Survey of Product-Line Verification and Validation Techniques

---

Task #1 Deliverable
Product Line Verification of Safety-Critical Systems

FY2007 SOFTWARE ASSURANCE RESEARCH INITIATIVE PROPOSAL
for the
NASA SOFTWARE IV&V FACILITY

Prepared by:
Robyn Lutz

Dated:
May 15, 2007

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and at NASA Ames Research Center, under a contract with the National Aeronautics and Space Administration. The work was sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA Software IV&V Facility. This activity is managed locally at JPL through the Assurance and Technology Program Office.

Summary

Survey of Product-Line Verification and Validation Techniques

This report presents the results from the first task of the SARP Center Initiative, "Product Line Verification of Safety-Critical Software." Task 1 is a literature survey of available techniques for product line verification and validation.

- *Section 1 of the report provides an introduction to product lines and motivates the survey of verification techniques. It describes what is reused in product-line engineering and explains the goal of verifiable conformance of the developed system to its product-line specifications.*
- *Section 2 of the report describes six lifecycle steps in product-line verification and validation. This description is based on, and refers to, the best practices extracted from the readings. It ends with a list of verification challenges for NASA product lines (2.7) and verification enablers for NASA product lines (2.8) derived from the survey.*
- *Section 3 provides resource lists of related conferences, workshops, industrial and defense industry experiences and case studies of product lines, and academic/industrial consortiums.*
- *Section 4 is a bibliography of papers and tutorials with annotated entries for relevant papers not previously discussed in sections 2 or 3.*

Section 1. Introduction

Statement of the problem. Product-line engineering of NASA systems offers the opportunity for significant cost savings and increased quality control. Reuse of domain-engineered product-line assets can reduce the cost and time to market of the new system, and can improve the quality of the developed products.

However, with this opportunity come new verification challenges. Specifically, we seek to answer the questions:

- "How should we verify that delivered software conforms to the product-line requirements and architecture levied on it and how do we document that conformance?"
- "How should we verify that safety-critical software built using product-line assets is safe?"

Solutions exist and have been applied successfully in industry, but need to be customized for NASA's unique needs.

Need for verifiable conformance. NASA is, with the rest of industry, turning to product-line engineering to reduce costs and improve quality by effectively managing reuse. In product-line engineering, assets such as a common architecture and shared requirements are reused to build each new system in that product line. Experience in industry has shown that it is the *verifiable conformance* of each system to the product- line specifications that makes or breaks the product-line practice. Verification that the software for each project satisfies its intended product-line constraints is thus essential. This report surveys product-line techniques that exist in industry as a first step toward that goal.

Software product line. A software product line is defined to be "a set of software-intensive systems sharing a common, managed set of features that satisfy the particular needs of a specific market segment or mission and that are developed from a common set of core assets in a prescribed way" [Clements and Northrop].

Domain Engineering. Product-line development is typically divided into two phases: Domain Engineering, in which the product line assets are specified and developed, and Application Engineering, in which the product line assets are reused to build each new system in the product line. The first phase, Domain Engineering, depends on the knowledge and skill of domain experts to produce a set of optimized product-line assets. These typically include a product-line architecture and a domain model that specifies both the software requirements common to all systems in the product line, and the variation points at which the systems will differ. Most of the work to date in academia and at NASA has been on effective Domain Engineering of product lines, both since it comes first and since correct scoping and specification of the product line is basic to its success.

Application Engineering. In the second phase, Application Engineering, the product line assets, such as a common architecture and shared requirements, are reused to build each new system in that product line. It is this second phase that will make or break the product-line approach for NASA because it is the phase that depends on the conformance of each individual project to the product-line specifications previously levied on it. Verification that the software for each project satisfies its intended product-line constraints is essential. This report focuses on Application Engineering, specifically on verification and validation techniques needed for the Application Engineering of Exploration Software systems built using product line assets.

Goal of the literature survey. This survey is a step toward the goal of developing a baseline verification process to show that the software in each system in the product line is, in fact, product-line compliant. The verification process will check the consistency of the system software intended to be built with product-line assets against the product-line requirements themselves. In addition, it will help assemble a case that the software is compliant with product-line constraints associated with it. That is, we want to answer the question: "How should we verify that the software of interest conforms to the product-line requirements and product-line architecture levied on it and how do we document that conformance?"

Unique challenges of product lines. Verification of product lines differs from verification of single systems in the greater opportunities for reuse of the product-line framework and product-line assets, including requirements specifications, models, and test cases, with potential attendant cost-savings and improved quality. Verification of product lines also differs from verification of single systems in the added complexity of developing and supporting multiple variants, multiple configurations for different customers, and multiple component versions across differing product lifecycles within the product line.

Unique challenges of NASA product lines. Verification of NASA product lines will differ from verification of other product lines. We list here some of the key differences:

- Many product lines will be safety or mission critical. These product lines will thus also be built to satisfy NASA GB-8719.13, the Software Safety Guidebook. This is unlike many existing consumer product lines, which do not involve safety issues. There are, however, a growing number of safety-critical product lines, such as cockpit displays [Lutz et al., 98], medical imaging systems [Schwanke and Lutz], pacemakers [Liu et al.], and satellites [Dehlinger and Lutz, 2006].
- Many NASA product lines will be built by contractors rather than in-house, whereas many (but not all) product-line engineering techniques assume that the domain engineers and the application engineers work together.
- Extended and remote missions require additional autonomy. To date, most product lines have not dealt with autonomy. Satellite product lines are a notable exception.

Section 2. Literature Survey

This literature survey is concerned with the Application Engineering phase, i.e., the reuse of the product-line engineering assets to build a set of new systems. The product-line engineering assets passed to the Application Engineering phase are the *reusable artifacts* that were previously defined, developed, and tested in the Domain Engineering phase.

Product-line assets. Typically, the artifacts stored in the product-line repository include [Gomaa, Pohl et al., Weiss and Lai]:

- common and variable requirements specifications (typically the result of the product-line level Commonality and Variability analysis)
- reference architecture
- protocols
- component design specifications
- component code
- test suites
- interface specifications and code
- configuration options
- infrastructure (feature models, tool support, etc.) for constructing new systems from the domain-engineered product-line assets.

- dependability case input (e.g., test results, development-process profiling, metrics for domain-engineered assets)

Reuse of product-line assets. These assets, or artifacts, are reused to build a sequence of applications. The different applications are each customized, or tailored, to the needs of the specific customers or stakeholders by selecting different combinations of features from among those defined in the domain-engineering of the product line. Each of these applications, built using the product-line assets (what Gomaa calls the product-line repository), is a member of the product line [Gomaa]. This report focuses on the specific problem in the Application Engineering phase of how to verify the conformance of a new system to the product-line assets. McGregor presents a framework for developing product line test assets in his tutorial [McGregor].

Applicability to Constellation Program. Because our report is to be cleared for public release, we merely note that the qualification and certification process of off-the-shelf software, including reuse and legacy software, as described in the NASA Constellation Program Software Verification and Validation Plan, CxP 70086 Baseline (Draft), is consistent with the process described here. In that plan, as here, there is an emphasis on traceability and on assuring that the documentation needed to verify and validate the software is provided.

2.1 Requirements verification

Verifying whether the requirements for the new system conform to the product-line requirements can only be accomplished if adequate documentation exists. Ardis and Weiss, in a tutorial at the 1997 International Conference on Software Engineering described how to identify and document the requirements for the product line. An essential output from the Domain Engineering of the product line is a documented Commonality and Variability Analysis (CVA) [Weiss and Lai]. The CVA documents the common and variable (optional and alternative) requirements on all members of the product line.

Product-Line assessment. In many cases, the product-line assets do not satisfy all the requirements for the new system. Early assessment of the delta enables an evaluation of whether it is feasible to add the missing features on top of the product-line assets.

- In such a case, the new feature(s) may be incorporated as additional product-line assets. This sort of systematic evolution of the product line maintains the product line by updating the hazard analysis, requirement specifications, design models, code repository, and test suites on an as-needed basis.
- Alternatively, the new feature(s) may not be incorporated into the product line but developed independently for the new system. Often in practice, the new features are subsequently added to the product line, since other systems subsequently also require those new features.
- Finally, if the features required for the new application differ too much from the product-line features, then the application is not a viable member of the product line. In that case the assessment may find that it is better to develop the new system separately [Gomaa].

Verifying consistency of requirements. The existence of a product-line feature model [Kang, et al.] supports the selection of requirements for the new system. It is important to verify the consistency of the requirements. In many cases, the selection of one feature will preclude or otherwise affect the selection of another feature. This topic, called feature interaction, has been extensively studied but outside of product lines and within product lines. Verification that the set

of variants selected for a new system is consistent is best done with tool support. Prototype tool support for consistency checking has been developed by [Padmanabhan and Lutz].

Model-driven development. Gomaa describes how static and dynamic UML models created for the product line (part of the domain-engineered product-line repository) can be selected and tailored according to the features selected for the application. Model-driven development of product lines supports an incremental approach to the development of the new application since the common (or kernel) features can be included first, followed by the optional features already modeled in the product line, followed by any components that need to be added in response to the requirements for this new system.

2.2 Safety Requirements verification

Verification that the software requirements satisfy the safety requirements for the system can likewise use results from the preliminary hazard analysis, as well as software fault tree analysis and software failure mode, effects and criticality analysis performed on the product line. Leveson and Weiss have described how software reuse can be managed in safety-critical systems [Leveson and Weiss]. Dehlinger and Lutz have shown how software fault tree analyses can be performed at the product-line level and reused, via tool-supported pruning to exclude fault paths not relevant to the new system's choice of features [Dehlinger and Lutz, 2006].

2.3 Architectural verification

Verification that the software architecture for the new system conforms to the product-line reference architecture traces the architectural elements in the new system back to their abstract elements in the product-line architecture.

There has been substantial work done on evaluating alternative architectures. ATAM (Architectural Tradeoff Analysis Method) [Kazman et al.] is perhaps the best known of these techniques. ATAM has also been used to evaluate alternative architectures for product-lines.

Verifying architectural changes. If changes have been made to the reference architecture for the new system, perhaps in response to new requirements, they must be carefully verified. Some changes are local while others have cross-cutting effects (i.e., to several components) on the architecture. Some changes may involve replacement of one component by another without changing the connectors, while others may affect the structure of the connectors or of the dataflow among components. The greater the changes, the more evaluation will be required (e.g., using ATAM) and the less verification effort will be saved by reuse of the reference architecture.

2.4 Design verification

Verification that the detailed design for the new system conforms to the product-line components' detailed design is enabled by traceability from the product-line requirements to the product-line component design. There is little work specific to product-line engineering in this area, perhaps because the issues are so similar to those in reuse. In general, if the product-line scoping has been done effectively (i.e., if the crystal ball has accurately envisioned future needs), then the range of options already available in the product-line assets should suffice. In that case, there should be few tweaks to the product-line assets, and the rationale and effect of those tweaks should be clearly documents. If application-specific needs cause changes to the product-line components, then the verification effort is increased. In that case, verification that the software design satisfies the software requirements for the system will be needed.

Model-based design analysis. The verification that the design satisfies the requirements may be assisted by modeling of the new system. Goal-oriented requirements engineering, described by van Lamseerde and Letier and Mylopoulos [Mylopoulos, et al.], supports the analysis of design satisfaction or satisficing of requirements. Liu, Dehlinger, and Lutz have shown how safety-related scenarios derived from the hazard analysis can be used to exercise state-based models of product-line components in order to verify that the as-designed behavior is safe [Liu et al.].The process of design verification for a new application again depends on accurate documentation of non-conformance to the product-line assets.

2. 5 Implementation verification

Verification that the implementation of the new system conforms to the product-line consists must check that the product-line assets (component code files, interfaces and parameters, configurations, libraries, and databases) that are used are consistent with the current configuration of those product-line assets and are integrated correctly. This helps provide assurance that the configuration management of the new system is consistent with the configuration management of the product line.

Problem reports. Analysis of problem reports for a product-line system is especially important because the problem reports generated during development of the new system may identify latent and unforeseen differences from the product-line components. Since both the product-line components themselves and all the possible compositions and configurations of those product-line assets have already been tested, the expectation would be that the number of problem reports would be reduced. This makes problem reports important for two reasons:

- Problem reports may indicate previously hidden problems that can affect other operational or to-be-built systems in the product line. Making other projects aware of the anomaly thus precludes similar anomalies on other systems.
- The problem reports may be the best locator of undocumented non-conformances between the new system and the product line. Adequate documentation of these deltas is, as we have seen, the baseline for scoping the verification effort on the new system.

2. 6 Testing verification and validation

Test infrastructure. Based on the previous verification steps, testing of the new system validates both that it meets its own requirements and that it conforms to the product-line requirements (since its requirements have been previously verified as consistent with the product-line requirements). Testing of product lines has been widely studied. For example, workshops have been held specifically on Software Product Line Testing (SPLIT) in 2004, 2005, and 2006.

Product-line tests. A key piece of the domain engineering of product lines is the development of reusable test suites. The goal is to save time and money during application engineering by providing product-line test assets. Each new system in the product line can then use this set of unit, interface, integration, and perhaps some system tests. Some product-line tests will be reusable in whole, because they test commonalities. Other product-line tests will need to be configured according to the new system's specific selection of product-line features and options. Similarly, the more similar the new system's domain is to the envisioned product-line domain the more reuse of domain-specific, product-line test suites may be useful.

Identifying and constructing test cases. Clement and Northrop describe how to structure the testing software for reuse, e.g., by providing traceability to product-line architecture and by

grouping the test code for a software component for traceability from test code to source code, by associating test cases with use cases or scenarios. The mapping associates the test cases, together with the test drivers and test data sets, with the commonalities. The goal is to product assets to test assets in order to facilitate reuse.

For example, since "the majority of each product's specification will be defined in a document generic to the document line" they suggest that we should "define a complete set of functional tests for that specification" [Clements and Northrop]. They also propose definition of a set of interaction tests to ensure that product-specific functionality will not interact to cause failure, but do not discuss how this is to be done.

Clements and Northrop distinguish between testing of the core asset software, of the product-specific software, and of the interactions between them. They recommend weighting testing toward points of variability in the product line architecture since most changes (and probably the most errors) will occur there.

Product-line test plans. They also recommend organizing test plans and test reports hierarchically (including sampling strategies and coverage criteria) to parallel the relationships among the products in the product line. This helps ensure that common features will be tested the same way in every product.

The test plan for the new system identifies the scope and rationale for the reuse of the product-line test suite and identifies new or changed features for which new tests to be developed. The test plan also identifies new or changed domain environments which may affect reuse of domain-engineered test suites.

Independent V&V. Knauber and Hettrick distinguish between development testing and validation in product lines. The latter they assign to an independent quality organization. They divide the testing of a new application into testing at the component level, the feature level (with the features being the integration test units) and the product level (system testing). "During application engineering, assets from this test infrastructure should be reused to speed up testing and thus overall product development." "The expectation here is that test assets are customized by using the same resolutions in the same decision model that are used for customization of the other product line assets" [Knauber and Hettrick].

Use-case-based testing. Gomaa recommends basing integration test cases product lines on the use cases of the components to be integrated. The white-box integration testing tests the interfaces between the components that participate in each use case. The black-box functional testing cases are built for each use case.

Variant absence tests. Pohl et al. suggest designing and running specific tests to ensure that features that are options in the product line but that should have been left out of this new system are, in fact, left out. These tests, called "Variant Absence Tests", would check such things as compile-time configuration mechanisms (such as IFDEF statements), or run-time configuration mechanisms (such as registry), as well as calling a function provided by the (supposedly absent) variant and observing the result [Pohl et al.].

Identifying relevant test cases from the product-line test suites. This depends on good traceability from requirements to component design and test cases in the product line. It also depends on good documentation of any deviations of this system (e.g., new features, changed code) from the product-line baseline. Reuse of test cases is limited by the number and degree of

such deviations. In such cases the product-line test suites will have be adjusted or enhanced to accommodate the changes.

Effect of variability bind time on testing. Pohl et al. point out that adequate documentation regarding when the binding time of a variability occurs (e.g., whether an alternative is selected at design time or at implementation time) affects when the variability can be detected. If it is bound at runtime then tests will need to be run on all possible configurations (e.g., operational modes) that can involve it.

Generating acceptance tests. Geppert et al. report success in using the decision model, traditionally used to generate code, to generate acceptance test cases. In an application to parts of a legacy acceptance test suite of a product line with about 600 commonalities, 100 variabilities, and 200 modules, they analyzed 30 test cases encompassing 174 description items and 98 result items. They found that by using seven parameters to generalize the test cases, they could reduce the number of description items from 174 to 23 and the number of expected result items from 98 to 18, a reduction of duplication effort of around 85% [Geppert, et al.].

Testing acquired product lines. The SEI has produced an on-line volume, "Software Product Line Acquisition: A Companion to A Framework for Software Product Line Practice, Version 3.0," targeted to Department of Defense acquisition. It is available at http://www.sei.cmu.edu/productlines/companion.html. The Testing section of this document notes:

> "For acquisitions, planned tests are performed on evolving products, not simply the end product. These tests not only provide feedback for the developers of the software, but they also provide measurable evidence of progress toward delivery.
>
> Results of the tests are analyzed and compared to the agreement requirements to establish an objective basis to support the decision to accept the product or to take further action" [Bergey04].

The document emphasizes the importance of spelling out in the supplier agreements the content of the product-line verification process (including early testing, conformance with product-line specifications, and delivery of product-line testing assets). The recommendations provide useful insights into some of the ways in which acquiring and verifying product line software from providers may differ from in-house development.

2. 7 Verification challenges for NASA product lines

Organizational considerations. A major consideration for verification of NASA product lines is how the development organization will be structured. Weiss and Lai make the point that for large, complex product lines, decomposition into subdomains may be the answer. They give the example of an organization that might have a device subdomain, a display subdomain, a database subdomain, and so on. Each member of the product line is then built by building a member from each subdomain and then integrating the members. Each subdomain thus forms a product line, and the systems built form a product line of product lines. The organization responsible for integrating the domain members is distinct from the organization for each domain. Weiss and Lai note that, "Particularly well-disciplined organizations may find that they can create and maintain subdomains that are used in a variety of product lines, thereby gaining additional leverage from each subdomain."

Many-customers, many-product-families. Weiss and Lai distinguish between three situations: the single-customer, single-product-family situation, the many-customers, single-product-family situation, and the many-customers, many-product-families situation. NASA will face the latter. The advantage is that investment in each domain is amortized over many products. Weiss and Lai note that this situation "may also require a change in organizational structure to make domain ownership clear and to ensure feedback to the domain owner (who employs the domain engineering) from the domain users (who employ the application engineers)" In other words, communication, documentation and traceability are essential.

Ommering, at Philips, similarly notes that, "Our organization contains a set of domain engineering teams that produce subsystems. We also have application engineering teams that create products. Typical of our approach is that there is an "m" to "n" relationship between subsystems and products and also between subsystems mutually."

Reluctance to invest for other projects. In their introduction to the special section on the Eighth International Conference on Software Reuse, Frakes and Kang state, "A key idea in software reuse is domain engineering (aka product line engineering)." They then distinguish between centralized organizational approach and a distributed, collaborative approach in which a reuse program is implemented by projects in the same product line. They note that "There is a danger that projects may be willing to use other's products, but will be reluctant to make investments for others." This is another way in which an organization can impede product-line development and verification.

Contracted product line development. Ommering briefly discusses in-house versus third-party software development in Philips consumer electronics product lines. He states that shifting software development to the supplier of the hardware and to independent software vendors helped manage the problems they encountered when scale of the project grew beyond a few hundred developers for a specific project. The outsourcing of software mapped nicely to their software architecture except that they see a middleware layer forming between the platform and application layers. This middleware layer is where they expect the vendors to provide the software [Ommering].

Controlling the delta. Inevitably, the needs of specific applications will need additional or alternative features not initially provided in the product-line assets. Such adaptations may occur at the requirements, design, or implementation level. For example, at the requirements level, a new feature may be required. At the design level, an alternative design mechanism may be selected. At the implementation level, new coding standards may drive change. Each of these changes has the potential to change the product-line. Some changes may also drive up the cost of the new system.

The decision as to whether to incorporate the changes into the product-line assets and how broadly to disseminate the changes (e.g., whether to update other products similarly) will be difficult and important. The project's Change Control Board for the specific system considering such adaptations will probably be the first line of defense in maintaining the product-line assets and in judging which proposed changes merit moving away from the product-line standard set of verified components.

Bergey points out some of the threats to verification that evolution can entail [Bergey]. Not folding evolutionary changes made to meet the requirements of a new system back into the other systems in the product line keeps the cost of evolution down, but can over time lead to

degradation of the product line and the erosion of commonality. On the other hand, propagating each change to the product line can make the re-verification costs unmanageable.

2. 8 Verification enablers for NASA product lines

Several process features that enable high-quality product-line verification are identified in the literature. We briefly describe several that appear to be most relevant to future NASA missions:

- *Rigorous traceability* is a key to being able to perform verification on product lines [Bergey]. Similarly, traceability is the first keystone for structured reuse, according to Pohl et al.
- *Capturing the requirements delta* information in the application requirements specification is the second keystone noted by Pohl et al. [Pohl]. This allows scoping of features and code that will have to be developed beyond the product-line assets.
- *Sufficient documentation* is essential. Verification that the new system conforms to the product-line requirements, uses the reference architecture, and passes the product-line test suite requires good documentation of the new system and its development phases.
- *Access to source code* for the product-line components is needed for thorough testing. Pohl et al. point out that, since any executable evaluation copy contains bound variants, additional artifacts such as source code and compilation and linking instructions need to be available in order to adequately verify the code.
- *Start small.* Weiss and Lai recommend first applying product-line engineering to a legacy system "in which there is frequent change occurring at relatively high cost. Such a domain is often an isolatable section of the system where the changes can be encapsulated and where a single group of software developers is responsible for making the changes" [Weiss and Lai].
- *Use small configuration units.* Rockwell Collin described the tradeoff in their recent experience building a large product line: "Having more items leads to a finer grained decomposition of functionality, which, in turn, leads to higher levels of reuse and systems that are easier to test and get through airworthiness qualifications. However, having more items also means there are more configuration items to keep track of and maintain, more complexity, and a steeper integration curve" [Clements and Bergey].

Section 3. Resource List

3.1 Conferences

- International Software Product Line Conference. The SPLC are the leading forum for product line research and practice. The 11th SPLC will take place in Kyoto, Japan, Sept. 10-14, 2007.
- International Conference on Software Reuse. The ICSR is the major conference for software reuse. The 9th ICSR was held in Turin, Italy, June 11-15, 2006.

3.2 Workshops

For product-line testing, the most important workshop is the Software Product Lines Testing Workshop (SPLIT). The fourth SPLiT will be held in Kyoto, Japan, Sept. 10, 2007, in conjunction with SPLC.

The SPLC has several other product-line workshops associated with it. This year, the others are:

- International Workshop on Dynamic Software Product Line (DSPL 07)
- Managing Variability for Software Product Lines
- Open Source and Product Lines 2007, Asia OSSPL07 Asia
- Service Oriented Architectures and Product Lines - What is the Connection? (SOAPL - 07)
- Value-Based Product Line Engineering (VBPL'07)
- Workshop on Visualisation in Software Product Line Engineering (ViSPLE 2007)

There have also been several ICSE workshops on software product lines, e.g., on software product lines: economics, architectures, and applications.

3.3 Industrial Experience

The International Software Product Line Conference hosts a "Product Line Hall of Fame" with a website describing the best-practice experience of the winners at http://www.sei.cmu.edu/productlines/plphof.html. For example, past winners include:

- RAID controller firmware product line, LSI Logic - Engenio Storage Group (2006)
- General Motors Powertrain (2004)
- Ericsson AXE Family of Telecommunications Switches (2004)
- Salion's Product Line of Revenue Acquisition Management Systems (2004)
- Philips Product Line of Software for Television Sets (2004)

The experiences of some of these developers with the product lines have been documented in case study reports of varying levels of detail, with links provided at the site.

Gomaa's product-line book focuses on UML modeling of design. He presents three lengthy case studies of product lines (microwave oven, e-commerce, factory automation), but with little consideration of application engineering. Clements and Northrop's book includes three substantial case studies: engine software, ground software for satellite control, internet stock market analysis software.

The FAST process (Family-Oriented Abstraction, Specification, and Translation) defined in Weiss and Lai has been used at Lucent and Avaya to build product lines [Weiss and Lai]. Frakes and Kang in their review of various product line engineering approaches note that only FAST (of the ones they review) is a process model rather than a development technique [Frakes and Kang].

FAST defines the "Engineer Application Activity" as an iterative process for constructing application systems that are members of the product family to meet customer requirements. The activity contains two process sub-states—"Produce_Application" and "Delivery_And_Operation_support" and an operation, "Model_Application" [Weiss and Lai].

This process pays careful attention to the conformance of the final new system to its product-line specifications. For example, an item in the Analysis List for the Application Engineer role is: "Final_Product_Validation_Analysis: Check whether all the decisions made in the application model exist in the final product."

FAST assumes standard testing techniques for the system: "Test the integrated application"; "The application should be tested to ensure that it meets the customer's requirements"; "Verify the

integrated family member." An exit condition for development is that code and documentation have both been reviewed.

Weiss and Lai assume the existence of an Application Modeling Language. The emphasis is on correct formal specification (in terms of meeting customer requirements) and valid specification (in terms of being a valid family member) and on a tool-supported application-engineering environment. With an application modeling language, they do auto-generation of code and documentation for as much of the new member as can be generated, manual development of any customized portions, and integration of the code.

There is an emphasis on traceability among the application model, the design decision model, the application-modeling-language expressivity, the documentation, the defect reports, the validation tests, and the existence of tool support in the application-engineering environment for generation.

3.4 Defense Industry Experience

The eighth DoD Software Product Line Workshop was held in 2005 to share Department of Defense product line practices, experiences, and issues in DoD software product lines. Presenters were from Austin Information Systems, Raytheon, Naval Undersea Warfare Center, Northrop Grummann, Aerospace and SEI. While the focus in not on verification, the empirical and anecdotal descriptions of successes and obstacles provides useful insights into designing and transferring product-line practices [Bergey et al.].

The SEI has produced a lengthy on-line report to guide the acquisition of product lines and product-line components by the Department of Defense http://www.sei.cmu.edu/productlines/companion.html The testing section of this document is discussed above in 2.6.

The U.S. Army's Common Architectural Avionics System is a recent example of a successful product line. The product line is of a set of helicopter mission and avionics software systems. It was developed by TAPO and its prime contractor, Rockwell Collins, which has experience in product lines. The development of the product line is bringing cost savings in development, integration, flight testing, and maintenance costs. According to the authors, "CAAS systems benefit from strategic software reuse of around 80% or more" [Clements and Bergey]. Similarly, "Airworthiness qualifications happen in a much shorter time." While the focus in not on verification, the description of the development environment, communication channels needed between customers and developers, and influence of organizational cultures is quite applicable to NASA applications.

3.5 Product Line Consortia: industry/academia

The best-known site for information about product lines is that of the Software Engineering Institute at Carnegie Mellon University: http://www.sei.cmu.edu/productlines This site, divided into "Technologies", "Learning", and "Community" resources, provides, among other links:

- a catalog of product lines
- a detailed framework for product line development
- links to other product line websites
- a product line bibliography
- teaching resource materials for product lines

The focus of the material is on software architecture. A description of the relevant parts of the SEI verification process is included above in the description of Paul Clements and Linda Northrop's book.

The Fraunhofer Institute for Experimental Software Engineering (IESE), http://www.iese.fraunhofer.de/fhg/iese/research/development/pla/index.jsp, similarly focuses on product-line architecture. Their work includes a searchable bibliography maintained through 2004. IESE developed the KobrA method for component-based product-line engineering, documented in Atkinson's book [Atkinson].

The European Space Agency and the EU have invested heavily in product families. For example, the ESA has developed a small Geostationary Platform product line to "enable European players to compete effectively in the commercial telecom market for small platforms." http://telecom.esa.int/telecom/www/object/index.cfm?fobjectid=28211

Similarly, the ITEA (Information Technology for European Advancement) project, FAMILIES, is being undertaken by a consortium of European industries and universities. FAMILIES stands for FAct-based Maturity through Institutionalisation Lessons-learned and Involved Exploration of System-family engineering, http://www.esi.es/Families/ FAMILIES is a follow-on to the previous ESAPS and CAFÉ project to mature, standardize and disseminate product-line engineering technologies. To date the project has begun work toward standardization and has developed a collection of methods not yet supported by available software tools.

There are, in addition, several university/industry collaborations centered on academic research labs. For example, the Aspect Oriented Model Driven Product Line Engineering group (led by Dr. Awais Rashid at the University of Lancaster and funded by the EU) focuses on improved modeling and analysis of variabilities to improve adaptability http://www.ample-project.net/ . AMPLE correctly notes that architecture models are related to requirements models in an ad-hoc fashion and that implementation tends to rely on pre-processors which are inadequate substitute for proper programming language support for variability. AMPLE also describes the lack of a systematic traceability framework for relating variations across a SPL engineering lifecycle. The tools page is not yet populated.

Section 4. Bibliography

Note that papers discussed above are not annotated here.

Ardis, Mark and Weiss, David M., "Defining Families: the Commonality Analysis", Tutorial at 19th International Conference on Software Engineering, Boston MA, 1997.

Atkinson, Colin, et al., Component-Based Product Line Engineering with UML. Addison-Wesley, 2002.

Bergey, John K, Cohen, Sholom, Donohoe, Patrick, and Jones, Lawrence G., Software Product Lines: Experiences from the Eighth DoD Software Product Line Workshop, Dec. 2005, CMU/SEI-2005-TR-023.

Bergey, J.; Campbell, G.; Cohen, S.; Fisher, M.; Gallagher, B.; Jones, L.; Northrop, L.; & Soule, A. Software Product Line Acquisi-tion: A Companion to A Framework for Software Product Line Practice, Version 3.0. 2004.
http://www.sei.cmu.edu/productlines/companion.html

Bertolino, Antonia and Gnesi, Stefania, "Use Case-Based Testing of Product Lines." In Proceedings of the 9th European Software Engineering Conference held jointly with the 10th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE), pages 355--358, September 2003. The authors use use cases to generate generic test cases and then derive test scenarios for specific applications. They demonstrate the technique on a user's playing of a game on a mobile phone.

Clements, Paul and Bergey, John, "The U.S. Army's Common Architectural Avionics System (CAAS) Product Line: A Case Study," September, 2005, http://www.sei.cmu.edu/pub/documents/05.reports/pdf/05tr019.pdf

Clements, P. and L. Northrop, Software Product Lines. Boston: Addison-Wesley, 2002.

Dehlinger, Josh and Lutz, Robyn R., "A Product-Line Approach to Promote Asset Reuse in Multi-Agent Systems," In Software Engineering for Multi-Agent Systems IV, Lecture Notes in Computer Science 3914, pp.161-178, 2006.

Dehlinger, Josh and Lutz, Robyn, "PLFaultCat: A Product-Line Software Fault Tree Analysis Tool", Automated Software Engineering, 13(1): 2006, pp. 169-193.

Feng, Qian and Lutz, Robyn, "Bi-Directional Safety Analysis of Product Lines", Journal of Systems and Software, 78(2), Nov., 2005, pp. 111-127.

Frakes, William B., and Kang, Kyo, "Software Reuse Research: Status and Future", IEEE Transactions on Software Engineering, Vol. 31, No., 7, July 2005, pp. 529-536.

Geppert, Birgit, Li, Jenny, Rößler, Frank and Weiss, David M., "Towards Generating Acceptance Tests for Product Lines," International Conference on Software Reuse, 2004. J. Bosch and C. Krueger (Eds.): ICSR 2004, LNCS 3107, pp. 35–48, 2004.

Gomaa, Hassan, Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architecture, Addison-Wesley, 2005.

Kang, K. C, Kim, S., Lee, J., Kim, K., Kim, G. J. and Shin, E., "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures,". Annals of Software Engineering, 5, 1998, pp. 143-168.

Kauppinen, Raine, Taina, Juha and Tevanlinna. Antti, " Hook and Template Coverages for Testing Framework-based Software Product Families." In Proceedings of the International Workshop on Software Product Line Testing (SPLiT 2004), pages 7—12. This paper notes that "surprisingly little is written about testing of product families" and offers two new coverage criteria for framework-based product families: hook coverage and template coverage. These measure how much of the functionality of the application's code for extending an application framework has been covered with existing test suites.

Kazman, R., Klein, R. and Clements, P., "ATAM: Method for Architecture Evaluation." CMU/SEI-2000-TR-004.

Knauber, Peter and Hetrick, William, "Product Line Testing and Product Line Development — Variations on a Common Theme," in Geppert, Birgit, Krueger, Charles, and Trew, Tim, eds.

International Workshop on Software Product Line Testing, Rennes, France, Sept. 26, 2005, co-located with Ninth International Software Product Line Conference.

Leveson, N. and Weiss, K. A., "Making Embedded Software Reuse Practical and Safe," Proceedings of Foundations of Software Engineering, 2004.

Liu, Jing, Dehlinger, Josh and Lutz, Robyn, "Safety Analysis of Software Product Lines Using State-Based Modeling," To Appear Journal of Systems and Software.

Lutz, Robyn, "Extending the Product Family Approach to Support Safe Reuse,". Journal of Systems and Software, vol. 53, 2000, pp. 207-217.

Lutz, Robyn and Gannod, Gerald, "Analysis of a Software Product Line Architecture: An Experience Report," Journal of Systems and Software, vol. 66, no. 3, June, 2003, pp. 253-267.

Lutz, Robyn, Helmer, Guy, Moseman, Michelle, Statezni, David, and Tockey, Steve, "Safety analysis of safety requirements for a product family," Proc. 3rd IEEE Int.l Conf on Requirements Engineering, 1998.

McComas, D., Leake, S., Stark, M., Morisio, M., Travassos, G., and White, M., "Addressing variability in a guidance, navigation, and control flight software product line," in: Proceedings SPLC1, Product Line Architecture Workshop, August, 2000. This paper describes the modeling of a NASA product line but does not address verification.

McGregor, John D., "Reasoning about the Testability of Product Line Components," in Geppert, Birgit, Krueger, Charles, and Trew, Tim, eds. International Workshop on Software Product Line Testing, Rennes, France, Sept. 26, 2005, co-located with Ninth International Software Product Line Conference. This discusses how to make product line more testable with a focus on software architecture.

McGregor, John D., "Building Reusable Test Assets for a Product Line", tutorial, First Software Product Line Conference, Pittsburgh, PA, SEI, CMU, 2000. http://www.cs.clemson.edu/~johnmc/conferences/ProductLineTutorial.html

McGregor, John D.. Testing a Software Product Line. Technical Report CMU/SEI SEI-2001-TR-022. http://www.sei.cmu.edu/publications/documents/01.reports/01tr022.html
This is a good introduction to the testing of product lines. It emphasizes traceability to the software architecture.

Mylopoulos, J., Chung L., and Nixon, B., "Representing and using non-functional requirements: a process-oriented approach", IEEE Transactions on Software Engineering, June 1992.

Padmanabhan, Prasanna and Lutz, Robyn, "Tool-Supported Verification of Product Line Requirements," Autom. Softw. Eng. 12(4): 447-465 (2005).

Pohl, Klaus, Bockle, Gunter, and van der Linden, Frank J., "Software Product Line Engineering: Foundations, Principles and Techniques, Springer, 2005.

Schwanke, Robert and Lutz, Robyn, "Experience with Architectural Design of a Modest Product Family," Software Practice and Experience, Vol. 34, No. 13, pp. 1273-1296, 2004.

Stephenson, Zoë, Zhan, Yuan, Clark, John, and McDermid, John, "Test Data Generation for

Product Lines -- A Mutation Testing Approach." In Proceedings of the International Workshop on Software Product Line Testing (SPLiT 2004), pages 13--18. This is a useful paper that addresses how characteristics of the product line (using a Simulink model) can be used as a way of reducing the test data search space. The emphasis is on testing for variant features.

van Lamsweerde, Axel and Letier, Emmanuel, "Handling Obstacles in Goal-Oriented Requirements Engineering," IEEE Trans. Software Eng. 26(10): 978-1005 (2000).

van Ommering, Rob, "Software Reuse in Product Populations," IEEE Transactions on Software Engineering, Vol. 31 , No. 7 (July 2005), pp. 537 – 550.

Weiss, David M. and Chi Tau Robert Lai, Software Product-Line Engineering, Addison-Wesley, 1999.

Weiss, D. and C. Lai, Software Product Line Engineering: A Family-Based Software Development Process. Reading: Addison-Wesley, 1999.

Zelkowitz, Marvin V. and Rus, Ioana, "Defect evolution in a product line environment," Journal of Systems and Software, 70 (1-2 (2004), pp. 143-154. This paper discusses lessons learned from defect analysis across the multiple releases of the Space Shuttle's flight control software. It confirms the importance of testing changed software.

End of File