プロダクトライン開発セミナー

pure::variants

バリアント/バライアビリティ管理ツールと、ユーザ事例

FUJI SETSUBI

© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

■プロダクトライン開発セミナー　：バリアント/バライアビリティ管理ツールと、ユーザ事例

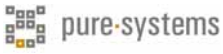
◇日時：2008年2月8日(金) 午後1時00分～4時45分　◇参加費無料　◇会場：秋葉原コンベンションホール（5Ｆ）

■市場投入までの時間を短縮し、製品の品質を上げる。
プロダクトライン開発は、再利用技術に基づいた、製品ファミリー全体の戦略的、体系的な開発です。その実現のためには、製品を共通性と変動要素に分類し、要求、設計、ソフトウェア部品、テストにまで至るコア資産として管理することが必要となります。とりわけ、製品ファミリーのライフサイクルに渡って、バリアントとバライアビリティを分類し、管理することで、再利用される資産は効率良く開発され、メンテナンスできるようになります。

本セミナーでは、プロダクトライン開発の概念を、バリアント/バライアビリティの管理に着目して紹介し、要件管理、構成管理、デザイン、テストなど各種ツールとの、実践的な連携についても説明します。また、ユーザ事例を紹介し、現実的な取組みとして、現行の製品開発に於いてレガシーコードを、どの様にしてバリアントとバライアビリティに分類・管理し、体系的再利用としてのプロダクトライン開発を進めていくことができるかという点にも触れます。

Software Product Lines and pure::variants

Danilo Beuche
danilo.beuche@pure-systems.com



© pure-systems GmbH 2007

目次

- ソフトウェアプロダクトラインについて
- pure::variantsによるバリアントとバライアビリティーの管理
- レガシーシステムをソフトウェアプロダクトラインに転換する
- 管理の観点からのソフトウェアプロダクトライン
- バリアント：テストと欠陥へのリンク
- pure::variantsとモデルベースツール

Who Needs Product Lines?

プロダクトライン開発による効果。例えば図にあるような、自動車メーカごとにデジタルやアナログの表示パネルを提供している場合、その中身の処理は、基本的に同等のものであることが想像されます。このような製品ファミリーを作っているメーカであれば、既存の機能、フィーチャを組み合わせることで、製品開発ができれば、効率や品質において、高い改善効果を望めるでしょう。

pure-systems 社

- **業務範囲**
  - 開発ツール
  - ソフトウェア開発
  - コンサルティング、特注
  - トレーニング
- **顧客**
  - 主に組み込みシステム向け
- **２００１年ドイツ、マグデブルグに設立**

講師

- Danilo Beuche
  - 2001-: ドイツ、マグデブルグ　pure-systems社　代表取締役
    - 組み込みソフトウェア開発とプロダクトライン開発におけるコンサルティング
  - 1997-2003: ドイツ、マグデブルグ大学にて、研究員、研究助手（博士）
    - 組み込みシステムにおけるソフトウェアファミリーに於ける博士
    - 組み込みOSファミリーに着目したOS研究グループに所属
  - 1995-1997: ドイツ国立情報処理研究所（GMD FIRST　現、フラウンホーファー研究所）研究助手

© pure-systems GmbH 2007

講師紹介

Dr. Danilo氏は、独マグデブルグ大学、及び1995年からのGMD First（現、フラウンホーファー研究所）における、組み込みシステムに関する研究、フィーチャベースの開発ツールに対する取組みを基に、2001年に、pure-systems社を設立しました。組み込みシステムを中心に、プロダクトライン開発を支援する、バリアント/バライアビリティ管理ツールの開発を行い、コンサルタントとして顧客のプロダクトライン導入を支援しています。

Dr. Danilo Beuche is managing director of the pure-systems GmbH. pure-systems is a software company specialized in services and tool development for the application of variant and variability management and product line technologies in embedded software systems. When he joined the GMD First (now Fraunhofer FIRST) in 1995, he started to work in the field of embedded operating systems and software families and continued at the University Magdeburg, where he also received his PhD in this area. His work on tool support for feature based software development finally lead to the founding of pure-systems in 2001. At pure-systems he works also as consultant in the area of product line development mainly for clients from the embedded systems industry.

He has been tutorial presenter, speaker, workshop organizer and panelist at conferences such as AOSD, ISORC, SE, SPLC and OOPSLA. He is also author of articles in scientific journals and software developer magazines. During his university career and also to a limited degree later on he has been teaching students as tutor, teaching assistant and lecturer in the areas of operating system development and software engineering

ソフトウェアプロダクトラインについて

© pure-systems GmbH 2007

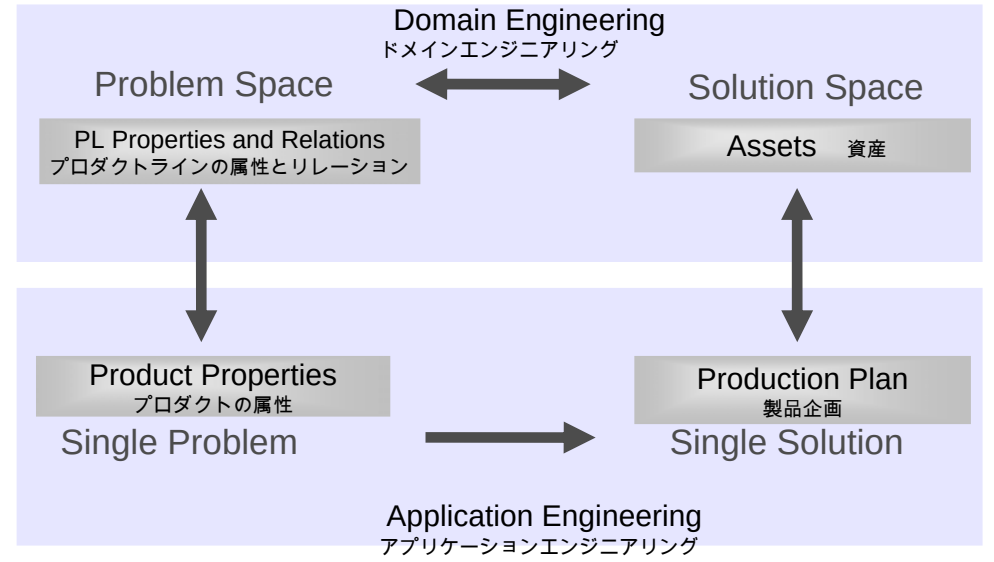
用語

- このセミナーで使用するソフトウェアプロダクトラインの用語
  - Problem Space vs. Solution Space
    プロブレムスペースとソリューションスペース
  - Domain Engineering vs. Application Engineering
    ドメインエンジニアリングとアプリケーションエンジニアリング
  - Variant vs. Version　　バリアントとバージョン
  - Variation Point　　バリエーションポイント（変動点）
  - (Core) Assets　　（コア）資産

© pure-systems GmbH 2007

ここでは、pure::varitants　だけでなく、プロダクトラインエンジニアリングでも使用される用語を紹介します。これら用語の定義に関しては、この後のスライドで詳しく説明します。

用語（２）



© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

これは、プロダクトラインに関する資料で良く見かける図です。

最初の項目は、プロダクトラインプロパティーです。これは、プロブレムスペース（左上）と呼ばれ、プロダクトライン内の、製品の機能的特徴を表します。例えば、自動車の場合、プロパティーの例は、コンバーチブル、ステーションワゴン、リムジンなど。これらそれぞれのプロパティーは、一製品に相当します。これらプロパティーと、それらの関連情報はプロダクトラインエンジニアリングに於いて、重要な要素です。

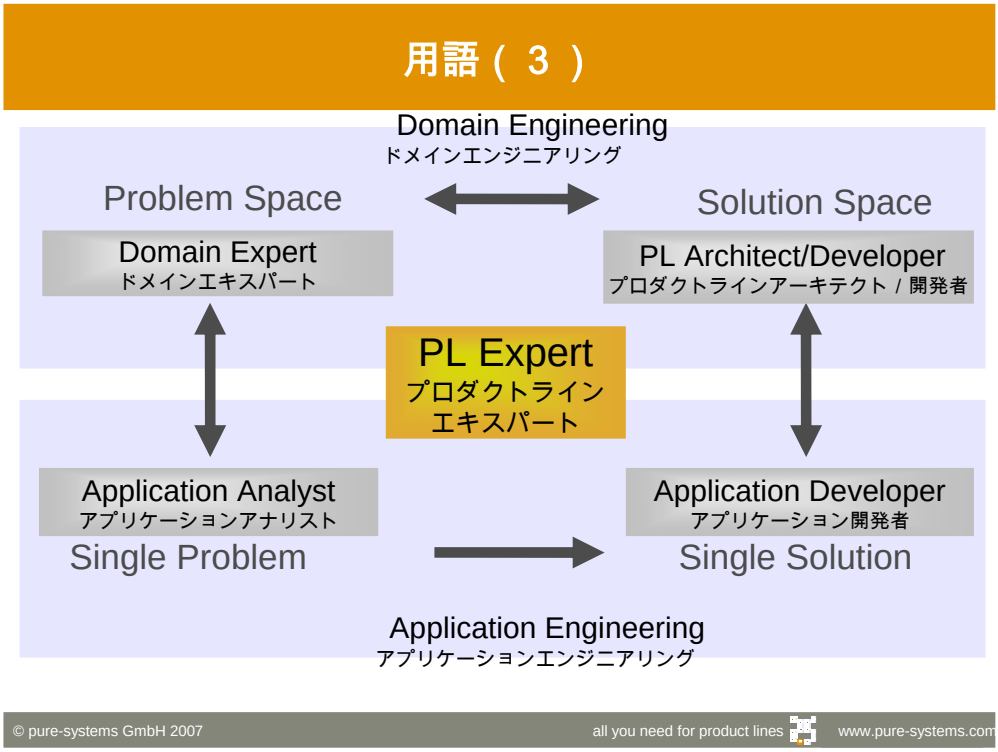
また、ソリューションスペース（右上）は、例えば、機械的な部品やソフトウェアコンポーネントの事です。特定の製品ファミリーを構成する、コンポーネントやオブジェクトなどの資産です。

プロダクトラインプロパティーから特定のプロパティーを選択し（シングルプロブレム　左下）、プロダクトラインの資産を組み合わせます。

これら全ては、プロブレムスペースに関連付けられています。

その後、製品企画により、単一の製品が作られます（シングルソリューション　右下）
これら全ては、互いに関連付けられています。

別の観点から見ると、プロダクトラインエンジニアリングには、２つの作業があります。プロダクトラインで何が出来るかをプロブレムスペースとソリューションスペースに構築するドメインエンジニアリング（上半分）と、シングルソリューションや単一のプロダクトに何が必要かを確認するアプリケーションエンジニアリング（下半分）です。つまり、プロダクトラインには、１つのドメインエンジニアリングと並行して行われる複数のアプリケーションエンジニアリングがあります。



プロダクトラインを進める為には、幾つかの役割があります。

ひとつは、プロダクトラインのプロパティーと、それらのリレーションシップを抽出し、それらを定義するドメインエキスパートです。ここでプロダクトラインのスコープを決定し、外部へのインターフェース（外部からのプロパティーの見え方を）を、プロダクトラインの資産を開発し、その用法を定義する、プロダクトラインアーキテクトや開発者に提供します。ここで定義する内容はテクニカルである必要は無く、製品ユーザーの観点で識別できるようにします。

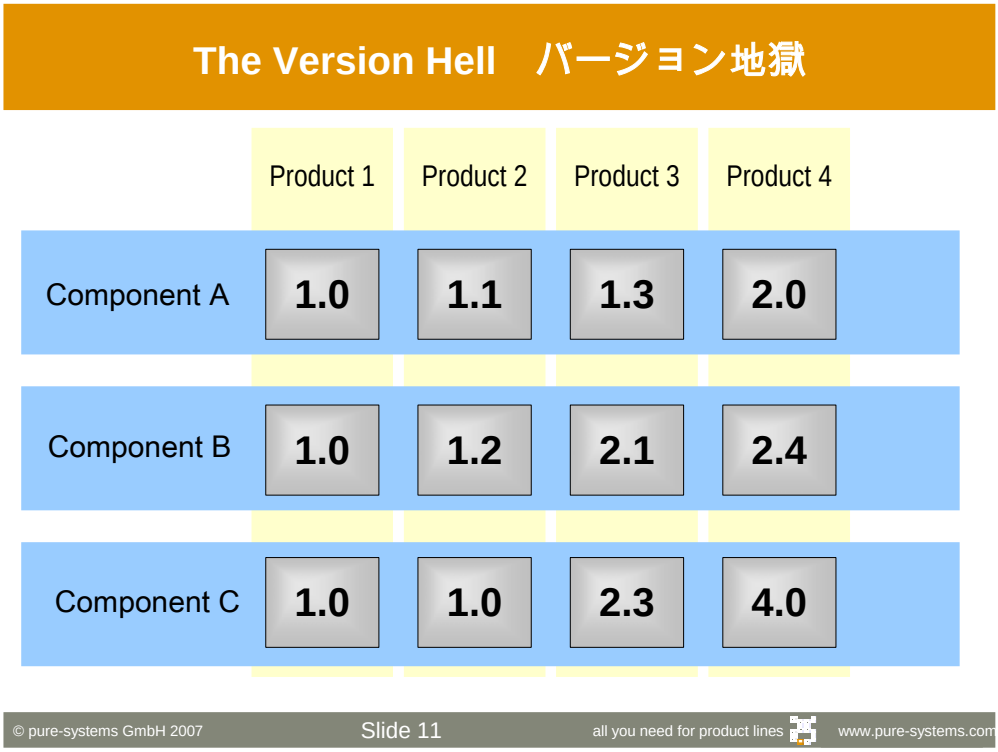
プロダクトラインアーキテクトや開発者は、コア資産をテクニカルな観点でソリューションスペースに定義します。

アプリケーションアナリストは、プロダクトエンジニアリングを満足する特定のアプリケーション（単一のプロダクト、シングルソリューション）を詳しく調査し、個々の顧客の要求を満足する為に必要な機能を考えます。

理想的には、シングルソリューションでは、個々のアプリケーションが開発者にテストされるだけであることです。

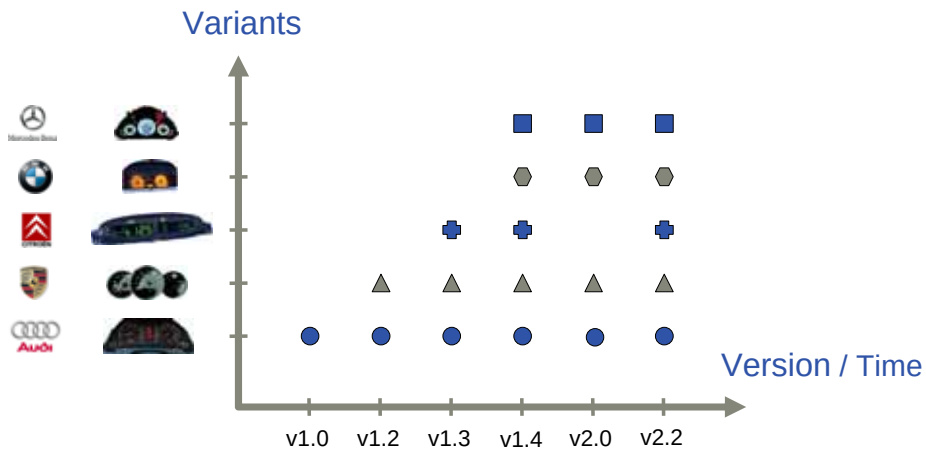
作成されたアプリケーションをテストし、問題があればドメインエンジニアリングにフィードバックされます。

プロダクトラインエキスパートは、開発者、アナリスト、ドメインエキスパート、プロダクトラインアーキテクトと上手くコミュニケーションし、プロダクトラインが上手く成立するように管理する役割を担います。この管理は、単一のプロダクトを開発するよりも複雑な工程になります。



体系的な管理を行わずに複数製品を開発しようとすると、バージョン地獄と呼ばれる状況に陥ります。例えば、再利用可能なコンポーネントで構成される幾つかのプロダクトがあるとします。ここでは、コンポーネントA、B、Cとして表されています。図から解るように、長期間にわたって開発されてきた各々のプロダクトは、異なったバージョンのコンポーネントで構成されています。各プロダクトごとで、機能追加されたり、その結果として既存コンポーネントが修正されるなど、個々に進化を始めるからです。これら異なったバージョンのコンポーネントで構成され、違う時期にリリースされた、異なったプロダクトは、保守が難しく、他のプロダクトに効率的に修正を反映させるのも難しくなります。プロダクトラインを成功させる方法の1つは、バージョン地獄と呼ばれる状況を回避する為の正しい考え方のプロセスやツールを準備することです。

Orthogonality of Variants and Versions
バージョンとバリアントの直交性



図は、プロダクトラインのシナリオです。縦軸は製品バリアント、横軸はプロダクトラインのバージョンです。個々のプロダクトの機能は、使用するプロダクトラインのバージョンから得られるようになっています。最初の段階では、１つの特定の製品のみですが、新しい機能を追加し、新たな製品を作り、プロダクトラインとして管理する事で、簡単なプロパティーの選択で、顧客毎に特定のソリューションを提供出来るようになります。　顧客が必要としなければ、この図のＶ２．０の上から三つ目のように、バリアントが無いこともありえるでしょう。改めて必要になったときには、いつでも追加できるでしょう。あくまでもビジネスとしての選択肢であり、技術論は問われることが無く、このような仕組みが実現されます。

用語（４）　バージョンとバリアント

- Version (also Revision)　バージョン（リビジョン）
  - １オブジェクトのバージョンは、異なる時点のオブジェクトを指示す。異なるバージョンのオブジェクトは、変更されている場合も、いない場合もある。
- Variant　バリアント
  - バリアントは、ユーザーによって識別可能な属性を持ったオブジェクト。これらは、プロダクトラインの共通の基本オブジェクトを基にして、異なるプロパティによって提供される。通常、ユーザの観点で識別可能（自動車の色など）

© pure-systems GmbH 2007

ここでは非常に大切な、バージョン或いはリビジョンと、バリアントの明確な違いについて、説明します。バージョンとは、ある時点におけるオブジェクトのこと。オブジェクトなり、資産が何であるか、どうであるか、と言うことではなく、オブジェクトのある時点を示している。

バリアントは、互いに異なるオブジェクトのこと。例えば、2つのプロダクトバリアントは、プロダクトラインの共通のベースからなる2つの製品間で、オブジェクトに対する異なったプロパティ（顧客の観点から見た）をもちます。車のプロダクトライン於ける、製品の色の違いが判り易い例。つまり、プロパティのインスタンス間の違いをユーザーが識別できる。これらプロパティは、プロダクトラインにおけるバライアビリティです。

用語（５）　バリアントとバージョンに関する課題

- バリアントはバージョンの変更で提供されることがある（オブジェクトを修正し、機能を加える場合など）
- バリアントは単一バージョンのオブジェクトからも作られる（コンフィグレーションなどで振舞いを切り替える場合など)
- １バリアントが、１オブジェクトの異なるバージョンで構成され続ける

▶ 単一のシステム開発に於いては、バリアントとバージョンは同等に扱われることが多い。　バリアントが直接変更されることなく、バリアントは特定のバージョンと言うことになる。

© pure-systems GmbH 2007

ここで問題、あるいは課題は、一般にバリアントが、バージョンに相当することです。なぜなら、オブジェクトの新たなプロパティーを実現すること、例えば、編集機能を追加するには、それを実現するコードをオブジェクトに追加する必要があるので、古いバージョンと新しいバージョンのオブジェトは違うものになるからです。

また、単一バージョンのオブジェクトから、バリアントが生まれることもある。例えば、コンパイラスイッチやコンフィグレーションファイル等による、オブジェクトの異なった振舞いは、ユーザの観点において異なったプロパティを実現することになる。車の色の場合は、このようにはいかないでしょうが、データベースへのアクセスを設定で切り替える等のケースは、想像できるでしょう。

バリアントは、単一のバージョンのファンクショナリティ（機能）で定義され、異なったバージョンから生み出され続ける。厄介なのは、単一システムにおいて、バリアントとバージョンの互換性が失われること。各バリアントが、特定のバージョンに帰属してしまうこと。

用語（６）バリエーションポイント

- Variation Points　バリエーションポイント（変動点）
  - プロダクトラインのメンバー間で異なる全ての場所。
  - プロブレムスペース、ソリューションスペースの両方にある
  - リンク時、コンパイル時、実行時などに、解決される
  - 例えば：
    - Problem space:　プロブレムスペース

      “車は、２つ又は４つのドアをもつ”

    - Solution space:　ソリューションスペース

      #ifdef / #else / #endif　で囲まれたコード領域

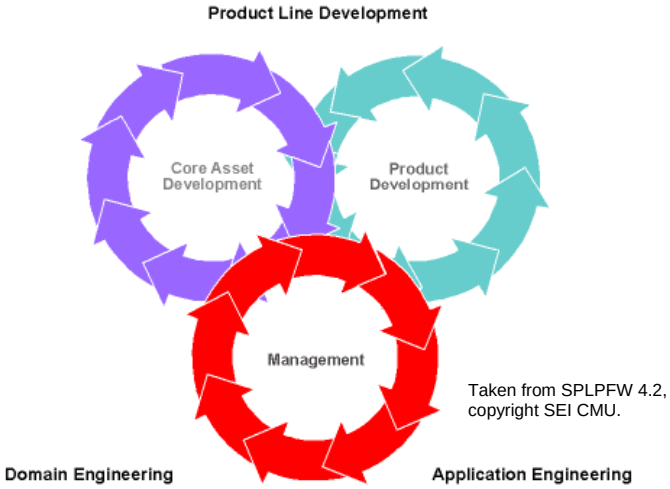
© pure-systems GmbH 2007

バリエーションポイントは、プロダクトライン開発における本質的な要素です。プロダクトラインを構成するものの、異なる全ての場所のことです。これは、プロブレムスペースとソリューションスペースの両方にあります。なぜなら、バリエーションはプロブレムスペースで選択可能な選択肢として表され、その影響がソリューションスペースに及ぶからです。バリエーションポイントのプロパティは数々有りますが、その中で重要なものはバインディングタイムと言われるものです。例えば、ソリューションスペースのバリエーションポイントは、コンパイル時の、＃IFDEFです。また、ランタイムな変数によって、異なる振舞いをさせることもあるでしょう。

プロブレムスペースの例は、異なる数のドアを持つ車など。

SPL開発

SPLD活動（SEI SPLP フレームワーク）



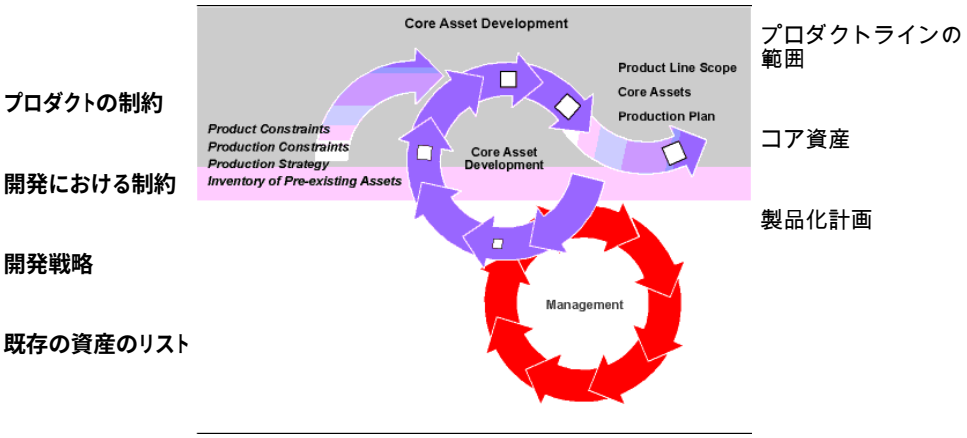
Taken from SPLPFW 4.2, copyright SEI CMU.

© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

プロダクトラインエンジニアリング／プロダクトライン開発のアクティビティーは、3つのコアアクティビティに分割されます。1つはコア資産開発で、ソリューションスペースの開発を行います。2つ目はプロダクト開発で、アプリケーションエンジニアリングを体系化します。3つ目は、これら2つのコアアクティビティーの管理です。これは、テクノロジーではないけれども、基本的に最も重要な項目です。なぜなら、個別に、独立して行われている、コア資産開発とプロダクト開発を、同期させて、再利用可能な状態にすることはより困難な作業だからです。このようなプロダクトライン開発における管理は、最も大きな課題です。

SPL開発

コア資産開発に於ける入力と出力(SEI SPLP Framework)



Taken from SPLPFW 4.2, copyright SEI CMU.

© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

開発組織の幾つかのポイントから始めます。現行のクリティカルなオペレーションの1つは、コア資産開発です。なぜなら、製品の制約、製品化の制約、再利用の戦略、利用／再利用が必要な既存の資産（プロダクトラインを開始するには、資産が事前に開発されていないといけない）等の沢山の入力があり、これらの知識や資産からコア資産を作ったり、プロダクトラインスコープ（実現可能なのプロダクトラインの境界）や製品化計画（資産を使って個々のソリューションを作る方法）を提供する必要があるからです。

ソフトウエア開発の“経済性”を高める

Parnas (1976):

„We consider a set of programs to constitute a family, whenever it is worthwhile to study programs from the set by first studying the common properties and then determining the special properties of the individual family members.“

“ファミリーを構成するプログラムに対して、共通部分の属性を評価し、個々のファミリーメンバーの特性を確認すること“

ファミリーを見出して、効率を得ること
プロダクトラインエンジニアリングはビジネスコンセプト（テクニカルと言うよりは）

デイビッド・L・パーナス（David Lorge Parnas）

１９７６年にデイビッド・L・パーナス**がスソフトウェアプロダクトラインエンジニアリングに於いて、非常に重要なことを言っている。曰く**

”ファミリーを構成するプログラムに対して、共通部分の属性を評価し、個々のファミリーメンバーの特性を確認すること”

つまり、ファミリーを見出して、効率を得ることが重要であり、プロダクトラインエンジニアリングは（テクニカルと言うよりは）ビジネスコンセプトである。

Reuse
再利用

従来手法との違いに着目して、再利用について述べます。

再利用技術　-　課題

- バリエーションポイントが見付けづらい
- バリエーションがあらゆるレベルに跨っている（ソースコード、ドキュメント、コンフィグレーションファイル等）
- 形式化の程度が低い
- 責任範囲が、非常に多くの観点を持ち、利害関係者間に跨っている

再利用を推進する上での課題を上げています。

これら課題を如何に解決するかが、再利用を推進する上で重要になります。

再利用ではバリエーションポイントを見つける事がもっとも重要ですが、見つけるのが難しい事が課題になります。バリエーションが色々な場所に渡っている（例えば、ソースを変えると、テストやドキュメントも修正しないといけない等）ことも課題です。ソースのバリエーションは、例えば、“車のドアは2ドアか4ドアである”と言うように、明確に記述す事が出来ません。色々な利害関係者に跨っていることも課題であり、同じ事を違う観点でみる必要があります（ユーザー、マーケティング担当者、営業、開発で観点が違う）。UMLを使ってもこれを解決することは出来ません。

再利用プロセス　-　再利用が失敗する理由（１）

- 製品開発が外部委託で行われる場合
  - プロジェクト費用は、工数により算出される（業者は再利用に興味が無い）
- 以下の場合、再利用可能な資産を作成・保守しようとすると、コストがかさむ
  - 所定のドメイン / 組織内における技術のミスマッチ（C/C++/Java等が混在）
  - 再利用に対する利害関係者の教育が不十分
  - 異なるプロジェクトチームが、同じような機能をたくさん並行開発している

再利用に於いて起こる問題を紹介します。以下が再利用が失敗する幾つかの例で、今までの経験からリストアップしました。再利用のプロセスには、効率的に保守･管理が出来るようなプロセスが必要です。殆どの再利用における問題は、技術的な問題でなく、経済的な問題です。1つの例は、製品開発を外部委託している時に、再利用は多くの場合失敗します。なぜなら、プロジェクト費用が工数から算出される為、再利用によって開発時間が短くなっても、利益にならないからです。彼らは再利用から得られる利益について興味が無く、顧客が再利用を推進するように依頼しても、再利用を成功させようとは考えません。2つ目の例は、再利用可能な資産を作成･保守しようとするとコストが嵩む幾つかのケースです。1つは、そのドメインや組織に於いて、技術の再利用が非常に困難な場合。2つ目は、利害関係者に対する再利用の教育が不十分なケース。管理の立場から見て、全ての利害関係者に、再利用に関する十分な知識を与える為のコストがかかりすぎる。3つ目は、異なるプロジェクトチームが同じような機能を沢山開発しているケース。これらのプロジェクトでは、少しずつ違う機能が、実装されており、これを再利用する場合、プロジェクト間の調整が困難です。

再利用プロセス　-　再利用が失敗する理由（２）

- 使用されない / 使用に適さない資産の開発
  - コア資産開発チームが開発したコンポーネントをアプリケーションチームが使わない
  - プロダクトが顧客の合否基準に合わない
- 管理側が開発体制を十分にサポートできない
  - 高い再利用性を実現するプロジェクトは、開発初期にコストがかかる傾向にあるが、管理者は低コストで開始することを要求する

3つ目の例は、使用されない或いは使用に適用されない資産を開発するケースです。これには、コア資産開発チームが開発した資産を他のチームが使わないケースがあります。これは組織上の問題で、何処でも起こることではありません。あるチームが他のチームが開発したコンポーネントや資産を信頼できない為に、使わないというケースがありました。また、コア資産が十分でなかった為に使われなかった例もありました。

4つ目の例は、非常に重要な問題で、管理側が再利用の為の開発体制を十分にサポートできないケースです。なぜなら、高い再利用性を実現するプロジェクトを始めようとすると、単一のプロジェクトを開発するよりコストが嵩み、開発者が低コストで生産することを要求するとトラブルになるからです。

再利用　-　キーになる課題

- 経済的側面
- 教育面
- 意思疎通と連携
- 技術面

再利用を成功させる為に気をつける点は、重要である順番に、
プロダクトラインの経済的側面、全ての利害関係者対する教育、全ての利害関係者間の意思疎通と連携となります。
そして、最後に、それほど問題になることはありませんが、技術的側面もキーになります。

Variant and Variability Managment

これらの問題を解決して、最良のソリューションを見つけるには？
バリアント、バライアビリティ管理のギャップに注意を払うことです。

Variant and Variability Management
バリアントとバライアビリティーの管理

- **以下の詳細記述と、そのメンテナンスに気を配る**
  - **プロダクトラインのバリエーションポイント情報（バライアビリティー）**
  - プロダクトラインの実体として提供される（バリアント）
- 再利用可能な資産と、それらから構成される実体のギャップを小さくする（バリアントとバライアビリティーの間の関連付けを十二分にとる。関連が曖昧な場合には、管理・メンテナンスが容易でなくなる）
- 様々な領域に渡って複雑さに対処すること

バライアビリティーとバリアントの管理に於いては、最初の段階のバリアントポイント情報を作成するときに、以下の定義と保守に気を配る必要があります。同様に、プロダクトラインの実体であるバリアントに気を配ることも重要です。なぜなら、たとえ、バリエーションポイントが少ししかなくても、最終的に作成されるバリアントが多くなってしまう事があり、そこに気を配らないと、保守が困難になる事があるからです。また、再利用可能な資産とそこから生成される実体のギャップを小さくすることも重要です。

Variant and Variability Management
バリアントとバライアビリティーの管理

- ... is complex because　複雑である理由
  - バライアビリティーはほぼ指数関数的に増加する
    - 各オプションは取り得るバリアントの数を倍加する
    - n 個の選択肢を持ったグループは、それぞれ、更に n 倍のバリアントを作る
  - 予期される / 予期されないバリエーションの要求を扱う必要がある
  - 単一のシステム開発に比べて、多くの利害関係者が関与する

バリアントとバライアビリティーの管理が複雑である理由は、

バライアビリティーが指数関数的に増加するからです。各オプションは取り得るバリアントの数を倍加し、n個の選択肢を持ったグループは、それぞれ、更にn倍のバリアントを作ります。

予期される／予期されないバリエーションの要求を扱う必要があるから。なぜなら、プロダクトラインを始める段階で、どの程度のバリエーションがあるかが分かる人はいないからです。

単一のシステム開発に比べて、多くの利害関係者が関与する。その為、調整や意思疎通が複雑になる。

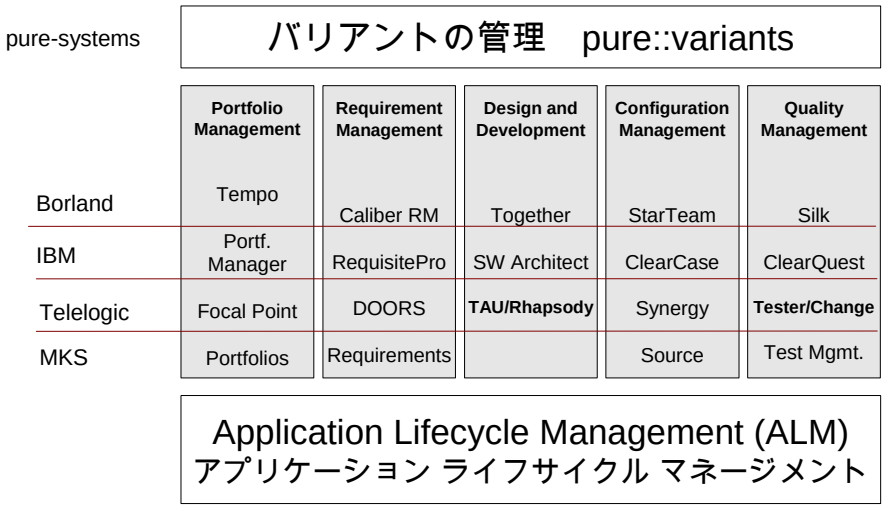
pure::variants

による
バリアントとバライアビリティーの管理

© pure-systems GmbH 2007

プロダクトラインエンジニアリングが、再利用の最大化のキーになります。
また、バリアントとバライアビリティー管理が、それらの問題を解決するキーになります。
Pure::variants**は、これらに必要となる情報を登録し、メンテナンスできる仕組み･ツールです。**

ALM and Variant Management
ALMとバリアント管理



これらは、アプリケーションのライフサイクルで使用される、典型的な管理ツールです。

pure::variants**はこれらのツールと連携する仕組みを持っており、これらツールのデータのバライアビリティーを取り込む事で、ライフサイクル全体に渡ってバリアント管理が行えます。 このセミナーでは、**pure::variants **自身の説明に加え、いくつかのツールへのコネクション機能にも触れる予定です。**

pure::variantsは、

既存の開発プロセスへ統合が容易

- 既存のコード、ツールチェーンを使い続けられる
- バージョン管理をサポート
- テクノロジーに依存しない（C、Javaなど、あらゆる言語や、HWの管理にも）

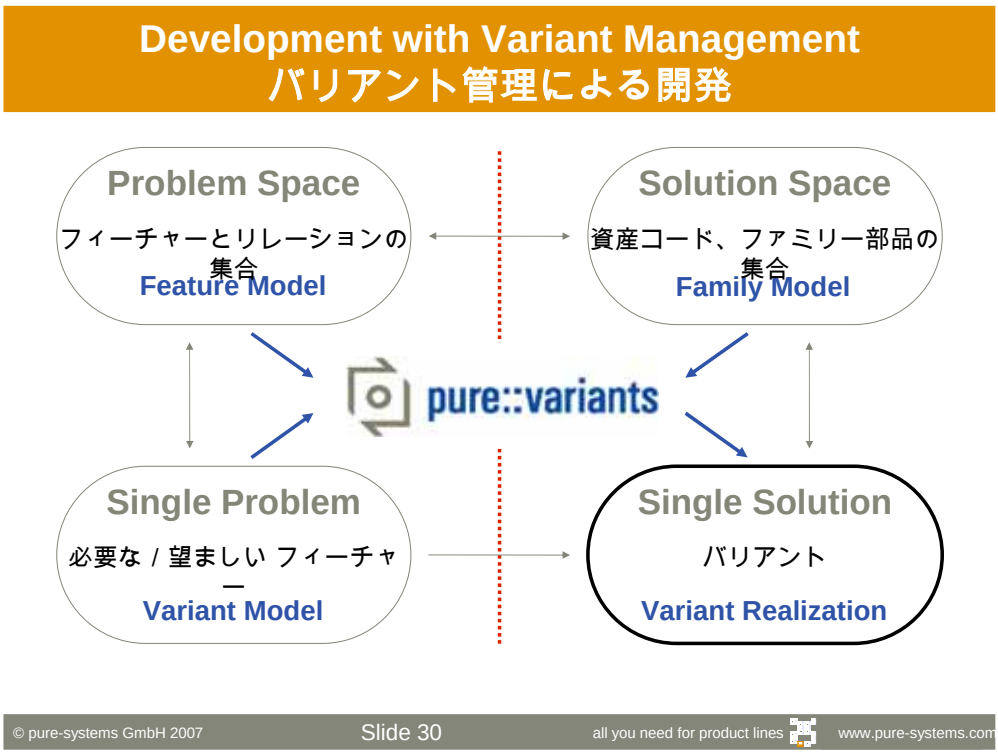
効果的なバリアントモデリング

- フィーチャーの組み合わせに関するノウハウを蓄積し、共有できる
- バリアントの構成を検証

自動的にリソースを活用して、バリアントを生成

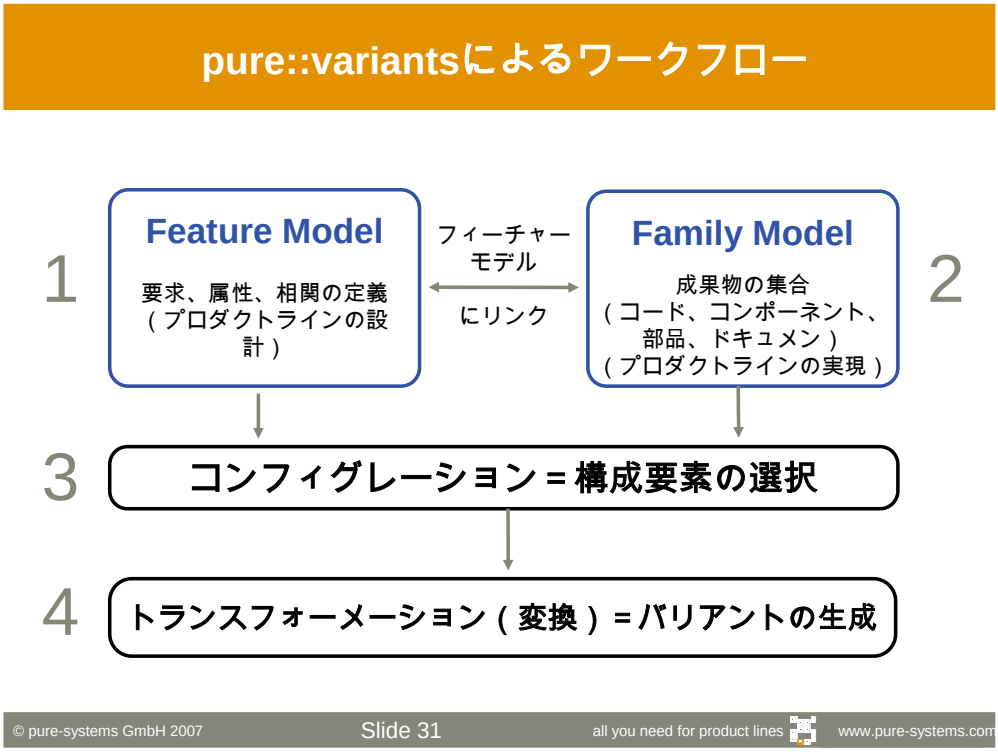
- ソースコードをパッケージに（例：バージョン管理ツールから）
- コード、ドキュメント、部品表の生成

pure::variantsは、既存の開発プロセスへの統合が容易です。なぜなら、既存のツールチェーンやコードを使い続けられるからです。また、既存の開発環境を変更する事無く使えるからです。バージョン管理ツールもサポートしてます。特定の実装テクノロジーに依存しない（C、C++、JAVAなどあらゆる言語に対応し、HWの管理にも使える）点も重要です。
これらの特徴により、効果的なバリアントモデリングが可能で、その為、構成を行う上での知識が蓄積できます。
また、バリアントの構成を検証することも可能です。
自動的にリソースを活用してバリアントを作成する機能も持っています。
例えば、バージョン管理ツールからソースコードを取り出して、バリアントコード生成する等、ソースコードのパッケージ化が可能です。
コード、ドキュメント、部品表の作成もできます。



この図は、最初の方に出て来た図にpure::variantsの用語を適応したものです。

プロブレムスペースはフィーチャーモデルで、単一のソリューションはバリアントモデルで表されます。ソリューションスペースはファミリーモデルで表され、バリアントモデルを選択する事で、バリアントが作成されます。



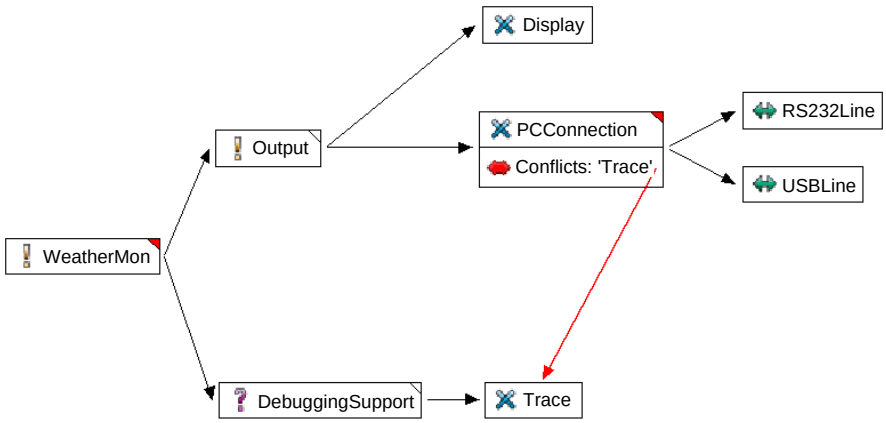
この図は、pure::variantsの通常のワークフローを表しています。最初にフィーチャーモデルを作ります。これは、ドメインエキスパートの仕事です。フィーチャーモデルには、要求とそれらの属性や関係を定義します。次にファミリーモデルを作り、フィーチャーモデルにリンクします。フィーチャーモデルとファミリーモデルの変更を含めて、プロダクトラインアーキテクトやプロダクトライン開発者の仕事です。次に、これらのパーツの選択方法の定義（コンフィグレーション）を行い。最後に、バリアントを生成（トランスフォーメーション）させます。

Weather Stationの例：プロダクトのバリアント

Variant	Feature
Thermometer:	Display, Temperature
Indoor:	Display, Temperature, Pressure
Outdoor: Wind	Display, Temperature, Pressure,
Deluxe:	+ PC Data Recording
Internet Edition:	+ TCP/IP
PC Interfaces:	+ Serial Interface RS232
	+ USB Interface

Weather Station**でのプロダクトバリアントの例**

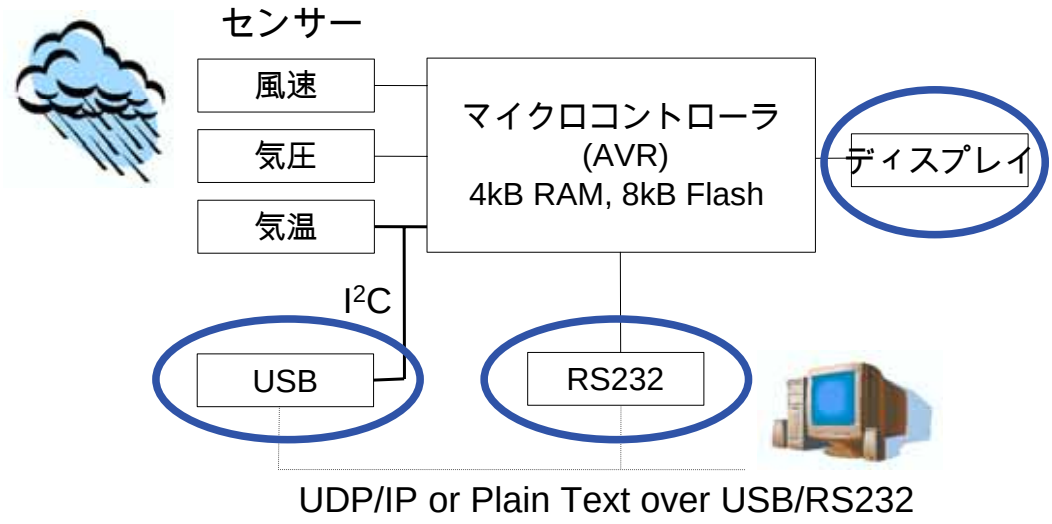
Problem Space – Feature Model
プロブレムスペース　- フィーチャーモデル



partial view

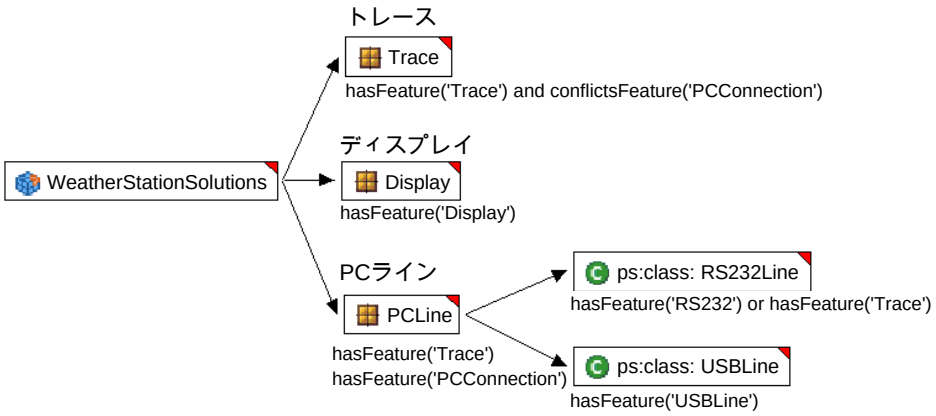
ここでは、ＲＳ２３２ＣとＵＳＢの２つのAlternative型のフィーチャーがあり、ＰＣにデータを送る手段を定義しています。Alternative型は双方向矢印のアイコンで表され、”グループ内でどちら１つだけが選択”され無ければなりません。これらはPCConnectionフィーチャーの下に接続されており、PCConnectionフィーチャーが選択されたときにのみ選択されます。PCConnectionフィーチャーは、Outputフィーチャーの下に定義されており、同じグループにDisplayフィーチャーも定義されています。バツのアイコンはOrタイプのフィーチャーを表し、”グループ内で少なくとも１つが選択される”必要があります。これらのフィーチャを選択することで、どの様な出力が必要かを定義できます。左側にはWeatherMonフィーチャーがルートフィーチャーとして定義されており、エクスクラメーションマークのアイコンで表されています。これは、Mandatory型のフィーチャーで、”必ず選択される”フィーチャーです。Outputと同じグループに、Optional型のDebuggingSupportフィーチャーも定義されています。Optional型はクエッションマークのアイコンで表されており、”選択してもしなくても良い”フィーチャーを表します。DebuggingSupportフィーチャーの下には、Or型のTraceフィーチャーがあります。殆どのフィーチャーはこれらの型で設定できますが、より複雑な関係を設定することも可能です。例えば、コンフリクトという設定があり、図の赤い線がそれを表しています。ここでは、PCConnectionフィーチャーとTraceフィーチャーが同時に選択できない事を表しており、もし同時に選択した場合は、モデルの評価を行ったときにエラーになります。

Weather Stationの例：ハードウェア



この例のハードウェアを見てみましょう。コントローラがあり、ディスプレイとRS232Cが直接つなっがています。USBと温度センサーは、IICで接続されており、気圧センサーと風速センサーはバスに直接繋がっています。もし、気圧を測定しないのであれば、気圧センサーを外すことも可能です。このハードウェアアーキテクチャーもファミリーモデルのバリエーションポイントになっています。

Solution Space – Family Model
ソリューションスペース　-　ファミリーモデル



partial view

ファミリーモデルの例です。ここには、Trace、Display、PCLineの３つのコンポーネントがあります。これらコンポーネントはアプリケーション内に必要なソフトウェア機能を表します。各コンポーネントにはルールが規定されています。hasFeature()は、括弧内のフィーチャーが選択された時にその工レメントが有効になることを表します。例えば、hasFeature('Display')は、'Display'フィーチャが選択された時に、Displayコンポーネントがバリアントに組み込まれることを意味します。PCLineコンポーネントの下には、RS232CとUSBLineのクラス工レメントが登録されています。このクラス工レメントは、実際のアプリケーション内のRS232CやUSBの通信行う為のソースに関連付けられています。また、各々のクラス工レメントもルールを持っています。

pure::vatiants

at work

© pure-systems GmbH 2007

pure::variantsの製品構成

pure::variants Professional 版

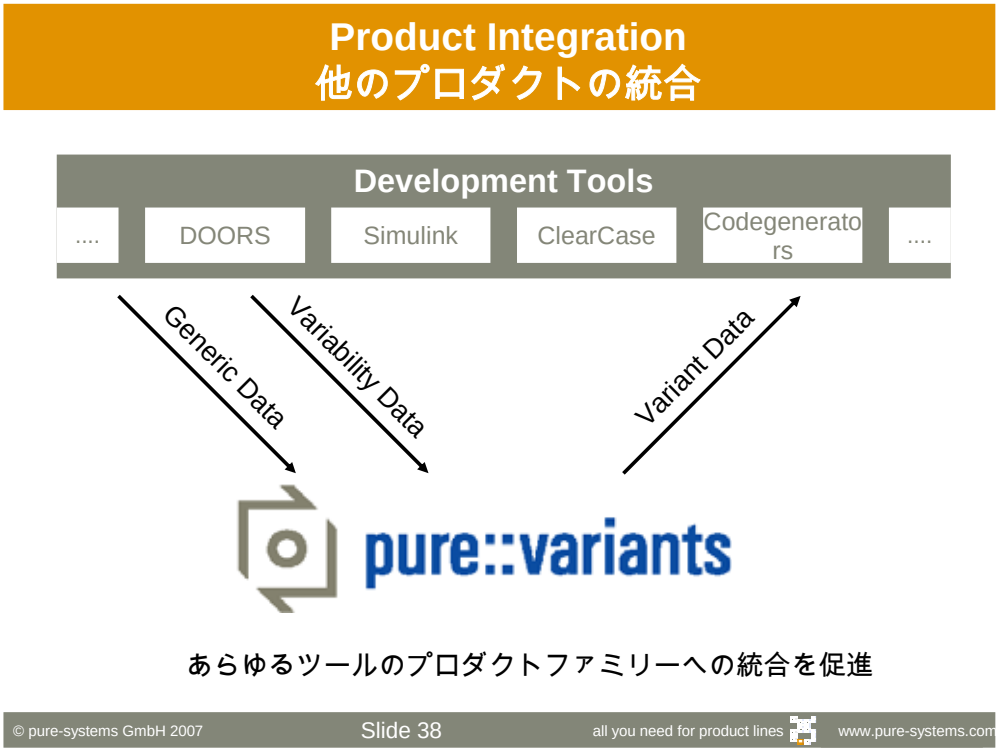
- モデルはＸML形式でローカルファイルに保存
- 標準の構成管理ツール（ CVS, Subversion, Perforce など ）による共同作業
- 構成管理ツールでバージョニングとブランチに対応

pure::variants Enterprise 版

- モデルはデータベースで集中管理
- 全ての接続ユーザが変更を自動的に参照できるリアルタイムの共同作業
- レポート生成と履歴管理を統合
- バージョニングとブランチは、pure::variantsのサーバーが行う
- 全てのProfessional版の機能

■バライアビリティ管理ツール　pure::variants とは？

pure::variants　では、フィーチャ・モデルとファミリー・モデルでプロダクトラインをモデル化します。従来のフィーチャ・モデルベースの手法ではなく、pure::variants　では製品の機能・課題をフィーチャ・モデルに、プラットフォームなどソリューションをファミリー・モデルに保存します。ファミリー・モデルはコンポーネントで構成されて、これらの相互接続や制限、条件なども含みます。ファミリー・モデル内の各コンポーネントはプロダクトライン内で1つ以上の機能要素に相当します（クラス、オブジェクト、関数、変数、ドキュメントなど）。フィーチャ・モデルでは、個別製品ごとのフィーチャの相互関係を定義します。これらのモデルにより、製品ファミリーの全ての変動要素、共通要素は一貫して管理されます。



各種ツールとの連携をサポート

プロダクトライン開発ではバリアントの管理は必要不可欠ですが、従来のソフトウエア開発ツールは、たいてい単一システム開発にしかフォーカスしていません。pure::variants はモデルベースでバリアントの定義とバライアビリティのモデリングを開発全フェーズでサポートし、既存のツールは、バライアビリティとバリアントを効率的に扱えるように増強されます。オープンインターフェイスを提供し、バリアントの情報は要求エンジニアリング、システムデザイン、実装、テストに渡って一貫して提供されます。新しいツールとの統合は容易であり、すでに多くのツールをサポートし、さらに増え続けています。

要求管理ツール DOORS(R)　CaliberRM(TM)
モデル駆動開発ツール MATLAB(R)/Simulink(R)　openArchitectureWare
ソフトウエア構成管理ツール　CVS　Subversion(R)　ClearCase
テスト管理・バグ管理　Bugzilla(TM)　ClearQuest(R)
　他、Enterprise Resource Planning　SAP(R)　など

Extensions
エクステンション

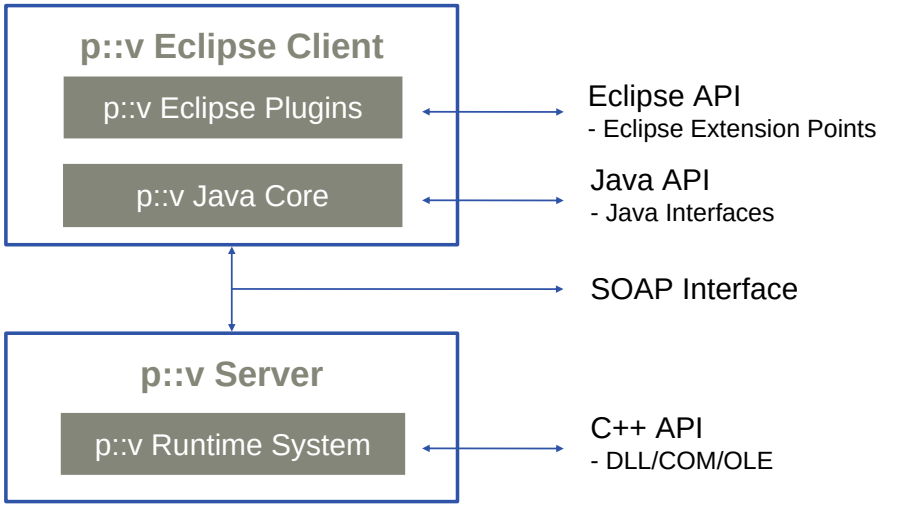
pure::variants　既存ツールチェインを統合するエクステンション

- Synchronizer for **DOORS**
- Synchronizer for **CaliberRM**
- Connector for **MATLAB / Simulink**
- Connector for **Source Code Management**
- Connector for **Version Control Systems**
- Connector for **SAP**
- Connector for **ClearQuest**
- Connector for **ClearCase**
- Connector for **Bugzilla**
- Connector for **Reporting with BIRT**

(API **は公開されているので、他のツールの統合も容易**)

これらが現在pure::variantsが提供しているエクステンション（外部ツールとの接続機能）です。
インターフェースは一般に公開されていますので、ここにないツールであっても、統合する事が出来ます。

Integration Interfaces
統合の為のインターフェース



pure::variantsが提供しているインターフェース（API）です。
pure::variantsのサーバーコンポーネントは、モデルの管理を行っており、C++API（DLL/COM/OLE)で接続可能。
クライアント側は、UIを司っており、SOAP、EclipseAPI（拡張ポイント）、JavaAPIをサポート。

Transforming Legacy Systems into Software Product Lines

レガシーシステムをソフトウェアプロダクトラインに転換する

モデル化に必要な情報を、既存ソフトウエアから抽出する方法を紹介します。既存製品・ソフトウエアからバリエーションポイントを抽出する方法を、このあと紹介して行きます。一からプロダクトラインを作るより、既存システムから行うほうが実践的であり、緩やかに適応できます。

Transition and the SPLP-Framework
レガシーからの移行とSPLP フレームワーク

- 基本的に、移行プロセスは、フレームワークに記述されている全ての情報を、既存の生成物から作成・収集する作業である

- １から始めるのとなにが違うのか？
  - 非常に有用な情報が既に提供されている（隠れているだけ）
  - プロセスや生成物の変更は**簡単ではない。しかしながら**、適切なプロセスや生成物を、始めから導入することは、もっと難しい。

© pure-systems GmbH 2007

ここまで、ソフトウェアプロダクトラインとバリアント管理について紹介しました。ここから、既存のソフトウェアから、バリエーションポイントと必要なモデルを抽出する方法を紹介します。

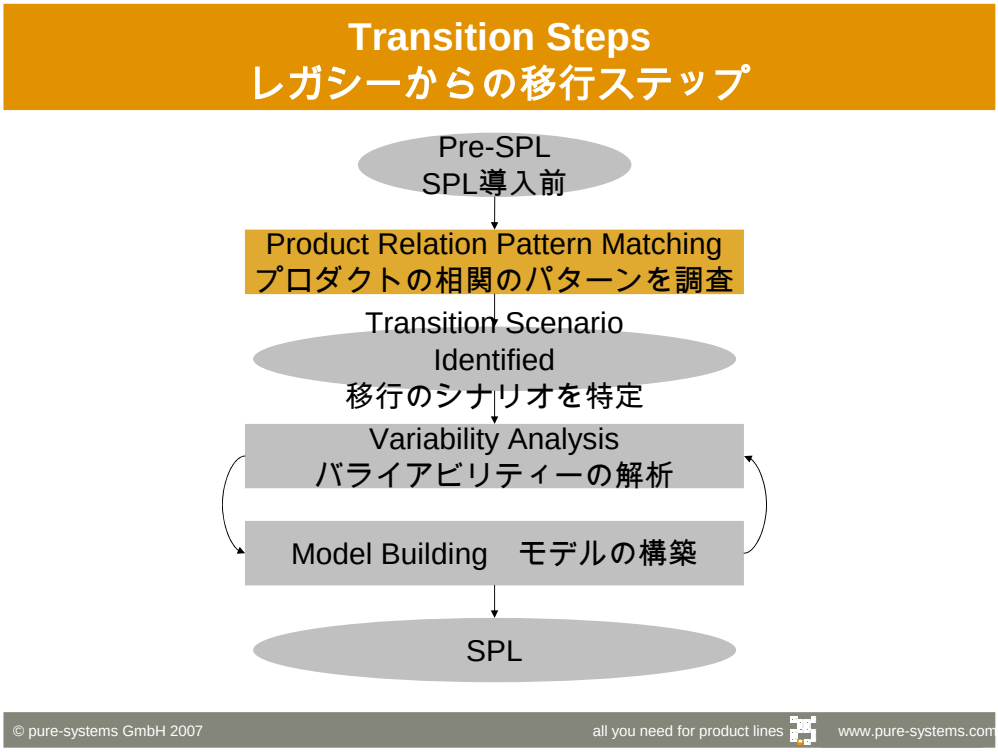
始めに、一から始めるのとでは、何が違うのかを説明します。既存の資産（レガシー）には、モデルを作成する為の殆どの情報が入っています（隠れているだけ）。ただし、プロセスや生成物の変更は簡単ではありません。例えば、一旦開始した、既に製品を開発した、プロセスを変更することは、非常に難しいですね。しかしながら、適切なプロセスや生成物を始めから用意すること（製品が無い所から、プロダクトライン開発を始める）は、もっと難しいでしょう。

Questions Before Starting a Transition
レガシーからの移行を始める前の疑問

- SPL/SPLDの為に、我々の組織に足りないものは何か？
  - それは、
    - ... 体系的なコア資産開発？
    - ... プロブレムスペースとソリューションスペースの規定？
    - ... これらの上で開発されたアプリケーション？
- なぜ移行を行うのか？
  - それは、
    - ... 開発コストを**全体的に**削減？
    - ... 似通ったプロダクトの開発時間の短縮？
    - ... 製品品質の向上？

© pure-systems GmbH 2007

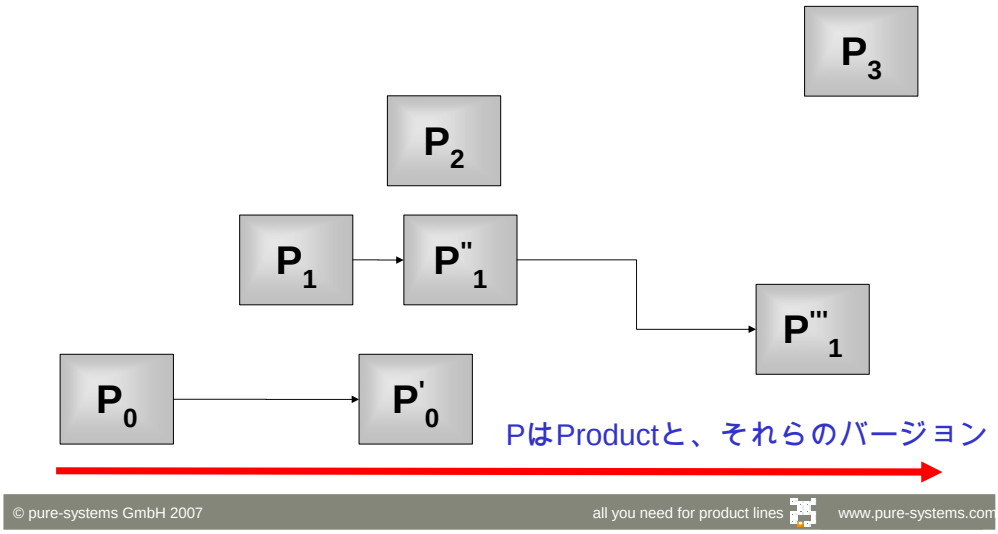
ソフトウェアプロダクトラインに移行する前に、これらの問いかけを行い、回答を出す事が重要です。



SPLを適用していないプロダクトをSPLに移行させる為の基本的なフローです。先ずプロダクトの相関のパターンを見つけます。次に、移行のシナリオを決定します。そして、バライアビリティーを解析して、バリエーションポイントをモデルに反映させる工程を繰り返す事で、プロダクトラインへの移行をすすめます。 これらの各工程について、次からのスライドで説明します。

Product Relation Pattern: Product Forest
プロダクトの相関パターン：プロダクトフォレスト

共有するコア資産が無い
ユーザーに見える機能は似通っているが、、



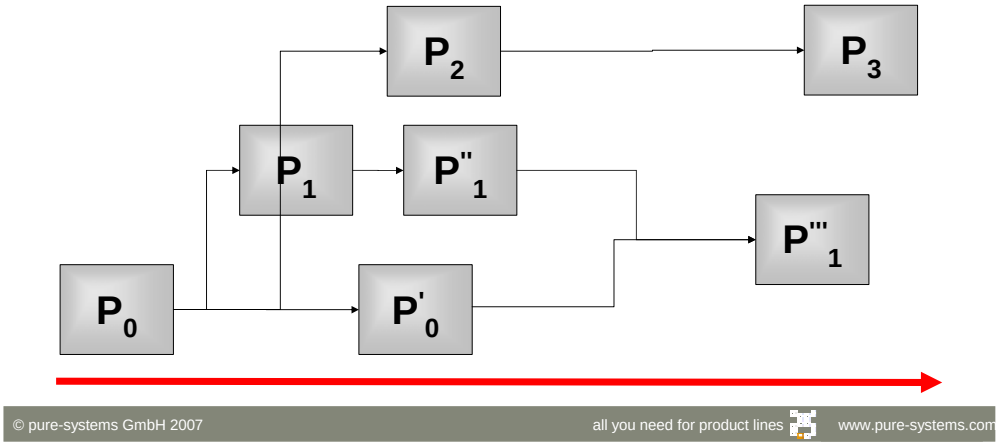
© pure-systems GmbH 2007

最初のステップは、プロダクトラインを適用する対象の構造を特定することです。

1つ目のパターンは、ユーザーから見える機能は似通ったプロダクトであるが、殆ど共有のソフトウェアコア資産を持たない構造です。プロダクトフォレストと呼ばれます。

Product Relation Pattern: Product Bush
プロダクトの相関パターン：プロダクトブッシュ

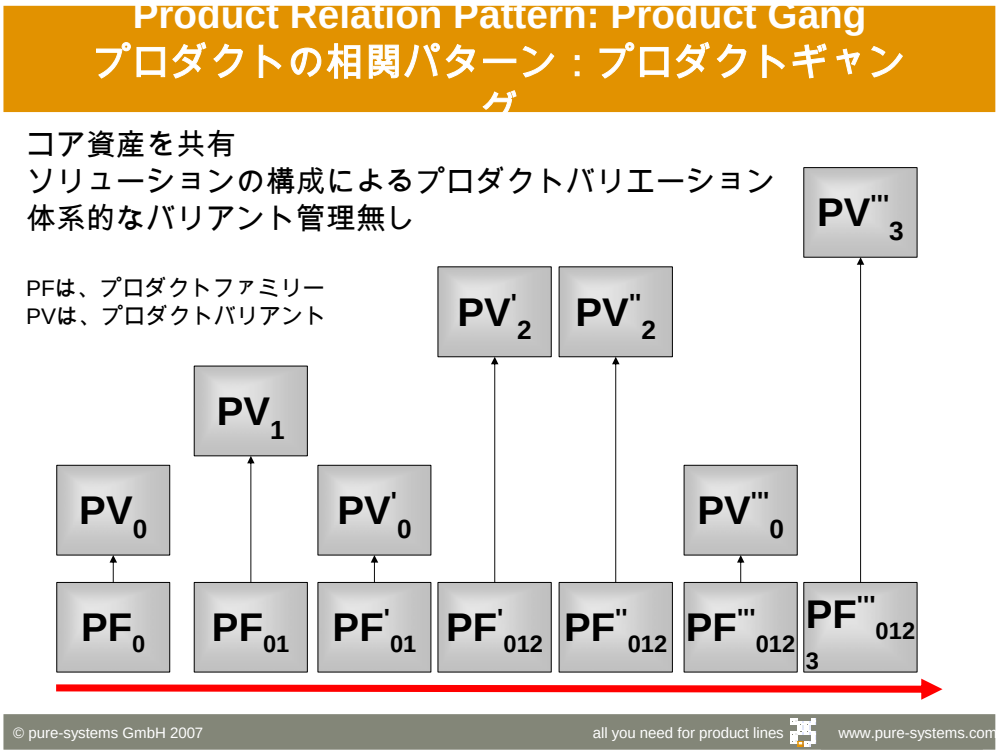
コア資産を共有しているが、
開発経路の分岐により、プロダクトバリエーションが作成されている



© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

2つ目は、プロダクトブッシュと呼ばれる別のタイプのプロダクト構造です。

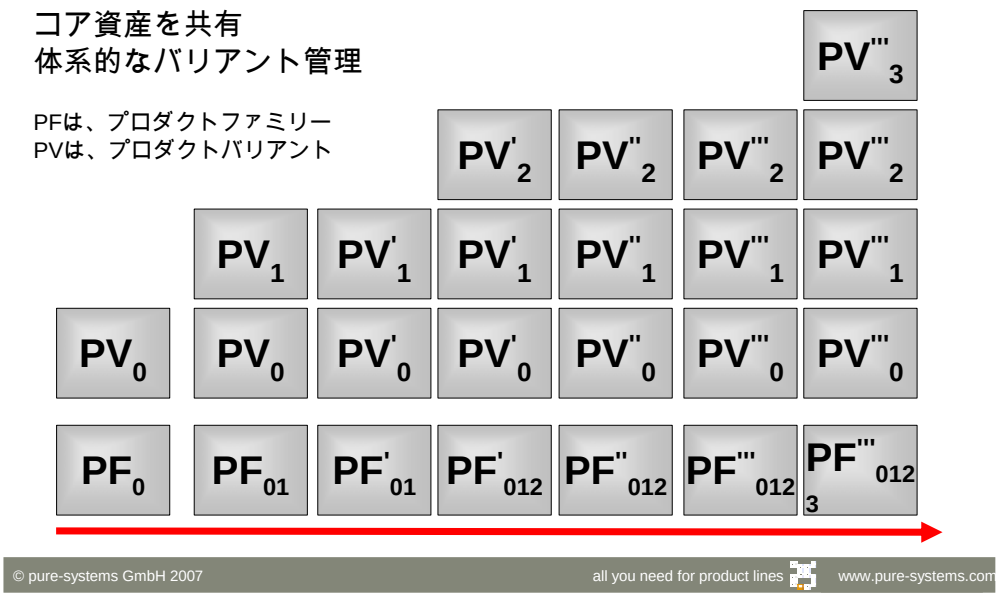
殆どの開発がこの構造に当てはまるでしょう。この構造は、プロダクトが互いに派生しあうけれども、一緒には開発されないような再利用の形態です。つまり、あるブランチに変更を行うと、変更を全てに反映させる為には、他のブランチにも手作業で同じ変更を加えなければならないような形態です。特に、この図のP0とP1のケースで、P0とP1を統合するような場合を想像してください。進化している、独立した2つのソフトウェアをマージする事が非常に大変な作業であると解るでしょう。（セミナー当日のアンケートでも、この構造であると答えられた方が、最も多かったです）



3つ目は、プロダクトギャングと呼ばれる構造です。

共有のコア資産を持ち、コンフィグレーションファイルを使って、ソリューションを構成する事で、プロダクトの生成を制御出来ます。その為、プロダクトファミリー（コア資産）から、要求されるプロダクトを即座に見つけ出せます。しかしながら、これらは各々独立しています。つまり、誰が開発したか、誰がコンフィグレーションファイルを作ったか等が解らない状態です。誰かが問題を発見して、プログラムを修正したいと考えた時に、その修正が他のどの部分に影響を与えるかがわからないような状況です。

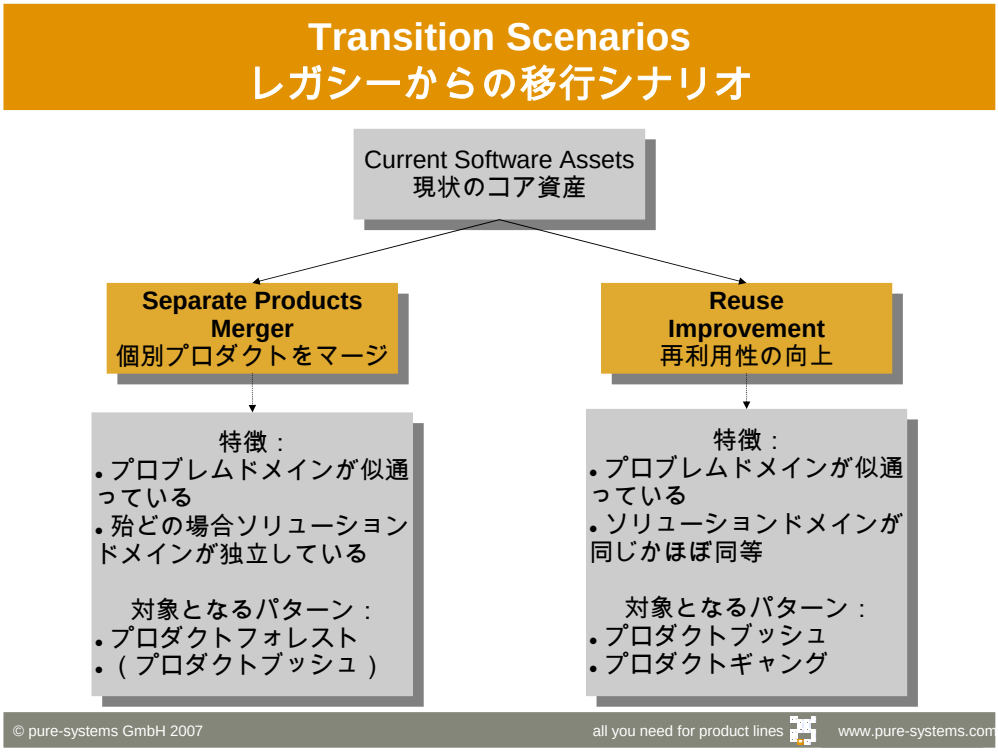
Product Relation Pattern: Product Family
プロダクトの相関パターン：プロダクトファミリー



理想的なプロダクトファミリー構造。

異なった或いは関連したバリアントに対する沢山の部品があります。これらの知識を使って、プロダクトラインで最大の利益を、リスク無しに得る事が出来ます。これが、我々が達成しようとしているプロダクトファミリーのコードの理想的なケースです。

どの時点であっても、ファミリーがどの様に使われているかが分かっている状態で、どこかに変更があっても、全てのプロダクトに正しく反映される状態です。pure::variants **を用いることで、これらの管理は容易になります。**

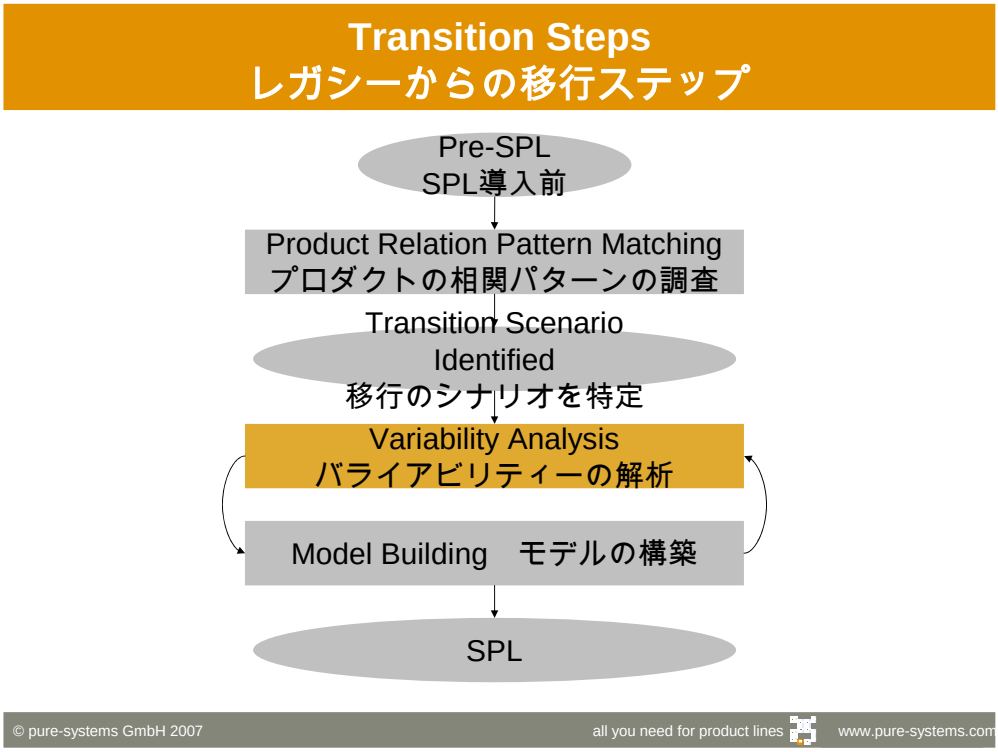


移行のシナリオには、2種類の方法があります。

1つは、異なるプロダクトをマージする手法 (separate product merger) で、似通ったプロブレムドメインを持ち、大部分が分離したソリューションドメインを持つ場合に有効な方法です。つまり、プロブレムドメインに関する知識は共有しているが、ソリューションドメインの既存の資産は再利用できないような場合です。これらは、独立して開発されているので、生成物を集約するのが困難です。プロダクトフォレストと一部のプロダクトブッシュ (ブランチがかけ離れているケース) に適用されます。

もう1つは再利用を向上させる手法 (reuse improvement) で、似通ったプロブレムドメインを持ち、同じか非常に似通ったソリューションドメインを持つ場合に有効な方法です。プロダクトブッシュとプロダクトギャングに適用されます。Danfos社の例が、このプロダクトブッシュに当たります。

これら2つの手法の違いは、何を利用しているかです。separate product mergerでは、主に、既存のソリューションドメインから特定のコア資産を使うであろうプロブレムドメインのフィーチャーの抽出にフォーカスしている。しかし、reuse improvementでは、既存のコア資産を出来るだけ使おうとしています。



移行のシナリオを特定すれば、次は、バライアビリティーの解析に進みます。

Variability Analysis
バライアビリティーの解析

- やるべきこと
  - プロダクトラインのバリエーションポイントの抽出と特定
  - バリエーションポイントの制約の抽出
- 見るべき場所
  - ソースコード
    - 構造、アルゴリズム、テクノロジー
  - ドキュメント
    - ユーザーマニュアル、内部ドキュメント、コードのコメント
  - 管理方法
    - バージョン管理、コンフィグレーション

© pure-systems GmbH 2007

先ずは、プロダクトのバリエーションポイントを特定･抽出します。同時に、それらバリエーションポイント間の関係も抽出します。

これには、幾つかの見るべき場所があります。ソースコードがその1つで、その構造、アルゴリズム、テクノロジー等から、バリエーションポイントが見つかります。ユーザーマニュアルやマーケティング用資料、コードのコメント等のドキュメント類も重要です。(プロダクトが違うバージョンや違う機能で構成されている場合に) バージョン管理ツールやコンフィグレーション方法からも有用な情報が得られるでしょう。

Variability Analysis (2)
バライアビリティーの解析（ 2 ）

- 得られた情報から定義を始めてみる
  - プロブレムスペースの最初のバリエーションポイント
  - ソリューションスペースの最初のバリエーションポイント
  - バリエーションポイントの最初の制約
- 狙い
  - 以下に対する入力を準備
    - product line scoping　プロダクトラインの範囲
    - problem space definition　プロブレムスペースの定義
    - core asset identification　コア資産の特定
    - production plan creation　生産計画の作成

© pure-systems GmbH 2007

得られたバリエーションポイントの情報から、プロブレムスペースとソリューションスペースに対して、バリエーションポイントの最初の手掛かりを定義します。また、これらバリエーションポイントの制約の最初の手掛かりも定義します。ここで、最初の手掛かりと表現しているのは、プロダクトラインを進めていく上で、これらバリエーションポイントの情報が変化し続けるからです。

また、これらバリエーションポイントを定義する事で、プロダクトラインの範囲の特定、プロブレムスペースの定義、コア資産の特定、製品計画の作成を行うときの入力を準備できます。

Variation Point Description
バリエーションポイントの定義

- バリエーションポイントは以下で定義される
  - そのポイントでの有効な選択肢 (フィーチャー)
  - バリエーションの制約 (requires/conflicts)
  - 帰属するスペース (プロブレム / ソリューション)
  - プロダクトラインのバライアビリティに関する関連性 / 重要性
  - 解決されるタイミング
    - 製品構成時、コンパイル時、リンク時、インストール時、ユーザーによる構成時、実行時
  - 解決方法
    - スタティック　ダイナミック
  - 使用するバライアビリティーのパターン

© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

バリエーションポイント定義の構成要素について説明しています。

解決方法のスタティックとダイナミックのちがいは、スタティックとは例えば、車の色が赤とか黒という場合、ダイナミックとは、例えば、カーナビが付いていたらバックモニターが作動すると言う様な場合です。

Analysis Direction
解析の方向性

- どちら側から解析を行うか
  - プロブレムスペースのバライアビリティーから始める
    - ユーザーマニュアルや製品仕様が既存製品間のバライアビリティーの実態を上手く反映している場合に最適：プロダクトフォレストに向く
    - 既存のコア資産を確認すること無しにプロダクトラインを適用できる
  - ソリューションスペースのバライアビリティーから始める
    - ソフトウェアのデザインやアーキテクチャーがプロブレムスペースを上手く反映している場合に最適：特にプロダクトブッシュとプロダクトギャングに向く
    - プロブレムスペースのバライアビリティーを抽出する事で、プロダクトラインの適用が始められる

© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

ユーザーマニュアルや製品仕様が、既存製品間のバライアビリティーを上手く表現している場合には、プロブレムスペースのバライアビリティーから始めるのが良い方法です。既存のコア資産を確認する事無く、プロダクトラインの適用を開始できます。プロダクトフォレストに最適な方法です。

逆に、ソフトウェアのデザインやアーキテクチャーがプロブレムスペースを上手く反映している場合には、ソリューションスペースのバライアビリティーから始めるのが良いでしょう。ソリューションスペースからプロブレムスペースのバライアビリティーを抽出する事で、プロダクトラインの適用が始められます。プロダクトブッシュとプロダクトギャングに最適な方法です。

Danfos**社では、最初にプロブレムスペースからの方法を試み、1ヶ月かけても成果が上がりませんでしたが、ソリューションスペースからの方法に切り替えることで、2週間で成果を上げることができました。**

マーケティングや営業などのユーザーに近い人たちは上の方法を、技術者は下の方法を好む傾向にあります。

Variation Point Extraction
バリエーションポイントの抽出

1. バライアビリティーを探す場所を決定（マニュアルやソースコードから）

2. ソフトウェアの代表的なバリエーションポイントのパターンのリストを作成

3. バライアビリティーパターンを探す

   - 手動で

   - ツールを使って

© pure-systems GmbH 2007

バリエーションポイントを探す方法は、
先ず、バライアビリティーを探す場所（マニュアル、ソースコード等）を決定し、
次に、代表的なバリエーションポイントのパターンをリストアップし、
最後に、手作業（残念ながら、殆どの場合）又はツールを使って（使える場合もある）、上記リストにマッチするパターンをバリエーションパターンとして見つける
と言う手順です。

Problem Space VP Extraction
プロブレムスペースのVP抽出

- 典型的な入力
  - ユーザードキュメント
    - インストール / ユーザーマニュアル
    - ホワイトペーパー
  - 開発ドキュメント
    - 要求仕様、ユーズケース
    - アーキテクチャー設計ドキュメント
    - バージョン管理システムのログ（特にプロダクトブッシュ）
    - コンフィグレーションファイル（特にプロダクトギャング）
    - 顧客とのコミュニケーション（機能要求等）

© pure-systems GmbH 2007

この後の数枚のスライドを使って、プロブレムスペースとソリューションスペースのバリエーションポイントの抽出方法を解説します。

先ず、プロブレムスペースのバリエーションポイントです。インストールマニュアル、ユーザーマニュアル、ホワイトペーパー等のユーザードキュメントが典型的な入力になります。また、要求仕様、ユーズケース等の開発ドキュメントを参照することもあります。後のスライドでも出てきますが、顧客とのコミュニケーションもプロダクトラインでは重要な項目です。なぜなら、顧客が必要としている機能を盛り込む事は、製品にとって重要だからです。

プロブレムスペースから共通性とバライアビリティを抽出　アプローチの事例

- フラウンホーファー　IESEのCAFEプロジェクトで、PLSEのフレームワークの一部として開発（ **P**roduct **L**ine **S**oftware **E**ngineering Framework ）
- ユーザードキュメントにフォーカス、開発ドキュメントにも使用可能
- パターンに着目して、テキストを抽出して、比較する

© pure-systems GmbH 2007

プロブレムスペースから共通性とバライアビリティーを抽出する為のアプローチとして、以前に使用して上手く機能した方法を紹介します。

それは、フラウンホーファー研究所の研究プロジェクトで“プロダクトライン・ソフトウェア・エンジニアリング・フレームワーク”の一部として開発されたCaVEという手法です。

これは、ユーザードキュメントにフォーカスした手法ですが、開発ドキュメントにも適用可能で、パターンに着目して、ドキュメントをテキストベースで比較する方法です。

CaVE – Approach

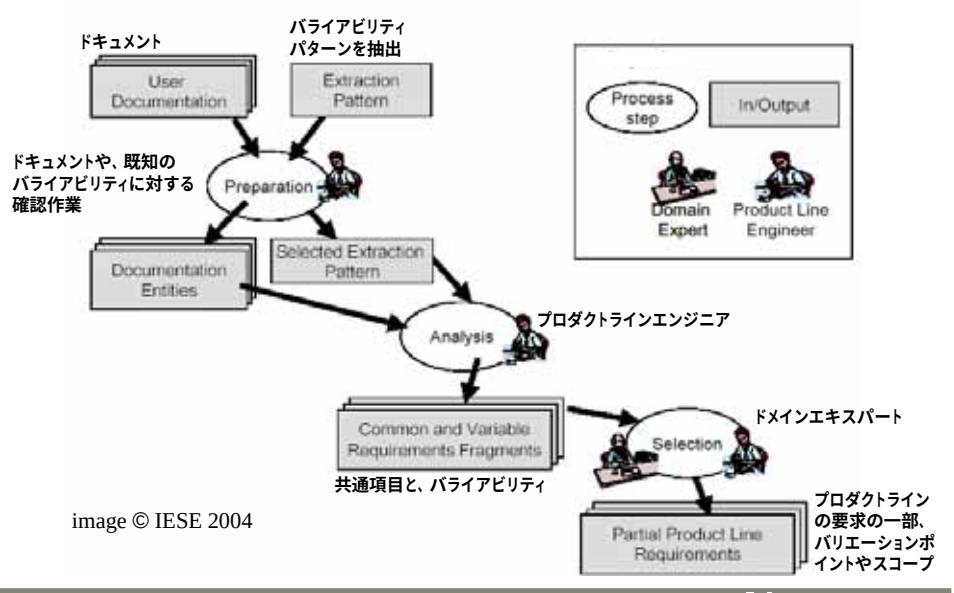


image © IESE 2004

© pure-systems GmbH 2007

この図の左上の部分にドキュメントと抽出パターン（バライアビリティーパターン）があります。プロダクトラインエンジニアがこれらドキュメントや既知のバライアビリティーから、本当に必要なものを抽出します。抽出された結果から、さらに、プロダクトラインエンジニアが共通項目とバライアビリティーを構成します。最後にドメインエキスパートと一緒にこれらをチェックし、プロダクトラインの要求（バリエーションポイントやスコープ）を導出します。

興味深い点は、ドメインエキスパートとプロダクトラインエンジニアの両方の作業である点と、プロダクトラインエンジニアは、ドメインに詳しくなくてもよい点です。

ドメインエキスパートは通常非常に忙しいため、プロダクトラインエンジニアを別のチームや外部委託に出来る事が、大きなメリットになります。

CaVE パターンのテンプレート

Name	The name of the pattern.
Short Description	A one sentence description of the pattern.
Input	The input model element.
Output	The output model element.
Recall	The average recall (the percentage of the total relevant elements retrieved by the pattern).
Precision	The average precision (the percentage of relevant elements in relation to the number of total elements retrieved).
Value	The value or relevance this pattern has for the stakeholder “domain expert” (given as --,-, o, +, ++).
Transition	The input and output level for the pattern in the conceptual model.
Long Description	A longer description of the pattern, including keywords.
Related Pattern	A list of other patterns this pattern is related to (e.g. composed of these pattern).
Example	An example for an input element where the pattern holds and the elicited result.

© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

CaVE**パターンのテンプレートです。**

パターンの名前 (Name)**、簡単な説明** (Short Description) **、入出力** (Input/Output) **モデル、入出力の形態** (Transition) **：何から何を取り出すか** (**例：ドキュメントの項目からフィーチャーエレメント**) **、詳細な説明** (Long description) **、例** (Example) **、関連する他のパターン** (Related Pattern) **等を記述します。**

また、このパターンによって取り出されるエレメントの割合 (Recall) **、抽出された全エレメントに対して、正しく抽出されている割合** (Precision) **、ドメインエキスパートの視点で見たこのパターンの価値** (Value) **：このパターンで抽出して意味があるか？　等の評価も記述します。**

CaVE - Pattern Example (1)

Name	Headings
Short Description	Headings usually represent features
Input	Headings
Output	Feature
Recall	+
Precision	++
Value	-
Transition	Transition Documentation -> Product Line Artifact
Long Description	Since features describe functionalities that are of importance for the user, they are found at prominent places in the user documentation.
Related Pattern	
Example	In a mobile phone user documentation “Sending an SMS” is a heading that describes a feature

© pure-systems GmbH 2007

ドキュメントの見出しをパターンとして設定している例です。ユーザの観点で記載されているため、ユーザドキュメントの見出しから、フィーチャを抽出できます。例えば、携帯電話の“SMSメッセージ送信”など。ただ判りきったことも多い為、ドメインエキスパートにとってはそれほど重要な情報でない事が多いです。

CaVE - Pattern Example (2)

Name	Parameter-Value
Short Description	If Sentences or Phrases are identical in different documents but include a different numerical value, this can be a parametrical variability or alternative values
Input	Sentence; phrase
Output	Alternative
Recall	-
Precision	++
Value	+
Transition	user documentation -> requirements concept
Long Description	Parameters in the systems that have a different value could be realized differently in the software
Related Pattern	
Example	“The phone can send SMS with at most 124 characters” <-> “The phone can send SMS with at most 136 characters”

© pure-systems GmbH 2007

より複雑な例、異なるドキュメントに、同じ文やフレーズがあり、それに続く数値が異なっているものを抽出するパターンの例です。

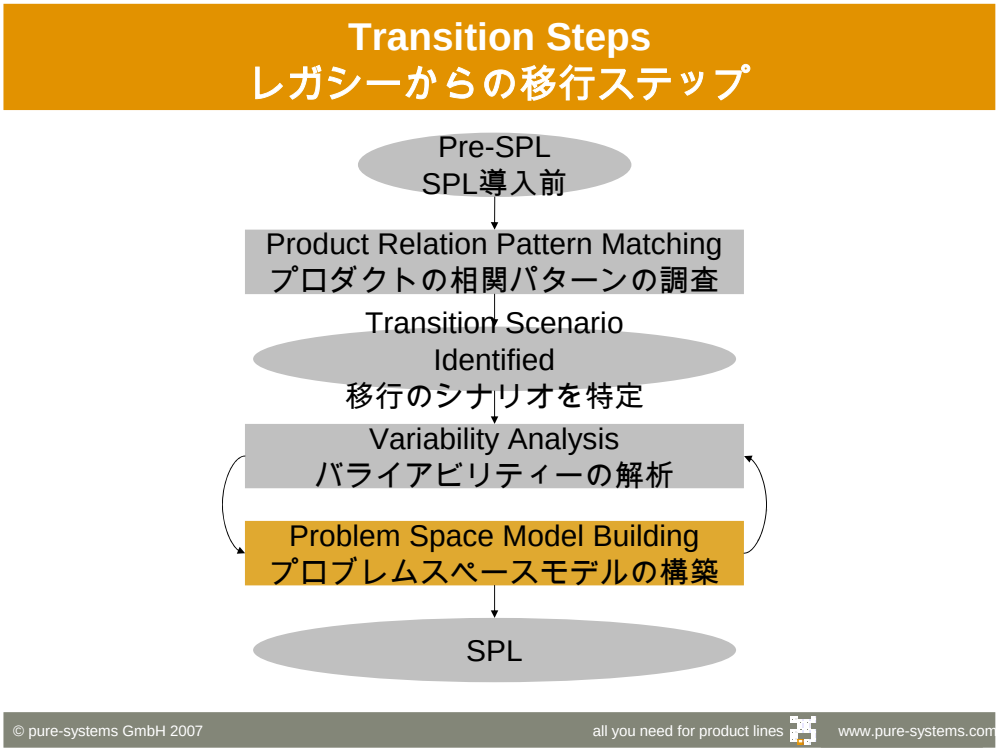
これらは、異なるパラメータを持ったバリエーションになるます。

CaVE - Summary

- 長所
  - プロダクトラインエンジニア、プロダクトライン開発のエキスパート、社内のプロブレムスペースのエキスパートの仕事を分離し、ドメインエキスパートの負荷を軽減する
  - 産業におけるケーススタディーが[CaVE]に十分文書化されている
- 短所
  - 非常に多種に渡る或いは大きなドキュメントによる実例ない
  - サポートするツールが殆ど無い

© pure-systems GmbH 2007

CaVEアプローチの長所と短所をまとめました。



ここまで、プロダクトの相関パターンの調査、移行シナリオの特定、バライアビリティーの解析を行ってきました。

次に、これらの情報を基に、プロブレムスペースモデル（フィーチャーモデル）を構築します。

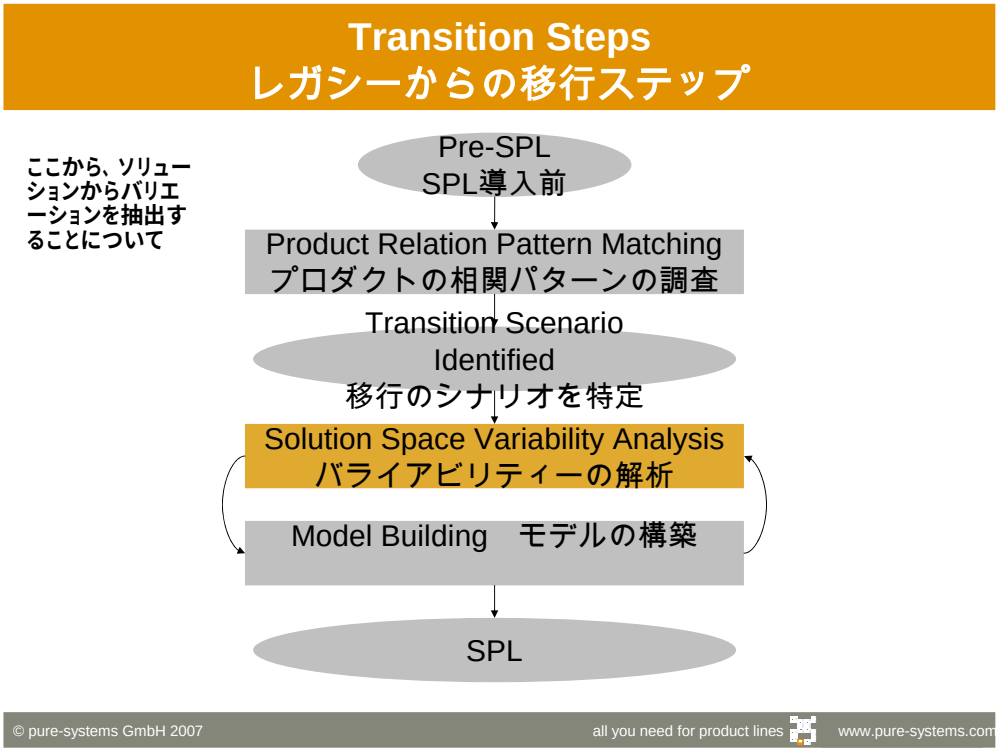
プロブレムスペースのモデリング：フィーチャーモデリング

- フィーチャーモデルはバリエーションポイントをフォーマルに記載するために、最も頻繁に用いられる。
  - フィーチャーモデルのコンセプトは分かりやすい
  - 複雑なフィーチャーモデルでさえ、扱いが簡単
  - 殆どの実社会の課題を表現する十分な能力を持つ

© pure-systems GmbH 2007

フィーチャモデルの特徴を紹介しています。

フィーチャモデル或いはフィーチャモデリングという用語は、pure::variants**独特の用語ではなく、プロダクトライン開発に於いて一般的に使用される用語です。 また、“この手法に対して、**UML**はどんなものでも含める事が出来るので難しい”といわれています。**



ここまで、プロブレムスペースからバリエーションを抽出する方法について説明しました。
ここから、ソリューションスペースからバリエーションを抽出する方法について説明します。

Solution Space VP Extraction
ソリューションスペースVPの抽出

- 典型的な入力
  - アーキテクチャー設計ドキュメント
  - ソースコード
  - コンフィグレーションスクリプト
  - プロジェクトのビルド定義（例えば、makefile）
  - バージョン管理システムの構造と用法
  - コンフィグレーションファイルとファイルフォーマット

© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

これら入力により、フォーマルなモデル化が実現できます。ソリューションスペース、パターンを定義するのは、フォーマルであることが望ましいです。

これらの中から幾つかのパターンについて解説します。

Solution Space VP – Pattern #1

Name	#1 Preprocessor-based Text Fragment Selection
Short Description	If source code files contain preprocessor statements which conditionially (de)activate program statements, this usually identifies a variation point
Input	source file
Output	selection conditions and functional differences
Environment	C/C++ code, frame/macro processor input files, code generator input
Precision	+
Value	o
Transition	source code -> variation point
Long Description	The textual exchange of program statements is an often used concept to encapusulate platform specific code or to optionally add functionality statically during compile time.
Related Pattern	#2, #6

Example	`#ifdef FEA_TEMP_ALARM`
	`  if (t > t_al) send_alarm(TEMP_ALARM,t);`
	`#endif`

© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

CやC++でよく使われる、＃IFや＃IFDEFを使って、ソースコードを切り替えているパターンの例です。

Solution Space VP – Pattern #2

Name	# 2 Constant/variable based configuration
Short Description	If a source code files contains almost exclusively constant or variable definitions, it may be used for configuration purposes.
Input	source file
Output	selection conditions and functional differences
Environment	All programming languages
Precision	+
Value	+
Transition	source code -> variation point
Long Description	Constants defined in a file/files referenced by many other files usually provide access to shared information such as compile and/or runtime configuration
Related Pattern	#1

Example	C header files:
	`#define FEA_TEMP_ALARM 1`
	`const int max_buffer_size = 512;`
	`int max_buffer_count = 8;`

© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

定数を定義する部分を抽出するパターンの例です。定数は、バリエーションのパラーメータになります。

Solution Space VP – Pattern #3

Name	# 3 Use of Build Configurations
Short Description	IDE or build tool uses configuration files to produce variants
Input	build configuration information
Output	alternative oo hierarchy and/or functional differences
Environment	IDE or build tool like make, ant
Precision	++
Value	++
Transition	configuration file -> structural and/or functional variability
Long Description	IDE or project build tools such as make can be parametrized and thus produce different output for the same input (file structure).
Related Pattern	#6
Example	make could be used with different makefiles/make variable settings as input to generate different builds of a project. The differences in makefiles may be stored in different files or as different versions&branches of same file in version control systems. Differences in parameters are often visible in build scripts.

© pure-systems GmbH 2007

コンフィグレーションファイルからバライアビリティーを抽出する例です。

Solution Space VP – Pattern #4

Name	# 4 Conditional code execution
Short Description	Conditional code execution is controlled by configuration values
Input	source code
Output	conditions for alternative state flow/event processing/...
Environment	All programming languages
Precision	o
Value	o
Transition	source code -> functional variability
Long Description	Some conditional execution paths are controlled by configuration data instead of user data. Indicators are that the condition references configuration constants or uses methods to access configuration values such as getters for global configuration objects
Related Pattern	#2
Example	`if (config.getValue("TEMP_ALARM" && t > t_al) { alarm.send("TEMP_ALARM",t); }`

© pure-systems GmbH 2007

(コンフィグレーション) ファイルに記述された値によって、プログラムの動作が変わる部分を抽出する例です。

Solution Space VP – Pattern #5

Short Description	#5 Customer-specific product variants are managed using branches in version control systems
Input	version control system usage policy / version structure
Output	list of customer specific product variants
Environment	version control system
Precision	o
Value	+
Transition	version branches -> product variants
Long Description	Customer-specific product variants are often create by copying parts or whole software systems into separate branches in version control systems. Often naming of branches reveals such a policy. Also combined with labeling of version representing product variants.
Related Pattern	
Example	List of branch names: BMW_V1 PSA_V1 VW_V2

© pure-systems GmbH 2007

バージョン管理システムのブランチをバリアントとして抽出する例です。

Solution Space VP – Pattern #6

Name	# 6 File level variation
Short Description	Different files with similar names are used as alternative implementations
Input	file name list
Output	variation point
Environment	any language
Precision	+
Value	+
Transition	files -> variant point alternatives
Long Description	For implementation of static code variability where the code changes significantly between variants, often each variant is placed in a separate file. The configuration mechanism selects one of the files at compile time latest.
Related Pattern	#1, #3
Example	List of file names: temp_normal.c temp_alarm_sound.c temp_alarm_visual.c

© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

ビルドするファイルにバリエーションがあるときに、それを抽出する例です。

Solution Space VP – Pattern #7

Name	# 7 Configurable Factory object
Short Description	The software uses the factory pattern and the factory object is configurable
Input	factory pattern members
Output	variation point and variation point constraints
Environment	any language
Precision	o
Value	o
Transition	architecture pattern -> variant point alternatives
Long Description	The factory pattern is used to decouple the act of object creation from the place where the object is created. If the type or implementation of the factory object is changeable by configuration, it may represent a variability
Related Pattern	
Example	

© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

製造時にバリエーションを持たせるような製品のパターンを抽出する例です。

Solution Space VP Detection
パターンからソリューションスペースのVPを検出

- 殆どの場合、少しのコードを良く見ると、使用されているパターンが明確になる（プログラマーは繰り返しパターンを使いがちであるので、小さな範囲内で得られるパターンは限られる）
- 大抵コード構造が限定された構文に組み込まれるので、ソリューションスペースのパターンは、単純なツールを使って、より簡単に検出できる。
- 問題は、検出されるバリエーションポイント候補の数は、大抵非常に多くなる（ #if def など）

© pure-systems GmbH 2007

ソリューションスペース内のパターンは、ここにあるような特徴を持っています。

ユーザドキュメントなどと比較してフォーマルであるので、自動的に解析するツールを見つけたり、作ったりする事が容易でしょう。

ソリューションスペースのVP　解析ツールの例

Pattern #1　パターン　1

UNIXの標準ツール（Windows版もあり）

全ての#if[n]?def/#elif (#define)ステートメントを数える

```
find  -name "*.[hc]*"  -exec egrep "(#if[n]?def|#elif)" {} ¥; |
awk '{ print $2 }' |sort | uniq -c | sort -r >ifdef.lst
```

共通部分を解析する

```
awk '{ print $2 }' ifdef.lst defines.lst | sort | uniq -d
```

#if[n]?def/#elif を使っているファイルを抽出する

```
ifnames `find -name “*.[hc]*”`
```

© pure-systems GmbH 2007

pure::variantsは、これらのパターンを自動的に解析する機能は持っていませんが、市販或いは無料で使えるツールが幾つかあります。

この例は、UNIXの標準コマンドを使った例です。

ソリューションスペースVP　解析ツールの例

CVSNT（CVSのクロスプラットフォーム版）

- 327 KLOC
- 1079 different constant used in `#if*` conditions
- 3775 different `#define` constants
- Constants defined in
  - makefiles
  - (generated) header-files
  - implementation files
  - Visual Studio Project files

Top 20 #if* constants

# Occur.	Name	Defined internally
154	_WIN32	
124	__cplusplus	
109	SERVER_SUPPORT	yes
74	SUPPORT_UTF8	
61	XML_DTD	yes
52	DEBUG	yes
43	WIN32	yes
33	emacs	
32	IPV6	
26	SUPPORT_UCP	yes
25	HAVE_CONFIG_H	yes
25	CVS95	yes
24	_DEBUG	yes
24	USE_SHARED_LIBS	yes
23	XML_NS	yes
21	UTIME_EXPECTS_WRITABLE	yes
19	CVSGUI_PIPE	yes
14	_UNICODE	yes

© pure-systems GmbH 2007

CVSNTというオープンソースのプログラムのソースを解析した結果を例として示しました。

#IFで1079個の定数を使っている、#defineで3775個の定数を定義していると言った結果や、右側には

ON/OFF制御されるSERVER_SUPPORTフラグや

ドメインに詳しくなくても、機能としてのバリエーションポイントであると想像できるSUPPORT_UCPなどのフラグが見つかっており、ソース内での使用回数の情報が表示されています。

Solution Space VP Analysis Tools
ソリューションスペースVP　解析ツール

DMS (SemanticDesign):　言語アナライザーと変換プログラムをベースにしたルール、沢山の言語に対応

PUMA*:　C/C++　パーサーとマニピュレータ（AspectC++の一部）

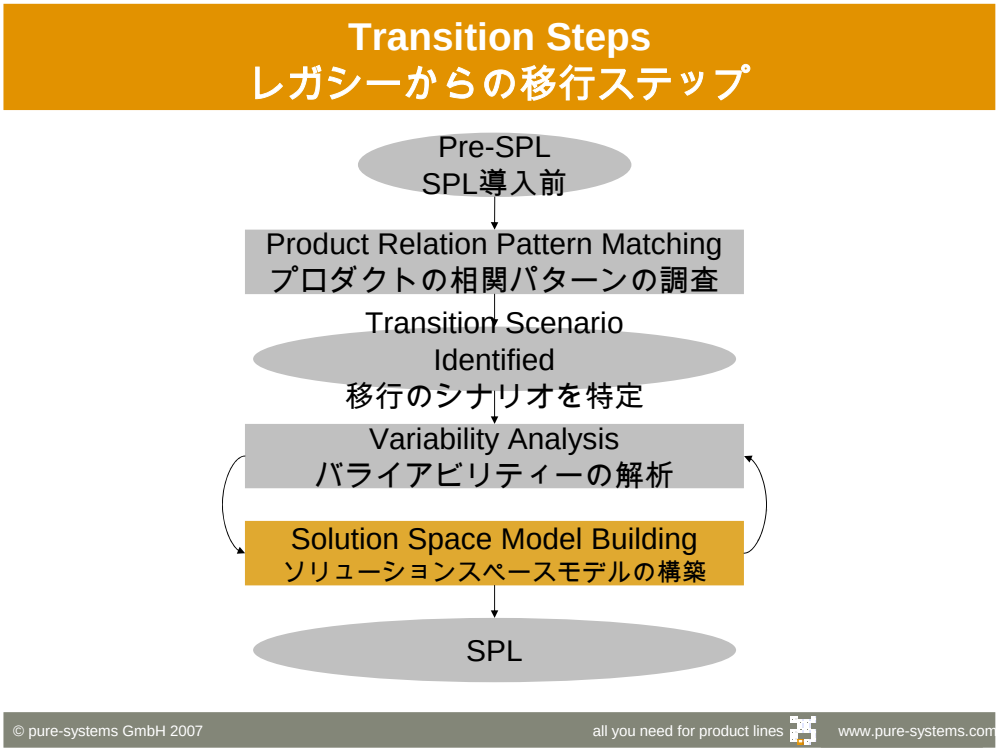
AspectJ*: Java, pointcuts and “declare”

Eclipse JDT*/CDT*: internal analysers, good analysis quality esp. for Java　内臓の解析機能、（特にJAVAで）解析性能が良い、

* open source/freeware

© pure-systems GmbH 2007

その他のツールの例です。
パターン、プロジェクトのサイズ、使用する言語によって選択肢が変わるでしょう。
DMSと言うツールは、沢山の言語に対応したツールで、非常に高機能ですが、非常に高価です。ボーイング社が使っているそうです。それ以外は、オープンソースのソフトウェアで、特定の言語に特化しています。



続いて、ソリューションスペースモデルの構築方法を説明します。

Solution Space – Architecture
ソリューションスペースの基本設計

- 参照される基準としてのアーキテクチャー
  - プロダクトから復元できる
    - プロダクトギャング、プロダクトブッシュ
  - 単一のプロダクト又はスクラッチから構築が必要
    - プロダクトフォレスト
- コア資産の特定
  - コンポーネントにアーキテクチャー特定の重み付け
    - 将来の製品での使用可能性
    - 既存製品での使用率
    - 適用により期待できる効果
  - 全ての既存資産を使う必要は無い！

© pure-systems GmbH 2007

ソリューションスペースのモデルは、設計情報やソースコードを基にして行われます。

また、モデルを作成する上で、コア資産の特定も行います。コンポーネントに、使用率や将来の使用可能性、効果による重み付けを行い特定します。全ての既存資産を使う必要は無く、バリエーションポイントを作成する事で十分な効果が得られる部分を抽出する事が重要です。

製品化計画：プロブレムとソリューションのマッピング

- プロブレムスペースの選択が、常にソリューションスペースのバリーエーションポイントに１対１にマップするわけではない
- プロブレムスペースからソリューションスペースのマッピングには、プロブレムスペースの構成からコア資産を構成 / 選択する事が必要
- pure::variantsのファミリーモデルがソリューションスペースへのモデリングとマッピングに利用される。大抵、他のツールと組み合わせて使われる

© pure-systems GmbH 2007

pure::variants **では、ファミリーモデルとしてソリューションスペースをモデル化し、フィーチャモデルへとマップすることができます。このファミリーモデル上のバリエーションポイントは、マップされたフィーチャモデルから独立している為、外部のコードジェネレータで必要となる情報を生成することにも利用できます。**

Production Plan Automation: Tool Support
製品化計画（自動化）：ツールによるサポート

- スクリプトベースのコンフィグレーションツール

  autoconf*: UNIXのコンフィグレーションツール、特にC/C++言語を使ったマルチプラットフォーム開発で使われる

- MDSD tools　モデル駆動型システム開発ツール

  openArchitectureWare*: JAVAベースのモデル変換、コード生成ツール、適当な入力モデルから特有のコードを生成する

- Product Line tools　プロダクトラインツール

  pure::variants: explicit solution space modelling with integrated/external model transformation　ソリューションスペースモデルから、外部のモデルに変換できる（コード生成ツールなどに対し）

© pure-systems GmbH 2007

ソリューションスペースから特定の製品を作成する為のツールの例です。

管理の観点からの
ソフトウェアプロダクトライン

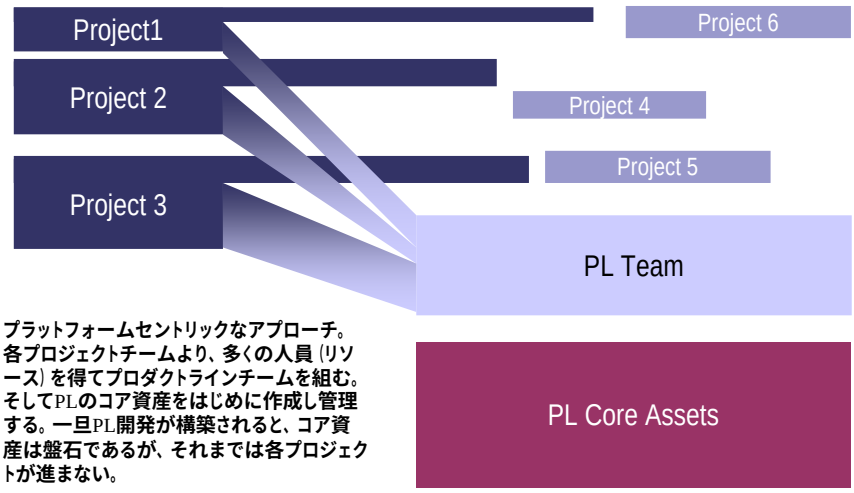
管理の観点から見たソフトウェアプロダクトラインについて説明します。

バリアント管理への移行

- レガシーシステムからプロダクトライン開発への移行は、一瞬では起こらない。プロセスである（学習し、適合させる）
- 次のことを前提に、正しいアプローチを選ぶ
  - 履歴と期待される将来
  - 準備できる資金
  - 製品化までの時間の要求（そして、どの位早くにプロダクトライン開発を立ち上げる必要があるか）
  - 組織のスキル

ソフトウェアプロダクトラインの管理は、複雑なプロセスになります。以降の幾つかのスライドで、移行のプロセスについて説明します。ここでは、移行に際して注意が必要なことを記述しています。

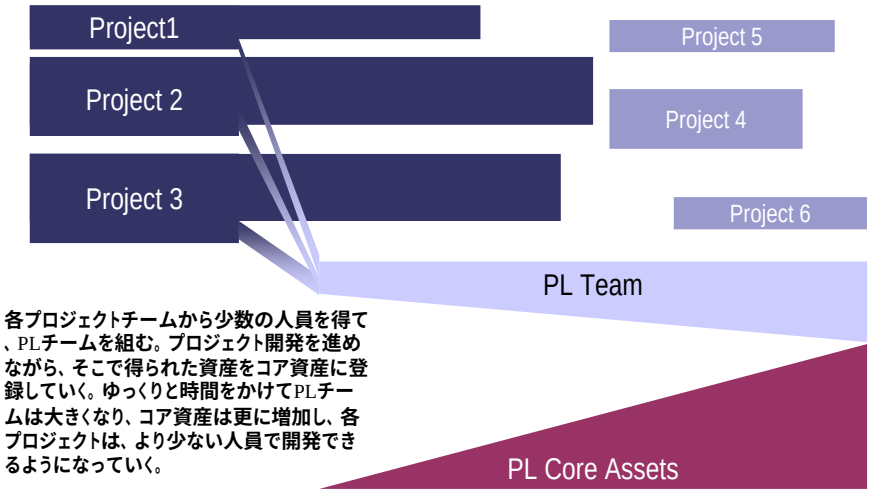
アプローチ：プラットフォームセントリック



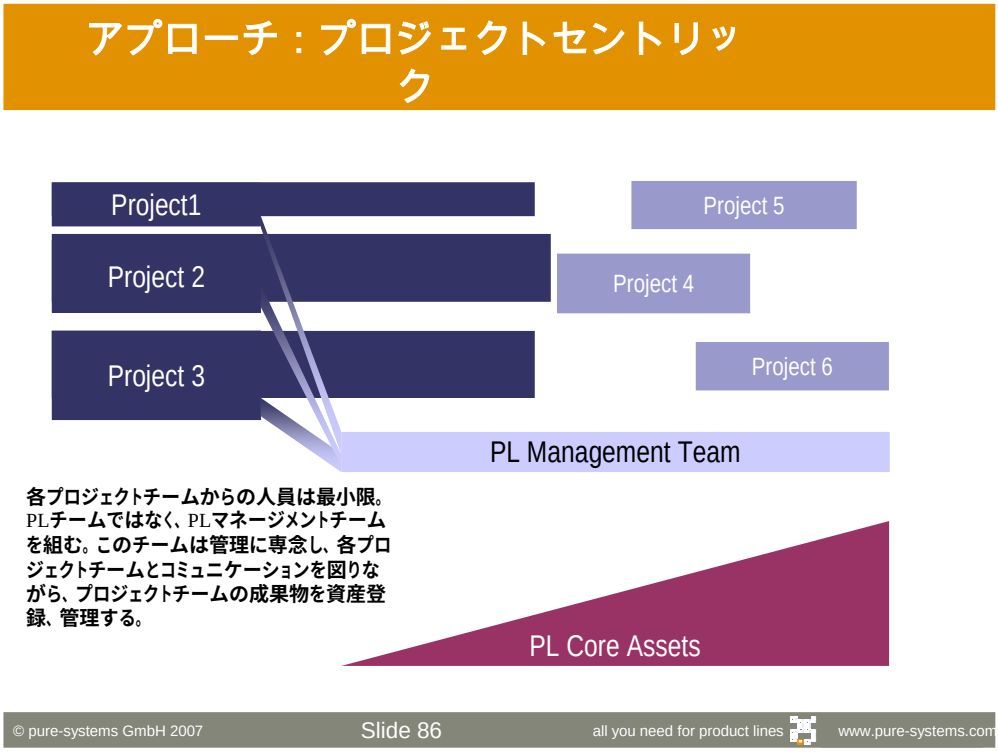
プラットフォームセントリックなアプローチ。
各プロジェクトチームより、多くの人員（リソース）を得てプロダクトラインチームを組む。
そしてPLのコア資産をはじめに作成し管理する。一旦PL開発が構築されると、コア資産は盤石であるが、それまでは各プロジェクトが進まない。
各プロジェクトのバス幅が細いことに注目。

プラットフォームセントリックなアプローチです。各プロジェクトチームより、多くの人員（リソース）を得てプロダクトラインチームを組織します。そしてプロダクトラインのコア資産をはじめに作成し管理します。一旦PL開発が構築されると、コア資産は盤石となりますが、それまでは各プロジェクトの進行が遅れます。（各プロジェクトのバス幅が細いことに注目）

アプローチ：進化型プラットフォームセントリック



各プロジェクトチームから少数の人員を得て、プロダクトラインチームを組むアプローチです。プロジェクト開発を進めながら、そこで得られた資産をコア資産に登録していく方法で、ゆっくりと時間をかけてチームは大きくなり、コア資産は更に増加し、各プロジェクトは、より少ない人員で開発できるようになって行きます。



各プロジェクトチームからの人員が最小限になるアプローチです。プロダクトラインチームではなく、プロダクトラインマネージメントチームを組む方法です。このチームは管理に専念し、各プロジェクトチームとコミュニケーションを図りながら、プロジェクトチームの成果物を資産登録、管理する事でコア資産を作成して行きます。

アプローチの比較

Platform-Centric

+ 長所
  - プロジェクトへの影響が最小
  - レガシーが無くても始められる
- 短所
  - 組織への影響大
  - 開始に時間がかかる
  - リスクが高い

Incremental Platform

+ 長所
  - 限定的なリスク
  - 徐々に組織を変えて行ける
- 短所
  - 再利用性の向上が遅い目

Project-Centric

+ 長所
  - 最小のリスク
  - 組織的な変更が少ない
  - 投資回収が早い
- 短所
  - 最善の再利用規則が求められる
  - レガシーシステムを再利用する必要がある

© pure-systems GmbH 2007

3つのアプローチの長所と短所を纏めました。
顧客事例で紹介したDanfos**の例が**Project-Centric**の例になります。**
Incremental-Platform－Centric**が最もよく利用されるアプローチです。**
Platfor-Centric**のアプローチを取った場合、失敗するケースが多いです。**

Summary
サマリー

- コスト削減のためには、バライアビリティーを少ない目に。バリエーションの構築· 管理に要するコストと、効率の比較
- バライアビリティーの設定が必須、あるいは採算が合う場合は、バリアント、バリエーションポイントを明確に定義すること。（これらは多くなり、その関係は複雑なので）
- 系統だったバリアントとバライアビリティーの管理が、再利用成功の鍵（管理、保守すること無しに達成し得ない）
- 全てを一度に行おうとしない。徐々に進めていくことが労力とリスクの両方を最小にする。
- 最初に行うときに、経験者の意見を聞く（良い経験も悪い経験も）

どのアプローチを取るかは、組織の構造や移行のシナリオに依存するでしょう。
ここに、移行を成功させる為に注意すべき点を纏めました。

サマリー

- 既存のレガシーソフトウェアをプロダクトラインに転換する事が可能
  - 移行を始める前に、念入りなリスク評価が必要
- 結果は以下に依存する
  - 抽出できる知識の量と質
  - 移行チームのスキルと管理者のサポート

© pure-systems GmbH 2007

また、プロダクトライン開発を実現させると言う根本理念の確立が最も重要です。

References & Tool Links

[SPLP-FW] Software Engineering Institute, *A Framework for Software Product Line Practice Version 4.2*, 2005, http://www.sei.cmu.edu/productlines/framework.html

[CaVE] John, I.; Doerr, J.; Schmid, K.: User documentation based product line modeling, IESE-Report, 004.04/E
http://www.iese.fraunhofer.de/pdf_files/iese-004_04.pdf

[FODA] Kang, K. et al: Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990

[Kolb] Kolb, R.; Muthig, D.; Yamauchi, K.: *Migration existierender Softwarekomponenten in eine Produktfamilie* (engl. Migration of existing software components in product families), ObjektSpektrum 04/2005 (in german)
http://www.sigs.de/publications/os/2005/04/yamauchi_kolb_OS_04_05.pdf

AspectJ: www.eclipse.org/aspectj

autoconf: www.gnu.org/software/autoconf

DMS: www.semanticdesigns.com

Eclipse CDT: www.eclipse.org/cdt

openArchitectureWare: architecturware.sourceforge.net

PUMA: www.aspectc.org

pure::variants: www.pure-systems.com

© pure-systems GmbH 2007

参考資料やツールの紹介です。

バリアント：
テストと欠陥へのリンク

この後の幾つかのスライドで、バリアントをテストや欠陥にリンクする方法を紹介します。

課題

バリアントが多いシステムにおいて、全てのバリアントに対して、テストされているか、欠陥管理されているか？

以下の疑問に答えなければならない

- どのテスト、不具合、要求の変更が、どのバリアントに対応しているか？

- 個々のバリアントに対して、これらのアイテムの現状は？

アプローチは既存のデータベースやツールに対応できること

多くのバリアントを持つシステムに於いて、全てのバリアントに対するテストと欠陥の管理が必要です。既存の資産は既にテストされ、その結果がデータベース化されているでしょう。従って、既存のデータベースやツールを使って、バリアントごとのテストと欠陥が管理できる必要があります。

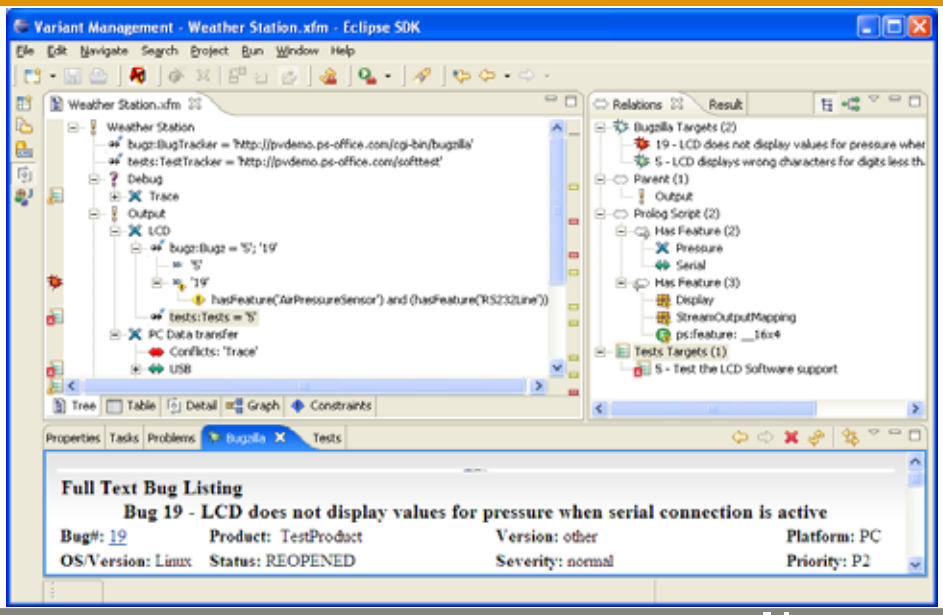
pure::variantsのアプローチ

- pure::variantsはプロダクトのバライアビリティーの管理に用いられる。各々の関連するプロダクトはバリアントモデルで定義される
- 各々のpure::variantsのエレメント（フィーチャー、コンポーネント、パーツ、ソース）は、ターゲットリストアトリビュートを使って、外部のアイテム（テスト、テスト結果、バグ、変更要求）にリンクできる
- 接続された各々のツールやデータベースへのリンクを、pure::variantsのコネクタープラグインがオンデマンドで、リアルタイムに解決する
- ツールによっては、不具合やテストケースの追加のようなアクションをpure::variants側から、コネクターを通して実行できる

pure::variants**は、（**BUGZILLA、ClearQuest**等の）外部ツールと統合する事で、この問題を解決します。**

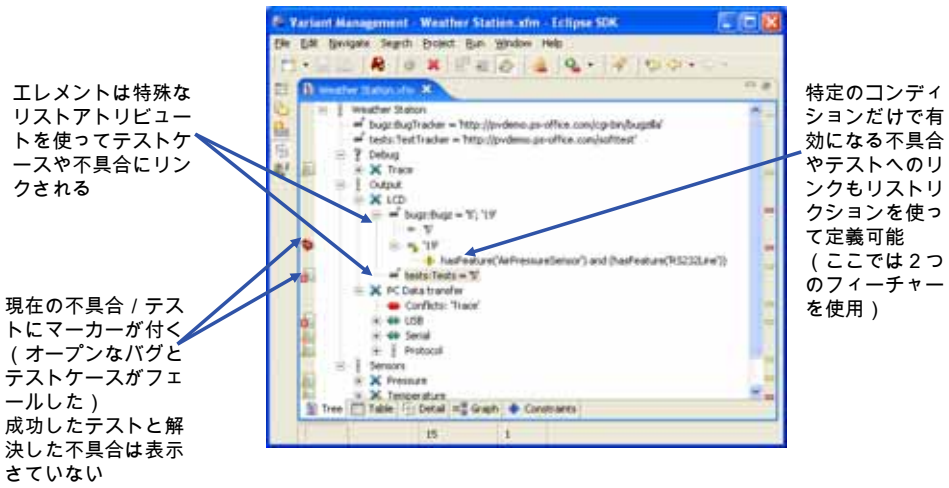
以降の数枚のスライドで、その例を紹介します。

pure::variants – Sample Workspace



BUGZILLAを使って、バリアントの欠陥管理を行っている例です。

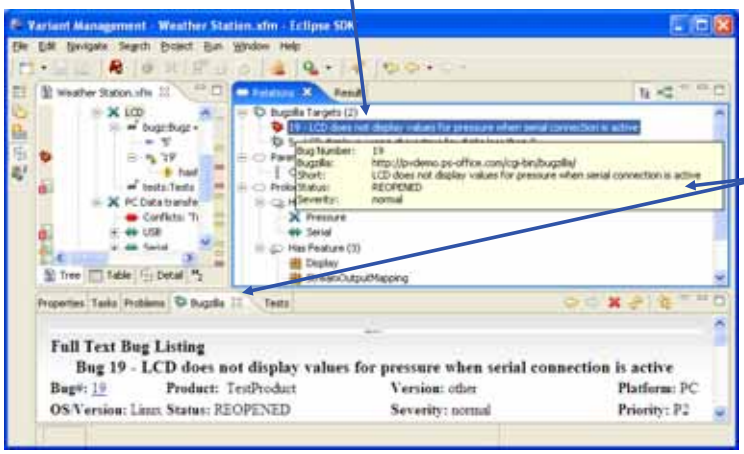
Model with Test and Defect Information



フィーチャーにマップされた欠陥やテストにマーカーが表示されています。

Relations View – Quick Target Info Access

リレーション表示では、選択された工レメントに関連した
リレーションのリストが表示される
不具合とテストの為に、サマリーが表示されている

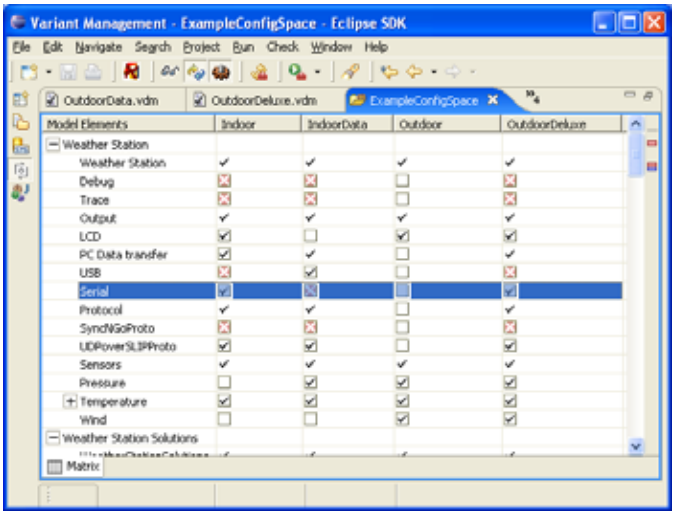


専用のビューとツールで使って、より詳しい情報にアクセス可能。

また、外部ツールに直接アクセスする為のインターフェースも持っています。

Matrix View – Element Selection States

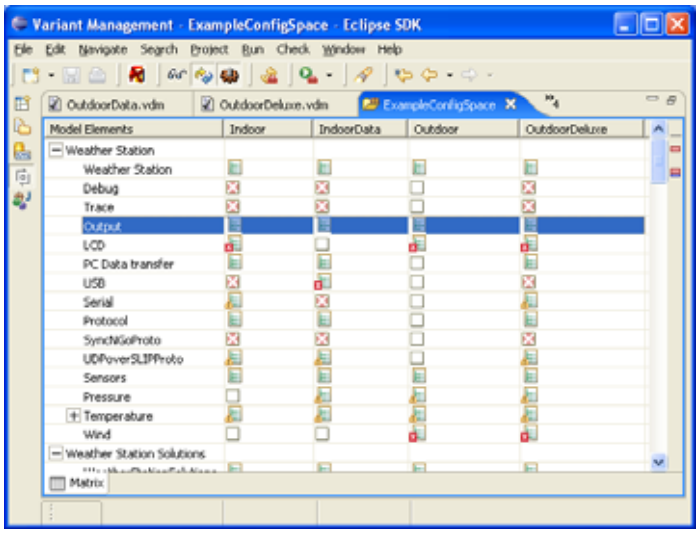
- マトリクスビューで複数バリアントの簡単な外観を表示。ここでは選択内容を表示（デフォルトの表示）
- 各々の列はバリアントを表し、行は選択されたエレメントを表す。
- バリアント特有の不具合とテストステートを表示可能



複数のバリアントをマトリクス表示する事で、プロダクトラインの全体像を捉えることも出来ます。

Matrix View – Element Test States

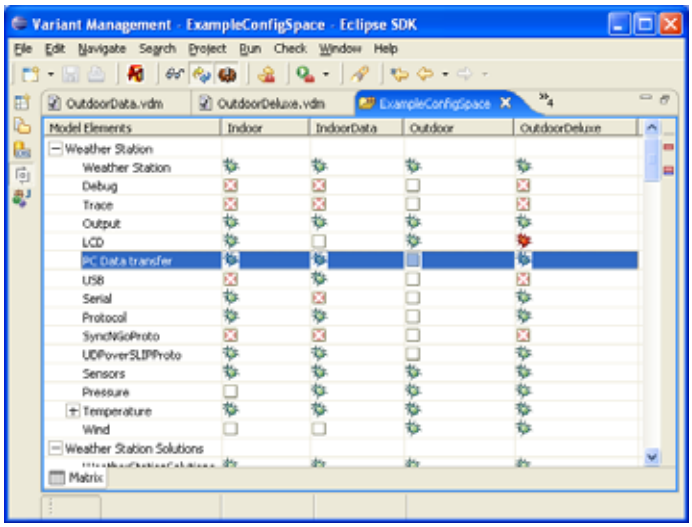
- テストステート表示では、全ての選択されたフィーチャーに対して、関連するテストが成功したことを表示（複数のアイコンで表示）
- ここでは、全てのバリアントが同じテストのセットを持つ事が分かる



テストステートをマトリクス表示させることも出来ます。

Matrix View – Element Defect States

- デフェクトステート表示に、選択されたフィーチャーのオープンな不具合が表示される。（複数のアイコンで表示）
- 全てのバリアントが同じ不具合を持っているのではないことが解る。



欠陥をマトリクス表示させることも出来ます。

pure::variants – Extension Availability
利用可能なエクステンション

- リリース２．４でコネクターＡＰＩを追加
  - ＡＰＩはサードパーティーが利用することが可能
- 以下のコネクターを２．４で提供
  - pure::variants Connector for ClearQuest
  - pure::variants Connector for Bugzilla

pure::variants　と
モデルベースツール

ここから、pure::variantsとモデルベースツールの接続について紹介します。Simulinkとの接続の例を紹介しますが、この例は、pure::variantsが提供しているSimulink Connectorではなく、ある自動車メーカーの依頼で特別に作成した接続例です。

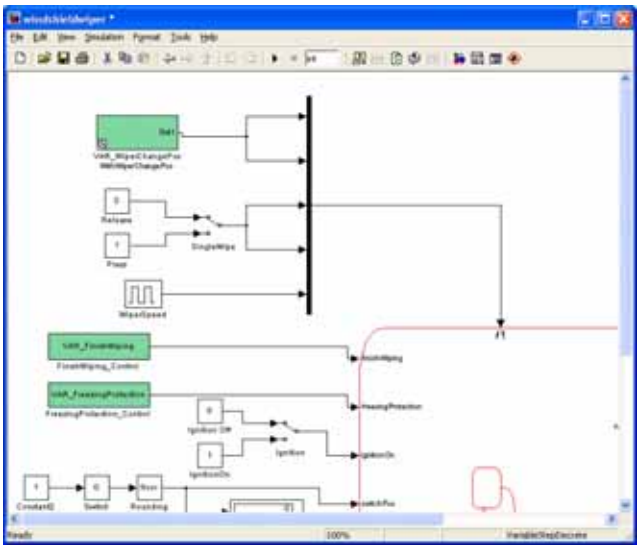
Connection Approaches
接続アプローチ

- フルモデルの読み込みと生成
  - 作成したモデルがpure::variantsモデルに完全に読み込める
  - pure::variants で、ルール等の追加や修正が行える
  - pure::variantsモデルからバリアントスペシフィックなツールのモデルが生成
  - Examples: **Connector for Simulink**
- リンクと通信
  - バリエーションポイントに関連した情報だけを外部ソースから取り出せる
    - 或いはほとんどの情報が外部ツールに入力される
  - コンフィグレーション時にバリエーションポイントを適用する
  - ツールに関連したコンフィグレーションを行う時は、ツールと通信する（オンライン / オフライン）
  - Examples: **Connector for Doors, new Simulink Configurator**

© pure-systems GmbH 2007

モデルベースツールとの接続には、ここに紹介している2つのアプローチがあります。
この後に紹介するSimulink Configurator**は後者の例です。**

Simulink Model with Variation Points



© pure-systems GmbH 2007 all you need for product lines www.pure-systems.com

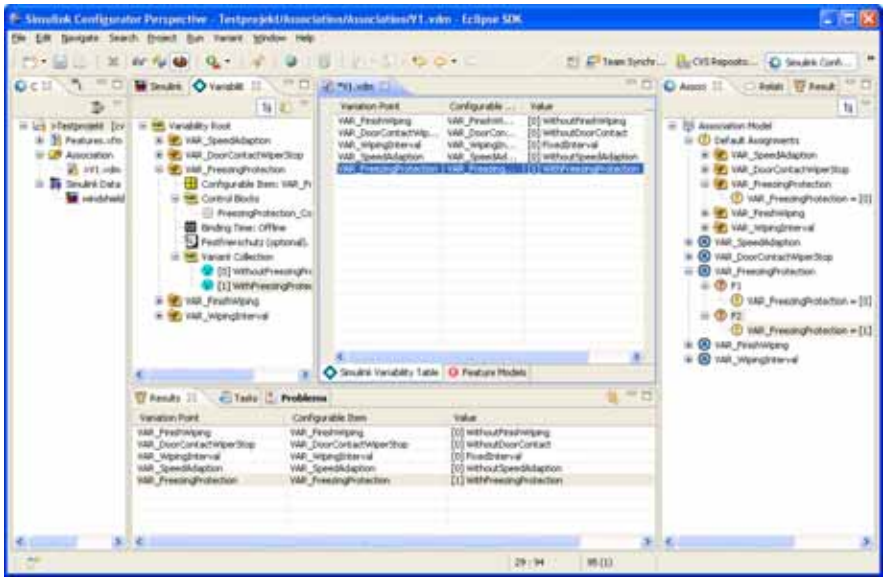
Simulinkのモデルです。緑色の四角がバリエーションポイントになります。これをpure::variants にリアルタイムにロードする仕組みです（MatLab Simulinkにオンラインでつなげて）。

Simulink Variation Point Setting

© pure-systems GmbH 2007

このモデルでは、変数FreezingProtectionが0にセットされています。

pure::variants based Simulink Configurator



© pure-systems GmbH 2007

先程のFreezingProtection**を含めた、**Simulink**モデルのバリエーションポイントが**pure::variants**に読み込まれています。**

FreezingProtection**を**pure::variants**上で**1**に変更すると**

Simulink Variation Point Setting

© pure-systems GmbH 2007

実際のSimulinkモデルでも1に変わります。

プロダクトライン開発支援ツール

MetaCase ＊MetaEdit+

DSM　**専用モデル環境構築ツール**
高い抽象度のモデルで、完全にコードを自動生成

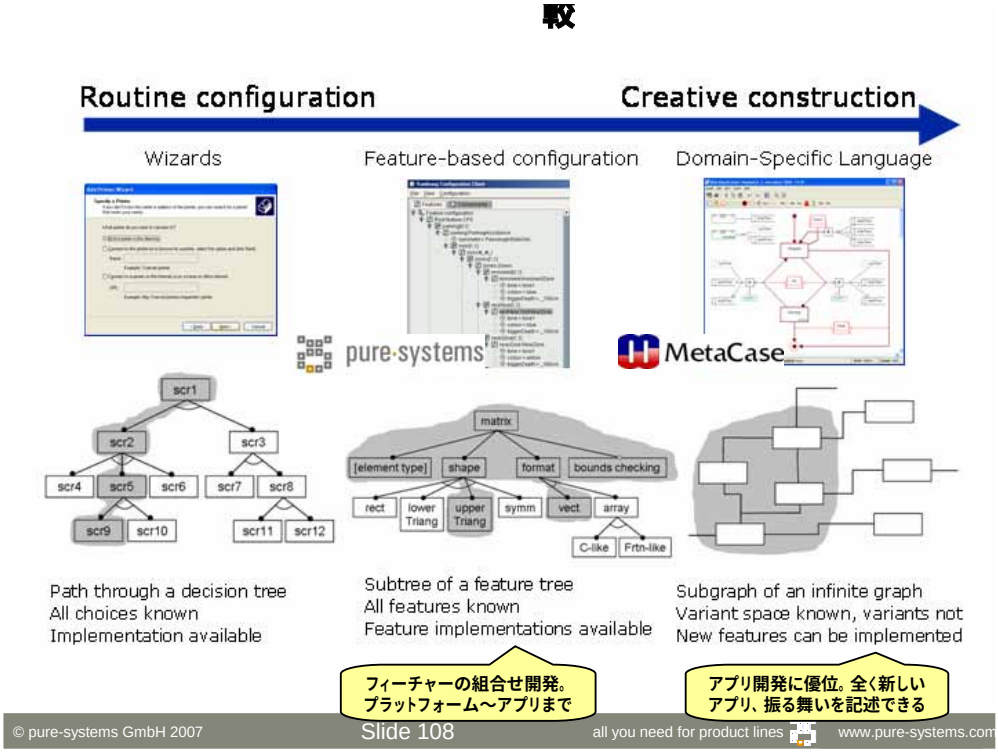
pure-systems ＊pure::variants

バライアビリティ・バリアント管理ツール
DOORS、CVS、ClearCase、Simulink、SAPなど
各種ツールと統合

© pure-systems GmbH 2007

2種類のプロダクトライン開発支援ツールを紹介します。
ここまで紹介して来たpure::variants**と、**MetaEdit+**という、ドメイン・スペシフィック・モデリングツールです。**

較

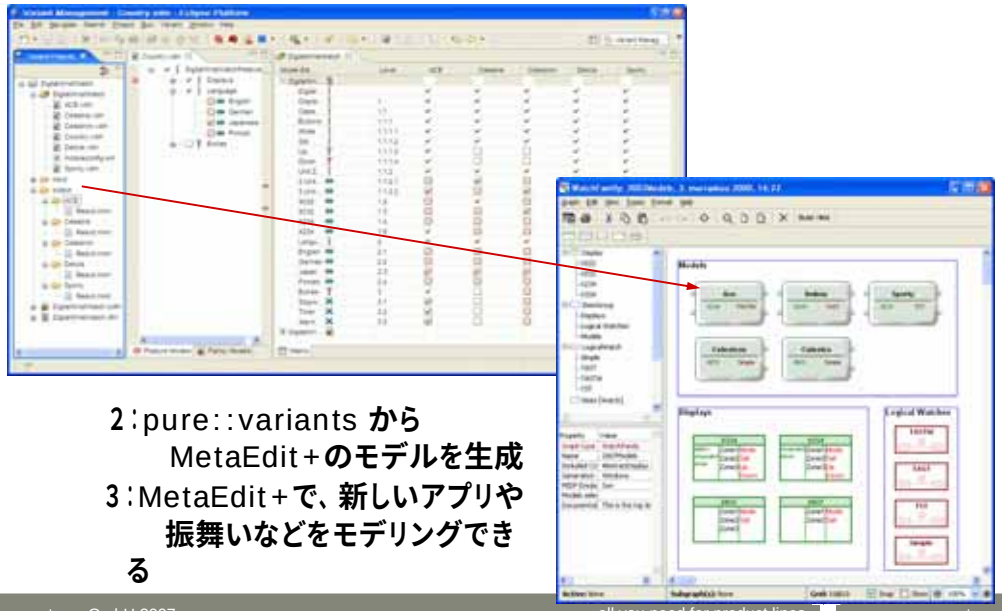


共に、再利用資産を活用し、モデルからコードの自動生成までをサポートしていますが、この二つのツールの違いは、

pure::variants**が、既存フィーチャーの管理を得意とし、それらの組み合わせ開発ができること。**
MetaEdit+ **は、新規のアプリ開発や、振舞も記述できることです。**

pure::variants と MetaEdit+ の統合

1：pure::variants にMetaEdit+のモデルをインポートし、管理



2：pure::variants から
MetaEdit+のモデルを生成
3：MetaEdit+で、新しいアプリや
振舞いなどをモデリングできる

© pure-systems GmbH 2007

そのため、pure:variants で既存資産を管理させて、MetaEdit+ を用いて新しいアプリや振舞いを開発することができるようになります。

価格情報

プロフェッショナル版(Floating User – 108**万円**)

- plugin for Eclipse IDE
- local model management in users file system
- uses Eclipse integrated or external version management (e.g. CVS or SVN)

エンタープライズ版 (Floating User – 135**万円**)

- client plugin for Eclipse IDE
- centralized model management on server host
- real-time collaboration
- integrated version management
- data storage file based or SQL DB

評価･検討をご希望いただける方の為に、1ヶ月間の無償デモライセンスを用意しています。

pure::variants**の価格情報です。**
CVS = Concurrent Versions System
SVN = Subversion

Extensions

Synchronizer (9万円)

- DOORS
- CaliberRM

Connectors (9万円)

- Connector for MATLAB / Simulink
- Connector for Source Code Management
- Connector for Version Control Systems
- Connector for SAP
- Connector for ClearQuest
- Connector for ClearCase
- Connector for SAP
- Connector for BIRT

外部ツールと接続する為の、エクステンション・オプションの価格です。

BIRTによる業務帳票の出力

EclipseのBIRTプラグインを介して

フィーチャーモデルからドキュメントを

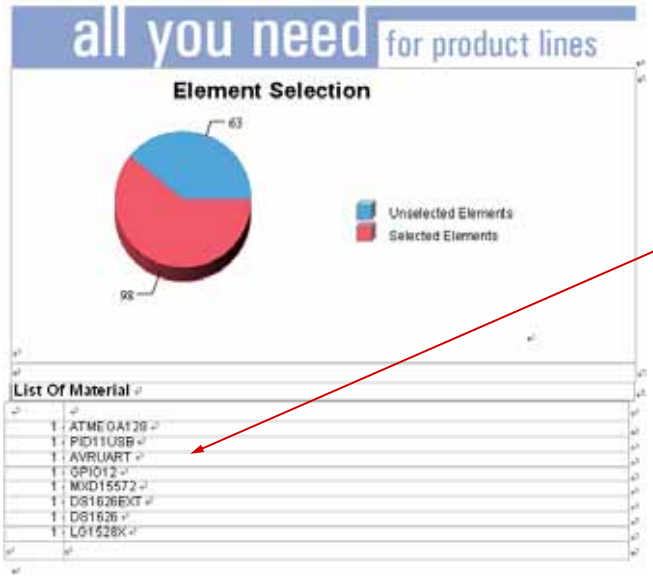
次期バージョンから

モデル内に、グラフィックを登録し出力できるようになる

© pure-systems GmbH 2007

pure::variantsから業務帳票を出力している例です。

BIRTによる業務帳票の出力



ファミリーモデルに

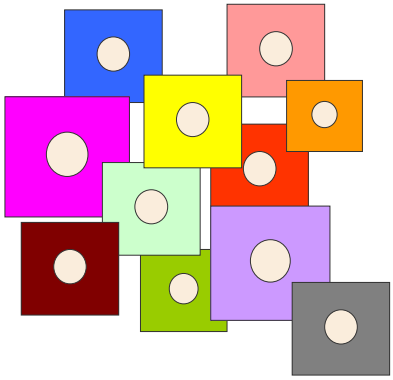
ハードウェア情報を登録し、部品表を作成できる


pure::variants

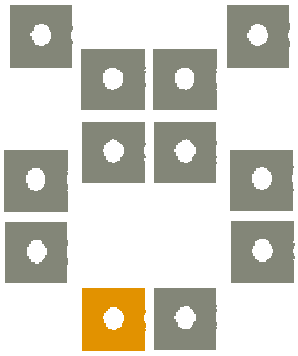
Variant Management
バリアントの管理

Many different variants and product lines

非常に多くのバリアントとプロダクトライン



pure·systems



are handled systematically

体系的に扱える！

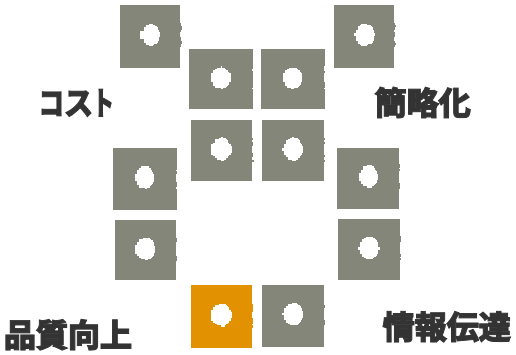
The Idea アイデア

- 重複を回避 :
  - ソフトウエアコンポーネント再利用の促進
  - HWとSWの選択と組合せをサポート
  - 自動的に組み立てる
- 基本ブロックの構成によりテーラメイドの製品を

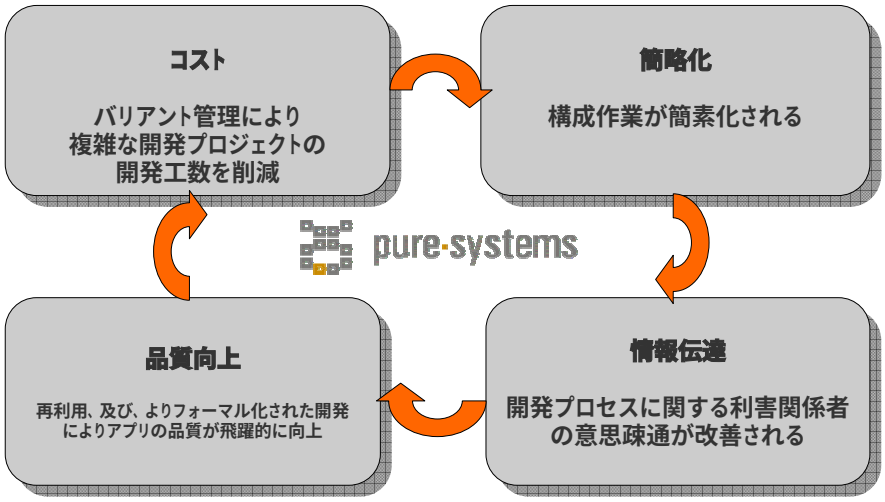
pure::variants による革新

- プロブレムとソリューションの両モデルを完全に分離。再利用、構成を促進。
- 自動的に機能間の矛盾、不一致を検出。
  バリアントモデリング時に実施し、早期段階に問題を排除できる
  - 最新鋭の技術で論理的ルールを解釈
- 低い導入障壁:
  - 既存資産、ファイルシステム構造を変えることなく導入できる
  - 製品バリアントは 制約無しに自在に生成できる。複数のバリアントを同時に生成できる。(ファイル構造などの仕組みに依存することなく、完全に独立した部品としてコア資産を管理し、利用できる)
- エクリプス、クライアント/サーバアーキテクチャをベースに、柔軟に拡張できる
  - ローカル、あるいはサーバ管理してマルチユーザでモデルを管理できるので、大規模システムにもフィットする
  - 多くのエクステンションにより、既存開発環境への統合や改善が行える
- これらのコンビネーションは、まさに独創的

Drivers for Variant Management
バリアント管理をする理由



Where helps Variant Management？
バリアントマネジメントの効果は何処に？



Benefits of Variant Management
バリアントマネジメントにより得られる効果

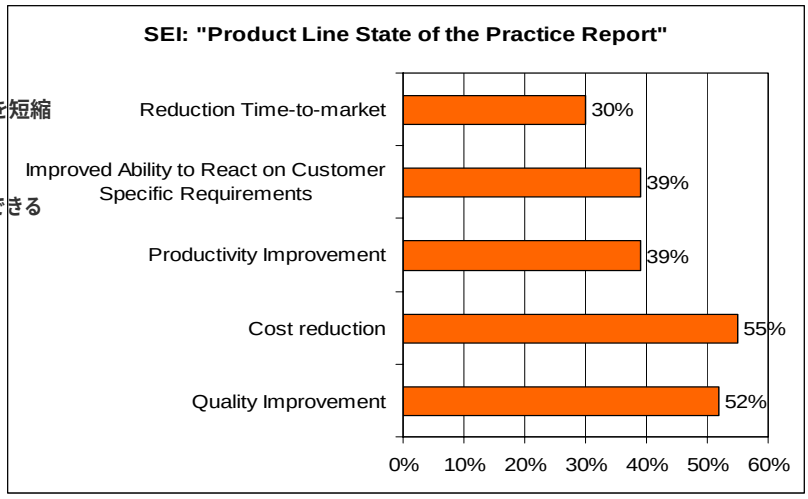
製品化までの時間を短縮

顧客特有の要求に対して迅速に対処できる

生産性の向上

コスト削減

品質向上



Benefits of Variant Management
バリアントマネジメントにより得られる効果

SEI: „Software Product Lines State of the Practice Report“

Benefits for a DoD Project　Department of Defense、《米》国防総省

Quality	1/10 of normal level of errors
Performance	comparable to conventionally developed systems
Development Time	integration in weeks insted of months, only 15 instead of 100 developers
System Size	76% lesser than planned
Productivity	50% less cost 50% shorter project duration than planned

品質： 通常レベルのエラーは1/10 に
性能： 従来開発製品と匹敵
開発工数： 統合は数ヶ月から数週間へ。100人の開発者は、15人に。
システムのサイズ： 予定サイズの76%
生産性： 50%コストダウン。予定の50% (半分) の期間

pure::variantsの特徴

pure::variants により得られる効果

- コスト削減
  - 開発期間の削減
  - 製品バリアントのサポート＆メンテナンスが簡素化される
- 品質向上
  - 証明されたソリューションを再利用できる
- 新規マーケットへ
  - 更なるプロジェクト･製品が従来と同じ労力･工数で得ることができる
- 初期導入コストを低く抑える
  - 既存のツールチェインへの統合

© pure-systems GmbH 2007

More Information

- Telephone: +49 391 5445 69 -0

  the fast path for all questions – we look forward to your call
- Internet: http://www.pure-systems.com

  here you will find detailed information on pure::variants and the pure-systems GmbH
- e-mail: info@pure-systems.com

- 富士設備工業㈱電子機器事業部
  http://www.fuji-setsu.co.jp

© pure-systems GmbH 2007