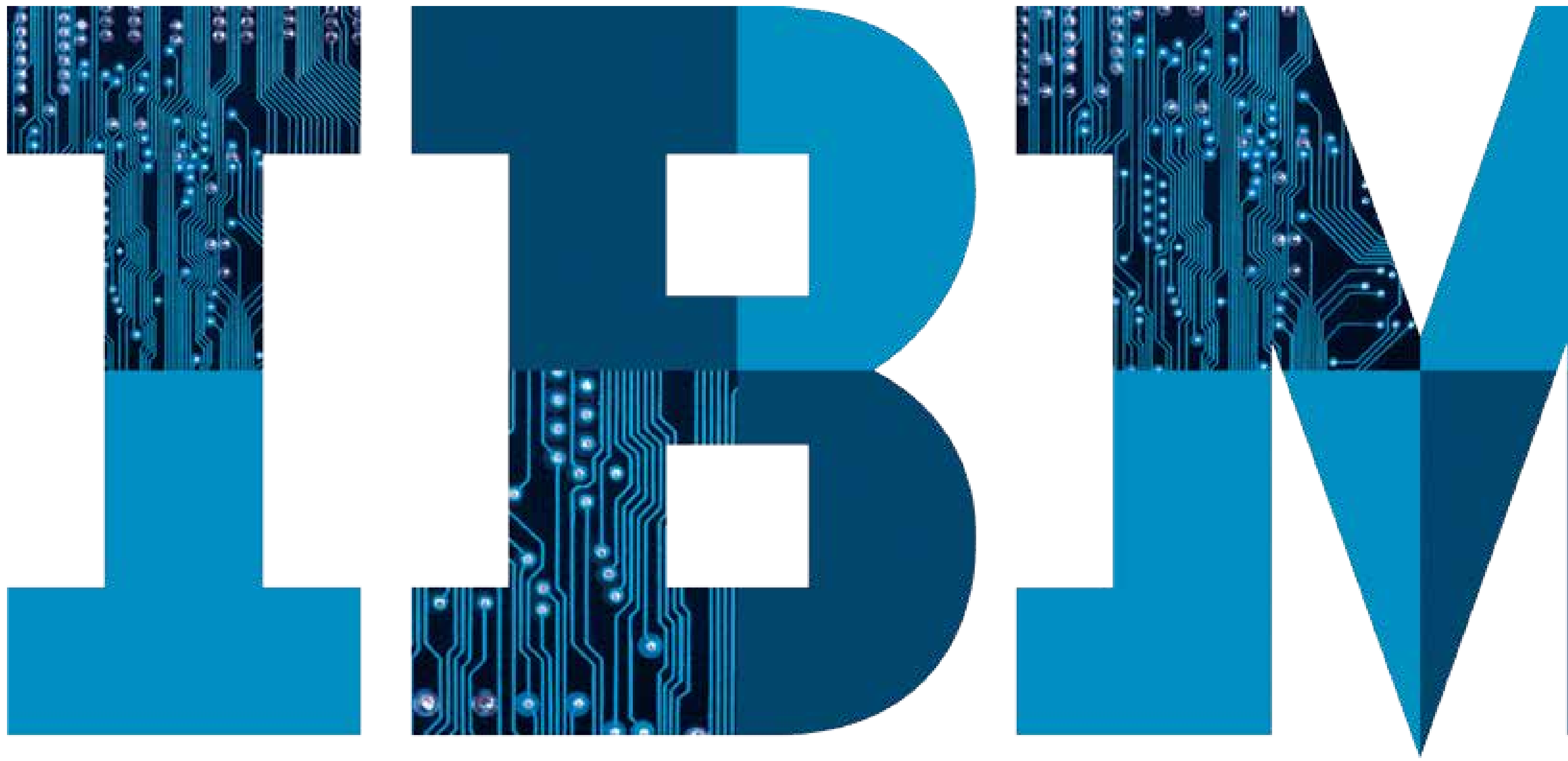
IBM Analytics

Tame complexity with product line engineering

Why strategic reuse is the key to mastering complexity in product development—and how you can use it for simplified, more accurate engineering



Introduction

A little story about modern manufacturing

Since 1948, Acme Manufacturing has been a successful manufacturer of widgets. Customers all over the world use Acme widgets in a variety of applications.

Lately, Acme has been facing increasing design and market pressures. First of all, most of the widget functionality, previously dependent upon mechanical and electrical systems, is now dependent upon software. Acme has never been a “software development” company, but feels as if it must become one.

Secondly, Acme widgets are increasingly connected, not only to each other but to related systems that provide even more functionality to the end user, some of which are barely recognizable to Acme’s traditional core business. These include maintenance and warranty systems, customer relationship systems, and web-based interfaces and smartphones that augment the functionality of Acme products.

Finally, Acme’s markets are no longer homogeneous. In the past, a single widget design could be used worldwide, but now each market demands tailoring and

customization. This results in a proliferation of widget models that is increasingly difficult to control.

Acme engineers are feeling the competitive pressure to meet market demands, but are struggling to adapt to the increased complexity in engineering its widgets to meet the new software and connectedness requirements. But an even bigger problem is the escalating effort it takes to manage all the variants of widgets required to address worldwide market demands.

In fact, if a particular widget model encounters a problem, Acme engineers can't be sure whether the same problem is or is not occurring in other models. Similarly if they fix the problem in one model, it's hard to figure out which other models need the same fix. The amount of engineering data is increasing exponentially as the number of models grows, and keeping track of it all is proving almost impossible.

Acme engineering has notified management that because they can't trace all of their engineering data across all of their widget models, they risk not complying with safety standards. But just as important, their time to market is lagging and their engineering costs are going through the roof. After more than six decades in business, Acme Manufacturing is starting to lose both customers and profitability.

Challenges for product development

Acme Manufacturing is not alone in having to manage unprecedented complexity. Any company developing products for the emerging Internet of Things must manage continuous change within a dynamic environment. Plus, the increase in connectivity places more of a burden on engineering to avoid a similar increase in failures.

Complex engineering products that operate in a complex environment demand new levels of discipline and competence. Engineers cannot afford *not* to rigorously manage requirements to meet new standards, to utilize systems engineering principles to tame complexity, and to verify and validate designs early and often to prevent costly rework later.

Many companies, however, manage engineering data as they would textual documents. They copy and paste an existing design to create a new one, and then modify the new design from that starting point.

Importantly, an engineering data management strategy based on copying can have unintended consequences. Teams struggle to work in parallel on different product models because changes made by one team aren't seen by the other. These invisible changes inhibit collaboration across development teams. Managing change and proving regulatory and standards compliance become monumental tasks that should be straightforward as a product of good engineering practice and information management.

Strategic reuse and product line engineering

Fortunately, there is a way to address these challenges. The idea is simple: create development pathways like the branches of a tree, with each branch representing a new product variant. This strategy allows engineers to support variation and reuse data that is common across product lines. This is similar to the way products are typically sold, with a "base" version that offers lots of options.

With this strategy, changes to the "common" components of the design—which typically can be up to 85 percent of the total—will be reflected in all the product models. The result is that it's much easier to organize what is the same within product models, and to deal efficiently with the differences.

Organizing product development into product lines in order to reuse design data is highly effective. In fact, companies in all industries have been doing it for years. For instance, a bill of material will specify use of a common component (e.g., a screw or a power supply) and indicate instances of that component as appropriate. The idea also extends to software, where various software components were designed to be reused in a variety of software applications.

The discipline, known as product line engineering (PLE), then extends these concepts to all engineering data—including requirements, designs, models, tests and simulations—used throughout the entire product development lifecycle. It helps engineers find and use the right engineering data, no matter what product model they are working on. The goal is to increase the reuse of things that are "known good," resulting in less duplicate effort, less wasted time, fewer errors and reduced overall cost. Another result is improved product quality, since the use of a proven common design allows engineers to focus more effort on the new, variable part of their design.

Why this approach to PLE is better than past approaches

A brief history lesson

The need for efficiency in managing product lines is by no means new; engineers have been exploring different approaches for decades. So before examining PLE in more detail, it's worth considering the strengths and weaknesses of some of the more common historical methods of managing data and component reuse.

The clone-and-own approach

By far the most common approach to reuse is to identify existing products that are the most similar to the desired end product and create copies (or "clones") of their specifications. This establishes a baseline for the next product variant. Engineers on the team proceed to develop and maintain these cloned specifications to suit the specific needs of their particular variant as if it were a standalone product.

There are a number of obvious failings with this cloning approach. For example, if a regulatory standard that is applicable to both of the product variants changes, then the engineering effort required to assess and apply that change must be duplicated to make both products compliant. Also, if a product defect is found in one variant, it would be very easy to overlook the impact on adjacent product lines. Often little or no traceability is retained to the original source data, so there is little chance to track the points of commonality as each product evolves. And common aspects are likely to diverge over time, limiting future reuse. All this can quickly lead to product inconsistencies and wasted engineering effort.

The tag-and-filter approach
Another common method is a manual parametric process, sometimes referred to as the "tag-and-filter" approach. In an attempt to avoid the problems of clone-and-own, product features and product diversity for a product line portfolio are managed as a single dataset. Development teams use attributes and various other metadata to identify ("tag") specifics about which product feature alternatives to include in any particular product.

Although this approach eliminates the need for cloned data, the data structures involved often grow large and unwieldy over time as more products and features are added to the portfolio. Other problems can manifest themselves when points of commonality need to diverge and data structures need to be reworked to account for the change. This often leads to a considerable amount of manual intervention by the engineer.

Managing variants in CRM, MRP/ERP and MRO
Customer relationship management (CRM), manufacturing requirements planning/ enterprise resource planning (MRP/ERP), and maintenance, repair and operations (MRO) systems manage products during one lifecycle stage. As a consequence, tools in each of these domains offer capabilities targeted at managing reuse and variability. In most cases, engineering specifications are represented as high-level data objects (such as spreadsheets and documents) referenced to the manufacturing bill of materials.

However, this coarse level of traceability, combined with each tool offering its own standalone configuration models of the product families, can make it difficult to identify which definition of variability should be regarded as the reference basis.

The IBM approach to PLE

In constructing a next-generation solution for PLE, the overall intent of IBM is to enable engineering teams to improve their degree of reuse in product design by employing reuse practices that make sense for the teams and their industry. In many cases, companies are looking to offer many more product variants in order to capture incremental market share. As a result, they need the development teams to engineer products in ways that more closely align with how the company sells them. In this context, development patterns are of three types: multi-stream, parameterized and feature-driven.

Multi-stream PLE

The multi-stream approach is the fundamental capability that provides a core mechanism to manage product variants across development streams. The essence of this approach is to apply consistent configuration management concepts, similar to those in source control and software configuration management (SCM) systems, across all lifecycle disciplines, based on a federated configuration management framework. This enables management of variation across the spectrum of engineering data including requirements, designs, simulations, tests, configuration data and software.

A branching mechanism is used to create new variants from previously defined products or variants. Each branch, called a “stream,” manages the artifact versions for a variant. In cases where a common platform is maintained, there will also be a stream that manages the common platform artifacts.

Each product variant is a collection of linked lifecycle artifacts—requirements, design models, code, tests and so on. Baselines capture the state of the artifacts at various points in time as they evolve. Each baseline records a particular state for the artifacts in the variant. When a child variant is branched from a parent variant, the child typically starts from a baseline of the parent and then continues with its own timeline of baselines. Immediately after branching, the child stream contains all of the artifacts and links of the parent configuration (reuse by reference), and any changes made in the child are insulated from the parent stream.

It's important to understand how multi-stream is different from "clone and own." Simply making a copy of an artifact—say, a requirement, model or test procedure—means that it evolves by itself. But if that artifact has been branched under configuration management, it remains connected to the original, so changes can be coordinated and its relationships to other artifacts are maintained.

Parameterized PLE

If you do enough multi-stream style reuse, you may end up with lots and lots of variations of engineering artifacts—and it can be hard to see the forest of your product for all the trees of artifact variations. This is where parameterized PLE comes in.

Parameters are used to specify various aspects of the product at a high level and then drive the creation of variations as needed. Parameterized assets consist of optional elements and property values for the parameters, which provide the means to instantiate the asset. Parameter configurations define how to instantiate variants from a product line. This differs from the tag-and-filter approach in that the degree of automation is much more significant and therefore the administrative overheads are much reduced.

Feature-driven PLE

Feature-driven PLE goes a step further by building models of feature combinations from the customer level down. Feature models abstract the variability of the system for external stakeholders, and specify variants based on feature profiles. Features are mapped to parameters so a particular feature profile can drive the feature configurations that can instantiate variants.

Features can be represented in models. The models can be created top-down, so parameters are driven from features, or they can be created bottom-up, so features are mapped to existing parameters. Feature modeling is done by interfacing with popular feature modeling tools that integrate with IBM solutions.

Deeper dive: How the IBM solution works

A collaborative vision with unique contributions

IBM® Rational® Jazz™ is an outgrowth of an IBM initiative for improving collaboration across the software and systems lifecycle. Inspired by the artists who transformed musical expression, Jazz is an initiative to transform software and systems delivery by making it more collaborative, productive and transparent, through integration of information and tasks throughout the development lifecycle. The Jazz initiative consists of three elements: platform, products and community. (https://jazz.net/about/)

As with jazz musicians, collaboration among talented, creative engineers is the result of a common vision, while each person contributes uniquely. Too often engineering teams have difficulty achieving timely and highly productive collaboration because their specialist tools hold their data captive in their own silos. In these instances, teams—and their managers—often turn to manual compilation of information and long, in-person meetings to create the common context the extended team needs to achieve its mission. But there is a better way.

Some general principles

IBM tools, best practices and services help organizations create the connected products at the heart of the Internet of Things. They follow—and enable—a set of general principles for the ideal PLE solution.

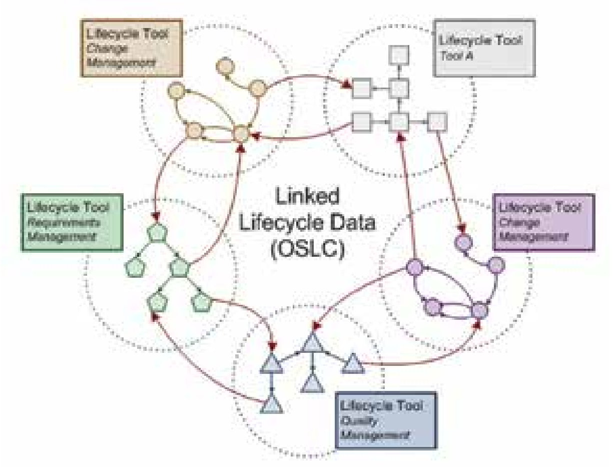
Using multi-vendor tools with multiple data repositories

As demonstrated in the growth of Internet protocols and applications, practical success for large distributed systems comes through building "just enough" integration. Applying this to systems and software development means addressing only the most important integration scenarios and providing simple, flexible capabilities to address new integration needs as they arise.

By contrast, past approaches have attempted to provide a single vendor, single repository solution or massive data synchronization among tools—approaches that have tended to become expensive and long-running application integration projects. The result can be brittle systems and significant challenges in achieving adequate performance, data freshness and the ability to evolve the solution with a practical amount of effort, time and cost.

Managing data using linked data platform principles

The data stays in the tool that created it, and stable URLs provide data access. The data represents engineering artifacts—for example, tasks, defects, plans, requirements, tests, designs and simulations. For more information, please visit: http://www.w3.org/2012/ldp/



Source: Open Services for Lifecycle Collaboration (OSLC) Community

Figure 1: Linked engineering data

Using links to establish relationships among artifacts
Links within and across tools can be used for organizing work, understanding project status and dependencies, analyzing the impact of changes, and generating reports and documents. The links enable an engineer to traverse the engineering web representing a system under development or one that was developed sometime in the past.

Link types provide a way to express particular kinds of relationships. Open Services for Lifecycle Collaboration (OSLC) specifications define various link types, including satisfies/satisfied by, validates/validated by, implements/implemented by, derives/derived from, affects/affected by, blocks/blocked by, link to/link from, and tracks. In addition, some applications define additional link types or provide the means for an administrator to define custom link types.

Other principles include:

- **Data mashups—**Providing views of related data across tools in dashboards, reports and documents. A lightweight approach to data integration reduces the cost of creating and maintaining integrations as well as the load on the system. This approach has the added benefit of enabling views of the data independent of any one tool's frame of reference.

- **Tools providing representational state transfer (REST)ful interfaces—**Allowing other tools and web browsers to interact with their data, and enabling a service-oriented architecture for engineering tools. The interfaces for each tool are discoverable, and one tool needs to know little or nothing about another tool to make use of the other tool's services. For example, when a user hovers over an OSLC link, the users' tool makes a request to the tool owning the artifact at the other end of the link; the other tool returns a preview of more information about that artifact, including the html layout of that information. The user's tool needs to know only how to ask the other tool. This lightweight coupling of tools lowers the cost of building and maintaining integrations.
- **Open specifications—**Managed by the World Wide Web Consortium (W3C) and the Organization for the Advancement of Structured Information Standards (OASIS) to define interfaces and data formats, including those defined in OSLC technical committees. For more information, please visit: http://oasis-open.org
- **Implementation of OSLC specifications—**Enabling any tool or repository to participate. The Eclipse Lyo project provides guidance and a software developer kit (SDK).

Acknowledging exceptions: Linked data is not always the answer

Sometimes it's necessary to transfer data rather than link to it—for example, to exchange requirements with a supplier outside the firewall or to import information that started its life in a spreadsheet or document.

High levels of reuse with greater team efficiency

Tools implement configuration management of the artifacts they manage. Artifacts change over time and tools manage these versions. Teams create and modify sets of artifacts in insulated development streams while baselines provide uneditable snapshots of these streams. Streams and baselines are configurations.

Artifacts can exist in multiple development streams and baselines. They are reused by reference. For example, if Requirement 123 has 10 versions, and Version 6 is used in three streams and five baselines, these streams and baselines point to Version 6 using linked data principles. There is only one copy of Requirement 123 Version 6.

When an engineer selects a stream or baseline to work with, his or her tool loads the right artifacts at the right versions with the right links. It's an automated, deterministic process.

The tools implement other common configuration management operations, for example:

- Branch from baseline to create a new stream
- Compare two configurations
- Deliver changes to a stream

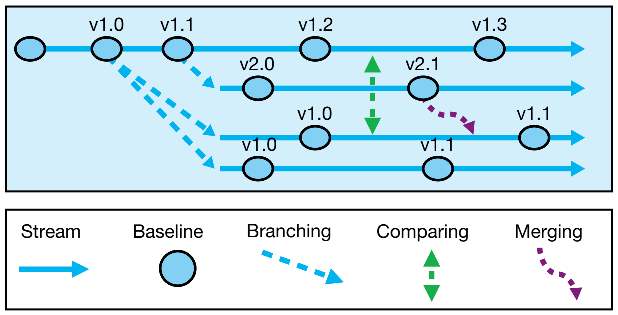


Figure 2: Configuration management operations

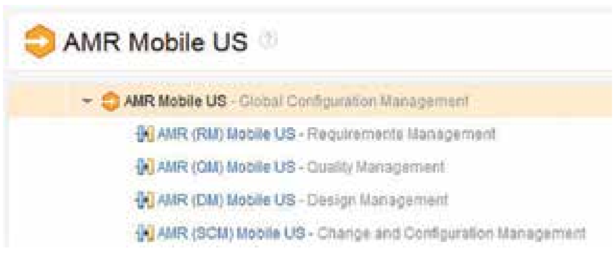
These capabilities also help teams manage change. For example, in the requirements application, changes can be grouped in a change set, which has a meaningful name. These changes can be associated with a change request work item, which has a custom-defined lifecycle. The change set can be delivered to every relevant stream.

Working context across tools with global configurations

IBM Jazz Team Server includes a Global Configuration Management application that organizes configurations of components. A component may include global streams and global baselines, with each global configuration including streams or baselines contributed by other tools that manage requirements, designs, tests, source code or other engineering information.

The links between artifacts are interpreted by the configuration context. For example, when a user is navigating a link from a quality management (QM) tool to the requirement it satisfies, the requirements management (RM) tool selects the specific version of the requirement that is part of the requirements management configuration that is part of the user’s current global configuration.

Note that the tools contributing their streams or baselines to a global configuration may use different data stores (some file-based, some using databases), and it’s possible to include two or more contributions of the same type, for example source code from two source code management systems or multiple requirement contributions.



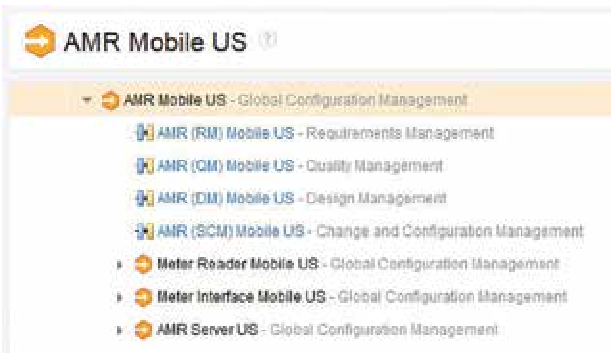
AMR Mobile US

AMR Mobile US - Global Configuration Management
AMR (RM) Mobile US - Requirements Management
AMR (QM) Mobile US - Quality Management
AMR (DM) Mobile US - Design Management
AMR (SCM) Mobile US - Change and Configuration Management

Figure 3: A global stream containing streams from four tools

Defining products, applications and systems with a hierarchy of components

A global configuration can include other global configurations. This enables nested hierarchies of components that reflect the systems, subsystems and components of a product.



AMR Mobile US

AMR Mobile US - Global Configuration Management
AMR (RM) Mobile US - Requirements Management
AMR (QM) Mobile US - Quality Management
AMR (DM) Mobile US - Design Management
AMR (SCM) Mobile US - Change and Configuration Management
Meter Reader Mobile US - Global Configuration Management
Meter Interface Mobile US - Global Configuration Management
AMR Server US - Global Configuration Management

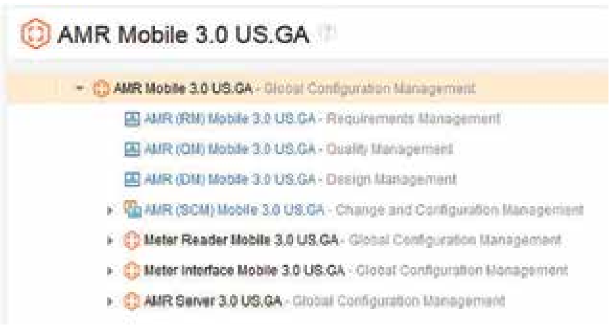
Figure 4: A global stream containing child components

Components in the Global Configuration Management application are the unit of reuse and provide the boundaries of a stream or baseline. Component teams may be working to their own unique release schedule. Product teams using that component can select the component baseline that meets their needs.

This enables high levels of reuse in two dimensions.

- **Changes over time** (temporal variability): When developing the next release of a product, only a subset of the components may change. These components will use newer baselines; the other components can use the same baselines as the last release.
- **Changes for product variants** (functional variability): If a product variant is being

created to address a new price point, geographic market or other market segment characteristic, a global configuration can be branched to represent this new variant. All artifacts and links are reused without copying; insulated changes can then be made in the new development stream as described above, including adding, removing or replacing child global components and streams or baselines from the contributing tools.



AMR Mobile 3.0 US.GA

- AMR Mobile 3.0 US.GA - Global Configuration Management
  - AMR (RM) Mobile 3.0 US.GA - Requirements Management
  - AMR (QM) Mobile 3.0 US.GA - Quality Management
  - AMR (DM) Mobile 3.0 US.GA - Design Management
  - AMR (SCM) Mobile 3.0 US.GA - Change and Configuration Management
  - Meter Reader Mobile 3.0 US.GA - Global Configuration Management
  - Meter Interface Mobile 3.0 US.GA - Global Configuration Management
  - AMR Server 3.0 US.GA - Global Configuration Management

Figure 5: A global baseline for V3.0 of one product variant

User roles

Someone or some group needs to have a deep understanding of the strategy adopted for streams, baselines and component structure. This is the role of the *configuration lead*, who creates and manages the global configuration hierarchy as well as the streams and baselines contributed by each tool. The much larger number of *system engineers* and *application engineers* do not need to understand all these details; they simply select a global stream or baseline in which to work, and their tools show them the right artifacts at the right versions with the right links. Some engineers may have an intermediate set of responsibilities, for example as a *baseline maker*.

Making use of configurations in and across tools

Global configurations are valuable even in their simplest forms. For example, a global stream has two RM contributions: a baseline of system requirements and a stream of subsystem requirements. The engineer can author, review and approve the subsystem requirements, and baseline them. Then the engineer can baseline the global stream. The global stream and baseline provide the context for the “satisfies” links between subsystem requirements and their corresponding system requirements.

Similarly, consider a global stream with a RM baseline and a QM stream. The test engineer develops tests against approved requirements in the baseline. Once tests are approved, they are baselined. Once tests have been completed, the test artifacts and their results are baselined again. The global stream is baselined too. At any time in the future, for example if there is an accident or failure in the field—or an audit—an engineer can go back in time to view the requirements, tests, test results and links between requirements and tests. He or she can run reports or generate documents against exactly the same data that may have been baselined last week, last year or a decade ago.

Another example: Given a global stream representing a development trunk and including some combination of contributing streams of requirements, designs, tests and implementation, teams can use side streams to stabilize changes before completing delivery.

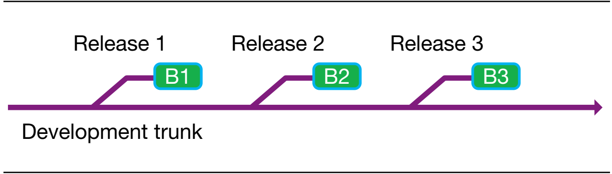


Figure 6: Development trunk and side streams

There are many good reasons to use side streams, for example:

- Making changes to artifacts and links across the system, reviewing and approving them, and then delivering them to the development trunk.
- Importing, reviewing and approving requirements or tests from a supplier before merging them into the main development trunk.
- A/B testing or spike experiments, in which a set of changes may be discarded or become part of the development trunk. The side stream provides a place for development of these artifacts insulated from the development trunk—as well as control over whether and when these changes go to the trunk.
- Representing the various states of the gradual deployment of a product or system. For example, if a company is deploying a new product to a set of customers, each exact configuration of the product can be maintained as a separate variant.
- Representing a particularly complex product, for example an airplane or a locomotive, after it has been manufactured and through the numerous upgrades it receives during its lifetime.

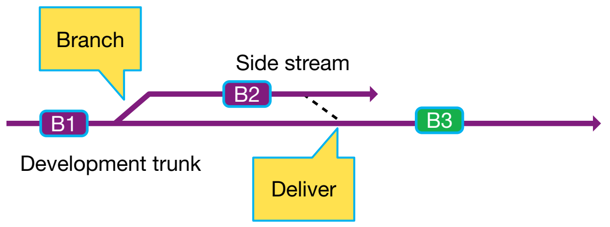


Figure 7: Side stream with delivery back to the development trunk

Complex user scenarios from global configurations

Reuse can become even more complex when working with global development teams and markets.

Branching to create new product variants

A stream representing a development trunk enables sequential development over time. Baselines capture snapshots of the state of the artifacts and their links at specific points in time. Other streams branched from the trunk can represent product variants. Alternatively, one can branch from an existing product variant to create a new variant.

This approach has the advantage that it does not require a lot of up-front planning. Imagine a company's flagship product is selling well in the country, and the company sees an opportunity to introduce the same product in a neighboring country. But regulatory differences require the engineering team to make a small number of changes to a few components of the product. Perhaps the engineering team never expected the company would offer variants of the product, so they didn't create the artifacts as a product line.

The configuration lead can branch the global baseline for the current version of the flagship product and then branch only the included components that need to change. In the future, if a defect is found in the initial product, an engineer can "find where used" to see whether the defect is in other product variants.

This is much easier and more reliable than the previous approach: copying the engineering artifacts for the flagship product, modifying them as needed, and then searching exhaustively through (potentially) many discrete copies. Once the artifacts have been copied, the team loses any relationship to the initial product, and future reuse and impact analysis becomes more difficult.

While this branching approach is simpler to implement and requires relatively little up-front planning, it becomes challenging to manage when there are a lot of product variants. In that scenario, there is no common place holding the definition of the variation in the various products, and variants must be compared to determine where changes made in one stream also need to be delivered.

Using a superset stream to represent product variants
A more sophisticated approach uses a stream as the superset of all potential product versions. Changes are made in this stream, and product variants are derived from this superset stream.

The derivation can be automated by use of requirement attributes, selection of parameter-value pairs or features. In any of these cases, a branched stream can hold the output of the derivation, and a baseline can capture the state after derivation. The generated artifacts can be compared with those in other streams, and then reports and documentation can be generated from these artifacts. They are analogous to software build artifacts: they are read-only and can be regenerated if needed.

Consider feature modeling tools, which work against a superset stream and generate artifacts for a particular product variant based on a set of selected features in a feature profile. The feature model represents the possible variation points and expresses constraints among the features. The feature model is unaware of changes over time (and thus the feature model itself must be version managed). The global configuration identifies all of the components that represent the superset and which version of each superset component to use when generating a particular product variant.

Note that the approaches described are possible when there is one global stream representing an entire product as well as when there are many components in a hierarchy five, 10 or more levels deep.

Some implementation details

Here are some details on how you can implement the IBM PLE solution.

Work items, plans and releases

Work items in the IBM solution are not versioned—they don't need to be. When someone is performing work related to a work item, he or she is doing work on a specific stream—there aren't multiple "versions" of that work. In most cases, work items represent a task or a defect that is addressed and then closed. Work items often reference specific releases, for example, the release (also known as the product version) in which a defect is found or the delivery target for this work.

Work items often do link to a versioned artifact—a requirement, test or design—so they need to be resolved in the context of a configuration. To simplify this for the user, an administrator establishes a mapping between releases defined in the work item system and global streams or baselines. Then, in work items, the user sets the "found in" or "planned for" field (or any other work item attribute that is of "deliverable" type). Then when a user hovers over a link in a work item, or navigates a link from a work item to a versioned artifact, the user sees the right version of the artifact.

Reporting and document generation

A lifecycle index provides a lightweight approach to data integration. Tools provide feeds of change events in the form of OSLC Tracked Resource Sets, which are similar in concept to news feeds. This information is collected in a lifecycle index managed by a Lifecycle Query Engine (LQE). Visualizations, reports and dashboard widgets can be generated from this data, with queries making use of the data and links—independent of the tool that originally provided the data. The visualizations, reports and dashboard widgets typically provide links back to the originating tools, and users can get more information from the originating tool by hovering over the name of an artifact. Similarly, they can click on the hyperlinked name and navigate to the user interface of the originating tool.

Any tool or repository can contribute data to the lifecycle index by implementing a Tracked Resource Set feed. The Eclipse Lyo project provides an SDK.

Additionally, tools offering reportable REST interfaces can be queried by IBM Rational Publishing Engine to generate documents based on a predefined or custom template. IBM tools that use the same runtime as Rational Publishing Engine can generate documents from the same templates. For example, IBM Rational DOORS® Next Generation, IBM Rational Quality Manager and IBM Rational Rhapsody® Design Manager use this technique.

Any tool can participate by providing a data source that can be interpreted by the Rational Publishing Engine.

Enabling configuration management

RM and QM project areas default to configuration management turned off for two primary reasons:

1. Configuration management concepts are inherently more complex than a non-versioned approach. The simpler solution is valued by teams that do not need configuration management capabilities to do their work.
2. Linking behavior is different when using configuration management; links to tools that have not implemented the OASIS OSLC Configuration Management specification may not work as users expect.

Enabling configuration management in RM and QM projects areas is a two-step process:

1. Get a free activation key from www.jazz.net or your IBM support representative. See your product release notes for more information. Then enter the key in an administrative panel.
2. As an administrator, go to the project area properties for each RM and QM project area that you wish to enable, and turn on configuration management.

This is a one-time process. Note that you cannot turn off configuration management for a project area once you have enabled it.

Change and configuration management (CCM) project areas are automatically enabled for configuration management. For the work item system, this includes the release mapping to global configuration as already mentioned. Of course the IBM Rational Team Concert™ source control management is also enabled; it too is delivered in the CCM application.

Additionally design management project areas are automatically enabled for configuration management.

Integrating other tools
Tools can participate in global configurations by implementing the OASIS OSLC Configuration Management specification. To learn more, contact the OSLC Change and Configuration Management Technical Committee at www.oasis-open.org

How to get started

Designing a product line

To help ensure success, there many factors to consider when beginning to implement a PLE solution.

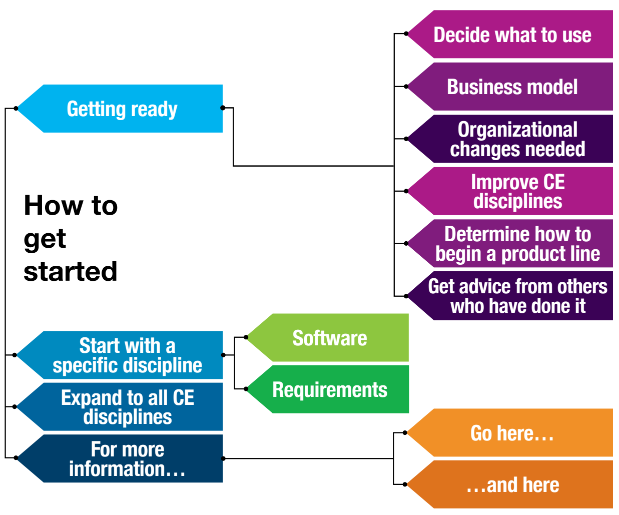

How to get started
Getting ready
Decide what to use
Business model
Organizational changes needed
Improve CE disciplines
Determine how to begin a product line
Get advice from others who have done it
Start with a specific discipline
Expand to all CE disciplines
Software
Requirements
For more information…
Go here…
…and here

Figure 8. Comprehensive planning is the key to successful implementation.

Product lines may be developed either from existing products (reverse engineering) or from new product development ideas (forward engineering). In forward engineering, a core product platform is designed to meet common customer needs, with variation points in mind. Requirements sets are designed based on needed variation points and multiple levels in the product architecture. For example, both a US version of an appliance and a European version are needed, so the common platform contains the elements common to both, while allowing for variation in elements such as power supply subsystem, cabinet size and construction materials.

Reverse engineering a product line from existing products involves choosing which components in an existing product set will be common with other products and which will diverge. Of course, the challenge is in the details. If an organization's products have evolved independently of each other, it is likely that there are differences between even closely corresponding product components. Two products may each contain a 12 volt DC power supply, for example, but there may be differences in interfaces, tolerances, maximum current, weight, physical dimensions or other details. Two kinds of differences must be considered—arbitrary differences and necessary differences.

Arbitrary vs. necessary differences

Arbitrary differences result simply from the fact that different people were involved in each product's design, and for whatever reason, did things a different way. In the absence of an industry, company or de-facto standard, different choices are likely. A vehicle may function just as well with a positive-ground electrical system as with a negative-ground system, but the difference will make sharing common components rather difficult. Arbitrary differences may also result from arbitrary customer choices, that is, the customer required it to be a certain way, but without a clear reason for that choice.

Necessary differences result from the need to meet differing customer requirements, industry or regulatory standards, the use of evolving technologies, or even the changeover from one component supplier to another. Arbitrary differences do not really need to be differences at all, while necessary differences do.

Each difference, whether arbitrary or necessary, results in a variation point in the new product line architecture. In an ideal world, arbitrary differences are eliminated from the product line architecture as existing products are merged into a unified product line, while necessary differences are naturally maintained as variation points. Attempting to engineer a product line while maintaining variation points for all arbitrary differences can add a great deal of unnecessary complexity, reducing the benefits of product line engineering.

Organizational changes

Computer scientist Melvin Conway famously observed that the structure or architecture of a system or product will inevitably mirror the structure of the organization that produced it.[1] If this is true, then it is likely that the move to a product line oriented engineering process will require changes to the organization, since it is likely that the current organization mirrors the current product architecture.

Two business units, each independently producing similar products, will need to decide which components will be common in the new product line architecture, and how these common elements will be produced. Politically, it may be expedient to form a new, third business unit to produce the common components and supply them to the other two.

It may also be wise to consider outsourcing the engineering and production of a common component, especially if it is a generic device and represents no particular engineering value to the organization. It may be that the two organizations are already outsourcing this component, possibly from two different sources, and by making the requirements and specifications common between the two, a single source can be identified. Savvy organizational designers may note that using a combination of an in-house common components group and potential outside vendors introduces a subtle note of competition which may counteract the unfortunate dynamics of in-house monopolies.

Organizational processes and standards may also need revision and convergence so that a unified product architecture can be developed. Verification and testing processes in particular should be examined for optimization.

One of the major benefits of a PLE approach is the reduction in overall testing effort required due to reuse. This benefit can be realized only by designing a verification approach that follows the product line architecture. If two products have 70 percent common components, but are tested as if they shared nothing, no gain is realized during testing.

Integration approaches may also need to change. Integrating common components into common subsystems and verifying these subsystems independently can reduce the total integration and verification effort across products as a whole.

Organizing engineering information

A product line strategy leads to a product line architecture with the organizational changes necessary to produce it. The next step is to examine how engineering information is organized and maintained, and how a product line approach may be implemented.

The most intuitive and obvious way of reusing engineering information within a product line is to simply make a copy of any information that is to be reused in a product variant. With this approach, creating the engineering information for the XL version of a car is as simple as making a copy of the engineering information from the base version and making whatever changes are appropriate for the XL. This clone-and-own approach is simple, but has a number of disadvantages when it comes to long-term management of the product line. The obvious multiplication of largely redundant engineering artifacts results in a multiplication of the number of engineers required to support the product line.

Even more than the additional effort, however, is the resulting inability to propagate changes in core product information to variant products. Making a change to the radar system in a military aircraft requires careful manual changes to all of the variants of that aircraft. Discovering a defect in the Navy version of the aircraft requires similarly manual and error-prone changes to correct this defect in the Army and Marine versions as well as in the core product library.

When engineering information is stored primarily in documents, manual clone-and-own processes are inevitable. However, with the advent of comprehensive engineering information management systems and linked data, much greater control and automation is possible, greatly multiplying the benefits of PLE.

Engineering information is evolving from being stored in documents to being stored in special-purpose databases, such as a requirements database, model database or testing database. As these databases are enhanced to allow the management of versions and variants, engineering information can be effectively shared across a product line and its members. Enabling these databases with linked data capabilities allows the creation of global configurations, representing versioned sets of requirements, designs, models, tests and other engineering information to be combined into a versioned product line member.

Building PLE on a solid foundation

In some organizations, the first step to effective PLE doesn't involve product lines at all. Instead, their focus should be on building competency and discipline in good engineering practices—with tools that help the team implement them. Requirements currently stored in documents or spreadsheets must be brought into a requirements database system, just as models created in a simple drawing tool must be brought into an engineering modeling system.

When all involved disciplines are using such a system, the next step is to enable the linking of these systems. While single-vendor solutions may provide some of the needed capabilities, the ultimate solution will involve multi-vendor standards. Lightweight standards such as the OSLC specifications enable the linkage of engineering information across multiple engineering disciplines, tools and vendors.

Many organizations find that the road to effective PLE begins with requirements management. Structuring requirements information and using a requirements management tool that supports needed PLE capabilities is an important foundation upon which a PLE architecture can be built.

Conclusion

Now imagine

We began our story by talking about how Acme Manufacturing could not keep up with the complexities brought on by the rise of software and connectedness in product development and manufacturing. When we left them, Acme seemed to be spiraling down into a state of non-competitiveness and unprofitability, a sad fate for any manufacturer. But it doesn't have to be that way.

With advances in product lifecycle engineering, we can imagine a brighter future. Imagine a world where there are no disconnected products. Anything more complicated than a hammer is connected to the Internet. Everything is connected to everything.

Science fiction? Perhaps, but much of this is real and some think only a few years away. Your car, your computer, your phone, your home, your doctor, your appliances, your bicycle, your running shoes, even your heart, are all able to provide and consume information.

At work, engineers and their work are connected to marketing, manufacturing, finance, quality assurance, distribution, field service and even the consumer. A trouble

signal on a car speeding down the San Diego freeway signals an engineer in Detroit that this is the fifty-sixth such signal in the last two days and is worth considering as a design change in the future.

What will make this possible is standardization combined with variation and customization. A common platform with endless variations on that theme to meet new needs. Compatibility will be at the level of platform, not individual products. Product line engineering makes it all possible.

For more information
To learn more about IBM solutions for product line engineering, contact your IBM representative or IBM Business Partner, or visit: ibm.com/continuousengineering

Additionally, IBM Global Financing can help you acquire the software capabilities that your business needs in the most cost-effective and strategic way possible. We'll partner with credit-qualified clients to customize a financing solution to suit your business and development goals, enable effective cash management, and improve your total cost of ownership. Fund your critical IT investment and propel your business forward with IBM Global Financing. For more information, visit: ibm.com/financing

[1] Melvin E. Conway, "How Do Committees Invent," *F. D. Thompson Publications, Inc.*, reprinted from Datamation, April 1968. http://www.melconway.com/research/committees.html