Product Line Engineering on the Right Side of the "V"

Susan P. Gregg
Denise M. Albert
Lockheed Martin
Moorestown, New Jersey 08057 USA
susan.p.gregg@lmco.com
denise.albert@lmco.com

Paul Clements
BigLever Software
Austin, Texas 78730 USA
pclements@biglever.com

ABSTRACT

Product line engineering (PLE) is well-known for the savings it brings to organizations. This paper shows how a very large, in-service systems and software product line is achieving PLE-based savings in their verification and validation phase of development. The paper addresses how to achieve the sharing across product variants while the products being tested are evolving over time. Additionally, we will give a pragmatic set of decision criteria to help answer the longstanding issue in PLE-based testing of whether to test on the domain side or the application (product) side of the product derivation process.

CCS CONCEPTS

• Software and its engineering → Software product lines;

KEYWORDS

Product line engineering, software product lines, feature modeling, feature profiles, bill-of-features, variation points, product portfolio, product configurator, second generation product line engineering, PLE factory, AEGIS Combat System

ACM Reference Format: Susan P. Gregg, Denise M. Albert, Paul C. Cleements. 2017. Product Line Engineering on the Right Side of the "V." *In Proceedings of SPLC '17*, Sevilla, Spain, September 25-29, 2017, 10 pages.

DOI: 10.1145/3106195.3106219

1. INTRODUCTION

This paper tells the story of how a very large systems and software product line is harvesting the benefits of product line engineering on the right-hand side of the engineering "V" model – that is, in the verification and validation activities associated with product deployment.

Product line engineering (PLE) is well-known for the game-changing savings it brings to organizations (for example, [3][14][19][25][22][23]), compared to one-at-a-time development or worse, parallel development. Case studies have, in the past, most often focused on the development activities involved in the *creation* of system artifacts – activities on the left side of the "V". PLE literature certainly acknowledges and touts the potential savings available from the testing side, but case studies focusing on how to gain those benefits in practice are not as plentiful.

This paper shows how a very large, industrial-strength, in-service systems and software product line is achieving PLE-based V&V savings. The product line that is the subject of this paper is the AEGIS Weapons System, a large and complex naval command and control system in wide use in several navies around the world; the developing organization is Lockheed Martin, the world's largest defense contractor, employing 125,000 people worldwide.

The paper confirms that significant savings can be achieved from sharing V&V activities and artifacts across product variants in a product line, but the story of how it does so takes an unexpected twist. How do you manage that sharing when the products you are testing are continuously evolving in response to updated requirements and the need for additional variants? We will show how technical management policies drive V&V-based savings.

Finally, a longstanding issue in PLE-based testing is whether to test on the domain side or the application (product) side of the product derivation process. We will show how Lockheed Martin answers that question pragmatically for AEGIS.

2. AEGIS

The product line being described here is the AEGIS Weapons System, which is the major command-and-control component of the AEGIS Combat System. AEGIS is a highly integrated naval ship combat system in service on over 100 ships in the U.S. Navy and elsewhere. AEGIS cruisers and destroyers constitute the majority of the U.S. surface Navy and will continue to form the core of the U.S. surface fleet for the next several decades. The AEGIS Combat System is capable of simultaneous warfare on many fronts: anti-air, anti-surface, anti-submarine, and strike warfare. AEGIS, or a carefully chosen functional subset, is deployed on some 100 naval vessels in the U.S. Navy, navies of several key U.S. allies across the globe, vessels of the U.S. Coast Guard, and even land-based ballistic missile defense installations. AEGIS is a system that protects assets from airborne attack from aircraft or missiles. It detects airborne threats, plans how to engage them, and launches missiles to intercept and neutralize them (Figure 1).

The mission of AEGIS includes

- self-defense (protecting the host platform from attack)

SPLC '17, September 25-29, 2017, Sevilla, Spain
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5221-5/17/09…$15.00
http://dx.doi.org/10.1145/3106195.3106219

- area air defense (for example, protecting a naval task force that includes the host platform)
- long-range air defense and ballistic missile defense (for example, protecting a geographical area from long-range ballistic missiles) [18].

Figure 1: The Aegis-class destroyer USS Hopper (DDG 70) launches a missile to intercept a short range ballistic missile (U.S. Navy photo/Released).

With more than 40 years of significant investment by the U.S. Navy and its allies, the Aegis Combat System is used globally by six navies. In addition to the U.S., AEGIS is the maritime weapon system of choice for Japan, the Republic of Korea, Australia, Norway, and Spain [16].

Figure 2: Combat Information Center on a U.S. Navy AEGIS cruiser (U.S. Navy photo/Released)

To give an idea of scale and complexity, the AEGIS Weapons System comprises several million lines of source code and several hundred thousand requirements objects.

3. THE "V" MODEL

To illustrate several of our points, we will appeal to the well-known "V" model from software and system engineering [6]:

> *"The V model is a simple variant of the traditional waterfall model of system or software development. As illustrated in Figure 1, the V model builds on the waterfall model by emphasizing verification and validation. The V model takes the bottom half of the waterfall model and bends it upward into the form of a V, so that the activities on the right verify or validate the work products of the activity on the left. More specifically, the left side of the V represents the analysis activities that decompose the users' needs into small, manageable pieces, while the right side of the V shows the corresponding synthesis activities that aggregate (and test) these pieces into a system that meets the users' needs."*

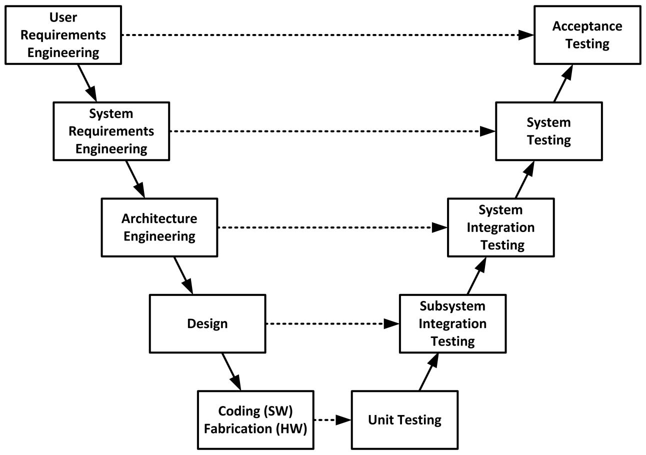

User Requirements Engineering
Acceptance Testing
System Requirements Engineering
System Testing
Architecture Engineering
System Integration Testing
Design
Subsystem Integration Testing
Coding (SW) Fabrication (HW)
Unit Testing

Figure 3: System engineering "V" model (example from [6])

The "V" model is more than just a re-shaped waterfall model. Organizations practicing Agile development – including Lockheed Martin in general and the AEGIS project in particular [7] – make use of the "V" by taking the indicated steps in short, rapid, iterative spirals (sprints).

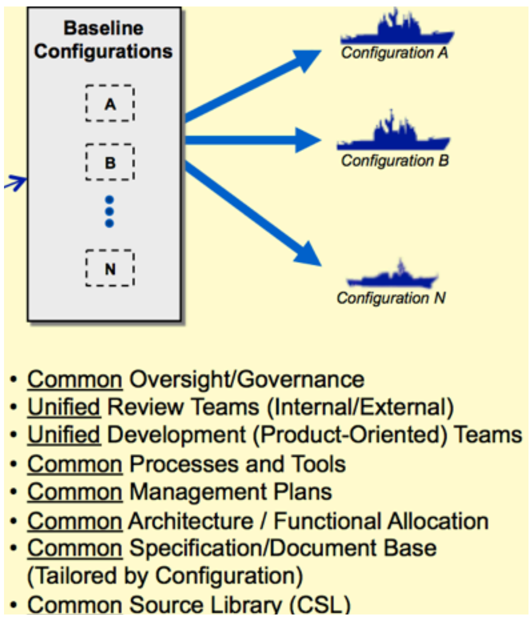
4. AEGIS AND PRODUCT LINE ENGINEERING

The AEGIS Weapons System (AWS) is developed, maintained, and evolved explicitly as a product line. Internally, and historically, the effort is known as the Common Source Library, although the product line comprises more than software. The AEGIS "products" come together as a single AWS that gets combined with other systems to create the AEGIS Combat System (ACS). Due to hardware, combat system, and mission differences, a single instance of the ACS requires creation of multiple variants. A significant saving across all phases of development is achieved by taking advantage of commonality across variants. Here is how Lockheed Martin's test process document describes the approach:

> *The primary objective of the Common Source Library (CSL) approach is to develop once and build and deploy many times from one set of common source code. This replaces the "clone and own" method of reuse for software and requirements. CSL supports the Navy's Rapid Capability Insertion Process (RCIP) objective. The CSL methodology supports the goal of minimizing cost and schedule for delivering computer program capability updates, as well as maximizing reuse across surface ship classes. One of the goals of a common product line approach is to achieve an economy of scale in cost, schedule, and reuse when compared to using a*

"Clone and Own" approach… The historical "Clone and Own" approach necessitated maintaining multiple copies of requirements and processing that are common to all stakeholders. Maintaining a single core software product that is common across multiple stakeholders implies that variations in core processing must be achievable across multiple programs each with potentially varying mission capabilities. Variation management is employed to separate core product functionality from capability unique functionality. Variation techniques provide the capability to develop and deliver product configurations to platforms or programs. [17]

Figure 4 continues the narrative with an excerpt of a Lockheed Martin slide showing how they characterize their PLE approach for AEGIS.



- Common Oversight/Governance
- Unified Review Teams (Internal/External)
- Unified Development (Product-Oriented) Teams
- Common Processes and Tools
- Common Management Plans
- Common Architecture / Functional Allocation
- Common Specification/Document Base (Tailored by Configuration)
- Common Source Library (CSL)

Figure 4: AEGIS as a product line

It has been reported that the product line approach has resulted in cost avoidance totaling $47 million annually, just from requirements- and software-related activities, when compared to what it would have cost to carry out those activities for each member of the product line separately [8].

Previous papers about this product line have described

- a three-level governance infrastructure put in place to balance the needs of the several U.S. Government agencies holding key stakeholder status in AEGIS [9]
- procedures and policies designed and adopted to ensure intellectual property protection in configured products; specifically to ensure that no export-restricted content is placed into products heading overseas [5];
- a closer look at the economics of the PLE Factory and how the cost avoidance benefits have increased as more shared assets are added over time [8];
- Lockheed Martin's experience of successfully combining Feature-based PLE and Agile-based development on AEGIS [7].

AEGIS remains a fertile proving ground for industrial-strength real-world PLE practices. It was inducted into SPLC's Product Line Hall of Fame in 2015 [24].

4.1 AEGIS AND FEATURE-BASED PLE

There are many ways to carry out PLE in practice. Some are based on simple reuse approaches where "core assets" are checked out of a repository and then modified in whatever way is needed to support specific products. Others involve the construction of a "common platform" upon which different products may be built.

Lockheed Martin is using a very specific approach to PLE that is neither of these, and in order to appreciate some of the points that follow, it is necessary to gain some insight into what that approach entails.

Their approach uses an automated commercially-available configurator to produce product-specific engineering artifacts from shared assets according to the feature selections specific to a product. This approach has often been referred to as "second generation PLE" [4][11][12][13] but we will refer to it here as "Feature-Based PLE," to align with a forthcoming ISO standard that will describe the approach by that name.

A product line includes various types of engineering assets, such as system or software requirements, design documentation, software source code, test cases and procedures, and more, that are used in the creation, deployment, and sustainment of products. In Feature-based PLE, the engineering assets are shared across the product line. Shared assets can be whatever artifacts are representable digitally. They either constitute part of a product or support the engineering process to create a product. These shared assets are created and maintained as *supersets*, meaning that they contain any content needed to support any of the products. A *configurator* (for example, and in Lockheed Martin's case, the Gears configurator and PLE environment [1]) produces product-specific instances by *actuating* a product — that is, exercising *variation points* in the supersets according to the *feature* choices for that product. A feature is a distinguishing characteristic that sets products in a product line apart from each other [10]. A variation point is a specification attached to a piece of content in a shared asset that stipulates the feature choices under which that content is needed in the product-specific instance of that shared asset. The collection of feature choices for a product is called a *Bill-of-Features*, and is drawn from all of the available feature choices for the product line, which are captured in a *feature catalog*.

Figure 5 illustrates these concepts. The shared asset supersets are shown in the "V" on the bottom left (the contents of the "V" figures are notional and can be whatever an organization chooses). Gear symbols denote variation points that are defined in terms of features in the product line's feature catalog. The shared assets, the feature catalog, the Bills-of-Features, the processes related to creation and evolution, and the staffed roles to carry it all out make up the *PLE factory*. In Figure 5, the PLE factory comprises everything to the left of the product subsets. Another name for the PLE factory is *production line*.

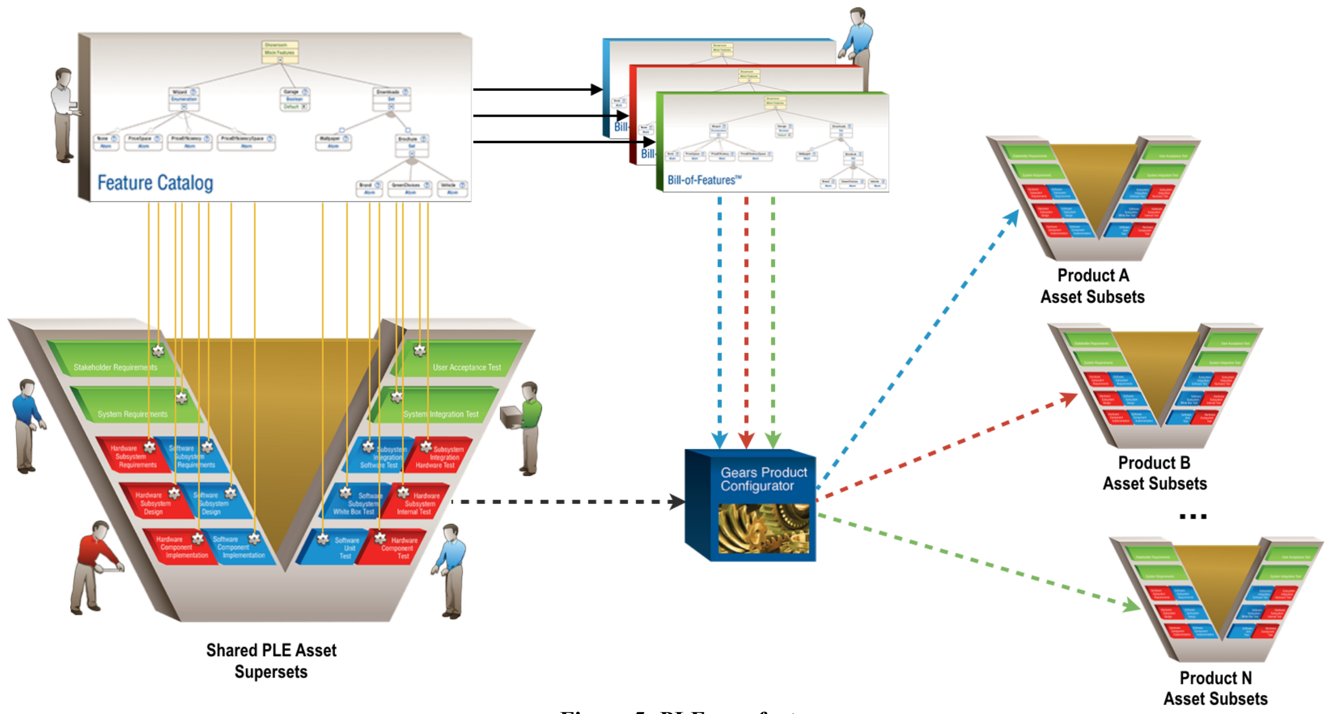


Figure 5: PLE as a factory.

Once this production line capability is established, products are instantiated – derived from the shared assets configured according to a Bill-of-Features – rather than manually created.

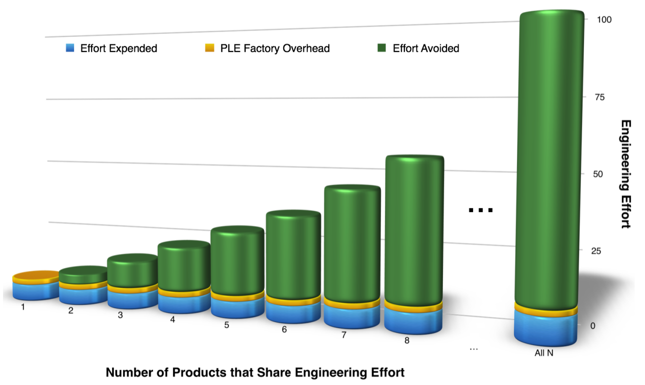


Figure 6: Effort avoidance due to sharing engineering effort across products

5. FEATURE-BASED PLE AND DEFECT MITIGATION

The unofficial name of this paradigm within Lockheed Martin and the U.S. Navy directorate responsible for receiving its output is "Fix it once!" [9]. Before Lockheed Martin adopted the Feature-based PLE approach for AEGIS, a single defect was fixed multiple times in multiples ways on the different AEGIS code bases, giving rise to egregious duplication of effort and expense. That expense was a primary motivation for moving to PLE. Now every defect is fixed once, inside the factory, and via the configurator the fix is propagated to every ship that requires it. On the flip side, an unintended consequence of moving to a feature-based PLE is that when a customer adds a new capability or improvement to the CSL, that feature is readily available to all the other CSL customers. This is managed using the feature catalog.

Figure 6 illustrates the idea. Every defect fix will affect some number of products. The blue component of each column accounts for the effort for making the fix, which is required in any case. The gold component accounts for the cost of setting up the PLE factory for the product line. The green component measures effort avoided by, in the case of defect solutions, "fixing it once" and not once per product.

6. TESTING ON AEGIS

Figure 7 explains the five levels of testing that must be accounted for on each and every AEGIS system. In the sections that follow, which discuss how testing activities can be most advantageously carried out in a PLE environment, these are the activities to which we are referring.

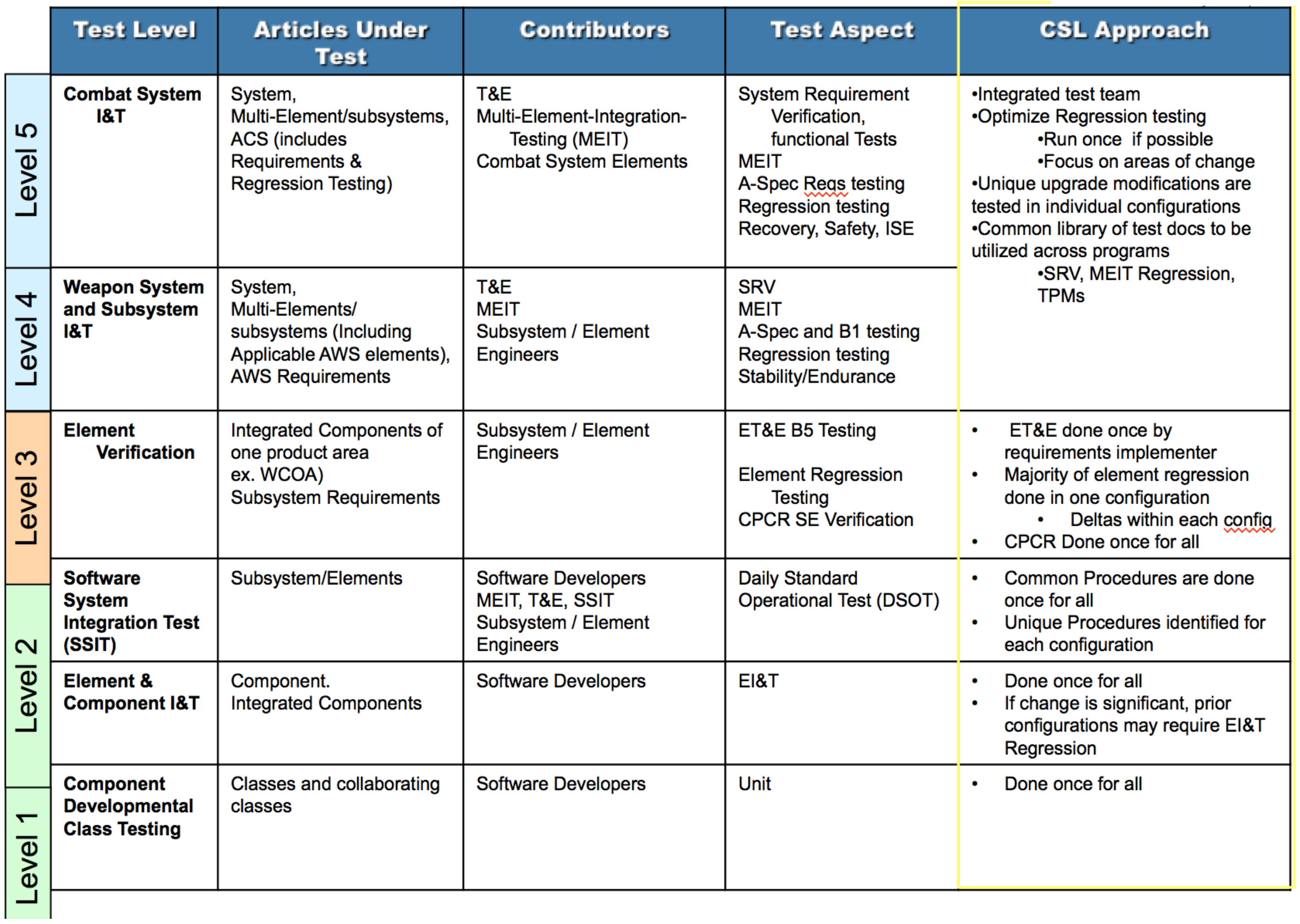
7. FEATURE-BASED PLE AND TESTING

Feature-based PLE can reduce the cost and effort associated with testing[1] in two ways:

- Increasing testing accuracy
- Decreasing testing activity
- For testing activities that do occur, Feature-based PLE offers the opportunity for dramatic savings by

[1] In the remainder of this paper, we will use the word "testing" as an informal shorthand for *any* verfication and validation activity.

performing testing activities inside the factory rather than once per product, outside the factory.



	Test Level	Articles Under Test	Contributors	Test Aspect	CSL Approach
Level 5	**Combat System I&T**	System, Multi-Element/subsystems, ACS (includes Requirements & Regression Testing)	T&E Multi-Element-Integration-Testing (MEIT) Combat System Elements	System Requirement Verification, functional Tests MEIT A-Spec Reqs testing Regression testing Recovery, Safety, ISE	•Integrated test team •Optimize Regression testing •Run once if possible •Focus on areas of change •Unique upgrade modifications are tested in individual configurations •Common library of test docs to be utilized across programs •SRV, MEIT Regression, TPMs
Level 4	**Weapon System and Subsystem I&T**	System, Multi-Elements/ subsystems (Including Applicable AWS elements), AWS Requirements	T&E MEIT Subsystem / Element Engineers	SRV MEIT A-Spec and B1 testing Regression testing Stability/Endurance	
Level 3	**Element Verification**	Integrated Components of one product area ex. WCOA) Subsystem Requirements	Subsystem / Element Engineers	ET&E B5 Testing Element Regression Testing CPCR SE Verification	• ET&E done once by requirements implementer • Majority of element regression done in one configuration • Deltas within each config • CPCR Done once for all
Level 2	**Software System Integration Test (SSIT)**	Subsystem/Elements	Software Developers MEIT, T&E, SSIT Subsystem / Element Engineers	Daily Standard Operational Test (DSOT)	• Common Procedures are done once for all • Unique Procedures identified for each configuration
Level 2	**Element & Component I&T**	Component. Integrated Components	Software Developers	EI&T	• Done once for all • If change is significant, prior configurations may require EI&T Regression
Level 1	**Component Developmental Class Testing**	Classes and collaborating classes	Software Developers	Unit	• Done once for all

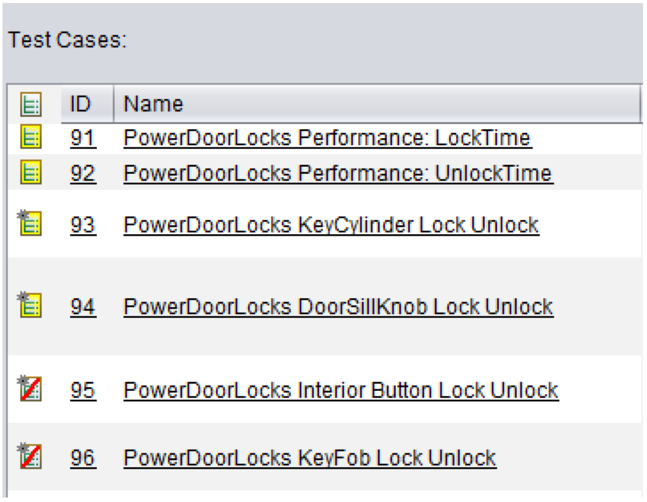
Figure 7: AEGIS levels of testing

7.1 INCREASING TESTING ACCURACY

Test artifacts – test plans, test cases, test procedures, and so forth – can be shared assets in a PLE factory. Like other shared assets, they can be imbued with variation points that allow the configurator to choose content for a specific product based on the feature choices (Bill-of-Features) for that product.

Figure 8 shows a screen shot from a test management tool (in this case, IBM's Rational Quality Manager) that has been integrated with Gears [2] to enable the creation and exercise of variation points to configure a suite of test cases to support a specific product. (This is taken from an example in the automotive domain.) In the six test cases shown, notice that test cases 93, 94, 95, and 96 have small gear symbols in their icon. This indicates that they are variation points, meaning that some products will require them and others won't. Test cases 91 and 92 do not have this icon, which means they are common – that is, every product requires them.

Each test case that is a variation point is accompanied by a logic statement (not shown) that indicates what feature selections cause its presence to be required in the end product. Figure 8 shows the results of an actuation in which feature choices were made that caused test cases 93 and 94 to be selected, and test cases 95 and 96 to be excluded, for the product. So, out of the test cases shown in Figure 8, test cases 91, 92, 93, and 94 should be executed for the product being built.



Test Cases:

	ID	Name
	91	PowerDoorLocks Performance: LockTime
	92	PowerDoorLocks Performance: UnlockTime
	93	PowerDoorLocks KeyCylinder Lock Unlock
	94	PowerDoorLocks DoorSillKnob Lock Unlock
	95	PowerDoorLocks Interior Button Lock Unlock
	96	PowerDoorLocks KeyFob Lock Unlock

Figure 8: Common test cases, test cases included, and test cases excluded, based on feature selections

The result is a suite of test artifacts perfectly tailored to the product being tested. This prevents running tests that are doomed to fail because the capability being tested was not even included in the product. It also prevents failing to run tests for a capability that was included. The first case is expensive. The second case, if a defect was allowed into the product, can be expensive to catastrophic.

Lockheed Martin is planning to take this approach with AEGIS, but is not yet carrying it out. They have focused on the second source of testing benefit from Feature-based PLE: Decreasing testing activity.

7.2 DECREASING TESTING ACTIVITY THROUGH SHARING

Testing can be a long and expensive activity for any system, but especially for systems as large and complex as AEGIS, where testing can approach 40% of the overall cost of fielding a system.

Carrying out testing in a Feature-based PLE context essentially involves answering the following question: What testing can you perform *inside* the factory, on the shared assets and the product line as a whole, and what testing must you perform *outside* the factory, separately on the actuated products?

Obviously, "inside the factory" is preferred. Analogous to the slogan for defect mitigation, this translates to "test it once." Were that possible in all cases, test effort and cost would be *optimally* decreased.

Of course, the more-expensive testing outside the factory on multiple product instances has arguments in its favor. One cannot know whether a system will behave correctly in its environment and context unless one tests it in that environment and context, which means in the context of a working product. In this view of the world, everything has to be tested in every variant of the product in which it is used, and no sharing is possible.

But that is prohibitively expensive, and so a middle ground must be found that balances resource expenditure with meeting reliability goals. The following sections explain how Lockheed Martin has found that middle ground to bring a very large portion of AEGIS testing inside the factory.

7.3 REDUCING TESTING OUTSIDE THE FACTORY

There are a number of ways that testing can be justifiably moved inside the factory, or at least avoided or reduced outside the factory, thus reducing the per-product duplication of effort inherent in outside-the-factory testing:

- **Standalone testing** is performed on the shared assets inside the factory, since no shared asset can be deemed "fit for duty" in a PLE factory unless it has been standalone-tested at a minimum. For software, this means unit testing. For hardware components, it means bench testing.
- **Build and test functional groups:** Some PLE organizations also combine software units that are always, or at least often, used in concert with each other in products. These essentially form larger units that can be pre-integrated and pre-(unit)-tested inside the factory [3].
- **Assess consequences of defects:** No program, not even a safety-critical one such as an avionics system, has the resources to test every aspect of every component to 100% reliability. The immediate conclusion is that the finite testing resources should be focused on the testing that matters the most. To help achieve this goal, DO-178B, the de facto quality standard of airborne systems, recognizes five *design assurance levels* for components in order to assist in applying pragmatic triage rules for testing [20]:

  A. **Catastrophic** - Failure may cause multiple fatalities, usually with loss of the airplane.

  B. **Hazardous** - Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the aircraft due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers.

  C. **Major** - Failure significantly reduces the safety margin or significantly increases crew workload. May result in passenger discomfort (or even minor injuries).

  D. **Minor** - Failure slightly reduces the safety margin or slightly increases crew workload. Examples might include causing passenger inconvenience or a routine flight plan change.

  E. **No Effect** - Failure has no impact on safety, aircraft operation, or crew workload.

  Where we have such a categorization, we could use it to triage our testing activities. For instance, under DO-178B it would make sense to rely on inside-the-factory testing for Level E components for sure, and very likely Level D components as well. For components of Levels A through C, unless one or more of the other criteria in this section apply, testing outside the factory would be indicated.
- **Assess past pedigree:** If a component or capability is stable and reliable, and has been so for an extended period of time across multiple usage contexts, and current usages contexts do not differ qualitatively from the past ones, it may make sense to scale back on its outside-the-factory testing. AEGIS uses the following scheme to assess the risk associated with any new development, as a way to address the confidence that may be assigned to it:
  - Type 1a: Low Risk, such as small intra-element changes or upgrades
  - Type 1b: Medium Risk, capturing larger or more complex changes not suitably described as low risk
  - Type 2: High Risk, including major re-architecture activities or capability upgrades, or changes that are cross-element in scope or affect system level threads.

  Each of these risk levels comes with a set of testing requirements appropriate for its risk level.
- **Assess sensitivity to context:** For AEGIS, Lockheed Martin assesses how "close" to an interface a unit of software resides in the software architecture. A unit of software buried deep down in lower layers, for example, is unlikely to be affected by different usage scenarios, and its per-product, outside-the-factory testing can be reduced. This metric can be applied proactively as well:

As code is re-structured over time, Lockheed Martin seeks to reduce the amount of software that is "near" an interface. For instance they push sensor and hardware dependencies to the boundaries, and not allow them to "infiltrate" the interior levels, to take advantage of this approach.

- **Exploit commonality and sharing wherever possible**. Whether testing activity happens inside or outside the factory, every opportunity to eliminate testing-related work that can be considered redundant should be taken.

  From inception, one of the primary tenants of the CSL was to maximize commonality so that savings could be achieved in every part of the development process, including test. Figure 9 shows how much commonality exists across the Baseline 9 variants.

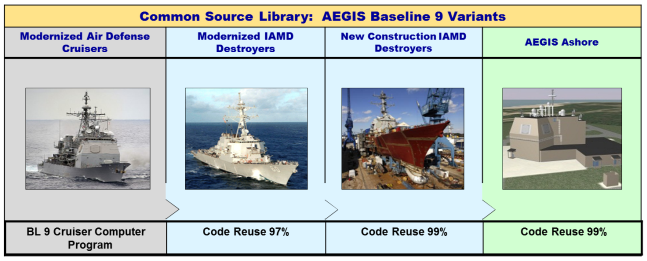

Common Source Library: AEGIS Baseline 9 Variants
Modernized Air Defense Cruisers
Modernized IAMD Destroyers
New Construction IAMD Destroyers
AEGIS Ashore
BL 9 Cruiser Computer Program
Code Reuse 97%
Code Reuse 99%
Code Reuse 99%

Figure 9: Commonality across Baseline 9 variants

  Lockheed Martin uses its AEGIS requirements database, also managed under the PLE Factory paradigm, as input to catalog what capabilities are shared across systems. If a requirement is shared, then the software that implements it is also going to be shared, and a candidate for exploiting commonality in testing.

  In addition to obvious candidates such as test processes, and standard formats and mechanisms to capture test results, Lockheed Martin reuses performance analysis, an important and expensive process in large real-time embedded systems, across the product line.

  The certification process for AEGIS, carried out by the Navy, uses a common set of test threads, applied across all product instances. Test results are captured in a DOORS database and provide objective evidence to support test re-use.

  In some cases it may be possible to structure a product line into "clusters," where products in a cluster have more in common with each other than with products in other clusters[2]. This may make it possible to at least share testing across a *cluster*, offering significant savings.

8. TESTING IN THE FACE OF CHANGE

Our story so far is that to minimize test cost under Feature-based PLE, you must reduce outside-the-factory testing as much as possible using any of the approaches listed in Sections 7.2 and 7.3 that are available to you. The remaining testing must be done on actuated products.

However, there's another factor that needs to be taken into account: Time.

Testing can be a long process for any system, but for AEGIS, *testing a new release can take up to a year and a half.*

The world doesn't stand still in that year and a half. While one release is being tested, the next release is being developed, and any defects in previous releases are being fixed..

By "release," we mean release of the entire product line – that is, the release of a new version of the shared assets, the feature catalog, and the Bills-of-Features. A release captures new or improved functionality that has been mandated by the governance bodies for AEGIS that determine its development priorities over time [9]. This is typically embodied within shared assets, but there may be new features as well, captured in the feature catalog and Bills-of-Features. With each new release, actuation is performed to produce the products, or at least the products that are taking the new release. Not every product is required to take every new release; a particular product may have little need for the new functionality, for instance, in which case they can wait for a future release.

The product line's shared assets and feature files are controlled in various configuration management repositories to provide history, version control, and reproducibility of the product line at specific points in time.

8.1 BRANCHING

When testing begins for an upcoming release, the developing organization needs a stable, relatively unchanging snapshot of the product line to work with. Lockheed Martin, like many development organizations, creates a *branch* to give them that stable snapshot. Under Feature-based PLE, however, Lockheed Martin branches the shared assets, the feature catalog, and the Bills-of-Features. Product-specific PLE subsets (actuated artifacts) are never used to create a branch. This is a crucial part of the formula to minimize testing cost.

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

Branching means to make a copy of an artifact so that special-purpose development can continue in parallel with "main" development activity. The "main" branch is called the *trunk*.

Before a branch terminates, the changes that were made either on the branch, or in the mainline after the branch was created, are merged. Merging means reconciling changes between a branch and the branch from which it was created. Merging should be done as soon as possible once development is complete since, as the lifespan of a branch increases, the chances of conflicting changes and potential magnitude and complexity of conflicts also increase, and the effort therefore required to reconcile any conflicts also increases.

Because copying and making changes to the copy is generally antithetical to PLE principles, branching should be done with extreme care and discipline to mitigate the effects of having multiple copies in existence. A sure way to destroy the product line's value is to fail to be disciplined about merging after a branch; the result is multiple copies of products with perpetual lives of their own – exactly the untenable situation that led to the move to PLE in the first place. Lockheed Martin and the US Government, in concert, have put in place extremely strong governance procedures and governing bodies that, among other things, explicitly control when branches may be taken, the content

[2] (Gears has the ability to represent this clustering in a structure called a product family tree [21].)

of each, and when new capabilities are ready to be merged into the mainline. These policies are described more fully elsewhere [9].

8.2 EVENT BRANCHES

When Lockheed Martin branches the factory in preparation for a release, they create what is known as an *event grooming branch*, or *event branch*[3] for short. Event branches are used to create a space where development can be "stopped" by isolating a stable, production-ready set of PLE superset artifacts away from the continued development taking place on the trunk (for example, on new capabilities not needed in the upcoming release). Event branches are used to support an upcoming test event or certification. Work performed on an event branch for a shared asset is focused on stabilizing the engineering of that asset needed for the event, and includes updates to fix software bugs or requirements defects. An event branch may be considered as a sort of "off-ramp" from the highway, or trunk, to a stable release.

No new development is performed on an event branch. Bug fixes and new development are performed back in the trunk. Only fixes that are relevant to the event or required for certification, are merged up into the event branch. The order of fixes is a matter of convenience. The number of changes allowed in an event branch is tightly controlled to ensure stability is maintained and to minimize wasted effort (since the event branch will be terminated). To maximize cost savings and speed to market (or speed to field capability upgrades), the length and number of event branches must be minimized and the frequency of releases must be maximized.

Since different releases of AEGIS are in different states at any point in time, there may be many event branches that need to be managed to support test and ship events.

8.3 TAKING ADVANTAGE OF PAST WORK

This strict change discipline, in conjunction with the Agile approach that divides work to be done into small identifiable chunks, makes it easy to chart what is new with each release. This in turn makes it easy to focus testing activity on that new content. Of course, regression testing is performed, but new testing focuses on new content.

9. CUSTOMER SUPPORT FOR THE APPROACH

The primary customer for AEGIS is, of course, the U.S. Government, and they are keenly invested (figuratively and literally) in this approach. They encouraged Lockheed Martin to take the product line approach in the first place, using the motto "Fix it once!" to encourage sharing of development effort. Now, "Test it once!" is sending the same message when it comes to sharing testing effort.

The Government eventually pays for the testing that Lockheed Martin carries out, of course, and so money saved by Lockheed Martin is money saved for the Government. Beyond that, the Government also bears a direct cost burden for its own participation in testing.

AEGIS systems are certified to go into operational service after a lengthy process that begins with requirements reviews. Since the CSL has multiple customers and the products have different missions there is a large community of subject matter experts involved in these reviews. Because Lockheed Martin is working in a PLE, these reviews can be shared across the product line, just like other test artifacts, and take advantage of commonality and previous pedigree of design decisions, just like other test artifacts. These saves the cost of Lockheed conducting many stand-alone requirements reviews and gives the CSL Customers insight into all changes that might affect their product.

Testing includes testing at LM and Government land based test sites, at-sea testing, and extensive performance analysis. This total cost to the Government of all of these activities, plus the cost of test facilities, can nearly double the cost of fielding capabilities. The commonality achieved from using the PLE process and leveraging it when certifying multiple variants results in significant cost savings. Individual certification for each variant would require additional test facilities, laboratories, equipment, and manpower that would be prohibitively expensive and directly impact the amount of much needed new capability that could be provided to the Sailor.

Finally, shipboard testing requires the participation of a ship. Every minute a U.S. Navy cruiser or destroyer is being used to field-test AEGIS is a minute when it is not fulfilling its primary operational duties, duties for which its many billions of dollars in cost were invested. Testing that fails to leverage previous test results, let alone testing each variant of AEGIS once per ship, is out of the question.

10. SUMMARY OF APPROACH

The over-arching goal of V&V is to be able to put a product into service with a high (in the case of AEGIS, an *extremely* high) degree of confidence that it will meet its requirements, which include strict safety and performance requirements. In practical terms, this means assembling a body of tests and artifacts that can be inspected to show testing success.

From the point of view, then, of testing particular products, below is a summary the steps we have laid out for testing in a Feature-based PLE context. We assume that the PLE factory is up and operating, and that a release is imminent.

1. When a major testing-related event is forthcoming, but as late as practical, create an event branch[4]. The event might be a release, or an important testing milestone activity. Until then, keep all development on the respective mainlines or "trunks" of all the shared assets, as well as the trunks of the feature catalog and the Bills-of-Features.
2. Continue any development work that is targeted only at the upcoming event, updating the shared asset supersets, feature catalogue, and Bills-of-Features as needed, on the event branch. (The goal of branching as late as possible is to keep as much work as possible on the trunks.)
3. Begin testing. Perform whatever tests are feasible on this new development, inside the factory. Use the criteria enumerated in Sections 7.3:
    a. Carry out standalone testing
    b. Carry out testing of functional groups

[3] This is similar to what some change management processes refer to as a release branch.

[4] A Branch/Merge plan for AEGIS, from which **Error! Reference source not found.** is taken, is managed jointly by Lockheed Martin and the US Government, and establishes the schedule for this.

    c. Assess consequences of defects
    d. Assess past pedigree
    e. Assess sensitivity to context
    f. Exploit commonality and sharing wherever possible

4. For *each* product variant that will participate in the upcoming event:
    a. As close to the release date as practical, in order to maintain shared artifacts for as long as possible, actuate to create the variant.
    b. Create a test "pedigree" for the variant.
        i. Capture the testing that was done inside the factory, whether on the event branch or back on the trunk.
        ii. Capture testing that was done that applies across an entire cluster to which this product belongs (see Section 7.3).
        iii. Executing the remaining tests on the actuated product and record the results.
5. After the event for which the event branch was created concludes, terminate the event branch.

11. CONCLUSIONS

Testing can be a long and expensive activity for any system, especially for systems as large and complex as AEGIS, where testing can approach half of the overall cost of fielding a system. Thus, any success in reducing the cost of testing activities yields a directly observable reduction in the overall cost of a fielding a new system.

In this paper we have shown how a large, complex, in-service product line is using PLE to reduce the cost of verification and validation activities. The central idea is to share (and therefore leverage) V&V activities across as many product variants as possible. Key principles discussed include:

- Share all assets and activities as much as possible, beyond just mainstream testing. This includes requirements reviews of all types, as well as performance analysis. Both of these can be time-consuming and complex, and by focusing on the common parts of systems, both can be substantially reduced.
- Decreasing testing activity by testing as much inside the PLE factory (that is, before product instances are created by the configurator) as possible.
- Exploit commonality wherever possible. Functionality that appears in more than one system should, to the extent possible, be tested once instead of once per system.
- Use branching policies to establish stable spaces where testing can be performed away from the instability of ongoing development.
- Keep things as common as long as possible. Delay branching and delay actuation as long as practicable.

These principles and approaches apply throughout all the test levels shown in Figure 7, from the software level up through system integration test and, ultimately, certification.

AEGIS employs a notion of "lead" and "follow" configurations where the follow configuration leverages the maximum amount of testing at all levels from the lead configuration. This ensures that test efforts are not replicated, by leveraging testing already completed or underway in similar configurations. Follow-on test efforts focus on testing specific to areas of functional or configuration differences, and do not repeat testing that is applicable across all configurations. This tailoring effort takes into account risk and change areas of functionality in follow on configurations, as well as the goals of test efficiency and affordability. Regression testing is performed for the other configurations as required.

The test approach presented allows for optimization of testing and test resources across multiple product instances without unnecessary overlap of test efforts. The test pedigree of one product line baseline enables reduced testing on subsequent baselines.

An unintended consequence of utilizing PLE for a large complex system is that the code base is continually being system tested. In the AEGIS case new development being done in the CSL uncovers bugs in the previously written code. These defects are addressed in the CSL trunk and included in the next release. This continuous testing and branching of new releases leads to fielding of higher quality systems.

PLE enables a "Build it once" (and therefore "Fix it once") capability. This capability has now evolved into "Test it once!" and is providing simultaneous levels of reliability and affordability that would not otherwise be available. Overall cost avoidance of some $47 million per year with the AEGIS PLE approach were reported in [8]. That figure results from sharing of requirements and software development work. In light of the fact that testing costs approach development costs in magnitude, and the very same sharing mechanisms are being brought to bear in testing as in development, it is reasonable to infer that cost avoidance from shared testing is commensurate with cost avoidance from shared development.

12. REFERENCES

[1] BigLever Software, "BigLever Software Gears," http://www.biglever.com/solution/product.html

[2] Bolander, B., Clements, P., Krueger, C. "It Takes a Village: Why PLE Technology Solutions Require Ecosystems of PLE Technology Providers," 26th Annual INCOSE International Symposium (IS2016), Edinburgh, July 18-21, 2016.

[3] Brownsword, L., Clements, P. A Case Study in Successful Product Line Development (CMU/SEI-96-TR-016, ADA315802). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996. http://www.sei.cmu.edu/publications/documents /96.reports/96.tr.016.html.

[4] Clements, P., Gregg, S., Krueger, C., Lanman, J., Rivera, J., Scharadin, R., Shepherd, J., and Winkler, A., "Second Generation Product Line Engineering Takes Hold in the DoD," *Crosstalk, The Journal of Defense Software Engineering,* USAF Software Technology Support Center, 2013.

[5] Clements, P., Krueger, C., Shepherd, J., Winkler, A. "A PLE-Based Auditing Method for Protecting Restricted Content in Derived Products," *Proc. 2013 Software Product Line Conference*, Tokyo.

[6] Firesmith, D., “Using V Models for Testing,” SEI Blog, Software Engineering Institute, Carnegie Mellon University, https://insights.sei.cmu.edu/sei_blog/2013/11/using-v-models-for-testing.html, November 2013.

[7] Gregg, S., Scharadin, R., Clements, P. “The Best of Both Worlds: Agile Development Meets Product Line Engineering at Lockheed Martin,” 26th Annual INCOSE International Symposium (IS2016), Edinburgh, July 18-21, 2016.

[8] Gregg, S., Scharadin, R., Clements, P. “The More You Do, the More You Save: The Superlinear Cost Avoidance Effect of Systems and Software Product Line Engineering,” *Proc. SPLC 2015*, Nashville.

[9] Gregg, S., Scharadin, R., LeGore, E., Clements, P. “Lessons from AEGIS: Organizational and Governance Aspects of a Major Product Line in a Multi-Program Environment,” *Proceedings, Software Product Line Conference 2014*, Florence, Italy, 2014.

[10] Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A. “Feature-Oriented Domain Analysis (FODA) Feasibility Study” (CMU/SEI-90-TR-021, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

[11] Krueger C., Clements, P., “Second Generation Product Line Engineering: A Case Study at General Motors,” in *Systems and Software Variability Management: Concepts, Tools, and Experiences,* Capilla, Bosch, and Kang, eds., Springer, 2013.

[12] Krueger, C. and Clements, P. “Systems and Software Product Line Engineering,” *Encyclopedia of Software Engineering*, Philip A. LaPlante ed., Taylor and Francis, 2013.

[13] Lanman, J., Kemper, B., Rivera, J., Krueger, C., “Employing the Second Generation Software Product-line for Live Training Transformation,” *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)* 2011.

[14] Linden, Frank J. van der, Schmid, Klaus, Rommes, Eelco. *Software Product Lines in Action*, Springer, 2007.

[15] Lockheed Martin, “Common is Anything But Boring: How Lockheed Martin Engineers Make Breakthroughs between Baselines,” http://lockheedmartin.com/us/news/features/2016/160706-Common-Anything-But-Boring.html, downloaded October 2016.

[16] Lockheed Martin, “Lockheed Martin to Bring Aegis Ballistic Missile Defense to Latest U.S., Korea and Japan Destroyers,” http://www.lockheedmartin.com/us/news/press-releases/2016/august/160815-mst-aegis-ballistic-missile-defense-to-latest-us-korea-and-japan-destroyers.html, August 2016.

[17] Lockheed Martin, “Common Source Library (CSL) Test Process,” AEGIS Program Technical Memorandum AP-C-SYS-T-2006, revised 9 June 2014.

[18] Naval Surface Warfare Center, “AEGIS Combat System,” http://www.navsea.navy.mil/nswc/dahlgren/ET/AEGIS/default.aspx

[19] Pohl, K., Böckle, G., van der Linden, F. Software Product Line Engineering: Foundations, Principles, and Techniques, Springer, 1998.

[20] RTCA, Inc., *DO-178B Software Considerations in Airborne Systems and Equipment Certification*, issued 12/1992, available from http://www.rtca.org/store_product.asp?prodid=581.

[21] Rubinstein, D., “BigLever draws ‘family trees’ in Gears,” SD Times no. 292, August 2013.

[22] Software Engineering Institute, “Benefits and Costs of a Product Line,” http://www.sei.cmu.edu/productlines/frame_report/benefits.costs.htm

[23] Software Engineering Institute, “Catalog of Software Product Lines,” http://www.sei.cmu.edu/productlines/casestudies/catalog/index.cfm

[24] Software Product Line Hall of Fame, “Lockheed Martin: U.S. Navy Aegis (Weapon System),” http://splc.net/fame/lockheed.html, downloaded January 2017.

[25] Software Product Line Conference (SPLC) Product Line Hall of Fame, http://splc.net/fame.html.

[26] Wozniak, L., Clements, P. “How Automotive Engineering Is Taking Product Line Engineering to the Extreme,” *Proc. SPLC 2015*, Nashville, 2015.