Product line management
White paper

Rational. software

IBM

Requirements engineering for systems and software product lines.

Charles W. Krueger, Ph.D., BigLever Software

Professor Ken Jackson, Rational software, IBM Software Group

Ready for IBM Rational software

IBM Business Partner

BigLever Software, Inc.

Contents

Introduction

The key to business success depends on the infusion of new ideas into the way products are brought to the marketplace. To achieve this delivery goal, today's product development organizations must deliver a product line—a portfolio of similar products with variations in features and functions—rather than just one or more individual products. Software product line (SPL) management—or more precisely, systems and software product line engineering and delivery—is a new approach that helps organizations develop, deliver and evolve an entire product line portfolio, through each stage of the development lifecycle, with much higher degrees of efficiency than before.

IBM and BigLever Software (BigLever) have joined forces to provide an innovative and pragmatic new SPL solution that offers organizations the infrastructure, tools, best practices and methods they need to create an advanced and efficient means of production for their systems and SPLs. Similar to what is seen in manufacturing, companies that adopt the SPL approach to engineering and delivery can experience a jump in productivity, quality, time to market and product-line scalability.

The SPL solution from IBM and BigLever includes the BigLever Gears SPL Lifecycle™ Framework, which supports second-generation product line engineering. The BigLever Gears framework integrates existing or new tools, assets and processes across the full system and software development lifecycle. It also comprises IBM Rational® software, which integrates with the BigLever Gears SPL Lifecycle Framework to extend the full systems and software development lifecycle capabilities for the delivery of system and software product lines. The tools can include IBM Rational DOORS®, IBM Rational Rhapsody®, IBM Rational ClearCase®, IBM Rational Synergy, IBM Rational Team Concert™ and IBM Rational Quality Manager software.

Highlights

The Rational DOORS/BigLever Gears Bridge solution allows you to create a single, consolidated collection of requirements for the entire portfolio.

Additionally, the SPL solution from IBM and BigLever includes the IBM Rational DOORS/BigLever Software Gears Bridge integration. The Rational DOORS/BigLever Gears Bridge product extends the requirements engineering capabilities of Rational DOORS software to support the requirements development of a product line. The Rational DOORS/BigLever Gears Bridge solution allows you to create a single, consolidated collection of requirements for the entire portfolio. The solution maintains just one copy of common requirements that are shared across all products while allowing requirements diversity to be expressed using explicit variation points. As a result, you can manage requirements that need to differ from one product to another.

A typical project has requirements at several levels: stakeholder requirements at the top level, system requirements at the second level and potentially more levels depending on the scale and complexity of the systems developed.

Requirements engineering for single systems

A typical project has requirements at several levels. The top level is a set of stakeholder requirements that expresses product requirements using a problem domain perspective. These should be completely independent of any proposed solution. System requirements are at the second level, where an abstract solution is postulated but the solution still allows many different possible implementations. In other words, these system requirements should be as abstract as possible and should not preempt the final design solution. At the third level are subsystem requirements. The number of levels below this level depends on the scale and complexity of the systems developed.

Highlights

Requirements at each level must exist in conjunction with a means of demonstrating that the developed product has met its requirements.

Figure 1 shows a typical model for the top three levels of requirements. It also shows that requirements at each level must exist in conjunction with a means of demonstrating that the developed product has met its requirements. At each level, the model aligns requirements with tests, which help ensure that the requirements have been satisfied. These tests have a "qualifies" relationship with the appropriate requirements, while the requirements at each level satisfy the requirements at the levels above and below.

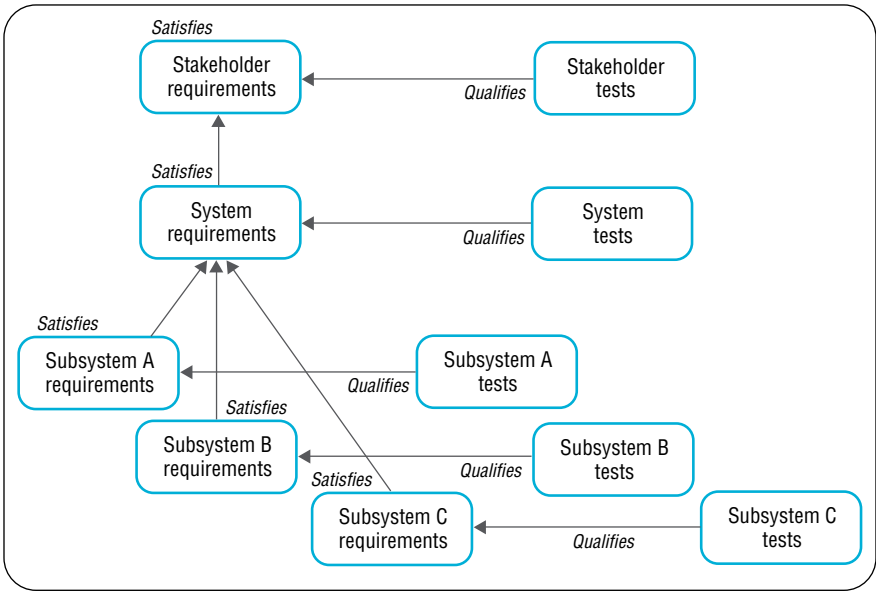

Figure 1: Top-level project requirements

Rational DOORS software helps organizations develop and maintain requirements for product lines.

Conventional approaches to managing product line requirements

For more than a decade, many organizations have used Rational DOORS software to develop and maintain requirements for product lines. The two most prevalent ways of using Rational DOORS software to create and manage requirements for a product line are in the next two sections.

Highlights

Traditionally, organizations have used Rational DOORS to support a clone-and-own approach to developing product line requirements.

Clone-and-own

The clone-and-own approach starts by copying all the Rational DOORS assets for an existing product into a separate folder or project. This is the cloning action.

The team responsible for this new variant then assumes ownership of the cloned version and makes the changes that are necessary to create the new product variant. These changes are valid only for the new product variant: they will not affect the original product from which the variant was cloned.

Figure 2 shows an example of two new products—product Q and product R—being created from the assets of product P.

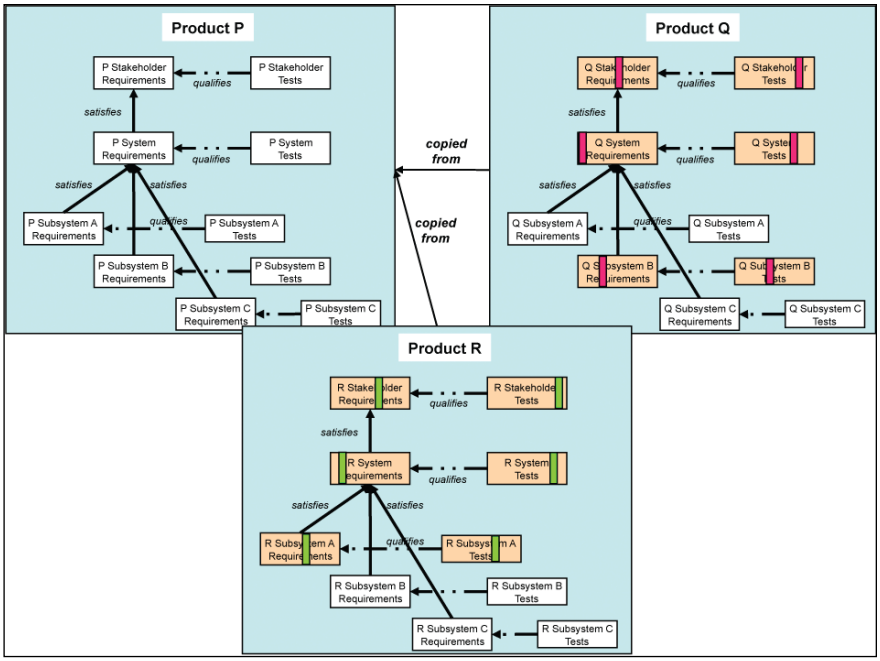

Figure 2: The clone-and-own approach

Highlights

Product Q requires changes to the stakeholder requirements and consequential changes to the systems requirements, which in turn change requirements of subsystem B. There are also consequential changes to the tests for each changed module.

Product R needs different changes to the stakeholder requirements and consequential changes to the systems requirements, which in turn change requirements for subsystem A instead of subsystem B.

The orange modules in figure 2 indicate the modules that must change to create products Q and R. The white modules are unchanged from the original product P. Although the diagram suggests that the changed modules are completely different, in practice, there are a relatively low number of changes compared to the overall number of requirements present. The percent of changed requirements is represented by the red rectangles in product Q and the green rectangles in product R. The unchanged requirements—represented by the rest of the orange module—are common requirements that are reused across the products.

The clone-and-own approach can be bogged down by information redundancy.

The main problem with this approach is that there is much information redundancy. The more products there are, the more copies will need to be made, and consequently the redundancy increases.

Highlights

The problem with redundancy is that it makes it harder to homogenize common requirements: as the number of products increases, so does the difficulty of ensuring that the common requirements are kept in step.

Since disk space is relatively cheap these days, multiple copies do not pose a problem. The real problem is keeping the common requirements up-to-date. Often a common requirement must be changed across the entire current product line. As the number of products increases, so does the overhead of ensuring that the common requirements are kept in step. Sometimes there are many changes to many common requirements at differing levels throughout the product variants' requirements assets—making the updates more time consuming and complicated.

The problem can be further exacerbated when there are multiple versions of each product variant. In this case, it is necessary to determine which versions of the products will keep the old requirements and which will incorporate the new ones.

Attributes

Another way of using Rational DOORS software to derive a new product from an existing product line is to use attributes.

Another way of using Rational DOORS software to derive a new product from an existing product line is to use attributes. There are several ways of doing this. You can create a multivalued attribute that contains the names of all products and sometimes versions of products. Then you tag the requirements in each module to indicate the products for which each requirement is valid. The number of enumeration values can become very large, making it quite difficult to ensure that the requirements are tagged correctly.

Another way of using attributes that avoids the large number of enumeration values is to use a Boolean attribute for each product. Developers set the attribute value to true if the requirement is in a product and false if it is not. With this approach, it is much easier to see the requirements that are relevant for each product, and it is possible to create views based on Rational DOORS filters to view the requirements for a specific product or to compare a set of products. However, there is still the problem of multiple product variants. Here, instead of having a large number of enumeration values, one has a large number of attributes.

Highlights

The clone-and-own and attributes approaches do not scale well to support the complex product line requirement sets found in products today.

It is not easy to see which requirements are common to all products using either attributes approach. Introducing a new common requirement entails setting it as valid for all products, which is both time-consuming and error-prone.

Conclusion on using conventional approaches to manage product variants

For small numbers of requirements, products and product variants, the product line requirements management methods listed above may be sufficient. However, they do not scale well to support the typical product line requirements sets found today. More important, the tactical requirements-engineering challenges are large enough to inhibit a company's ability to expand the scale and scope of its portfolio, take advantage of strategic marketplace windows, competitively price products and optimize product quality.

At the heart of the problem is the product-centric perspective adopted by companies. This is illustrated in figure 3. The vertical blue bars highlight the product-centric focus on the development lifecycle of the individual products in a product line. The red lines illustrate the complex, tangled and labor-intensive interactions, dependencies and coordination activities required to take advantage of what is common and to manage all the variations among the similar products in the product line portfolio.

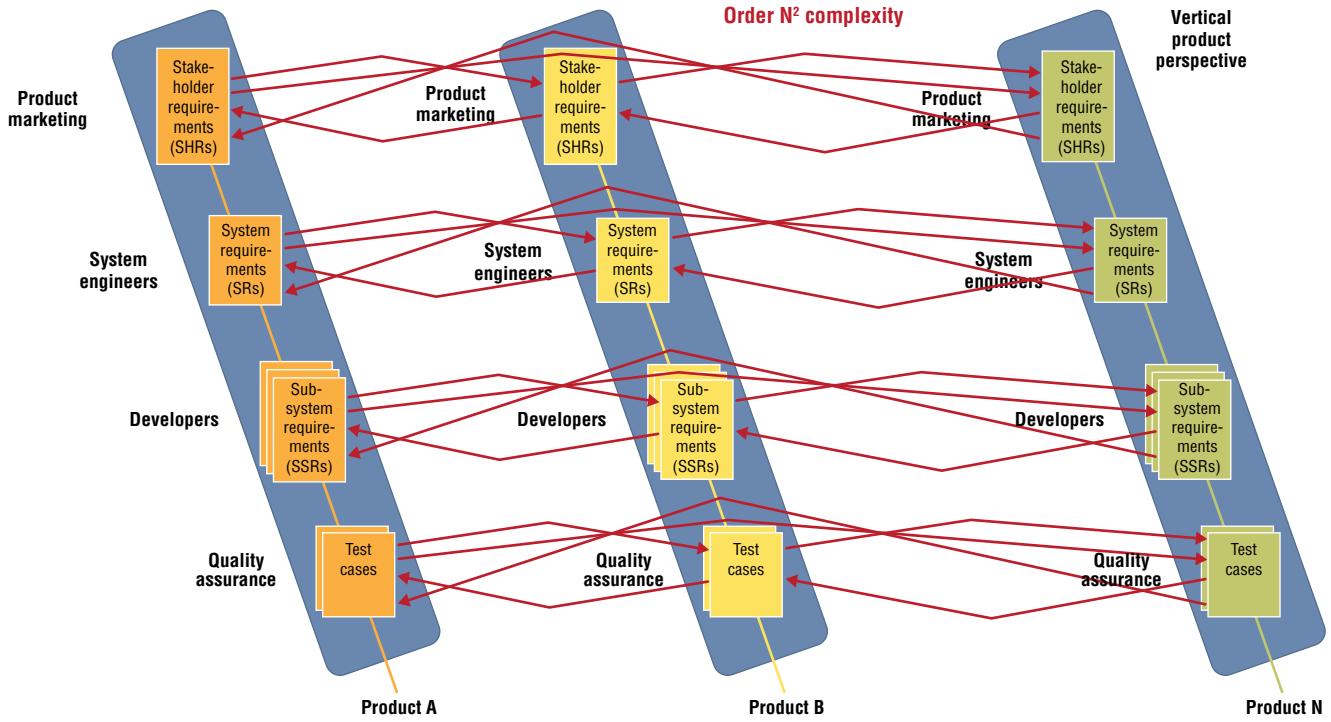

Figure 3: Complex interdependencies from the product-centric perspective

The crux of the problem lies in the fact that the number of red "interdependency" lines grows by the square of the number of products in the product line. That is, each of the N products illustrated in the product line have interdependencies with the other N-1 products, so the total number of interdependency lines is proportional to N×(N-1), or N^2-N. This explains why engineering complexity and effort increase exponentially faster than the growth of a product line. Making matters worse, the conventional product-centric traceability relationships between the different stages of the lifecycle for an individual product interact with the product interdependency relationships, multiplying complexity and introducing dissonance across the stages of the lifecycle.

Highlights

The second-generation product line approach is a new development paradigm in which a team addresses a complete product line.

It is more effective to view product line engineering as creating a means of production rather than viewing it as creating a multitude of interrelated products.

These tactical development challenges have led to the search for an alternative paradigm that directly tackles the complexity issue to provide a more efficient and scalable approach to the development of product lines. The rest of this paper describes a new approach: second-generation systems and software product line development.

The second-generation product line approach from IBM and BigLever

The second-generation product line approach is a new development paradigm. In the conventional product line approach, each product is worked on by a team devoted to that product. In the second-generation product line approach, a team addresses a complete product line.

> *"We can't solve problems by using the same thinking we used when we created them."*
>
> —Albert Einstein

A shift in perspective to an efficient means of production

Organizations mired in the complexity, inefficiency and pains of product-centric product line engineering experience a second-generation product line epiphany when a shift in perspective reveals a simpler solution. Analogous to engineering a product line of hard goods, it is much more effective to view product line engineering as creating a means of production—a single system capable of producing all of the products in a product line—rather than view product line engineering as creating a multitude of interrelated products. The powerful, though subtle, essence of the second-generation product line epiphany is the focus on that singular means of production rather than on a multitude of products.

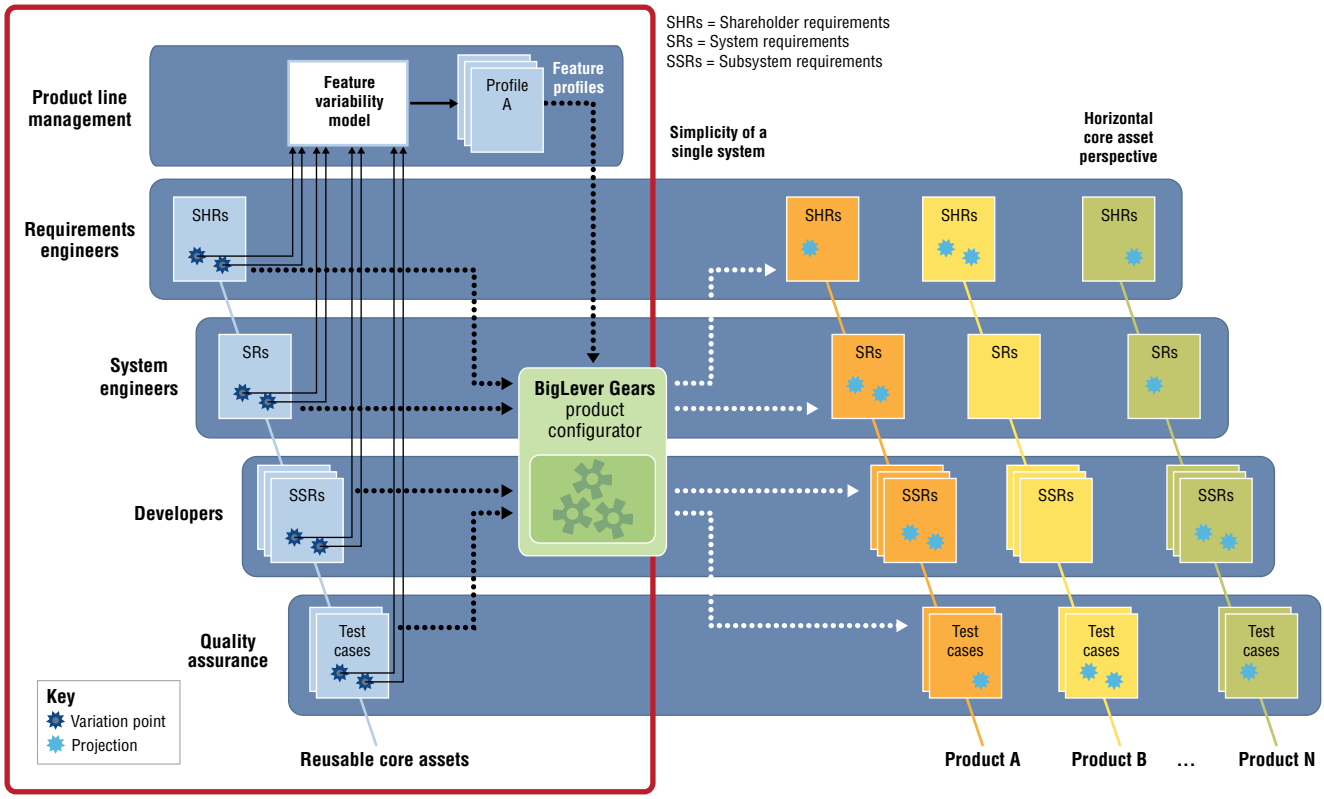

Figure 4: Change of perspective

Figure 4 shows the single-system perspective for producing the same product line as in figure 3.

Highlights

However, now the focus is on the means of production, which is inside the red box. The same products, A through N (on the right side of the diagram), are automatically produced by a singular means of production comprising:

- *Feature variability model (top left), which describes optional and variable features for the products in a product line.*
- *Feature profiles (top right), which describe how each product in a product line is uniquely defined in terms of choices for each of the optional and variable features.*
- *Reusable second-generation product line assets (left), which contain variation points with logic that references the features in the feature variability model. These variation points indicate the circumstances under which any asset—such as a requirement, architecture, model and design, source code component, test case, and document—will be included, excluded or modified, depending on features selected in a feature profile.*
- *Second-generation product line product configurator (center right), which automates the composition and configuration of products from the reusable second-generation product line assets, using the feature profiles to interpret the variation points in the assets.*

By shifting perspective to focus on the singular means of production rather than the multitude of products, you can dramatically reduce the complexities of managing product interdependencies.

As highlighted in figure 4, the blue bars provide the critical shift in perspective from the vertical product-centric focus of figure 3 to the horizontal single-system focus in figure 4. By shifting perspective to focus on the singular means of production rather than the multitude of products, you can dramatically reduce the complexities of managing product interdependencies. Moreover, automated production can result in dramatic increases in the number of products that you can effectively create, deploy and maintain. And with the single-system perspective, you can base the scale and the scope of diversity of a product line on business opportunities and profitability—rather than being constrained by the complex limitations imposed by the product-centric perspective.

Highlights

In moving from a product-centric way of working, we have introduced an important layer of abstraction that is the feature variability model, which maps assets to features instead of products.

A very important aspect of the shift in perspective indicated in figure 4 is the introduction of product line management. This is where the marketing department identifies the nature and extent of the required feature variability and defines future products using a feature profile. From an efficiency point of view, it is vital to realize that, in moving from a product-centric way of working, we have introduced an important layer of abstraction that is the feature variability model. The assets are then mapped, not to individual products, but to specific features. This means that, once a mapping has been created in the assets for any feature, then that feature can be configured in any future product at zero cost of development. This is where a huge savings in effort occurs, producing a massive improvement in business efficiency and flexibility.

Using the BigLever Gears SPL Lifecycle Framework, organizations can plan, develop, deploy and evolve product line portfolios across the development and delivery lifecycle.

BigLever Gears SPL Lifecycle Framework

BigLever Gears SPL Lifecycle Framework provides an automated means of production for your product line. By focusing your business and engineering teams on the operation of your production line, your organization can plan, develop, deploy and evolve your product line portfolio, seamlessly and efficiently, across the development and delivery lifecycle.

The framework enables the second-generation product line integration of existing or new tools, assets and processes across the system and software development lifecycle. It provides a common set of industry-standard second-generation product line concepts and constructs for virtually all tools and assets, including:

- ***A feature model**, which is designed to uniformly express the full product line feature diversity for all assets in all stages of the system and software development lifecycle.*
- ***A single variation point mechanism**, which is designed to be uniformly applied to all tools and their associated assets in all stages of the system and software development lifecycle.*
- ***An automated product configurator**, which can automate—with the push of a button—the assembly and configuration of assets from each stage of the lifecycle to help produce all products in a product line.*

Highlights

The feature model of the BigLever Gears SPL Lifecycle Framework enables you to model features in terms of one or more production lines.

The feature model of the BigLever Gears SPL Lifecycle Framework enables you to model features in terms of one or more production lines. Figure 5 shows how the framework supports interrelated production lines in the overall production line for a midsize family car. This production line refers to an engine production line and a telematics production line, which, in turn, refers to three lower-level production lines: driver information, entertainment and navigation.

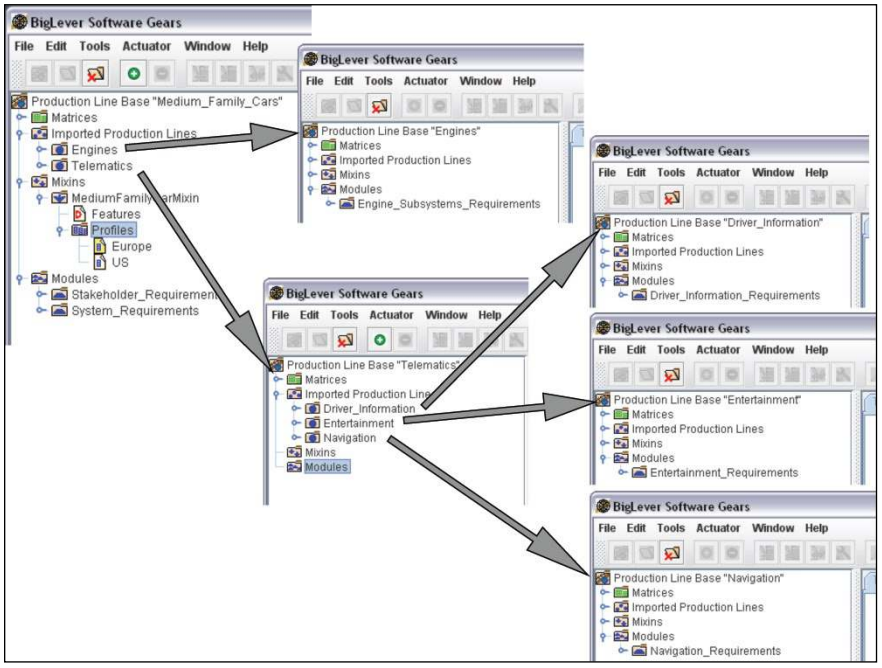

Figure 5: A hierarchy of production lines from the BigLever Gears SPL Lifecycle Framework

Highlights

BigLever Gears production lines include a set of modules, a set of mix-ins, a set of imported production lines, a set of matrices that define specific product instances, assertions, features and feature profiles.

The following sections briefly describe BigLever Gears production lines and their key elements.

Production lines

Each production line has the following potential elements:

- *A set of modules that:*
  - *May have private features*
  - *Reference a core asset, such as a Rational DOORS module, a Rational Rhapsody model, software source code or documentation*
  - *Include a set of feature profiles*
- *A set of mix-ins that also provides features and profiles that:*
  - *Are common to all the modules within the production line*
  - *Can be used in other production lines*
- *A set of imported production lines*
- *A set of matrices that define specific product instances*

Features

A feature can be:

- *Optional—it is either selected or rejected.*
- *Enumerated—it is one (and only one) of an enumerated set of options.*
- *Set—it is zero, one or more of an enumerated set of options.*

Assertions

Assertions can be added to features to signify interdependence between them. For example:

- *If feature A is selected, then feature B must also be selected.*
- *If feature C is selected, then feature D cannot be selected.*

Assertions are logic statements and can be of arbitrary complexity and can reference any features that are in scope. This includes local features (in a module or a mix-ins set) and features in local mix-ins or mix-ins imported from other production lines.

Highlights

BigLever feature profiles provide a way of reducing combinatorial complexity.

Feature profiles

A feature profile is a partial product that identifies valid subsets of all the possibilities offered in a set of features. A simple example of a set of production lines—based on the hierarchy shown in figure 5—has 15 feature definitions spread across six related production lines. Each feature set has no more than six profiles, so it is quite easy to comprehend. However, the BigLever Gears software calculated the total number of possible combinations of these 15 features to be 9,408. This means that even a limited set of 15 features could yield 9,408 different potential products. Therefore, feature profiles provide a way of reducing combinatorial complexity.

By allowing production lines to be interrelated, you have a means of scoping the overall system into areas of specific domain knowledge while still allowing information to be shared among the production lines as necessary.

Matrices define finished products made up from lower-level components.

Matrices

The matrices define finished products made up from lower-level components. They have rows for each specified product and columns in which the user selects the relevant options for each part of the system. The relevant options are defined by:

- *The profiles for each module in the local production line.*
- *The product names in the matrices for each imported production line.*

Highlights

Variability expressed in the BigLever Gears SPL Lifecycle Framework production lines maps into core assets by introducing variation points into those assets.

The Rational DOORS/BigLever Gears Bridge solution

The variability expressed in the BigLever Gears SPL Lifecycle Framework production lines is mapped into the core assets by introducing variation points into those assets. In Rational DOORS software, any object can be made into one of three types of variation points:

- ***Optional***—*The object and its children are included or excluded depending on the variation point logic statement, which is held in Gears Logic—a BigLever Gears–defined attribute—and created by the Rational DOORS/ BigLever Gears Bridge solution.*
- ***Variant***—*Variation point logic defines which variant in a set of variants will be selected by the configurator when it builds a product based on a product profile. Gears Variant—a BigLever Gears–defined attribute—defines the variants.*
- ***Text substitution***—*Variation point logic defines a pattern-based substitution, usually in the Rational DOORS main column. You can also use this type of variation point to load information into attributes instead of the main column.*

The Rational DOORS/BigLever Gears Bridge solution provides access to a BigLever Gears editor for the variation point logic that can reference features that are in scope for the module concerned.

The BigLever Gears development methodology encourages developers to work within the context of a product line and therefore to be aware of all the possibilities that exist for a module.

The BigLever Gears development methodology encourages developers to work within the context of a product line and therefore to be aware of all the possibilities that exist for a module. BigLever Gears software provides a way to view the results of applying the variation point logic to a module via an actuation tool. This tool can be invoked from within BigLever Gears or Rational DOORS software. In both cases, BigLever Gears software enables you to select which particular product to use as the basis for the projection. The pick list is derived from the relevant production line matrix.

Highlights

Rational DOORS and BigLever Gears software provide organizations with a holistic view into second-generation product line management.

When invoked from within Rational DOORS software, the actuation shows the effects of the actuation on that module. When activated from within BigLever Gears software, the actuation visits all the modules connected to all the relevant production lines and applies the variation point logic to each module. The visited modules can be Rational DOORS requirements modules, Rational Rhapsody models, Rational Quality Manager test cases, documentation or source code.

The result of activation within Rational DOORS software is shown in the BigLever Gears main projection column—which shows where objects have been removed (optional variation points), which variant has been selected (variant variation points) or the text that results from text substitution. This main projection view is ideal for checking whether the variation point logic is behaving correctly.

However, it is sometimes necessary to provide a comprehensive Rational DOORS module that contains only the projection for a specific product. You can accomplish this with BigLever Gears software by activating the product in a separate Rational DOORS module from which all the BigLever Gears information has been removed. In this module, the main projection goes into the Rational DOORS main column. Any other attributes that are in the projection are shown in separate columns using attributes of the correct type.

Highlights

With the Rational DOORS/BigLever Gears Bridge solution, requirements engineers can efficiently and effectively manage requirements for an entire product line.

The solution enables you to create a single, consolidated collection of requirements for the entire portfolio.

It provides a reduced level of complexity that enables you to quickly and efficiently deliver more new products and features while reducing the development effort and optimizing product quality.

Conclusions

With the Rational DOORS/BigLever Gears Bridge solution, requirements engineers can efficiently and effectively manage requirements for an entire product line. You can realize the advantages and strategic benefits of second-generation product line engineering—increased productivity, higher-quality products, faster time to market and increased portfolio scalability—in the requirements stage of the SPL development lifecycle.

The Rational DOORS/BigLever Gears Bridge solution enables you to create a single, consolidated collection of requirements for the entire portfolio, maintaining just one copy of common requirements that are shared across all products. It also allows you to express requirements diversity using explicit variation points, thereby facilitating the management of requirements that differ from one product to another.

Rational DOORS requirements in a BigLever Gears software production line are treated just like other assets, in which a superset of requirements for a full product line is selected and configured to create subsets of requirements for different product instances. Using Rational DOORS modules as first-class second-generation product line reusable assets, you can now automate the configuration of different instantiations of product requirements without a clone-and-own approach and linking among requirements sets. You can express the full feature diversity in requirements without the need for low-level attribute scripting or one-size-fits-all requirements.

This reduced level of complexity enables you to quickly and efficiently deliver more new products and features, while reducing the development effort and optimizing product quality.

For more information

To learn more about how you can bring innovative products to the marketplace faster using IBM Rational DOORS and BigLever Gears software, contact your local IBM sales representative or IBM Business Partner, or visit:

ibm.com/software/rational/partners/biglever/requirements.html

ibm.com/software/awdtools/doors/

© Copyright IBM Corporation
© Copyright BigLever Software, Inc.

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

BigLever Software, Inc.
10500 Laurel Hill Cove
Austin, TX 78730
U.S.A.

Produced in the United States of America

All Rights Reserved

IBM, the IBM logo, ibm.com, Rational, and DOORS are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at **ibm.com**/legal/copytrade.shtml

BigLever Software Gears is protected by U.S. Patent No. 7,543,269 and Patents Pending.

Other company, product, or service names may be trademarks or service marks of others.

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates. The information contained in this documentation is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this documentation, it is provided "as is" without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this documentation or any other documentation. Nothing contained in this documentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM (or its suppliers or licensors), or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

Any performance data for IBM and non-IBM products and services contained in this document was derived under specific operating and environmental conditions. The actual results obtained by any party implementing such products or services will depend on a large number of factors specific to such party's operating environment and may vary significantly. IBM makes no representation that these results can be expected or obtained in any implementation of any such products or services.

Any material included in this document with regard to third parties is based on information obtained from such parties. No effort has been made to independently verify the accuracy of the information. This document does not constitute an expressed or implied recommendation or endorsement by IBM of any third-party product or service.

RAW14186-USEN-00