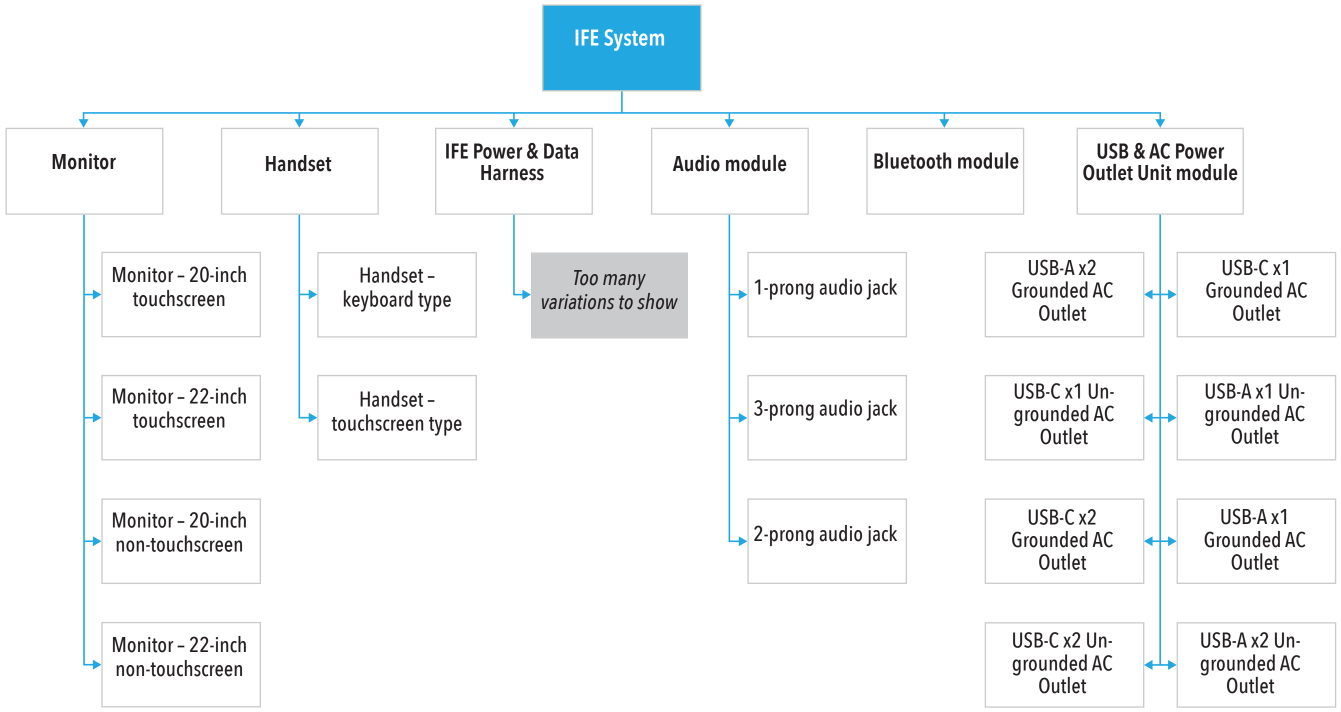

IFE System
Monitor
Handset
IFE Power & Data Harness
Audio module
Bluetooth module
USB & AC Power Outlet Unit module
Monitor – 20-inch touchscreen
Monitor – 22-inch touchscreen
Monitor – 20-inch non-touchscreen
Monitor – 22-inch non-touchscreen
Handset – keyboard type
Handset – touchscreen type
Too many variations to show
1-prong audio jack
3-prong audio jack
2-prong audio jack
USB-A x2 Grounded AC Outlet
USB-C x1 Grounded AC Outlet
USB-C x1 Un-grounded AC Outlet
USB-A x1 Un-grounded AC Outlet
USB-C x2 Grounded AC Outlet
USB-A x1 Grounded AC Outlet
USB-C x2 Un-grounded AC Outlet
USB-A x2 Un-grounded AC Outlet

Figure 4: A 150% PBS with identified variability for an IFE system (demonstration purposes only).

APRIL 2021
VOLUME 24/ ISSUE 1

Table 1: Description of the variability for the IFE System

PBS Element	Description of Variability	Dependencies
Monitor	Dependent upon airline selection	Affects monitor shroud, mounting bracket, power consumption
Handset	Dependent upon airline selection	Affects mounting bracket
IFE Power & Data Harness	The harness must provide support (e.g., pigtail and connector) to all connected peripherals and systems. Some peripherals are always present (e.g., monitor, audio module), but some are optional and may be omitted (e.g., Bluetooth module).	Affected by selection of all IFE peripherals, selection of suite-level Active Acoustic Control system
Audio module	Dependent upon airline selection	May affect mounting bracket
Bluetooth module	Dependent upon the selected IFE Supplier (only available from some suppliers), and dependent upon airline selection	Affects Power & Data Harness, mounting bracket
USB & AC Power Outlet Unity module	Numerous options available, dependent upon airline selection. The actual choice is nested—first choose the AC outlet, then choose the number USB ports, then choose the type of USB ports. Even with this variation, the mechanical form factor and integration may be unchanged.	Affects Power & Data Harness; may affect mounting bracket

The resulting solution provides the complexity needed to characterize the total product line and representative variability with the familiarity of viewing a traditional PBS. Refer to Figure 5 for the hybrid PBS-VM of the previous IFE system example.

After peer-reviewing with the engineering teams, it was apparent this hybrid PBS-VM approach met its objectives and allowed the multi-disciplinary team to review and agree upon the new product line architecture and initial variability

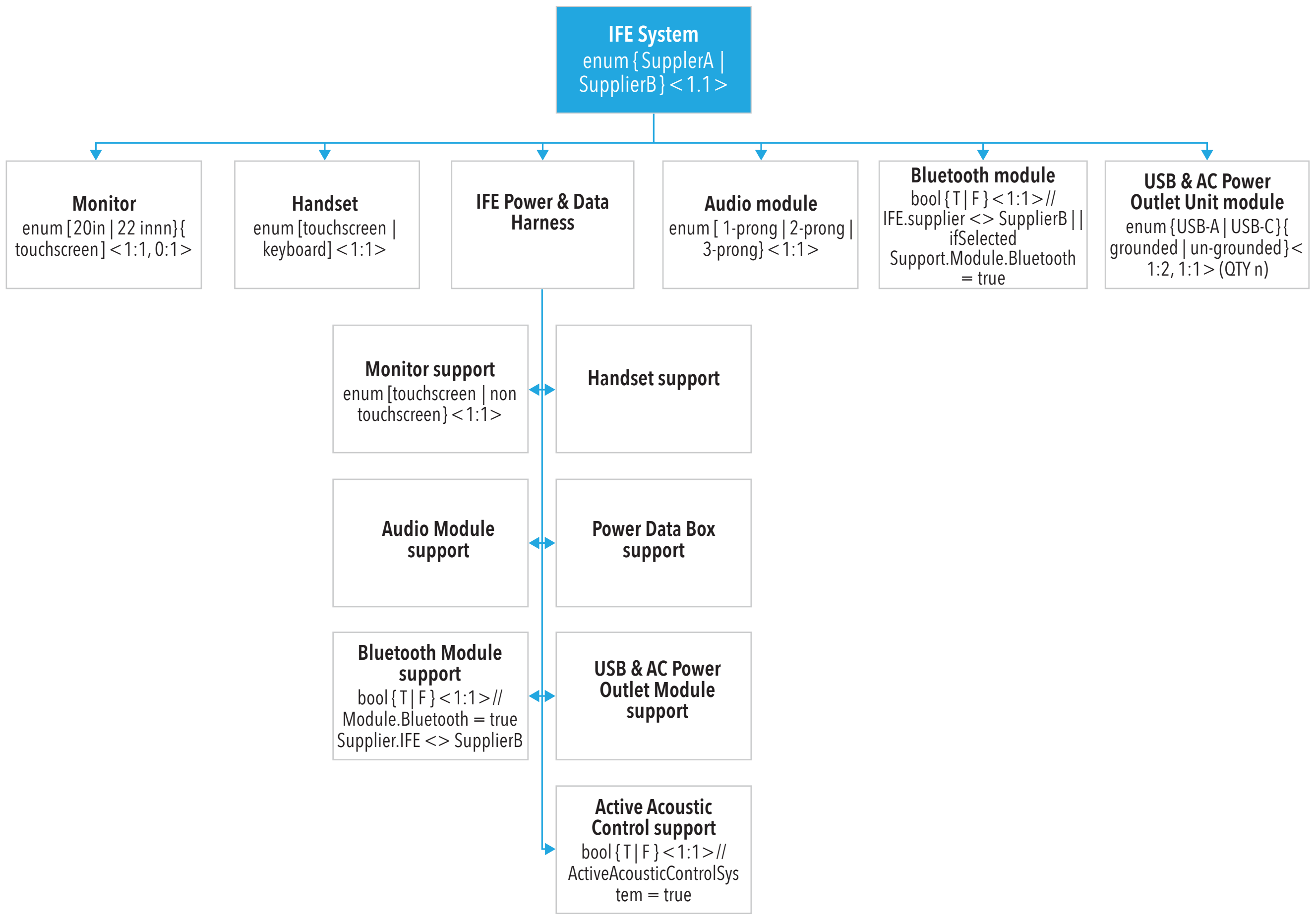

IFE System
enum { SupplerA | SupplierB } < 1.1 >
Monitor
enum [20in | 22 innn}{ touchscreen] < 1:1, 0:1 >
Handset
enum [touchscreen | keyboard] < 1:1 >
IFE Power & Data Harness
Audio module
enum [1-prong | 2-prong | 3-prong} < 1:1 >
Bluetooth module
bool { T | F } < 1:1 > // IFE.supplier <> SupplierB | | ifSelected Support.Module.Bluetooth = true
USB & AC Power Outlet Unit module
enum {USB-A | USB-C}{ grounded | un-grounded}< 1:2, 1:1 > (QTY n)
Monitor support
enum [touchscreen | non touchscreen } < 1:1 >
Handset support
Audio Module support
Power Data Box support
Bluetooth Module support
bool { T | F } < 1:1 > // Module.Bluetooth = true Supplier.IFE <> SupplierB
USB & AC Power Outlet Module support
Active Acoustic Control support
bool { T | F } < 1:1 > // ActiveAcousticControlSystem = true

Figure 5: Because of the simplified structure combined with embedded syntactical descriptions, this hybrid PBS-VM retains the visual simplicity and recognizability as a basic PBS (Figure 3) while showing the variability of a 150% PBS (Figure 4) and capturing complex relationships and dependencies (Table 1).

approach without having to understand the complexity of a dedicated variability model. This approach bridged the gap between the traditional engineering methodologies with their inherent restrictions and the new PLE paradigm and the associated complexity.

It is important to note, however, this method still has limitations in the amount of data shown and, therefore, in the level of detailed engineering represented. As a rule, maintain this relative simplicity, while providing just enough detail to address the question within the current design detail level context. Of course, additional engineering artifacts need developing to fully define the product line, and design and development can recursively use these methods throughout.

VM for Full PLE Implementation

When ready for the full PLE tool and terminology set, a proper VM should address the remaining concerns of the hybrid PBS-VM and its ability to support viewing the product line architecture effectiveness, including reusability and commonality. Additionally, a dedicated variability modeling tool can interface with configuration management (CM) and product lifecycle management (PLM) tools to properly control the product line evolution throughout the lifecycle. As such, the hybrid PBS-VM usefulness may reduce as the product line systems engineering capabilities mature within the organization or as a program progresses through the design process.

GUIDE FOR DEVELOPING AND USING A HYBRID PBS-VM

The necessity to describe a product line's complexity in the traditional systems engineering practice and tool context created the hybrid PBS-VM, thereby easing the transformation to a product line-based organization. Furthermore, with the ability to describe the product line and its variability comes the ability to manage the variability and the architecture.

The new model merges four key concepts:

1. PBS: constructional system decomposition, presented in a hierarchical form
2. 150% PBS: a PBS adapted to address the product line breadth by adding the product line variability to over-define a product solution
3. Decision model: a decision set adequately distinguishing among application engineering product family members and to guide adaptation of application engineering work products (Software Productivity Consortium Services Corporation 1993)
4. Feature model: captures features —"distinguishing characteristic[s] that [describe] how the members of the product line differ from each other" (INCOSE 2019)—and the relationships among them (Kang et al. 1990)

The following sections discuss the steps to create an effective hybrid PBS-VM and define the syntax developed to describe variability.

Recipe

The following recipe can create an effective hybrid PBS-VM for a technical system. So far, only four aircraft seat product lines have deployed this method—two premium business class seats, one basic business class seat, and one economy class seat. However, the recipe can adapt to other industries and systems and is usable for smaller systems as well as extensible to larger and more complex systems.

Step 1: Create a basic PBS (refer to Figure 3). Use a top-down method to develop a traditional PBS, focusing on capturing the overall physical system architecture in the constructional feature context. It may be useful to consider an anticipated "standard" product version to capture the generic system architecture and to begin defining the scope. Also, recall a PBS should not intend to match either a work breakdown structure or a drawing tree—these are supplemental system visions that work in complement to fully define a system and facilitate the management, engineering, procurement, and manufacturing activities (INCOSE 2019 and United States Department of Defense 2018).

Step 2: Develop into a 150% PBS (refer to the top two levels of Figure 4). Add detail to the basic PBS to capture the full option range offered on the product line. The focus is on identifying the scope and overarching architecture and decomposition—defining relationships, rules, and logic between options or features happens later. This PBS version is not instantiable into a single customer solution, as it over-defines the system (Krob and Le Sauce 2015).

Step 3: Identify variation points (refer to Figure 4). Identify each feature of the 150% PBS possessing variability, focusing on identifying the highest level in the PBS (assembly level instead of component level) where the variation is capturable. For example, identify the handset assembly as variable (touchscreen system versus mechanical keyboard system) rather than separately identifying the front face, rear face, wiring, and connector as having their own variability. Identifying the "correct" place to capture variability depends on the product, procurement strategy, and design strategy. Leverage the team member knowledge—product line manager, architects, and domain experts—as needed to create an accurate picture (Haughey 2020).

Step 4: Define the variability (refer to Figure 5). Defining the variability involves employing the syntax described in the following section. The syntax describes the variability level and type of each variation point, as well as dependencies between features—whether physical dependencies or market-based dependency decisions. By adding variability definition directly into the 150% PBS model, this step represents the key hybrid PBS-VM approach capability.

Step 5: Collaborate with the teams. Throughout the process, and especially to obtain final product buy-in, iterate with the development teams. This ensures the hybrid PBS-VM represents the intended as-specified product, ensures consistency with the engineering team organization, and ensures the hybrid PBS-VM supports the other stakeholder needs. For example, the PBS may be the basis for defining scope for suppliers and work packages; therefore, the new hybrid PBS-VM must also support this use case.

Syntax

The syntax used for the hybrid PBS-VM is a modified version of the decision model syntax presented in Czarnecki et al. (2012) Figure 1 (original sources are Dhungana, Grünbacher, and Rabiser 2011; Software Productivity Consortium Services Corporation 1993; and Schmid and John 2004). The syntax, implemented in a tabular format, enables formulaic development and eases data management.

Each model element uses the attributes identified in Table 2 (next page) to define the variation definition. If the attribute does not apply, then the visual model omits it. Together, these attributes define the element, if the element varies, how the element varies, the variability options available, and dependencies with other elements.

As examples of the above approach, we will use two elements from the IFE system installed in a business class suite: the USB & AC Outlet Unit module (Figure 6) and the Bluetooth Module (Figure 7). We want to understand if the inclusion is required or optional, the types allowed, how many to include, and if there are any dependencies.

From the Figure 6 information block, we can determine the following information:

- Name: the element name is "USB & AC Power Outlet Unit module"
- The following defines variability:
  - enum { USB-A | USB-C } { grounded | un-grounded } → there are two 'enumerated' type decisions to make. The first decision is for the USB port

Table 2: *Explanation of the attributes and syntax used to define variability of each element of the hybrid PBS-VM. These fields correspond to the example in Table 3*

Name	Description	Syntax
	To assist in identification of major points of variation or joining of sub-product lines; also used for visual aid in the visual representation	Top = top of the tree, Leaf = lowest defined element, Abstract = no direct physical implementation (e.g., Spare Parts System)
Variability Type	Identify where an element is invariant across all instantiations of the product line; otherwise, variation is dependent upon higher level variation points	Variable = element has variants, Invariant = element is unchanging across all instantiations, None = element variability is defined by other elements
Decision	Identify the type of variation—item is required, item is optional, item is purely custom	Choose = select from enumerated list, Include = select to include or exclude, Custom = customized element
Decision Type and Range	Identify the type of decision to be made, and identify the variants available for each decision	bool {T/F} for type Boolean, enum {list} for type enumerated
Cardinality	Identify the minimum and maximum number of each item that may be selected from the identified options	< min : max >
Quantity	Identify the number of elements that exist within each product instantiation; implies recursive selection of the element block	(QTY integer)
Condition	Must be met in order for the variation point to be activated	// logic statement
Constraint	Dependency with other elements	|| function statement

USB & AC Power Outlet Unit module

enum { USB-A | USB-C } { grounded | un-grounded } < 1:2, 1:1 > (QTY n)

Figure 6: *A hybrid PBS-VM syntax example for the USB & AC Power Outlet Unit module.*

type, with the range of options being *USB-A* or *USB-C*. The second decision is for the AC outlet type, with the range of options being *grounded* or *ungrounded*.

- < 1:2, 1:1 > → Cardinality tells us to choose 1 or 2 of the available options from the first decision and to choose only 1 of the available options for the second decision. In this way, a customer may choose to include *both* USB-A *and* USB-C ports.
- (QTY n) → The element quantity must be selected. For example, an airline may choose to include 2 "USB & AC Power Outlet Unit module" assemblies in the instantiated product, which would prompt the airline to choose the features of both block instances.

From the Figure 7 information block, we can determine the following information:

- Name: the element name is "Bluetooth Module"
- The following defines variability:
  - bool { T | F } → There is one 'Boolean' type decision to make: whether to include the module (T = True) or not (F = False).
  - < 1:1 > → Cardinality tells us to choose 1 of the available options from the first decision—True *or* False.
  - // IFE.supplier <> SupplierB → This constraint indicates this block can only activate if the IFE Supplier is *not* Supplier B.
  - || ifSelected Support.Module. Bluetooth = true → This condition indicates an outward dependency. If selecting the Bluetooth Module, then the Bluetooth Module Support (of the IFE Data & Electrical Harness) must also be True.

The hybrid PBS-VM visual model can directly use these blocks, and formatting can adjust to suit the user needs (refer to Figure 5 for the formatting used by the author).

Tools

The author developed the present examples using a combination of Microsoft® Excel® and Microsoft® Visio®. The author used formulas to assemble the syntax in Microsoft® Excel® (Table 3). The Microsoft® Visio® then imported the Microsoft® Excel® using the "Hierarchical Import Wizard" and custom blocks for formatting (Figure 5). Formatting occurs automatically using custom blocks and conditional formatting based on metadata.

However, Microsoft® Excel® could feasibly solely maintain the hybrid PBS-VM, with or without Macros for assistance in visualization. Additionally, using

Bluetooth Module

bool { T | F } < 1:1 > // IFE.supplier <> SupplierB || ifSelected Support.Module.Bluetooth = true

Figure 7: *The hybrid PBS-VM syntax example for the Bluetooth Module.*

Table 3: Tabular form of the syntax used to define each hybrid PBS-VM element. (*) denotes information visible in the element block. (**) denotes information used expressly during import into Microsoft® Visio® to create the hierarchy. Retain other information as meta-data and to assist in item definition. The data fields correspond to the descriptions in Table 2.

**Unique Identifier	*Element Type	*Name	**Hierarchy	Variability Type	Decision	*Decision Type	*Range	*Cardinality	*Quantity	*Condition	*Constraint
ID1	top	IFE System									
ID2	leaf	Monitor	ID1	variable	choose size and type	enum	{ 20in | 22in } { touchscreen }	< 1:1, 0:1 >			
ID3	leaf	Handset	ID1	variable	choose type	enum	{ touchscreen | non-touchscreen }	<1:1 >			
ID4	none	FE Power & Data Harness	ID1	variable	configure						
ID5	leaf	Audio module	ID1	variable	choose type	enum	{ 1-prong | 2-prong | 3-prong }	<1:1 >			
ID6	leaf	Bluetooth module	ID1		include?		{ T | F }			// IFE.supplier <> SupplierB	|| ifSelected Support.Module. Bluetooth = true
ID7	leaf	USB & AC Power Outlet Unit module	ID1	variable	choose type	enum	{ USB-A | USB-C } { grounded | un-grounded }	<1:2, 1:1 >	n		
ID8	leaf	Monitor support	ID4	variable	choose type	enum	{ touchscreen | non-touchscreen }	<1:1 >			
ID9	leaf	Handset support	ID4								
ID10	leaf	Audio Module support	ID4								
ID11	leaf	Power Data Box support	ID4								
ID12	leaf	Bluetooth Module support	ID4		include?	bool	{ T | F }	<1:1 >		// Bluetooth.module = true, IFE.Supplier <> SupplierB	
ID13	leaf	USB & AC Power Outlet Module support	ID4								
ID14	leaf	Active Acoustic Control support	ID4		include?	bool	{ T | F }	<1:1 >		// ActiveAcousticControl = true	

basic Microsoft® Visio® tools, Microsoft® PowerPoint®, yEd® Graph Editor, or other similar types of tools presenting graphical hierarchies can create the model. The author recommends programs possessing capabilities for automatic reorganization and spacing for ease of manipulation and visualization, especially with larger datasets.

Indeed, a hybrid PBS-VM benefit is its reactivity—because typical desktop tools can develop and manage the model, there is a low barrier to its usage in an organization.

Usage

While adjustable to individual organization needs, the hybrid PBS-VM is specifically useful in the following situations:

- For organizations already using a PBS, or organizations looking to introduce variability management capabilities
- Developing initial models supporting architecture trade studies
- As a complete VM for small systems or subsystems reasonably controlling configuration and variants without dedicated software tools
- As an intermediary or tool to aid in developing a formal VM
- As a means of communication to internal or external teams, suppliers, or customers; to facilitate an understanding of the breadth, complexity, and interdependencies of a product line and architecture

Additionally, the hybrid PBS-VM is extensible for hardware-only systems, software-only systems, and combined hardware-software systems. Additional syntax may be necessary to differentiate between configuration item types.

CONCLUSIONS

The work presented leads to the following conclusions:

- A traditional PBS is insufficient to describe the product line architecture and its effectiveness

- A hybridized syntax, combining decision model and feature model concepts, can concisely capture product line reusability and variability
- A hybrid PBS-VM can provide an effective bridge between traditional project-based engineering and PLE approaches
  - A hybrid model is readily understandable, yet captures the product line approach effects
  - A hybrid model can be an intermediate step to developing a formal VM
- Effectiveness maximizes when used early in the product line lifecycle, with small systems, or with organizations with a low maturity level in formal PLE

FUTURE WORK

In developing this new model, future work will exercise the model over a broader product line range—both hardware and software—to discover and resolve deficiencies; investigate how to integrate the model with enterprise tools; and ensure consistent, high-quality, and reproducible approach implementation. Specifically, the author has identified the following activities:

- Use the hybrid PBS-VM in developing new product lines and systems including hardware, software, and integrated systems, both large and small.
- Develop the tools to enable automation and enforce implementation and syntax consistency.
- Identify means to integrate the hybrid PBS-VM into enterprise tools, such as model-based systems engineering (MBSE) tools, requirements-based systems engineering (RBSE) tools, CM tools, PLM tools, and computer-aided design (CAD) tools.
- Develop detailed usage guidelines, including decision logic in decomposing systems to maximize the correctness and utility of the resulting model.
- Expand the model to support additional approaches for variability and reuse—reusable modules, standardized interfaces, and product lines-of-product lines.
- Expand the model to support additional techniques for variety—fixed versus variable, combination, multi-functionality, range, and trend and margins (Safran 2020).
- Align the methodology and terminology with the upcoming standard ISO/IEC DIS 26580 "Software and Systems Engineering—Methods and Tools for The Feature-Based Approach to Software and Systems Product Line Engineering." ■

REFERENCES

- CESAM. 2017. *CESAM: CESAMES Systems Architecting Method, A Pocket Guide*. Paris, FR.
- Czarnecki, K., P. Grünbacher, R. Rabiser, K. Schmid, and A. Wasowski. 2012. "Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches." White paper, Association for Computing Machinery. https://dl.acm.org/doi/10.1145/2110147.2110167
- Dhungana, D., P. Grünbacher, and R. Rabiser. 2011. "The DOPLER Meta-Tool for Decision-Oriented Variability Modeling: A Multiple Case Study." *Automated Software Engineering* 18 (1): 77–114.
- Haughey, D. 2020. "Project Management Tools." *Project Smart*, 20 October. https://www.projectsmart.co.uk/project-management-tools.php
- International Council on Systems Engineering (INCOSE). 2019. "Feature-based Systems and Software Product Line Engineering: A Primer." INCOSE, San Diego, US-CA.
- Kang, K., S. Cohen, J. Hess, W. Nowak, and S. Peterson. 1990. "Feature-Oriented Domain Analysis (FODA) Feasibility Study." Technical report, CMU/SEI-90TR-21.
- Krob, D., and L. Le Sauce. 2015. "Product Families Architecture." CESAMES, Paris, FR.
- Safran. 2020. "Develop Handbook 3 Predesign and Design the System." In *Internal Handbook*, 20. Paris, FR.
- Schmid, K., and I. John. 2004. "A Customizable Approach to Full-Life Cycle Variability Management." *Science of Computer Programming*, 53 (3): 259–284.
- Software Productivity Consortium Services Corporation. 1993. "Reuse-Driven Software Processes Guidebook" Version 02.00.03. Technical Report, SPC-92019-CMC.
- United States Department of Defense. 2018. "Work Breakdown Structures for Defense Materiel Items." United States Department of Defense Standard Practice, MIL-STD-881D.

ABOUT THE AUTHOR

Evan Helmeid is currently the product line systems engineer at Safran Seats in Paris, France. He is responsible for developing, implementing, and advocating for product line engineering processes across the Safran Seats division, including locations in the United States, Great Britain, and France. Since beginning his career in 2011, he has supported private start-ups, large public companies, and various government entities. He has held technical leadership, project management, project engineering, and systems engineering roles across launch vehicle, satellite, drone, and ground systems programs. He received his BS in aeronautics and astronautics from Purdue University and his MS from the University of Alabama in Huntsville.

APRIL 2021
VOLUME 24/ ISSUE 1

The Convergence of Struggles! Reusability Assessment of Inner-Source Components for Product Lines

Thomas Froment, thomas.froment@thalesgroup.com, and **Guillaume Angier de Lohéac**, guillaume.angierdeloheac@thalesgroup.com

ABSTRACT

Inner source is establishing open source-like collaborations within an organization. Product Line Engineering (PLE) is the approach for engineering a related product portfolio in an efficient manner, taking advantage of products' similarities while managing their differences. These two well-documented approaches propose smart techniques for reuse, but they use different terminology. A language characteristic is to be polysemic and polymorphic. Indeed, PLE and inner source do not use the same words to refer to equivalent concepts. This could badly affect project performance when evolving in a multi-domain context. What if there was a way to better integrate PLE, inner source, modeling, data management, hardware and software engineering, and integration, verification, validation, and qualification (IVVQ) through the similarity concept?

This paper shows it is possible to build a common way to assess the components (also called building blocks) contributing to a product line, thanks to a process to determine the component maturity levels using the similarity approach. After detailing the commonalities between the inner source and PLE domains, we present the Inner Sourcing Process Maturity Level (ISPML) as a key engineering practice. Why is it important for engineering? If engineers reuse components defined by their engineering assets, it is important to have a formalized, common way to do this across the company to integrate reusable multi-domain assets with a certain confidence level. This paper introduces a simple method for multi-domain organizations to better determine whether sharing an engineering asset is favorable or not.

INTRODUCTION:

Why it is a linguistic issue?

There are multitudes of ambiguous words in all languages (Cisse 2007), which can create misunderstandings. Approximately one in two words out of its context is ambiguous in Indo-European languages. There is no exception in day-to-day professional context. For instance, engineering domains like PLE and inner source do not use the same words to refer to equivalent concepts. This could badly affect project performance when evolving in a multi-domain context. Before focusing on engineering areas, it is necessary to clarify the problem. Philosophy and linguistics are the correct domains to define the problem properly.

Language is the capacity to express a thought and communicate by a system of signs endowed with semantics, and most often a syntax (Bergounioux 2021).

Ferdinand de Saussure (1857-1913), a linguistics founder, declares two things constitute the linguistic sign: the signifier and the signified (Bergounioux 2021). You cannot tell the two apart, but you have to understand the two together to make sense. The relationship between the signifier and the signified is conventional. All people in a linguistic community must learn which words correspond to which images. For instance, to mean "car," French speakers use *voiture* and Spanish *carro*. However, translation is rarely a "one for one."

The Sapir-Whorf Hypothesis better understands the translation issue. It states native languages strongly affect the way people think. It is a controversial theory championed by linguist Edward Sapir (1884-1939) and his student Benjamin Whorf (1897-1941). A well-known example to support their theory is the numerous words the Eskimo language has for snow when the English language has only one. In

English-speaking literature, we find examples such as George Orwell who will exploit this idea in his novel *1984*. In this work, a totalitarian power modifies the official language so the thoughts questioning it are not, in the long term, even expressible (the famous "Novlangue"). Beyond this example from popular culture, we must consider the relationship between the signifier and the signified involves variability. Context and social interaction influence language variability. Ferdinand de Saussure used "paradigm" to refer to a class of elements with similarities (Bergounioux 2021).

Another domain, philosophy of sciences, uses "paradigm" to refer to variability and studying social influence. Thomas Kuhn (1922-1996) defines paradigm as "universally recognized scientific achievements that, for a time, provide model problems and solutions for a community of practitioners." Kuhn reckons science does not progress via a linear knowledge accumulation but undergoes periodic revolutions, also called "paradigm shifts." Kuhn saw the sciences as going through alternating "normal science" periods, when an existing reality model dominates a protracted puzzle-solving period, and "revolution," when the reality model itself undergoes a sudden drastic change. Generally, guided by the paradigm, normal science is extremely productive: "when the paradigm is successful, the profession will have solved problems its members could scarcely have imagined and would never have undertaken without commitment to the paradigm" (Kuhn 1962). According to Kuhn, adherence to a paradigm is a sociological phenomenon, which requires creating a community of thought, methods, and objectives around common tools (journals or conferences). The term "paradigm" developed by Thomas Kuhn, which he moreover suggested disciplinary matrix should replace, tends to designate all the beliefs, values, and techniques shared by scientific community members during a theoretical consensus period.

According to him, "the paradigm is a framework which defines problems and legitimate methods, and which thus allows a greater research efficiency: a common language favors disseminating the work and channels the investigations." The most typical paradigm examples cited by Thomas Kuhn are the Ptolemy paradigm (geocentrism), the Copernicus paradigm (heliocentrism), the Newton paradigm, and the general relativity paradigm (Einstein).

In an engineering context, PLE and inner source are two paradigms (or disciplinary matrices) with common methods. One way to better integrate them is to first check commonalities and gaps, then share problems to solve and solutions to make a common vision possible. Before getting to the heart of the matter, we need to introduce inner source.

Inner Source Definition and Motivation in Corporate Companies

Inner source takes the lessons learned from developing open-source software and applies them to the way companies develop software internally. In other words, inner source stands for "group-wide" internal open source.

Why is the inner source practice developing on a large scale in an ever-increasing number of large and even medium-sized technology companies?

The observation is: the open-source world is, without any discussion today, where innovation is the most dazzling and has been for more than 20 years now. Companies sometimes find it hard to innovate and, even when they have good ideas, putting engineering into practice for a multi-domain, multi-entity, or multi-country project often comes up against major difficulties and often leads to failure.

So why implement inner source in the industrial world?

- **Reuse:** This is the most obvious reason. Reusing components, building blocks made by other entities, is a way to save time and therefore optimize costs while reducing time to market.
- **Transparency and serendipity:** Beyond the financial aspect, the founding inner source practice element is an engineering practice that is very disruptive and based not on a method strictly speaking, but on a principle: transparency. All the actions, all the stakeholder decisions are in full view of everyone, even those who are very indirectly involved in the company. Beyond the behavioral change this induces, which is part of the transforming management mode (REF) general logic, this transparency in itself generates new opportunities—not foreseen at project start—techniques and even business, within the company. We will address "serendipity" which literally means finding a new idea "by a happy coincidence." But applying transparency at all times somehow "provokes" this chance, and this is where we often find major innovations.
- **Quality:** Finally, it is also a marker, counter-intuitive for some, of open source: excellence in the delivered quality. Although not guaranteed, best practices in software craftsmanship such as delivering well-tested code, implementing test-driven development, doc as code, code reviews, and a complete continuous integration chain are not optional. Indeed, project stakeholders are often extremely diverse; do not share the same practices, even within the same company; and, are often unable to physically work in the same room or at the same time. The only "viable" way to deliver a working product then becomes not to break away from best practices and to be extremely rigorous in configuration management, continuous integration, and governance rules, especially for everything related to the product's "common parts."

COMMONALITIES AND GAPS BETWEEN THE INNER SOURCE AND PLE DOMAINS

In this chapter, we compare *signifier* and *signified* (or *word* and *definition*, the two linguistic sign parts defined by Ferdinand de Saussure), to check commonalities and gaps between the inner source and PLE domains.

Common sign: Same Signified and Same Signifier

The following words share the same definition in both domains.

Product: Intended to sell, directly or indirectly (internal product), to customers for satisfying their expectations and meeting their operational requirements. A product can be hardware or software equipment, a service, a system, a replicable combination of the previous items (Thales Group 2015).

Product Roadmap: The product lifecycle master plan, typically showing the major update releases with new features over time, the phase-out, and their associated estimated budgetary needs. It starts with the market rendezvous (market events, must-wins, and major targeted bids and projects). The product roadmap is part of the product plan.

Solution: A consistent set of systems, equipment, and services provided to the customer to meet his requirements. Rule: The solution should maximize product use according to its competitiveness and attractiveness (Thales Group 2015).

Unique to PLE

Business Plan: A financial modelling describing a business activity with a specific focus on its profitability over time. Formalizes the company's future development plan and profitability in product terms.

Feature: A distinguishing characteristic describing how the product line members differ from each other (INCOSE 2019). This provides a common language and defines the product line's scope of variation for the organization. Feature-Based Product Line Engineering is a specialized and highly

APRIL 2021 VOLUME 24/ ISSUE 1

efficient PLE form. Feature-Based PLE relies on a managed feature set to describe the distinguishing characteristics setting the products in the product line apart from each other (INCOSE 2019).

Even if feature is unique to PLE, it is a known concept in inner source, and it could then apply in some circumstances. We discuss its applicability for inner source later.

Product Line: A product group related in they have similar technical and functional specifications and address the same market segments or the same customer groups, same operational requirements (Thales Group 2015). Products and/or services for specific markets with explicitly identified commonalities and variabilities and developed on the same architecture.

Product Line Family: Product line groups designed to meet the common and variable needs of several market segments (Thales Group 2015).

Unique to Inner Source

TISS: Thales Inner Source Software is the inner-source program in Thales. This follows the internal open-source foundation model.

Contributor: Someone who contributed to a TISS project: contributions range from simple comments all the way to software code. A contributor or anyone else who is not a committer has no right to force their contribution to integrate into the main project. They can create a derivative project (a fork) to integrate their contribution.

Committer: A TISS project technical authority. A committer is either a Thales project team member who initially developed the asset(s) or a contributor who has received commit rights in a TISS project.

Project Management Committee (PMC): A TISS project product authority. Any TISS project PMC has one or many members. The PMC has authority regarding publishing TISS assets.

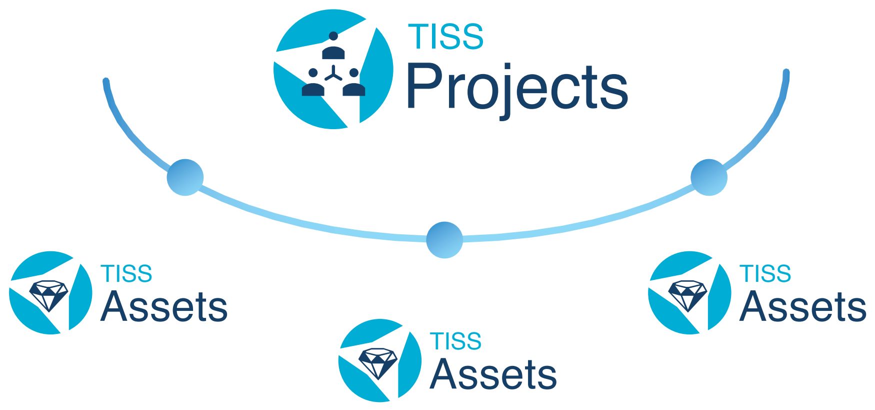
Core Team: The group of people including committer(s) and PMC member(s)

Gaps:

Same signified but a different signifier

Program (PLE)/Top-Level Project (Inner Source): A program is a group of interdependent projects coordinated to obtain benefits and control not available from managing them individually.

Building Block (PLE)/TISS Asset (Inner Source): A reusable modular element of a higher level product not sold directly to customers. It can be managed as a product as a configuration item engineered for re-use purposes (long-lasting interfaces and design).



A single TISS project may share one or many TISS Asset(s).

Derivation (PLE)/Fork (Inner Source): Operations allowing project initialization with the relevant product assets (product instances) that require product reuse.

Through inner source, a project fork happens when developers take a source code copy from one software package and start independent development on it, creating a distinct and separate software piece. The term often implies not merely a development branch.

Generalisation (PLE)/Pull (or merge) Request (Inner Source): Operation enriching the product by integrating re-usable assets developed in a customer contract to positively contribute to the product value. This operation either adds new product capabilities compatible with the product roadmap or contributes to technical debt reduction.

Through inner source, pull (or merge) requests mean *pulling* changes from another branch or fork into your branch and *merging* the changes with your existing code. This generally associates with contributing rules (typically, a code review and test coverage).

Different signified but the same signifier

Project (PLE): A unique time and cost-constrained activity set using resources to achieve stated objectives (usually deliverables up to quality standards and fulfilling requirements).

A "project" refers to the temporary organization and means established by the company to execute a contract with an external customer and to deliver the required goods and services (**the solution**), from which the company expects profits, as well as other business benefits such as growth, strategic market placement, and industrial footprint.

TISS Project: Software components shared in TISS (assets) built by teams working on Thales projects. So, a TISS asset, at least initially, designs and develops in a Thales project context. A TISS project is home to one or many shared TISS assets. Note TISS project and Thales projects do not have the same meaning: one Thales team working on a Thales project may share one or many TISS assets through one or many TISS projects.

Shared assets (PLE): Artifacts supporting product creation, design, implementation, deployment, and operation. They can be digitally represented and configured, and share across the product line.

TISS Asset (Inner Source): A reusable modular element of a higher-level product and not sold directly to customers. We can manage it as a product as a configuration item engineered for re-use purposes (long-lasting interfaces and design).

Chapter outcome

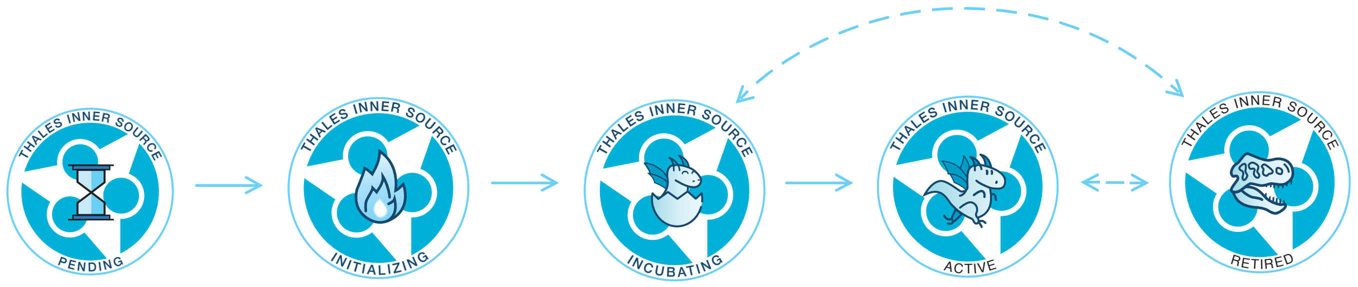
This review shows how any collaboration between the PLE domain and inner source would be difficult because of the language gaps. On the other hand, it illustrates how close the domain activities are. A way to find a solution is marrying two practices from each domain to build a common practice with common words. It is a first step to our convergence of struggles. The two selected practices are Inner Sourcing Process Maturity Level (ISPML) and Feature-based PLE.

INNER SOURCING PROCESS MATURITY LEVEL (ISPML)

Context: Current Thales Implementation
The process described below is the result of the Thales Inner Source Software (TISS) program deployment in Thales.

This program aims at:

- Providing facilities to share source code or components between any Thales entities, by applying best practices from the open-source communities.
- Fostering Thales engineer's group-wide collaboration by better reuse



and adaptation of existing code and components.
- Supporting inner source governance including Intellectual Property Rights (IPR) and licensing policies.
- Enabling a collaborative infrastructure including a full-fledged forge, a new engineering community, and any other relevant services (search engine and maturity assessment) improving collaboration between Thales developers.

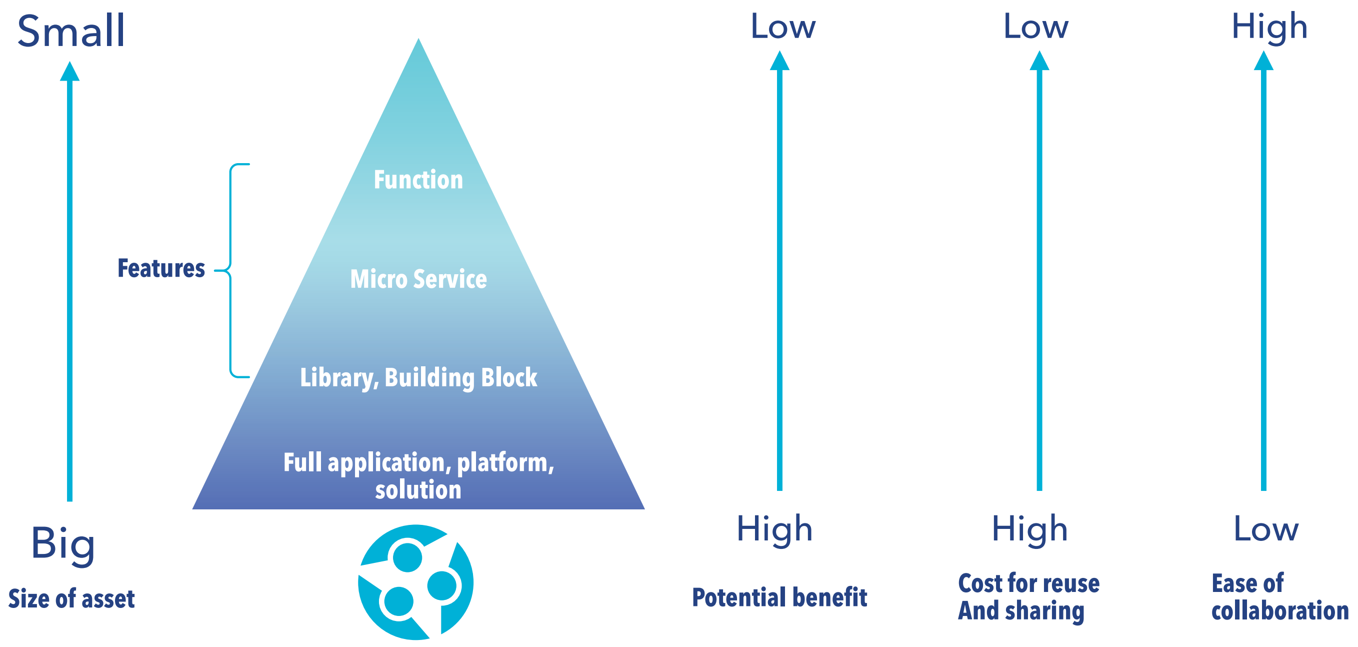
Workflow overview

This part describes how the TISS program measures the project maturity level regarding the inner source engineering practices.

- **Pending:** This is a preliminary step where the project team and organization evaluate the opportunity to go/not go for sharing a new TISS Asset.
- **Initializing:** This state corresponds to the phase where the project team gathers all prerequisite information to be ready for the collaboration. It includes:
  - Checking it complies with the product strategy, by getting official approval from product management and owner.
  - Legal Item Verification: confidentiality, export regulation, contracts, and IPR
  - Project pitch redaction describing its main purpose and positioning
  - Registration in a public catalog. This catalog allows project searchability. It is available from any place within the company.
- **Incubating:** Reached when the collaboration is in place. Thus, the TISS project team is ready to look for new external contributors. It implies:
  - Initial stakeholders clearly define roles
  - Open source inspires inner source roles, defined in the "Unique to Inner Source" section
  - Source code is available within the company
  - Legal license is packaged
  - Documentation is available, including: (1) product vision, (2) how to use, and (3) contributions rules. The project technical and product authorities define the contribution rules.
- **Active:** Reached when:
  - The TISS Project receives contributions from external contributors for at least one shared TISS asset. Contributions range from simple comments to software code. A contributor or anyone else who is not a committer has no right to force their contribution to integrate into the main project. They can create a derivative project (a fork) to integrate their contribution.
  - The project checks open source compliance and cybersecurity rules.
  - All regulation and legal questions lift, allowing any other user in the company to integrate TISS assets in a commercial product.
- **Retired:** Project is still in the TISS Project catalog and marked as retired, but it may still revive. It occurs when:
  - There are 18 months (or more) without any project activity.
  - There is no active member in the core team, and nobody is ready to take the role in the company.

Key Criteria and Tradeoffs for a Successful Reuse

Over the TISS program time and experience, which started 4 years ago, the ISPML evaluates the inner source project



chances of success. Indeed, the program only achieves its objectives when reaching the ACTIVE state, even if the preliminary states (INITIALIZING or INCUBATING) demonstrate the heterogeneous teams' ability to work collaboratively. In this context, the following decisive criteria appear:

- **Granularity:** The shared TISS asset size. The smaller it is (a Feature—equivalent to a PLE Feature, or a Micro Service), the easier sharing it will be, and the easier collaboration will be. Conversely, sharing an entire application, a platform comprising multiple services, or a solution will be much less likely to reach the ACTIVE state.
- **Architecture:** The shared asset architecture also plays a key role in the success criteria. The more dependencies on the environment, other assets, or third-party libraries the higher the reuse cost and the fewer contributors it will find.
- **Cost vs Benefit:** On these two axes (granularity and architecture), there is obviously a trade-off with the expected reuse benefit. Indeed, if the assets are small, or if they are completely self-supporting, this will be very favorable to collaboration, but the value and therefore the benefit will be less. Conversely, sharing a complex platform (for example, a solution for cloud computing) may take several months or even years to be ACTIVE, but will give the organization a huge benefit and therefore a potentially major advantage over its competitors.

CONCLUSION

Introducing ISPML in collaboration maturity monitoring helps measure the success criteria of multi-entity, multi-country cooperation in an industrial company framework.

Trade-offs must emerge to estimate the success probabilities in return on investment terms of reuse, quality, and new induced cooperation—innovation and serendipity. We present here the recapitulative pyramid of these tradeoffs.

So today, we in Thales work towards this convergence between PLE and inner source, allowing us to work on an assessment system to:

- Evaluate the opportunity to start an inner source process (or not) within the PLE approach framework.
- Confirm the appropriateness (or not) of adopting a Feature-Based PLE approach for inner source projects when sharing the following asset tip: function, micro-service, library, and building block.

The obstacles to overcome before implementing these tools were:

- The linguistic issue, which this article has highlighted.
- The difference in approach: Inner source comes from a very bottom-up culture built through experimentation, and is at the beginning, far from a structured approach through theory like PLE.

The two worlds come together, allowing field experience to confirm theory and theoretical contributions to enrich practice. This is a new field opening up promising perspectives. ■

REFERENCES

- Bergounioux, G. 2021. «Cours de Linguistique Générale, Ferdinand de Sausssure – Fiche de Lecture.» *Encyclopædia Universalis*. https://www.universalis.fr/encyclopedie/cours-de-linguistique-generale/
- Cisse, M. 2007. «Analyse Distributionnelle et Approche Pragmatique. Recherches sur les Phénomènes d'Ambiguïté et de Désambiguïsation Linguistiques.» *Annales de la Faculté des Lettres et Sciences Humaines de Dakar* 31 (1): 1-16.
- INCOSE. 2019. *Feature-Based Systems and Software Product Line Engineering: A Primer*. San Diego, US-CA: INCOSE.
- Kuhn, T. 1962. *The Structure of Scientific Revolutions*. Chicago, US-IL: University of Chicago Press
- Thales Group. 2015. "Product Line." *Chorus Reference System*.

ABOUT THE AUTHORS

Thomas Froment is Thales Inner Source Initiative Lead supporting engineering transformation deploying open-source best practices (tools and culture) as an enabler of Thales Digital Transformation. He previously worked on Thales Cloud Computing Infrastructure & Management (IaaS) Automation solution. Thomas holds 14 patents, is an IETF RFC author, Certified Scrum Master, and Certified SAFe Agilist. He has earned an MS in computer science and two additional engineering degrees.

Guillaume Angier de Lohéac is a product line engineering specialist at Thales. He supports and trains project teams in the deployment of methods and tools in the product line engineering domain. He monitors deployments, promotes sharing of information within the internal Thales community, and participates in the continuous improvement of engineering practices. Guillaume earned an degree in mechanical and electrical engineering and a license in philosophy.

APRIL 2021
VOLUME 24/ ISSUE 1

Product Line Engineering for Digital Product-Services

Guillermo Chalé Góngora, hugo-guillermo.chalegongora@thalesgroup.com; **Pierre-Olivier Robic,** pierre-olivier.robic@thalesgroup.com; and **Danilo Beuche,** danilo.beuche@pure-systems.com

■ **ABSTRACT**
Digitizing the value chain brings along new business opportunities to organizations wishing to adopt a service-oriented approach. This trend, referred to as digital transformation, has taken over the business world in recent years. Digital technologies allow a company to move towards outcome-based commitments, pricing, and contracts with its customers. However, these technologies can also make product portfolio management more challenging. Whereas transforming a product offering into a product-service offering through digitization does not in itself revolutionize product line systems engineering processes and methods, it is of the utmost importance to consider this transformation concerns more than a new culture or using new technology and requires, first and foremost, an alignment with an organization's global strategy and structure.

In this paper, we present a conceptual framework that helps define a high-level strategy to implement a product-service offer in an organization. The distinctive framework aspects include the Product-Service Product Line or PSPL concept (a product line of product-services), the elements to define the PSPL business model (pricing model, pricing metrics, and commitment types), a product-services typology, and a product line engineering method extension for architecting the PSPL (notably, a specific service building block type to support a composable design approach and a feature model including service-related, socio-technical features).

1. SERVICE-ORIENTED APPROACH ORIGINS

In recent years, growing numbers of manufacturing companies have integrated more services into their product offerings. Some have even changed their business models radically and started to sell their products as a service. The notion behind this approach is to better address the customer needs by assigning customer value to the operation performance, utility, or quality of a product and not to the physical product ownership. This new business model changes the manufacturing company's motivations and objectives. Their target is to maximise their product utility and to make them perform well for as long as possible, instead of selling the highest possible product volume. This usually happens by monitoring, maintaining, repairing, upgrading, and reusing their products more.

Turning to service-oriented business models represents a logical move to identify growth opportunities by manufacturing companies. To move into service-oriented business models, most companies will try developing new capabilities providing customers increased value through new technologies or by leveraging the company's expertise while trying to build longer and more profitable customer relationships (Queinnec and Tan 2018).

Observing a manufacturing company enhancing its commercial offer through service provisions is nothing new. Sales, delivery (including documentation and training), after-sales warranty, maintenance, spare parts, and repair are examples of services that have accompanied many commercial products for years, from household appliances to automobiles and airplanes. Yet, in most organizations, different teams working mostly in silos and following different processes carry out the service and product design and development (Polaine, Løvlie, and Reason 2014, and Schnürmacher, Haygazun, and Stark 2015). This silo effect amplifies when a company adopts digital technologies to support the services they provide. Besides incorporating personnel with different cultures and competencies, the most common reason for this amplification is many people in the company lose sight of the digital transformation purpose; digital technologies are suddenly *the objective* instead of *the means* by which a business can succeed in the new landscape. Whilst digital technologies offer a company enriched support for decision-making and the possibility to propose to its customers a higher engagement level in its core services, they can also make the product portfolio management challenging if they do not use a structured approach.

2. A PRODUCT LINE SYSTEMS ENGINEERING FRAMEWORK FOR PRODUCT-SERVICES

To face the challenges presented above, we propose a conceptual framework to define a structured approach for Product-Service Product Line Engineering. Observe the proposed framework is a *complement* to reference architectures or existing architecture frameworks and must be considered as such, rather than as a stand-alone, complete framework for architecting

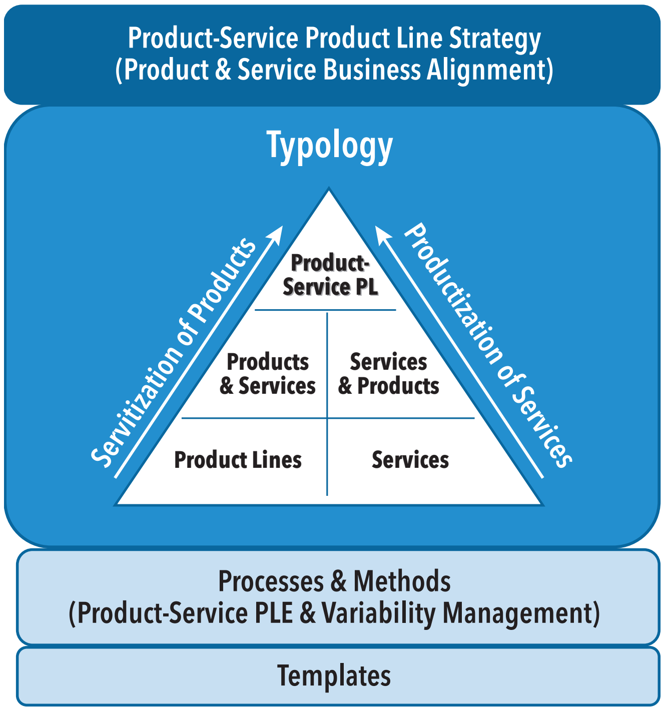

Product-Service Product Line Strategy (Product & Service Business Alignment)
Typology
Product-Service PL
Products & Services
Services & Products
Product Lines
Services
Servitization of Products
Productization of Services
Processes & Methods (Product-Service PLE & Variability Management)
Templates

Figure 1. Conceptual framework for product-service PLSE

a Product-Service Product Line. The framework components can indeed supplement existing reference architectures or easily incorporate into commercial frameworks by identifying the viewpoints to which our conceptual framework elements better relate (for instance the strategic, operational, service-oriented, or systems viewpoints in the Ministry of Defence Architectural Framework).

Adapted from Baines et al. (2007), the proposed framework comprises three main components plus a template set (not presented in this paper) to facilitate formalizing the different elements resulting from applying the framework (Figure 1):

- A PSPL typology to characterize the product type composing a product line, for example from traditional products (tangible and physical products) to services and product-services;
- Processes and methods for architecting and engineering the PSPL, in particular, for managing the PSPL variability, dependent on the product types composing the PSPL;
- Guidelines to align the traditional product line business strategies with those of their corresponding, and less traditional, service product lines.

Applying the framework results in characterising the elements typically defined in the business or mission analysis process: the business or mission problem or opportunity, characterising the solution space including identifying effectiveness measures, and determining potential solution classes to address the problem or take advantage of an opportunity (INCOSE 2015). These elements effectively frame or orient the remaining systems engineering technical processes requiring execution to achieve a successful Product-Service Product Line.

The triangle base and left side in Figure 1 have been the object of several studies (Ducq, Chen, and Alix 2012), but little has been said about the **right side** or the triangle **apex**. References are also scarce when considering the methodology definitions needed to implement a service-oriented approach in an organization that has historically developed complex products. Using Product Line Engineering to address these gaps appears extremely promising to the authors.

Compared to previous research on service typologies and product service systems (Cook, Chon-Huat, and Chen 1999, Wild et al. 2007, and SEBoK 2019), the proposed framework presents some distinctive characteristics:

- Extending service typologies to encompass the different service and product dimensions developed by our company (from “pure” products and services to integrated Product-Service Product Lines);
- Considering services as products and applying a product line approach to structure their development;
- Applying Feature-Based Product Line Engineering (INCOSE 2019) to the Product-Service Product Line design and development (that is, to a Product-Services *family*), as opposed to applying service engineering to developing *individual* product-service systems as often described in literature (Tan and McAloone 2010, Polaine, Løvlie, and Reason 2014, Kumar et al. 2017).

The following sections briefly explain the different framework elements.

2.1. Product-Services Product Lines Typology

2.1.1. Extended Products. Figure 2 shows the *extended product* concept. The core product is the traditional physical good offered on the market by a manufacturing company. In general, this core product completes with a product shell describing the “tangible packaging” of the product (delivery, installation, and user manuals). Supporting services are “intangible” additions to the core product facilitating or guaranteeing proper product use (maintenance plans or mobility warranties). Differentiating services allow individualizing the extended product and usually relate to operation services.

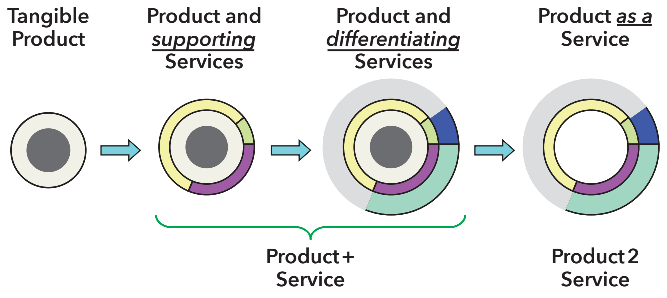


Figure 2. Extended product and the servitization process (Ducq, Chen, and Alix 2012)

2.1.2. Services. To classify product shells, supporting, and differentiating services, the service typology proposed in our framework somewhat follows principles from those typically used in service engineering. Service engineering usually bases classifications on the responsibility level and risk taken by the manufacturer (which follows a linear economic model), and range from services supporting a product to services supporting customers. Our framework bases service classification on the lifecycle stage delivering the service and on the scope of the service. Table 1 shows a partial view of the proposed service typology.

2.1.3. Product-Service Product Lines. As exposed above, products tend to come with a set of associated supporting and differentiating services. Service industries, on the other hand, tend to offer supporting systems (tangible products, such as infrastructure and platforms) associated with their services. These two tendencies (respectively known as *product servitization* and *service productization*) converge towards integrated Product Service System (PSS) offers in today’s business world (Baines et al. 2007). We have adopted this notion in the proposed framework, but we have

Table 1: *Service typology of the Product-Service PLSE Framework*

Studies – Service Engineering phase	Service Engineering & In service phases	Service Delivery – In service phase
• Provide Logistic Support Analysis (LSA) • Perform Reliability, Availability, Maintainability & Testability (RAMT) studies • Perform Life Cycle Cost studies	• Produce customer documentation • Provide customer training • Provisioning	• Manage obsolescence • Provide consultancy or technical assistance • Repair equipment or systems • Manage the information security of a system • Provide contact service to customers • Provide online service to customers • Manage spares, consumables & materials • Manage Support & Test Equipment (STE) • Deploy equipment or systems • …

expanded the PSS scope with the ***Product-Service Product Lines (PSPL)*** concept: a family of similar product-service systems.

The objective of a product-service is providing an **effective operational performance** to customers. Properly engineering a PSPL requires considering not only the tangible products plus the systems supporting their differentiating supporting services but also the *system of operation* as part of the whole ecosystem. This means considering the customer's organization, procedures, and stakeholders along with the technological PSPL components. Consequently, the PSPL consideration scope migrates from technological concerns to socio-technological concerns including human factors and organizational aspects.

2.2. Processes and Methods for Product-Service Product Lines

Product Line Systems Engineering (PLSE) broadens the systems engineering process activity scope with methods and tools from Product Line Engineering devoted to engineering a family of similar products exhibiting variations in their characteristics (INCOSE 2019, Chalé and Greugny 2017). This occurs through defining, using, and managing a shared configurable engineering asset set leveraging the commonality within the family, and through systematic and rigorous variation management amongst the family products using a feature model (Beuche 2008).

For an effective PLSE application to Product-Service Product Lines, certain enhancements of our traditional PLSE processes and methods appeared necessary. The next paragraphs present these enhancements.

2.2.1. Shared Assets and Feature Models. While the product line shared assets typically associate with the *engineering* product lifecycle in published case studies, Product-Service Product Lines require us to enhance these assets with aspects related to people, organizational governance, and business. These aspects represent new variability sources within the Product-Service Product Line, implying they introduce features differing from those found in feature-based case studies and raise the need to manage variability at different but interdependent levels (Figure 3):

- Business: commitment types, service level agreement, and pricing models
- Operational processes and governance: linked enterprise architectures and actors
- Means: a contact center based on PABX, Front Office, portal, or customer relationship management
- Technical interfaces and flow types: information, money, goods, or human resources
- People: skills, competencies, and job roles.

In Product-Service Product Lines, an additional feature modeling goal is supporting the efficient variability management stemming from contract commitment and from the product-service itself (including the variability of the enabling systems of the Product-Service, as explained in the next section). This facilitates the **customer value extraction** that can translate into higher performance and/or into a competitive advantage for the company.

2.2.2. Service-System Building Blocks. Service-systems (Service BoK AFIS 2018) are basic constructs possessing the capabilities and encompassing all the elements needed to ensure service delivery. A service-system (Figure 4) is a socio-technical system comprising:

- A System of Interest—Typically, a product expected to provide some service type by satisfying specified performance, behavioural, and operational needs;
- A System of Support enhancing the operational system of interest capabilities (equipment and systems delivered under availability or output-based commitments);
- A System of Operation enhancing operational capabilities

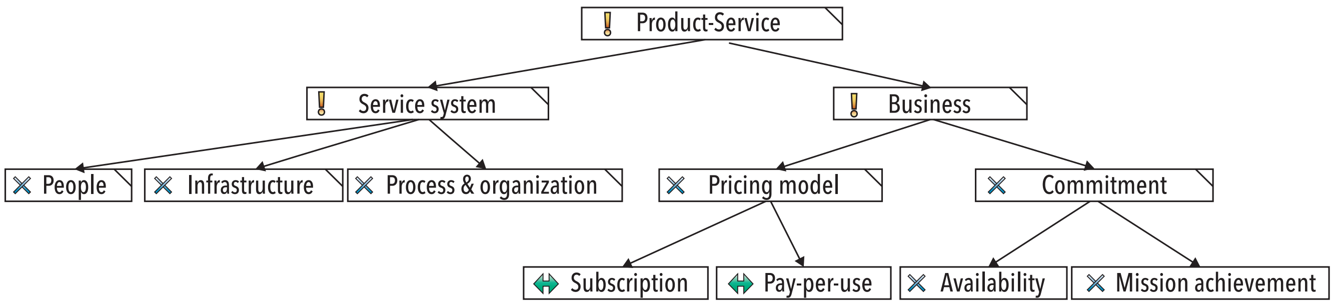


Figure 3. Service-Related feature examples for a product-service product line

to achieve overall operational performance for the customers and contributing to the success of their core missions (education, training, supply chain management, fleet management, and mission preparation).

Whatever their nature, behind any service there is always a socio-technical service-system.

To perform efficiently, the different service-system elements must operate under the same governance, using common infrastructures, assets and tools, and share common core competencies. Figure 5 shows a service-product description from this holistic perspective. Each element in this figure can configure and transform using a selected feature set (like those shown in Figure 3) which "activates" the required element variation points (for instance, including specific skills and competencies needed to operate the service-system). The service-system also provides interfaces to allow interactions across its boundary with other service-systems.

2.2.3. Composable Product-Services. The service-systems described above are the elementary, configurable building blocks (or cells) to define the product-services architecture via composition. Configuring and combining various service-system building blocks yields different *service capacities* (Figure 6). The *composed* service-system resulting from combining service-systems building blocks supports these service capacities.

Efficiently assembling different service-systems demands commonality analysis of its constituent building blocks. This analysis aims to optimize using and allocating means and people across the different composed service-system building blocks, as well as the coupling of the building blocks (the information, money, goods, and human resource flows they exchange), as advocated in socio-technical practices. In our approach, the configuration and automation mechanisms provided by Feature-Based PLSE effectively control and operate the optimization of the composed service-system. The feature selections to configure the overall service-system in fact cascade to all its building blocks, which in turn configure and transform via internal variation points. A user-defined ruleset, evaluated when operating the configuration and automation mechanisms, guarantees the consistency of the different transformations. Examples of these rules include: producing named data by one unique element or the presence of only one invoicing system.

These optimization mechanisms also enable us to guarantee service *continuity*, a key challenge in service-products. We

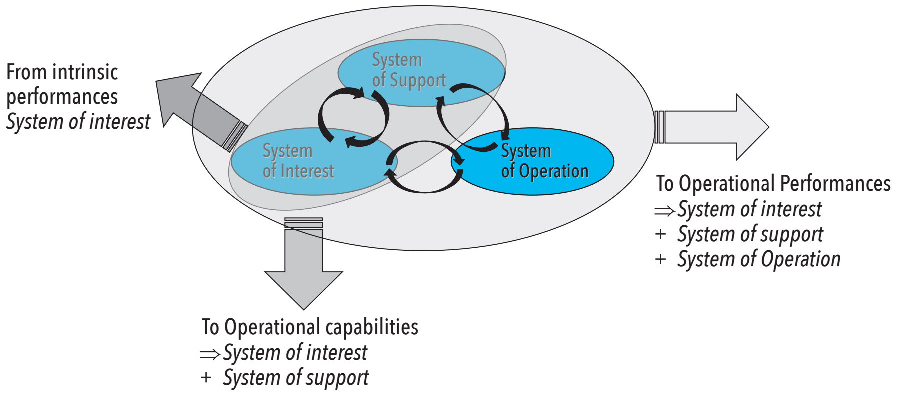


Figure 4. *A service-system and its three main components (Service BoK AFIS 2018)*

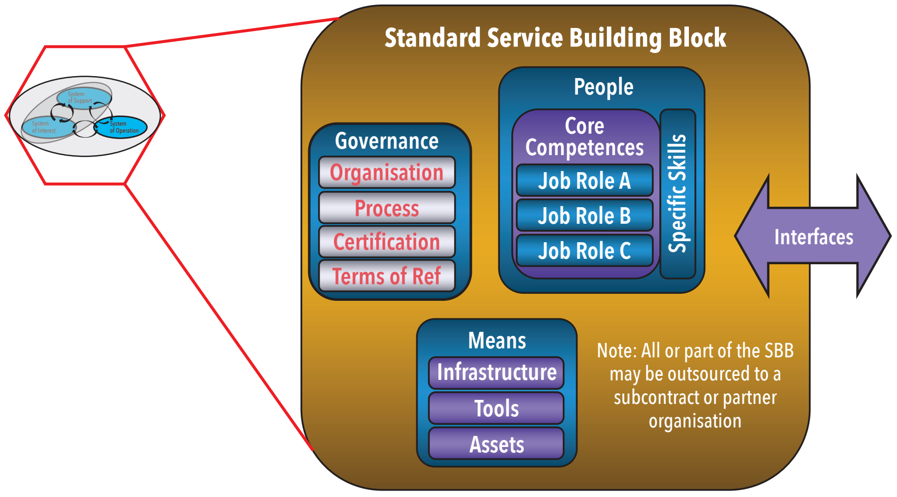


Figure 5. *Common product-service building block elements*

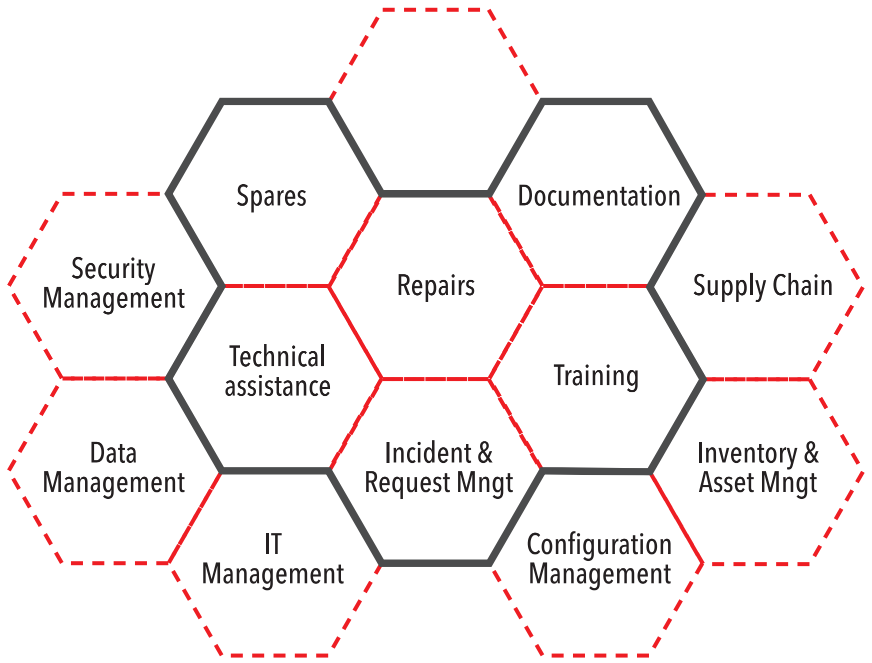


Figure 6. *A composable service-system and service capacities example (Service BoK AFIS 2018)*

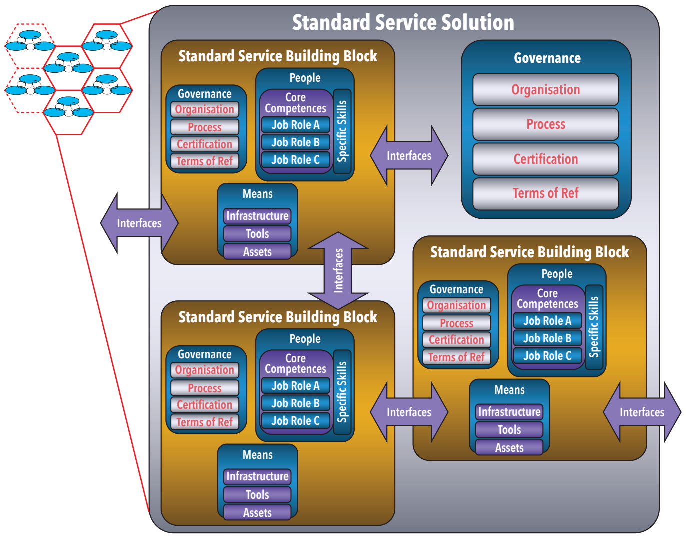


Figure 7. Service-System composition element including a specific governance element

obtain this by setting-up the service-system as a *dynamic product line*, which simultaneously embeds different actionable product-service configurations (different product-service building blocks compositions). The resulting product-service can then support multi-modal operation so the product-service can work in different configurations to provide the same service. Means, personnel, and functionality *dynamically allocate* to the service-system components (either by auto-configuration or via human action) according to the consistency rules mentioned above.

Service-systems depend largely on an agreed common purpose for its building blocks to work together towards the collective objective of providing an expected service set to a customer. Composed service-systems explicitly address challenges relating to authority, funding, and leadership, typically arising in systems-of-systems, through configuring a governance element for the overall service-system, as depicted in Figure 7.

2.3. Product-Service Product Line Strategy

The highest-level component of the proposed framework concerns the product and service business alignment to define the PSPL business strategy within an organization. Those applying the framework should instantiate this component first as it helps define the following elements.

2.3.1. PSPL Value, Pricing Models, and Pricing Metrics. One key aspect to consider when setting up a PSPL is the variability induced by the business dimension. The service **value** and its associated pricing model and pricing metrics are important elements since they are one of the more perceptible parts of the overall service experience offered to a customer. An inspiring example of service value enabled by digital technologies is the one proposed by the Brazilian advertising agency Mood, who teamed up with their customer Huggies to improve the customer experience using 3D printing technology. In this example (Figure 8), through using new technology, a blind pregnant woman experienced what so many other women cherish and enjoy: perceiving the baby to be born for the first time.

A 3D printer used the 3D data from a regular ultrasound scanner to print the results so the future mother could come into contact with her baby. The top of the 3D print even read in Braille “I am your son.” This product-service has remarkably fully integrated into the patient’s journey; the patient’s experience measures the value of the delivered service. In this example, digital technology really acts as an enabler to enhance the patient’s experience, even if there is a tangible outcome through the 3D print. The woman in Figure 8 is not emotionally charged by 3D printing; she is overjoyed at the experience 3D printing has enabled.

In the current business trend to move towards service-oriented business models, different possible pricing models and pricing metrics exist: subscription-based, pay-as-you-go, one-time upfront fee, deployment option-based, one-time fee for a preconfigured solution, and value sharing. Amongst the pricing metrics associated with these pricing models, we can cite the following examples: Messages or amount of cellular data, feature set, devices connected to a platform, base annual subscription, number of users, and property rights (class member).

Pricing models and metrics should entice the customer; they are a crucial part of a product-service offering. A formative example of a pricing metric supported by digital technologies is the “pay-per-laugh” pricing practiced at the Teatre Neu in Barcelona, Spain (La Réclame 2014). To implement this pricing model, sensors placed in the back of auditorium seats facing theatregoers detected smiles and laughs on the faces of every patron. They based the ticket price paid by a given patron on the number of smiles and laughs measured during the show for that individual, directly linking their price paid to their user experience.

2.3.2. Commitment Type. The second dimension of the PSPL business model is the commitment type between product-service providers and their customers. Indeed, various commitment levels can propose different services, such as quotation on request, warranty, availability, capability, or other outcome-based commitments. Here again, digital technologies can play a major role in helping an organization extend the commitment level it proposes to its customers and enhance its product-services with innovative features.

2.3.3. Configuring the Product Service Business Strategy. The service type, the commitment level, the pricing models, and their associated pricing metrics are all part of the PSPL variability domain. These elements, formalized as features in the PSPL feature model, configure a product-service matching a given customer’s needs.

As in typical feature-based approaches, the PSPL feature model also reduces the possible number of service, commitment, pricing model, and pricing metric combinations. This occurs by declaring the permissible combinations among the features (through exclusion or inclusion constraints) and by defining *standard*

Figure 8. A successful use of digital technologies (Adweek 2015)

APRIL 2021
VOLUME 24/ ISSUE 1

Table 2: Example of a service catalogue meta data

Reference	Commitment	Business
Name	Service Performance Level	Type of price
Reference	Service hours	Acceptance criteria
Description	SLA	Cost (internal view)
Interest (customer point of view) – Value Proposition	KPI & metrics	Price (customer view)
Business Domain	Quality of Service	Invoicing type (lump sum / on unit)
Type of Customer (Internal, external, both)	Service Level Monitoring (HUMS)	Delivery status (under design, provisioned, closed)
Target Customer segment	Risk level	Service provider
Link with technological product functions (if needed)	…	Capability Management (potential dimensioning)
Quality Management System Reference		Escalation process
…		…

PSPL configurations (pre-defined PSPL configurations including undecided feature choices). Many organizations transcribe standard configurations and the constraints among the PSPL features into a *service catalogue*. Table 2 shows an example of the metadata to structure a product-service catalogue. For a given product-service, a given PSPL configuration would generate the data to appear in the cells under the three columns in Table 2 (a set of selected PSPL features).

CONCLUSION

Digitizing the value chain brings new business opportunities to organizations wishing to adopt a service-oriented approach. Indeed, digital technologies allow a company to move towards outcome-based commitments, pricing, and contracts with its customers. When transforming a product offering into a product-service offering through digitization, it is nevertheless crucial to consider this transformation concerns more than a new culture or using new technology. It requires an alignment with the company strategy and organization (and not the other way around). Conducting appropriate change management is necessary to operate this alignment.

This paper proposed a conceptual framework to define a high-level strategy to implement a product-service offer in an organization. Its distinctive aspects include:

- The *product line of product-services* concept itself (Product-Service Product Line)
- The elements defining the PSPL business model (pricing model, pricing metrics, and commitment types)
- A product-services typology
- An extension of product line engineering methods for architecting the PSPL (notably, a specific *product-service building block* type to support composable architectures and a feature model including service-related, socio-technical features)

The PSPL feature model is a central part of the proposed approach since it allows configuring different PSPL business model elements, configuring the composable service-system service building blocks, and defining *standard PSPL configurations*. Configuring the business model elements for a given product-service is a crucial step. Whilst the proposed framework does not provide an infallible technique to validate the business model effectiveness, it does support a "test-fail-and-learn-fast" approach by providing mechanisms to reconfigure a new business model that an organisation can test quickly. ■

REFERENCES

- Adweek. 2015. "Huggies Helped This Blind Mom See Her Pregnancy Ultrasound by 3-D Printing the Baby." https://www.adweek.com/creativity/huggies-3-d-printed-fetus-so-blind-mom-could-see-her-ultrasound-164511/
- AFIS. 2013. Systems Product Line Engineering. Toulouse, FR: Cépaduès.
- Baines, T.S., H. Lightfoot, E. Steve, A. Neely, R. Greenough, J. Peppard, R. Roy, E. Shehab, A. Braganza, A. Tiwari, J. Alcock, J. Angus, M. Bastl, A. Cousens, P. Irving, M. Johnson, J. Kingston, H. Lockett, V. Martinez, P. Michele, D. Tranfield, I. Walton, and H. Wilson. 2007. "State-of-the-Art in Product Service-Systems." Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 221 (10): 15431552. DOI: 10.1243/09544054JEM858.
- Beuche, D. 2008. "Modeling and Building Software Product Lines with Pure: Variants." Paper presented at the 15th International Software Product Line Conference, Limerick, IE, 08-12 Sept.
- Chalé Góngora, H.G., and F. Greugny. 2017. "Where the Big Bucks (Will) Come From—Implementing Product Line Engineering for Railway Rolling Stock." Paper presented at the 27th Annual International Symposium of INCOSE, Adelaide, AU, 15-20 July.
- Cook, D.P., G. Chon-Huat, and H.C. Chen. 1999. "Service Typologies: A State-of-the-Art Survey." International Journal of Production and Operations Management 8 (3): 318-338.
- Ducq, Y., D. Chen, and T. Alix. 2012. "Principles of Servitization and Definition of an Architecture for Model Driven Service System Engineering." Paper presented at the International IFIP Working Conference on Enterprise Interoperability, Harbin, CN, 6-7 September.

REFERENCES *(continued)*

- INCOSE. 2015. “Definition of Systems of Systems.” In INCOSE Systems Engineering Handbook: A Guide for System Lifecycle Processes and Activities, 4th Edition, edited by D. D. Walden, G. J. Roedler, K. J. Forsber, R. D. Hamelin and T. M. Shortell, 8-10. INCOSE, San Diego, US-CA: Wiley.
- ———. 2015. “Application of Systems Engineering for Services.” In INCOSE Systems Engineering Handbook: A Guide for System Lifecycle Processes and Activities, 4th Edition, edited by D. D. Walden, G. J. Roedler, K. J. Forsber, R. D. Hamelin and T. M. Shortell, 171-175. INCOSE, San Diego, US-CA: Wiley.
- ———. 2019. Feature-based Systems and Software Product Line Engineering: A Primer. Product Line Engineering International Working Group. San Diego, US-CA: INCOSE.
- Kumar, A., D. S. Lokku, N. R. Zope, and J. K. Reddypogu. 2017. “Value Based Architecture for Digital Product-Service Systems.” Paper presented at the 27th Annual International Symposium of INCOSE, Adelaide, AU, 15-20 July.
- La Réclame. 2014. “Ne Payez Votre Place de Théâtre Que Si Vous Riez.” http://lareclame.fr/105885-theatre-pay-per-laugh.
- Polaine, A., L. Løvlie, and B. Reason. 2014. Service Design—From Insight to Implementation. New York, US-NY: Rosenfeld Media LLC.
- Queinnec, G., and Tan. 2018. “Product-as-a-Service Business Models – Preliminary study.” White Paper, Movin’On Labs.
- Service BoK AFIS. 2018. “System Engineering and Service Engineering.” https://afis.sharepoint.com.
- SEBoK. 2019. “Applications of Systems Engineering—Service Systems Engineering.” https://sebokwiki.org/w/index.php?title=-Service_Systems_Engineering&oldid=54449.
- Schnürmacher, C., H. Haygazun, and R. Stark. 2015. “Providing Product-Service-Systems—The long way from a Product OEM Towards an Original Solution Provider (OSP).” Paper presented at the Seventh Industrial PSS Conference, Saint-Etienne, FR, 21-22 May.
- Tan, A. R., and T. C. McAloone. 2010. “Service-Oriented Product Development Strategies: Product-Service-Systems (PSS) Development.” PhD diss., Technical University of Denmark (Kongens Lyngby, DK).
- Wild, J.P., J. Jupp, W. Kerley, W. Eckert, and P. J. Clarkson. 2007. “Towards a Framework for Profiling of Products and Services,” Paper presented at the Fifth International Conference on Manufacturing Research (ICMR), Leicester, GB, 11-13 September.

ABOUT THE AUTHORS

Hugo Guillermo Chalé Góngora, PhD, CPRE, is the product line engineering director for the Thales Group. Formerly the head of requirements engineering, train system functional architecture, and MBSE for Alstom’s rolling stock division, he has over 16 years of experience in systems engineering in the energy, infrastructure, automotive, and railway industries. His topics of interest include formal methods, architecture description languages, safety-critical systems, and autonomous systems. He holds an engineering degree in mechanical-electrical engineering, an MS in energy conversion and internal combustion engines, and a PhD on thermal and energy systems. Guillermo is founder and chair of the PLE International Working Group of INCOSE.

Pierre-Olivier is head of innovative services and digital champion services at Thales. He is a modeling expert in costs, risks, and services. Pierre-Olivier has an Master of Science in Telecommunication and Networks and a Master of Economics.

Danilo Beuche is CEO at Pure-Systems GmbH and an honorary professor for information systems at Leipzig University. His experience is in development of complex (embedded) systems, object-oriented development, change management, and requirements management. Danilo’s main focus is support of product line engineering activities. He has a diploma and a Doctor of Engineering in computer science.

APRIL 2021
VOLUME 24/ ISSUE 1

Key Issues of Organizational Structure and Processes with Feature-based Product Line Engineering

William J. Bolander, william.bolander@methodpark.de; and **Paul C. Clements,** pclements@biglever.com

ABSTRACT
Feature-Based Product Line Engineering (PLE) is a well-known approach for efficiently engineering product lines, which numerous case studies have shown to yield substantial benefits in cost, quality, and time to market. This article presents an approach for a necessary ingredient of successful PLE: Handling the temporal dimension, which is concerned with managing artifacts as they change and evolve. The approach relies on a foundation of proven traditional change control techniques but shows how they apply in the context of Feature-Based PLE.

INTRODUCTION

Product Line Engineering (PLE) has long been known for delivering significant improvements in time to market, quality, and cost of systems. Feature-based Product Line Engineering is a well-defined, repeatable, automation-centric PLE specialization that is now delivering even greater improvements throughout the most challenging engineering industries (INCOSE 2019). Feature-Based PLE centers around the factory metaphor. In a PLE factory, a *configurator* operates on *shared asset supersets* containing variation *points* to produce instances of those assets corresponding to a product (a product line member) described by a *bill-of-features*, which is a selection set made from a *feature catalog*.

That now-familiar narrative and its illustrating diagram (Figure 1) cover an up-and-running Feature-based PLE factory workflow. This workflow is important to convey how Feature-Based PLE works, so we can understand how it differs from earlier PLE forms and, for that matter, from system and software development forms that do not employ PLE principles at all.

But an organization intent on setting up one or more PLE factories so they can reap the benefits for their product line or product lines needs a different viewpoint—one

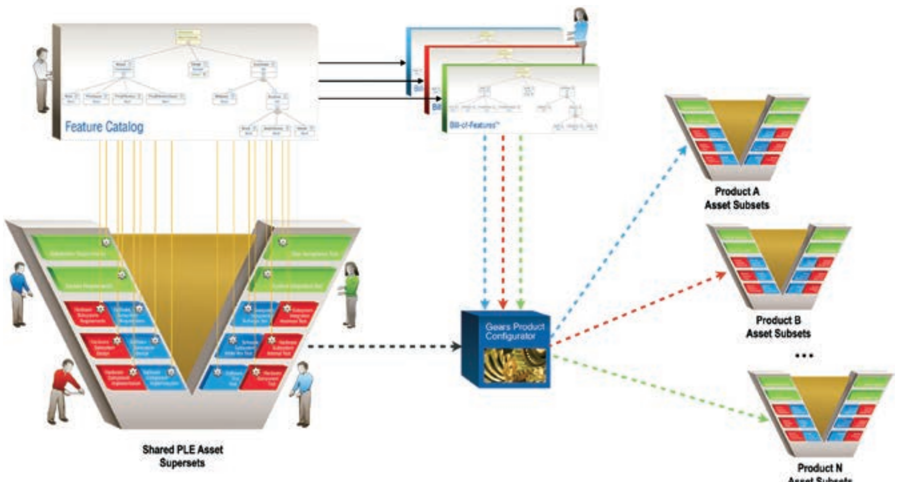


Figure 1. The Feature-Based PLE factory

focusing on the people involved and what they do to keep the factory operational on a day-to-day basis. In other words, the organization needs to know the roles and processes to add to the workflow.

Through many PLE applications in complex systems development, we have learned the needed organizational structures and the required roles to enable a successful transformation. This article describes an organizational structure for Feature-Based PLE based on the factory concept. It introduces the few roles that have no analog in other development disciplines; they are new to Feature-Based PLE. It also describes how traditional systems engineering roles carry out traditional systems engineering tasks, but with slight PLE-inspired extensions. Finally, we will explain why these changes are necessary.

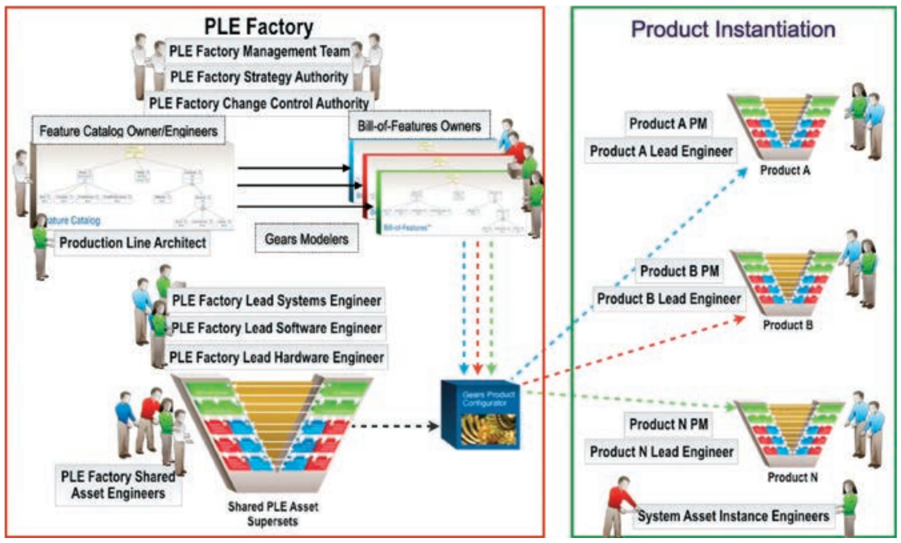

PLE Factory
PLE Factory Management Team
PLE Factory Strategy Authority
PLE Factory Change Control Authority
Feature Catalog Owner/Engineers
Bill-of-Features Owners
Production Line Architect
Gears Modelers
PLE Factory Lead Systems Engineer
PLE Factory Lead Software Engineer
PLE Factory Lead Hardware Engineer
PLE Factory Shared Asset Engineers
Shared PLE Asset Supersets
Product Instantiation
Product A PM
Product A Lead Engineer
Product A
Product B PM
Product B Lead Engineer
Product B
Product N PM
Product N Lead Engineer
Product N
System Asset Instance Engineers

Figure 2. Roles in the Feature-Based PLE factory

WHY MUST THE ORGANIZATIONAL STRUCTURE CHANGE?

Organizations adopting Feature-Based PLE often start from an organizational structure focusing on products rather than the product line. At the extreme, this entails one separately staffed project for each product in the product line. Each project executes the entire system engineering process to produce their product. The chief engineer for the product controls exactly what features and changes the product takes on without considering the others. When one product team finds an error, it is fairly easy for them to find the source and fix it; the other teams may not even know whether that defect is in their products.

In this setting, suppose a change request comes to the organization from a customer (or customer base segment). It might be a request to fix a defect, or a request for a new feature some (if not all) of the products need, or a request for an improvement. Every project affected by the change request will carry out the work on their respective products. Even if each team does the work as efficiently as humanly possible, as soon as we take a product line (not just a single product) perspective it becomes apparent the organization overall is not very efficient at all.

Feature-based PLE relegates this duplicative work to automation—the PLE factory configurator. All work happens inside the PLE factory—to the shared assets, the feature catalog, or the bills-of-features. Thus, no piece of work can occur more than once.

WHAT ORGANIZATIONAL STRUCTURE DOES FEATURE-BASED PLE NEED?

From the roles and responsibilities perspective, moving to Feature-Based PLE primarily means staffing the PLE factory with the engineering resources formerly allocated to individual product teams.

Figure 2 shows the organizational structure for a Feature-Based PLE factory. Any traditionally organized systems engineering project contains most of the roles shown:

- **PLE Factory Management:** A key insight in understanding Feature-Based PLE is the PLE factory is analogous to an engineering development project in a traditional (non-PLE) engineering context. Rather than engineering and developing a product, however, the goal is engineering and developing an automated capability to develop products. Where a traditional engineering project would result in an engineering artifact set supporting and reifying a system, the Feature-Based PLE Factory project results in shared asset superset, a feature catalog, and a set of bills-of-features that support and reify all the products in the product line. To re-state the point succinctly: In Feature-Based PLE, the PLE factory is a project. This project needs management support, oversight, and governance, just like any other. PLE Factory Management is responsible for ensuring the factory works on schedule and within budget, employs trained people in the right roles, and delivers high-quality results to its clients.
- **PLE Factory Strategy Authority and PLE Change Control Authority:** These are the governance bodies for PLE factory change management. Change requests can originate from any role in the organization. Typically, they originate from a product manager or product lead engineer, satisfying a customer need. They could also originate from business leadership (not shown in Figure 2) responding to a technology or product roadmap setting the PLE factory's over-arching direction for the future. Change requests causing the factory to do something new or different than what it currently does arrive at the PLE factory strategy authority. Change requests causing the factory to do what it currently does—a request to fix a defect—arrive at the PLE factory change control authority, which adjudicates the change and parcels it out to the affected individual roles for enactment.
- **Lead Engineers:** Various disciplines ensure the solution is technically sound.
- **PLE Factory Shared Asset Engineers:** Responsible for the project's specific engineering artifacts (requirements, code, tests, documents, models, parts lists, and so forth) just as they would be in a traditional project. Except here, the artifacts are supersets with feature-based variation points.
- **Product Managers and Product Lead Engineers:** Responsible, on the Product Instantiation side of Figure 2, to individual customers, and serve as the go-between (programmatic and technical, respectively) between a customer and the PLE factory. They are responsible for delivering feedback to the development organization, and for delivering solutions built by the organization to the customer.

This means people who learned these roles in a traditional engineering environment can be effective in a Feature-Based PLE environment with little or no additional training.

In Figure 2, the new roles Feature-Based PLE adds are across the middle of the PLE factory side.

APRIL 2021 VOLUME 24/ ISSUE 1

- Feature-Based PLE may represent systems of systems products as a product line of product lines. The **Production Line Architect** is responsible for designing this structure.
- The **Feature Catalog Owner** is responsible for a PLE factory's overall feature catalog content, fidelity, and quality. In Feature-Based PLE, a feature catalog may decompose into separate pieces; the feature catalog owner crafts standards and style guides to ensure the feature catalog style and content overall is consistent, any features are appropriately modelled and clearly explained, and represent useful differentiation abstractions not heavily tilted towards one particular shared asset type.
- **Feature Catalog Engineers** create and evolve the feature models for individual product line areas.
- A **Bill-of-Features Owner** is responsible for the feature-based description of a product in the product line

WHAT ARE THE KEY PROCESSES ISSUES FOR FEATURE-BASED PLE?

The organizational structure for Feature-Based PLE exists to facilitate the organization members executing the Feature-Based PLE processes. We describe some key process issues associated with Feature-Based PLE below.

Dealing with Product-Unique Content

If a product has unique functionality no other product requires, some organizations trying to practice PLE principles might let the product team for the affected product provide that unique functionality. However, our experience has shown the factory should provide all functionality and *not* delegate it to the team for the individual product needing it. The product can select it by choosing it as a feature in its bill-of-features. All other products can leave it unselected in their respective bills-of-features unless they also desire it, since it is now available as a feature across the product line.

If product teams can create their own content, it is extremely likely product teams will independently create redundant but different solutions to the same problems, which is exactly the situation PLE tries to avoid. This redundancy is also a signal that, whereas product teams may think they have special content, a good idea in one product is often a good idea for other products as well. When recognizing redundancy after the fact, agreement to move to a common solution is nearly impossible. There are all kinds of good reasons. The product may already have validated the solution. There would be an additional risk to the program by making a change to the common solution. It may even be too late—the product is near or already in production. Allowing this to happen is to step on the slippery slope back to product silos, and the death knell for the product line.

Defects Found by a Product Team

A key Feature-Based PLE tenet is when a defect is found, it is fixed inside the factory, not in the product discovering it. That way, every product benefits from the defect fix, even those who may not yet have discovered the defect. Fixing it inside the factory means the shared assets change, not the product-specific instances.

Unless the defect fix occurs inside the factory, the product teams will develop redundant but different solutions, and once validated or even shown to customers, convergence back to a common solution is nearly impossible. In any case, when one product team finds an issue, it is likely the defect also lurks in another product.

We don't want the inefficiency of multiple product teams making the same changes to engineering assets. Rather, we want the PLE factory to fix the defects once, in the shared PLE assets, and to propagate the changes everywhere.

Work Performed by the Product Tea

Whereas Product Teams are no longer responsible for development under the Feature-Based PLE paradigm, they still play a vital role, including:

- The product team works with the bill-of-features owner for the product, to make and capture the correct selections for all the features made available in the feature catalog.
- The product team will verify and validate the product produced by the PLE factory for their application.
- The product team will calibrate or tune the product to meet their customer's requirements, oversee delivery and deployment, and gather customer feedback.

PLE and Agile

The PLE factory is a project view described in the section on PLE factory management enabling Feature-Based PLE and agile to work seamlessly together. The core concept is to treat the PLE factory development and operation as a "project" managed using Agile (Gregg et al. 2020). In the Scaled Agile Framework (SAFe)—an Agile methodology for large projects (Scaled Agile Inc. 2020)—a value stream is the activity set undertaken to deliver value to a customer. Developing and operating the PLE factory is exactly that—it is developing the capability to deliver value to the customer. So, a program's value streams should align to its PLE factories (a program may have more than one). An agile release train receiving its work assignments from the PLE factory strategy authority can plan and coordinate the work in a PLE factory. In short, PLE factories can optimize using agile practices, thus achieving the best of both worlds.

WHAT TRANSITION ISSUES MUST WE CONSIDER?

Overall, our advice is: Be careful not to underestimate the cultural shift. PLE is a big transformation. The organization must move from absolute product team control to joint product team decisions. This transformation may require changes to product team management incentivization.

Incremental Transition is Key

Incremental, not big-bang, transition is the recommended approach for moving your organization to the preferred Feature-Based PLE structure (Gregg et al. 2020). "Incremental" may mean moving some, but not all, product line members under the PLE umbrella immediately and moving the rest over time. It might mean initially converting some, but not all, of your engineering artifacts to shared asset supersets with variation points and converting the rest over time. It might mean using a PLE factory to produce certain subsystems of your products initially and adding the remaining subsystems over time. Or it might mean a combination of these options. In any case, your PLE Factory will require a feature catalog (and a feature catalog owner), and bills-of-features to define the products (or the subsystems) the PLE factory will produce. As the PLE factory gains traction and capability, move people away from their work supporting individual products and into the PLE factory, where their work shifts to the entire product line. Each incremental step in growing the factory can bring more people in until the individual products' development resources have migrated into the PLE factory.

Importance of Adopting the Organizational Structure Right Away

Some organizations might allow people to remain attached to project teams, but tell them their work should support the entire product line. This, in our experience, usually does not work and organizations should avoid it. Here is what an experience report from one of the most successful Feature-Based PLE applications on record says in that regard (emphasis added):

"[We] first tried to instill the product line approach throughout the… program by senior management fiat. Despite sincere management intent, including a number of intense meetings in which the technical leaders were asked one by one to say how they were going to support the product line approach, the paradigm shift was never completely fulfilled. People doing the day-to-day work were allowed to drift back into configuration-centric activities and mindsets. *It was only after re-organizations occurred that re-structured the customer-specific teams (replacing them with smaller, leaner product delivery teams) and moved the resources into product-line-wide shared asset groups did the transition finally find traction*. [We] did a good job launching the product line, but the institutionalizing was not fully successful until after reorganization. This manifested itself during a delivery cycle for one of the [products] in which work was done under the new approach but under the old organizational structure. The delivery was eventually successful, but not without an alarming amount of re-work" (Gregg et al. 2014).

Lead Engineer and Product Team Buy-in

Buy-in for the PLE transformation by the lead engineers and product team leadership is key. These stakeholders need to understand the benefits to the company, and their customers, for the PLE factory to balance work for the company's different products. They must help manage what work the factory does for them based, not just on their own product's needs, but all the company's products. Everyone naturally wants what is best for them; it can be very difficult to give up something you want so a higher priority or higher revenue generator gets what they need.

The Power of the Purse

Resource allocation can play a large role in staffing product teams so they behave as desired. If the product teams have the capability to design and create unique solutions, they will! Therefore, the PLE factory resources must increase, while the product team's resources need to shrink to be the right size to perform only what they are responsible for. In one PLE adoption effort, a product team undertook some development work the PLE factory should have handled. The PLE manager "rewarded" the product team by moving one of their programmers from the product team into the PLE factory. It sent a powerful message.

Apply a Three-Tiered Adoption Strategy

We find three organization parts must engage to ensure successful adoption. An adoption strategy neglecting one or more of them is likely to fail. These so-called organizational tiers employ people who have different skillsets, motivations, organizational responsibilities, and therefore different perspectives when it comes to adopting a new methodology. Recognizing and addressing these three tiers is the first successful adoption strategy ingredient (Gregg et al. 2020). Figure 3 illustrates the three tiers.

The base technology tier places and maintains the tool and technology environment to operate the PLE factory. This tier is about installing tools and making them work together: The PLE factory configurator (such as BigLever's Gears), engineering tools with which the configurator integrates, CM tools, workflow tools, and whatever other tools are part of the organization's tool ecosystem needed to take products through their entire lifecycle. The tools must work together and perform well in the organization's IT and network environments, and users must have access to them. Think of this as the fully functional factory (in the conventional manufacturing sense) but without any people inside to run the machines.

The middle technical organization management tier focuses on the people, roles, and processes operating the PLE factory, and it contains the roles described in Figure 2. In combination with the technology layer, this layer provides a fully operational Feature-Based PLE factory producing the product asset instances for all products in a product line portfolio. To continue the analogy, this tier is about the people who walk into the technology tier's factory, trained and ready to operate the machines.

The top business organization management tier focuses on the people, roles, and processes using and leveraging the Feature-Based PLE factory to achieve the enterprise's business objectives. But it also focuses on the required processes for enterprise leadership to establish the PLE factory, oversee the adoption, remove obstacles to organizational change, and provide the necessary support for the PLE factory during its operation. Using the analogy to a conventional factory, the business organization tier provides guidance and support for the executive leadership. Think of this group as working in the office high-rise overlooking the factory.

All tiers aim to develop, operate, and maintain a PLE factory to build and deliver products.

The obvious tendency with a transition to Feature-Based PLE practice is to think bottom-up. That is, begin with the technology tier, and then start on the middle technical organization management tier, in sequence, after fully establishing the technology.

While this transition style is tempting, we have found it is more effective to address the tiers incrementally and in parallel. Imagine the incremental transition effort growing from left-to-right rather than bottom-up. Initially, the technology tier is incrementally addressed. Once it establishes sufficient capability, then activities can begin in the middle tier. The top tier activities can commence immediately, motivating and driving activities in the other two tiers.

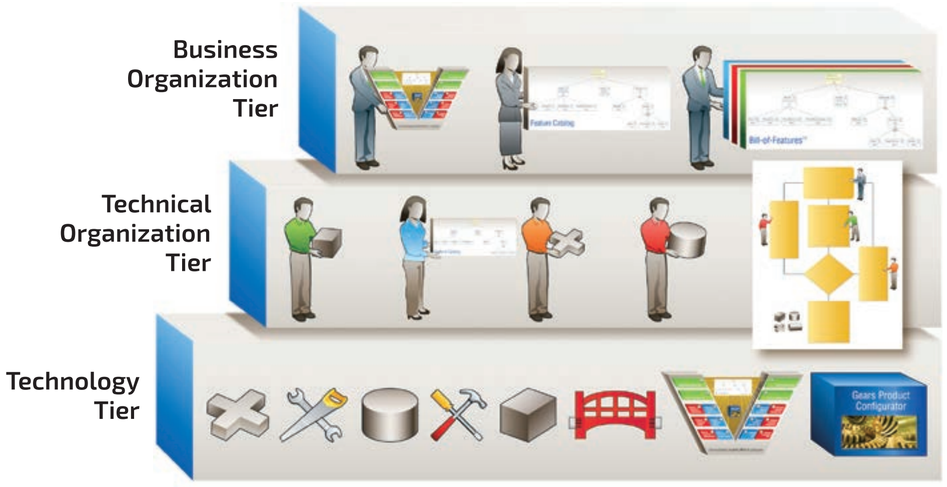

Business Organization Tier
Technical Organization Tier
Technology Tier
Feature Catalog
Bill-of-Features
Gears Product Configurator

Figure 3. Three Tiers of a Feature-Based PLE Organization

SUMMARY

Feature-Based PLE is producing unprecedented improvements in engineering productivity, product cost and quality, and return on investment. Compared to other engineering disciplines, it requires more collaboration and

APRIL 2021
VOLUME 24/ ISSUE 1

organizational support to be effective.

This article highlighted the organizational structure, key processes, and important transition issues associated with Feature-Based PLE. It may seem like these issues, taken together, present a barrier to entry. However, this article has aimed to show no issues present an insurmountable challenge. On the contrary, there is a plethora of practical experience codified and formulated that can allow any organization to deal with these issues quickly and effectively. ■

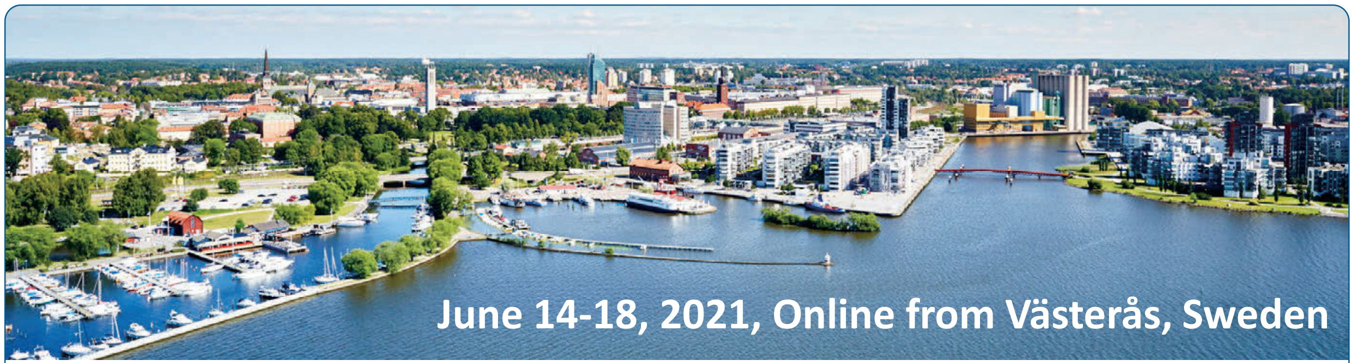
REFERENCES

- Gregg, S., D. Hartley, M. McAfee, R. Pitz, J. Teaff, and P. Clements. 2020. "Patterns for Success in the Adoption and Execution of Feature-Based Product Line Engineering: A Report from Practitioners." Paper presented at the 30th Annual International Symposium of INCOSE, virtual event, 20-22 July.
- Gregg, S., R. Scharadin, and P. Clements. 2016. "The Best of Both Worlds: Agile Development Meets Product Line Engineering at Lockheed Martin." Paper presented at the 26th Annual International Symposium of INCSOE, Edinburgh, GB-SC, 18-21 July.
- Gregg, S., R. Scharadin, E. LeGore, and P. Clements. 2014. "Lessons from AEGIS: Organizational and Governance Aspects of a Major Product Line in a Multi-Program Environment." Paper present at the 18th International Software Product Line Conference, Florence, IT, 15-19 September.
- International Council on Systems Engineering (INCOSE). 2019. "Feature-based Systems and Software Product Line Engineering: A Primer." Technical Product INCOSE. https://connect.incose.org/Pages/Product-Details.aspx?ProductCode=PLE_Primer_2019.
- Scaled Agile Inc. 2020 "SAFe 5 for Lean Enterprises." https://www.scaledagileframework.com

ABOUT THE AUTHORS

William Bolander is a principal consultant at Method Park America where, for the last 3 years, he has focused on engineering process improvement and Automotive SPICE. Before this, Bill spent 5 years as the Global Automotive Solution Executive with IBM's Rational team, where he strove to help the industry adopt smarter product development tools and processes. Prior to IBM, Bill spent 32 years at General Motors. This started at Saturn Powertrain, being part of the team that developed Saturn's engine control system. Bill then transferred to GM Powertrain to manage the Algorithm Development Group. In 2000, Bill was promoted to GM Technical Fellow responsible for GM Powertrains Global Control Engineering Processes. In this role, he led the Controls Engineering Process Group responsible for process improvement activities, including Product Line Engineering and CMMi. In 2009, Bill's role expanded to include all of GM's Electrical, Controls and Software development, for both Powertrain and Vehicle Product Development. Bill holds 16 US patents for automotive related innovations, including traction control system, coolant loss protection, clutch to clutch transmissions and several in the field of combustion knock control. Bill's contributions to GM's technology have earned him four "Boss" Kettering Awards, this award is GM's highest recognition for engineering invention. Bill was the first winner of the $500,000, Lemelson-MIT Prize, the USA's largest single prize for invention and innovation.

Paul Clements is the vice president of Customer Success at BigLever Software, Inc., where he works to help organizations adopt feature-based systems and software product line engineering. Prior to this, he was a senior member of the technical staff at Carnegie Mellon University's Software Engineering Institute, where for 17 years he worked leading or co-leading projects in software product line engineering and software architecture documentation and analysis. Prior to the SEI, Paul was a computer scientist with the US Naval Research Laboratory in Washington, DC. Paul has both a BS and MS in computer science from the University of North Carolina at Chapel Hill and a PhD in computer science from the University of Texas at Austin.



16th Annual System of Systems Engineering Conference

Conference theme: Autonomous Cyber-Physical Systems of Systems

http://sosengineering.org/

Highlights

IEEE System, Man, and Cybernetics Society (SMC), in cooperation with the International Council on Systems Engineering (INCOSE), is organizing the 16th International Conference on System of Systems Engineering (SoSE) to be held online, June 14-18 2021.

Systems of systems have vast ramifications in numerous engineering fields such as control, computing, communication, information technology and in applications such as manufacturing, defense, national security, aerospace, aeronautics, energy, environment, healthcare, and transportation. Papers on theories, methodologies, and applications of System of Systems Engineering in science, technology, industry, and education are welcome.

Papers should be five to six pages in length, in standard two-column IEEE Conference Proceedings format. Detailed instructions for paper submission and format can be found on the conference web site.

Invitations will be made to the authors of the best papers to submit an extended version of papers to IEEE Systems Journal and Journal of Enterprise Transformation.

Key dates for submissions

Special session proposals:	**Nov. 29, 2020**
Notification of special session:	**Dec. 13, 2020**
Technical papers & panels:	**Feb. 14, 2021**
Notification, papers & panels:	**March 21 , 2021**
Final manuscript:	**April 18, 2021**

Organizers

General Chair
Jakob Axelsson, Mälardalen University and RISE

Founding Chair
Mo Jamshidi, University of Texas San Antonio

Program Chairs
Martin Törngren, KTH, Sweden
Gerrit Muller, University of South-East Norway

Local Chair
Malin Rosqvist, RISE and Mälardalen University

Industry Liaison
Erik Herzog, Saab and INCOSE Sweden

Publication Chair
Patrick Benavidez, Univ. of Texas San Antonio

INCOSE and US industry liaison
Garry Roedler, INCOSE

Contact

For general and technical program inquiries about the conference, please contact the conference General Chair, Jakob Axelsson (jakob.axelsson@mdh.se).

Academic sponsors

MÄLARDALEN UNIVERSITY SWEDEN

RI. SE Research Institutes of Sweden

UTSA

Technical co-sponsors

IEEE SMC

INCOSE

31st Annual **INCOSE** international symposium | Virtual Event July 17 to 22, 2021

Preliminary Technical Program

The Premier International Systems Engineering Conference Accelerating through Adversity

6 Days, 5 Tracks, 4 Keynotes, 95+ Presentations, Panels, Tutorials and More!

95 + Papers, Presentations on Systems Engineering
Monday - Thursday

4 Inspiring Keynote Speakers

19 Countries Represented
Australia, Austria, Canada, Colombia, France, Germany, India, Israel, Italy, Japan, Lithuania, Netherlands, New Zealand, Norway, Saudi Arabia, South Africa, Sweden, Turkey, United Kingdom, United States

24 Application Domains
Top Domains
Defense, Enterprise SE, Aerospace, Academia, Automotive, Social/Sociotechnical and Economic Systems, Industry 4.0 & Society 5.0, Biomed/Healthcare/Social Services, Infrastructure, Oil and Gas

39 Topics Represented
Top Topics
System Architecture/Design Definition, MBSE, Systems Thinking, Needs and Requirements Definition, Modeling/Simulation/Analysis, Processes, Systems of Systems, Resilience, decision Analysis and/or Decision Management, Complexity, Product Line Engineering, Project Planning, Project Assessment, and/or Project Control, Technical Leadership, Verification/Validation, System Security, Measurement and Metrics, Artificial Intelligence, Machine Learning

11 Panels
Including Topics Like These Discussed With Global Leaders in Systems Engineering, Academia, Defense, Industry 4.0 & Society 5.0, Enterprise SE, Aerospace

12 Tutorials
Including Topics Like These Discussed With Global Leaders in Systems Engineering, Enterprise SE, Defense, Aerospace, Automotive, Biomed/Healthcare/Social Services, Industry 4.0 & Society 5.0

SPONSOR INCOSE IS 2021!

1 **VISIBILITY**
Unique brand of recognition and visibility for your organization

2 **PRACTICE**
Access to the latest thinking relevant to the practice of Systems Engineering

3 **SPOTLIGHT**
Put a spotlight on your organization's competency in Systems Engineering

4 **ASSOCIATION**
Be associated with the highest culture of professionalism and innovation

5 **SUPPORT**
Demonstrate organizational support to INCOSE's mission

6 **RELATIONSHIPS**
Develop sustainable business relationships

Lots of possibilities to interact with systems engineering communities

more information at www.incose.org/symp2021