ソフトウエアプロダクトライン開発セミナー

先人の経験に学ぶバリアント/バライアビリティ管理のコツ

Dr. Danilo Beuche

pure·systems

予定

- ソフトウエアプロダクトライン開発について
- ソースコードのバライアビリティの扱い
- 要件管理（要件の再利用に関する課題）
- モデルベース開発（UML、Simulinkモデルなど）
- テストの管理
- 欠陥（バグ）・変更の管理

講師

- **Dr. Danilo Beuche**
  - 2001-: ドイツ、マグデブルグ　pure-systems社　代表取締役
    - 組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 ツールとコンサルティングの提供
  - 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

Dr.Danilo Product Line Engineering Blog (日本語訳)
http://www.fuji-setsu.co.jp/products/purevariants/Danilo_Blog.html

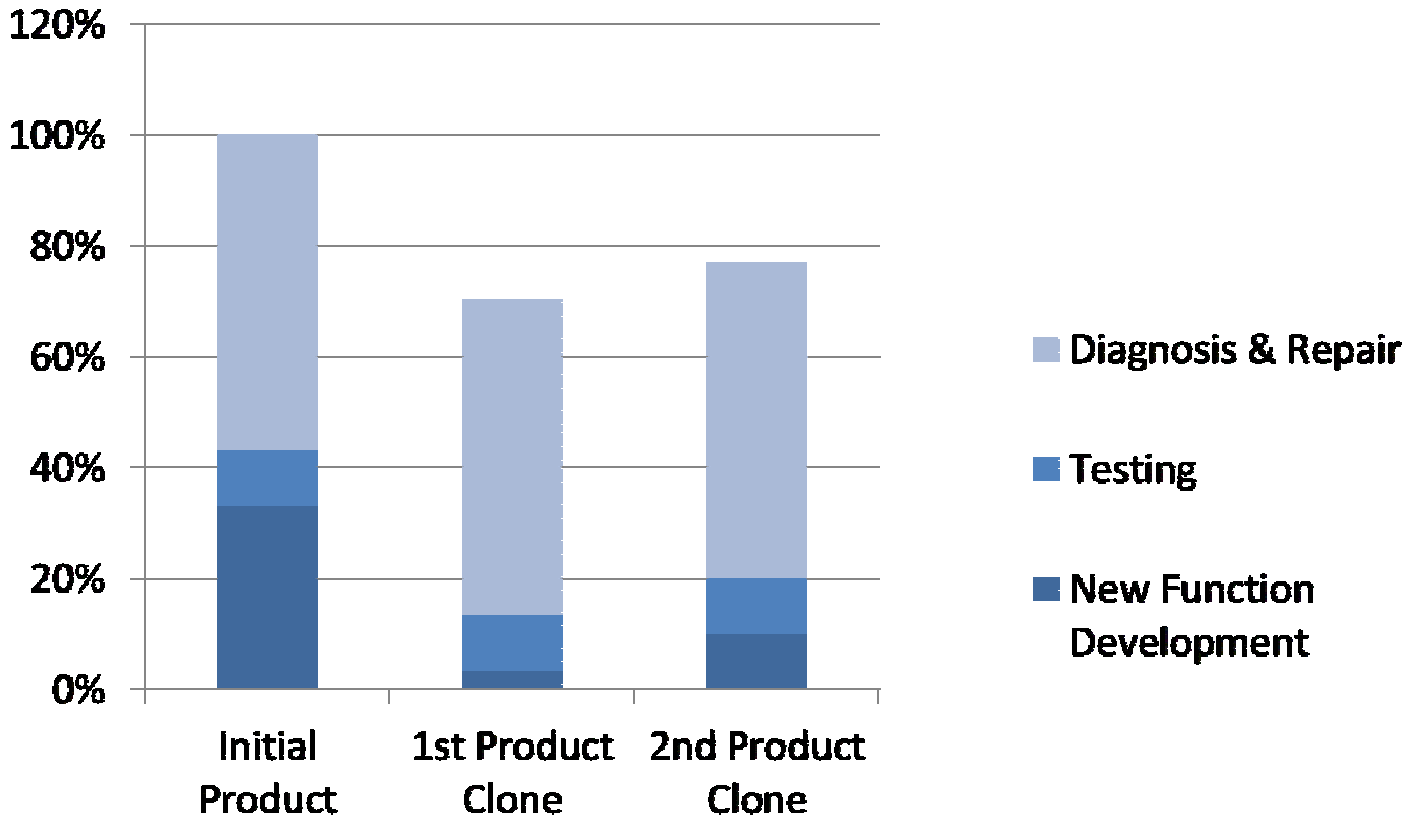
pure::variants

バリアント管理を支援するツール

pure-systems

組込み機器を中心に広く採用

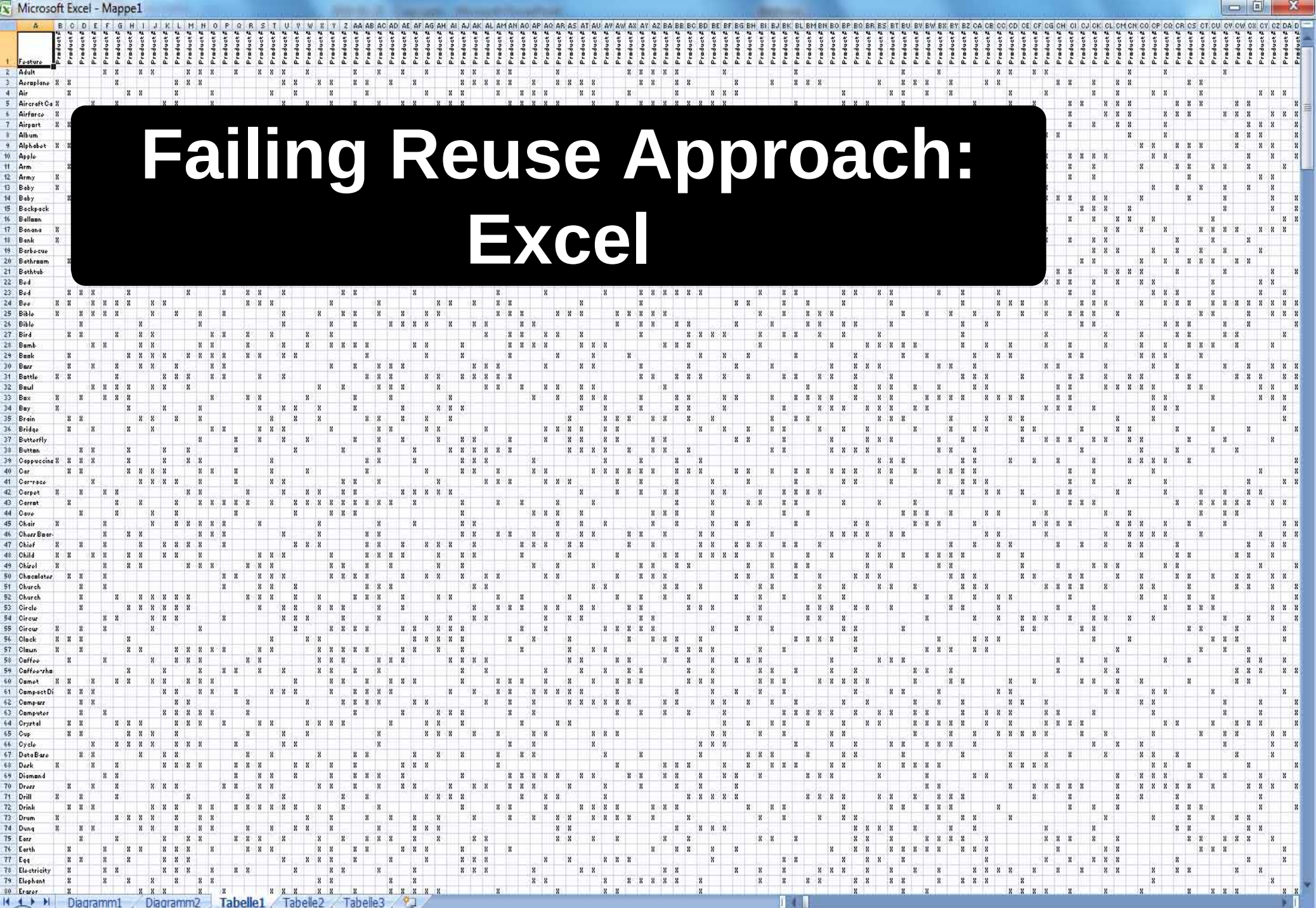
再利用開発の課題: コストが安くならない



10% **の新機能追加なのに** → ~80% **のコスト**

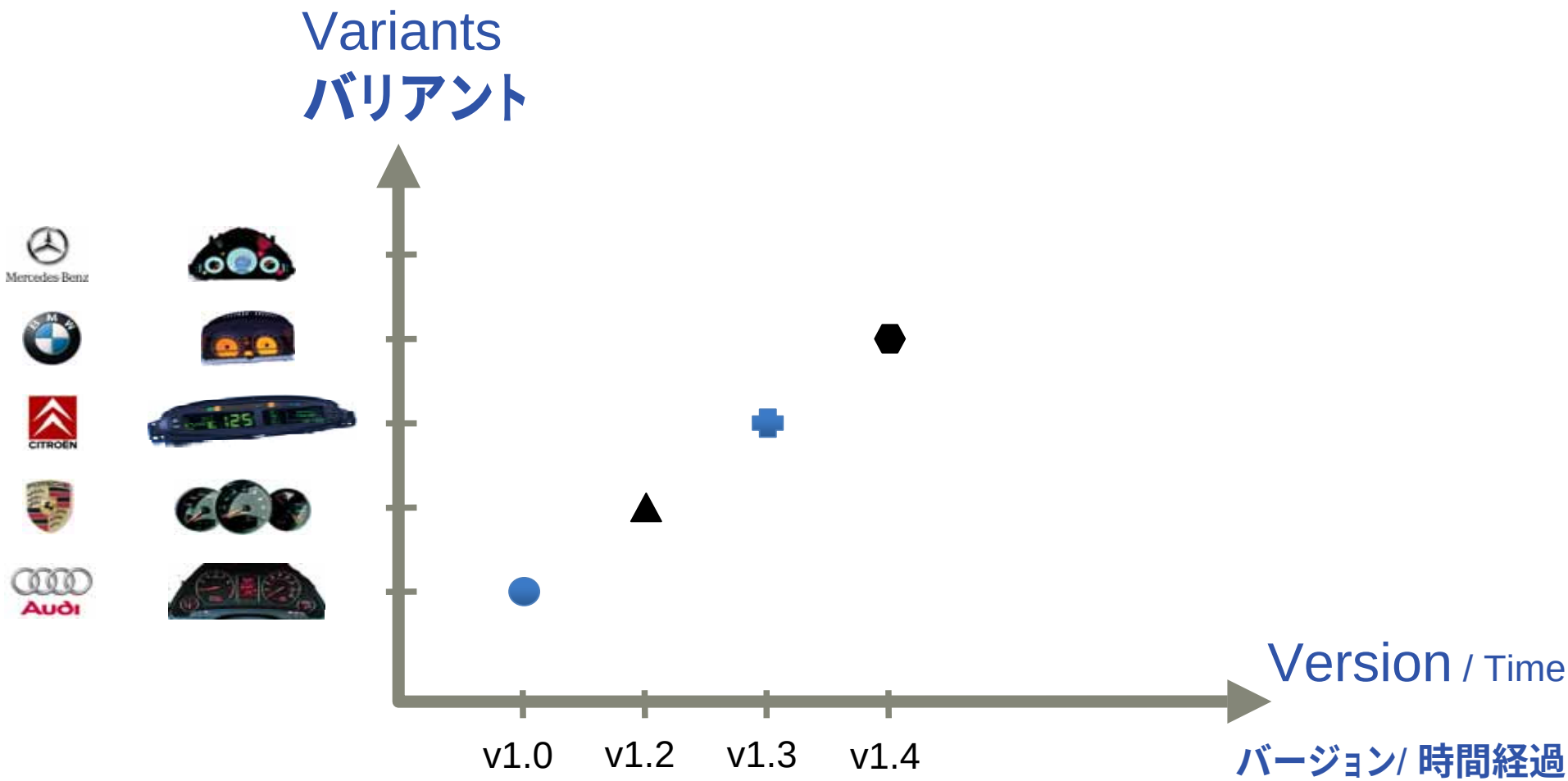
テスト・検証作業に多くの時間を要する

再利用のアプローチ:
既存ツールの組合わせ・調整だけでは、派生製品ファミリーの資産を上手く管理できない


Microsoft Excel - Mappe1
Failing Reuse Approach:
Excel
Diagramm1
Diagramm2
Tabelle1
Tabelle2
Tabelle3

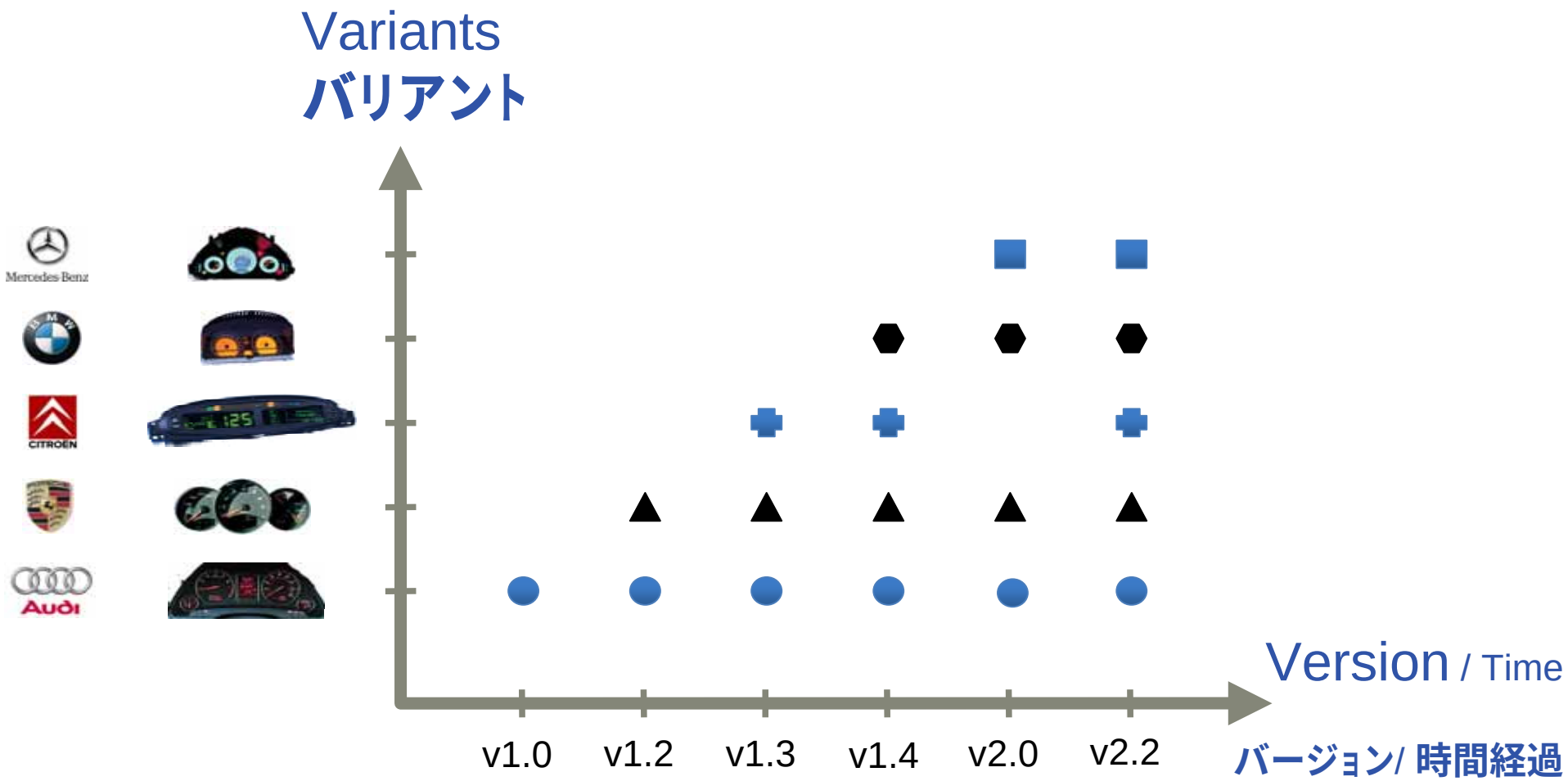

Failing Reuse Approach:
サイロ型 要件、設計、コード、テストなど
ツールごとで資産が別々に管理される

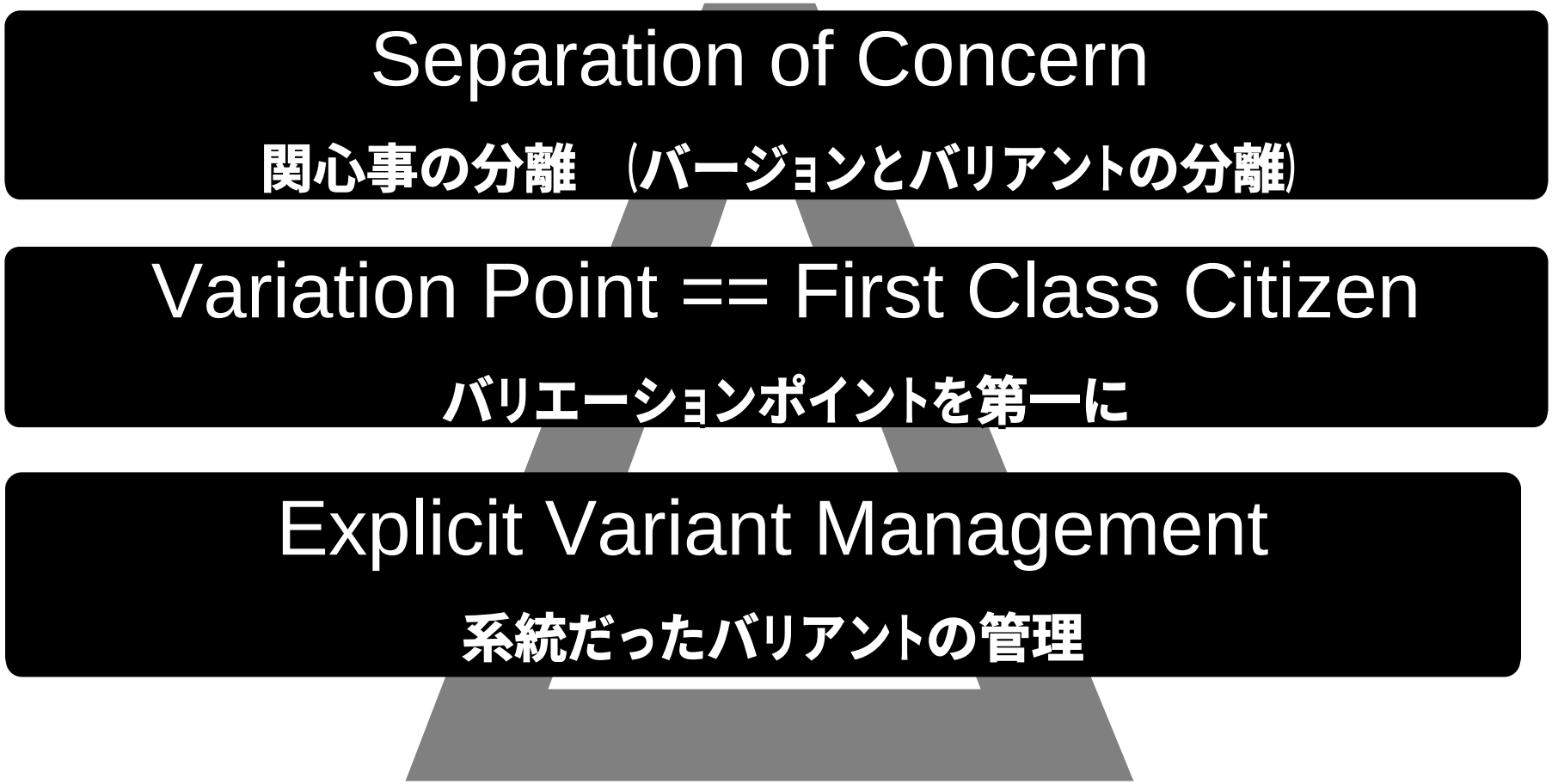
クローン & オウン の開発では



安くつくようでもテストやメンテナンスが軽減できない

Separation of Variants and Versions
バリアントとバージョンの分離

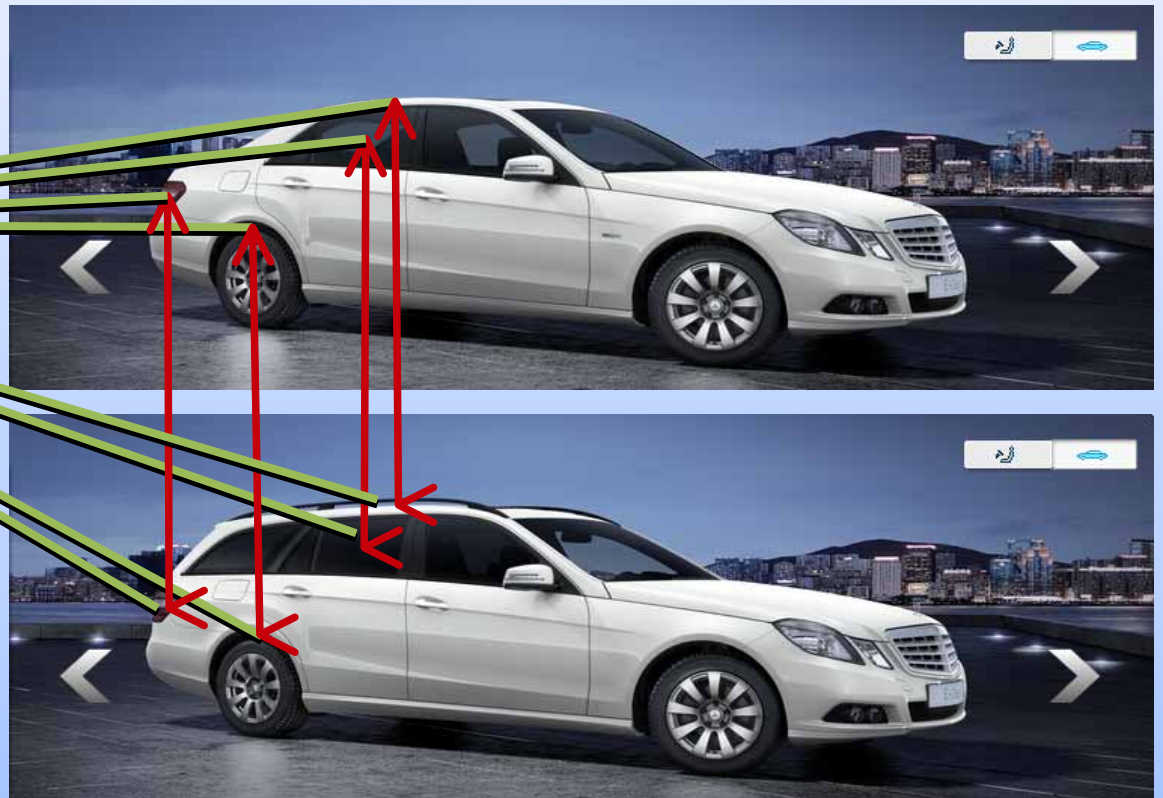




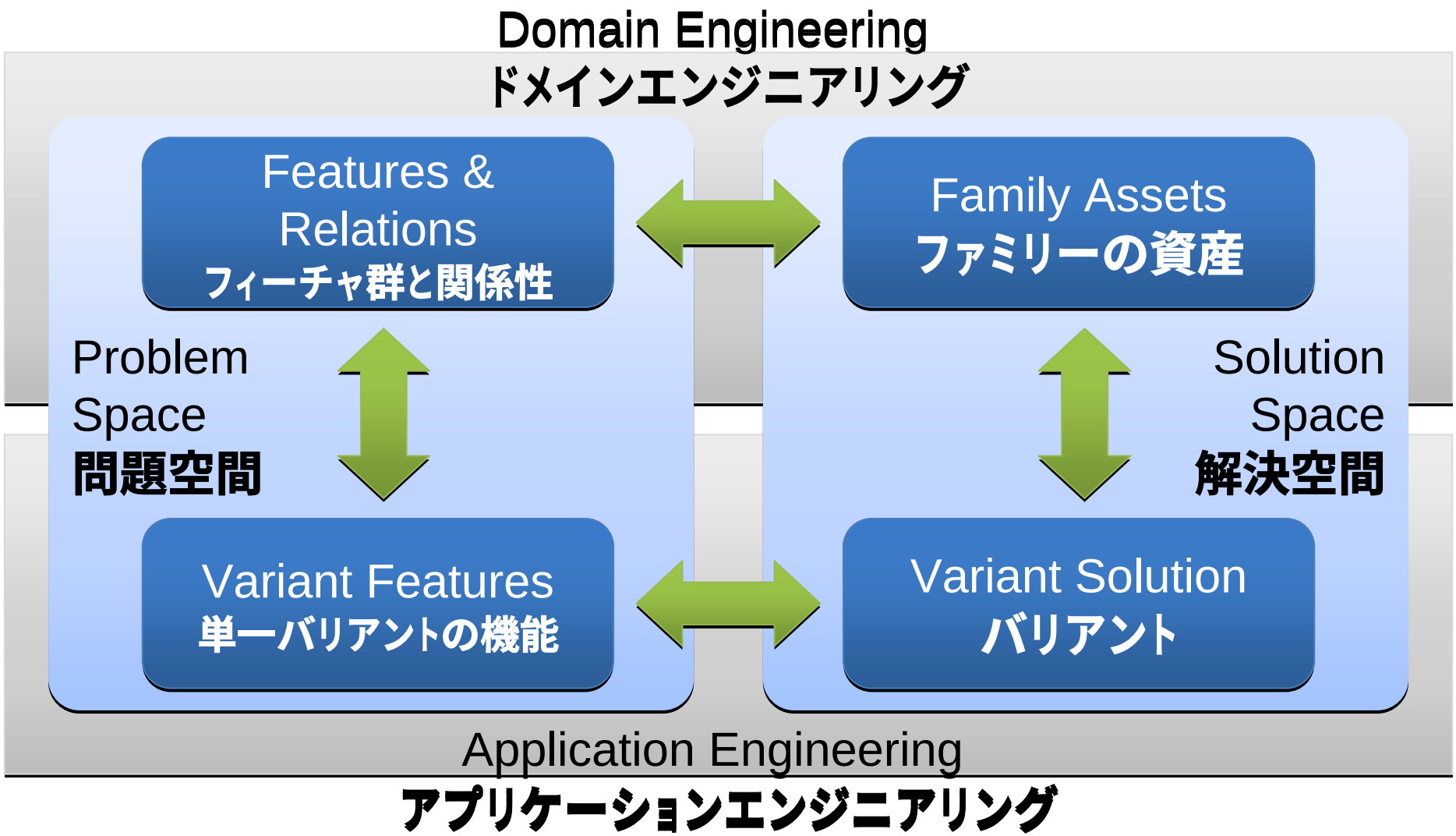
Variation Points
バリエーションポイント

Problem Space
問題空間

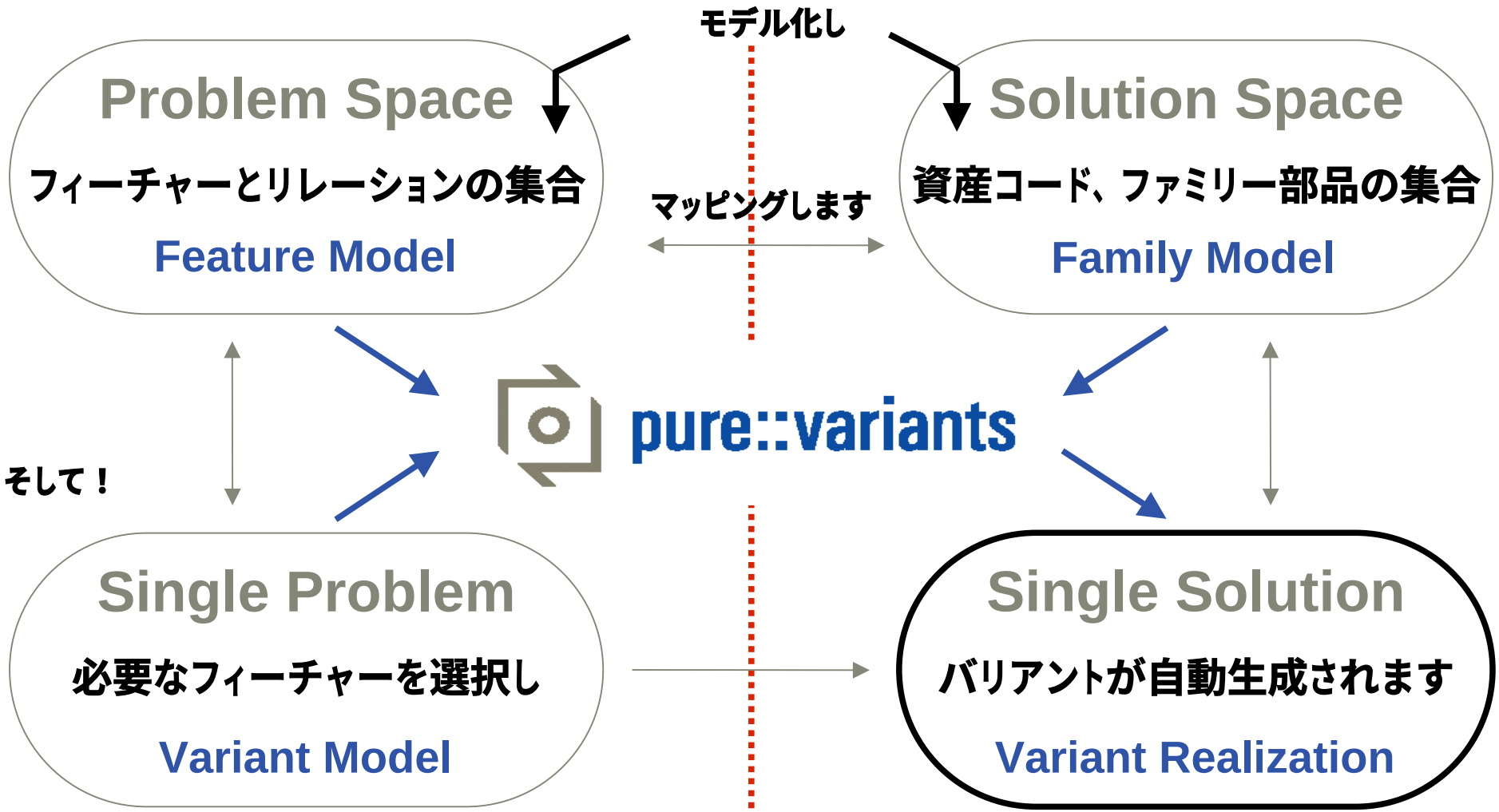
Solution Space 解決空間



Separation of Concern
関心事の分離



バリアント管理ツール



バリアント管理ツール

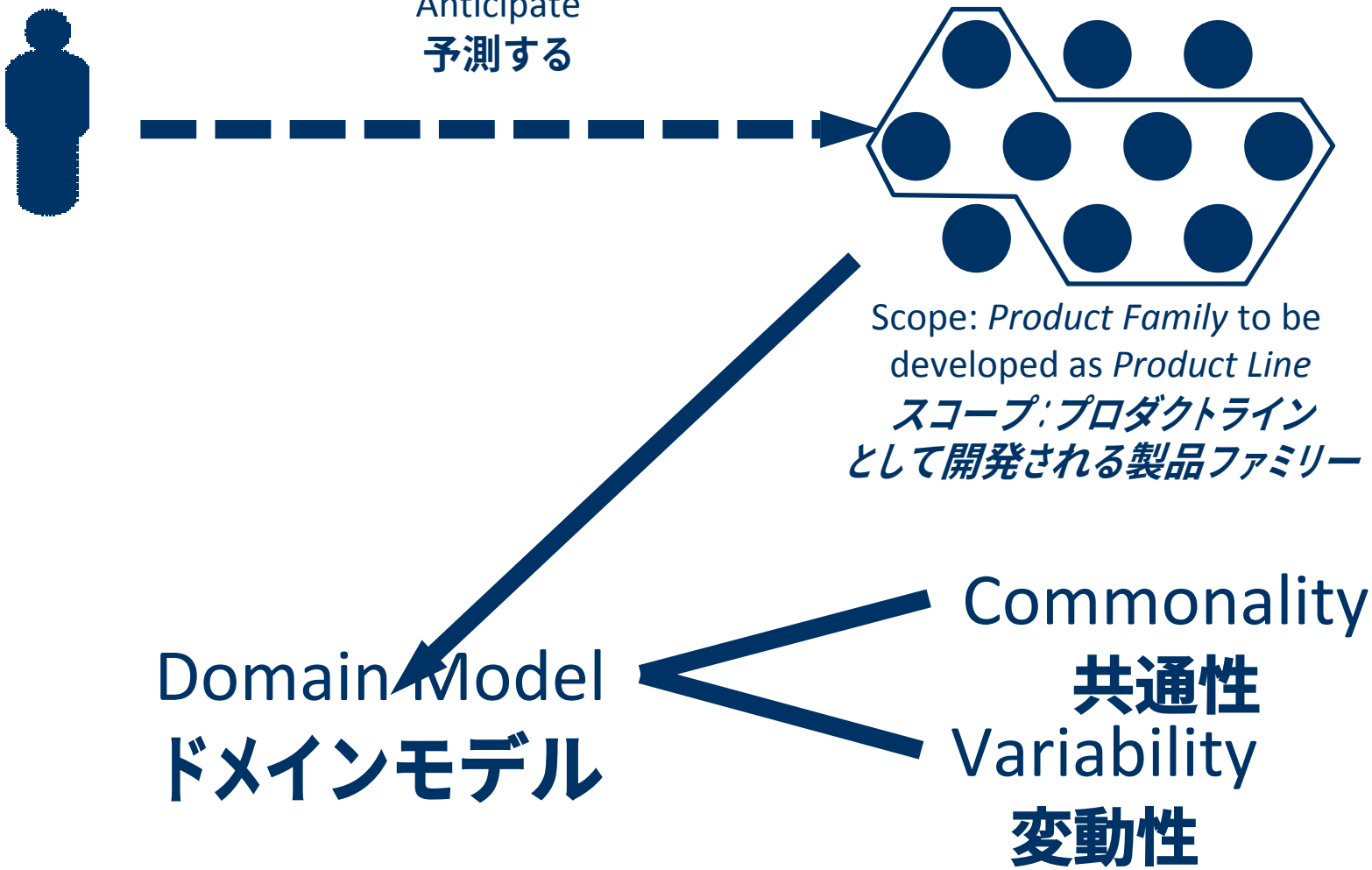
pure::variants

要件管理	デザインツール	構成管理	変更・バグ管理
MKS Requirements	Simulink	MKS Source	MKS Integrity
Borland Caliber RM	openArchitectureWare	Rational ClearCase	Rational ClearQuest
Telelogic DOORS	Rational SW. Architect *Rhapsody*	CVS	Atlassian JIRA
	Artisan Studio	Subversion	Bugzilla
	Enterprise Architect.	...	Softtest

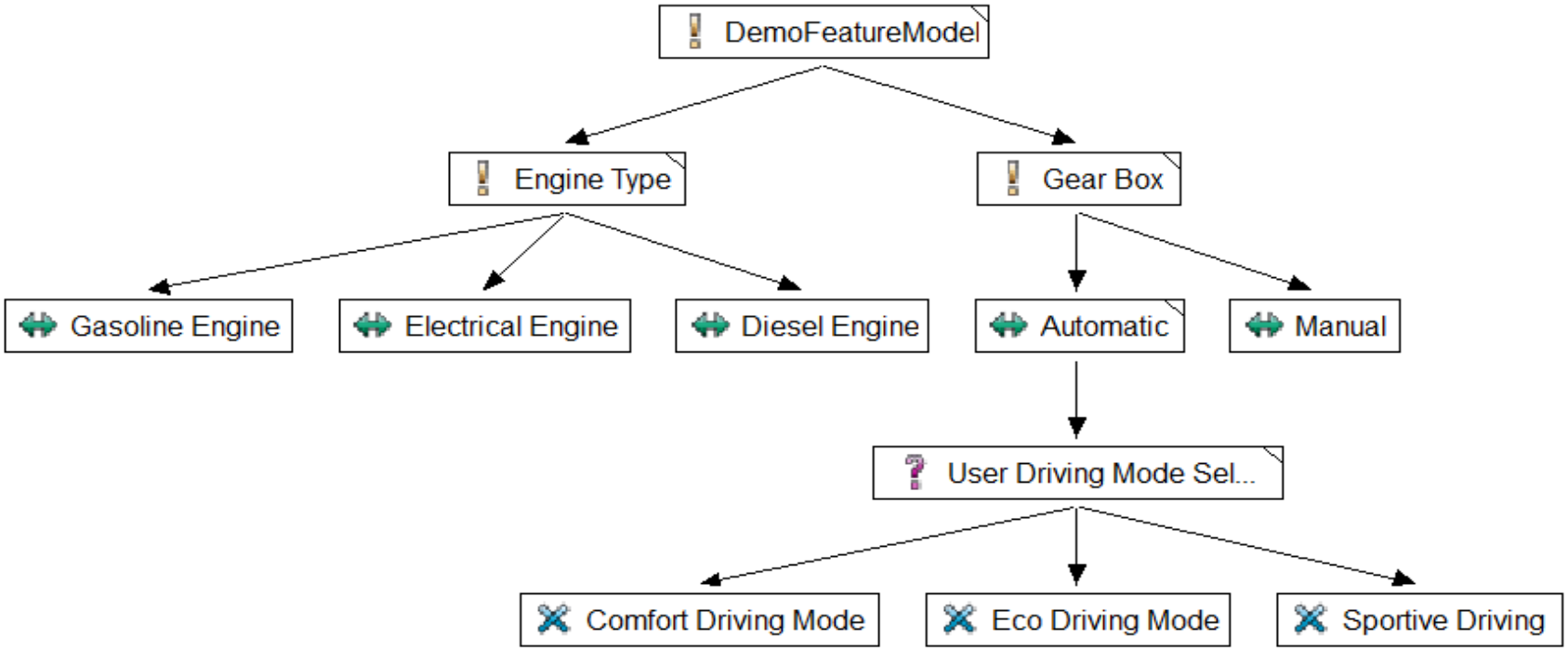
各種ツールを統合

プロダクトライン・ライフサイクル・マネジメント を支援

Problem Space: Define Domain Model
問題空間：ドメインのモデルを定義

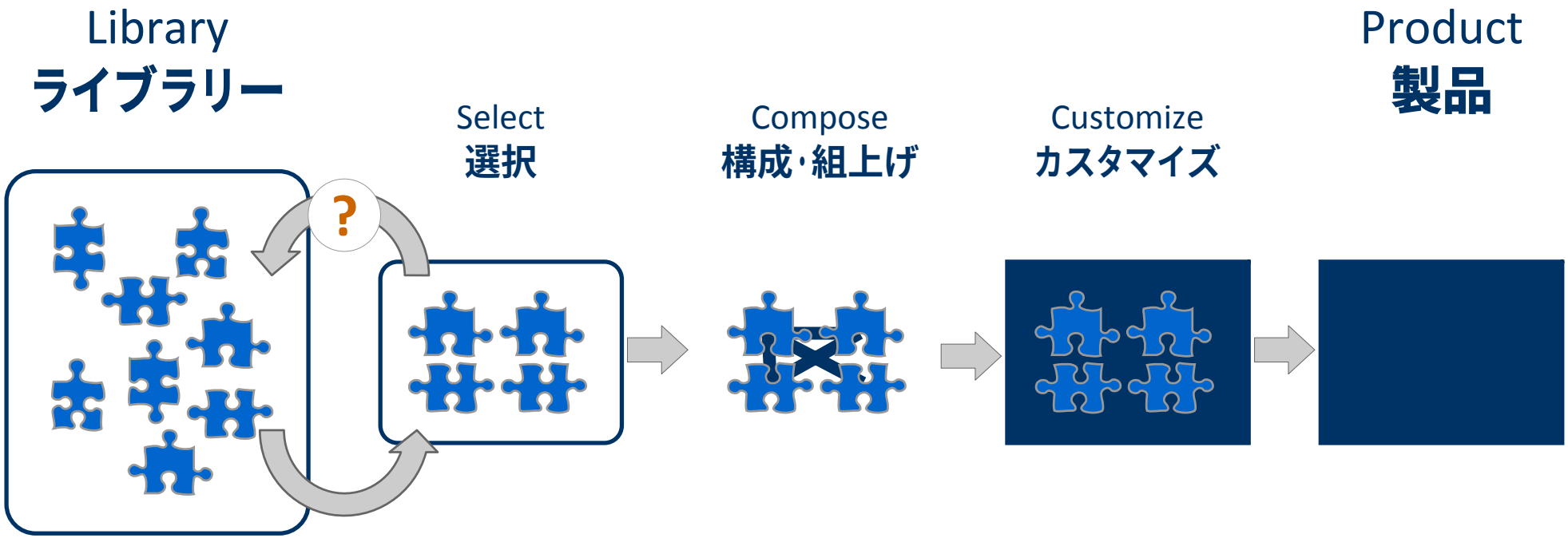


Describing Variability Using Feature Models
バライアビリティをフィーチャモデルで記述



Legend: = Mandatory : Optional Alternative Or

Reuse: The Old Way
再利用法：古いやり方　(ライブラリ)



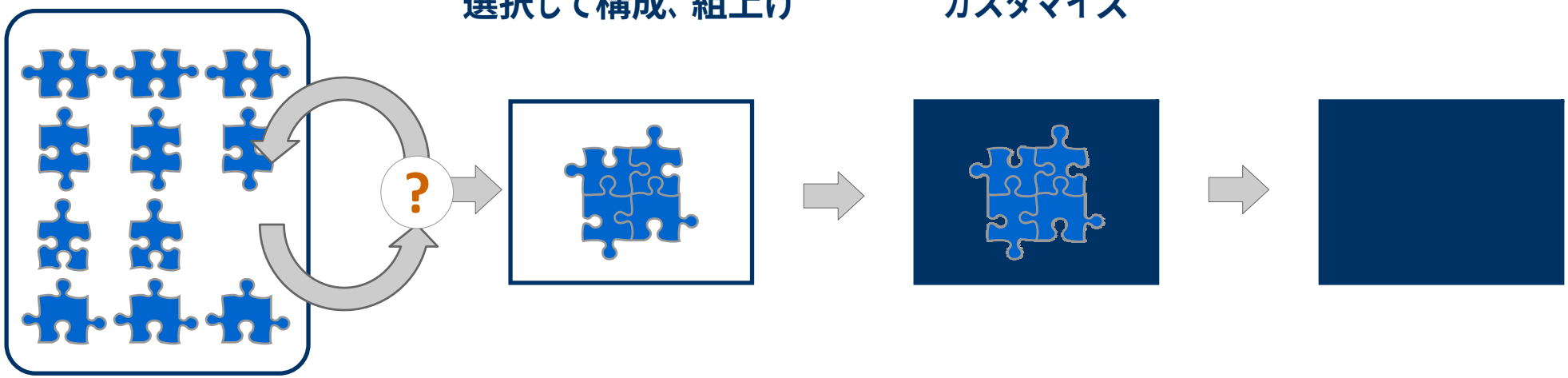
Reuse: Product Lines
再利用：プロダクトライン

Platform / Core Assets
プラットフォーム/コア資産

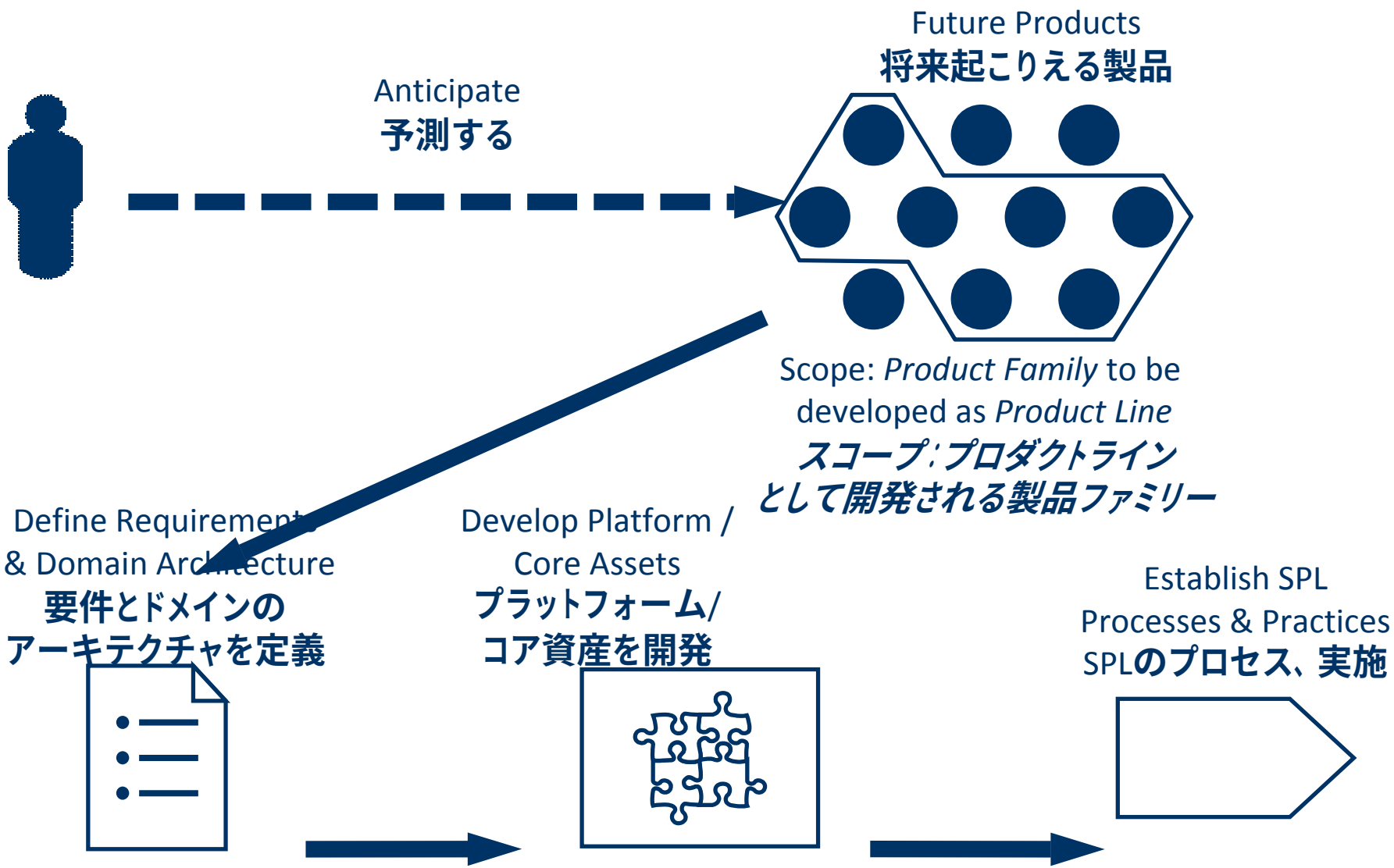
Select & Compose
選択して構成、組上げ

Customize
カスタマイズ

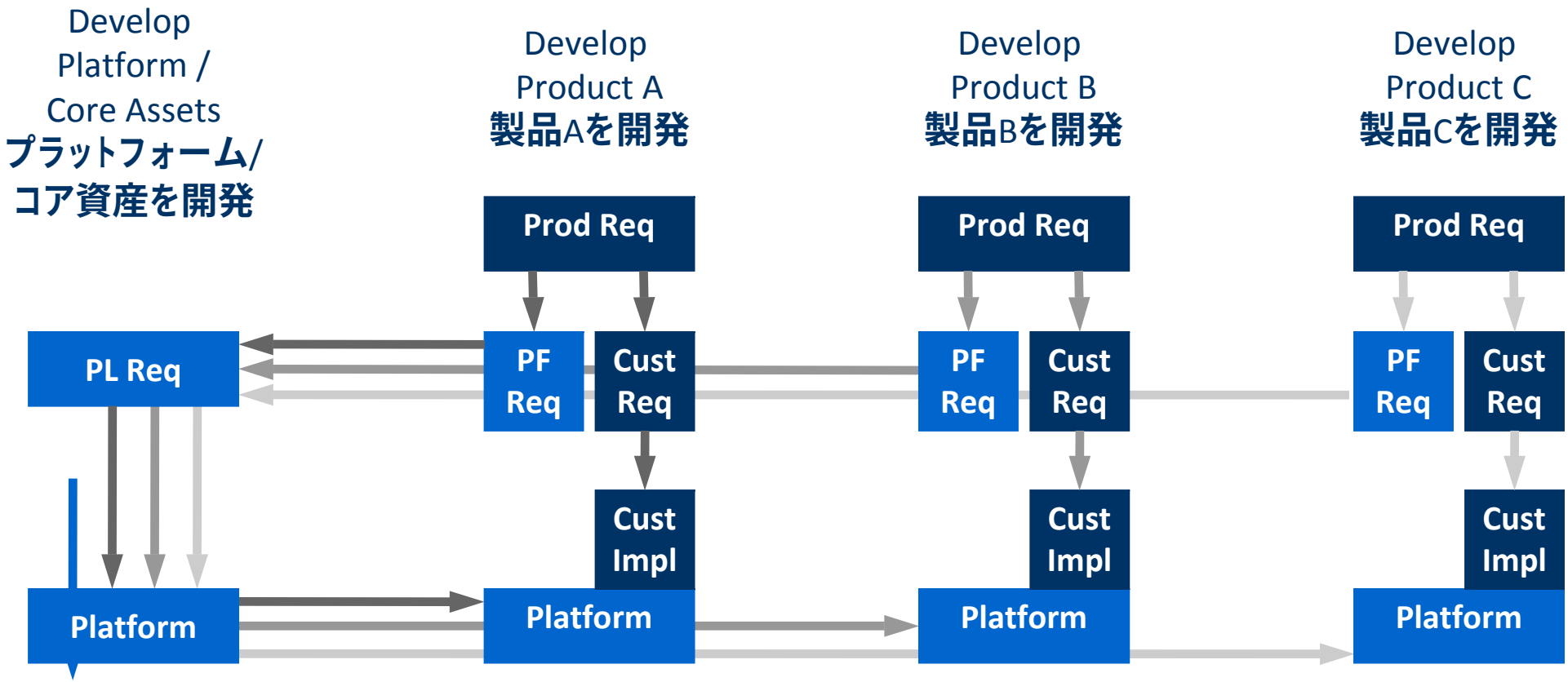
Product
製品



Setting up a Product Line
プロダクトラインの構築

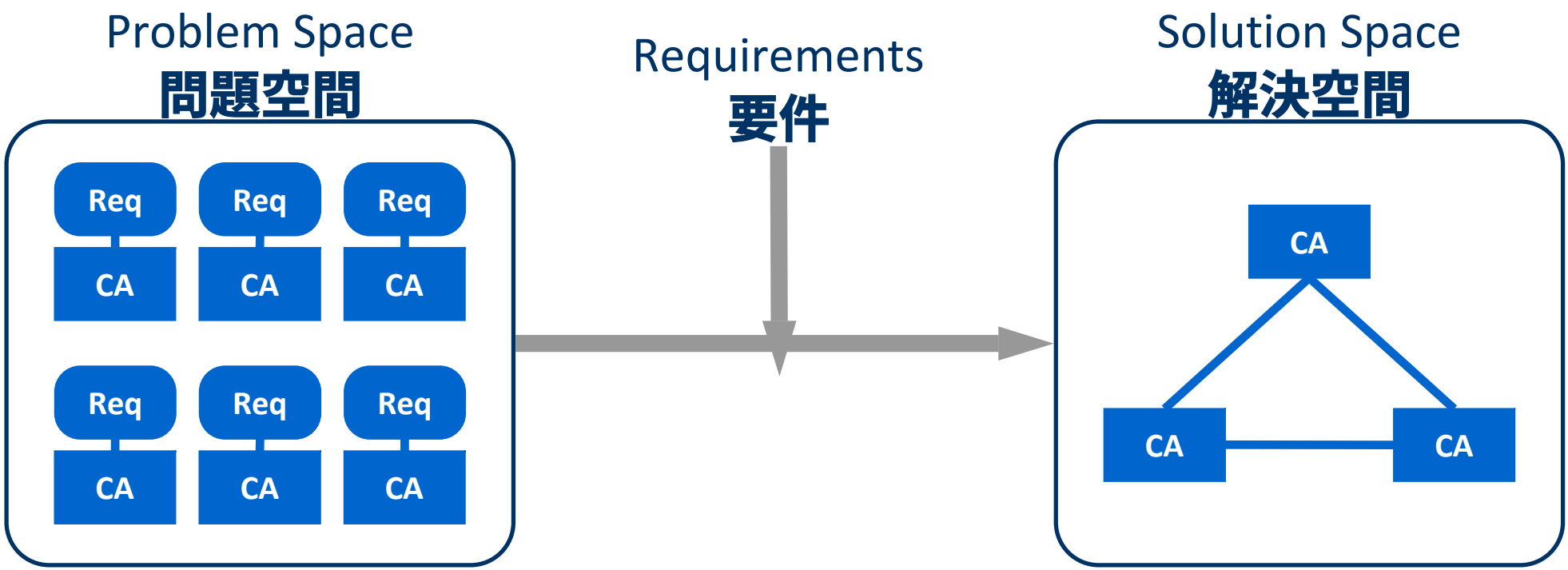


Operating a Product Line
(製品個別で再利用されないものもある)



Legend: PL = Product Line; PF = Platform; Prod = Product; Cust = Custom; Req = Requirement; Impl = Implementation

Development with Reuse: Functional Mapping
再利用による開発：機能のマッピング



f : (Req, (Req x Core Asset))

Reusable Solution
再利用可能なソリューション

Legend: Req = Requirement; CA = Core Asset

Reuse Items
再利用するアイテム　コードだけ？

Test Cases

Architecture

Estimates

Models

Code

Requirements

Subsystems

Documentation

Rules

Reuse Items
再利用するアイテム　全ての成果物！

Test Cases

Architecture

Estimates

Models

Code

Requirements

Subsystems

Documentation

Rules

Reuse Items
再利用するアイテム　要件について

Test Cases

Architecture

Estimates

Models

Code

Requirements

Subsystems

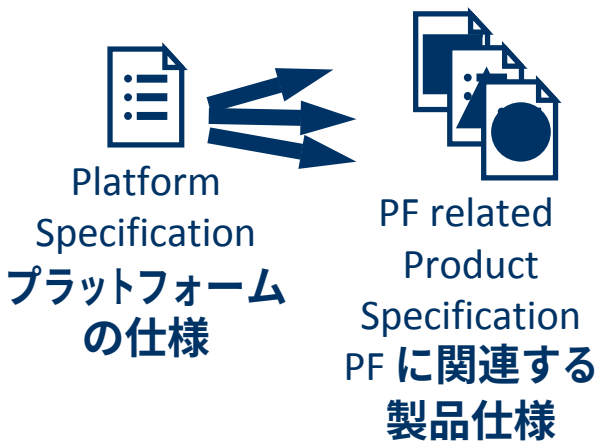
Documentation

Rules

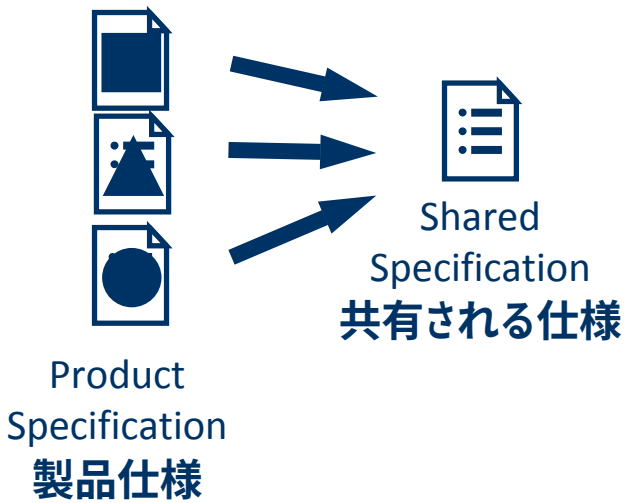
Scenarios (3)

Specification Approach
仕様のアプローチ

Derived 派生



Shared 共有



Individual 個別

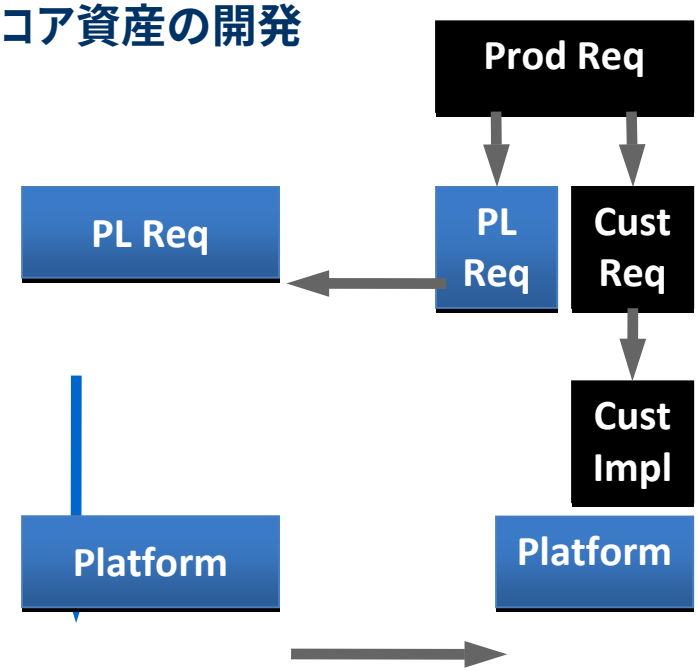
Exercise

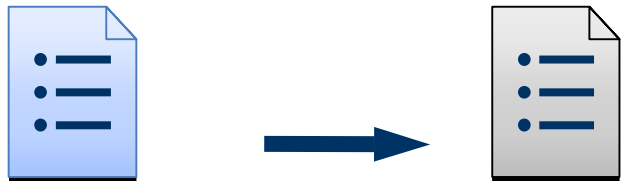
Describe your organization‘s approach!

Requirements Specification
要求仕様

Develop
Platform /
Core Assets
プラットフォーム/
コア資産の開発

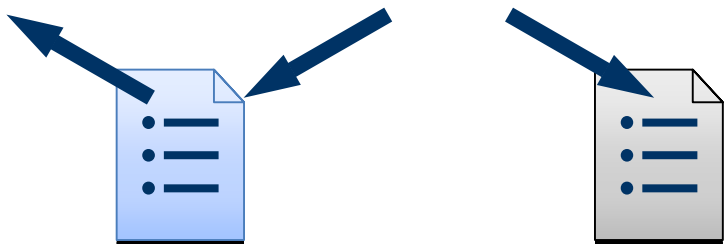
Develop
Product A
製品Aを開発





Platform
Specification
プラットフォームの仕様

Product
Specification
製品の仕様



Mapping to Platform
Specification &
Specification of Changes
プラットフォームの仕様
と変更の仕様

Custom Product
Specification
独自製品の仕様

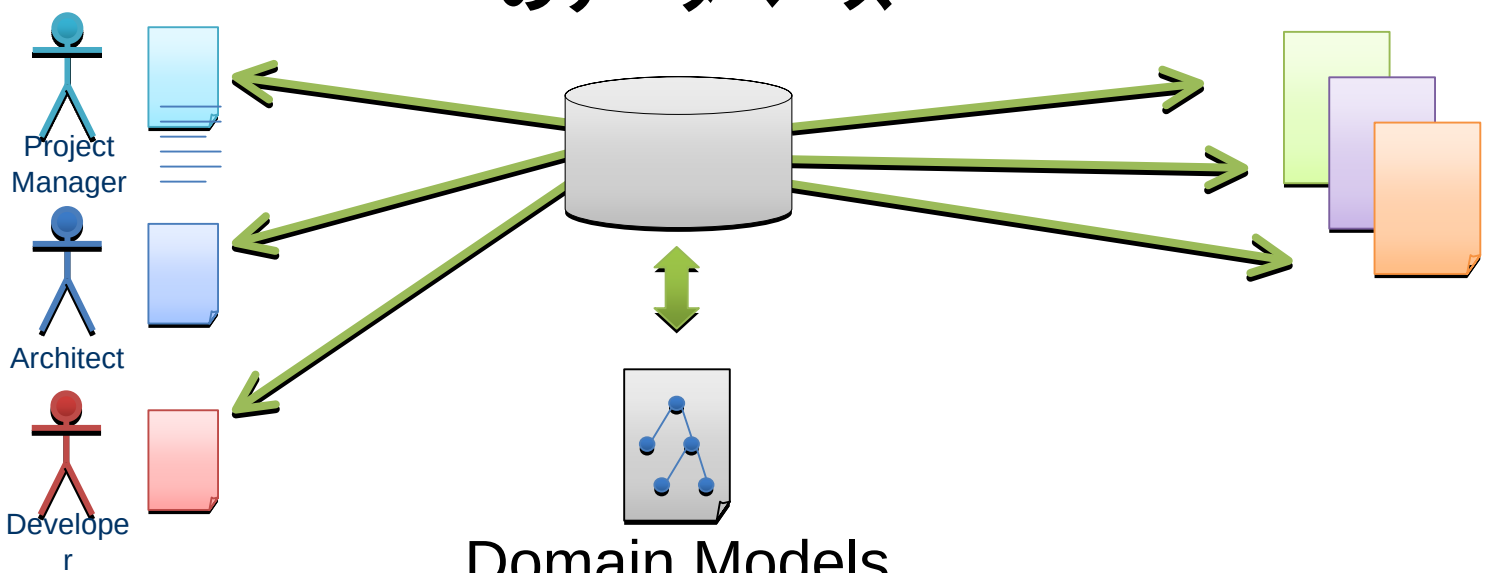
Legend: PL = Product Line; PF = Platform; Prod = Product; Cust = Custom;
Req = Requirement; Impl = Implementation

Continuous Requirements Management
継続的な要件管理

Stakeholder-Specific Views
利害関係者各自の観点

Central Requirements/ Feature Database
要件、フィーチャのデータベース

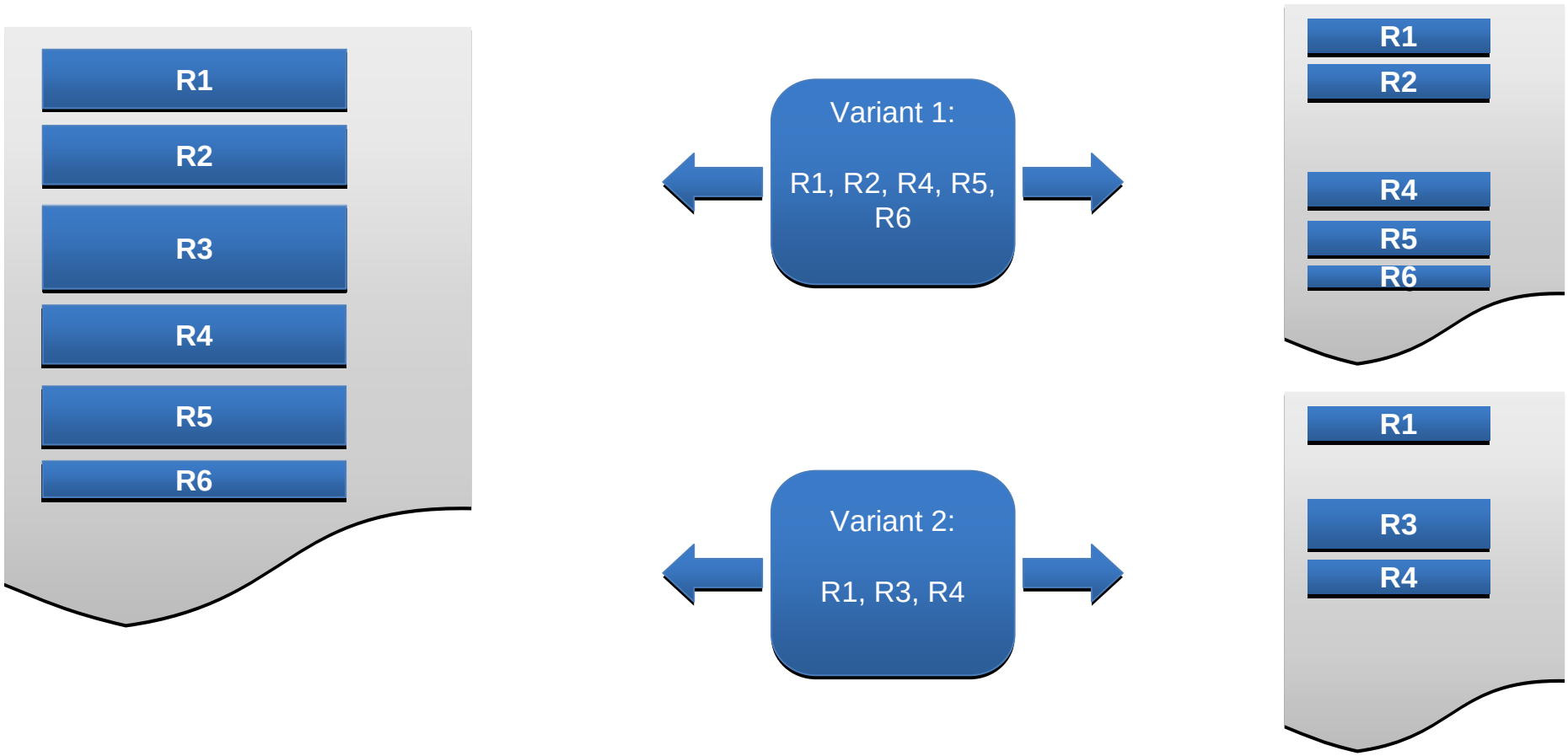
Supplementary Documents
補完的なドキュメント



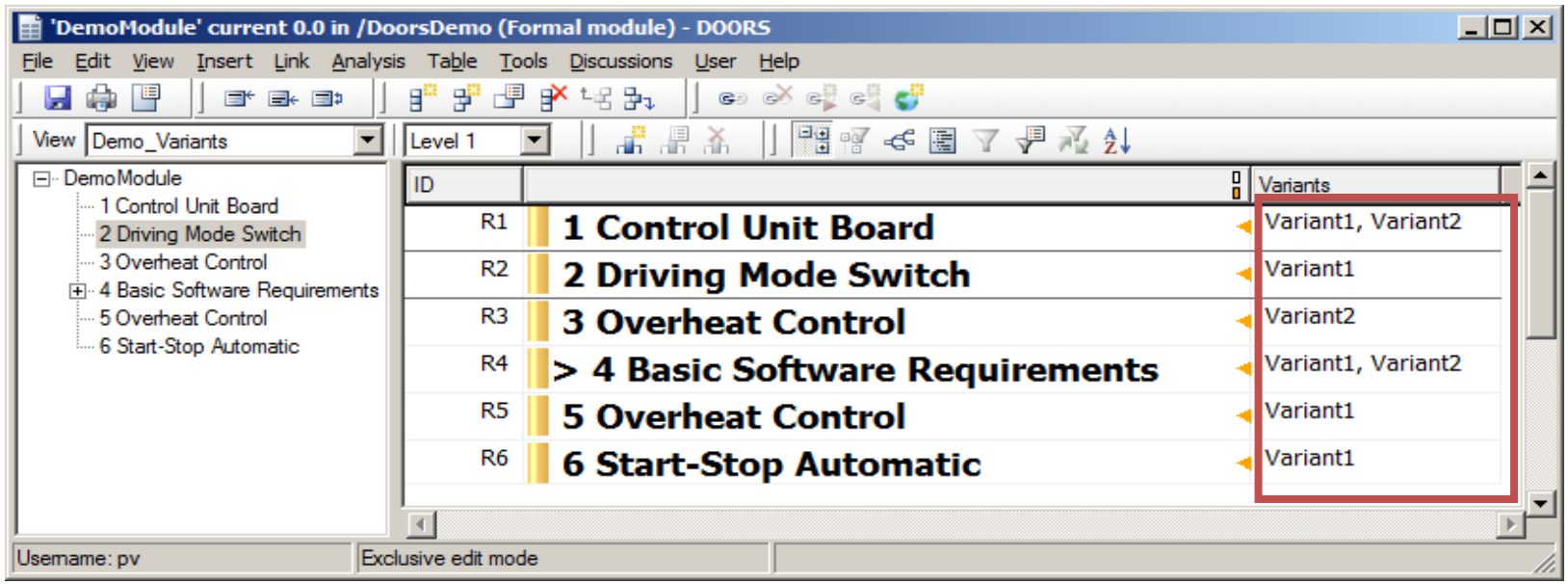
Domain Models
ドメインのモデル

Requirement Variability
Modelling Concepts in RM Tools
要件のバライアビリティのモデリング：要件管理ツール

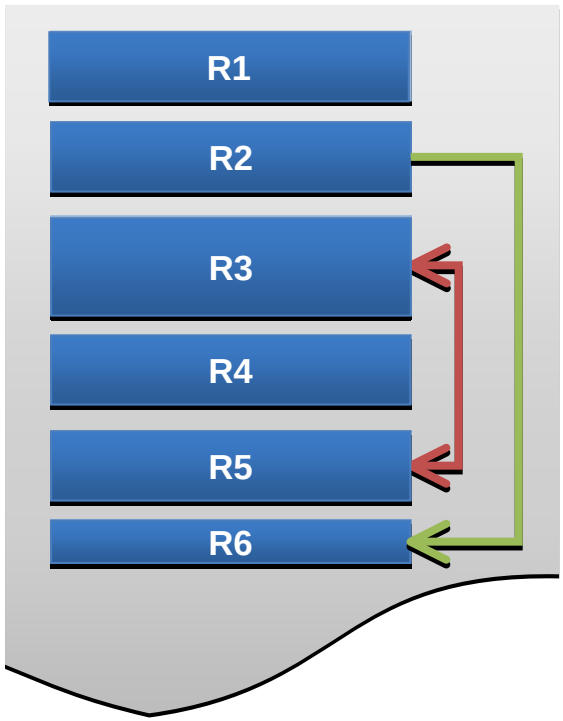
Requirement Selection
要件の選択

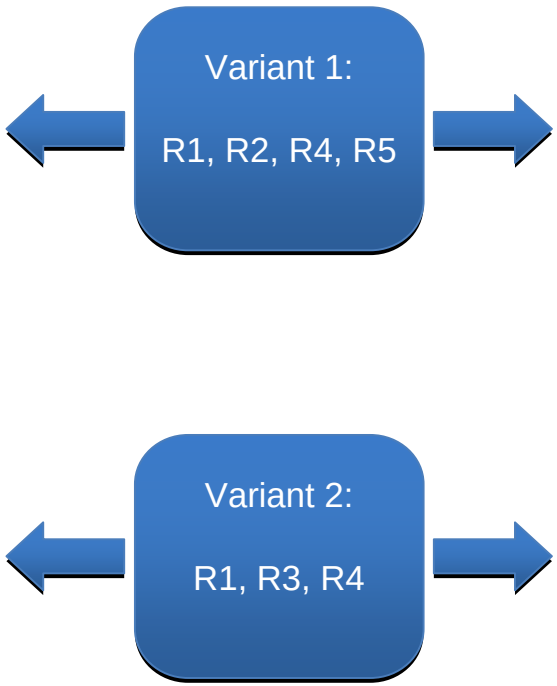


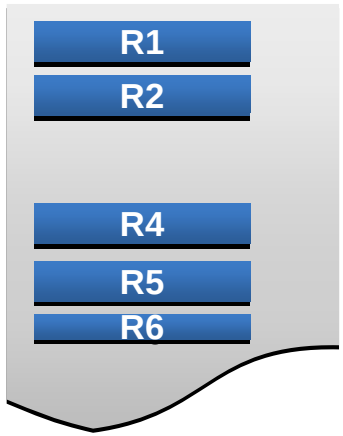
Requirement Selection
要件の選択

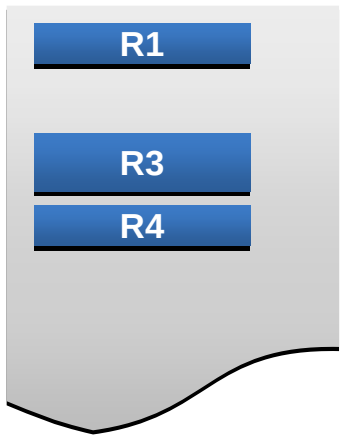


Dependency Rules
依存関係のルール

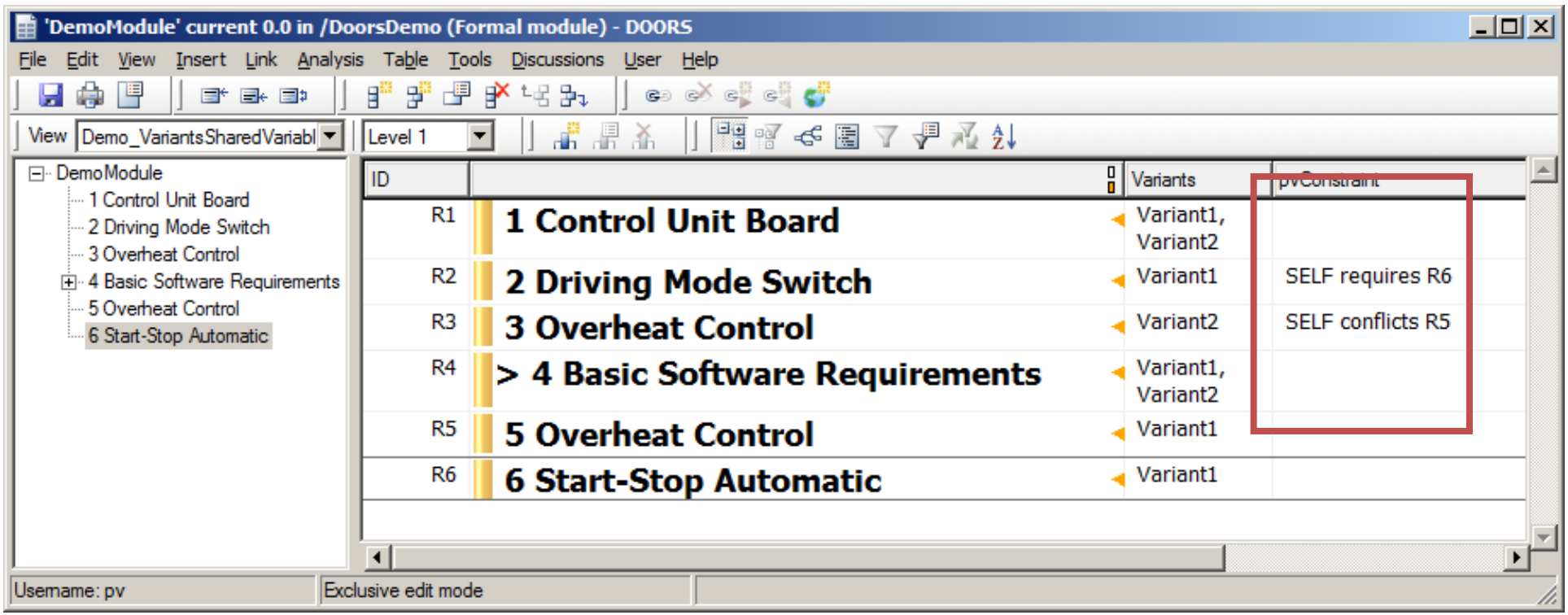




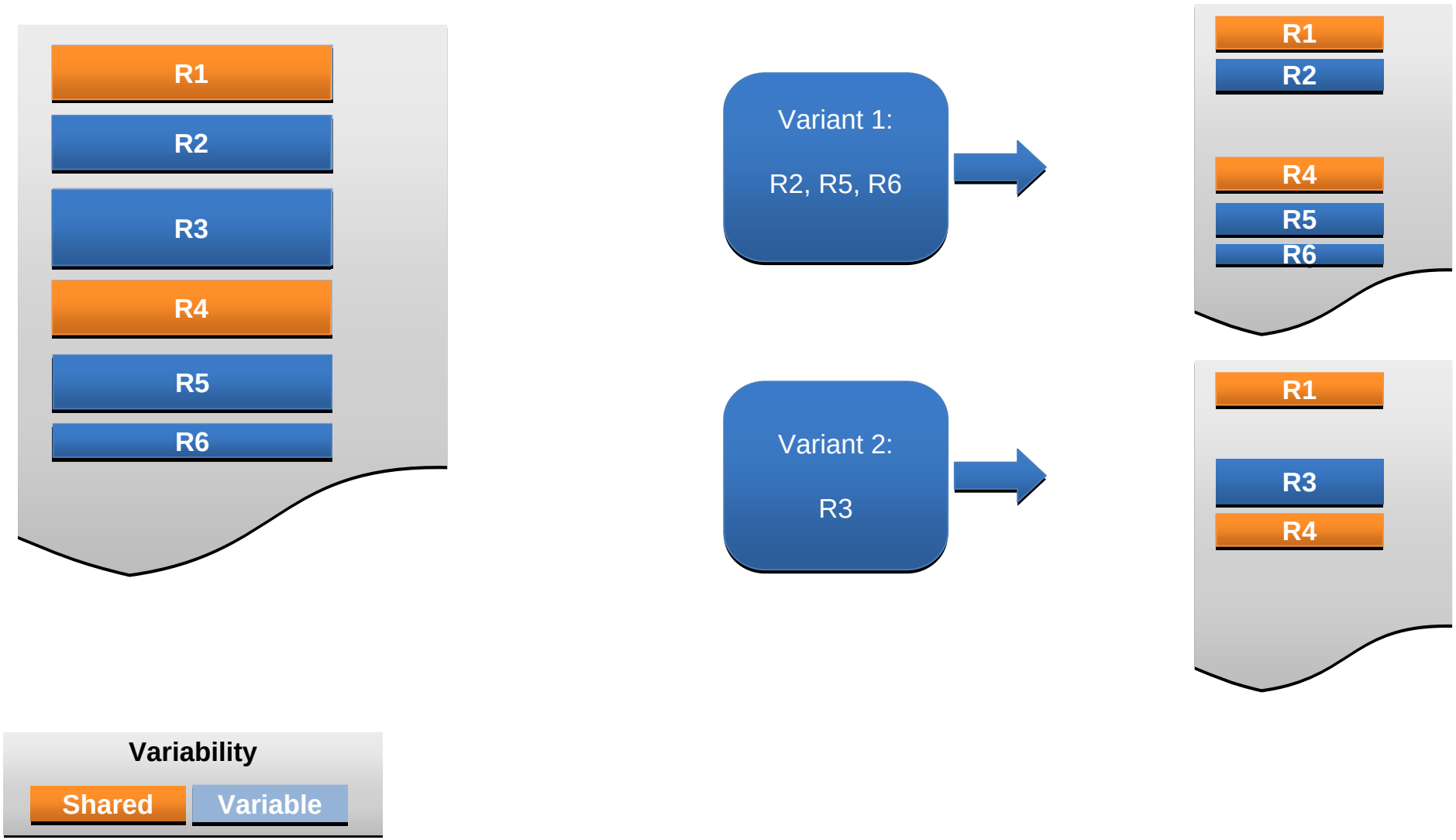




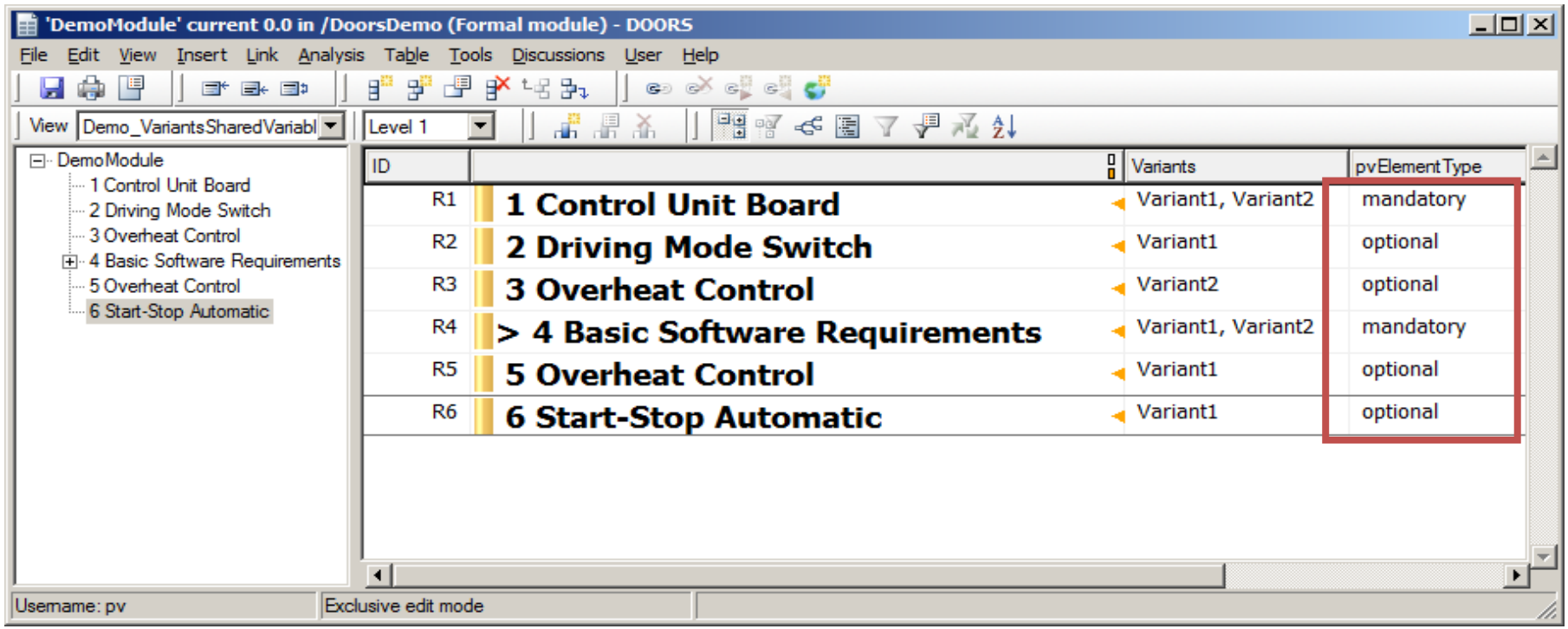
Dependency Rules
依存関係のルール



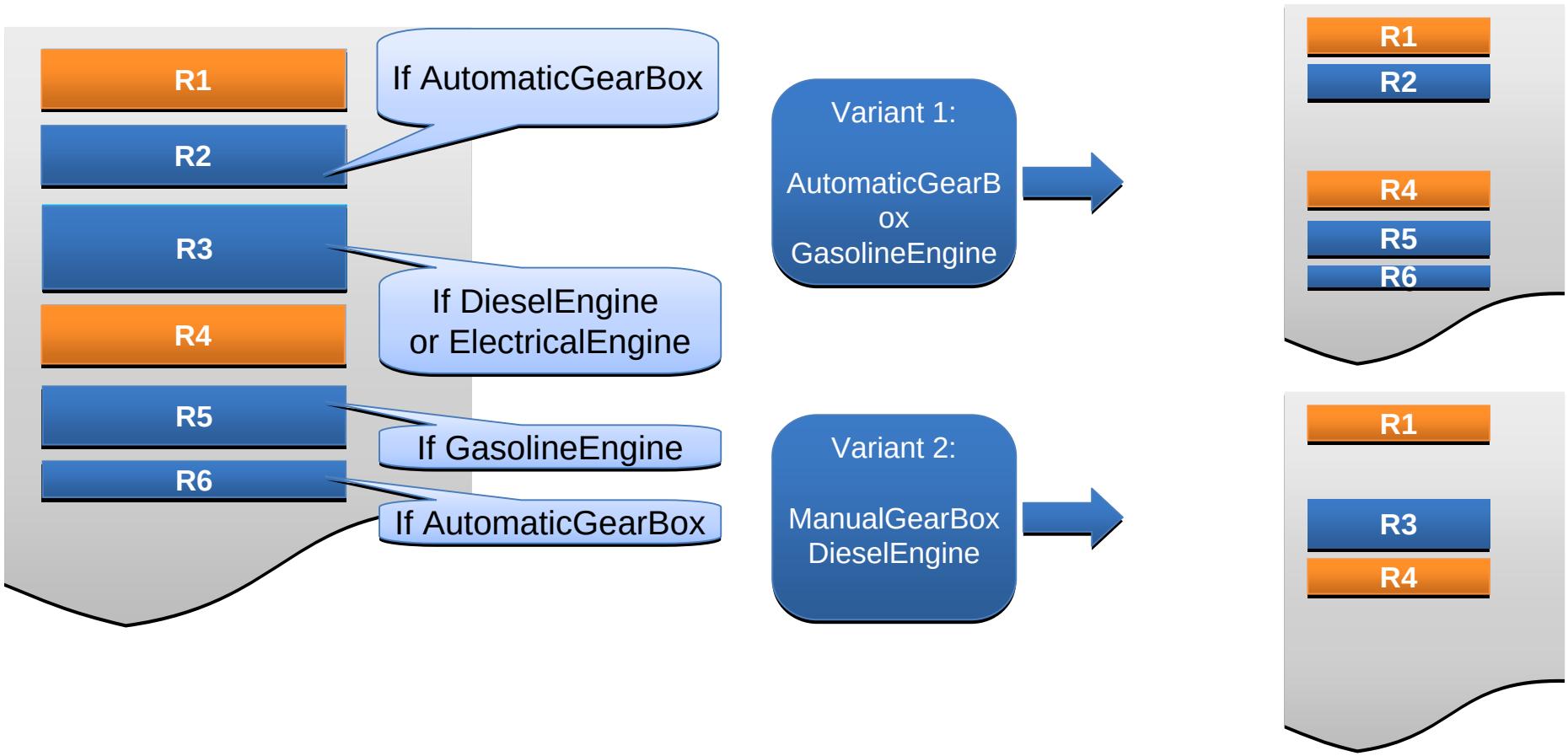
Shared / Variable Requirement Selection
共有/ 変動要件の選択



Shared / Variable Requirement Selection
共有/ 変動要件の選択



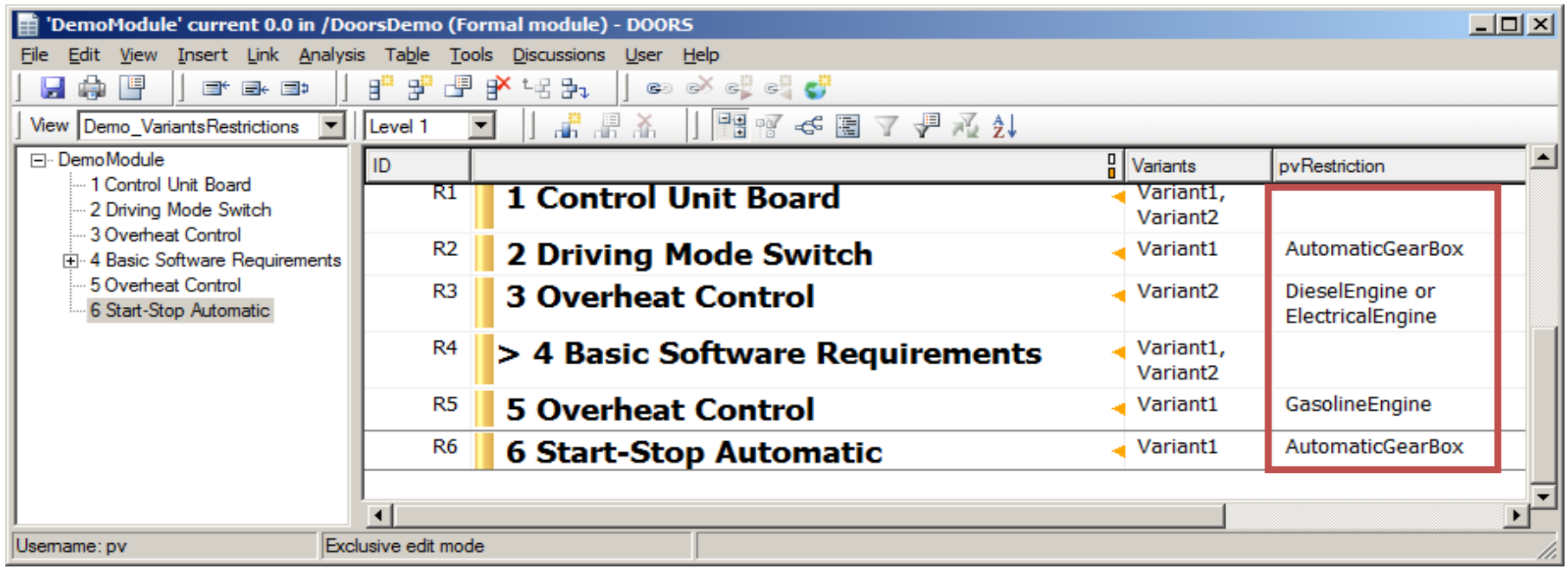
Selection Rules
選択のルール



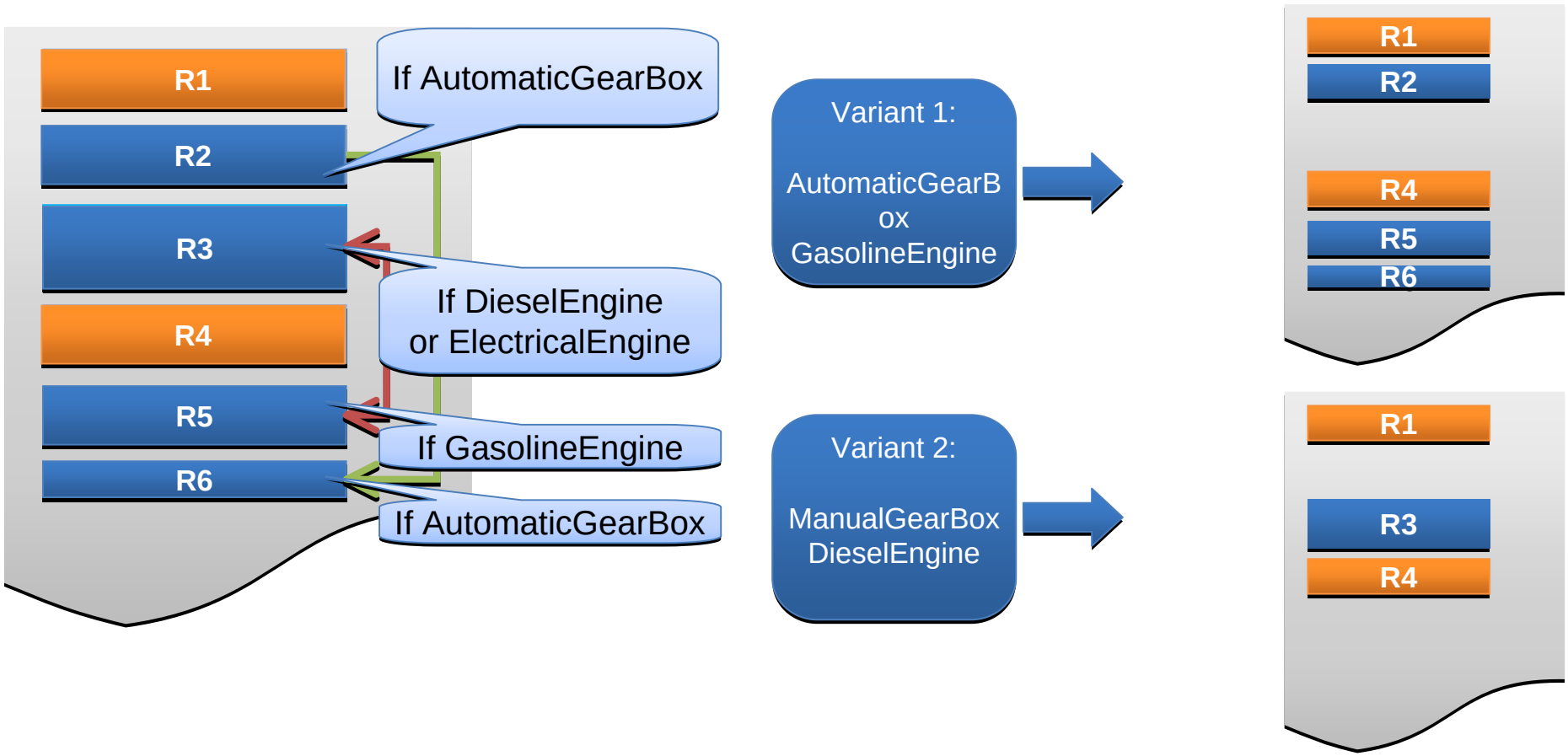
Variability

Shared **Variable**

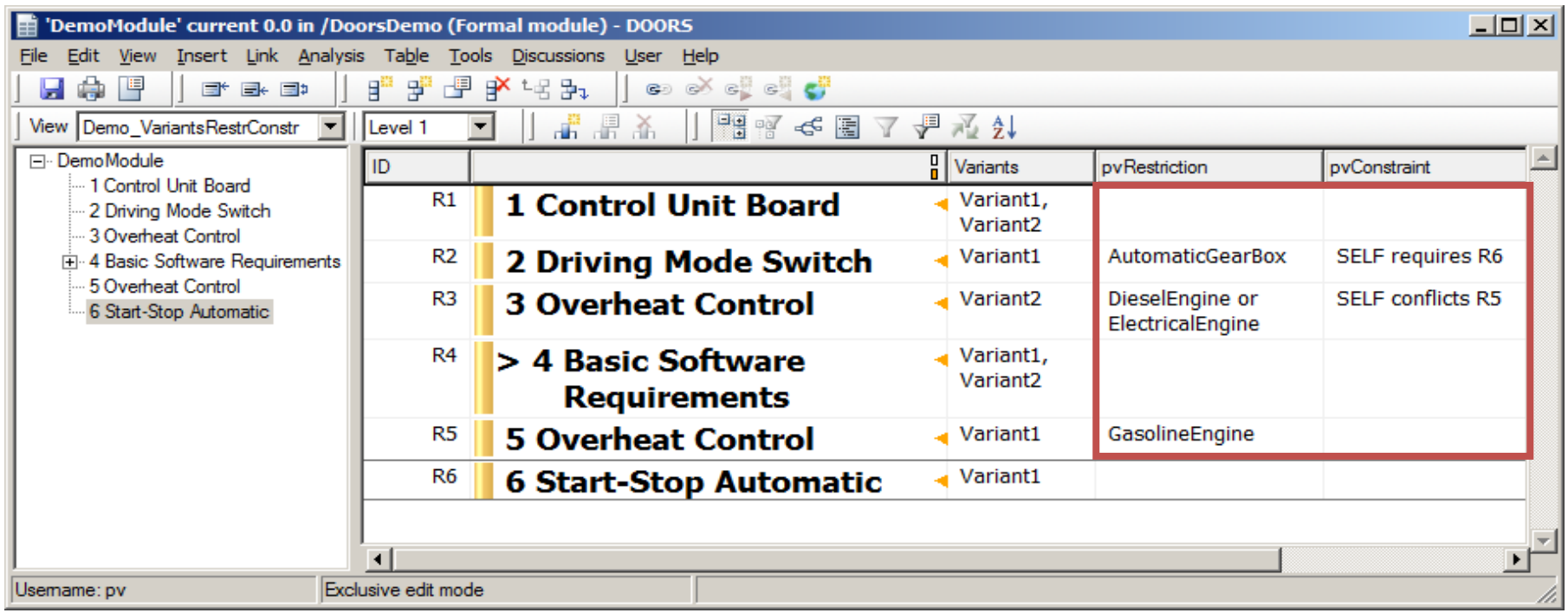
Selection Rules
選択のルール



All Combined
全てを融合



All Combined
全てを融合



Requirement Variation Point Modelling
要件上のバリエーションポイントのモデリング

Directly in RE tool

要件管理ツールに直接

Pro:

- Variability defined with requirements
  要件にバライアビリティを定義できる

Con:

- Changed RE information model
  RE**の情報モデルが変更されてしまう**
- Missing link to other variability models
  他のバライアビリティモデルへのリンクの欠如

In external tool

外部ツール

Pro:

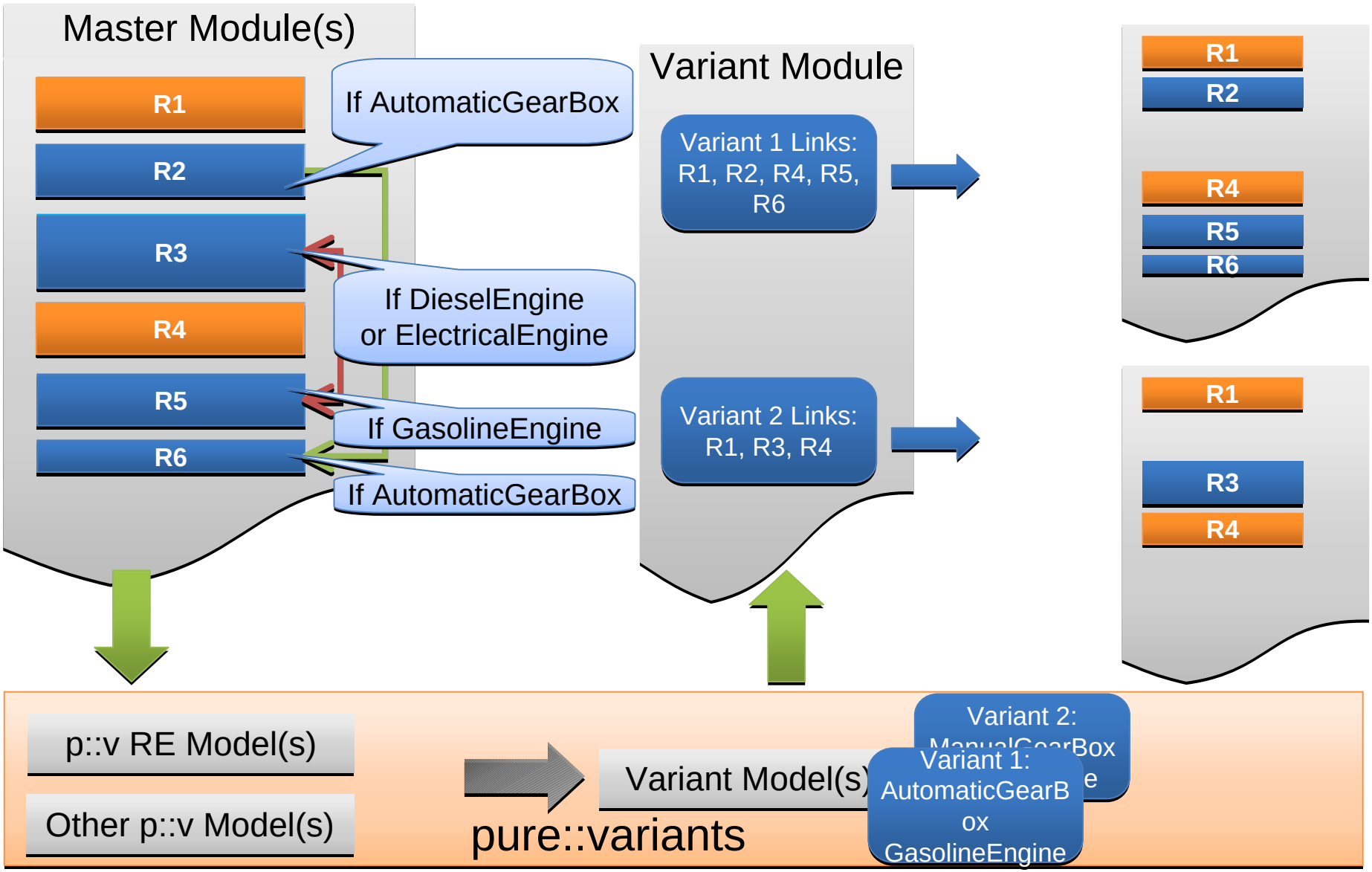
- Unchanged RE information model
  REの情報モデルが変更されない
- Uniform variability modelling
  バライアビリティモデリングを単一化

Con:

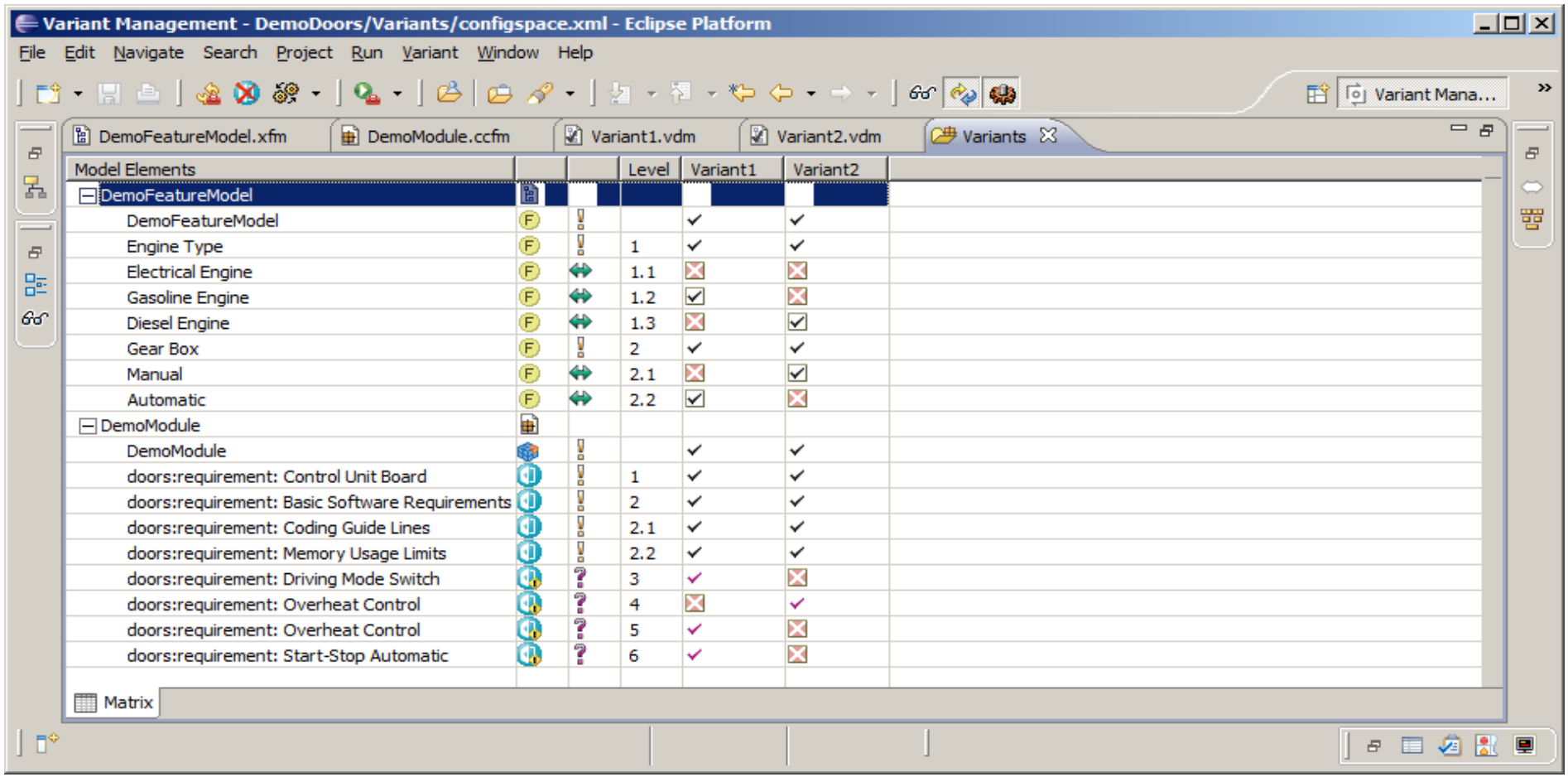
- Consistency Issues
  一貫性･整合性の課題
- Variability not visible in RE tool
  バライアビリティが要件管理ツールで見えない

pure::variants for Requirements

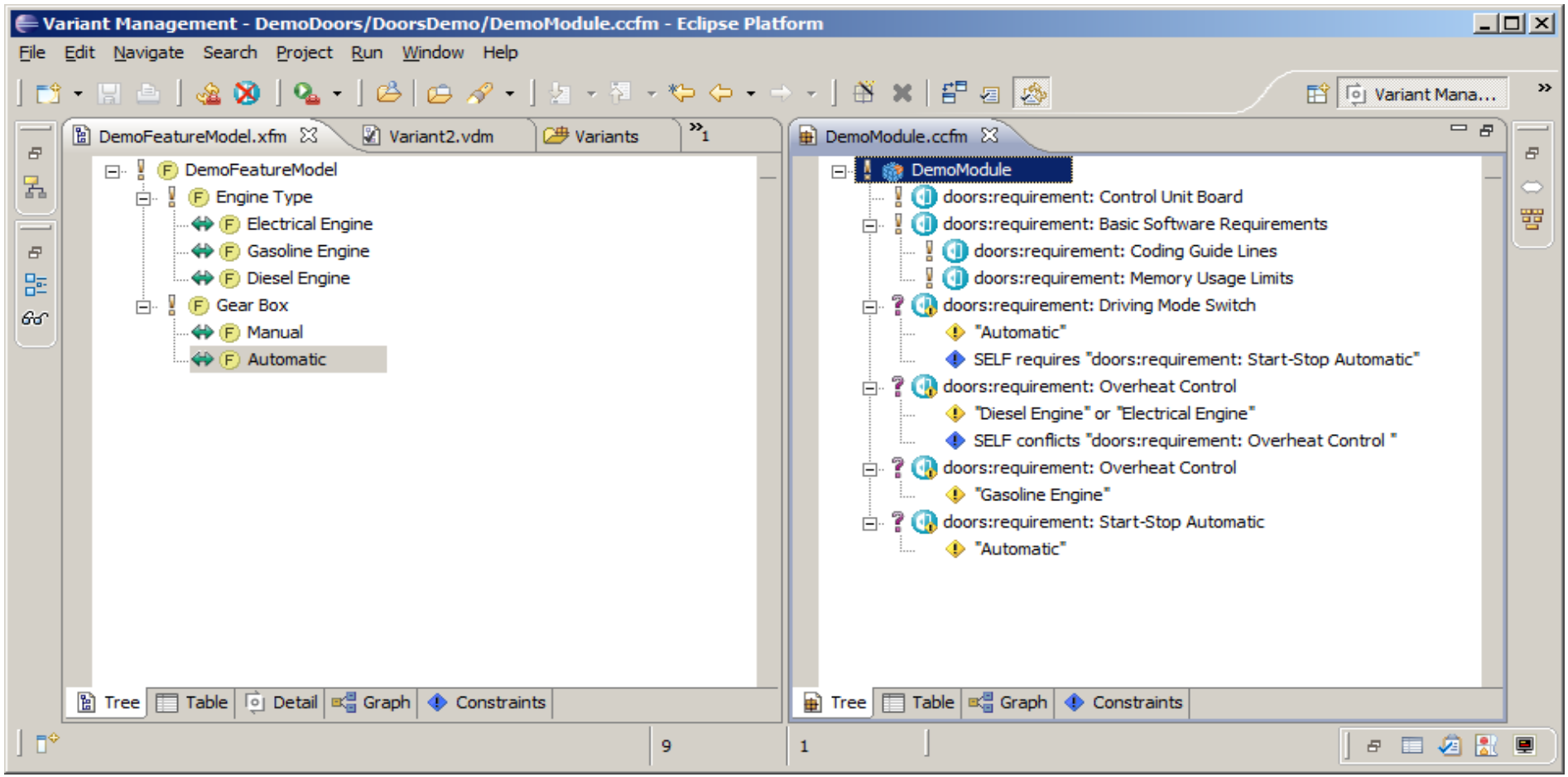
Best of Both – pure::variants for Requirements



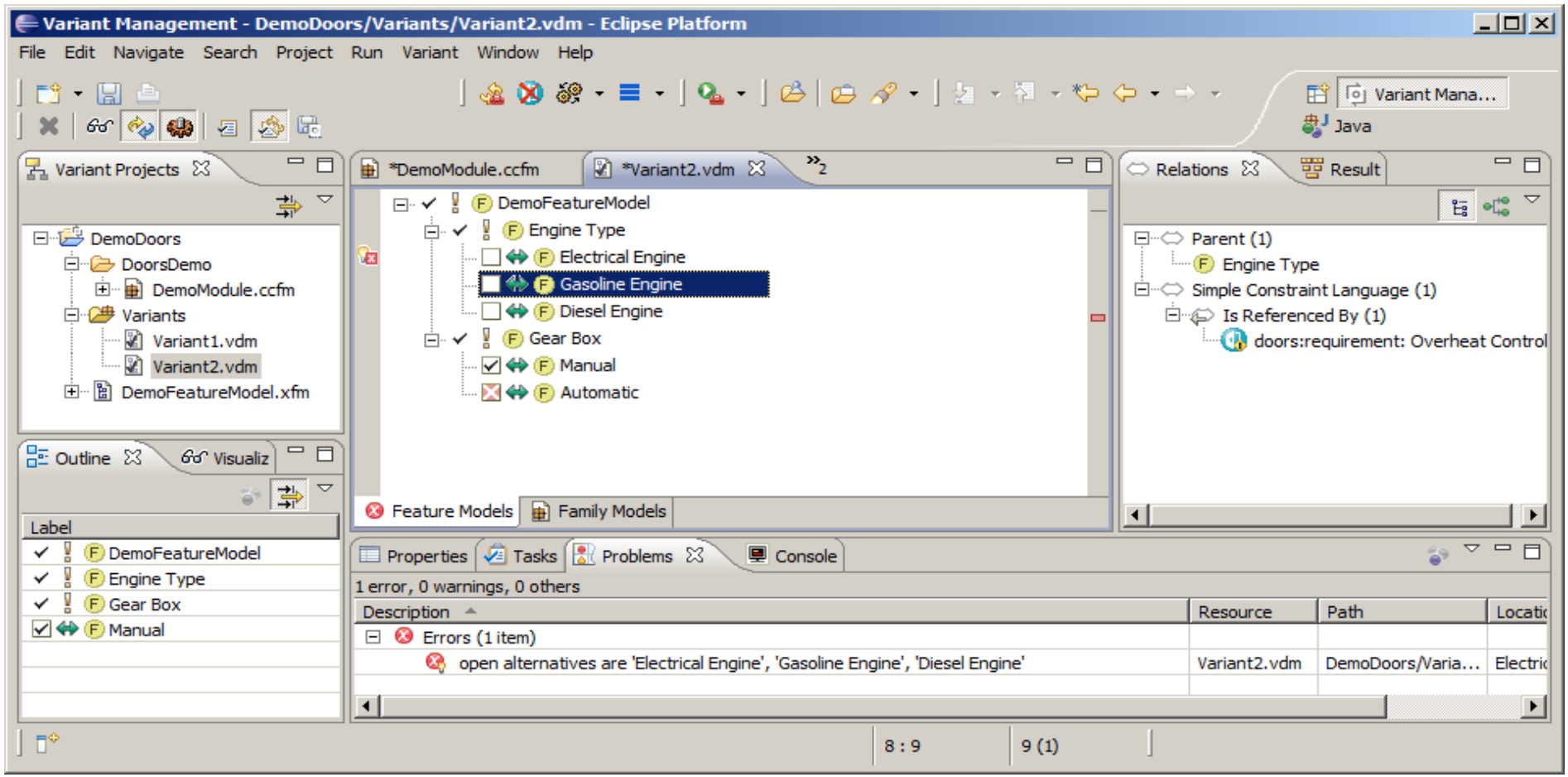
pure::variants Solution – Variant Matrix



pure::variants Solution – Models



pure::variants Solution – Configuration



Reuse Items

Test Cases

Estimates

Architecture

Code

Models

Requirements

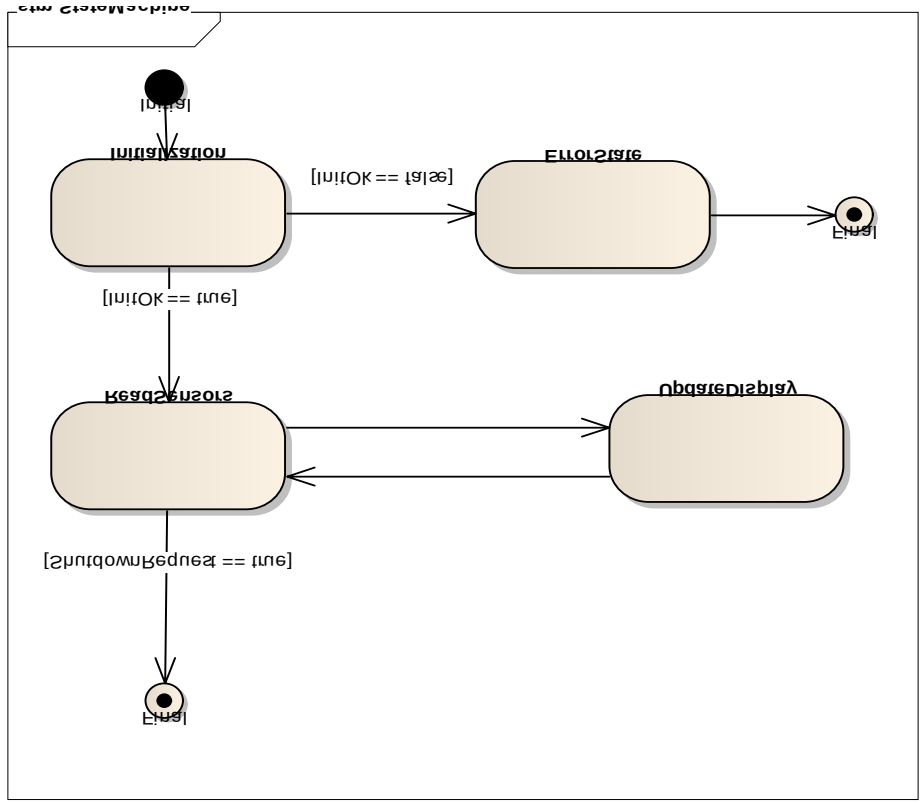
Subsystems

Documentation

Rules

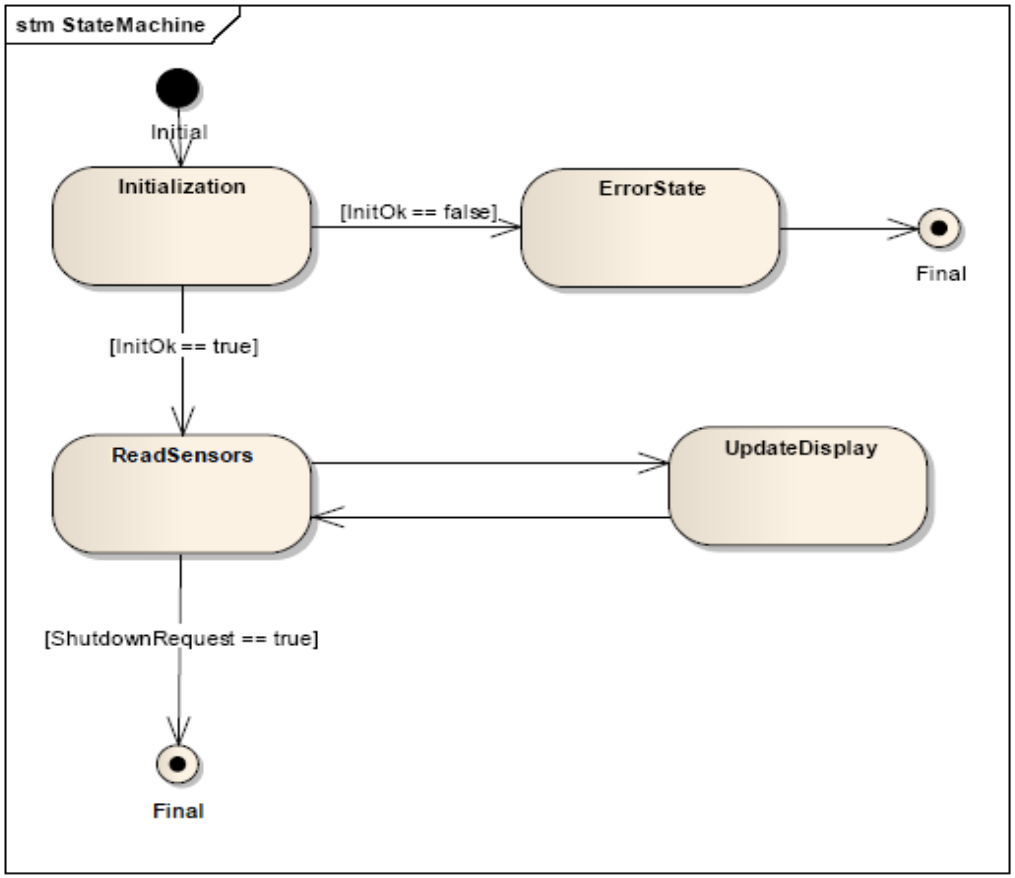
Complexity Reduction

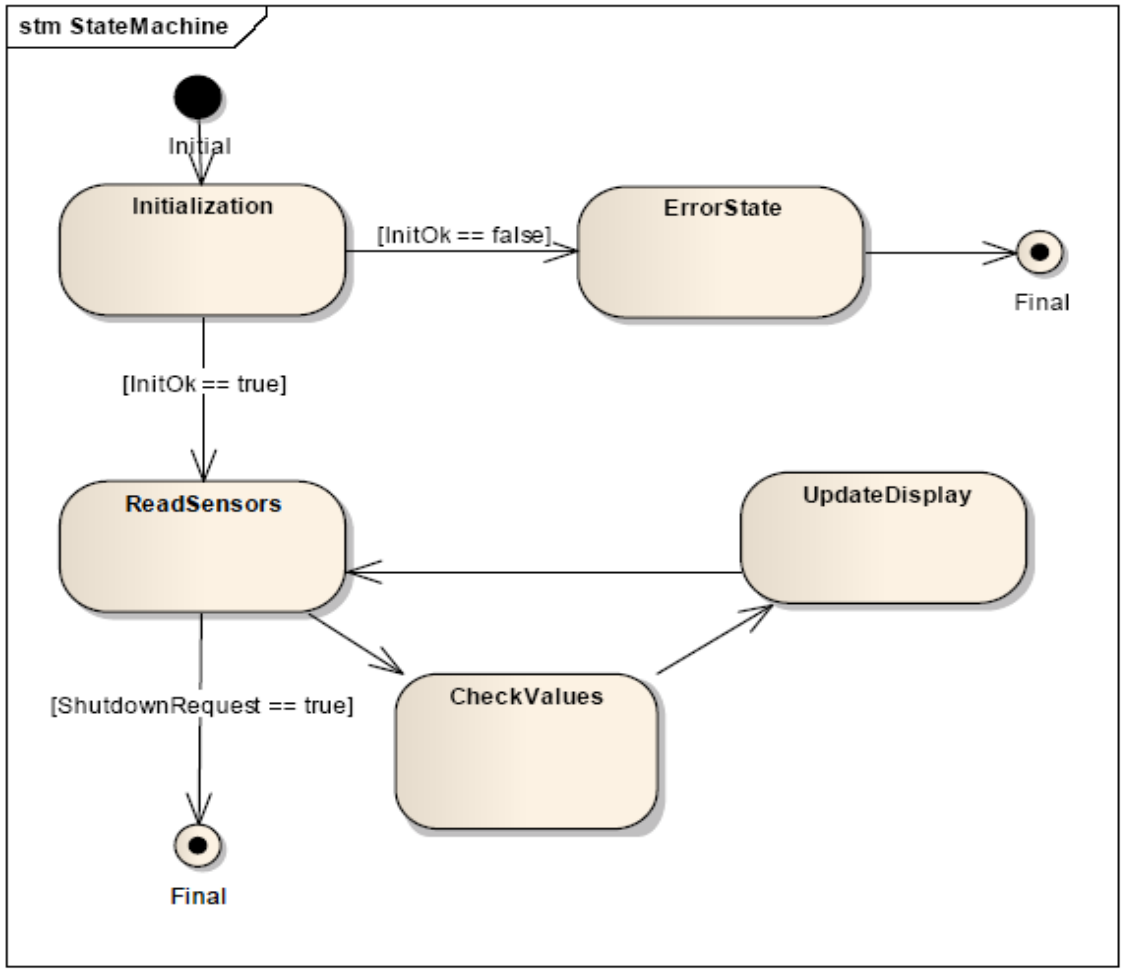
```
typedef enum states { INIT, FINAL, Initialization,
                      ReadSensor, UpdateDisplay, ErrorState
                    } state_t;

state_t state;
int main(int argc, char[]* argv)
{
    state = INIT;
    while (state != FINAL)
    {
        case INIT:
            app_init();
            state = Initialization;
            break;
        case Initialization:
            sensor_init();
            display_init();
            if (InitOK == 1) {
                state = ReadSensors;
            } else {
                state = ErrorState;
            }
            break;
        case ReadSensors:
            read_sensors();
            state = UpdateDisplay;
        if (ShutdownRequest == 1) {
            state = FINAL;
        }
        break;
        case UpdateDisplay:
            update_display();
            state = ReadSensors;
            break;
        case ErrorState:
            log_errorstate();
            state = FINAL;
            break;
    }
    app_shutdown();
}
```



Variability in UML

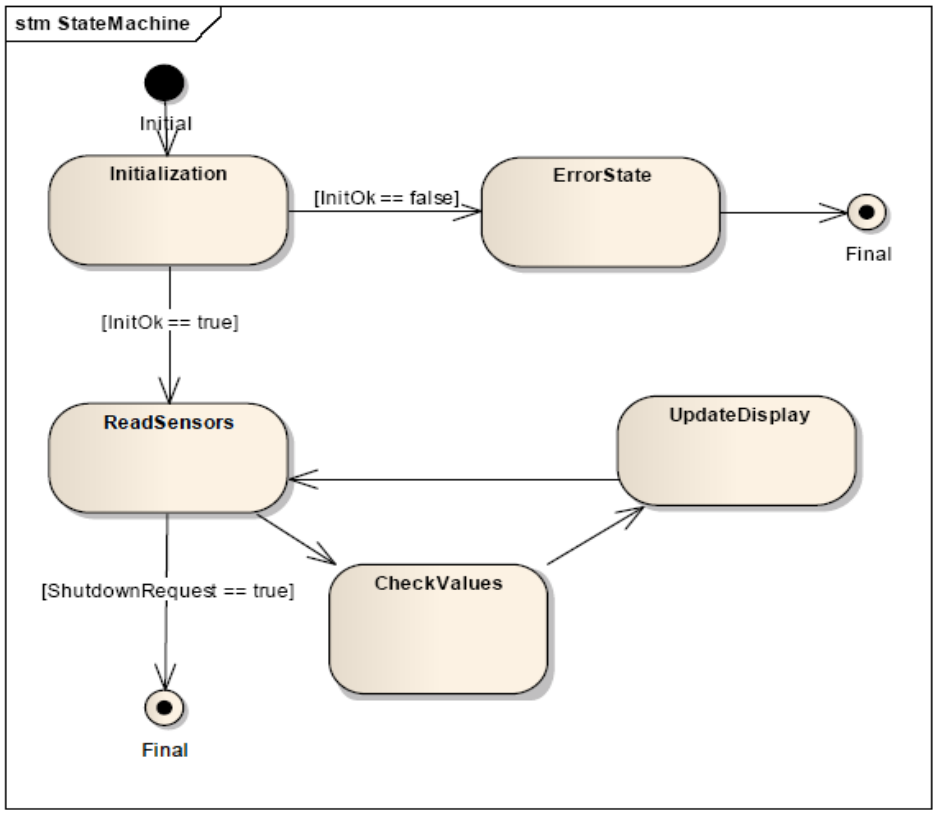
Initial UML Model

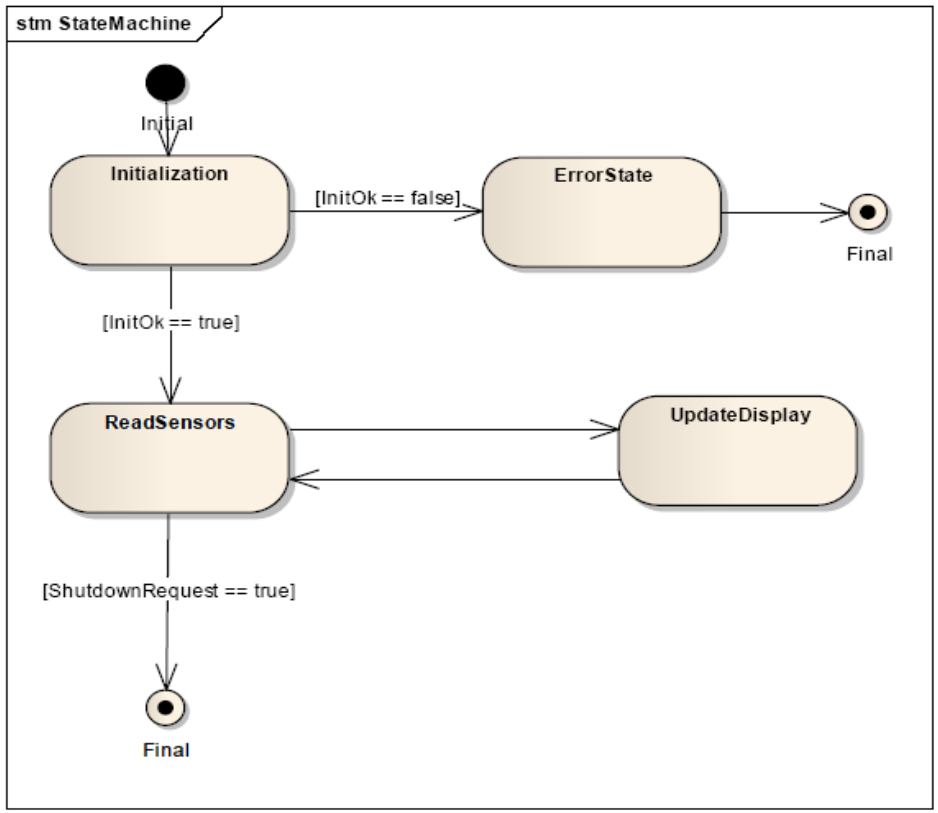




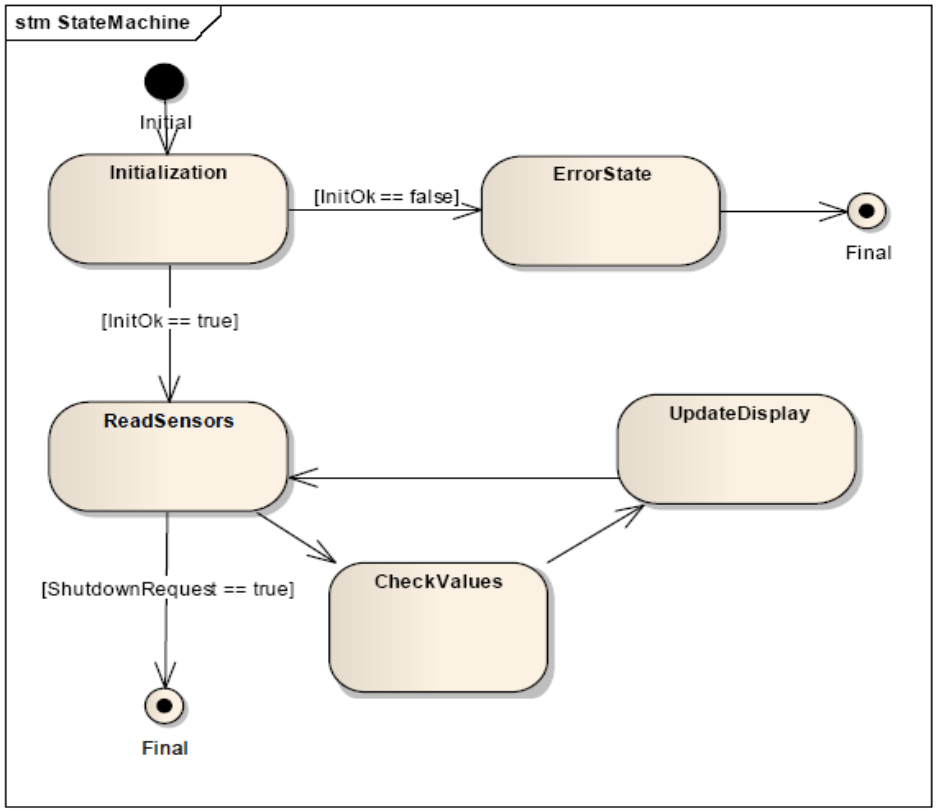
Changed Model

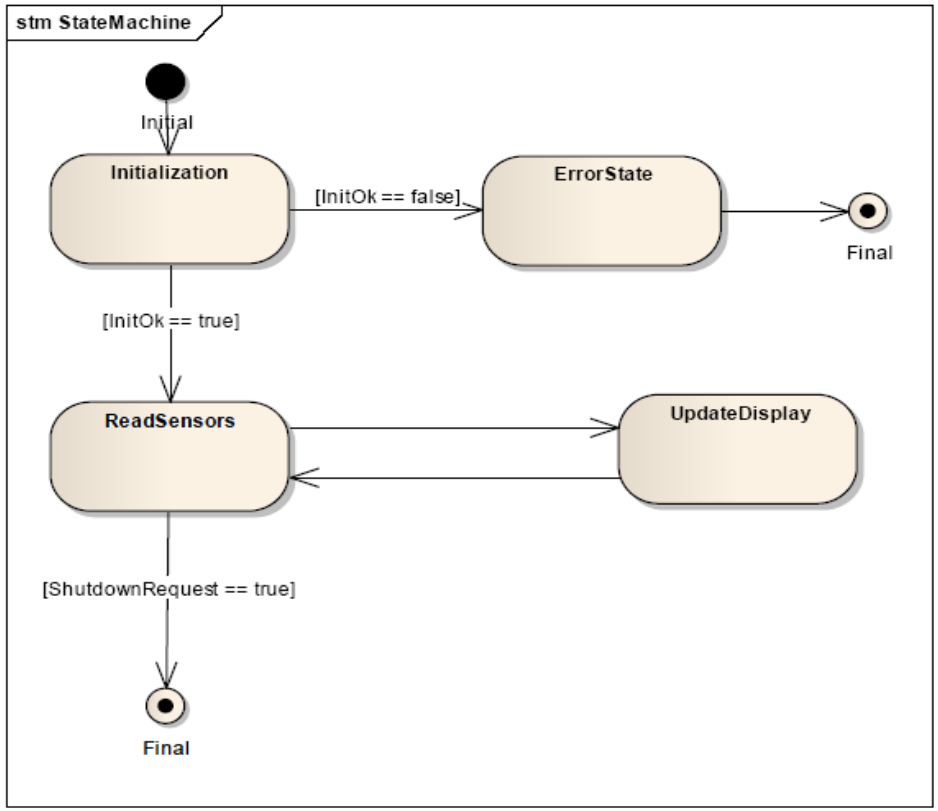
The Result – 2 Variants





Unified UML Model The Result – 2 Variants

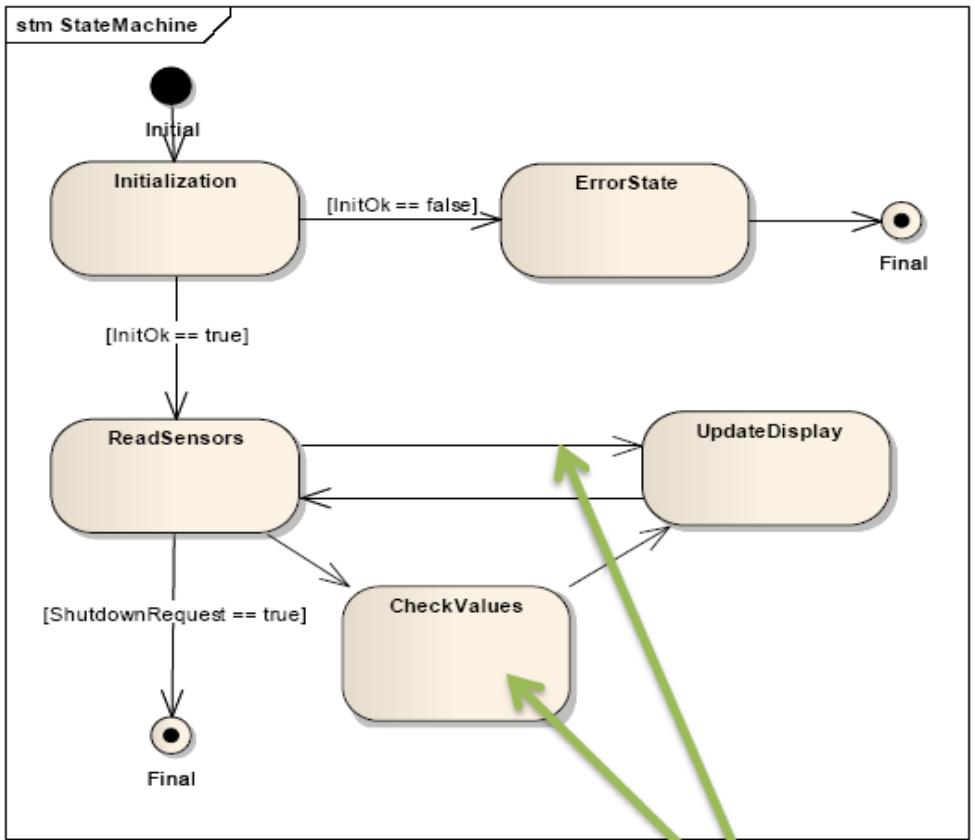




Variant 1

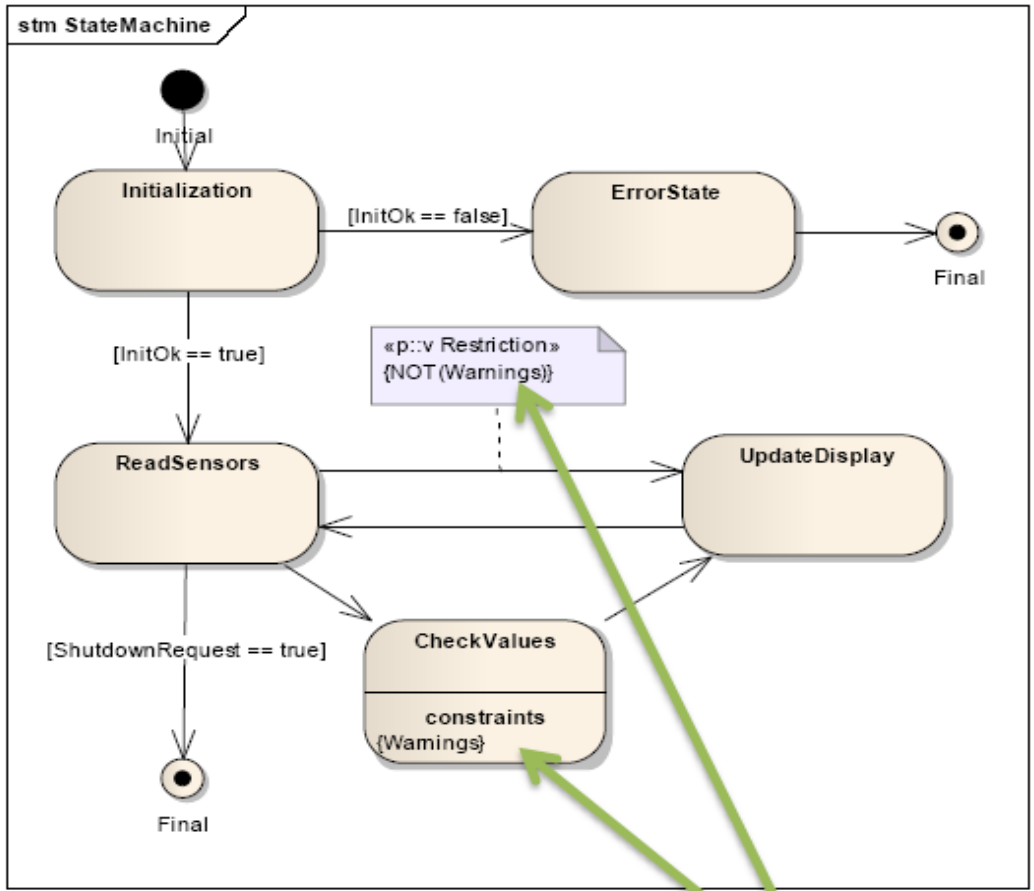
Variant 2

Unified UML Model



2 Variation Points

Linked to Variability Model

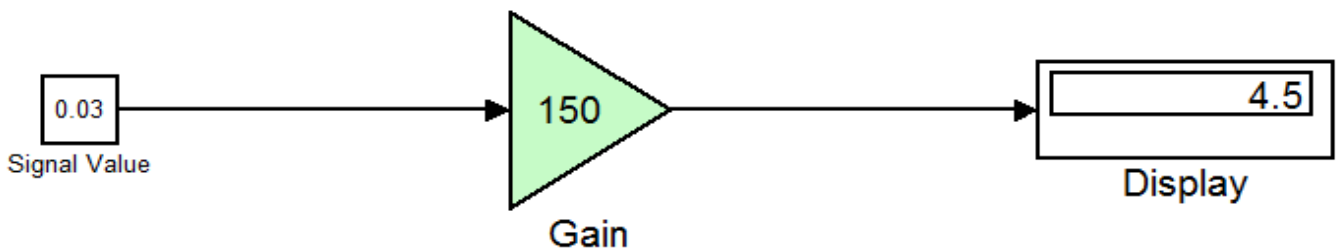


2 Rules

pure::variants for UML

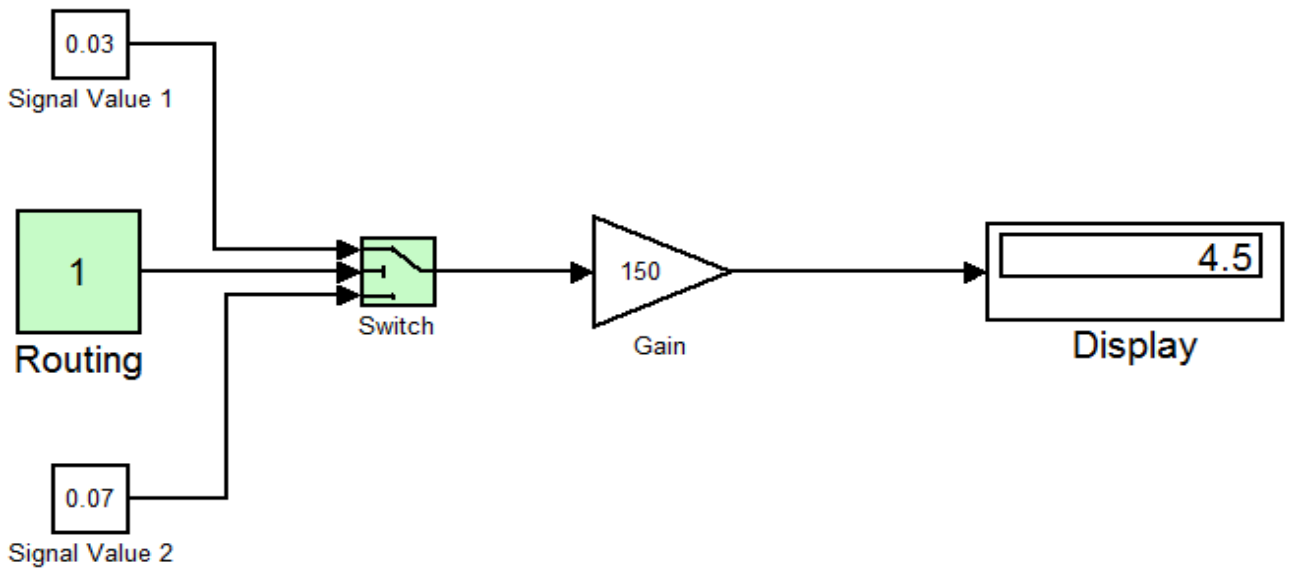
Variability in Simulink

Expressing Variability in Simulink



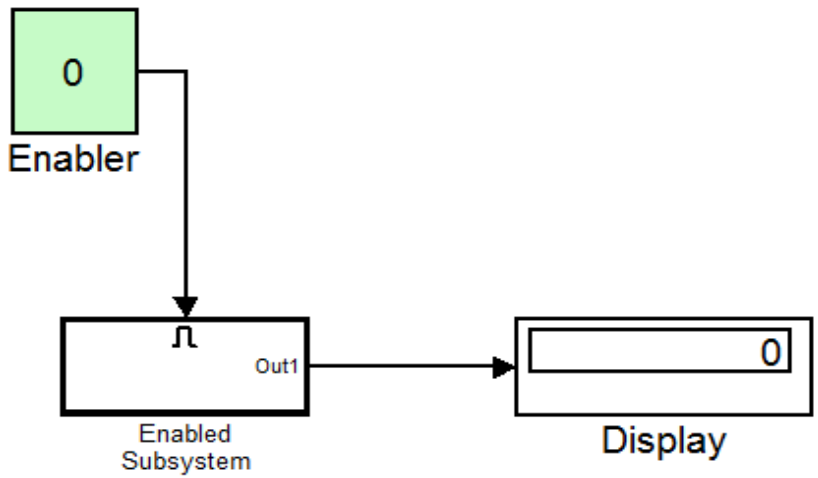
Parametrization

Expressing Variability in Simulink



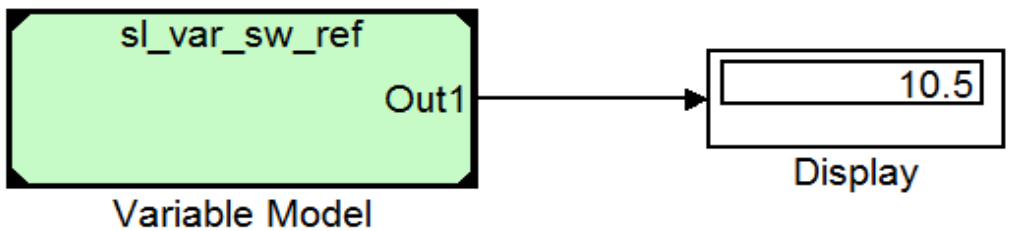
Signal Routing

Expressing Variability in Simulink



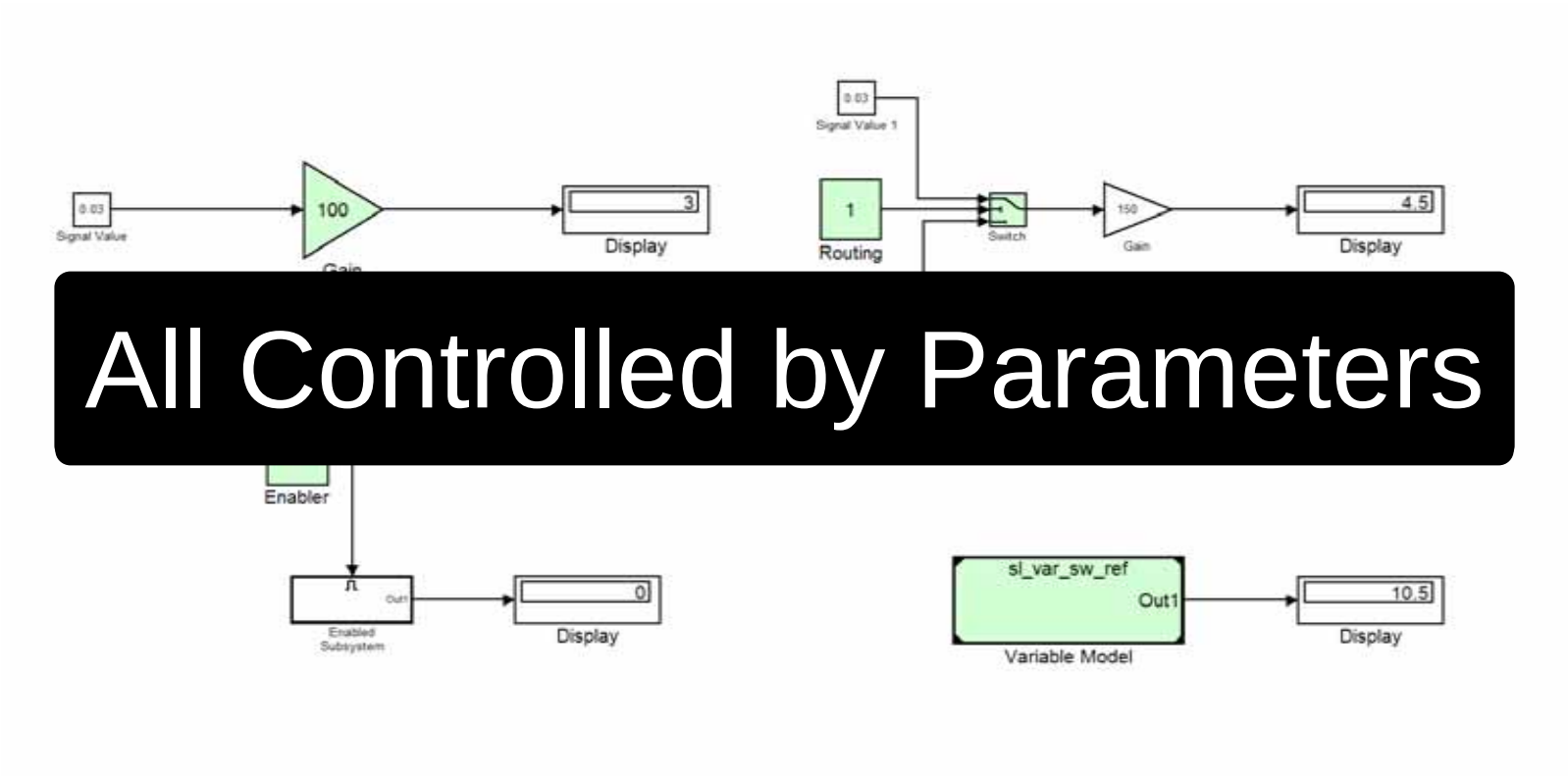
Subsystems

Expressing Variability in Simulink

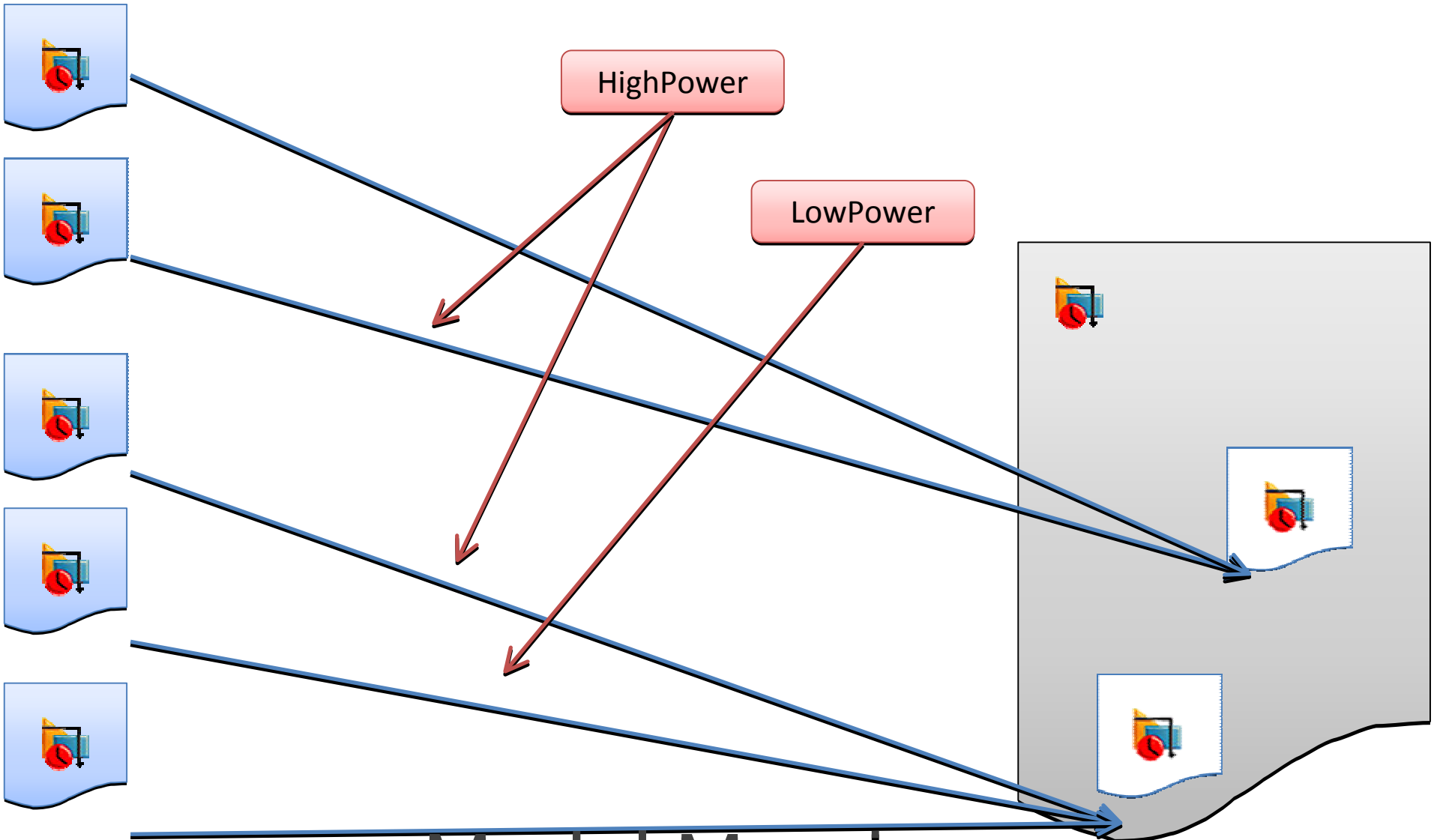


Model Reference

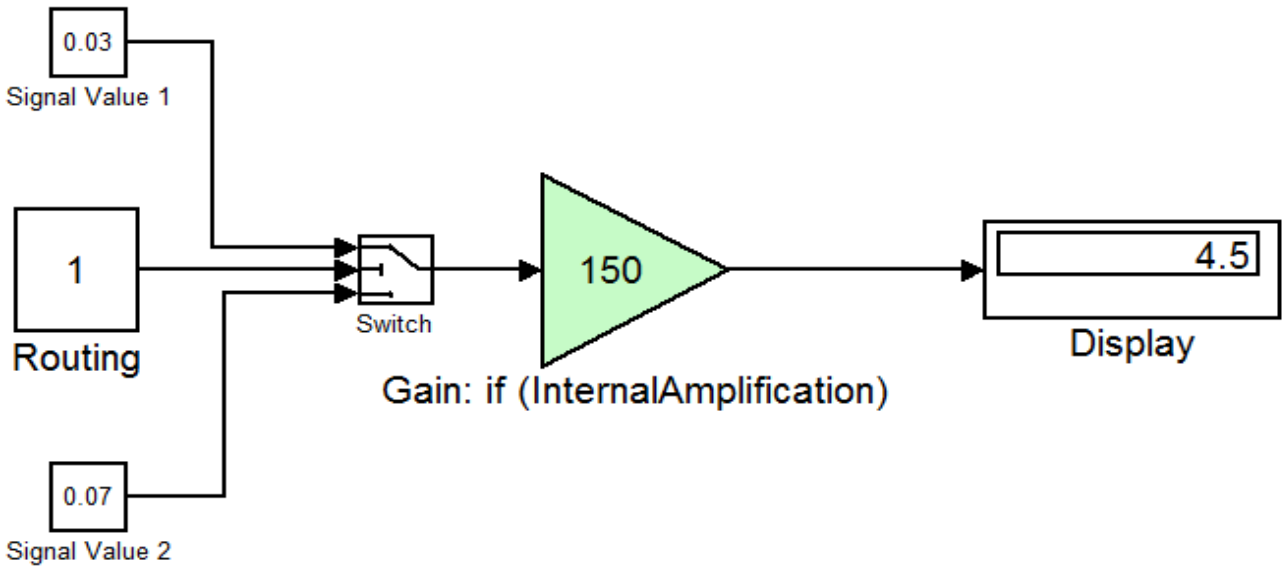
Expressing Variability in Simulink



Expressing Variability in Simulink

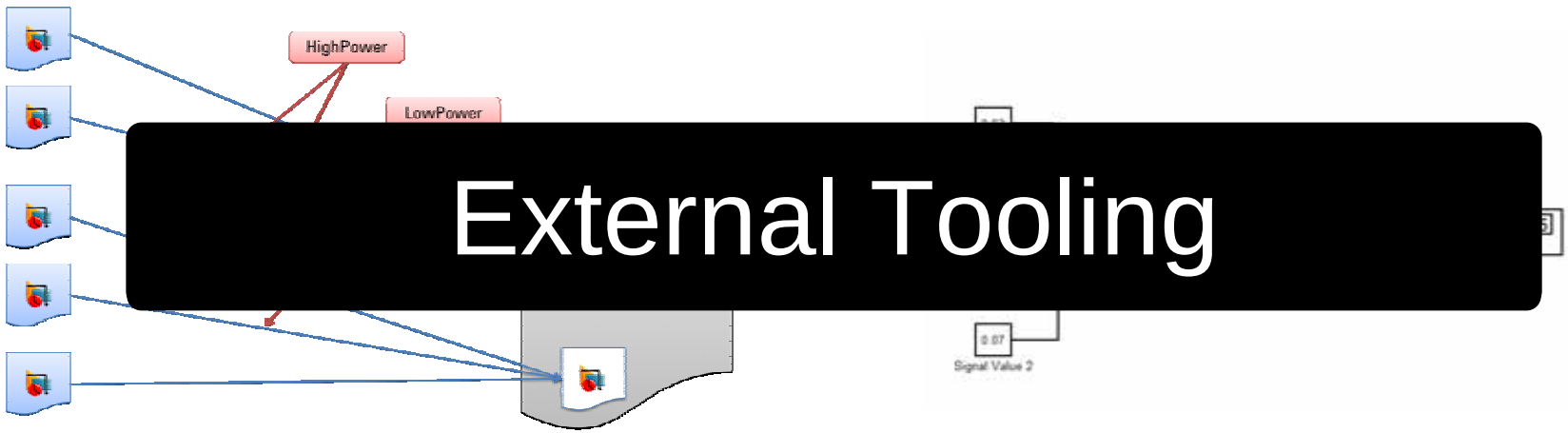


Expressing Variability in Simulink



Structural Manipulation

Expressing Variability in Simulink



Expressing Variability in Simulink

Using Code Generation (TargetLink):

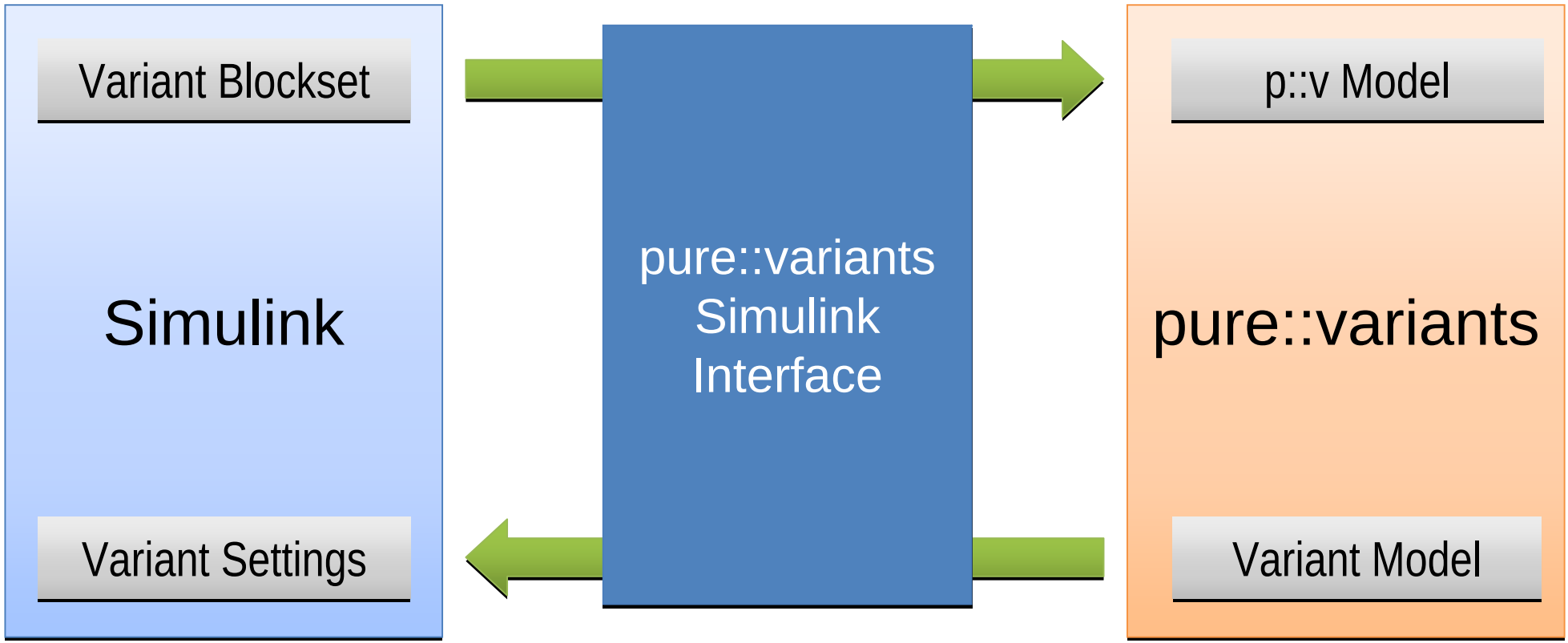
- Variable Classes
- Code or Data Variants in Data Dictionary
- Generated preprocessor statements

Variability in Stateflow:

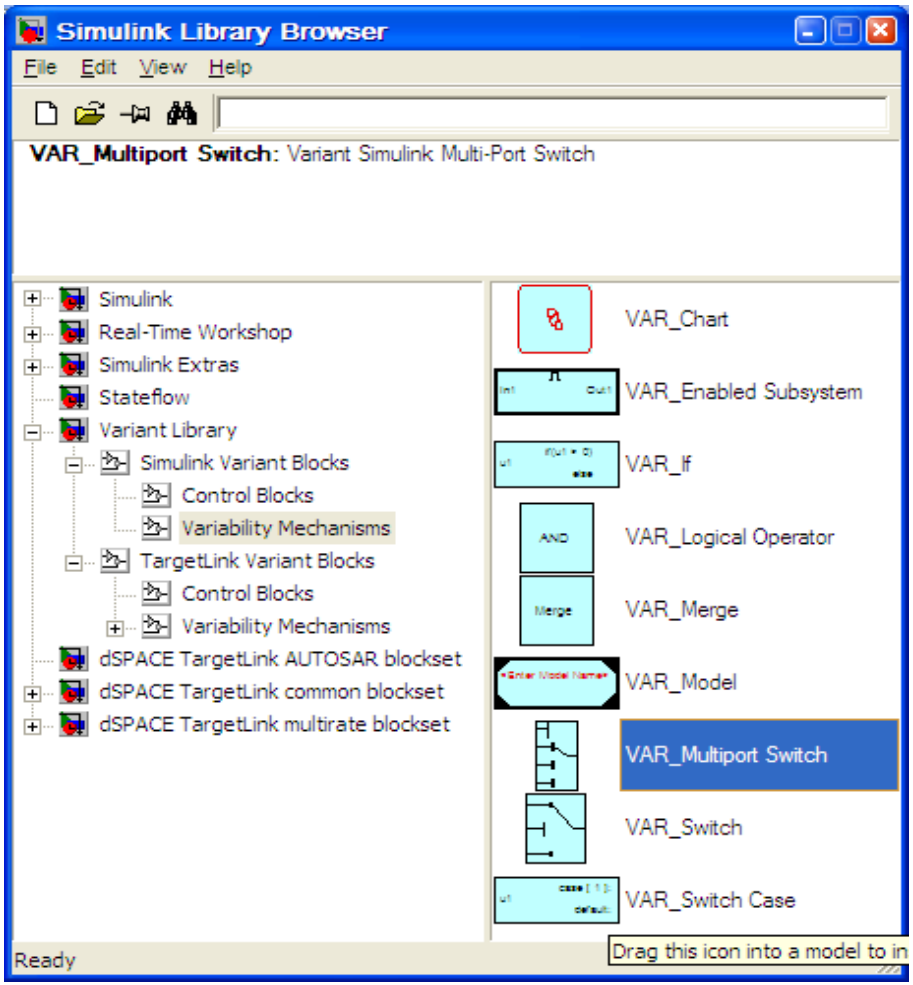
- Use of “variation point” inputs
- Model manipulation (pure::variants Connector for Simulink)

pure::variants for Simulink

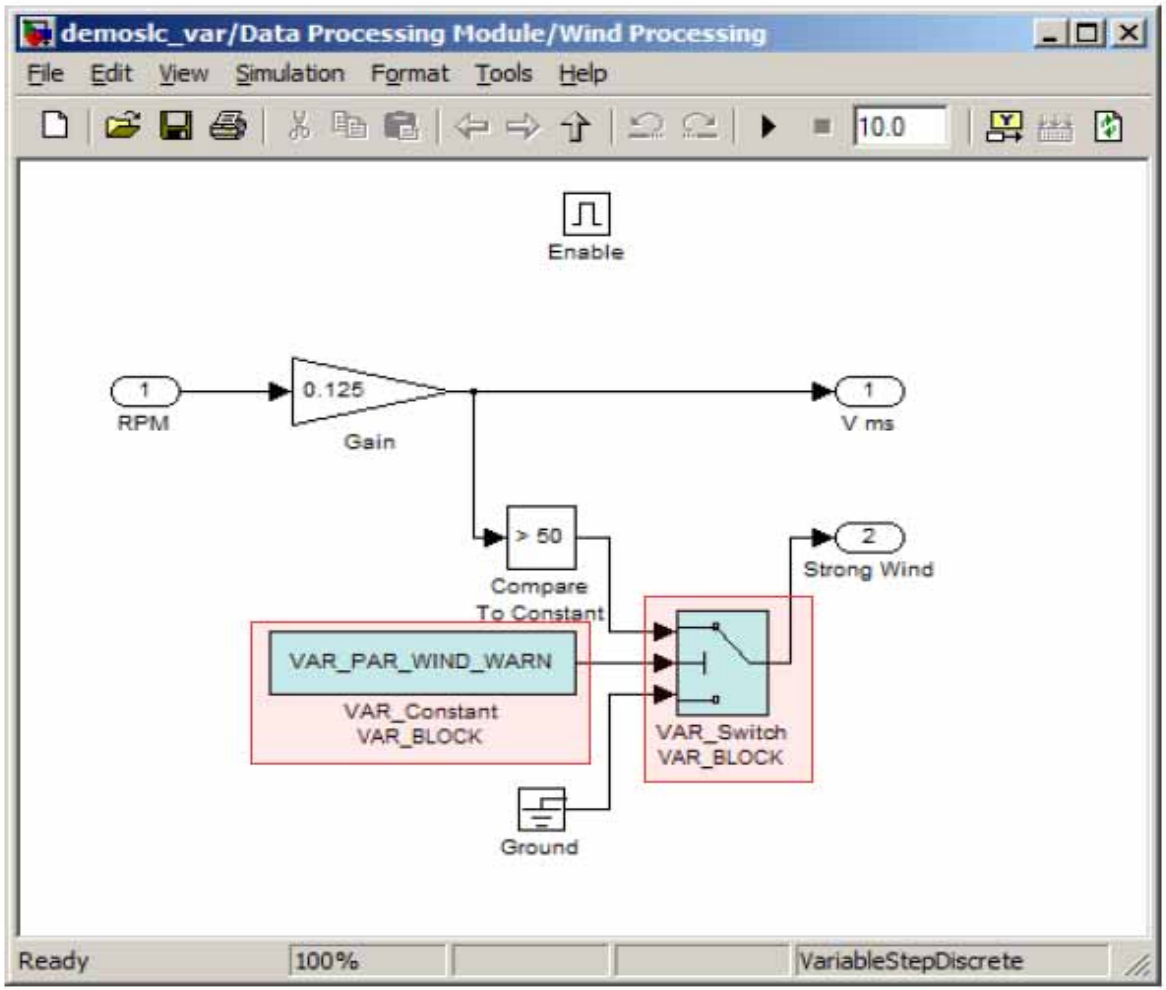
Overview



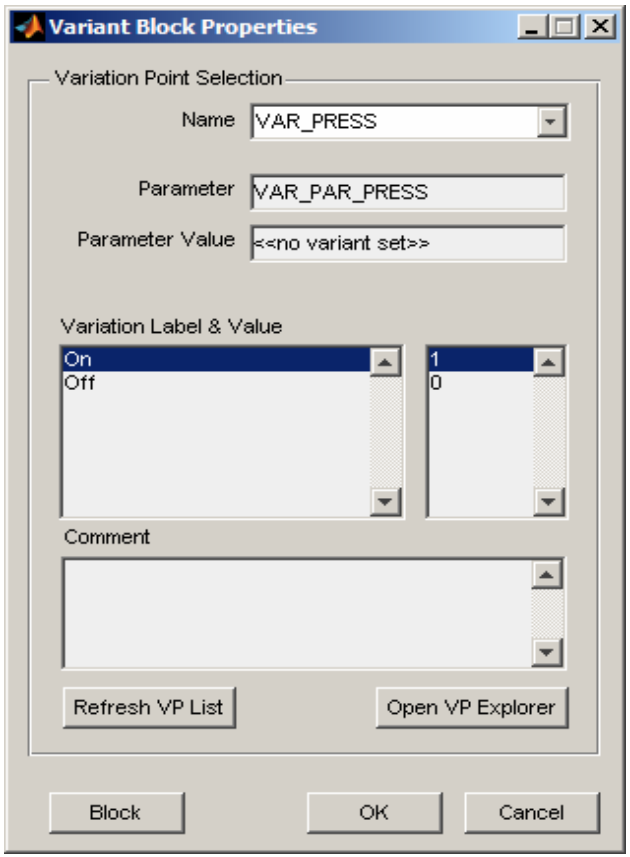
Variant Blockset

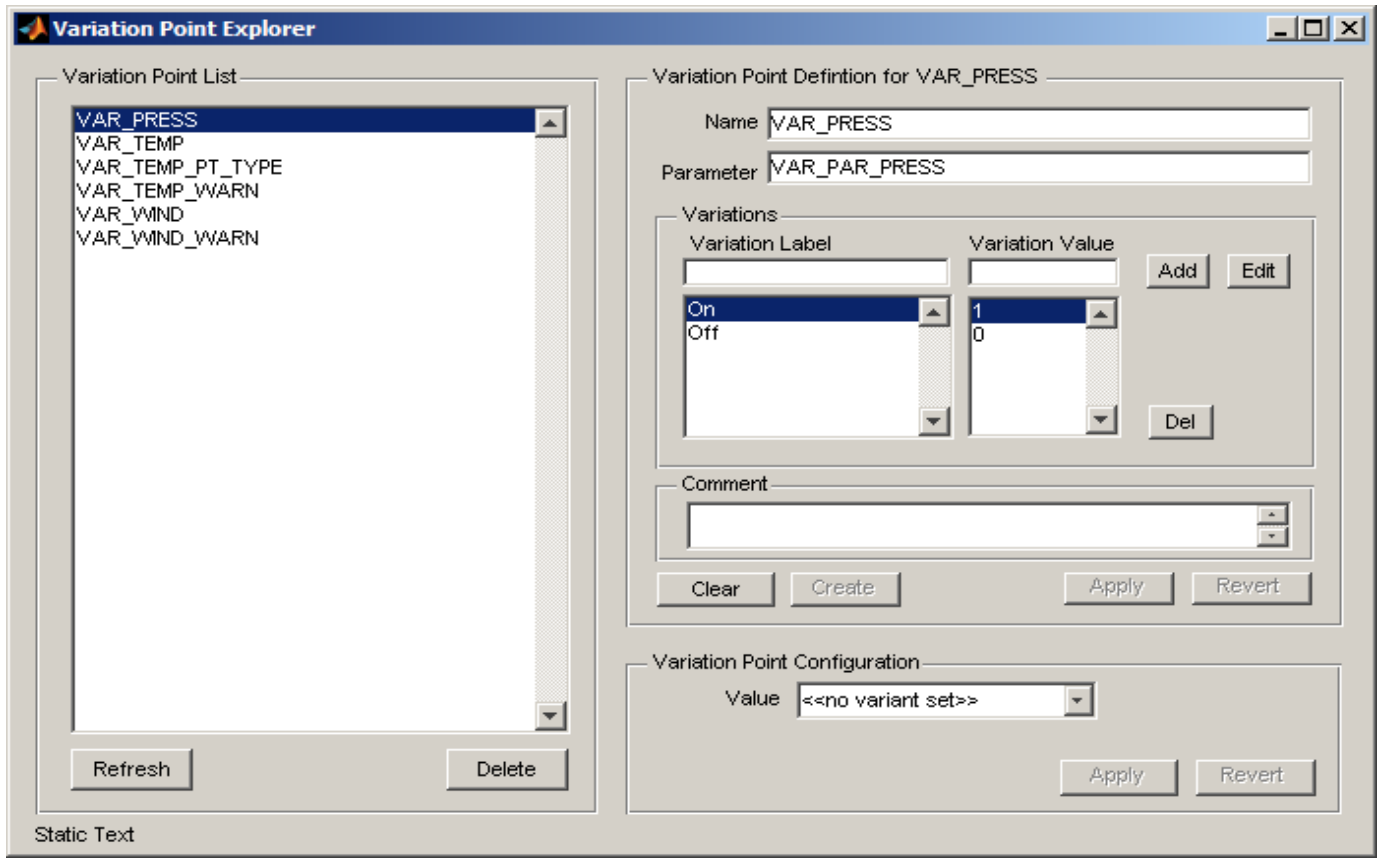


Variant Blocks in Model

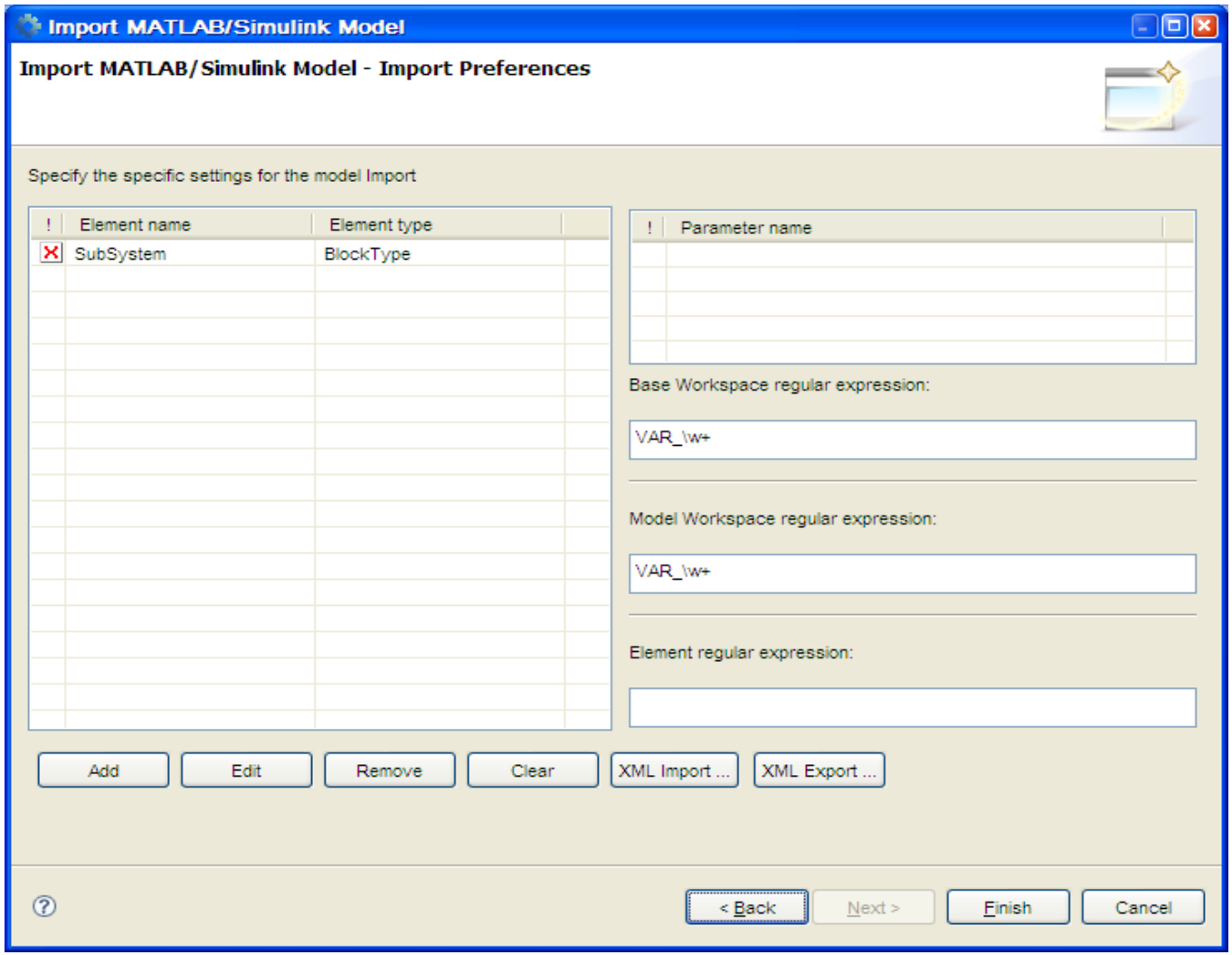


Variant Block Properties and Variation Point Explorer

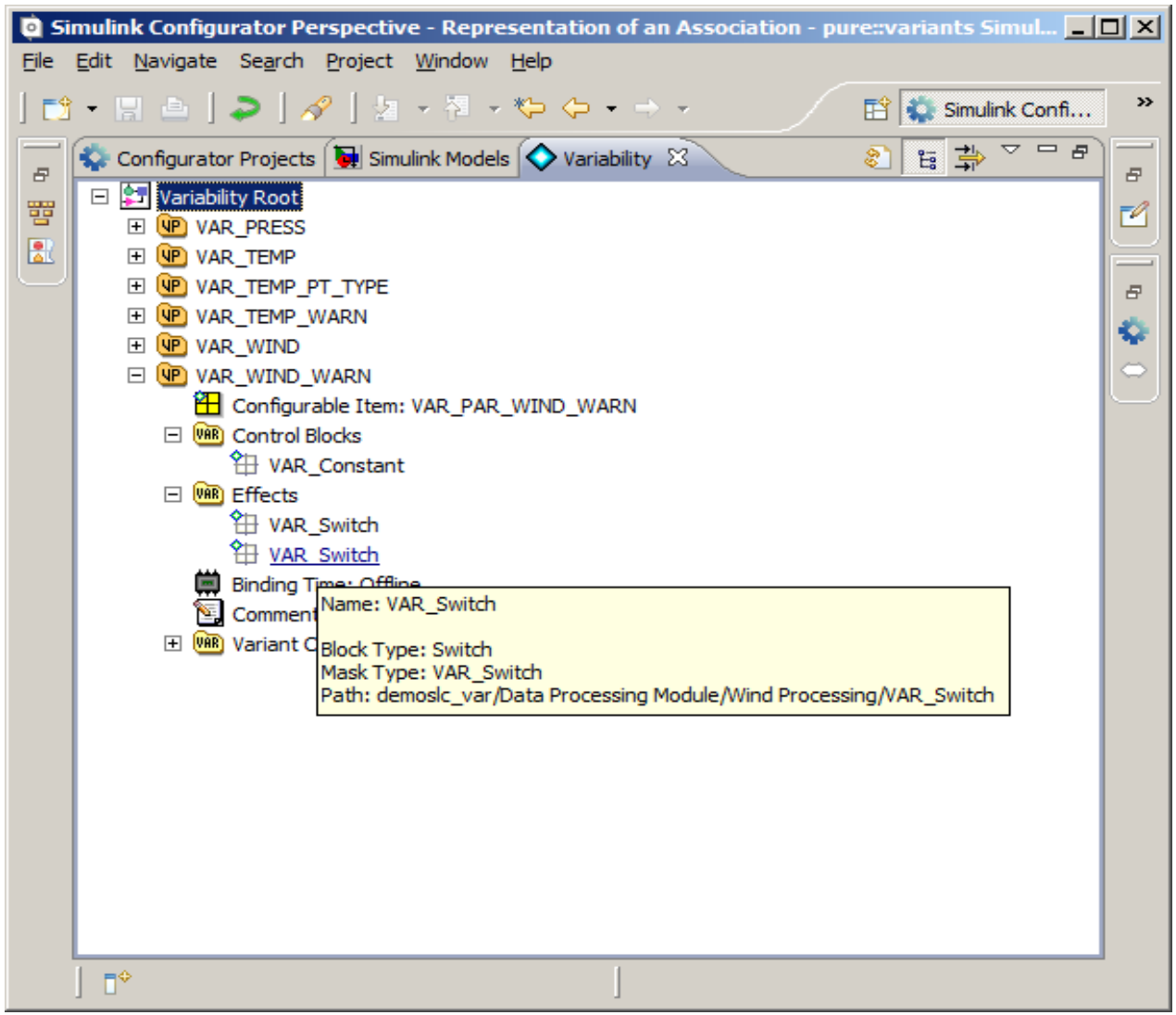




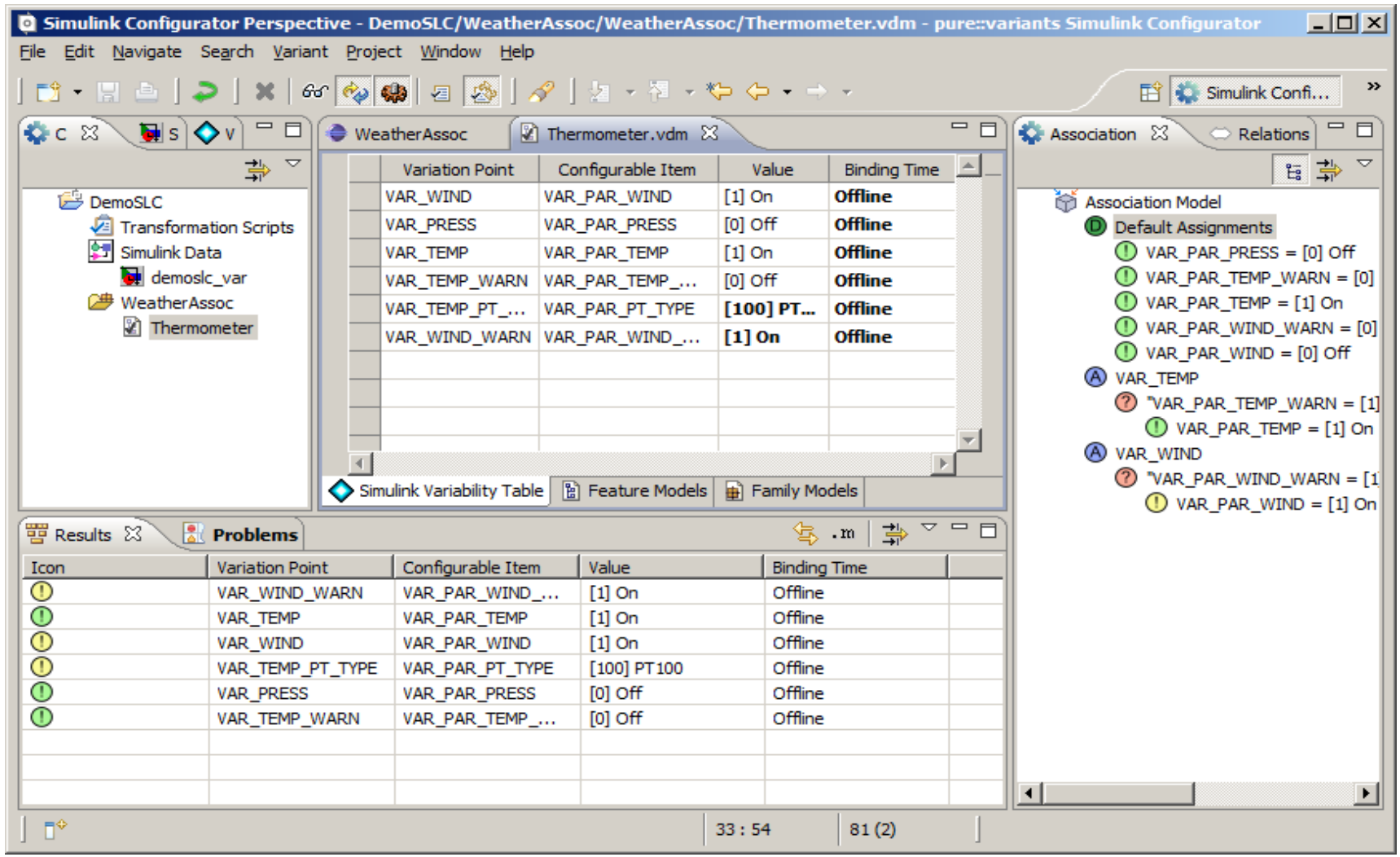
pure::variants for Simulink: Model Import



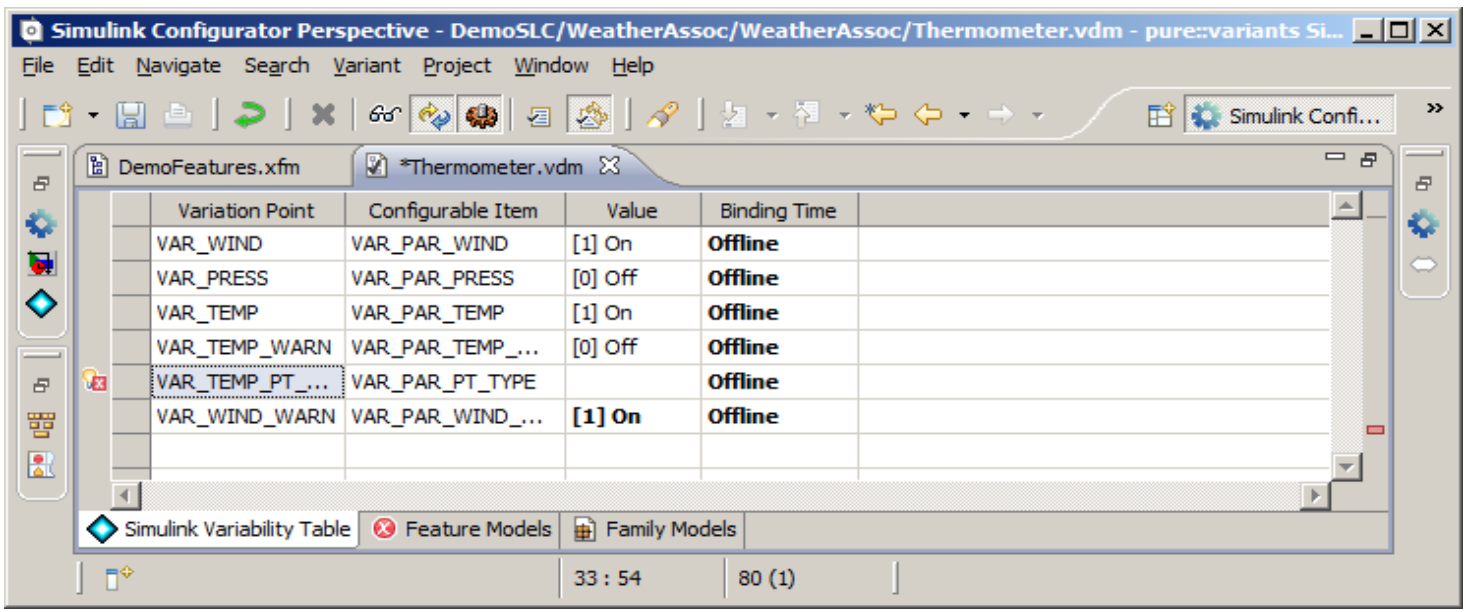
Variability View

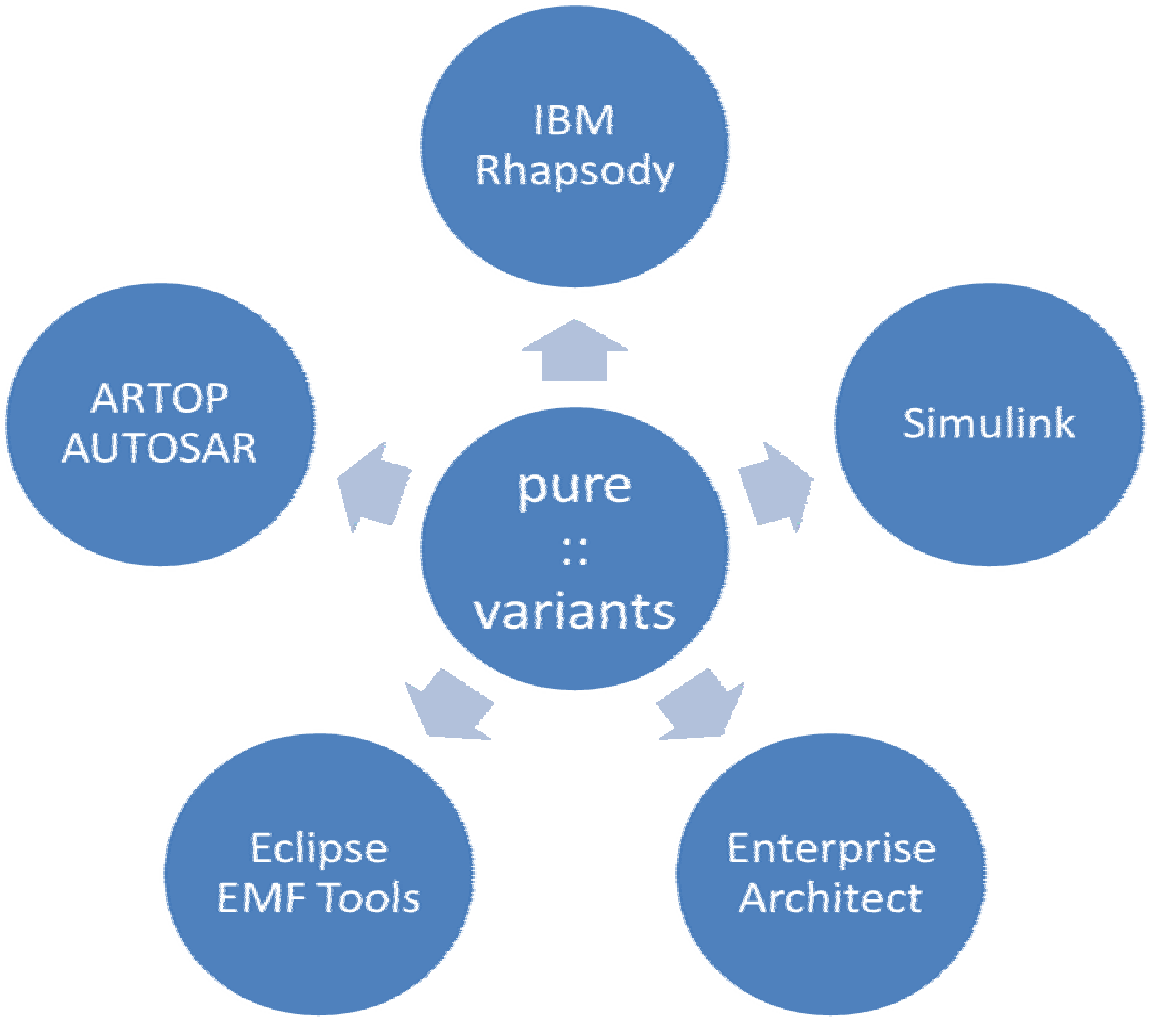


Configuration and Association Views



Variant Model





Reuse Items

Test Cases

Architecture

Estimates

Models

Code

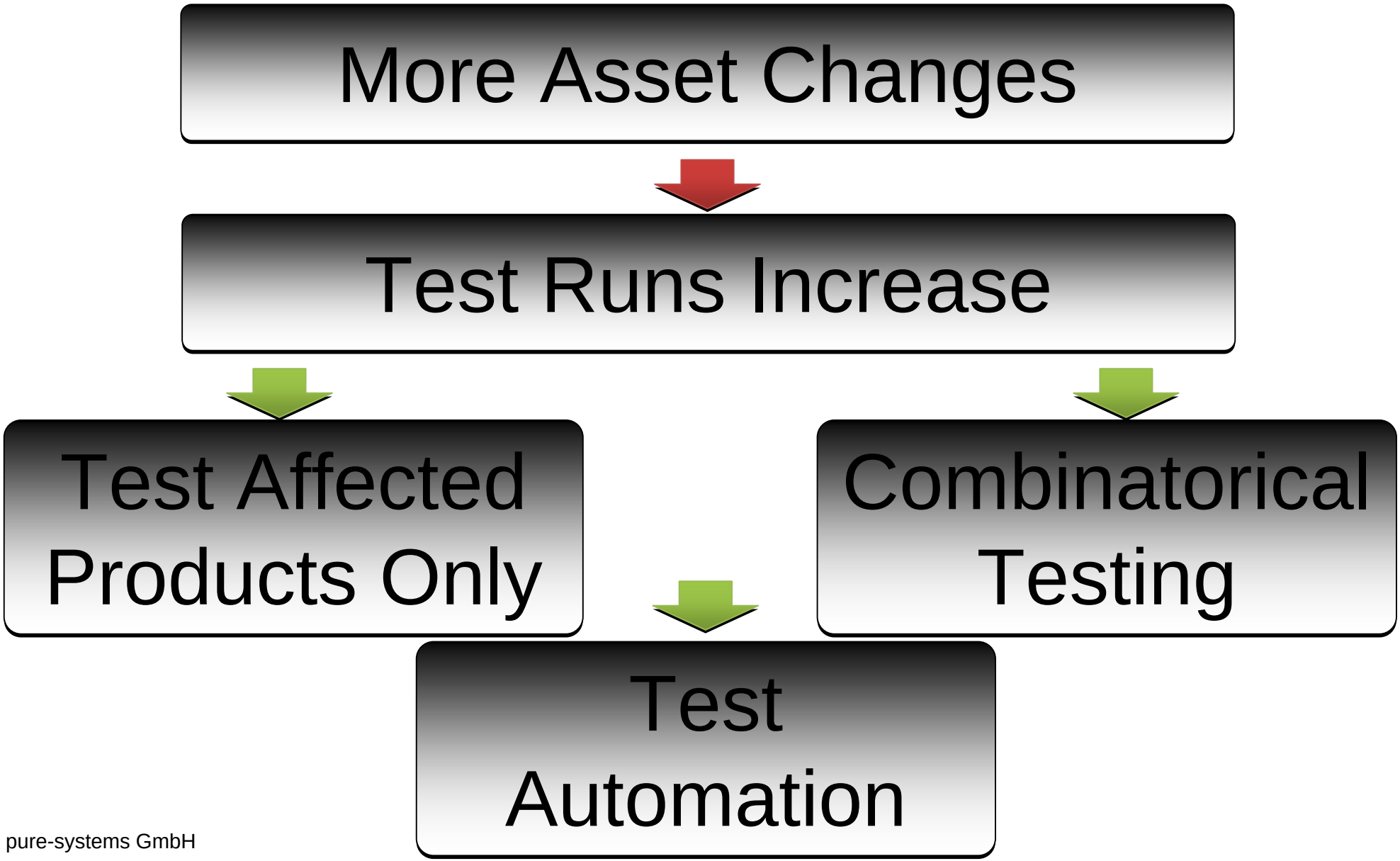
Requirements

Subsystems

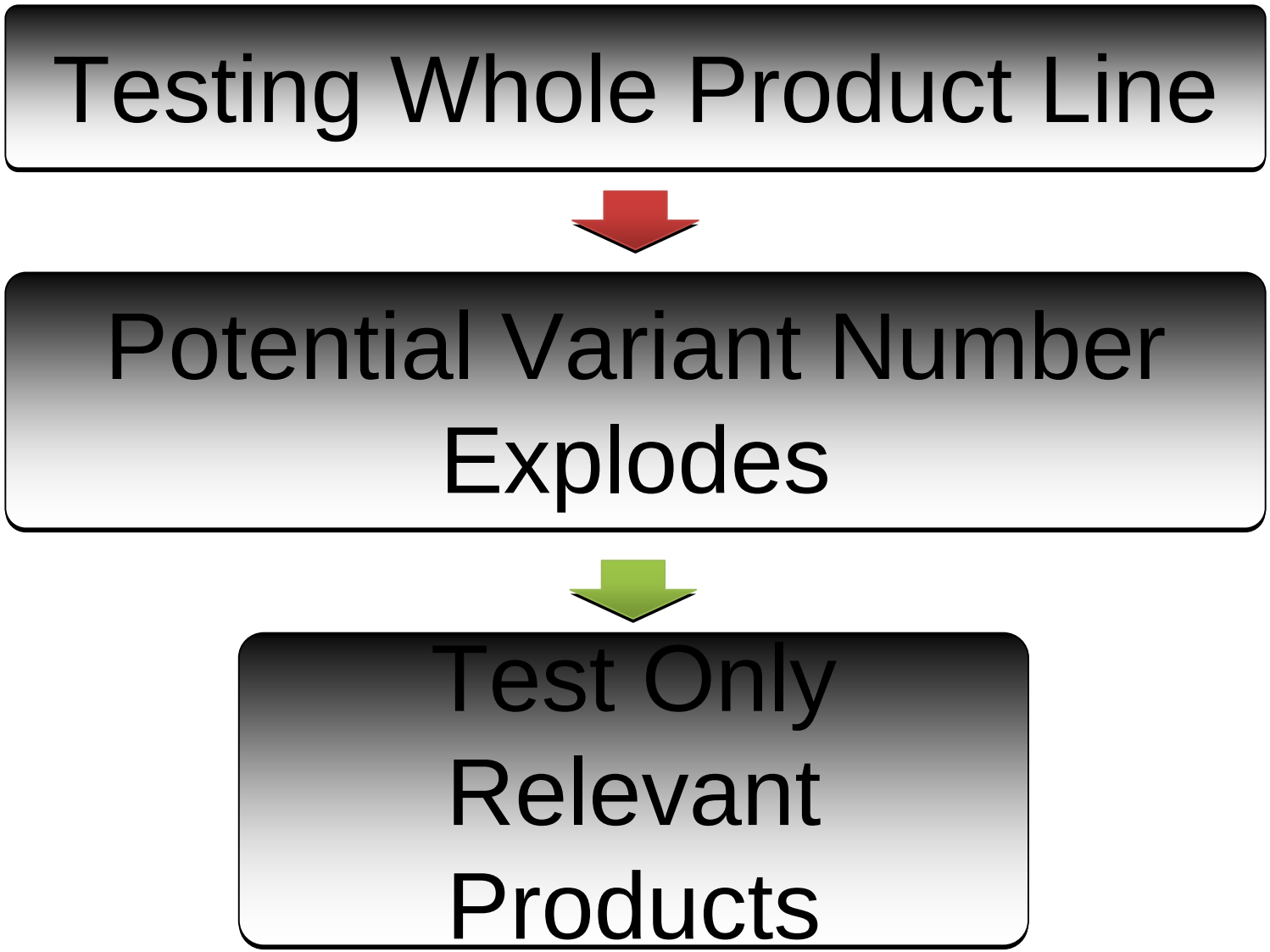
Documentation

Rules

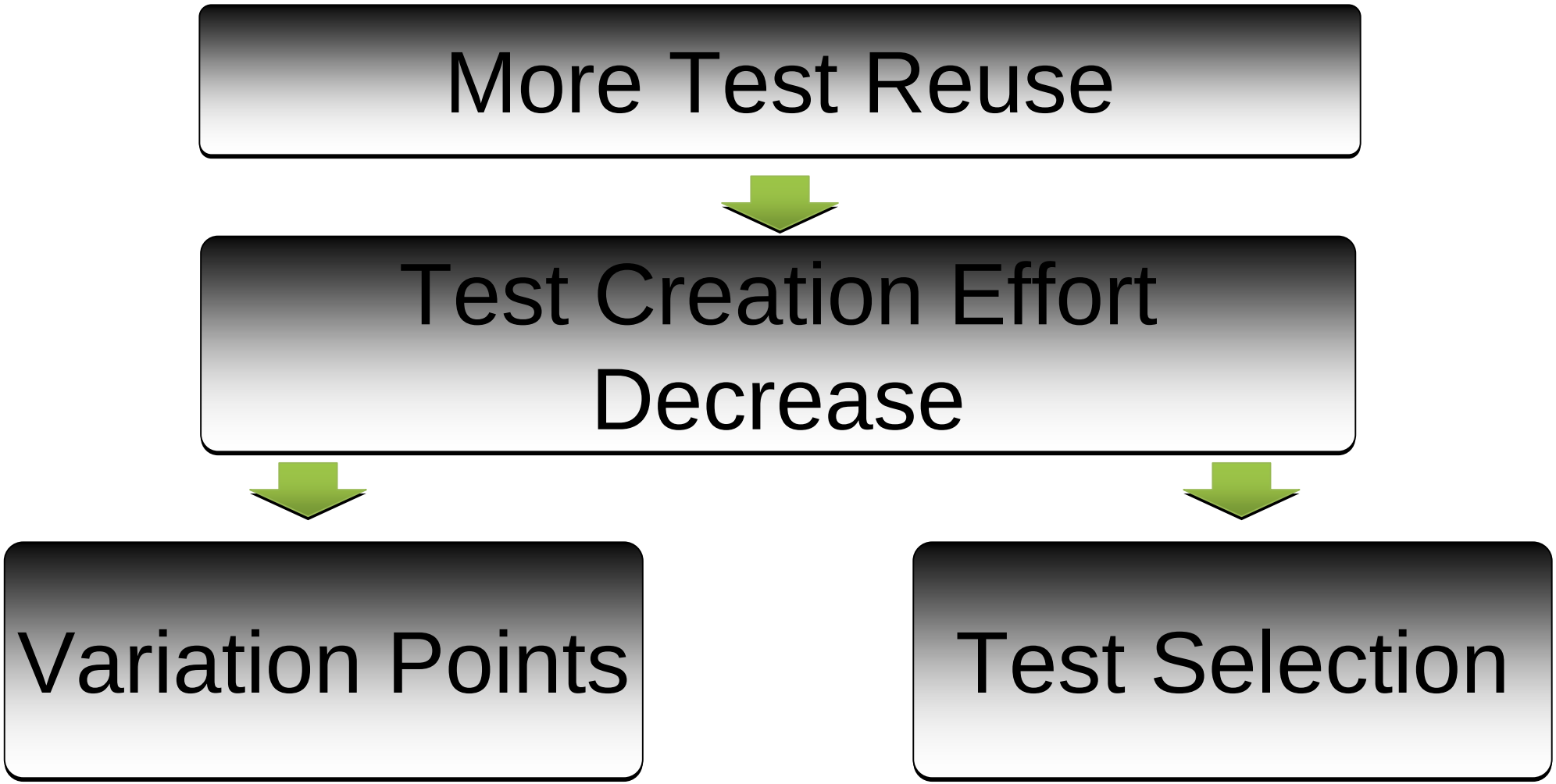
SPL Testing vs. Single Product Testing



SPL Testing vs. Single Product Testing

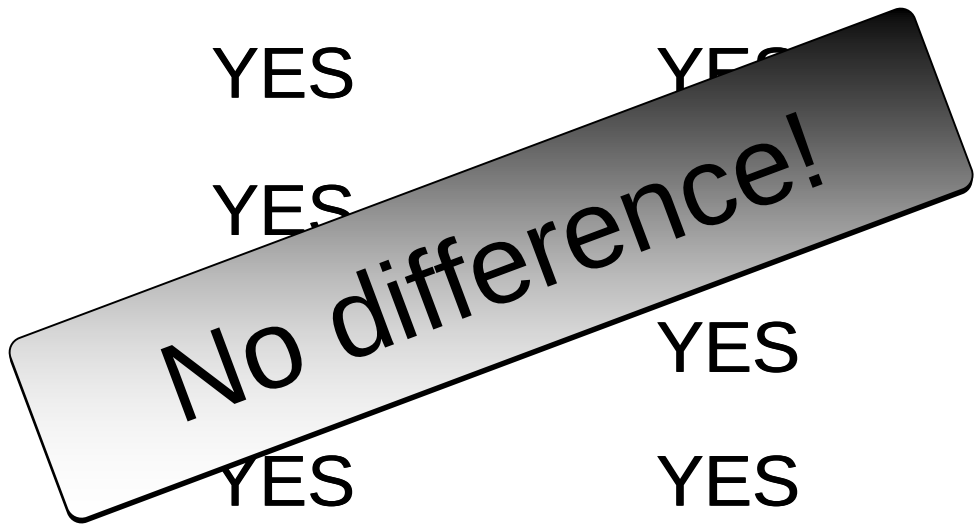


SPL Testing vs. Single Product Testing

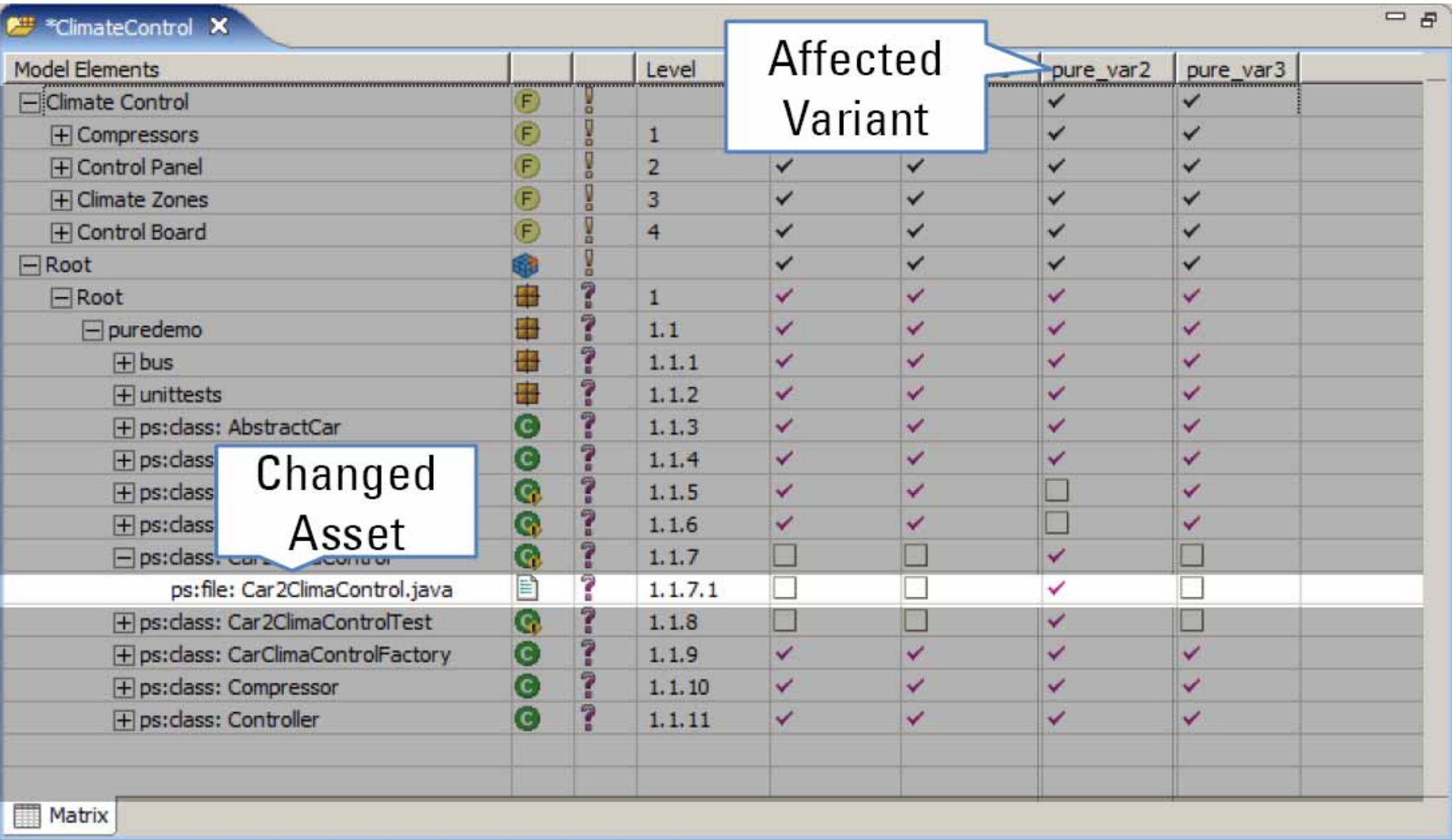


What's The Difference?

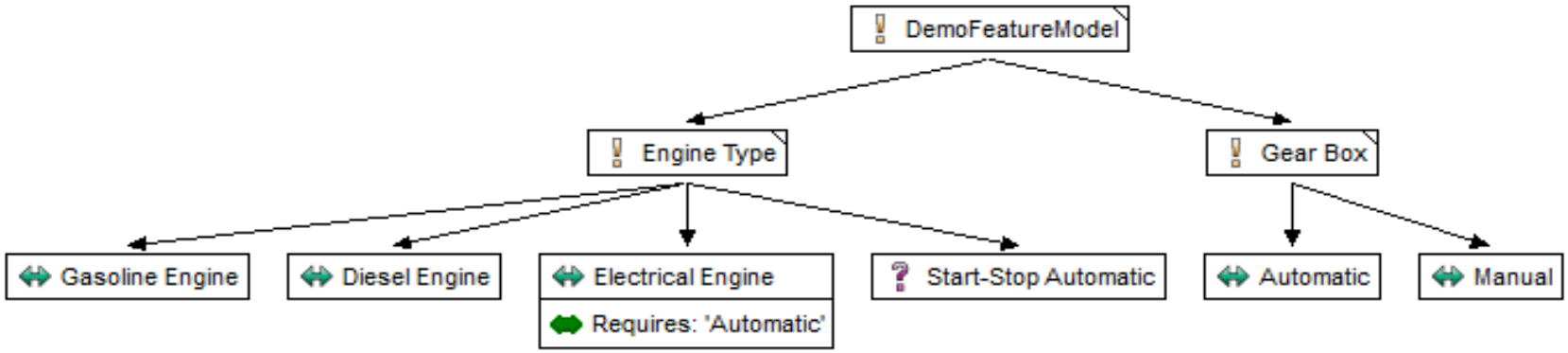
Does it have…?	PL Asset	Test Asset
Identity	YES	YES
Relations	YES	YES
Variations	YES	YES
Selection Rules	YES	YES



Test Affected Products Only



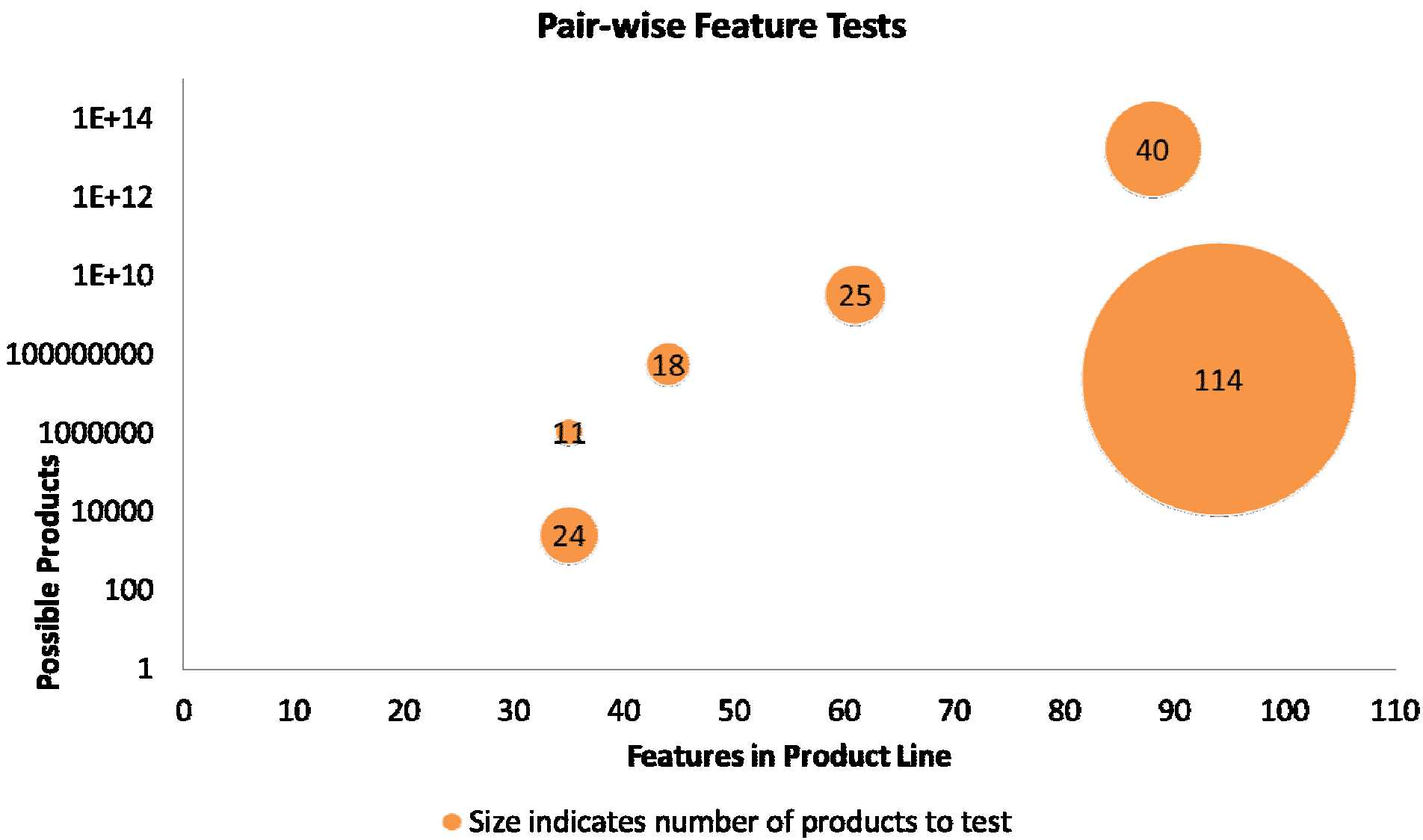
Combinatorical Testing



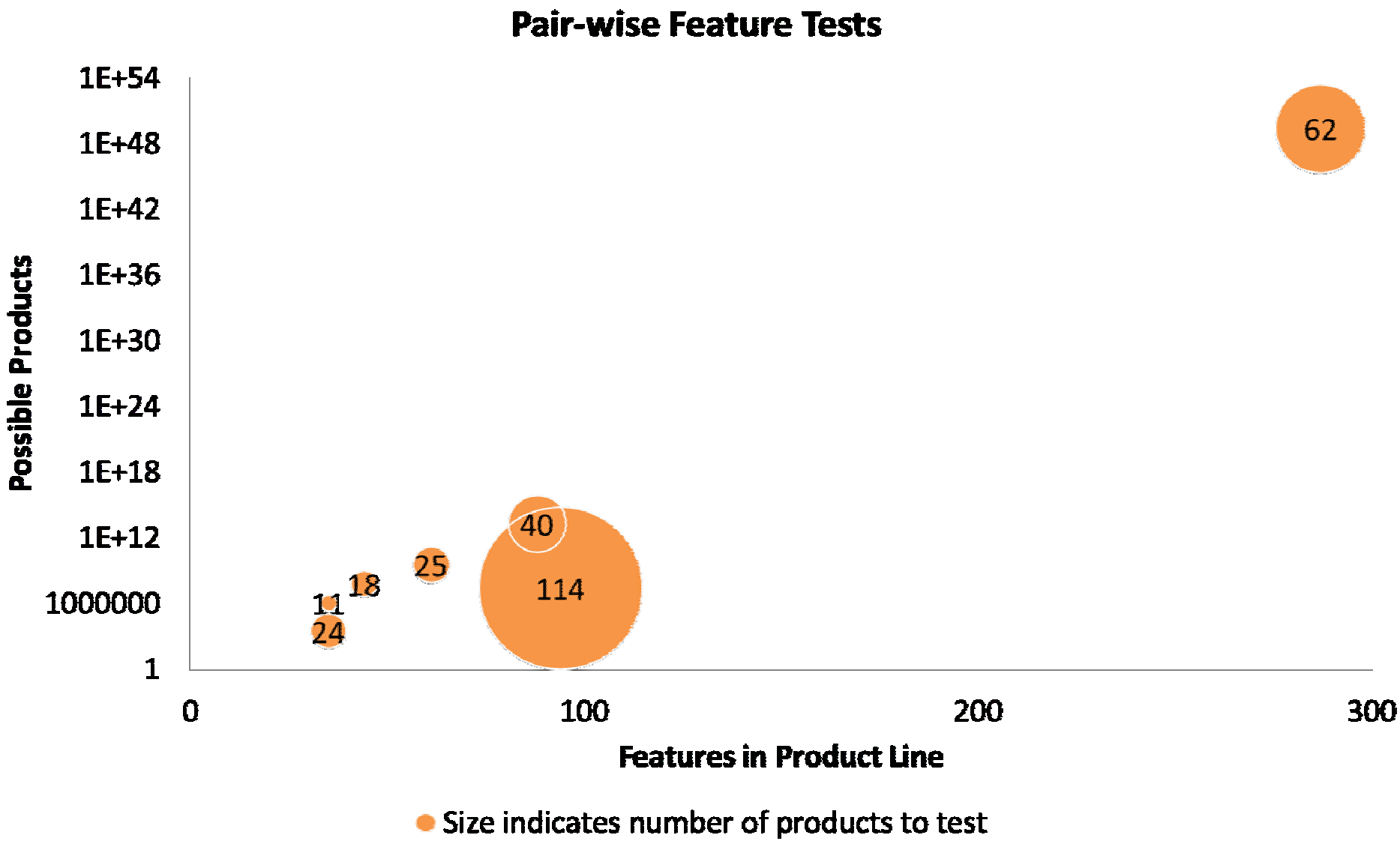
All feature pairs covered by other variants

Model Elements			L...	F...	F...	F...	F...	F...	F...	F...	F...	F...	F...
DemoFeatureModel													
Gasoline Engine			1.1	⛔	⛔	⛔	⛔	⛔	⛔	✓	✓	✓	✓
Diesel Engine			1.2	⛔	⛔	✓	✓	✓	✓	⛔	⛔	⛔	⛔
Electrical Engine			1.3	✓	✓	⛔	⛔	⛔	⛔	⛔	⛔	⛔	⛔
Start-Stop Automatic			1.4	⛔	✓	⛔	✓	⛔	✓	⛔	✓	⛔	✓
Automatic			2.1	✓	✓	⛔	⛔	✓	✓	⛔	⛔	✓	✓
Manual			2.2	⛔	⛔	✓	✓	⛔	⛔	✓	✓	⛔	⛔

Combinatorical Testing



Combinatorical Testing

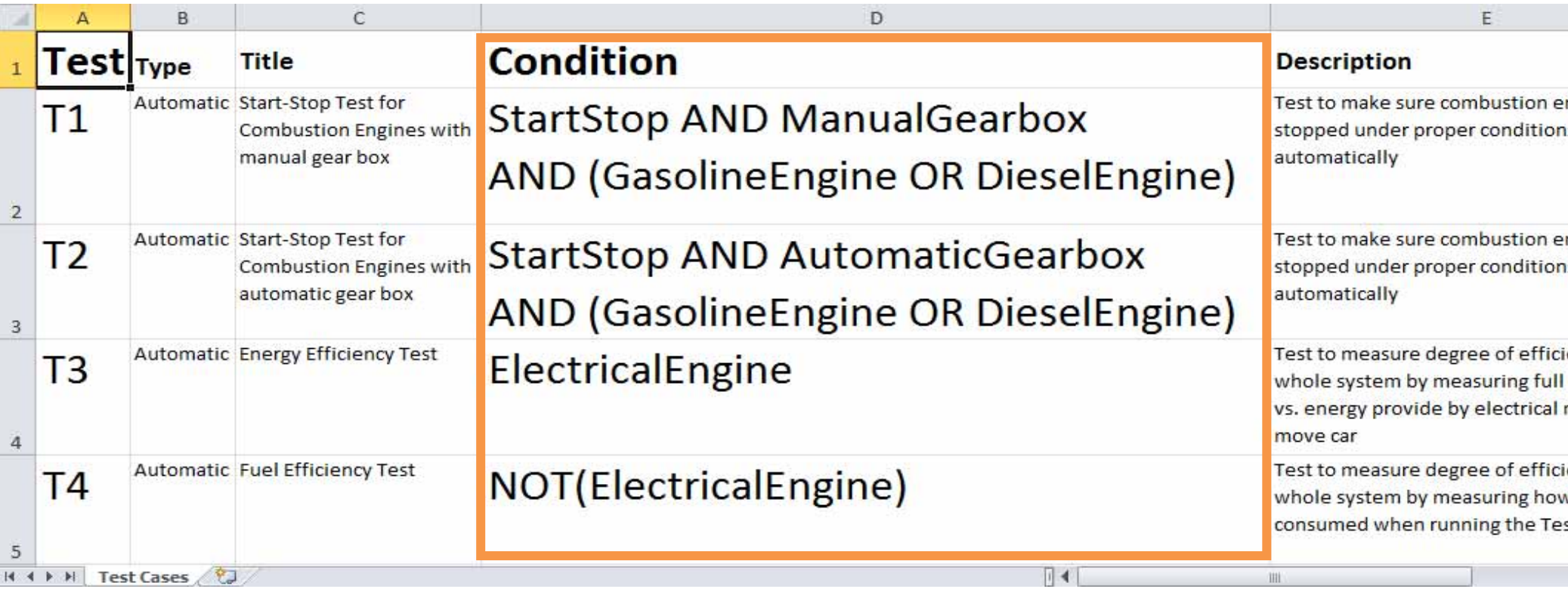


Test Selection

	A	B	C	E	F
1	**Test**	**Type**	**Title**	**Description**	**Execution Steps**
2	T1	Automatic	Start-Stop Test for Combustion Engines with manual gear box	Test to make sure combustion engine is stopped under proper conditions automatically	1. Accelerate to 50 km/h 2. Reduce speed to zero by braking 3. Press clutch 4. Move to neutral gear
3	T2	Automatic	Start-Stop Test for Combustion Engines with automatic gear box	Test to make sure combustion engine is stopped under proper conditions automatically	1. Accelerate to 50 km/h 2. Reduce speed to zero by braking
4	T3	Automatic	Energy Efficiency Test	Test to measure degree of efficiency in the whole system by measuring full charge energy vs. energy provide by electrical motors to move car	1. Run Drive Cycle 10 until battery charge is minimal 2. Charge battery and measure consumed energy
5	T4	Automatic	Fuel Efficiency Test	Test to measure degree of efficiency in the whole system by measuring how much fuel is consumed when running the Test Drive Cycle	1. Fill tank to maximum 2. Run Drive Cycle 20 3. Measure used fuel by filling up to maximum again.

Test Cases

Test Selection

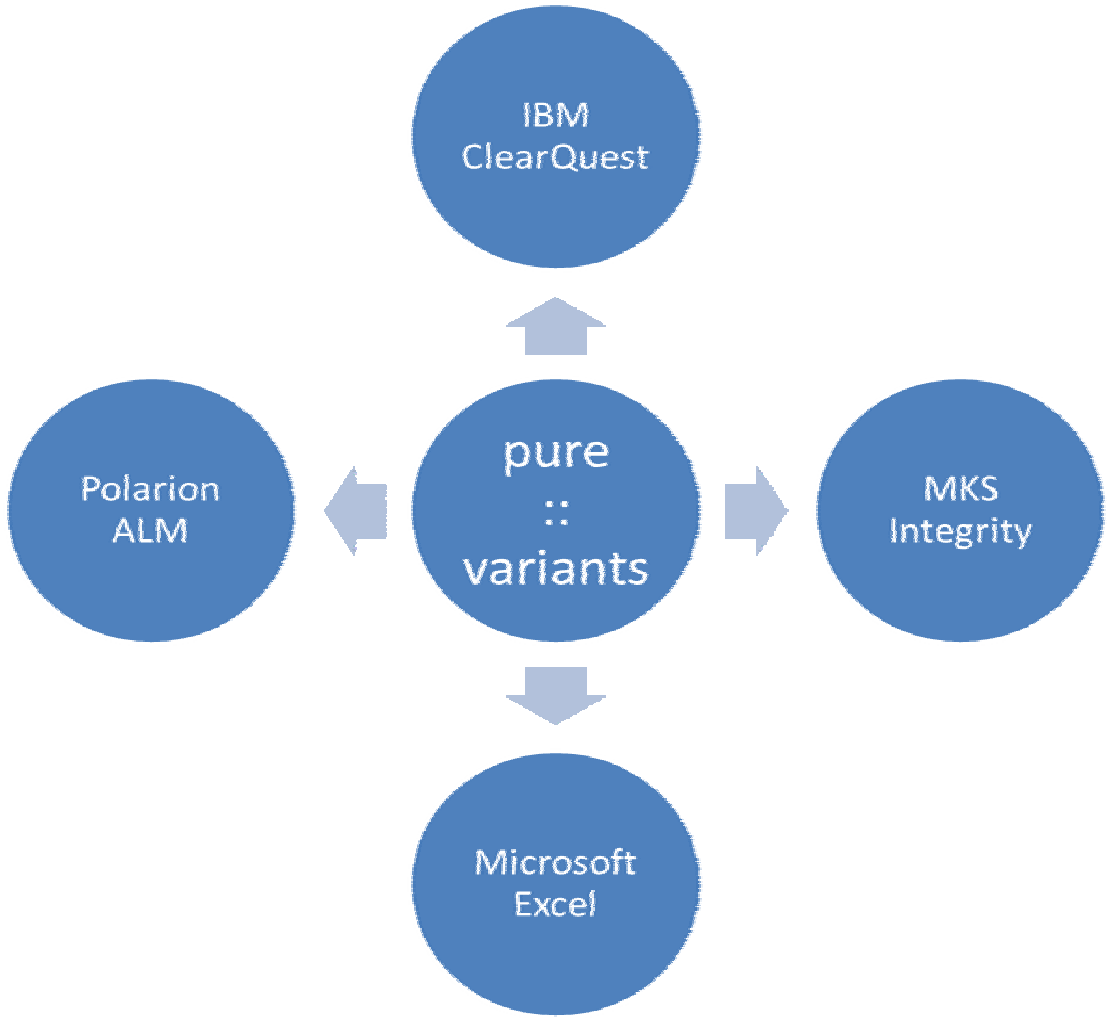


	A	B	C	D	E
1	**Test**	**Type**	**Title**	**Condition**	**Description**
2	T1	Automatic	Start-Stop Test for Combustion Engines with manual gear box	StartStop AND ManualGearbox AND (GasolineEngine OR DieselEngine)	Test to make sure combustion en stopped under proper condition automatically
3	T2	Automatic	Start-Stop Test for Combustion Engines with automatic gear box	StartStop AND AutomaticGearbox AND (GasolineEngine OR DieselEngine)	Test to make sure combustion en stopped under proper condition automatically
4	T3	Automatic	Energy Efficiency Test	ElectricalEngine	Test to measure degree of effici whole system by measuring full vs. energy provide by electrical move car
5	T4	Automatic	Fuel Efficiency Test	NOT(ElectricalEngine)	Test to measure degree of effici whole system by measuring how consumed when running the Tes

Test Cases

Test Selection

Model Elements			L...	F...	F...	F...	F...	F...	F...	F...	F...	F...	F...
DemoFeatureModel													
Gasoline Engine			1.1	⛔	⛔	⛔	⛔	⛔	⛔	☑	☑	☑	☑
Diesel Engine			1.2	⛔	⛔	☑	☑	☑	☑	⛔	⛔	⛔	⛔
Electrical Engine			1.3	☑	☑	⛔	⛔	⛔	⛔	⛔	⛔	⛔	⛔
Start-Stop Automatic		?	1.4	⛔	☑	⛔	☑	⛔	☑	⛔	☑	⛔	☑
Automatic			2.1	☑	☑	⛔	⛔	☑	☑	⛔	⛔	☑	☑
Manual			2.2	⛔	⛔	☑	☑	⛔	⛔	☑	☑	⛔	⛔
TestPlan													
TestCase_T1		?	1	☒	☒	☒	✓	☒	☒	☒	✓	☒	☒
TestCase_T2		?	2	☒	☒	☒	☒	☒	✓	☒	☒	☒	✓
TestCase_T3		?	3	✓	✓	☒	☒	☒	☒	☒	☒	☒	☒
TestCase_T4		?	4	☒	☒	✓	✓	✓	✓	✓	✓	✓	✓



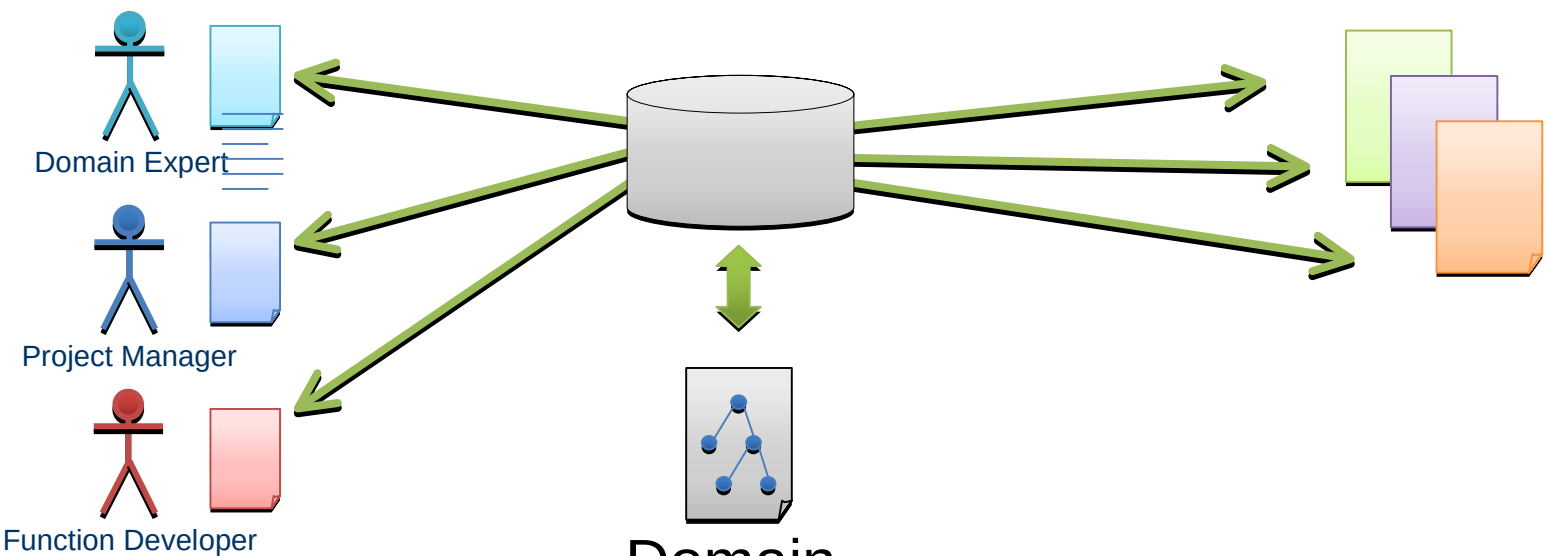
Running a Product Line – Change and Configuration Management

Variant Management and Development Artifacts

Stakeholder-Specific Views

Central Development Artifact Databases

Supplementary Documents



Domain Models

Challenges of PL: Change Management

Change Impact Increase

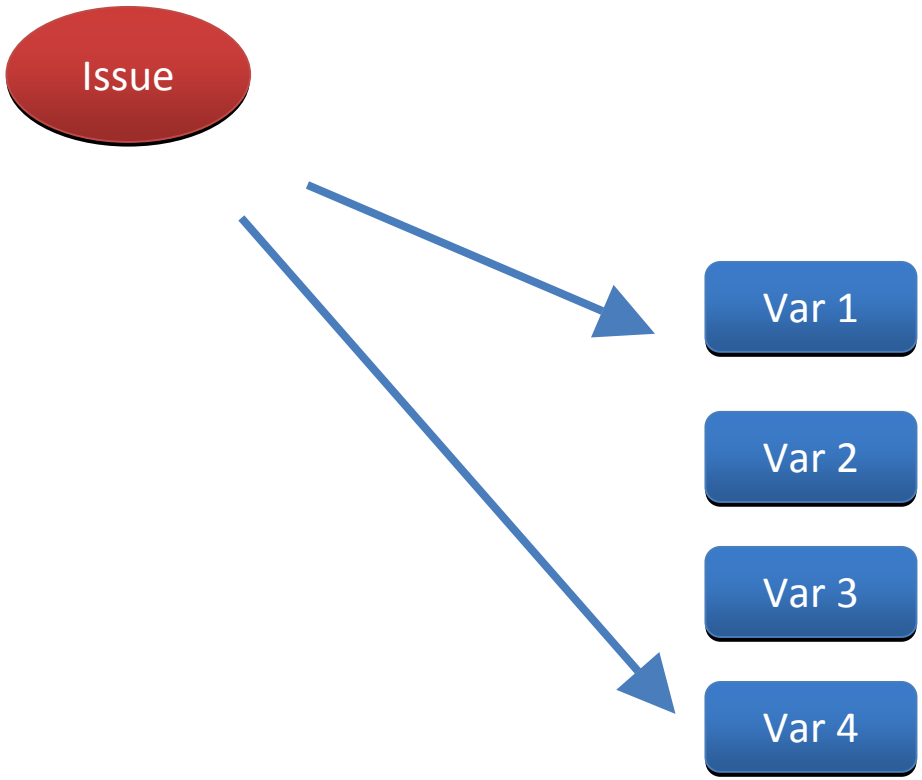
Number of Changes in Core Assets

Changes in Products vs. Changes in Core Assets

Controlling Speed of Change

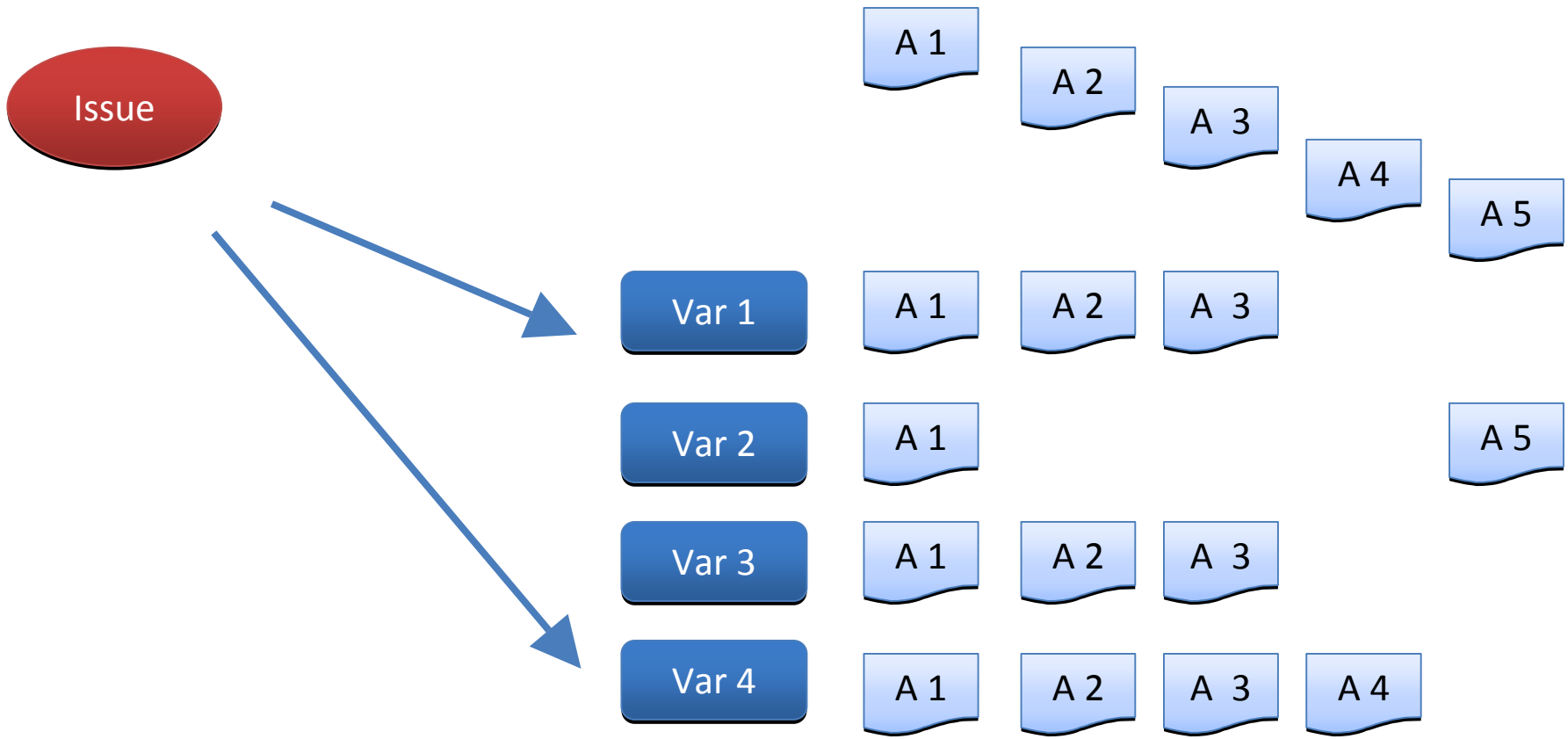
Coordination between many more stakeholders

Variants and Change Management



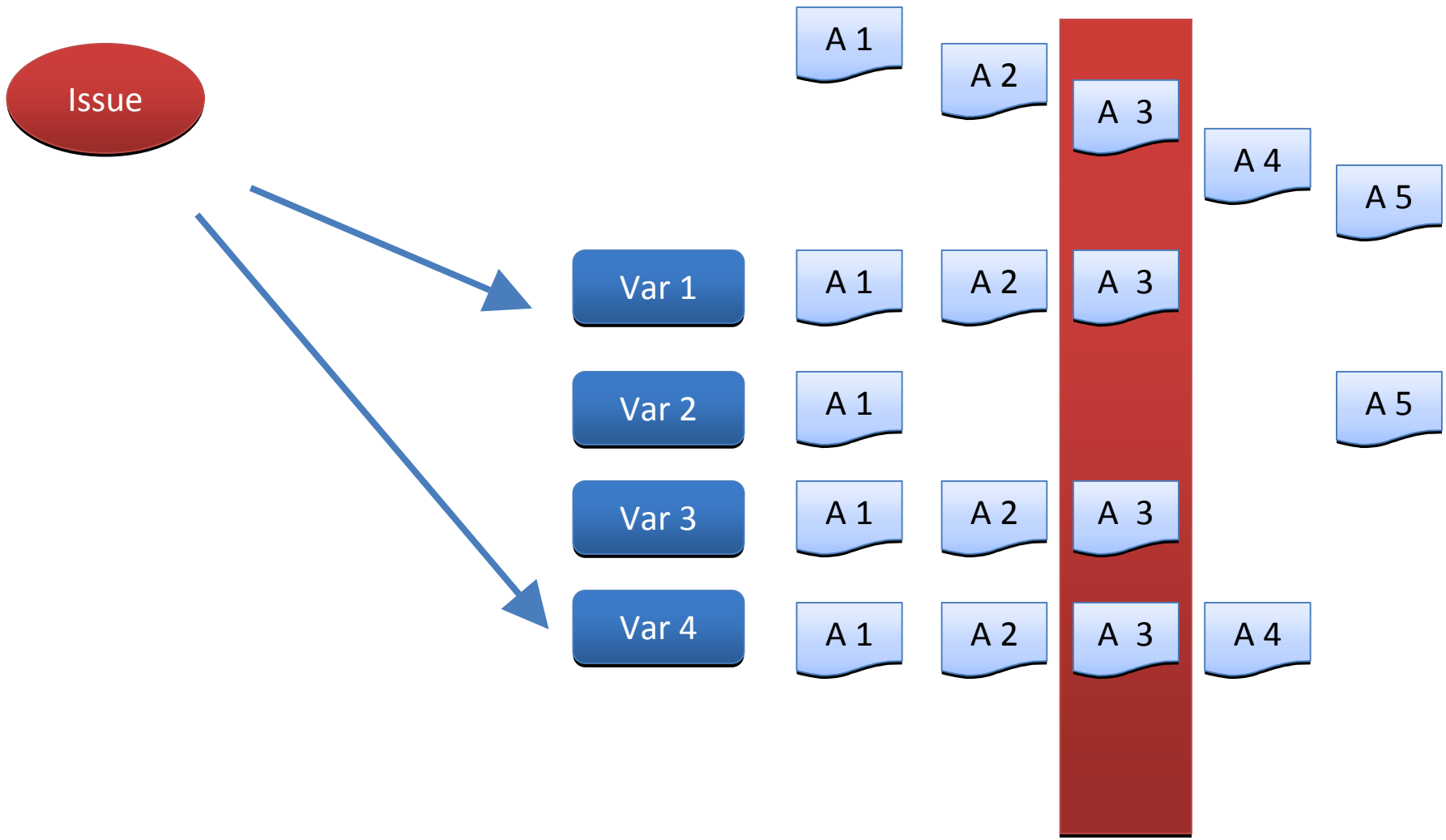
Issue to Products

Variants and Change Management

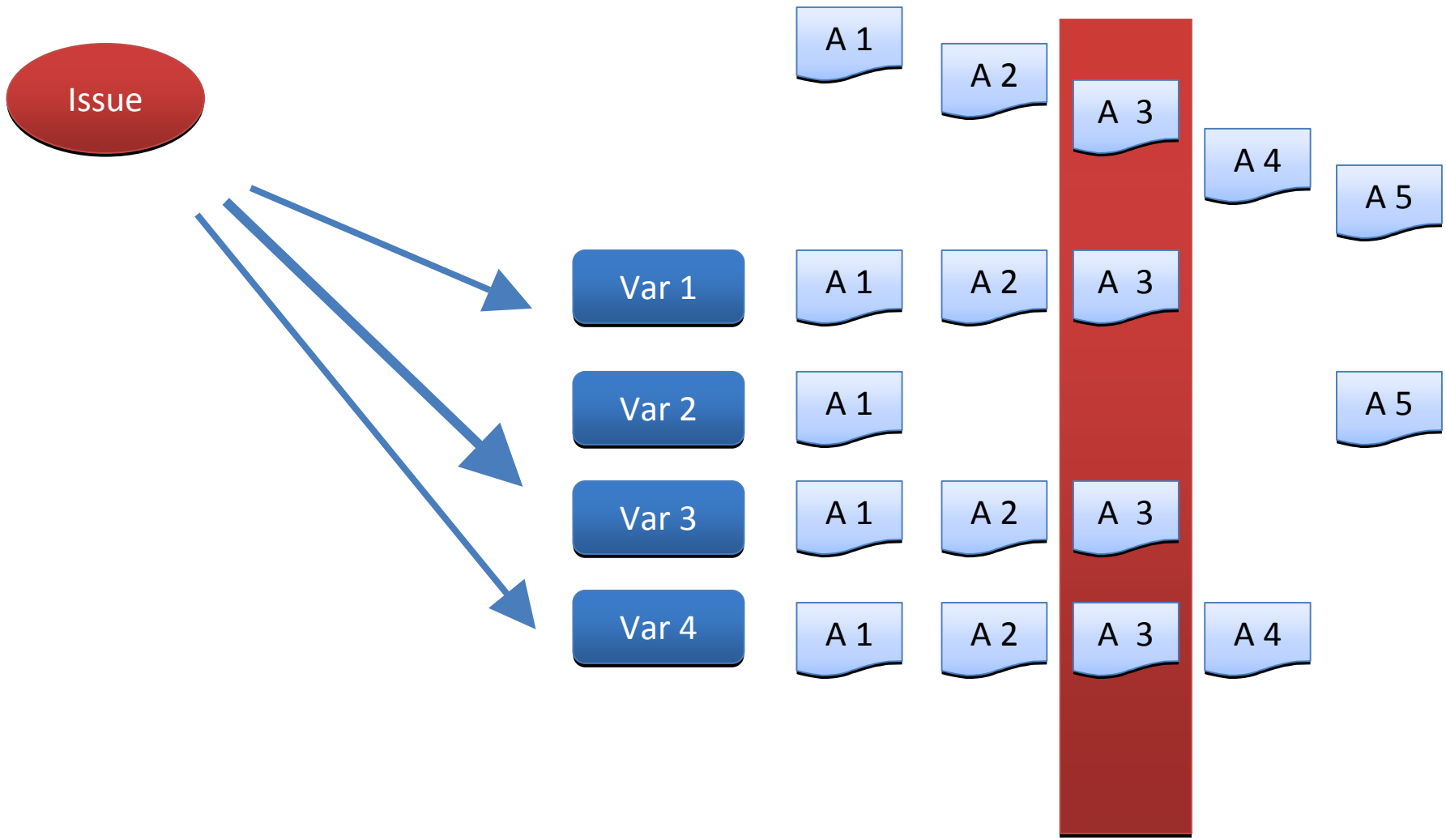


Products to Assets

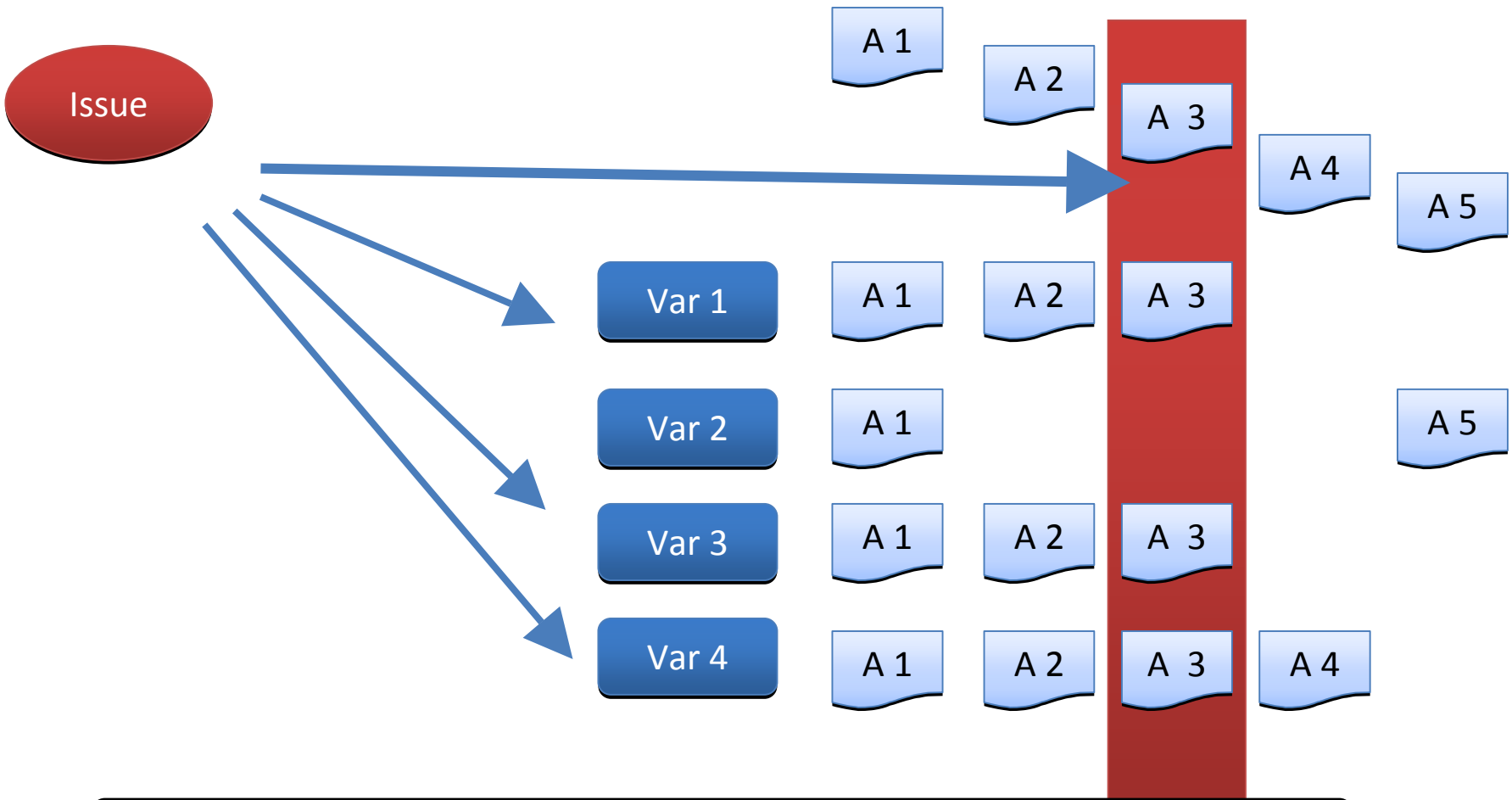
Variants and Change Management



Variants and Change Management

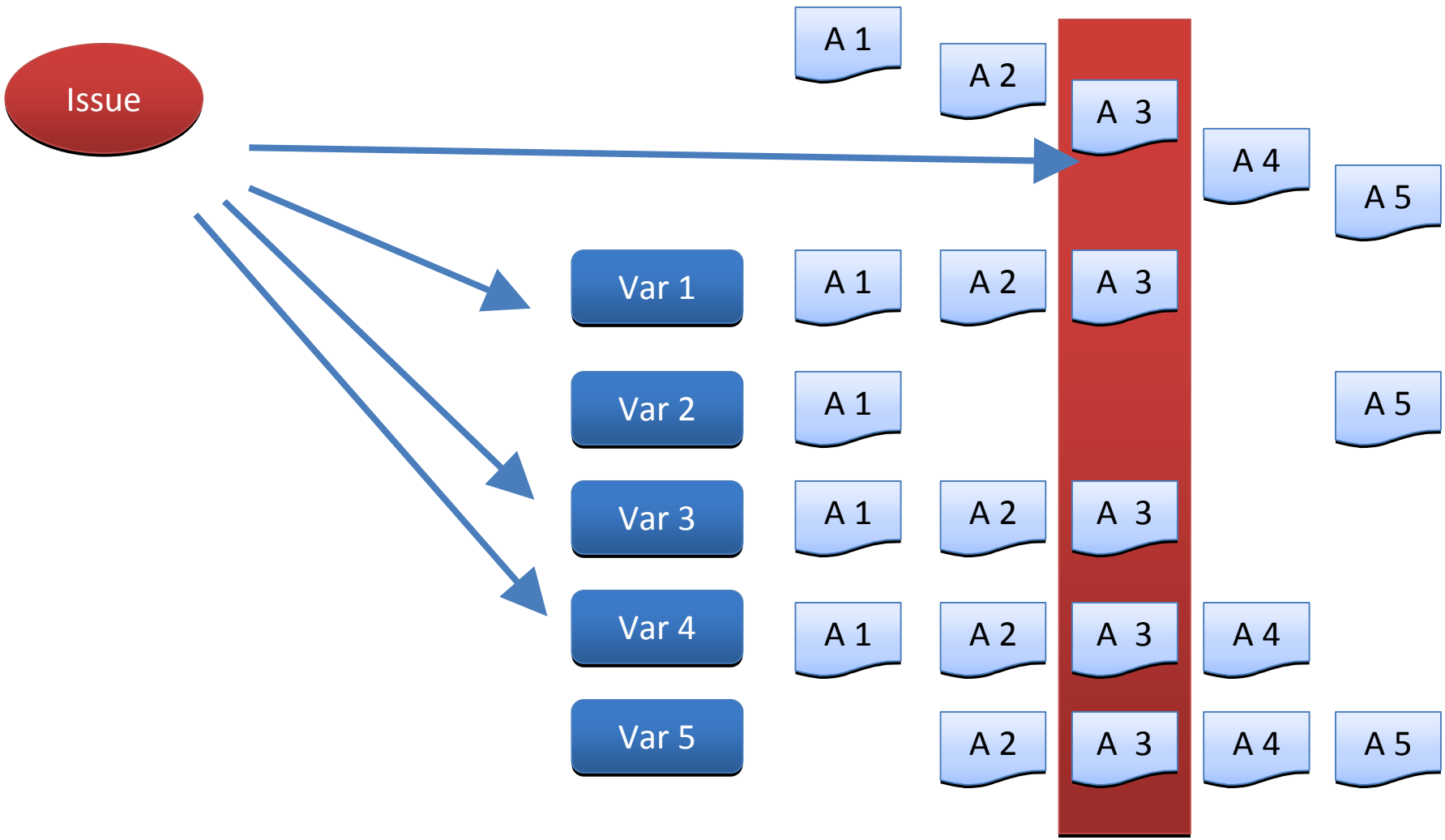


Variants and Change Management

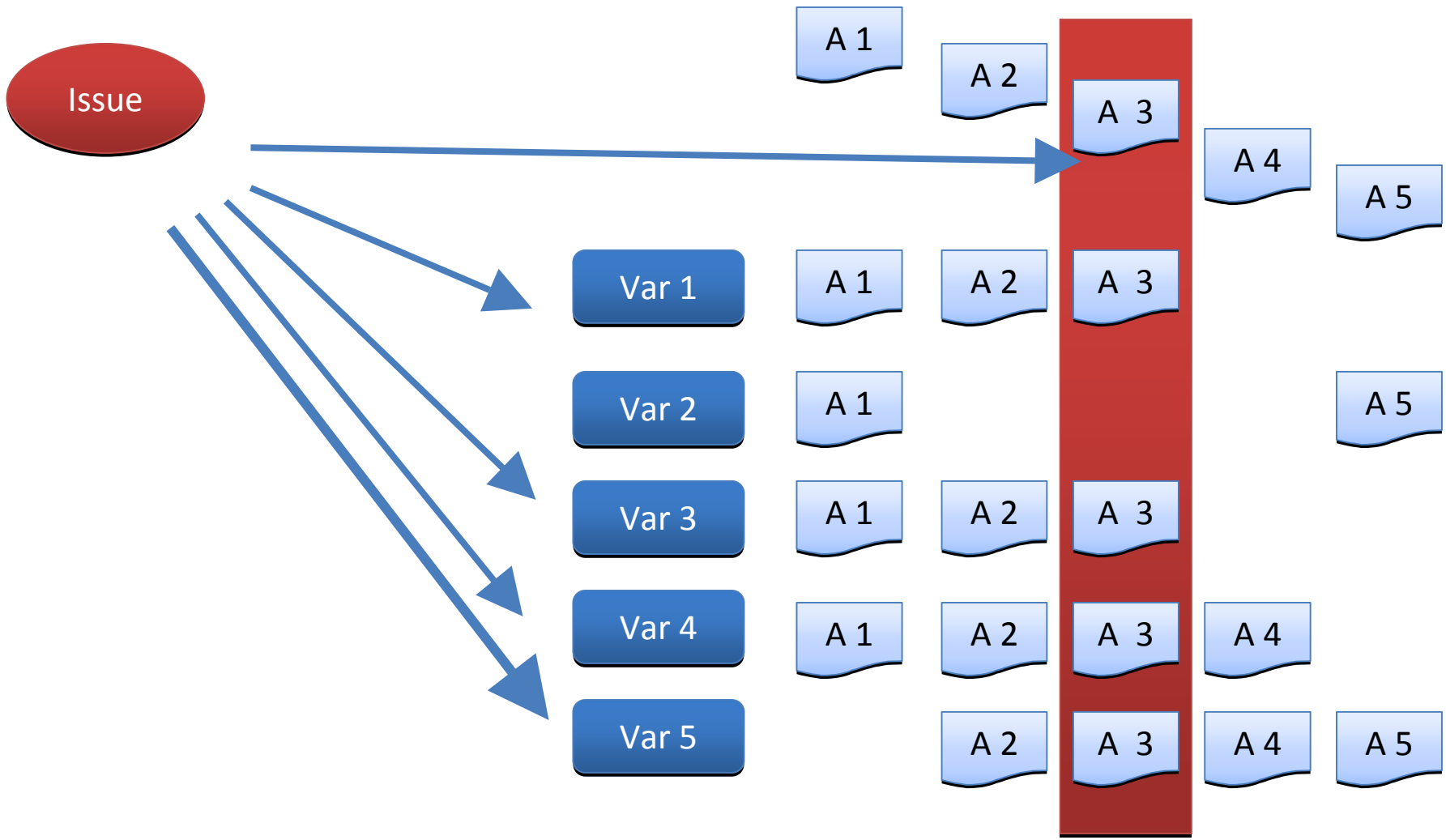


Issues to Assets

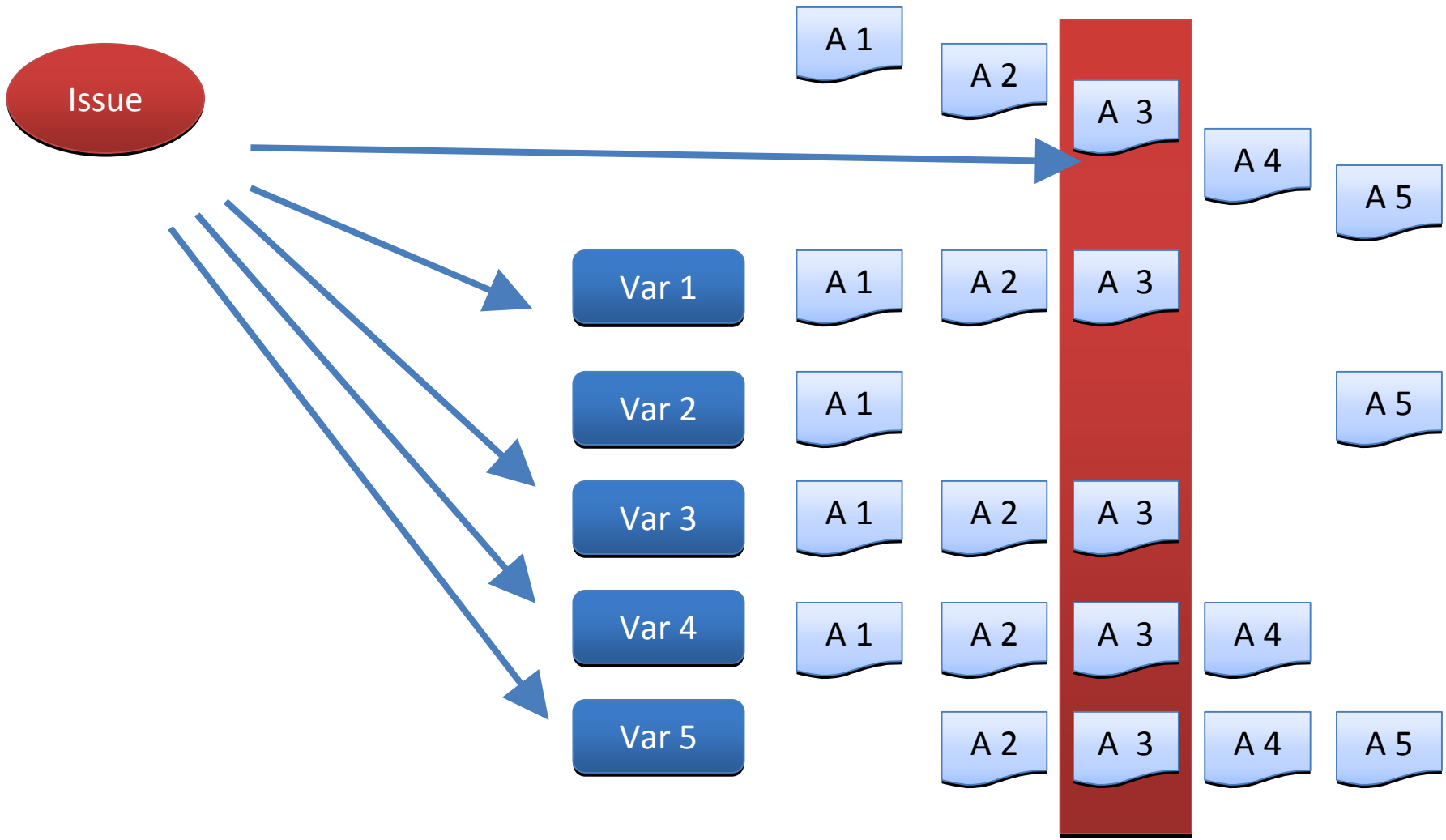
Variants and Change Management



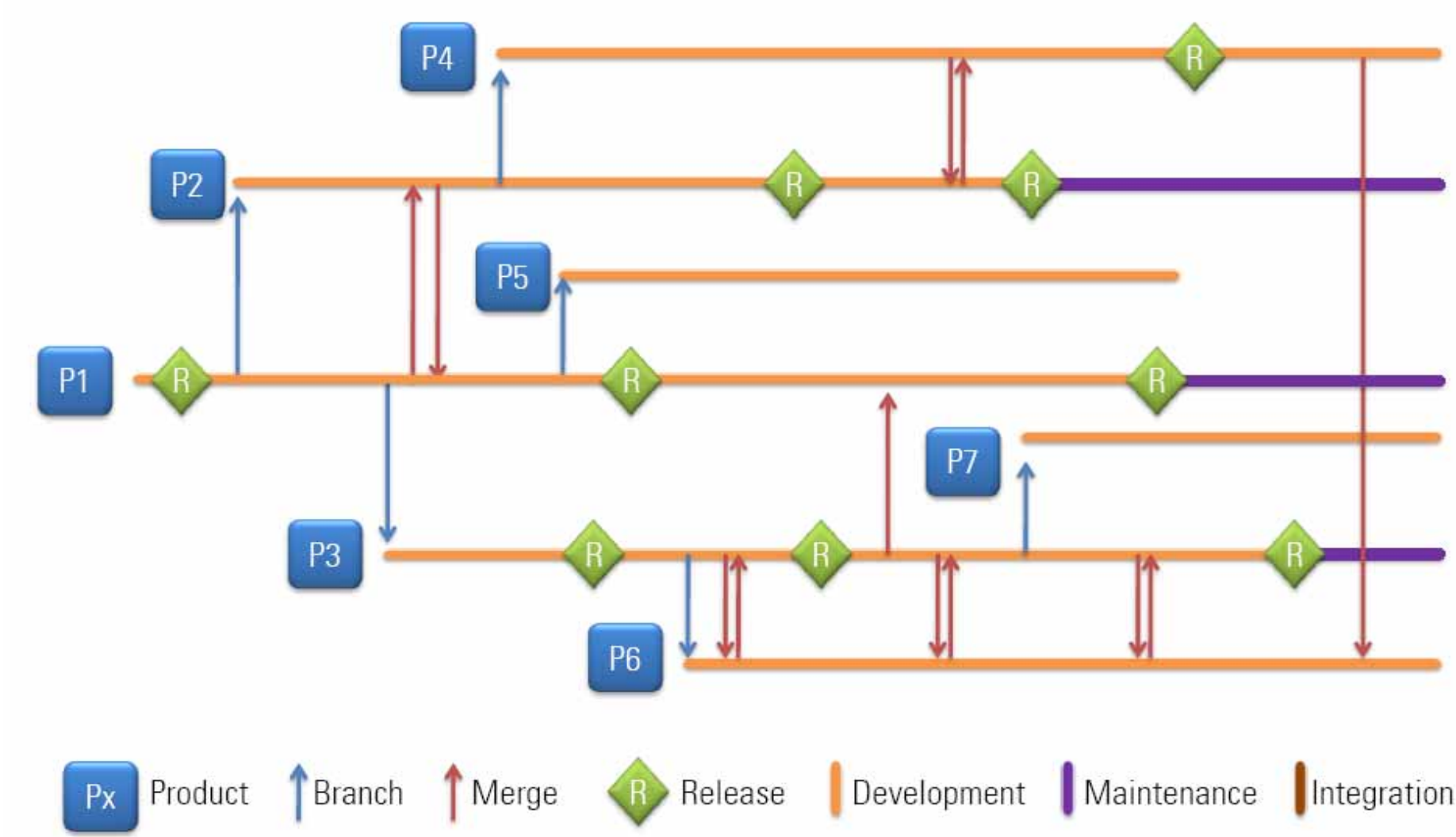
Variants and Change Management



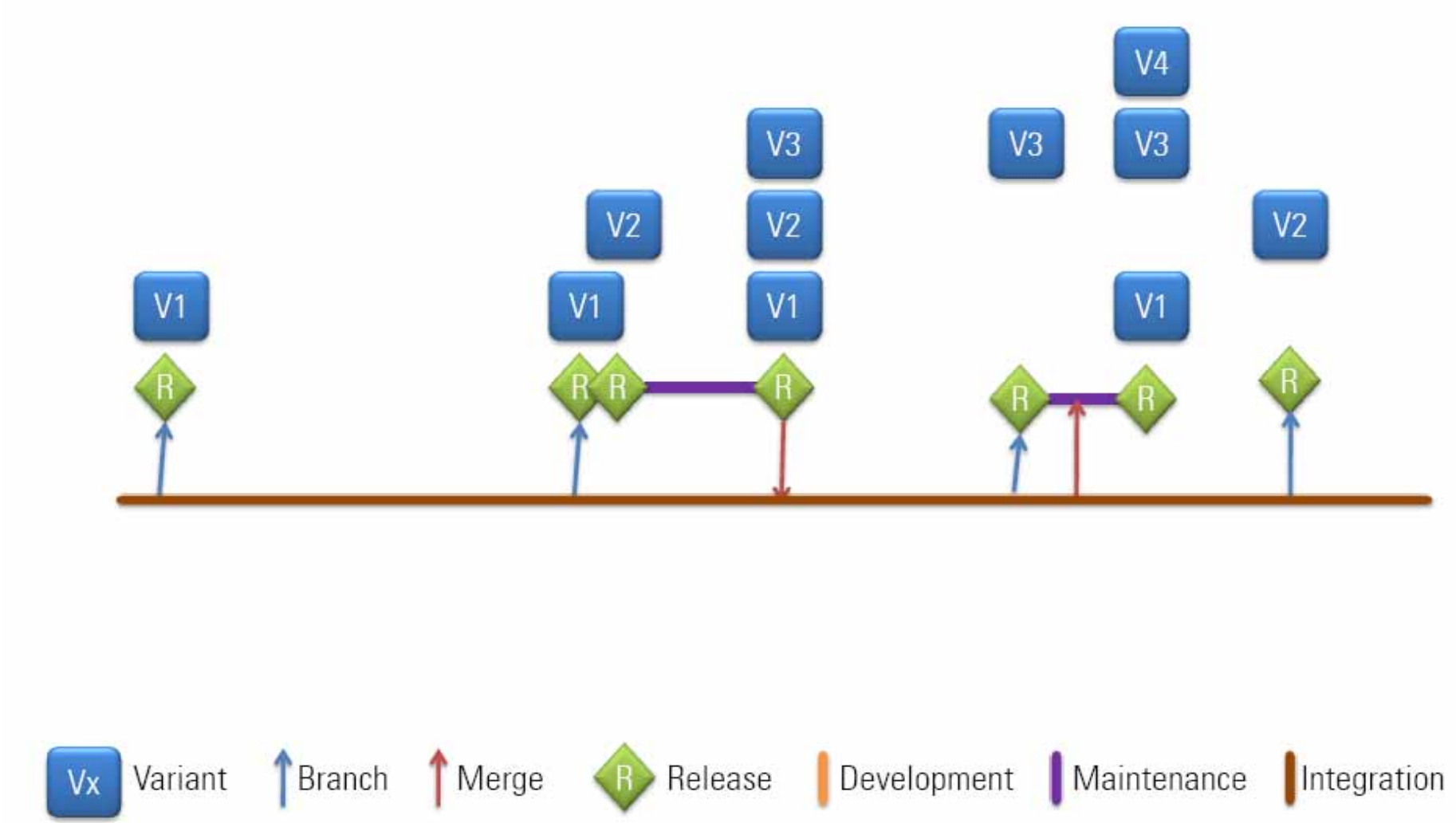
Variants and Change Management



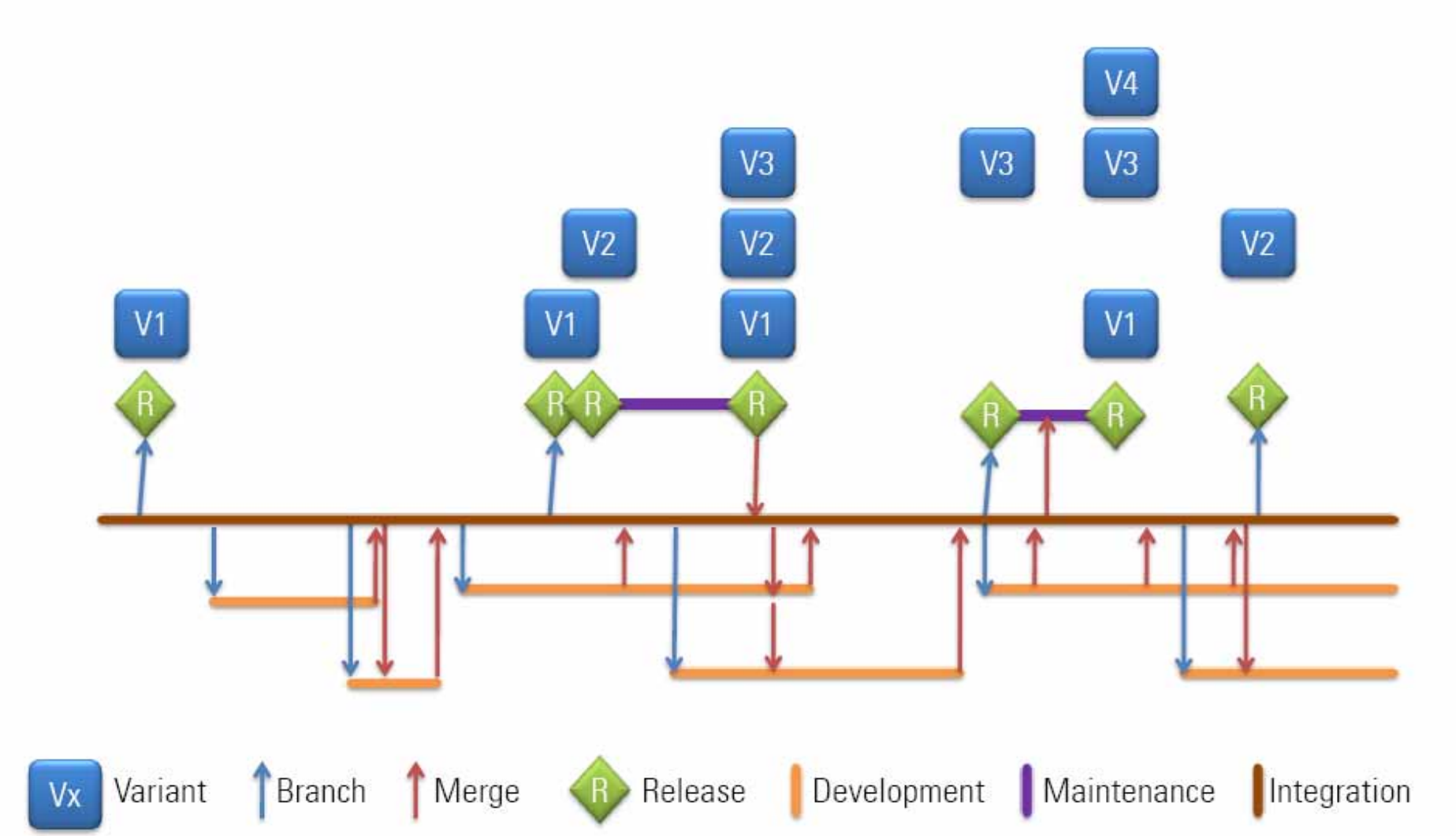
Variants and Configuration Management
Clone and Own with Branches

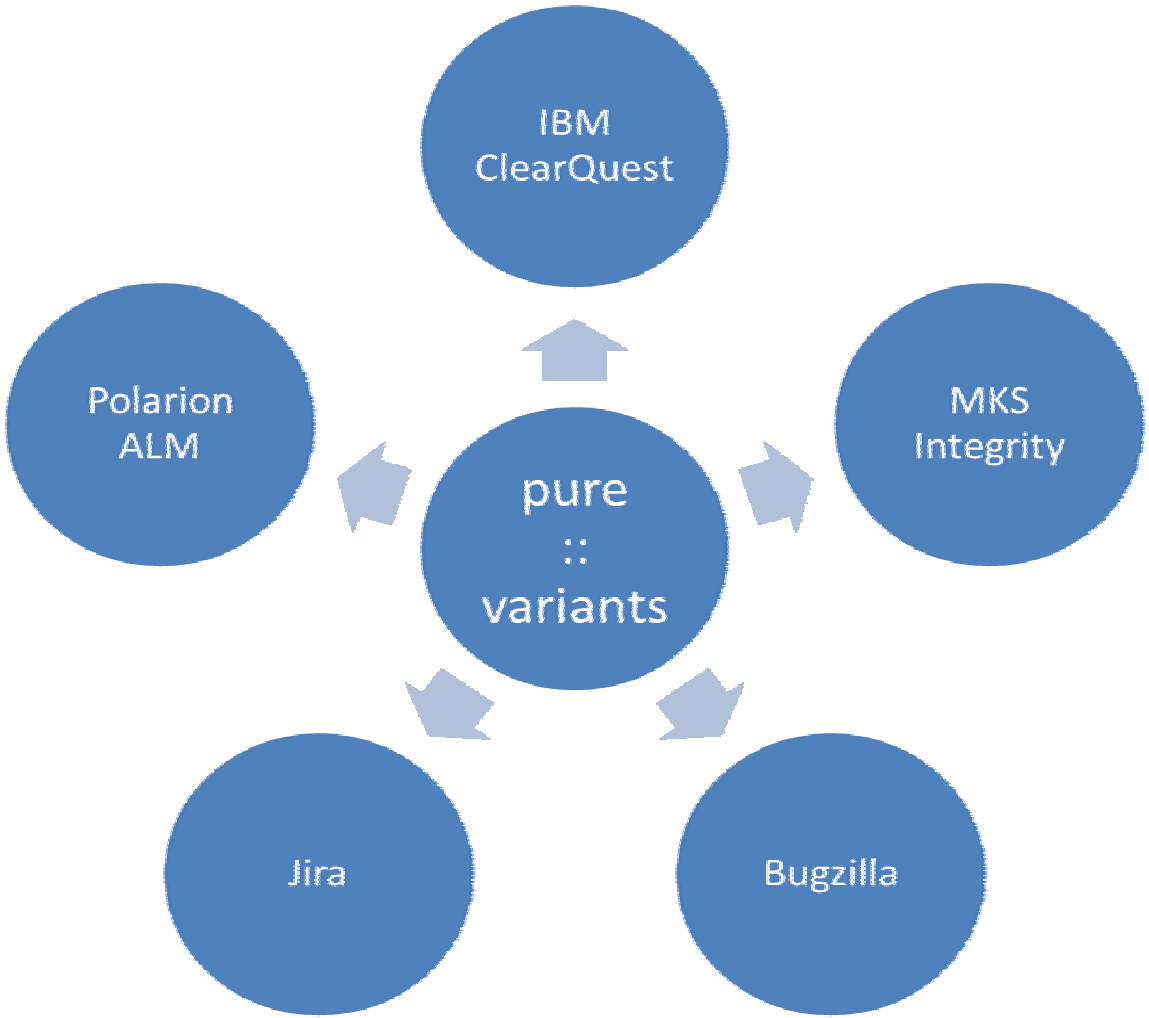


Variants and Configuration Management



Variants and Configuration Management





バリアント管理ツールに必要なこと

- **ツールに合わせる必要の無いこと**
  - 既存のプロジェクトの構成、成果物をベースに、それらをツールやその手法に合わせることなく、インクリメンタルにバリアント管理を、安全に取り入れることが出来ること。
  - 既存のビルドやその他のプロセスの変更は最小限に

- **大規模・複雑なシステムへの対応**
  - 追加のフィーチャにより、取り得るバリアントは簡単に倍増することを考慮に入れて、複雑で規模の大きいバライアビリティのモデリングにも対応できること

- **派生開発（差分開発）への対応**
  - 資産・プラットフォームなどの変更・修正・進化に順応できること

＊Eclipse のような共通のツールプラットフォーム、ユーザインターフェイス

- バグ管理・追跡ツールのようなツールとは、リアルタイムな協調が取れること

＊APIを介して、あらゆるツールとの統合が誰にでも出来ること

pure::variants バリアント管理ツール

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

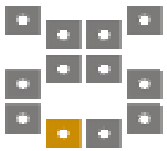
自動的にリソースを活用して、バリアントを生成

- 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 バリアント管理ツール

- 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

ありがとうございました！



pure-systems

- Telephone: +49 391 5445 69 -0

  the fast path for all questions – we look forward to your call

- Internet: http://www.pure-systems.com

  e-mail: info@pure-systems.com

FUJI SETSUBI

販売代理店：富士設備工業株式会社　電子機器事業部
TEL：072-252-2128　http://www.fuji-setsu.co.jp
担当：浅野　E-Mail：yoshio@fuji-setsu.co.jp