Configuration Management in a Software Product Line

John D. McGregor
School of Computing
Clemson University
Clemson, SC 29634
johnmc@cs.clemson.edu

Sholom Cohen
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA
sgc@sei.cmu.edu

Abstract

Software product lines offer a comprehensive strategy for successfully building products. This strategy also greatly expands the number and types of artifacts upon which that success depends. Managing these artifacts over the lifetime of the product line requires a carefully structured set of configurations and a rigorously enforced change management process. In this paper we present an approach to configuration management in a software product line organization that has evolved from our work with several product line organizations. The approach, which is added to traditional configuration management practices, managing information, including changes, about assets and products ensuring that they are exactly what they were intended to be. We also briefly describe our technique for interacting with groups to design effective configuration management processes for software product lines.

1 Introduction

A software product line organization manages a set of products throughout development and some, if not all, of their useful life. The organization manages the assets used to build those products and controls how those assets are allowed to change over time. Most development organizations have techniques for managing change within the context of developing a single product. A software product line organization extends this context to manage change within and among multiple projects both concurrently and over time. Further, the product line organization manages a web of dependencies, which exist because assets are shared among multiple products and assets are used to build other assets.

The scale of change management in a software product line organization makes it cost effective to take a comprehensive approach to the life cycles of assets and products. A comprehensive approach includes managing the supply chain of assets from their acquisition or creation to being used by product teams to create products. The approach includes coordinating, via the production plan, a variety of software supply chain activities such as developing an acquisition strategy, using externally available software, and mining existing assets to supply the component development and software system integration activities. It also includes managing those non-code assets that are core to product line operation such as a scope document, a business case, and a concept of operations.

Advances in configuration management (CM) and materialization (MT) tools provide the asset support needed for a software product line. New configuration management tools such as Subversion manage a wider range of information types than traditional source code control systems. This provides tool support for managing a wide range of asset types needed for product line production. Materialization systems such as Buckminster [1] work with a collection of CM repositories to provide a means of managing the complex relationships among requirements, code, test artifacts, documentation, and other artifacts. These systems provide asset management support to assemble components, products, and other required assemblies such as plans and documentation from the specified assets.

The contribution of this paper is the specification of a comprehensive configuration management practice resulting from a thorough analysis of product line operations and experiences with actual product line organizations. The practice defines how change management policies and practices apply to assets and products. The practice provides specific techniques for implementing the policies and procedures for version control, change management, and configuration management of the variety of assets characteristic of a software product line. We extend an existing meta-model, which decomposes the information needed to describe assemblies of code assets, to apply to an entire product line core asset base. The meta-data are manipulated rather than the asset itself eliminating referential problems that accompany strategic levels of reuse.

Our goal was to investigate techniques for implementing a complete configuration management method to support the configuration management practice area of the SEI's Framework for Product Line Practice. The data collected about configuration management practices come from repeated application of general product line diagnostic instruments and from direct engagements with customers concerning the development of configuration management practices.

To communicate clearly about our method, we define the terms shown in Figure 1. The product line is a set of software-intensive systems sharing a common, managed set of features that satisfy specific needs of a particular market or mission, and that are developed from a common set of core assets in a prescribed way [**?**]. A product is some grouping of functionality for which someone is willing to pay. Products are built using assets including code but also requirements descriptions, a software architecture, production plan, tests, and software components. Assets are assembled from, or are collections of, artifacts. An artifact is an atomic piece that may be used in multiple assets. Products, assets, and artifacts are referred to as configuration items (CIs) with respect to the configuration management system. A configuration description (CD) is a specification for a configuration item including perhaps a list of configuration items that constitute the CI. A request links one or more CDs and a means to realize the CDs. The request mentions one or more locators, which specifies the locations of repositories. The name of a asset, given in the request, is used to determine which repository is used. We will use the term repository to mean a collection of items that is controlled by a tool. For example, a configuration management tool could manage a code repository. A text management tool manage a repository for document chapters. We will expand on these definitions later.

Figure 1 also shows that a change request references specific assets or products that are to be changed. This entity is the interface between the information about the managed items and the administrative change process. We will focus on the managed items and tie it to the administrative process without giving details of the administration.

To illustrate the comprehensive CM approach we will use the SEI's Pedagogical Product Line (PPL). AGM, the company used in the PPL example, is adopting the product line strategy in order to launch a new vision for their business. The initial product line will provide a basis for future product lines. Many of the core assets being developed will be used in future product lines so managing their evolution is of strategic importance. AGM is an amalgam of characteristics from several real organizations and the reactions of AGM are consistent with our experience with actual product line organizations.

This paper describes a method of configuration management that embraces version control, change management, and management of asset and product configurations. The method defines processes, tools, and models needed in the configuration management practice in a software product line organization. In the next section we specify the problem. In the following sections we present our approach followed by an example applying the approach to the Pedagogical Product Line.

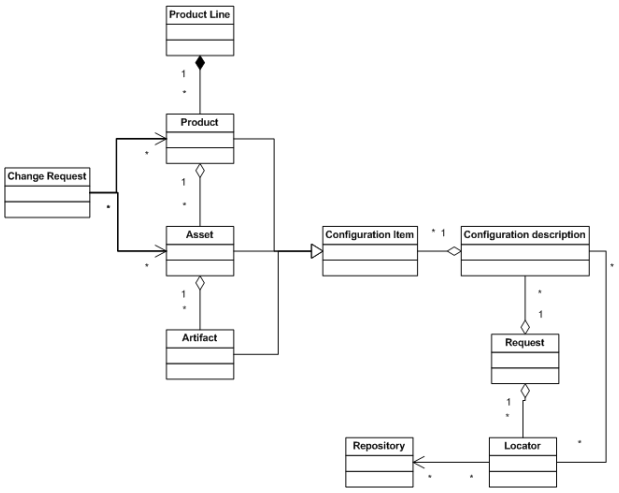

Figure 1. Basic concepts

2 Problem

Software product line organizations need to manage the changing definition and implementation of assets and products over their life time. Requirements change and the impact on the asset base needs to be traced and documented. Implementations change as developers adopt new technologies or defects are removed. These changes must be managed so that concurrent threads of development do not interfere with each other, products are assembled from the correct assets, and over the long term the asset base adds value rather than becoming a liability.

The product line approach has at least three characteristics related to changing assets that are the major drivers for the design of the configuration management method:

- The assets are longer lived than in a traditional single product environment and each asset is used in a variety of contexts resulting in more requests for changes. Many of the assets use variation mechanisms that require parameters that are non-primitive pieces and are subject to configuration themselves. Changes to one portion of the group of artifacts that comprise an asset will often impact other artifacts within the group.

- The assets to be managed go beyond the usual source

code to include plans, architectures, and documentation. These assets have traditionally been handled by a number of stovepiped processes. Many individual product line tasks require the coordinated use of, and changes to, several types of assets.

- The reuse of assets in multiple products creates dependencies among assets and products that propagate change and any implications of those changes. Assets are used to create other assets adding to the web of dependencies.

The exact configuration management method appropriate for a particular product line organization depends on many factors. To place those factors into context we created a product line scenario about AGM, from the PPL, that includes a number of variation points denoted by $< >$. Not all of the variation points will have a major impact on change management, but they are included for completeness.

AGM is a <multi-national corporation> that has software product lines, which <are housed within a single business unit>. The core asset team is <centrally guided> while the product developers are <matrixed to product teams from functional teams>. The product line is a <virtual> unit in the business unit. The core asset team members are <co-located>. The core asset and product teams are <separate entities>.

The PPL software includes <some open source software>. The core assets and product specific assets are managed by <multiple repositories>. Each product development team has a <read-only> copy of the asset base that is updated <hourly>. The assets are validated to <the companys internal> standards.

The assets are expected to remain in use in deployed products for approximately <5> years. Most assets will remain viable for <18 months> without upgrade. Owners of the development of a specific product <can> be forced to accept upgraded versions of the core assets.

Some of these variants that do impact the CM practice include:

- Scope of control – When the product line organization crosses some type of management boundary such as business units there is likely to be a need to access multiple physical storage locations. There may be different rules among the units for managing change. Contractors and suppliers may need access to assets belonging to another unit.

- Life time of assets – If a large percentage of the assets will be changing often, relative to the life of the product line, the configuration management method will need more resources than an asset base that is changing more slowly.

- Distribution of teams – When the organization is widely dispersed, the CM practice will bridge multiple networks, multiple cultures, and multiple time zones.There may be a need for mirror sites with periodic refreshing.

- Separation of core asset and product building – The CM method will facilitate communication between the consumers and producers of assets. The more separate the consumers and producers, the more support needed from the CM method.

- Validation constraints – Regulatory agencies, such as the Federal Aviation Administration (FAA) or the Federal Food and Drug Administration (FDA) in the United States, often have requirements that will impact the amount of traceability that will be required in the CM practice. Quality initiatives such as Six Sigma will also impose requirements.

A successful configuration management system is one that is sufficiently easy to use that it is not noticed but that still ensures the correct asset is provided in answer to a request. In many cases a CM tool will be integrated into other tools to minimize the time required to manage artifacts and assets. For example, the basic Eclipse platform has integrated access to CVS repositories. Managing and versioning development projects is done from the same menus as the development work requiring a minimum of extra work. In the next section, as we describe our approach, we will evaluate against these criteria.

3 A CM Practice

The CM practice for a software product line encompasses version control, change management, and configuration management. The practice includes an adoption process that defines the initial steps of planning how the practice will be carried out in a particular product line and operational processes for day to day operation. The organization makes decisions about what will be managed, how it will be managed, and who will do the management.

The CM practice supports both core asset development and product building. As such it should be sufficient to the job but not overly complex and require the minimum amount of resources to carry out the responsibilities listed in the previous section. An organization that must satisfy a regulatory agency will need a more complex practice than an agile organization that is producing interactive role-playing games.

The product line CM practice requires many of the same processes and responsibilities as a traditional practice but adds significant responsibilities.

- Communication - The CM practice in a product line provides the means of communication between the core asset developers and the product developers. Our experience shows that companies often try to maintain the same communication mechanisms, such as change requests, while also trying to adopt a more agile approach to development. Our technique makes use of open source techniques such as bug reporting tools that automate the communication.

- Change requests – The change control board members for a software product line take on the additional

role of investment advisors. Decisions about adding a variation point or adding a new variant to an existing variation point may be relatively simple to make taken in isolation but taken together these decisions may place the product line on a trajectory toward a long term destination that is not intended or appropriate. Our experience shows that some organizations keep the same CCB approach in which decisions are largely risk based and do not take into account the full scope of the proposed change.

- Evolution plans – Modification decisions require an impact analysis that describes the effect of the change on the asset base and products of the product line from several perspectives including the software architecture. Product line economic models, such as SIMPLE [xxx], can be used to quantify the impact of the changes over time. Our experience shows that few product line organizations have an effective approach to planning for the life cycle of the product line.

- Release plans - The configuration management system supports product releases. Whether there are formal releases annually or almost continuous releases including nightly builds, each publicly available product is a configuration that is versioned and can be recreated at any time. Our experience shows that organizations are not sufficiently comprehensive in what constitutes a release. Important documentation such as the entire test framework for the release is often overlooked because of the cumbersome nature of how such assets are used. Our technique uses references to versions of these assets rather than copying the assets.

Those configuration management practices that are different in a software product line are related to the large amount of reuse and the longer life time of those reusable assets. The configuration management practices should provide a means to resolve the many-to-many relationships that exist between assets and products, and, in some cases, asset to asset. The CM system should also maintain artifacts that provide traceability among these relationships to support the long term sustainment of assets and products.

The CM method encompasses the processes, tools, and models needed to manage configurations of products and assets and to manage changes to those configurations. CM staff will define the method and will design the repository structures. Everyone in the product line organization has responsibilities related to adding artifacts to be managed and to correctly applying the CM method in their other activities.

Since a product can be thought of as an asset, we will describe the processes in terms of assets realizing they can also be applied to products.

3.1 Asset management

The basic units for building products are assets, which in turn may be composed of assets or artifacts. An artifact is usually a fragment that is not usable by itself and must be composed with other artifacts. For example, pieces of code that state pre-conditions might be factored out of test software and reassembled with other pre-conditions in the appropriate assets.

An asset configuration describes all of the artifacts and assets needed to realize that asset. The configuration may reference other configurations. For example, a unit test asset configuration includes test software, test cases, and results and references the configuration of the assets for which it is a test. There may also be documentation assets whose configuration references the asset being documented. Figure 3 illustrates that the configuration for component A is referenced by both the configuration for Tests for Component A and for Documentation for Component A.

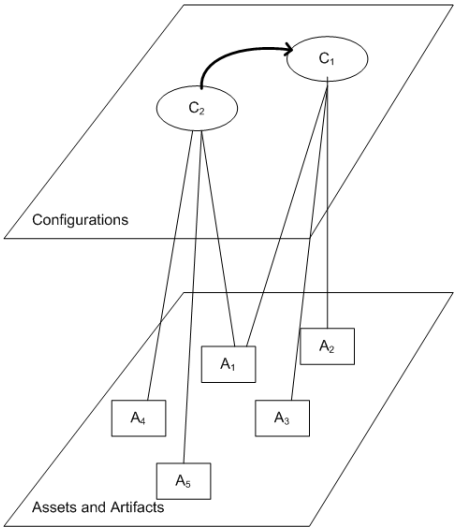

Figure 2. Assets and Asset Configurations

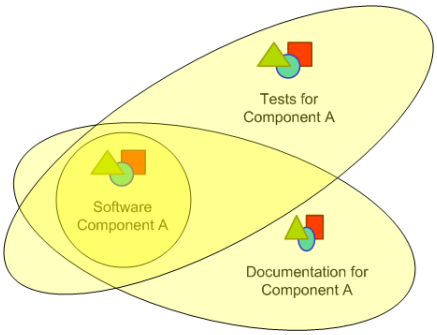

Figure 3. Configurations involving an asset

Physically each artifact under management is stored by a configuration management system in some repository on some storage device. There may be multiple repositories partitioned in one of several ways. The assets may be divided among repositories so that there is a central repository for core assets and multiple repositories for product-specific artifacts. There may be separate repositories for documents such as requirements and source code. There may be a repository at each development site in numerous countries. Mirror copies of the various repositories may be established at sites without adequate bandwidth into the central repositories. These will be refreshed with revised assets on an established schedule that is commensurate with the rhythm of the organization. Some organizations will do this every six weeks while others may do it every hour.

The repositories used to manage assets have a layered design as illustrated in Figure 2. Meta-data is placed in the top layer and basic artifacts and assets are stored in the bottom layer. Each artifact that is to be managed is a configuration item, represented by the rectangles in the bottom layer of Figure 2 and is specified in a configuration description, represented by the circles in the top layer of Figure 2. One configuration may reference another as shown in the arrow between C1 and C2. All of the configuration items and configuration descriptions shown in Figure 2 are stored in the configuration management system. The two layer structure of the repository is useful to separate the storage and versioning concerns of individual assets from the dependencies among assets.

The owner of an asset is responsible for the initial check-in of the asset. The owner checks it in to the appropriate repository. On the initial check-in, the owner is responsible for creating the meta-data entries for the asset as illustrated in Figure 4.

- Each asset is accompanied by a *description*. The description defines a name by which the artifact will be known, a list of dependencies the artifact has on other artifacts, pointers to tools that are used to build, test, or display the artifact.

- A set of artifact *locator* files is maintained in the configuration management system but separate from any of the artifacts or their descriptions. Each locator describes one or more repositories including the type of configuration management software used to manage the repository and the URI of the repository. This decouples the what from the where and how. An existing locator file may already declare a path to the repository so that the owner of the new asset only has to verify that there is a locator file that points to the repository.

- Finally a *request* file is developed that is used to retrieve the artifact from the repository. The request file contains the name of the artifact to be materialized from the repository and a reference to one or more locator files to be used to find the repository. The request is made available to anyone who wishes to use the artifact.

When a developer wishes to use an artifact, they request the artifact using a request file and the artifact is materialized. The request may trigger a number of actions. An asset may be assembled from several artifacts so the assets description may be a description that lists a set of request files that materialize all of the required artifacts and a description of the tools to be invoked to assemble the asset. In this approach the physical location of the artifact is immaterial. Multiple asset description files may all reference the same artifact through its description file. This effectively resolves the many-to-many relationship issue between assets and products from an assembly perspective.

When an artifact is modified and checked in, the version is incremented. It is the responsibility of the person making the change to update the request file, if they refer to a specific version number, to be certain that future requests refer to the correct version. The description and locator files do not have to be updated. The revised request file is published again.

Processes	The organization needs: a planning process to setup the CM infrastructure, a process for asset owners, and a process for asset users.
Models	There are two models that are important: locations of repositories types of relationships among configuration items
Tools	Repository management tools like subversion, specialized database tools such as DOORS, and materialization tools such as XVCL and Buckminster.

3.2 Asset ownership

The asset owner's process can be summarized in a few steps:

1. The owner of a configuration item is responsible for the initial meta-data creation and check-in.

2. Anyone who changes a configuration item is responsible for ensuring that the meta-data is modified appropriately.

3. As needed, reports can be generated by the CM system to show the change history of each configuration item.

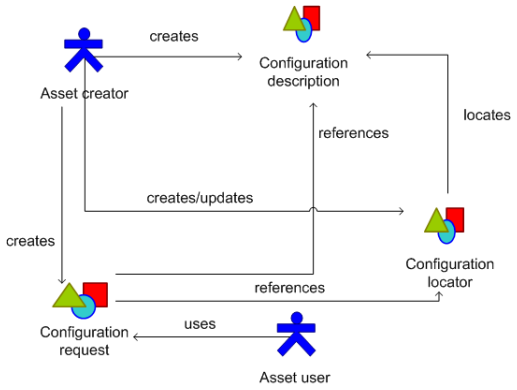

Figure 4. Configuration management process

3.3 Asset use

Assets should be used more often they are modified. Therefore the asset user's process should be optimized. The process includes:

1. Users will materialize the asset for use in a tool. The materialization can be configured to run sanity tests on the asset as it is retrieved if the user wishes.

2. The CM method will need a policy that guides the asset user on how to handle the need for a modification of a core asset. One such policy, intended to maintain the rhythm of the process, would allow the asset user to create a branch in the asset base to make the modification in parallel with submitting a change request. The process should set a time limit by which the modification is accepted by the asset's owner and incorporated into the asset or a new asset is created and added to the asset base. In either case the branch should be merged back into the asset's trunk.

3.4 Implementing the CM practice

Organizations will be at different stages in developing an effective CM method. In prescribing the following activities, we are assuming that the organization has reached at least level 1 or 2 on the SPICE process capability scale, meaning that CM is currently performed informally or is planned and tracked [2]. This process specifies a sequence of activities for planning and initiating a product line CM method.

1. Develop a revised CM method from the existing CM method.

   (a) Initial assessment – Conduct an initial assessment of current CM processes, models and tools. Identify what is working and will scale to the product line. Consider how the current structure of the repositories supports the development goals.

   (b) Commonality and variability – Use the commonality/variability model for the product line to determine appropriate assets. A region of commonality can be stored as a single configuration item. For a variation point there should be a configuration description that covers all possible variant values. Individual variants may be separate version trees if they are maintained by very different groups, such as a contractor, but ties to the variation point by the configuration.

   (c) Develop an asset dependency model – Using the commonality and variability analysis results, identify the types of dependency information needed to support the development process. For example, for an FDA-qualifying development project there is a need to be able to document linkages between assets, the test cases for those assets, the test and change histories and any other information related to verification and validation.

   (d) Develop processes and policies – The planning process answers many process and policy questions. What will be managed? Who will own each managed asset? How will the relationships be identified and captured?

   The answers to these questions depend on the characteristics from the scenario above. FDA compliant companies will manage more and track more relationships than an open source organization. Product line organizations that cut across business units will have a more complicated answer to the question of ownership.

   The processes needed include a change request process that links the product builders with the core asset builders, a configuration item (CI) management process that links owners of CIs with users of CIs, and a management process that supports traceability of the many relationships existing among CIs.

   (e) The typical change management processes are needed. A large product line may have a change control board for the core assets and one for each product or it may be a single unified board. The CONOPS should clearly define responsibilities and escalation paths.

   (f) Select tools – Organizations may have existing repositories of assets. The materialization process should be designed to "sit on top" of the set of repositories. Even if it is largely manual,

scripts can hide the details of retrieval procedures from the variety of repositories.

Repository tools are needed to manage all the necessary types of assets. This may be a general purpose tool such as Subversion or specialized tools such as CVS which only handles text files. The tool may combine development and management activities such as DOORS for requirements.

2. Conduct a pilot study using the new CM process.

   (a) Select an appropriate scope for the study – The product line organization can select a subsystem or other unit that will be developed early. The group selected to use the method first should be a typical development team in that organization.

   (b) Identify goals for the study – The study might investigate different granularities of configuration items or which types of dependencies are useful. To be feasible the study should also produce a production quality subsystem.

   (c) Track exceptions to the policies and procedures – Throughout the study the organization should track the violations of the processes and policies and attempt to understand the implications of those violations. If a violation has no consequence, it may be an unnecessary complication. If there are consequences, capturing these lessons learned and using them in training will strengthen the argument for the policy. The consequences can also guide how the processes are modified.

3. Refine the CM method based on the study.

   (a) The results of the study will help determine where the process is not well-defined or where there is so much flexibility that developers are unsure how to proceed. Some members of the team that used the method should participate in the refinement.

   (b) Make certain that ownership and ownership responsibilities are clearly delineated.

4 Example

The CM method described in this paper was implemented for the AGM pedagogical product line. The configuration items included the standard product line artifacts described in the SEIs Framework for Product Line Practice. In this section we provide examples of the configuration itemss that were identified. The Subversion configuration management product was used as the underlying repository management tool. The meta-data decomposition used was similar to that used by the Buckminster project of the Eclipse Foundation for source code [xxx]. We also decomposed some assets into artifacts and constructed XVCL frames that could be assembled into multiple assets [xxx]. Each basic artifact was the root of a version tree. Some of the artifacts, like the collection of XVCL frames, were complete assets and were located on a single tree. Other artifacts, such as data pools of test data were pieces of assets that could be assembled into several different assets and were setup as separate version trees so that they could be reused with many test cases for many products.

As a first example, the scoreboard feature of the AGM games, its production code and unit test code had been factored into an artifact hierarchy [xxx]. Each source file contained XVCL commands. The code for a specific product could be assembled by providing a set of parameters in an XVCL command file. The appropriate fragments are composed using the XVCL engine. The scoreboard code, including all variants, was managed as one version tree due to the small size and the interdependencies among the code fragments. The Ant build tool was used to invoke the XVCL engine as the set of files was extracted from the repository.

As a second example, for each code module a corresponding test project was created using JUnit. The JUnit project references the Java class definition under test. A configuration for the test class references a configuration for the class under test. This allows a product to be built using the class but without the test code and allows the test environment to be quickly established.

In a third example, the documentation for a product feature is incorporated in the manual of any product that includes the feature. A configuration is created for each feature and the documentation for that feature. XVCL was used to select the correct sections of the users manual depending upon parameter values. We used the RTF format of the manual to divide into features. The appropriate manual can be materialized by providing the appropriate parameters to the XVCL command file.The AGM example illustrated several facts we have stated earlier. The setup of the CM practice in a product line will take considerable resources but will represent substantial savings of time over the life time of the product line. It takes time to capture the meta-data and to plan for the structure of repositories. Once the infrastructure is established, the load on the individual asset creator is very small. Finally, the decomposition of the meta-data is such that the maintenance effort for this information over time is relatively small. One further result of the study was validation that a small set of techniques could be applied across assets. Meta-data descriptions, XVCL command hierarchies, and change histories were implemented in a uniform fashion.

5 Workshop

Product line practice spans a wide variety of system and organization types. To handle this amount of variability we have evolved a style of intervention that is a scenario-driven workshop. It is a repeatable process that can be applied to problems organizations have with many of the practice areas. It both facilitates outside consultants gaining sufficient organization and product line domain knowledge to make effective recommendations and engages the customers in thinking through their problems.

Our initial experience with this style of workshop for production planning was reported at last years SPLC [Chastek 07]. In addition to production planning, this style has been used for testing, variability, and configuration management interventions. Our experience encompasses the health care, defense systems, and telecommunication domains.

Each workshop has essentially three phases. Phase 0 focuses on information gathering and expectation setting. Phase 1 is a face-to-face meeting in which scenarios are generated, prioritized, categorized, and analyzed. Phase 2 is a feedback session in which the results of our analysis are presented.

5.1 Scenarios

Collecting scenarios establishes a context for analyzing the customers current practices and understanding their constraints.

Brainstorming to develop the scenarios can be a very productive exercise for the customer team. Often there has not been a careful and thorough analysis of the problem. In one workshop it became clear to the customer team that their problem was organizational rather than technical.

Analyzing the scenarios provides the opportunity for the consulting team to ask questions, gain domain knowledge, as well as expand the scenario descriptions. For configuration management, the scenario gathering focuses on product line practice patterns in which CM plays a role. For example, we have gathered scenarios that described CM as part of the Each asset pattern that develops individual assets and places them under CM. Once we have generated a significant number of these scenarios, we categorize them into groups such as asset development, product development, or product release support. We also prioritize them for more detailed analysis. In preparing feedback, we consider the state of the organization. If an organization is early in product line adoption, the analysis may produce requirements for their CM method the processes, models, and tools they will need to support CM. For organizations developing their production capability with CM underway, the analysis may point out shortcomings in their CM approach and recommend remediation steps.

5.2 Lessons Learned

1. Customers want immediate and specific solutions to what they perceive as their problems. The phrase surely there must be someone out there doing this, just tell us how they do it is heard often in the early phase of an engagement. Sharing the general context described above with them illustrates the diversity of product line organizations and partially explains the need for consideration of their particular context. Phase 0 of the workshop should include a specific explanation about how the scenario gathering and discussion processes are facilitating the solution process.

2. The participants in the workshop are chosen to represent specific perspectives on a practice. If a particular perspective will not be represented at a point in the workshop, it is better to reschedule the workshop. For example, if the system verification team is participating in a user acceptance test with a customer, scheduling a workshop to develop the product line test strategy will result in a plan that does not adequately address all aspects of testing.

3. The customer may not have adequate practices for even single system development in the specific practice area. The scenarios should be sufficiently comprehensive to describe all activities in a practice area not just build product line practices on top of existing, but deficient, single product practices.

4. The workshop often does not get into detailed solutions but it should result in a concrete action plan. Follow up sessions should be scheduled to ensure that the solution is correctly implemented.

6 Conclusions

Configuration management for a software product line organization manages a large population of diverse artifacts over a long time horizon. The CM practice must ensure that when requested, the correct asset or product is provided to the requestor. The approach described in this paper separates the specification of an asset or product from its implementation. This meta-data resolves the many-to-many relationships between assets and products that exist to support the strategic levels of reuse common in software product lines.

References

[1] Buckminster. http://www.eclipse.org/buckminster, 2008.

[2] SPICE. http://www.sqi.gu.edu.au/spice/title.html, 2008.