A PLE-Based Auditing Method for Protecting Restricted Content in Derived Products

Paul Clements
Charles Krueger
BigLever Software
10500 Laurel Hill Cove
Austin, Texas 78730 USA
+1 512 426 2227
pclements@biglever.com
ckrueger@biglever.com

James Shepherd
Andrew Winkler
Lockheed Martin
199 Borton Landing Road
Moorestown, New Jersey 08057 USA
+1 609 326 4685
james.t.shepherd@lmco.com
andrew.j.winkler@lmco.com

ABSTRACT

Many organizations that produce a portfolio of products for different customers need to ensure that sensitive or restricted content that may appear in some products must not appear in others. Examples of this need include complying with statutes in different countries of sale, protection of intellectual property developed specifically for one customer, and more. For organizations operating under these requirements *and* producing their products under a product line engineering paradigm that relies on automation in product derivation, there is a need for a method to ensure that the content restrictions have been met in the derived products. This paper describes an auditing method that meets this need. It was created for use in the Second Generation Product Line Engineering approach that is being applied by Lockheed Martin in their AEGIS ship combat system product line.

Categories and Subject Descriptors

D.2.2 [**Design tools and techniques**]: *product line engineering, software product lines, feature modeling, hierarchical product lines*

General Terms

Management, Design, Economics.

Keywords

Product line engineering, software product lines, feature modeling, feature profiles, bill-of-features, hierarchical product lines, variation points, product baselines, product portfolio, product configurator, product derivation, product audit, second generation product line engineering

1. Introduction

A significant challenge for many product line engineering (PLE) organizations is verifying that capabilities and content restricted for use to a limited class of products is not inadvertently leaked into other products outside of this limited class. Examples of this problem include:

- **Statutory compliance:** In PLE organizations that sell products in different countries, legislative differences might require a capability by law in one country and forbid that same capability under the laws of another country. For example, daytime running lights on automobiles are required in Scandinavian countries, but not allowed in Japan [3].
- **IP protection:** In PLE organizations that create custom product instances for different companies, a custom or license-restricted capability paid for by one customer might represent protected intellectual property that must never be used in the products sold to another company.
- **International Traffic in Arms:** In PLE organizations that create military or national security products that are sold in multiple countries, the government of the country where that PLE organization resides may have strict laws on the types of capabilities that can be exported to countries around the globe (for example [6]).
- **Classified information protection:** In PLE organizations that produce military systems that involve classified information, it may be necessary to strictly segregate that information away from scaled-down versions of the system that do not use the classified content.

The cost of inadvertently leaking restricted content can be extraordinarily high. Because these restrictions are often based on public safety laws, government use rights, or intellectual property laws, mistakes can result in large fines or legal judgments, protracted court cases, negative media coverage that damage the reputation of a brand, or (in extreme cases) even prison time.

In this paper we describe a method for verifiably protecting restricted content in product instances under the Second Generation Product Line Engineering (2GPLE) approach [2][3][5]. This work is based on industry experience with the AEGIS ship combat system, engineered by Lockheed Martin Mission Systems and Training Division using 2GPLE tools and methods, as well as experiences with other commercial 2GPLE practitioners. The AEGIS Combat System is an integrated warfare system deployed on over 100 naval vessels in the U.S. Navy and the navies of key U.S. allies across the globe. The issue of protecting restricted content is a critical concern in the AEGIS ship instances built for a diverse customer base..

(c) 2013 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the United States government. As such, the United States Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

SPLC 2013, August 26 - 30 2013, Tokyo, Japan

Copyright 2013 ACM 978-1-4503-1968-3/13/08…$15.00.
http://dx.doi.org/10.1145/2491627.2491629

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

2. Auditing Products for Capabilities and Content

Product line approaches often appear to characterize the portfolio of features and capabilities integrated into the product line as unvarying in their inherent applicability and sensitivity: Any feature can appear in any product. However, modern production environments include a great diversity in the origins, applicability, and often legality associated with the assembly of some sets of features and capabilities. A global marketplace and a diverse customer set introduce a myriad of additional considerations to managing a product line. An assessment to address these considerations can result in some product instances that are technically feasible but not viable as a product due to restricted content. This is especially true in products that include international stakeholders or customers who require that their indigenous products include components that they themselves provide but which must not be divulged to any other customer.

Product engineering based on one-of-a-kind products, clone-and-own, or even first generation PLE with its well defined *application engineering* silos support the need to isolate, validate and protect the restricted content in each individual product. In product silo approaches, ad hoc techniques for one-time manual inspection of content or automated scrubs of code and documents looking for sensitive terminology (keywords that suggest the presence of sensitive content) are often used, and can be sufficient. Once the product is created and scrubbed, considerations for restricted content become a tertiary concern, since those concerns can be addressed each time the product is changed, by making sure that the changes respect the restrictions. Hence, it is only necessary to perform the whole-product scrub (which can be quite labor- and time-intensive) once.

How do these product-centric auditing methods work? That is, how through the product line lifecycle development can stakeholders of these concerns (e.g., product line managers, legal experts, configuration managers, quality control managers) be sure that a given product line instance satisfies any restrictions? The key is the ability to assess the presence of restricted content over the course of the lifecycle through *capability auditing*.

A capability audit is a focused form of quality audit. Quality audits are "performed to verify conformance to standards through review of objective evidence. A system of quality audits may verify the effectiveness of a quality management system. Quality audits are essential to verify the existence of objective evidence showing conformance to required processes, to assess how successfully processes have been implemented, for judging the effectiveness of achieving any defined target levels, providing evidence concerning reduction and elimination of problem areas and are a hands-on management tool for achieving continual improvement in an organization" [7]. A capability audit is exactly this, focused on the singular "quality" of correct use of restricted content.

A "product manifest" is a useful technique to aid in auditing. Used across the different stages of the lifecycle, a manifest provides a set of checks and balances at different points of production and to provide an auditable paper trail so the source of "leaks" can be pinpointed. A manifest makes the executable image of an individual products derived from the product line self-documenting in terms of the origins of its content. The manifest describes the specific origins, heritage, and configured set of product line capabilities is thus included in each executable. Access to the manifest is of particular usefulness in engineering and integration environments where multiple product line executable instances, representing varying customer perspectives, may coexist.

Figure 3 shows useful audit points where manifest data associated with each executable is collected. Each executable exposes an executable fingerprinting method such as the Unix "what" utility, which can be invoked to expose the manifest. Each column in the figure represents a step in the creation and establishment of a specific product line instance in executable form. Software library labeling (such as ClearCase™ version labeling, used by AEGIS) is used to designate a set of software files representing a product specific instance and generation. That label is applied during product production to select and assemble, at the file level, common core and capability software versions, installing them into individual product-specific libraries. ClearCase labeling is again used to designate all the software content representing this specific product instance. The *tar* utility is then used to facilitate the accumulation of all the specific executables, libraries, and configuration files comprising the product instance. Following product installation, an executable fingerprinting utility can be used to access the metadata on a per-executable basis.

The manifest ultimately contains the information that is listed in Figure 2.

In addition to carrying its own parts manifest, engineers concerned with restricted content in products also use a comprehensive search for any customer sensitive terminology in which all the artifacts associated with a product (but principally its source code) are searched for keywords that would suggest the presence of illegal content. These searches for restricted content are fairly straightforward to carry out, but are labor-intensive. On AEGIS, whose product instances are very large and complex, they can consume up to 30 staff-months of effort. They suffer from a plethora of false positives that must be weeded out individually. They also are problematic if the right keywords are not included in the search list, allowing restricted content to slip by.

As we noted, these techniques suffice for product-centric approaches; they are applied essentially once when the product is built and then to each change to the produce thereafter. If every release of every product to every customer has to endure these complete scrubs, these methods become unsupportable and would, for all intents and purposes, break the product line.

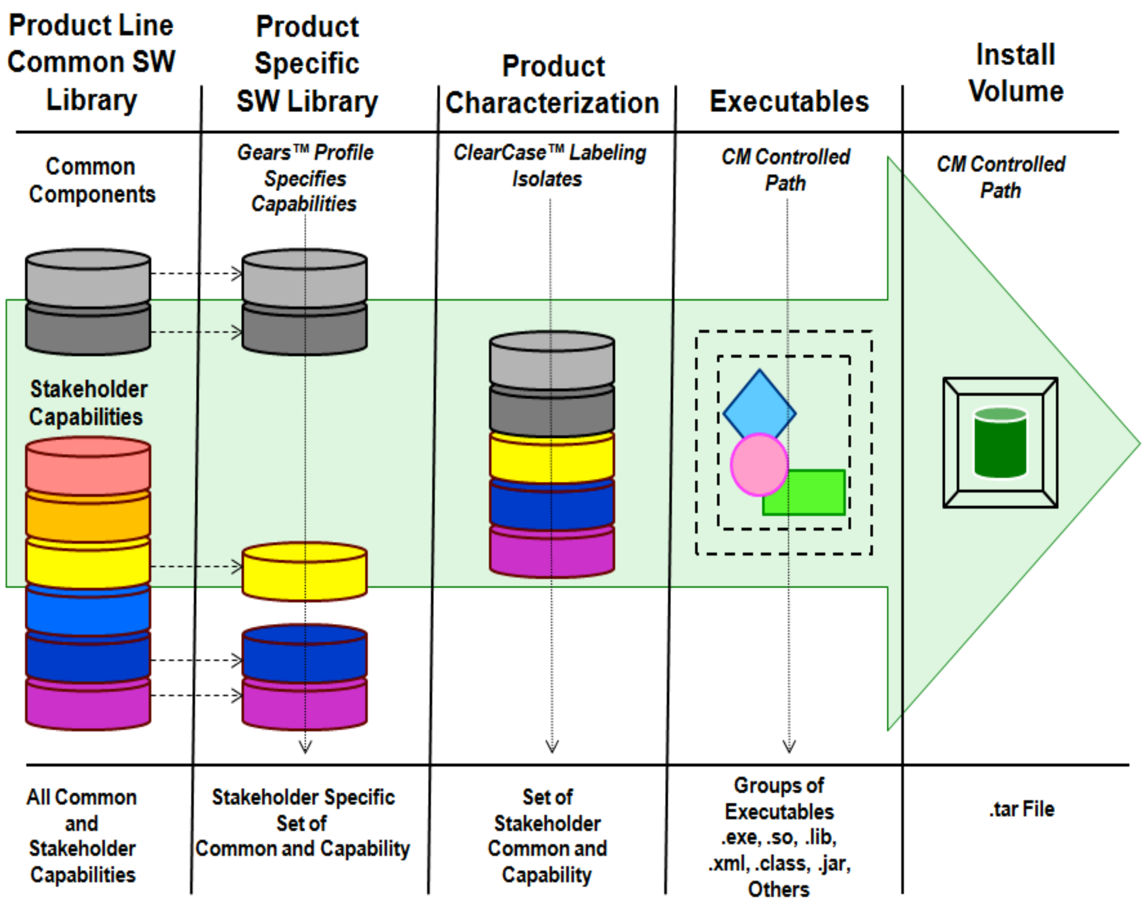
Product Line Common SW Library
Product Specific SW Library
Product Characterization
Executables
Install Volume
Common Components
Gears™ Profile Specifies Capabilities
ClearCase™ Labeling Isolates
CM Controlled Path
CM Controlled Path
Stakeholder Capabilities
All Common and Stakeholder Capabilities
Stakeholder Specific Set of Common and Capability
Set of Stakeholder Common and Capability
Groups of Executables .exe, .so, .lib, .xml, .class, .jar, Others
.tar File

Figure 3 Potential audit points for product instances (© Lockheed Martin)

- ***Component Name***
- ***Product Line Software Library Label***
  - ***Configuration Management Label***
- ***Product Instance Software Library Label***
  - ***Configuration Management Label***
- ***List of Features***
  - ***Included in the specific product executable being audited***
- ***Build Time***
- ***Build Information***

Figure 2 Typical contents of a product manifest, carried along in the product itself. For a manifest that documents the pedigree of software, the manifest is viewable by running a utility on the software's executable image.

3. Second Generation Product Line Engineering (2GPLE)

To understand the approach to capability and content auditing that we present in this paper, it is necessary to understand the basic tenets of the Second Generation Product Line Engineering (2GPLE) approach [2][3][5] employed by Lockheed Martin for the AEGIS program. Table 1 gives a brief summary of the four aspects of 2GPLE that are most germane to our audit approach.

Figure 4 illustrates the basic concepts of 2GPLE. Shared assets on the left (only a few examples of which are shown) are imbued with variation points (denoted by the gear symbol). The variation points are defined in terms of features. A feature profile, describing a product in terms of the features it exhibits, is fed to the product configurator, which configures the shared assets by exercising their variation points to produce a suite of asset instances specific to that product.

Because products are automatically generated, restricted product content is automatically configured into a product instance based on feature selections for the product and the mapping from feature selection to asset content selection used by the PLE configurator. Product instance development, evolution, fixes and enhancements are achieved by updating the feature selections or updating the shared PLE assets and then regenerating.

Most germane to the topic of auditing, 2GPLE re-generates any product to which a change is made. Contrast this to the product-centric (or even first-generation product line engineering) approaches in which a product, once derived, enters its own independent maintenance trajectory. If the "brute force" methods described in the previous section were all that were available to us, then under 2GPLE the cost of auditing would be prohibitive during a normal maintenance lifecycle, as it would have to be repeated with each re-generation, which would occur with each change.

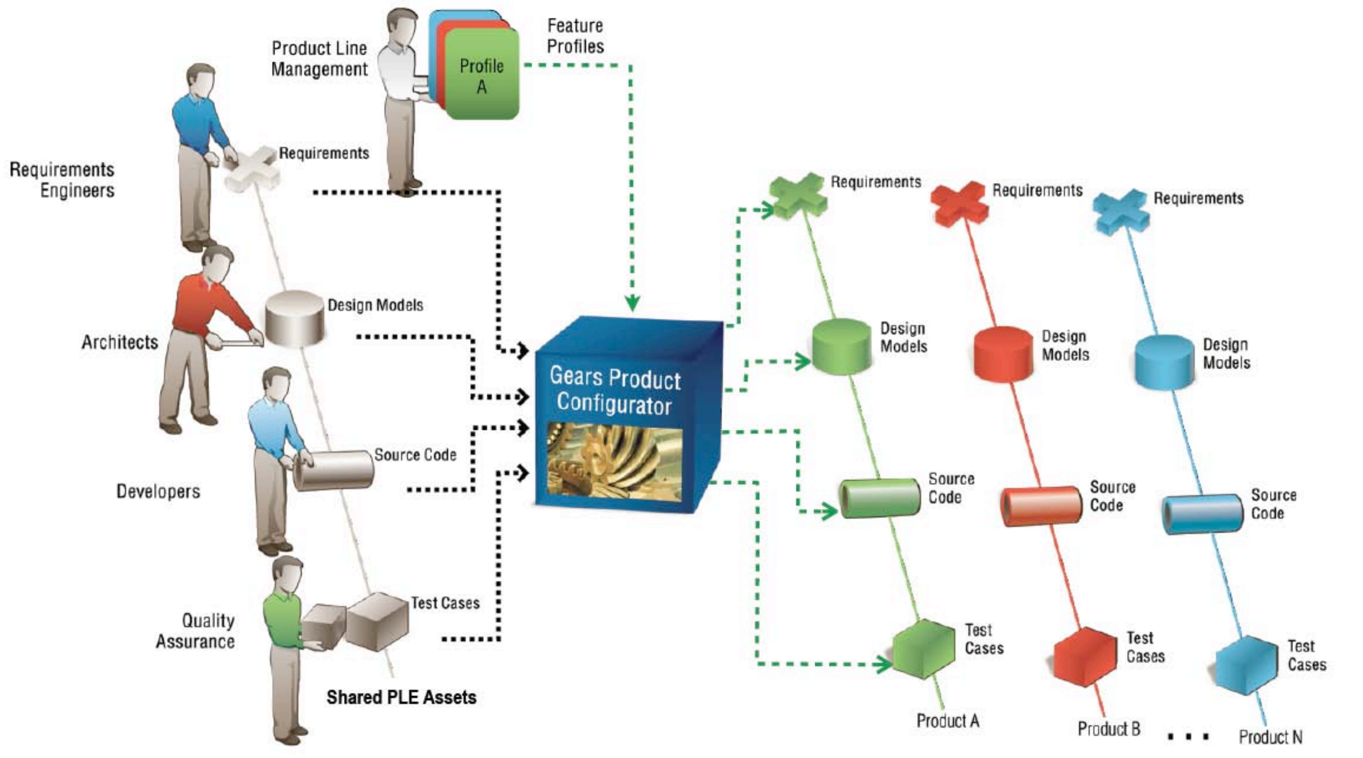

Figure 4 Basic concepts of Second Generation Product Line Engineering: shared assets configured according to feature profiles by a product configurator – in this case, Gears [1] (© BigLever Software)

Table 1 Aspects of 2GPLE germane to our audit method

Aspect of 2GPLE	How it relates to audit
Feature based: Products are described by the features that they exhibit, not (for example) by the parts that go into their construction. Following FODA [4], a *feature* is a distinguishing characteristic (often customer-facing) that sets one product apart from others. Features are captured in a feature model, which represents a set of choices available about each product. A *feature profile* is the set of feature choices actually made to define an individual product. For example, a feature model for an automobile might capture the fact that daytime running lights are available as low-beam headlights, or via the parking lights, or through special lamps dedicated to the purpose, or not on the car at all. A feature profile for a car would select one of these options [3].	In our approach to content protection, we will use features to represent the capabilities that must be avoided in some products.
All-asset perspective: In the systems and software engineering world, products comprise a number of artifacts that define their development and use. These artifacts include a broad array including requirements, design models, code, tests, user manuals, project management plans, blueprints, and many more. In 2GPLE, these artifacts are made generic – that is, applicable to every product – and thus become *shared assets*. In software product line engineering software enjoys the most attention and focus, but in 2GPLE all assets are first-class citizens. In fact, the products need not include software at all.	Our approach was crafted in the context of code as the primary delivered content, but because of 2GPLE's all-asset perspective, we believe it applies equally to any combination of assets.
Variation points: A shared asset is made generic across products by imbuing it with *variation points*, which are places in the asset where product-specific choices are inserted. In 2GPLE, the choices are expressed in terms of features, not directly referencing any product. In this way, the assets remain generic and can be re-used without change in any new product that is simply a new combination of already-existing features. Variation points come in a small and consistent set of forms, such as • omitting or including the asset; • selecting one variant file from an available choice of several, to serve as the product-specific instance; • performing text substitution inside the	As we'll see, our method uses variation points as a key focus of auditing attention. Each piece of restricted content (for example, a requirements passage, or a section of source code) is "protected" by a variation point; exercising the variation point will either include

asset; and including, omitting, or • choosing from among blocks of contiguous material inside the asset.	or exclude the restricted content.
Automated product derivation: The means of product derivation in 2GPLE is the mechanism that exercises the assets' variation points to produce configured versions that, together, constitute the artifact set for one of the products in the product line. The automation is called a *configurator*, which takes a feature-based description of a product (that is, a feature profile) and configures all of the assets (by exercising their variation points) to produce instances for that product. The configurator needs to be able to support the construction and management of feature models (including feature declarations, assertions, and profiles), assets and their variation points, and represent the logic that maps from feature choices to asset instances.	In our approach, the configurator in use provides useful information about our product line that relates to auditing assurance. For example, it will report any feature that is defined but never referenced, suggesting that somewhere a variation point that should have referred to it is amiss.

4. Classification of content violation errors

Understanding the basic tenets of 2GPLE enabled us to produce a classification scheme for the ways in which a content restriction violation could occur. There are two cases:

1. Restricted content is not correctly identified and demarcated as such. An example of this is when restricted content incorrectly shows up in a part of the source code believe to be common (that is, used in every product). Under our 2GPLE approach, demarcation happens when the part of the asset containing the restricted content is placed inside a variation point, and thus able to be included in some products and excluded from others.

2. Restricted content is correctly demarcated but not correctly chosen for use in products. This second possibility can occur in three forms, given our 2GPLE approach:
    a. Features are modeled incorrectly: For example, a restricted capability is not modeled as a feature, and so cannot be chosen for inclusion or exclusion from a product.
    b. The feature profiles are wrong. This occurs when a feature representing a restricted capability is chosen for a profile corresponding to a product in which the restricted content is forbidden.
    c. The logic for exercising the variation point is wrong. This occurs when the logic refers to the wrong features, or the logic itself is flawed. An example of the latter is when the logic expression that is some Boolean combination of feature values evaluates to true (and includes the restricted content) when false was intended.

Case 1 and Case 2 together partition the space of content violation errors, and in the case of the 2GPLE constructs we are using, the three possibilities under Case 2 enumerate all that can go wrong (other than tool failure): Either the features, the feature profiles, or the variation points are wrong. If our audit method can address each of these causes, we will have increased confidence of its robustness because it addresses each type of error.

Our classification scheme does not delve into cause, but merely effect. The source of one of these errors could be accidental, or malicious, and introduced at different times throughout the engineering process. Root cause analysis is not the point here; rather, the point is to try to catalog the errors to ensure that our auditing method intercepts each kind.

Our classification scheme, which will rely on inspection in places, does assume that we can recognize restricted content in an asset when we see it, which we believe is a reasonable assumption. Current practices assume the same thing, so we will not be less effective because of this assumption.

5. Audit Method

With the addition of international customers to the AEGIS family engineered under 2GPLE, the issue of verifiabily protecting content has taken center stage as an area of concern. Some AEGIS capabilities are targeted to specific customers only. Lockheed Martin has endeavoured to put in place safeguards that can demonstrate with high confidence that any limitations in the applicability of these capabilties are being. met. Lockheed Martin, for its part, wanted to put in place an audit method that was practical under the 2GPLE paradigm that enables rapid re-generation of any product at any time. The audit method we present in this paper emerged as a result.

Our method comprises three separate parts or stages:

1. **Careful construction**. This involves creating a set of best practices and style guides for the construction of the shared assets and feature models that will serve as the basis for the entire product line. The idea is to engineer the product line correctly (with respect to meeting content restrictions) from the start.
2. **Inspect the construction**. This stage involves auditing and reviewing the product line as constructed.
3. **Inspect the product**. A produt is built by actuating the shared assets against the feature profile for that product..

These stages are elaborated below. We also describe the capabilities of the particular configurator tool we are using, but discuss alternative approaches in case your configurator tool does not have the cited capability.

Stage 1: Careful construction

In Stage 1, best practices are codified in the form of style guides that engineers can use to do their work in building the product line – specifically, building the feature models and feature profiles, and imbuing the shared assets with variation points.

Table 2 enumerates the steps of Stage 1 and shows what parts of the content classification each one addresses. It also shows what capabilities of the configurator are assumed, if any.

If Stage 1 is carried out correctly, then no restricted content will be incorrectly included in any product not authorized to receive it. Stages 2 and 3 represent validation steps to catch any defects that slipped through Stage 1.

Table 2 Steps of Stage 1

Stage 1 Steps	Concerns addressed	Configurator capabilities used	Fallback if capability not available
Sound engineering practices			
Ensure by sound engineering practices (e.g., documented guidance, peer review) that no restricted content is inadvertently put in a place it doesn't belong – e.g., in a common part of an asset, or in an asset that will be used in a non-qualifying product.	1	None	N/A
Steps concerned with feature modeling			
Each restricted capability should be modeled as a feature. Each criterion that can define a content violation (for example, country of destination) should be modeled as a feature.	2a	Basic feature modeling	N/A
When building a feature profile, ensure that no illegal combinations are selected.	2b	Basic feature modeling	N/A
Write feature assertions that exclude illegal combinations of capability features and destinations[1].	2b	Ability to write feature assertions that must not be violated in any product	Rely on manual inspection.
Steps concerned with assets and variation points			
Each asset is chosen correctly for each product.	1 2c	Construction of variation points in shared assets, and logic to exercise the variation points based on feature values. Ability to place specific messages in the actuation report.	None
Each block of restricted content within an asset is contained in a variation point.			
The logic controlling each variation point correctly includes or excludes the restricted content based on feature values.			
The logic conrolling each variation point includes an output statement in the actuation report that says whether restricted content was, or was not, included in the instance of the asset.			

[1] Feature assertions represent a redundant quality mechanism. If no illegal feature profiles are created in the first place, feature assertions will never be violated, and can be said to have no effect. However, they represent an insurance policy against anyone creating illegal profiles in the future.

Stage 2: Inspect the Construction

In Stage 2, basic Quality Assurance techniques are employed to ensure that the practices prescribed in Stage 1 have been followed. The Engineering Review Board, already in place for AEGIS to enforce high quality system engineering practices, takes on additional scope to check that the product line is sound with respect to protecting restricted content.

While most of the work involves basic reviewing, the configurator can support certain parts of the task. Suppose that a source code file has a variation point in it that consists of two separate blocks of code. Suppose further that any product will contain exactly one of the blocks but the other will be removed. There must be logic to indicate which block is chosen and which is removed when the appropriate feature conditions apply. If the logic is stored separately from the blocks, then there must be a way for the logic to refer to the blocks – that is, the blocks must be named. This introduces the possibility of an error in which the logic writer mistypes the name of the block to be removed, thus potentially allowing restricted content into the wrong product[2].

Table 3 describes the verification steps of Stage 2 of our audit method.

The configurator at the heart of the 2GPLE paradigm is not only the engine that generates (and re-generates) product instances, but some configurators also can help in the audit task. It could detect any block name not referenced in the asset's logic file, and any block name mentioned in the logic that does not occur in the asset, thus helping to detect mis-typed block names. The tool could also check to see that every feature was used in a logic file somewhere, thus ensuring that it has a role in configuring at least

[2] This possibility exists even if asset instantiation is done not with a product configurator, but the simple #ifdef construct. The #ifdef statement refers to a variable that is #define'd (or not) elsewhere based on product-specific conditions. Thus, #ifdef also separates the controlled blocks from the logic that controls it. If the variable name is mistyped, the #ifdef has no effect.

one asset. And it could check to make sure that an asset that is treated as common to all products contains no variation points. These checks could be done manually, but are extremely tedious and, in product lines of any size, impractical. In Gears [1], the configurator chosen by use by AEGIS, this kind of checking is provided by a service called a Deep Semantic Audit. A Deep Semantic Audit provides a variety of information about the product line that has been defined, including the existence of possible anomalies that might warrant investigation. An example is an element (such as a feature) that has been defined but never used.

As with Stage 1, if all of the steps of Stage 2 are carried out correctly, then no product will have any restricted content that it is not authorized to contain.

Table 3 Steps of Stage 2

Stage 2 Steps	Concerns addressed	Configurator capabilities used	Fallback if capability not available
Check that Stage 1 steps have been correctly carried out: • All restricted capabilities are modeled as features • The feature profile for a product includes no feature that represents a forbidden capability • Feature assertions are in place to rule out any forbidden capability/destination combination • All restricted content is "protected" by a variation point in an asset that references the correct features that, when selected, allow the content to be included in a product and excluded otherwise • Every variation point in every asset includes selection logic that puts a message in the actuation report to indicate that restricted content is or is not being put into the generated instance.	1 2a 2b (as for Stage 1)	None	N/A
Ensure that block names are not mis-typed, thus undermining their meaning	2c	Deep Semantic Audit, an automated analysis of the entire product line to report anomalies, such as block names in an asset not mentioned in logic or block names in logic that don't occur in an asset	Manual check or script-based check to compare variation point names with names in logic, to search for variation points in common files, to search for unused features, etc.
Ensure that no variation points occur in common files	1		
Ensure every defined feature is used in at least one variation point	2a		

Stage 3: inspect and Analyze the Product

Whereas Stage 1 and Stage 2 applied to the product line at large, Stage 3 involves building and inspecting a product. In Gears the step of building a product is referred to *actuation*, which produces a product-specific set of asset instances by configuring their variation points according to the features of the product. After actuation, Stage 3 involves inspecting the product to make sure that, despite our best efforts in Stages 1 and 2, no restricted content has been improperly included.

Table 4 Steps of Stage 3

Stage 3 Steps	Concerns addressed	Configurator capabilities used	Fallback if capability not available
Review the actuation report to look for the embedded sensitive-content messages. Make sure that no messages are found that report unexpected	All	Produce a report describing the variation points exericsed, and how, during actuation	None

restricted content in the product			
Perform a scrub for sensitive terminology (keyword search in the product's artifacts; inspect the product's self-documenting manifest	All	None; these steps are performed by tools outside the configurator	N/A

The second step of Stage 3 is, essentially, a repeat of whole-product scrub that we described in Section 2. Recall that we said that this "brute force" scrub was too expensive and labor-intensive to be performed every time a product is generated or re-generated in the face of change. In our audit method, this step represents a last, redundant step of assurance before an actual customer delivery. In the AEGIS program, these do not happen often, and the program can support carrying this out on those occasions. This step will not be performed when products are generated during the development cycle (for example, during functional testing), and so the overhead is not considered prohibitive.

6. Conclusions and next steps

There are many circumstances under which careful auditing of products is vital to ensure that no restricted content is included in a product that is not qualified to receive it, because the cost of committing a statutory violation or failing to protect IP can be enormous. Organizations that wish to use the Second Generation Product Line Engineering paradigm, which is gaining wide acceptance for industrial-strength product line applications, must have a way to ensure that the generated products respect content rules.

This paper offers as its contribution an elaboration of the 2GPLE approach, in the form of an auditing process for derived products. This aspect of 2GPLE was previously missing.

The audit method we have presented has the following characteristics:

- It is based on careful consideration of the kinds of errors in a 2GPLE setting that can lead to a violation, and it has multiple steps that specifically address each error type.
- It relies on sound engineering and, in places, manual inspection, neither of which is immune to human error; however, to compensate, it takes a redundant "belts, suspenders, and shoelaces" approach that applies multiple remediation steps to cover each kind of error.
- Where helpful, the method relies on capabilities of the particular configurator in use on the AEGIS project, but where those capabilities might be considered unusual we have outlined fallback procedures for use in those cases in which a less capable configurator is in use.
- It is teachable, comprising three intuitive stages – build the product line correctly, review the product line that was built, then build and review any product being delivered – and a set of specific steps for each stage in checklist form.
- It was crafted to work for source code assets, but could be applied equally well to any kind of documentation-based artifact that (through the use of feature-based variation points) occurs in product-specific instances. Thus, the method is suitable for use with assets across the lifecycle.
- It incorporates current best practices in product-based auditing, such as a self-carrying manifest and keyword scrubs, as a last-ditch safety net.

The audit method was created to prevent restricted content from making its way into products forbidden to receive it. However, there is an interesting "reverse case" that often arises as well: A customer insists that some of its homegrown content be inserted in its product. This so-called "indigenous material" presents a case in which content must be *included* rather than excluded in a product (although it likely must be excluded from all the other products). The audit method can be straightforwardly adjusted to work for this case as well.

One outcome of the work is to show that 2GPLE can, in fact, produce auditable products. Features provide a good proactive protection mechanism that sets the stage from the beginning for representing restricted capabilities. Variation points, expressed in terms of features, are therefore also expressed in terms of the capabilities that need to be controlled. Thus, expressing when a capability must be included or excluded is as straightforward as can be using 2GPLE's feature-based variation approach.

The method has not yet been applied throughout AEGIS so there is no empirical data to present with respect to its cost and its relative effectiveness compared to present-day approaches. We can, however, reason about these important concerns:

- With respect to effectiveness, because it includes the currently used approaches (keyword search and product content manifests) applying our method can only be more effective, not less, than current practice.
- With respect to cost, the Stage 1 steps represent a best-practices approach for building a high-quality product line that respect content restrictions, becoming an inherent part of the product delivery lifecycle. These steps can be codified in style guides for product line engineers to follow when building their feature models, profiles, and variation points, and would be adopted as a quality improvement approach even if no auditing were involved at all. Additionally, the last step of Stage 3 represents today's approach. Therefore, the marginal cost brought about by the method is the Stage 2 inspections (several of which are or could be carried out by the configurator) and, in Stage 3, the examination of the actuation report (a step of almost zero cost).

A next step is to gather actual cost and benefit data for applying the method throughout the AEGIS program on product versions. Another next step (one that is specific to source code) is envisioned to provide still one more layer of protection and added confidence: It should be possible to modify the compiler's lexical scanner and parser to have it look for keywords associated with restricted content. This would provide a parallel approach that can be thought of as add baling wire to the belt, suspenders, and shoelaces of the audit method we have presented.

Our expectation is that the auditing approach for features and capabilities presented in this paper will quickly become the routinely used and preferred approach for auditing of product line in the customer pre-delivery context. An enhanced keyword scanning utility, optimized for scanning the content of common product line, including supporting features, would also be introduced, with the goal of reducing the search time from weeks to hours, mainly by reducing the number of false positives reported by brute force searches. The role of this modified scanning utility would be relegated strictly as a last-measure gating criterion each time a product needs to be delivered to a customer site (be it a lab or a ship). The rest of the audit method presented in this paper would do most of the work.

7. Acknowledgments

Glenn Meter of BigLever Software played a valuable role in helping to create the approach presented in this paper. Bob Mirto and Susan Gregg provided helpful reviews.

8. References

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

[2] 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, in publication.

[3] Flores, R., Krueger, C., Clements, P. "Mega-Scale Product Line Engineering at General Motors," *Proceedings of the 2012 Software Product Line Conference (SPLC)*, Salvador Brazil, August 2012.

[4] 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.

[5] Krueger, C. and Clements, P. "Systems and Software Product Line Engineering," *Encyclopedia of Software Engineering*, Philip A. LaPlante ed., Taylor and Francis, 2013, in publication.

[6] U.S. Department of State, "International Traffic in Arms Regulations (ITAR)," http://pmddtc.state.gov/regulations_laws/itar_official.html

[7] Wikipedia, "Audit," http://en.wikipedia.org/wiki/Audit