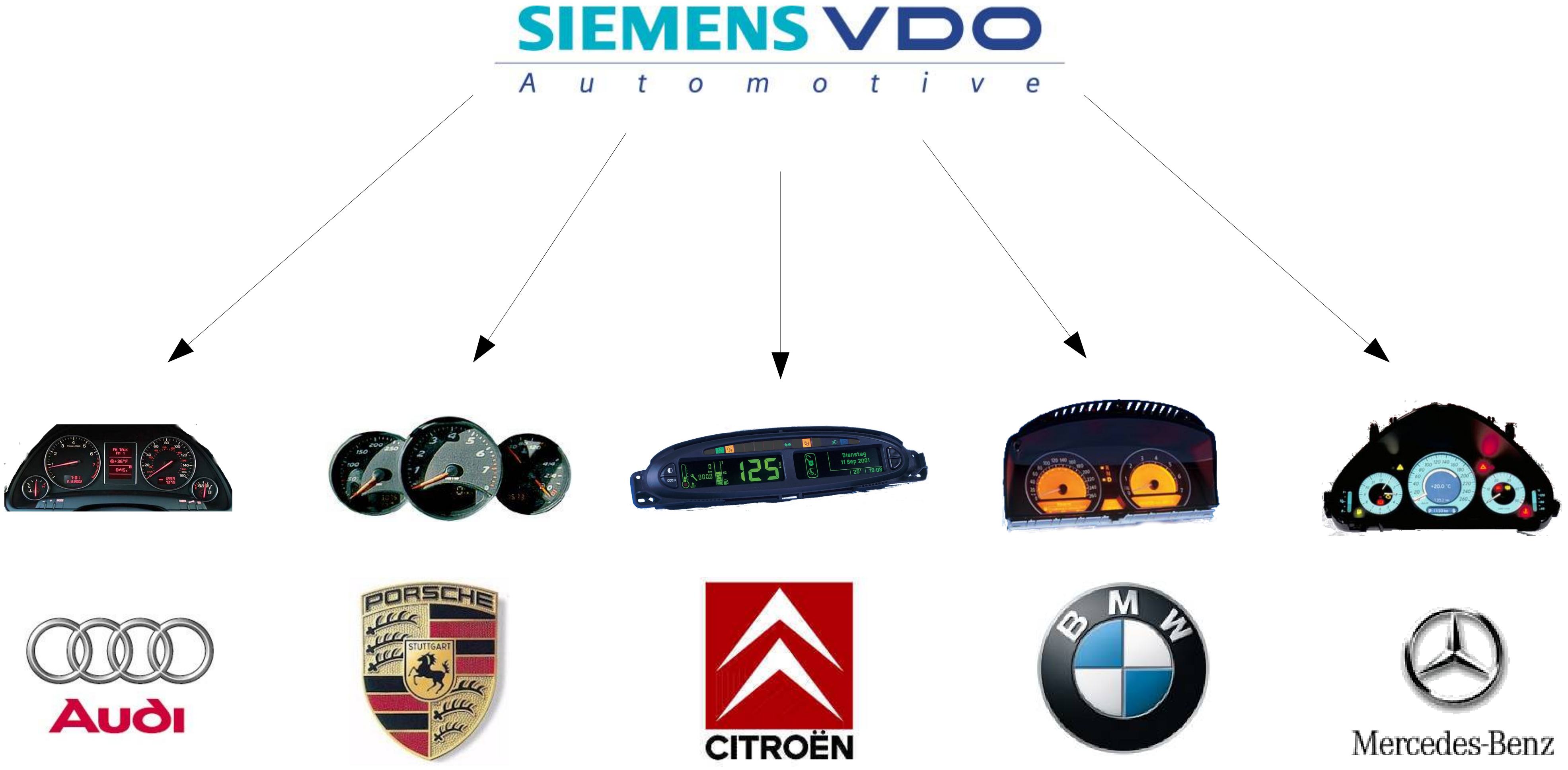
Seminar

Software Product Lines and pure::variants

Dr. Danilo Beuche
danilo.beuche@pure-systems.com

pure·systems

Who Needs Product Lines?



About pure-systems

- **Business Areas**
  - Development Tools
  - Software Development
  - Consulting & Professional Services
  - Training
- **Customer**
  - Mainly embedded systems manufacturers
- **Founded 2001, Location Magdeburg, Germany**

Presenter

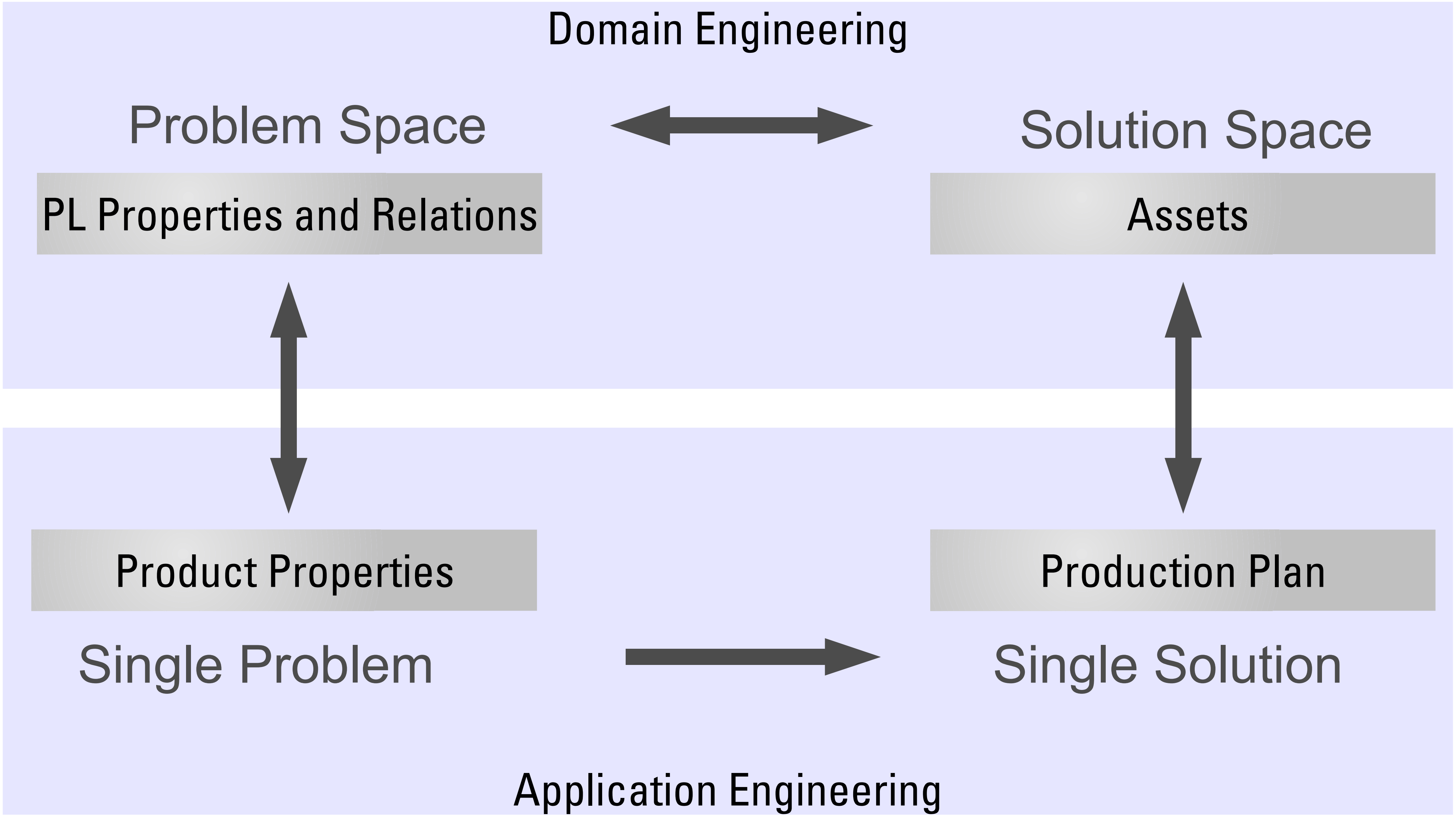
- Danilo Beuche
  - 2001-*: managing director pure-systems GmbH, Magdeburg, Germany
    - consulting in embedded software development and product line development
  - 1997-2003: Research assistant/PhD Student University Magdeburg, Germany
    - PhD on Software families for Embedded Systems
    - operating systems group, focus embedded operating system families
  - 1995-1997: Research associate GMD FIRST Berlin, Germany
    - parallel operating systems group, focus on families

Introduction to Software Product Lines

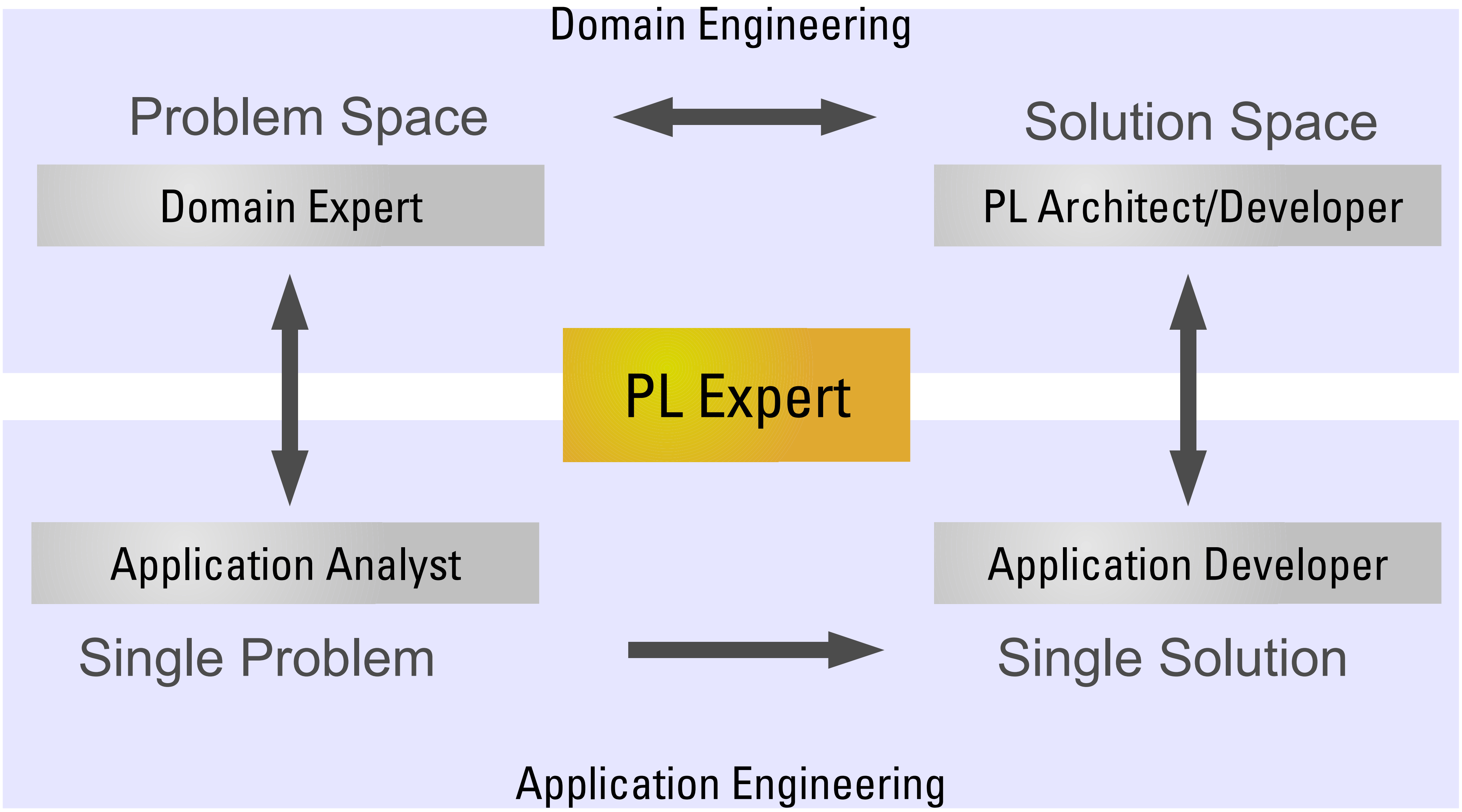
Terminology

- Software Product Line terminology is used throughout the seminar:
  - Problem Space vs. Solution Space
  - Domain Engineering vs. Application Engineering
  - Variant vs. Version
  - Variation Point
  - (Core) Assets

Terminology (2)



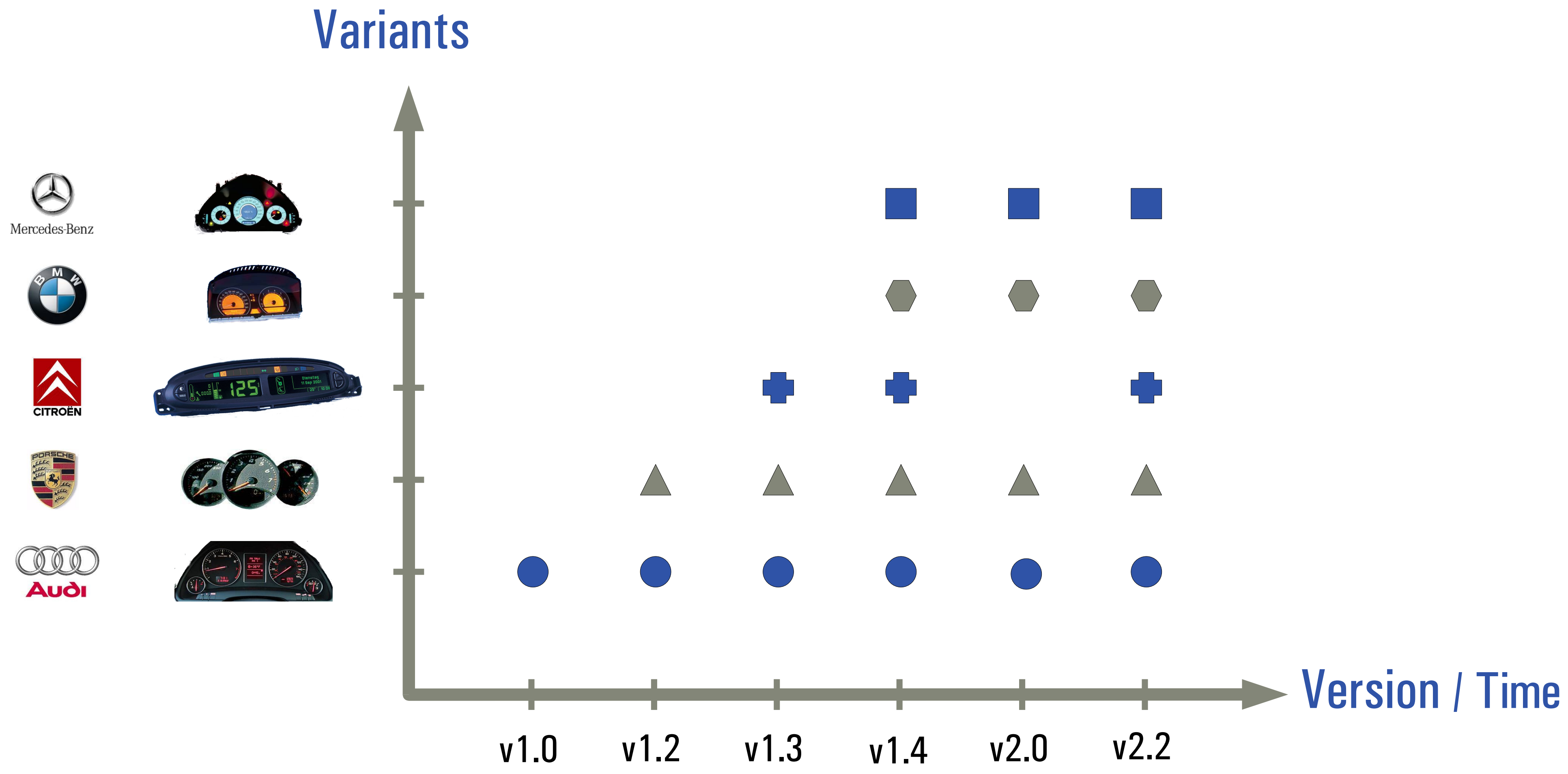
Terminology (3)



The Version Hell

	Product 1	Product 2	Product 3	Product 4
Component A	1.0	1.1	1.3	2.0
Component B	1.0	1.2	2.1	2.4
Component C	1.0	1.0	2.3	4.0

Orthogonality of Variants and Versions



Terminology (4): Version and Variants

- Version (also Revision)
  - Versions of an object represent the same object at different times. The object may or may not have changed in different versions.
- Variant
  - Variants represent objects with different user-distinguishable properties. They are derived from a common base object .

Terminology (5): Variant vs. Version

- Variants may be represented by versions

- Variants may be derived from a single version

- A single variant may over time be derived from different versions of an object

▶ In single system development variant and version are often used interchangeably, when variants are exclusively represented by changes over time and are thus always coupled to a specific version.

Terminology (6): Variation Point

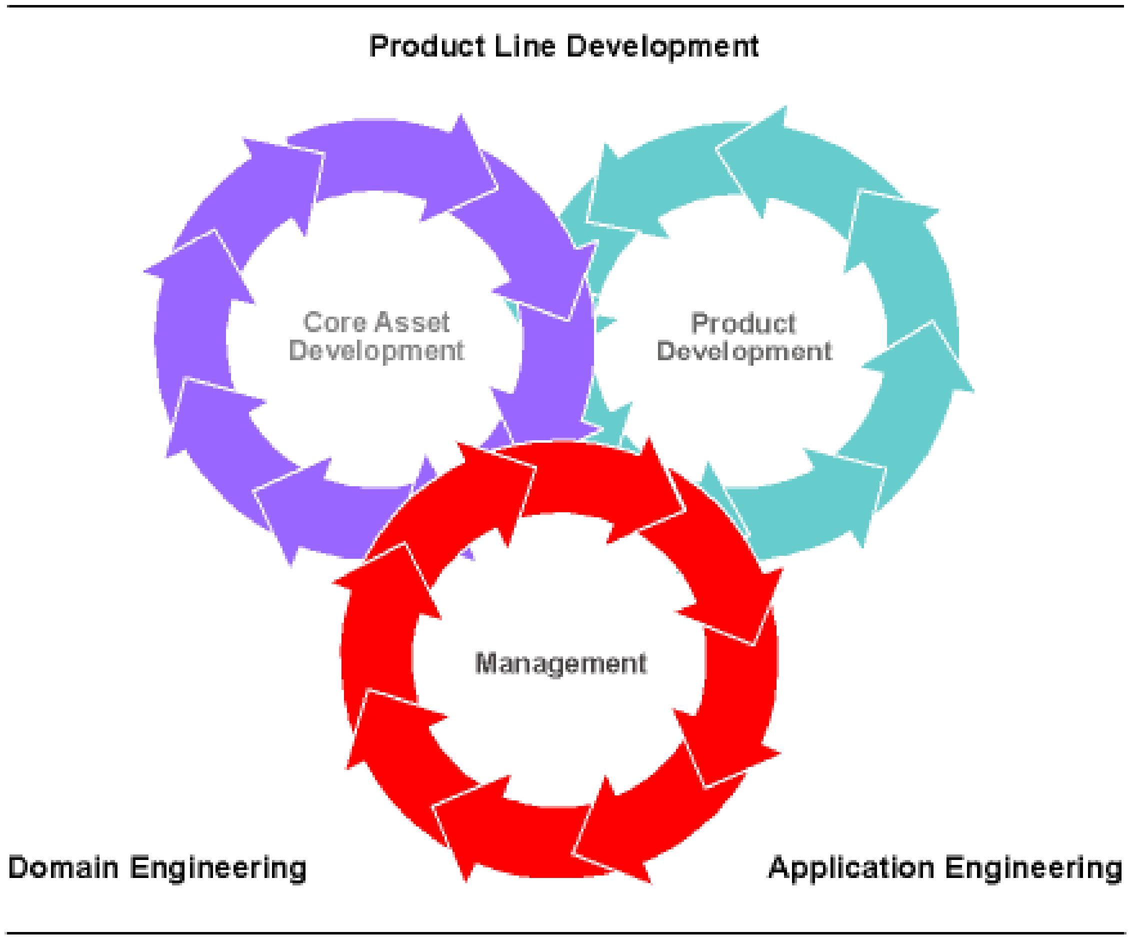
- Variation Points
  - identify all places where members of a product line may differ from each other
  - exist in problem and solution space
  - have a binding time such as compile time, link time or run time.
  - Example:
    - Problem space:

      "The car has either two or four passenger doors"
    - Solution space:

      #ifdef / #else / #endif encapsulated code fragments

Quick Look at SPL-Development

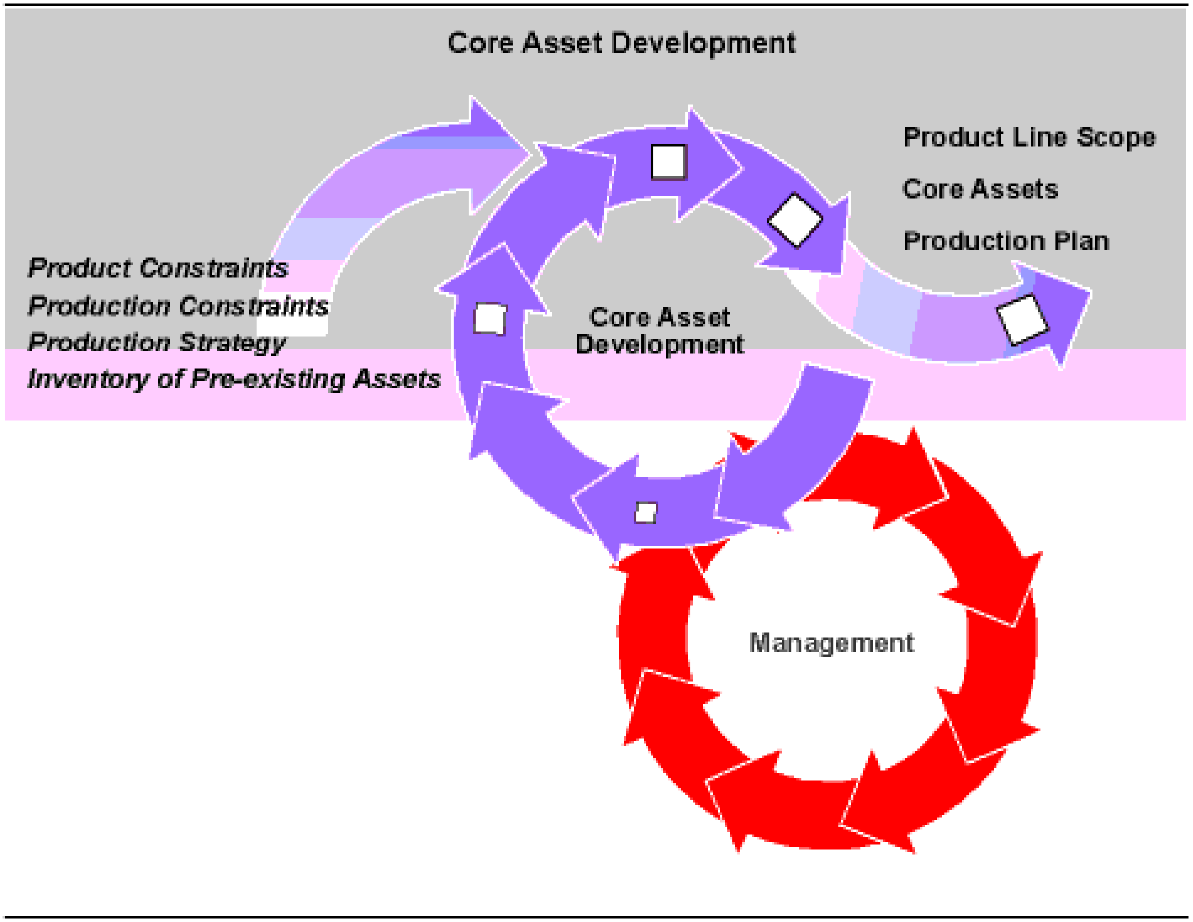
Essential SPLD Activities (SEI SPLP Framework)



Taken from SPLPFW 4.2, copyright SEI CMU.

Quick Look at SPL-Development

Core Asset Development I/O (SEI SPLP Framework)



Taken from SPLPFW 4.2, copyright SEI CMU.

It is all about Economics

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.“

Reuse

Reuse Technology – Challenges

- no first level concept of variation points available
- variations cross-cut all levels
- low degree of formalization
- shared responsibility between stakeholders with very different perspectives

Reuse Process - When and Why Reuse Fails (1)

- product development done by engineering companies
  - they sell projects typically based on the expected effort
- creating and maintaining reusable assets is more expensive than one-of-a-kind development style
  - technology mismatch for given domain/organisation
  - not enough education of stakeholders
  - highly parallel development of similar functionality in separate project teams

Reuse Process - When and Why Reuse Fails (2)

- development of unused/unusable assets
  - application team does not use components developed by a specific core asset team
  - products do not meet acceptance criteria of customers
- management does not support development organization well enough
  - projects with a high degree of reuse tend to cost more in the beginning but managers want a cheap start

Reuse - Key Issues

- Economics
- Education
- Communication and Coordination
- Technology

Variant and Variability Managment

Variant and Variability Management

- takes care of description and maintenance of
  - variation point information for a product line (variability)
  - instances of the product line (variants)
- closes the gap between reusable assets and instances of them
- is all about dealing with complexity in various areas

Variant and Variability Management

- ... is complex because
  - variability grows more or less exponentially
    - each option doubles the number of potential variants
    - each group of n alternatives still creates n-times more variants
  - it has to deal with anticipated and unanticipated variation requests
  - involves more stakeholders compared to single system development

Variant and Varibility Management with

pure::variants

ALM and Variant Management

pure-systems	Variant Management with pure::variants				
	Portfolio Management	**Requirements and Analysis**	**Design and Development**	**Configuration Management**	**Quality Management**
Borland	Tempo	Caliber RM	Together	StarTeam	Silk
IBM	Portf. Manager	RequisitePro	Softw. Architect	ClearCase	ClearQuest
Telelogic	Focal Point	DOORS	TAU/Rhapsody	Synergy	Tester/Change
MKS	Portfolios	Requirements		Source	Test Managem.
	Application Lifecycle Management (ALM)				

pure::variants

Integration into Development Processes

- keep and use existing code base and tool environment
- version management support
- independent from technology

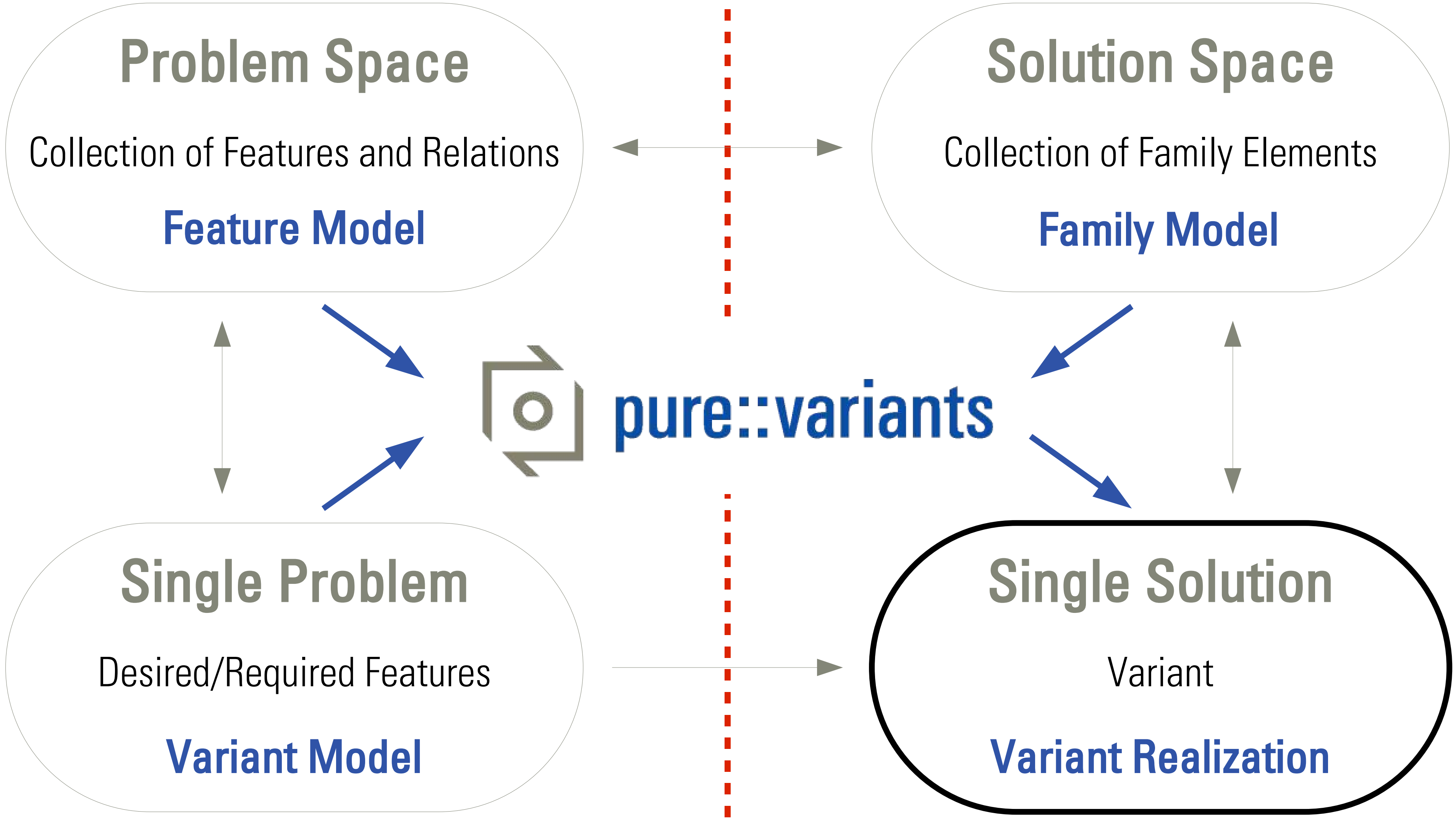
Efficient Variant Modelling

- accumulation of configuration knowledge
- validation of variant configurations

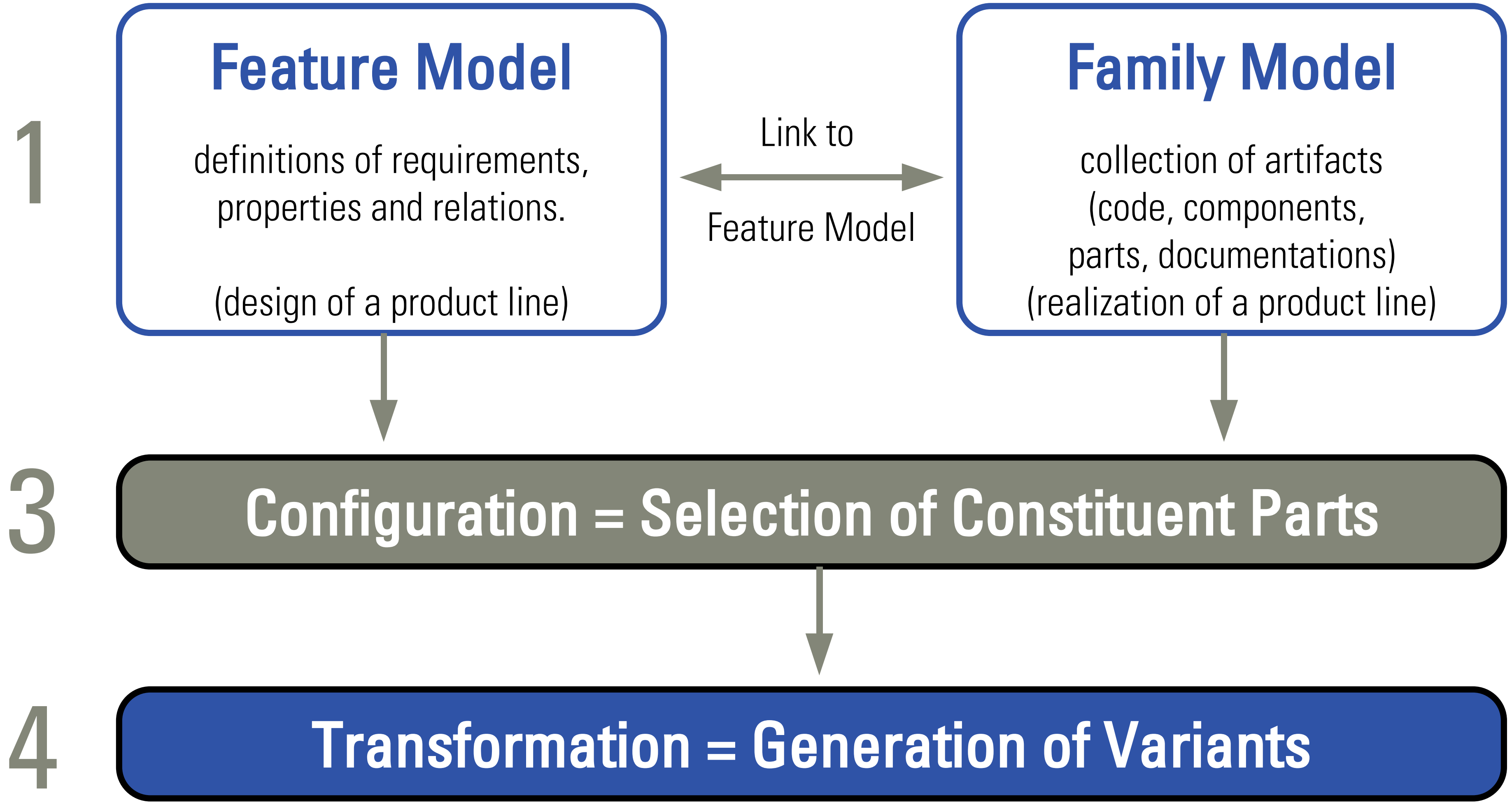
Automated and Resource-Efficient Variant Generation

- source code packaging (e.g. from version management repositories)
- generation of code, documents, bills of material

Development with Variant Management



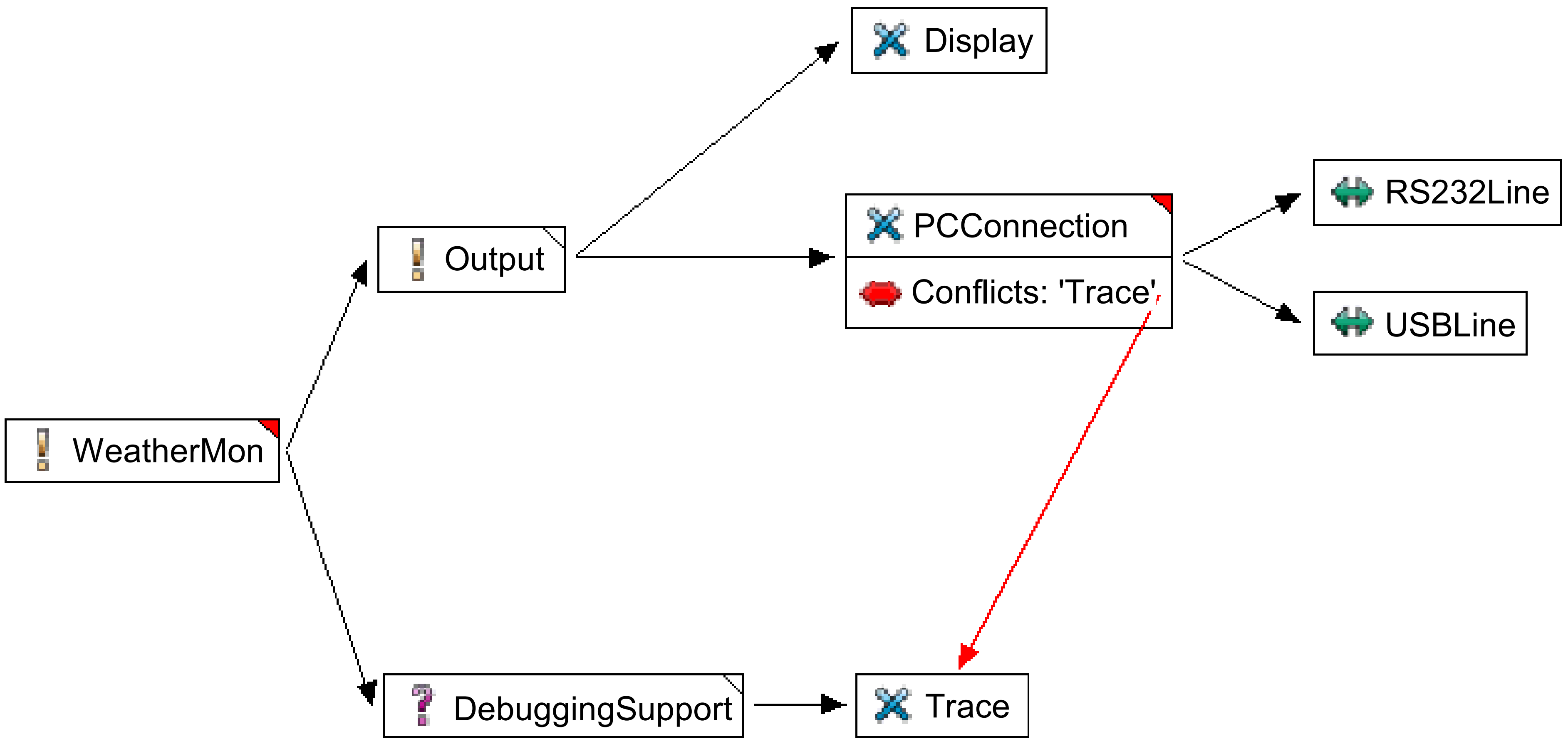
Workflow with pure::variants


1
Feature Model
definitions of requirements, properties and relations.
(design of a product line)
Link to
Feature Model
Family Model
collection of artifacts (code, components, parts, documentations)
(realization of a product line)
2
3
Configuration = Selection of Constituent Parts
4
Transformation = Generation of Variants

Example Weather Station: Product Variants

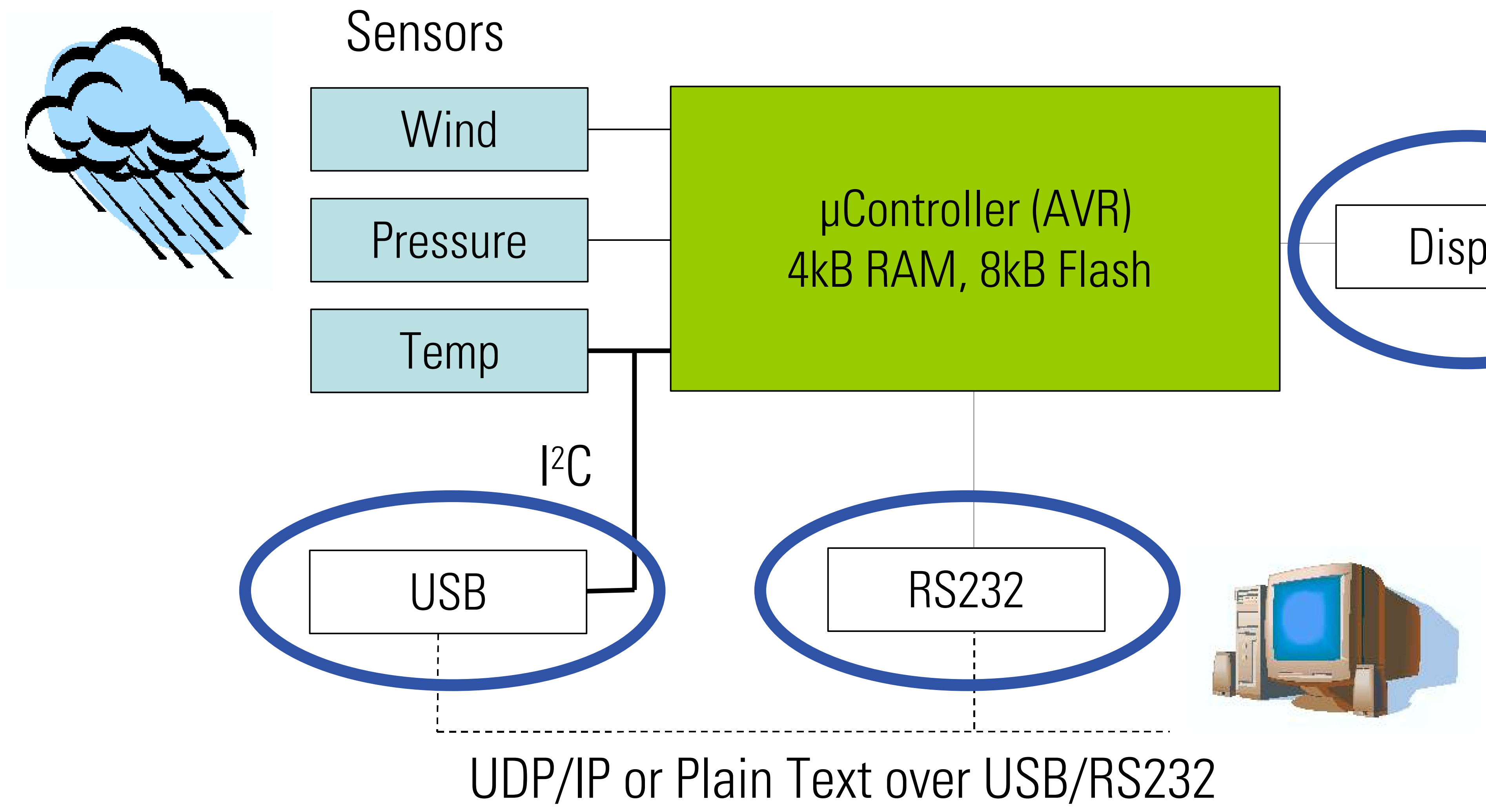
Variant	Feature
Thermometer:	Display, Temperature
Indoor:	Display, Temperature, Pressure
Outdoor:	Display, Temperature, Pressure, Wind
Deluxe:	+ PC Data Recording
Internet Edition:	+ TCP/IP
PC Interfaces:	+ Serial Interface RS232
	+ USB Interface

Problem Space – Feature Model

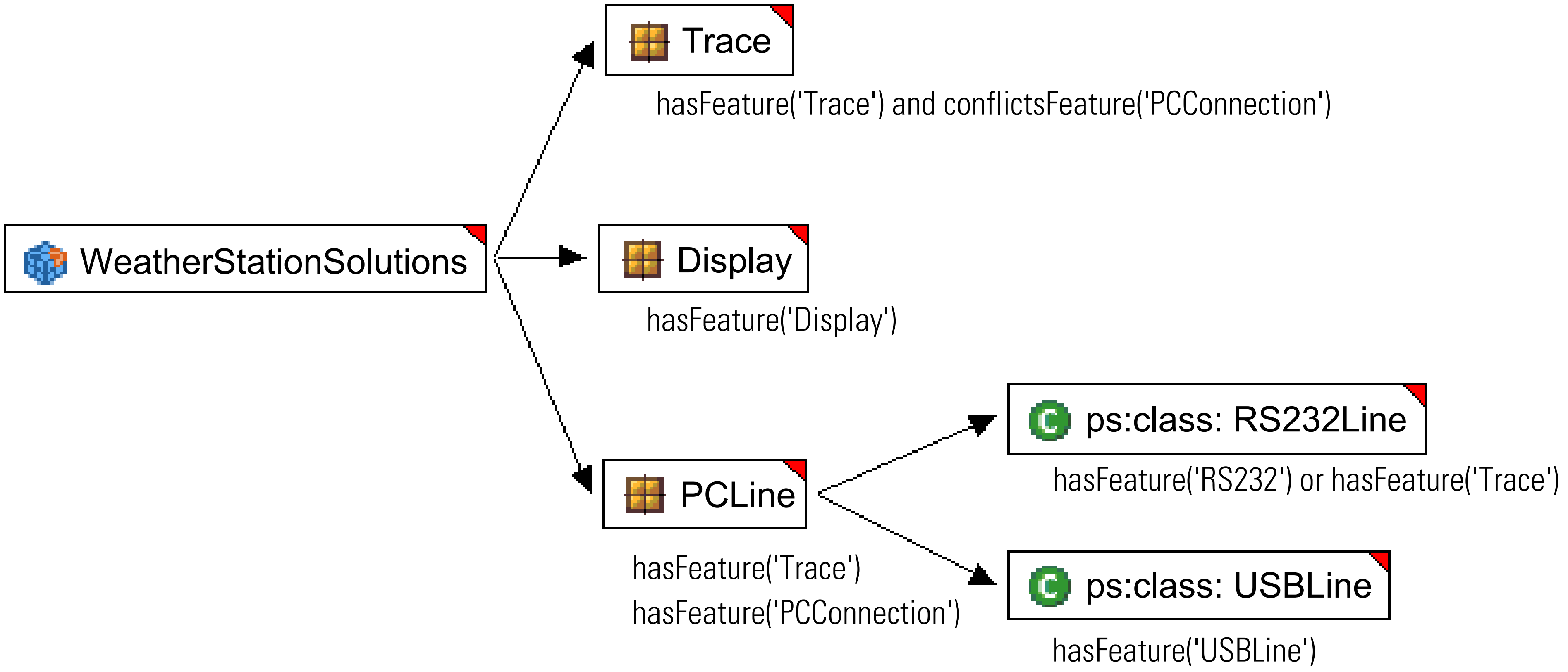


partial view

Example Weather Station: Hardware



Solution Space – Family Model



partial view

LIVE - DEMO

pure::variants

at work

Product Variants

pure::variants Professional

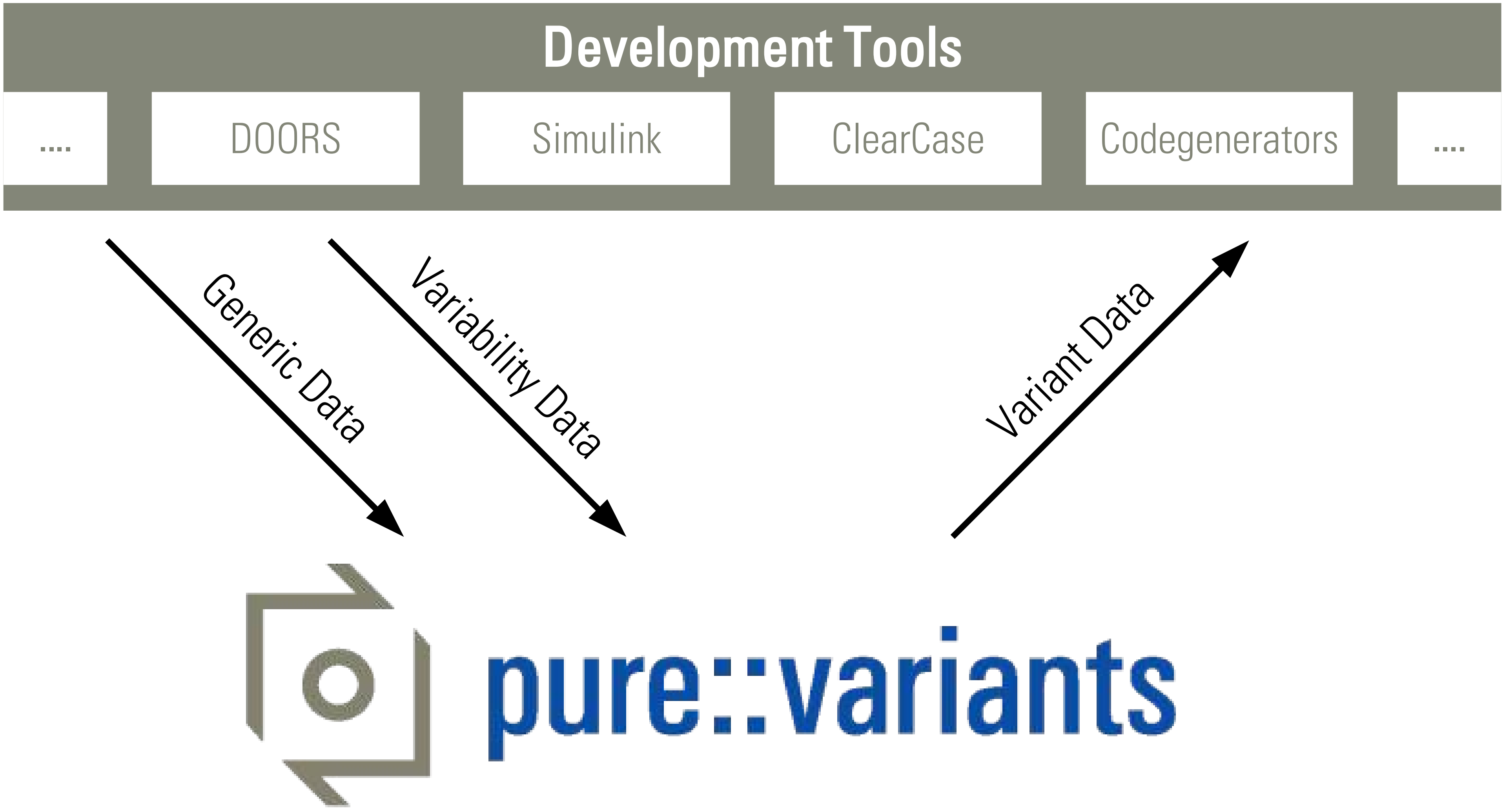
- models are stored locally in file system as XML
- collaboration using standard software configuration management tools such as CVS, Subversion, Perforce, ...
- versioning and branching handled by software configuration management tool

pure::variants Enterprise

- models are stored in a centralized database
- collaboration in real-time, changes will be automatically visible to any connected user
- integrated reporting and history
- versioning and branching handled by pure::variants server
- all functionalities of p::v Professional

- full support for japanese user interface in pure::variants 3.0

Product Integration



Extensions

Extensions integrate pure::variants optimally into existing tool chains.

- 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**

Telelogic ALLIANCE PARTNER

Borland

The MathWorks Connections Program

SUBVERSION

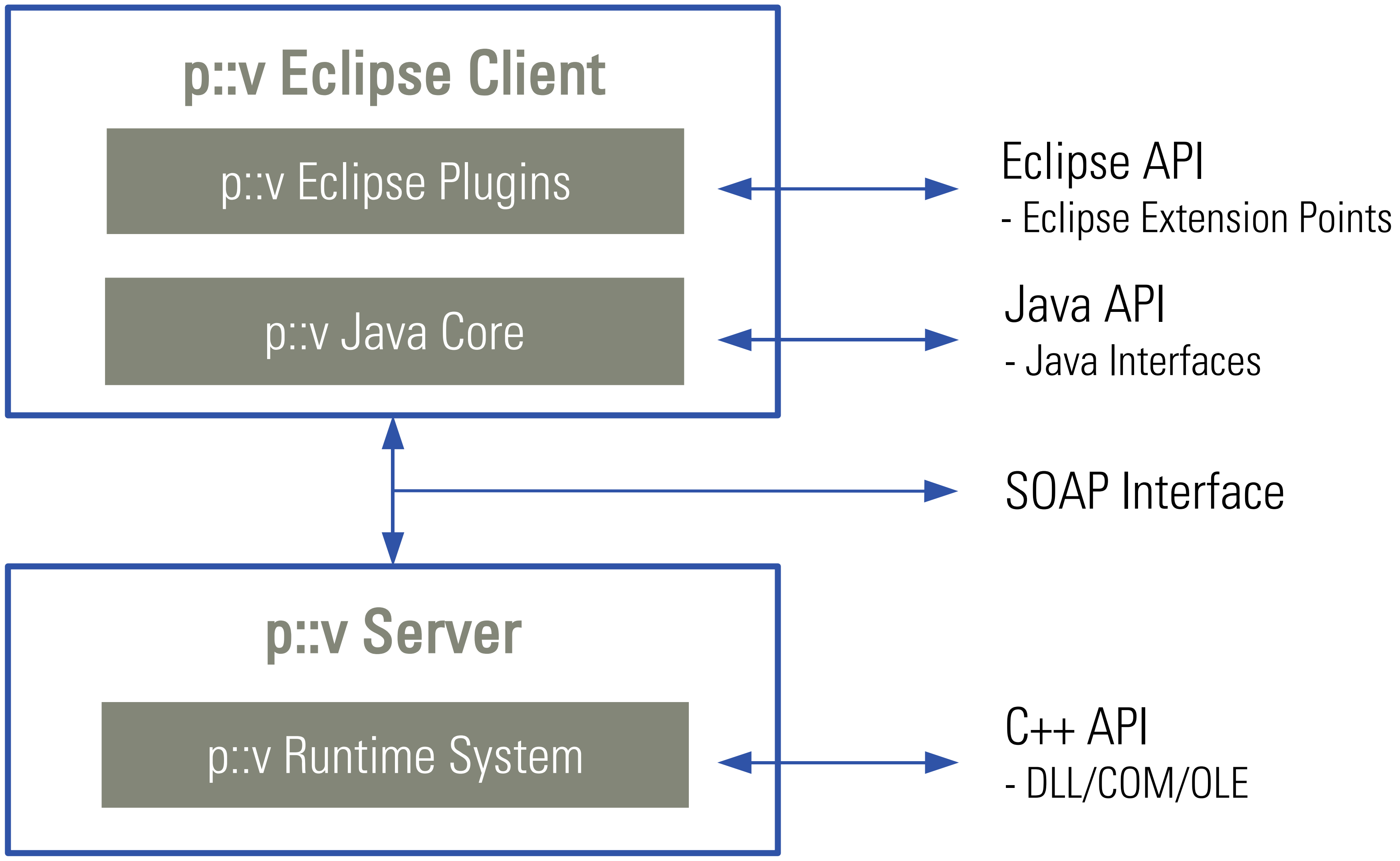
CVS

SAP

Rational. software

BIRT

Integration Interfaces



Example Danfoss

Problem

- For 4 market segments control software is provided by different, worldwide distributed teams (about 70 persons). Software is based on very similar hardware.
- Reuse mostly „ad-hoc“.
- Cost and development effort were high.

Task

- Migration to controlled and systematic reuse based on common software plattform.

Example Danfoss

Our Role

- Participation during process definition
- Coaching and evaluation of newly setup platform team
- Supporting the migration to pure::variants

Results

- 1st stage of migration finished successfully after 6 month: all projects develop based on common plattform
- 2nd stage of migration (introduction of pure::variants) almost completed
- Double number of products with about the same development capacity

Example Automotive Supplier

Problem

- Configuration of control software with about 2000 features could not be handled efficent manually.

Task

- Modelling of complete configuration knowledge in pure::variants and generation of configuration files.
- Distributed real-time remote access to models and configurations.

Example Automotive Supplier

Our Role

- Product supplier
- Adapation of pure::variants to customer demands: integration of customer specific rule language.

Results

- Configurations are created in minutes instead of days.

Transforming Legacy Systems into Software Product Lines

Transition and the SPLP-Framework

- Basically, the transition process is used to collect and create all information described in the framework from pre-existing artefacts

- What is different to a start from the scratch?
  - Much required information is already present, only hidden.
  - Changes to processes and artefacts may be hard to achieve. It is more complicated to introduce adequate processes and/or artefacts.

Questions Before Starting a Transition

- What is missing for a SPL/SPLD in my organsation?
  - Is it...

    ... systematic core asset development?

    ... problem and solution space specifications?

    ... applications build on top of p-space& s-space?
- Why make the transition?
  - Is it...

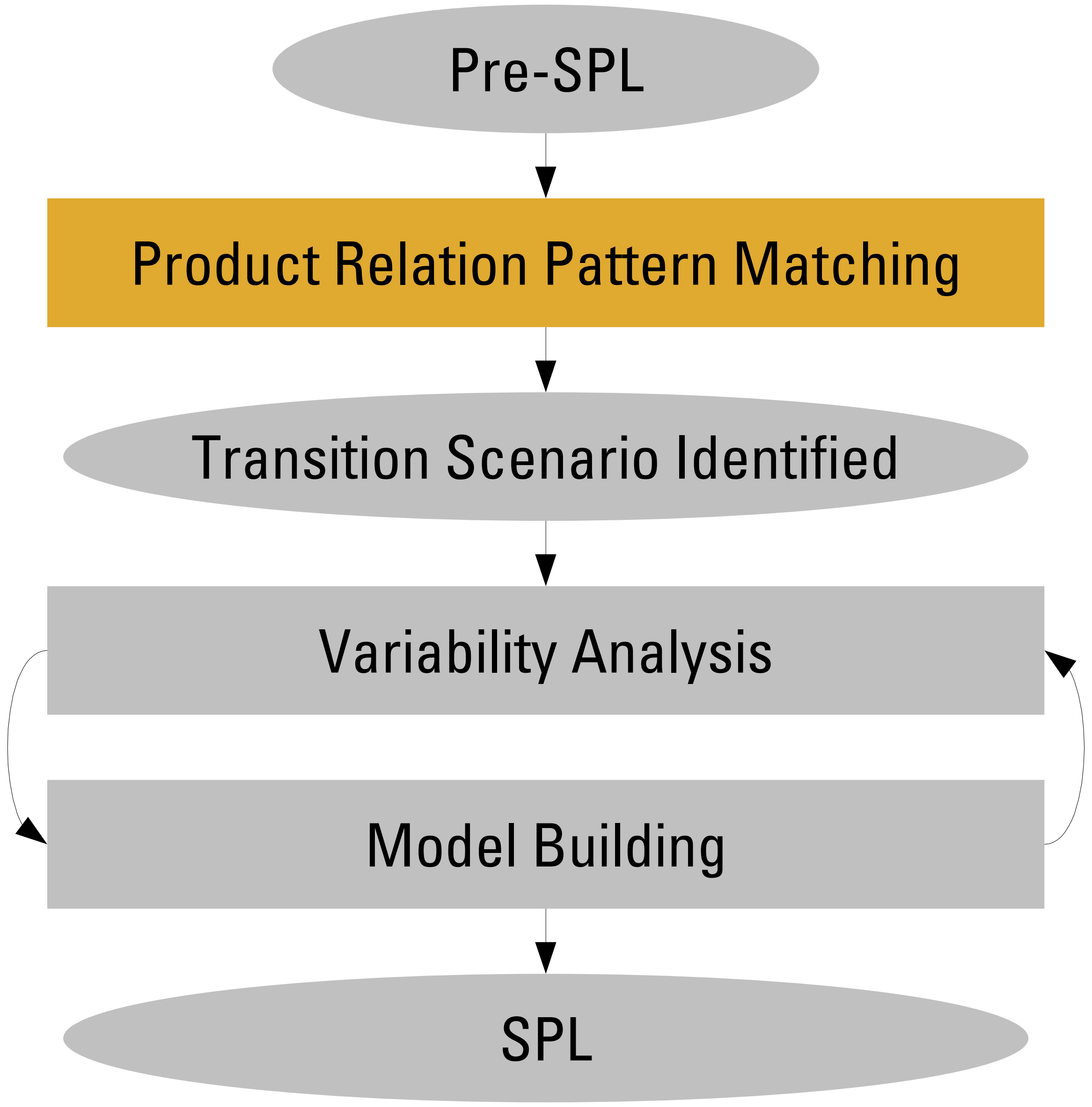
    ... overall development cost reduction?

    ... shorter development time for similar products?

    ... increased product quality?

    ... ?

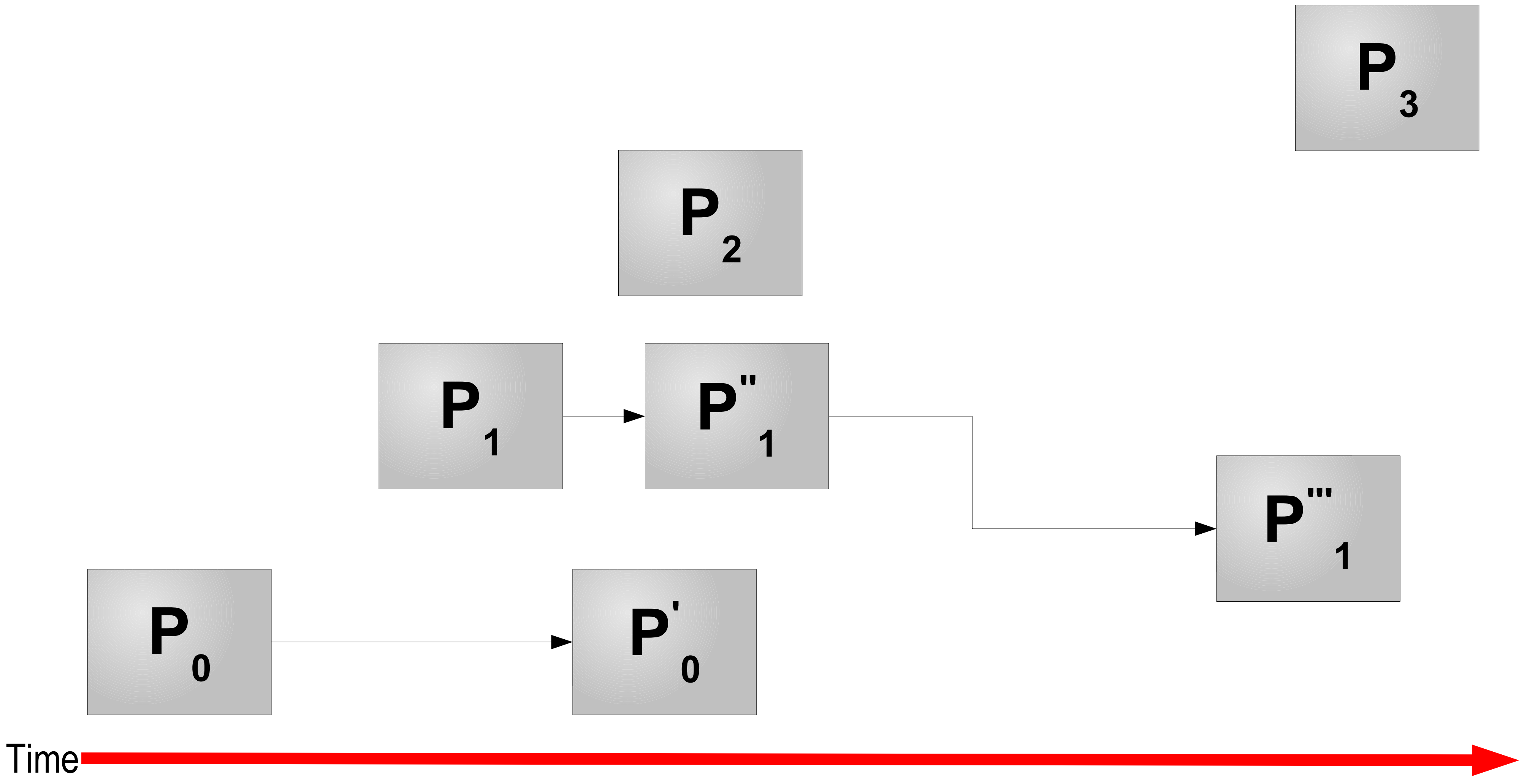
Transition Steps


Pre-SPL
Product Relation Pattern Matching
Transition Scenario Identified
Variability Analysis
Model Building
SPL

Product Relation Pattern: Product Forest

No shared software core assets

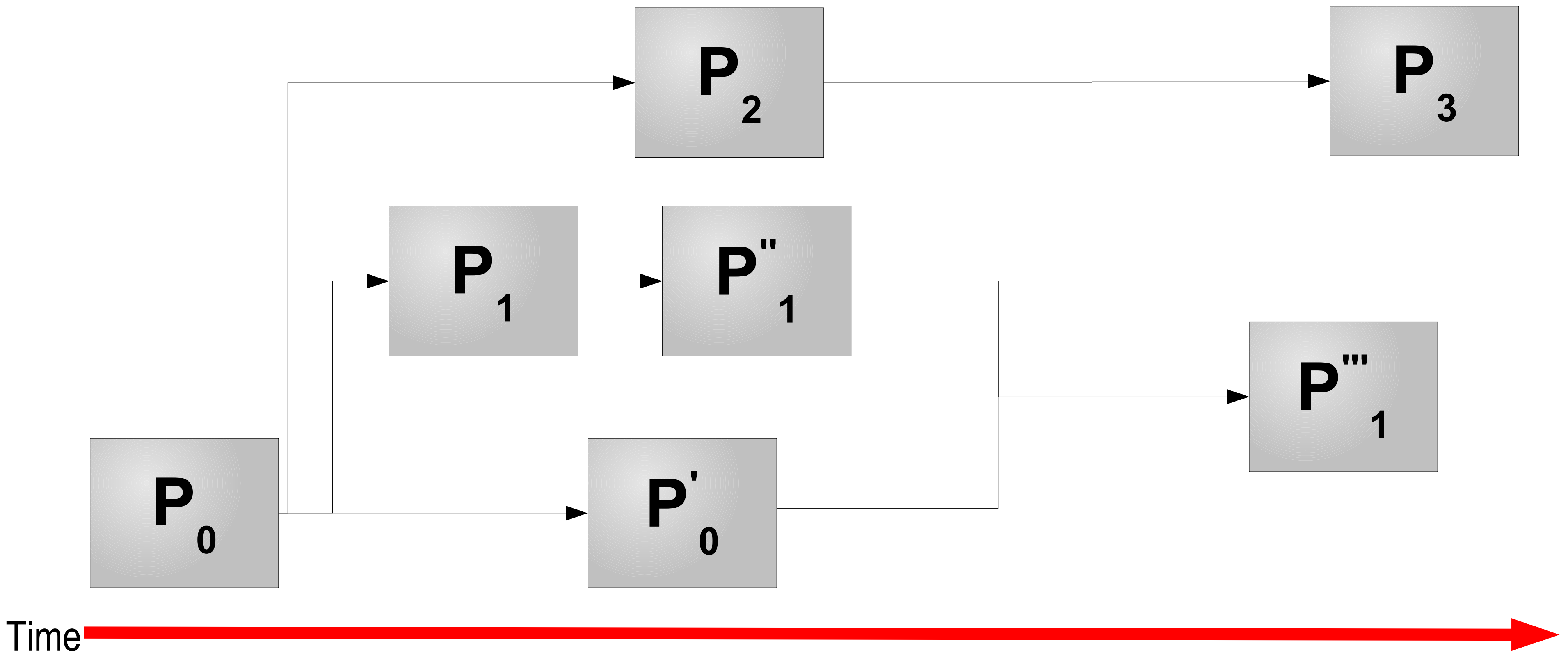
Similar user-visible functionality



Product Relation Pattern: Product Bush

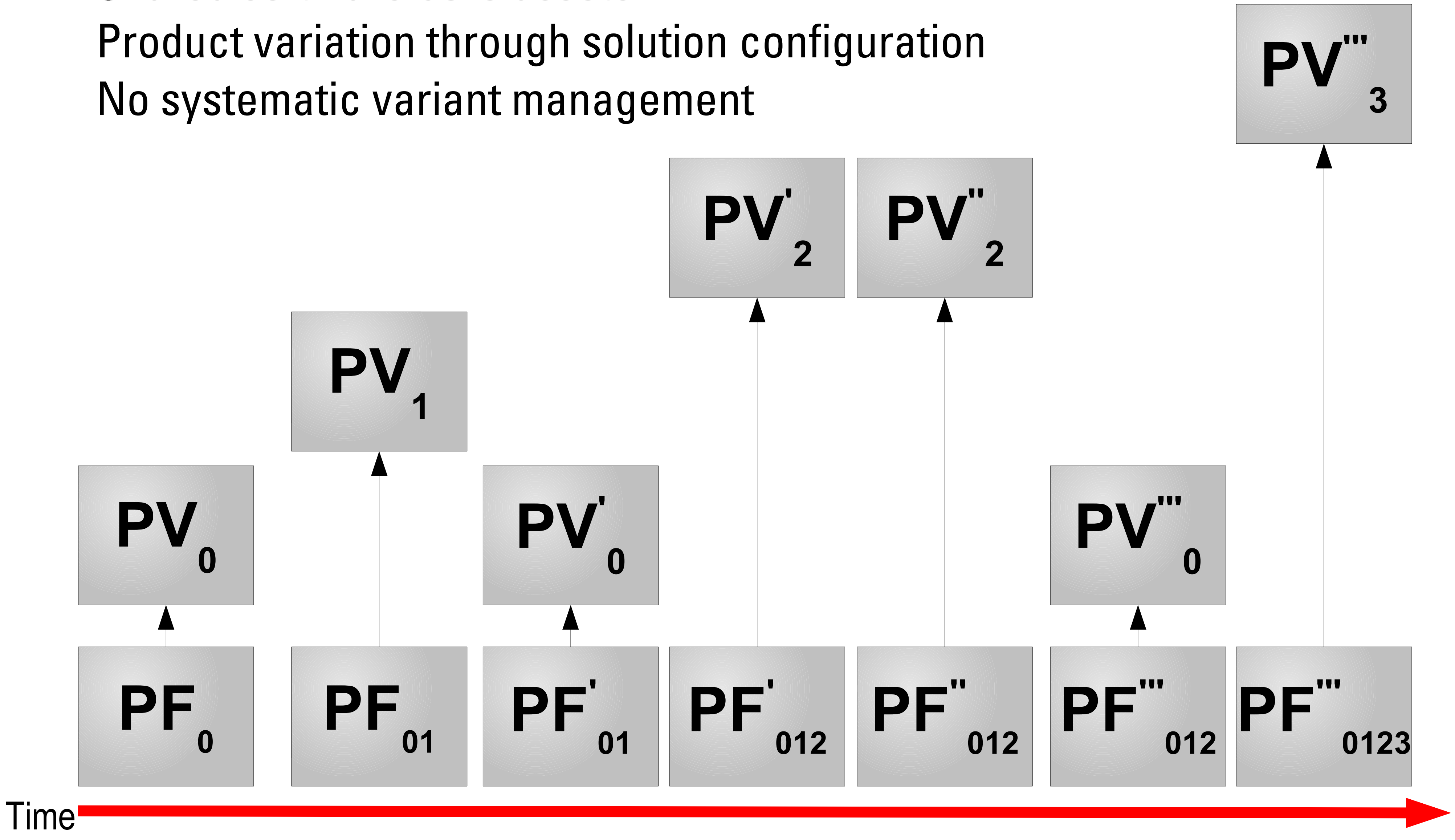
Shared software core assets

Product variation through development path branching



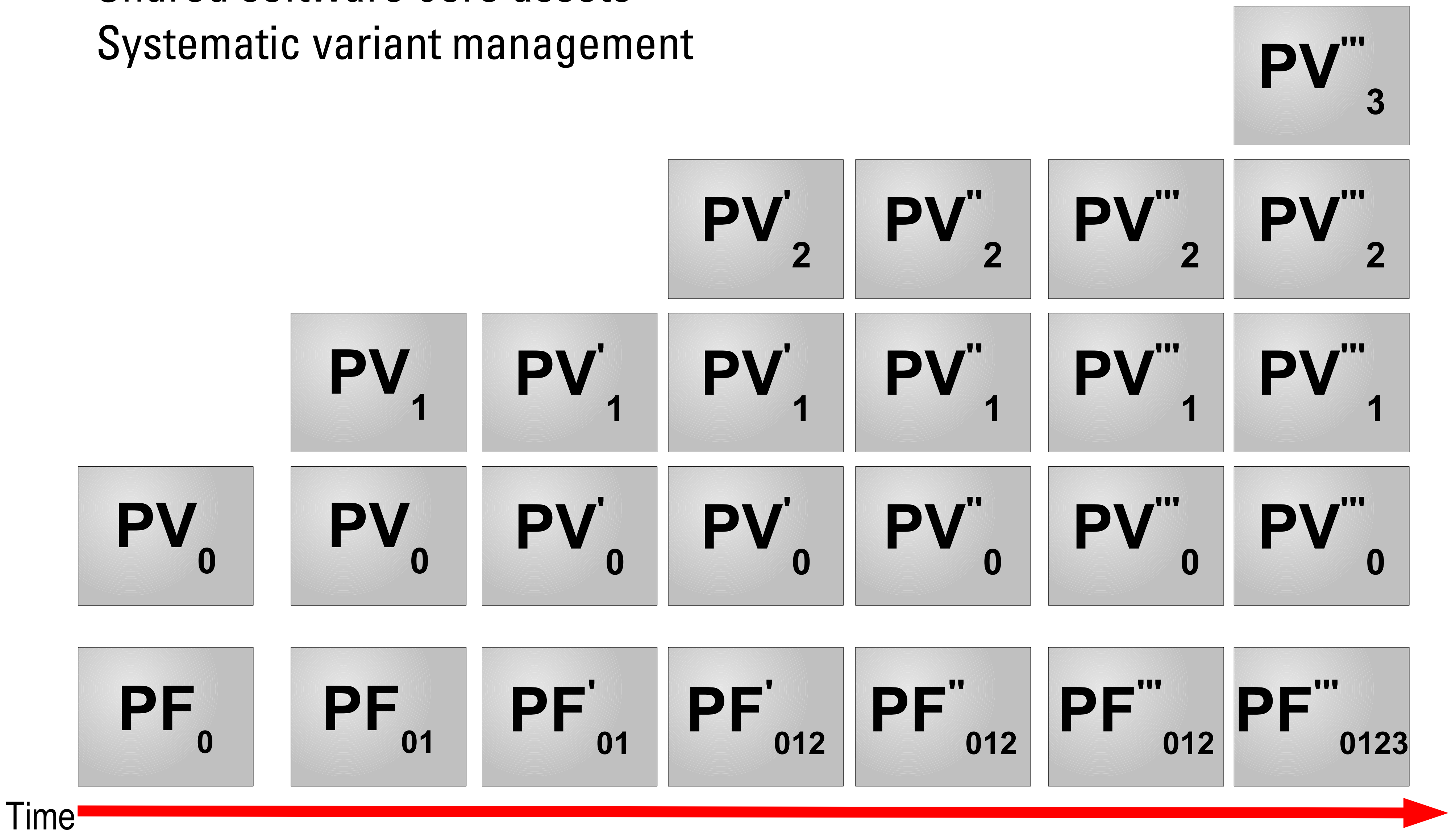
Product Relation Pattern: Product Gang

Shared software core assets
Product variation through solution configuration
No systematic variant management

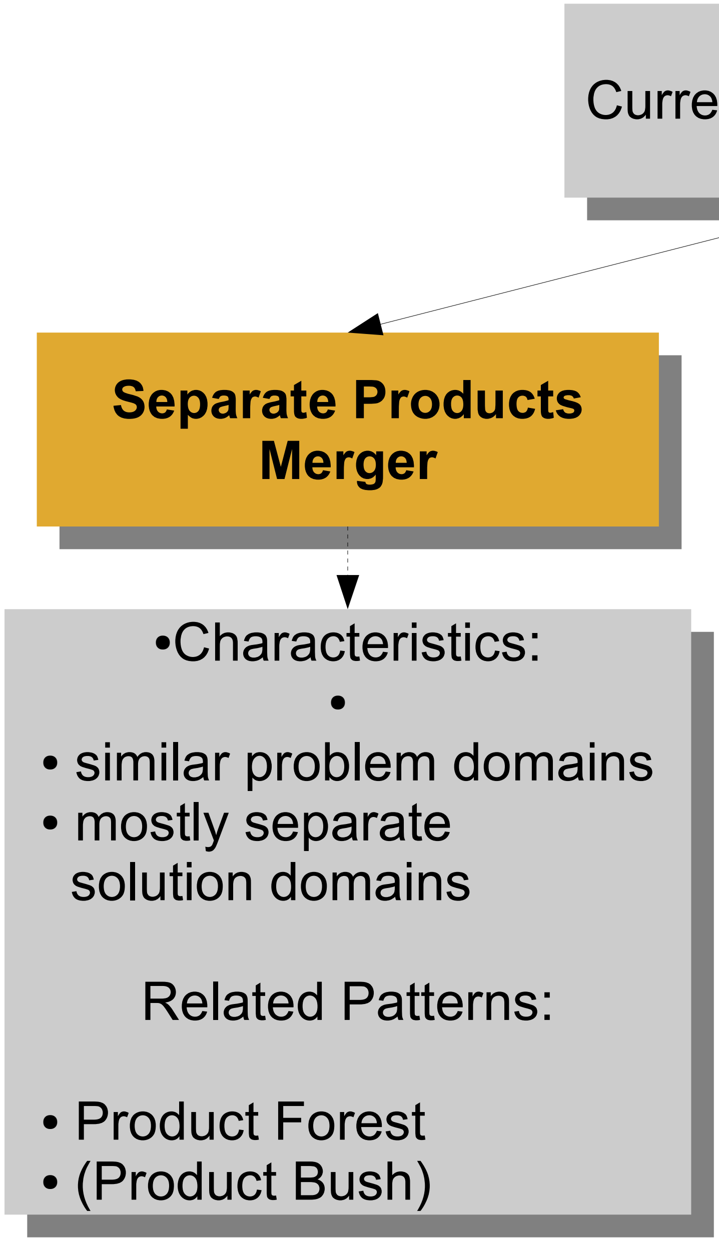


Product Relation Pattern: Product Family

Shared software core assets
Systematic variant management



Transition Scenarios



Separate Products Merger

•Characteristics:

•

- similar problem domains
- mostly separate solution domains

Related Patterns:

- Product Forest
- (Product Bush)

Reuse Improvement

•Characteristics:

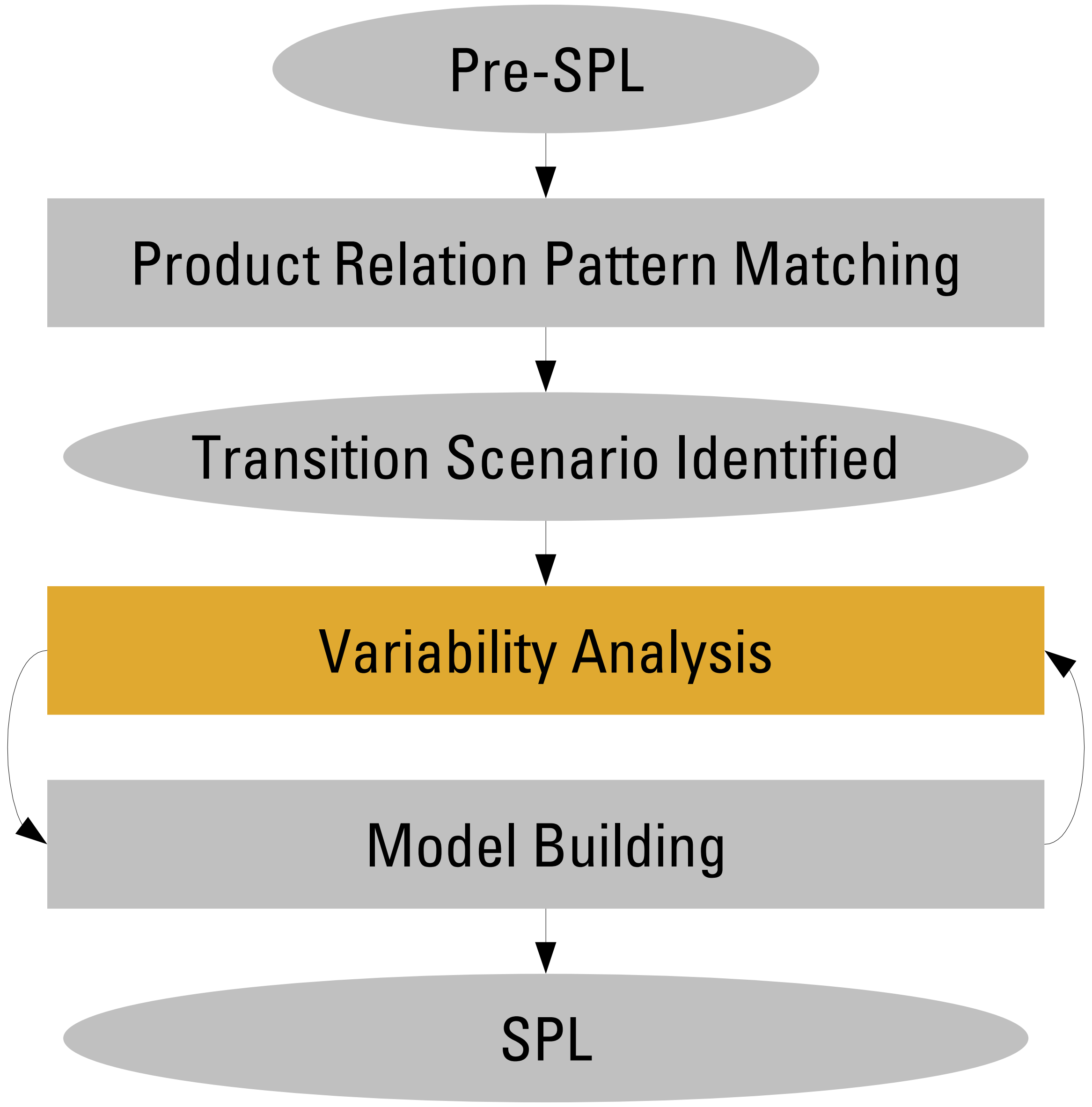
•

- similar problem domains
- same or very similar solution domains

Related Patterns:

- Product Bush
- Product Gang

Transition Steps


Pre-SPL
Product Relation Pattern Matching
Transition Scenario Identified
Variability Analysis
Model Building
SPL

Variability Analysis

- What to do:
  - Identification and extraction of (potential) variation points of the product line
  - Extraction of variation point constraints
- Where to look:
  - source code
    - structure, algorithms, technologies
  - documentation
    - user manuals, internal documents, code comments
  - management strategies
    - version control, configuration

Variability Analysis (2)

- Results:
  - initial set of problem space variation points
  - initial set of solution space variation points
  - initial set of variation point constraints
- Purpose:
  - providing input for
    - product line scoping
    - problem space definition
    - core asset identification
    - production plan creation

Variation Point Description

- A variation point description consists of
  - choices (features) available at that point
  - variation constraints (requires/conflicts...)
  - space it belongs to (problem/solution)
  - relevance/importance w.r.t. product line variability
  - binding time
    - product configuration time, compile time, link time, installation time, user configuration time, run time
  - binding mode
    - static vs. dynamic
  - used variability pattern

Analysis Direction

- Decide which direction to go:
  - forward: start with problem space variability
    - good if user manuals and/or product specification provide good insight into variabilities between existing products; works for product forests
    - permits to perform product line scoping w/o (closely) looking at the existing software assets
  - backward: start with solution space variability
    - good if software design and architecture reflect problem space closely; especially suited for product bushes and gangs
    - requires problem space variability extraction before product line scoping can be started

Variation Point Extraction

1. Decide where to look for variability.
2. Build list of representative variation point patterns for your software.
3. Search for variability patterns in your input material:
   - manually
   - supported by tools

Problem Space VP Extraction

- Typical inputs:
  - user documentation
    - installation and user manuals
    - white papers
  - development documentation
    - requirements specifications, use cases
    - architecture design documents
    - log files in version control system (esp. product bush)
    - configuration files (esp. product gang)
    - communication with customers (feature requests etc.)

The Commonality and Variability Extraction Approach

- developed by Fraunhofer IESE in CAFÉ project as part of their **P**roduct **L**ine **S**oftware **E**ngineering Framework

- focusses on user documentation, but may be used also with development documentation

- operates on textual structure and text comparison

CaVE - Approach

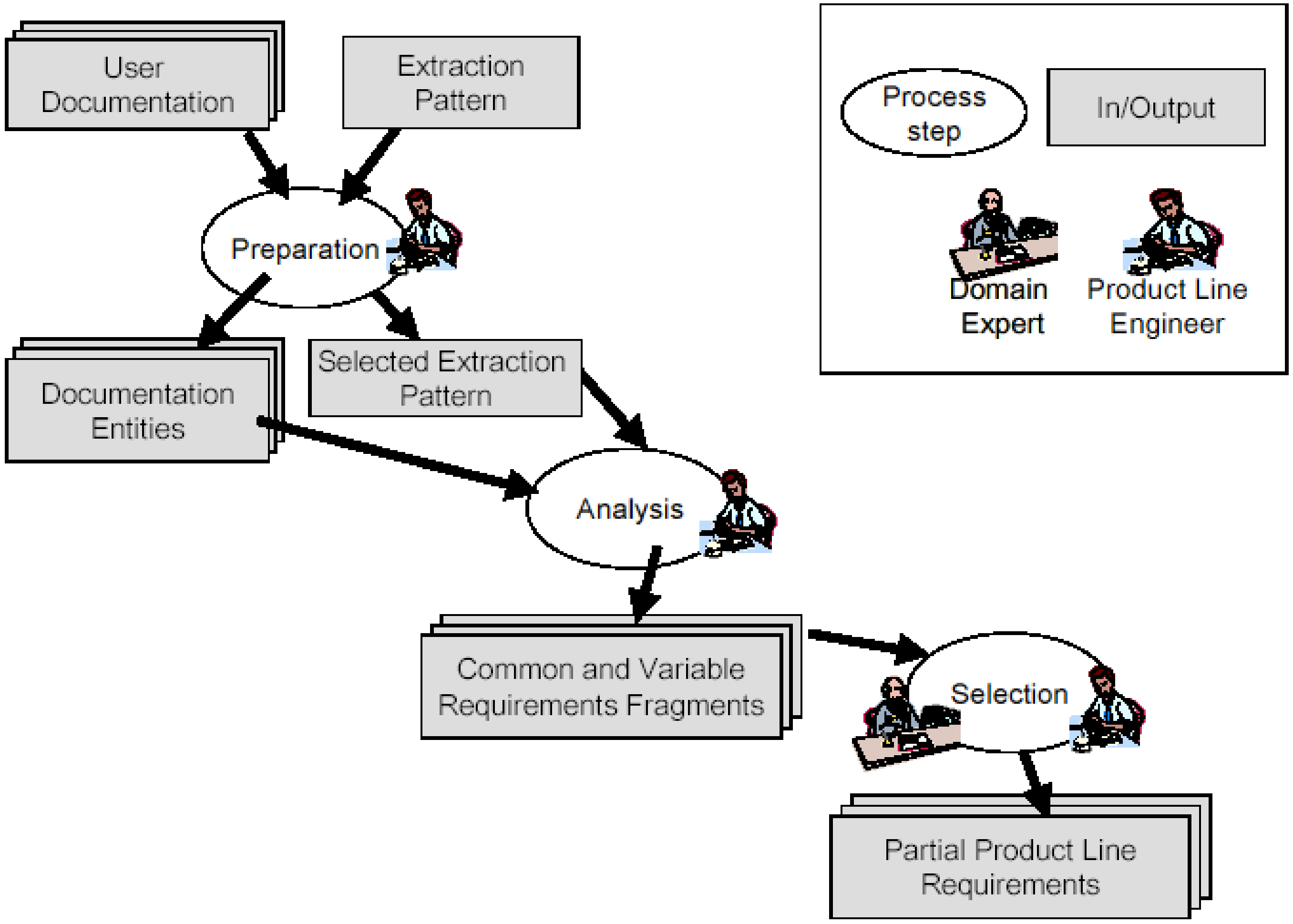


image © IESE 2004

CaVE - Pattern Template

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.

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

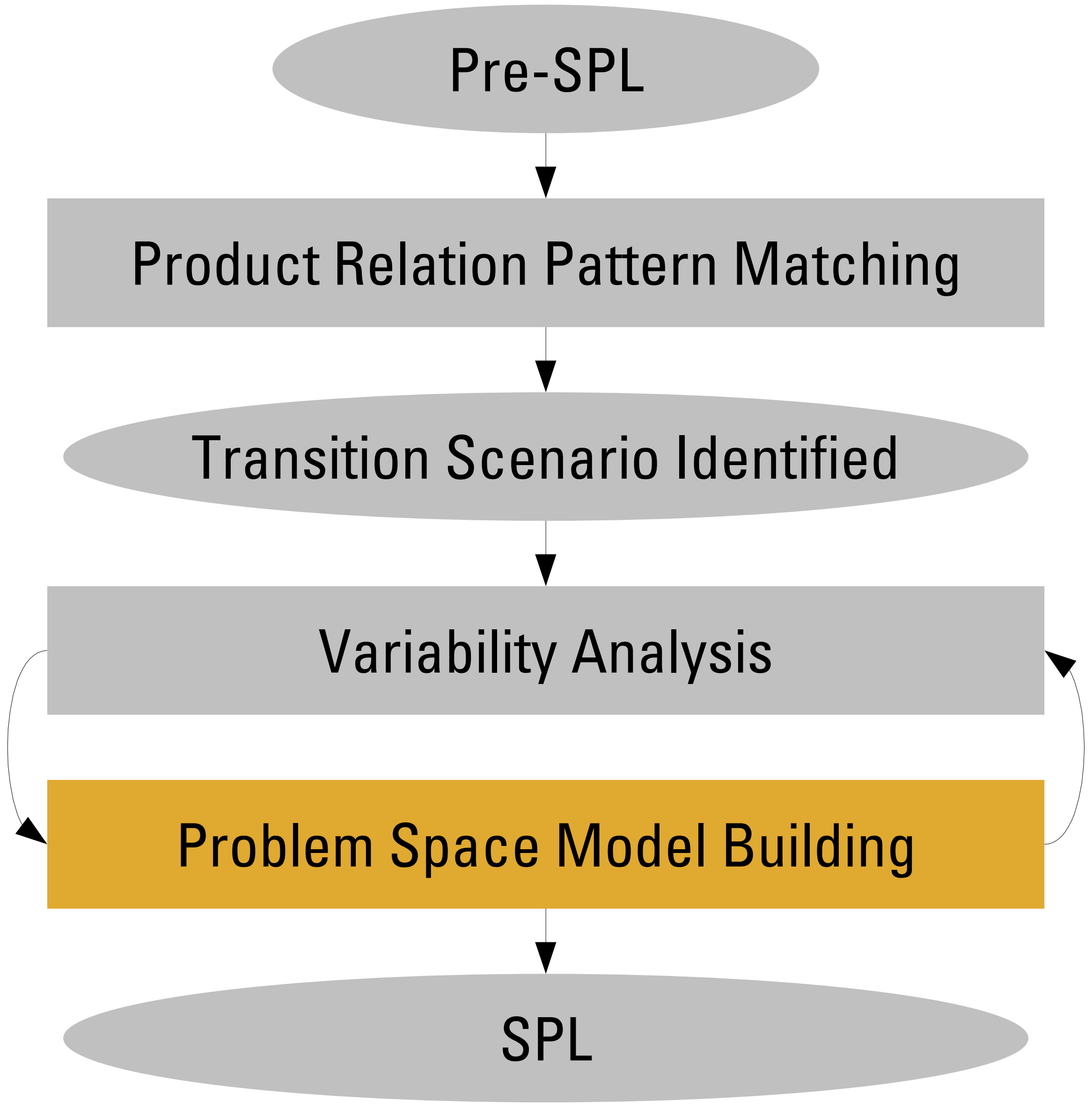
CaVE - Pattern Example (2)

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”

CaVE - Summary

- Pro:
  - splits work between product line engineer, an (external) expert in product line development, and the internal problem space experts, thus reduces work load for domain experts
  - well documented with industrial case studies in [CaVE]
- Con:
  - no (documented) experiences with larger and/or very diverse documentation
  - almost no tool support available

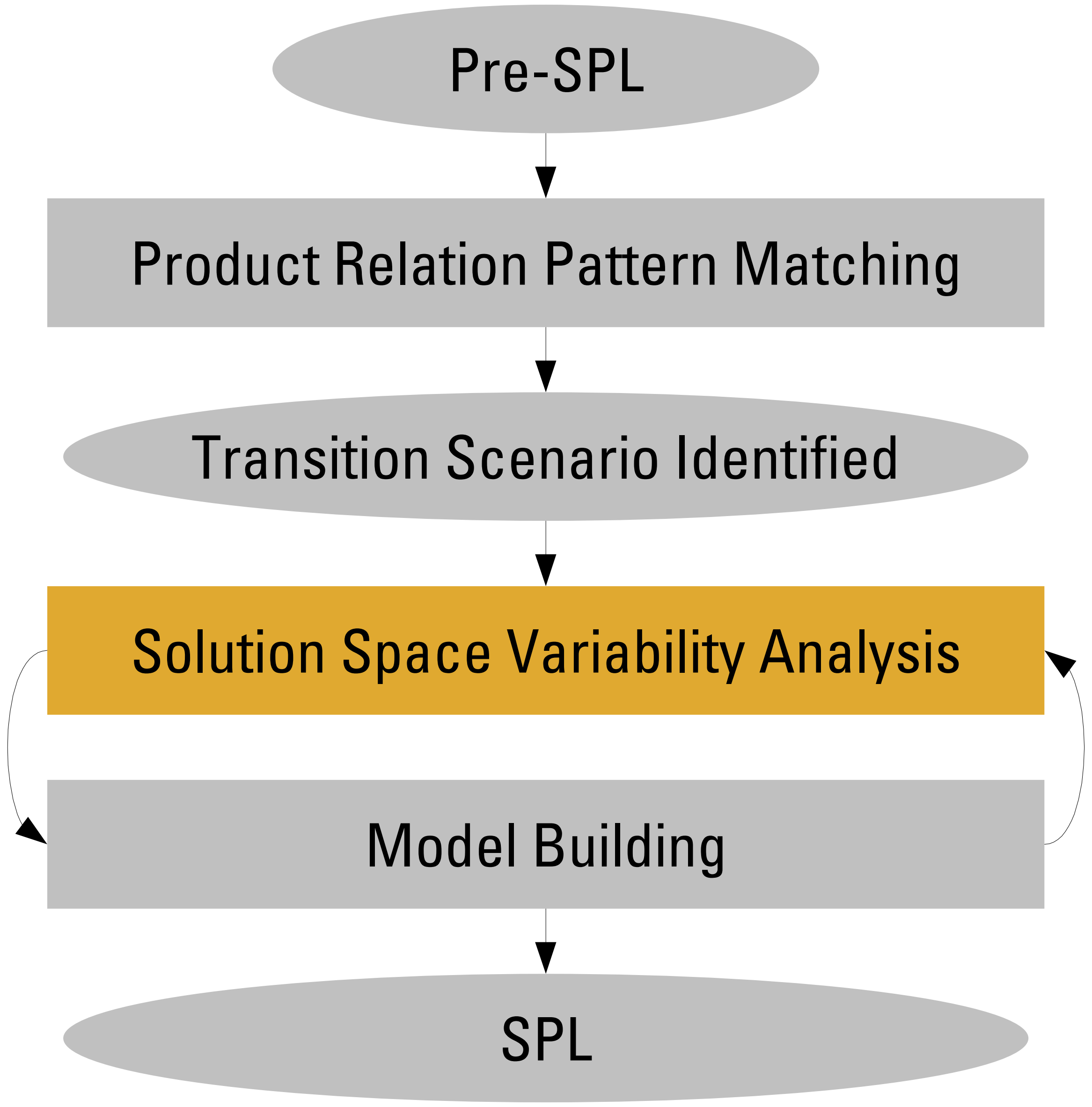
Transition Steps


Pre-SPL
Product Relation Pattern Matching
Transition Scenario Identified
Variability Analysis
Problem Space Model Building
SPL

Problem Space Modelling – Feature Modelling

- feature models are used most often because:
  - the feature model concept is easy to understand
  - even complex feature models are easy to navigate
  - have enough expressive power for most real-world problems

Transition Steps


Pre-SPL
Product Relation Pattern Matching
Transition Scenario Identified
Solution Space Variability Analysis
Model Building
SPL

Solution Space VP Extraction

- Typical inputs:
  - architecture design documents
  - source code
  - configuration scripts
  - project build descriptions (e.g. makefiles)
  - version control system structure and usage
  - configuration files and file formats

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`

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;`

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 struc-ture).
Related Pattern	#6
Example	make could be used with different makefiles/make variable set-tings as input to generate different builds of a project. The diffe-rences in makefiles may be stored in different files or as diffe-rent versions&branches of same file in version control systems. Differences in parameters are often visible in build scripts.

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);` `}`

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

Solution Space VP – Pattern #6

Name	# 6 File level variation
Short Description	Different files with similar names are used as alternative imple-mentations
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

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 creati-on from the place where the object is created. If the type or im-plementation of the factory object is changeable by configurati-on, it may represent a variability
Related Pattern	
Example	

Solution Space VP Detection

- In many cases looking at a small amount of code reveals used patterns. (Programmers tend to repeat themselves)
- Since often embedded in syntactically restricted code structures, solution space patterns can be more easily detected using (simple) tools.
- The amount of detected variation point candidates is often huge.

Solution Space VP Simple Analysis Tools Example

Pattern #1

Standard Unix tools used only (also for Win32)

Find and count all #if[n]?def/#elif (#define) statements:

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

Analyse name intersection:

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

Get all files where #if[n]?def/#elif is used

```
ifnames `find -name "*.[hc]*"`
```

Solution Space VP Simple Analysis Tool – Example Use

CVSNT (a cross platform re-implementation of CVS)

- 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

Solution Space VP Analysis Tools

DMS (SemanticDesign): rule based language analyser and transformer, many languages

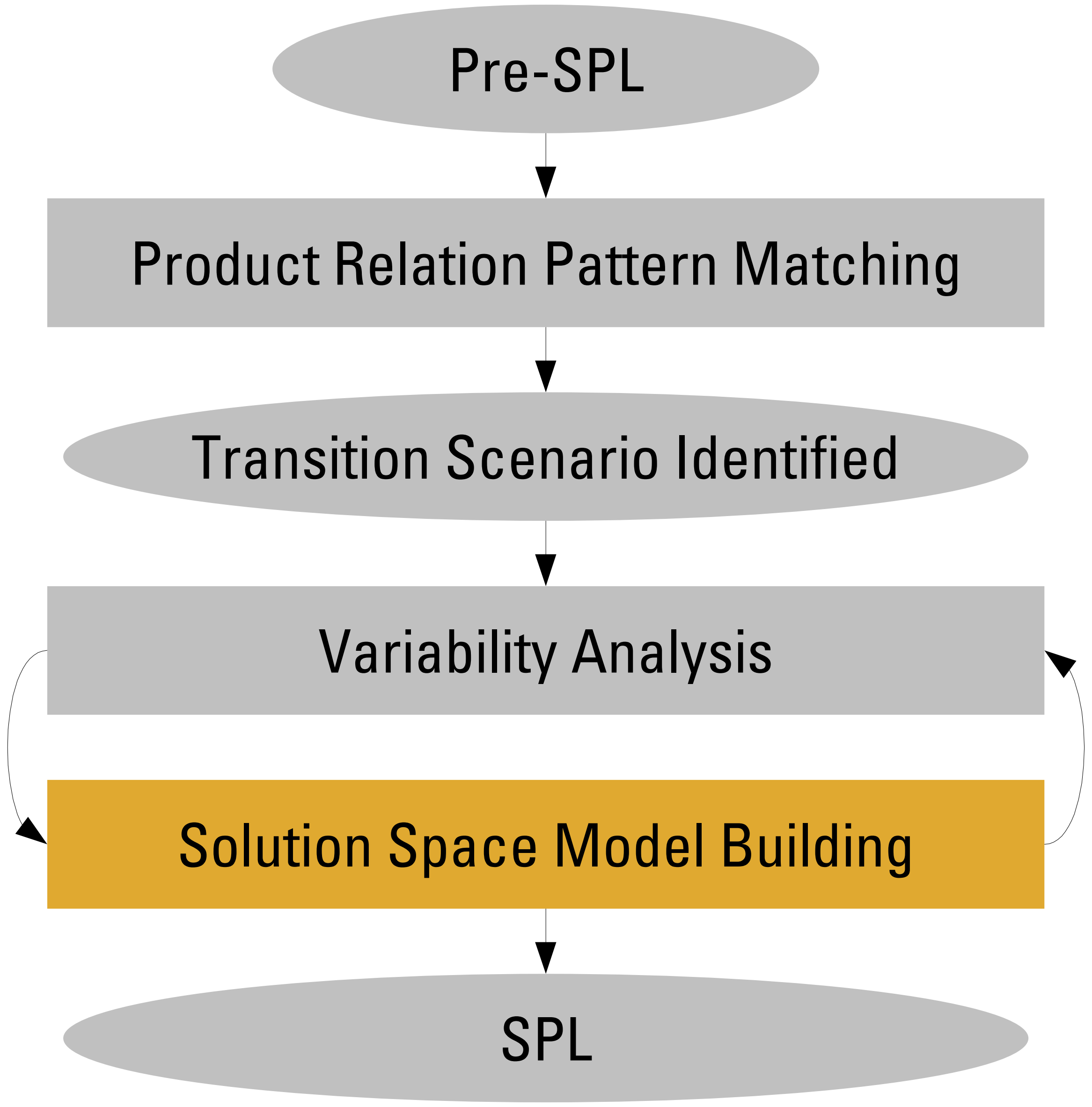
PUMA*: C/C++ parser and manipulator (part of AspectC++)

AspectJ*: Java, pointcuts and "declare"

Eclipse JDT*/CDT*: internal analysers, good analysis quality esp. for Java

* open source/freeware

Transition Steps


Pre-SPL
Product Relation Pattern Matching
Transition Scenario Identified
Variability Analysis
Solution Space Model Building
SPL

Solution Space – Architecture

- reference architecture
  - recovery
    - product gang, product bush
  - building
    - product forest
- core asset identification
  - weight identified architectural components by
    - expected use in future products
    - measured use in existing products
    - expected effort for adaptation
  - not all existing assets have to be used!

Production Plan: Problem – Solution Mapping

- problem space decisions do not always have one-to-one mapping into solution space variation points
- a problem space to solution space mapping is required to select/configure core assets from problem space configurations
- pure::variants' family models are used to model solution space and the mapping, often in combination with other tools

Production Plan Automation: Tool Support

- Script-based configuration tools

  autoconf*: Unix-specific configuration tool, especially for multi-platform development with C/C++ language

- MDSD tools

  MetaEdit+: powerful domain specific modelling tool

  openArchitectureWare*: powerful java-based model transformer and code generator, generates with appropriate input model variant-specific code]

- Product Line tools

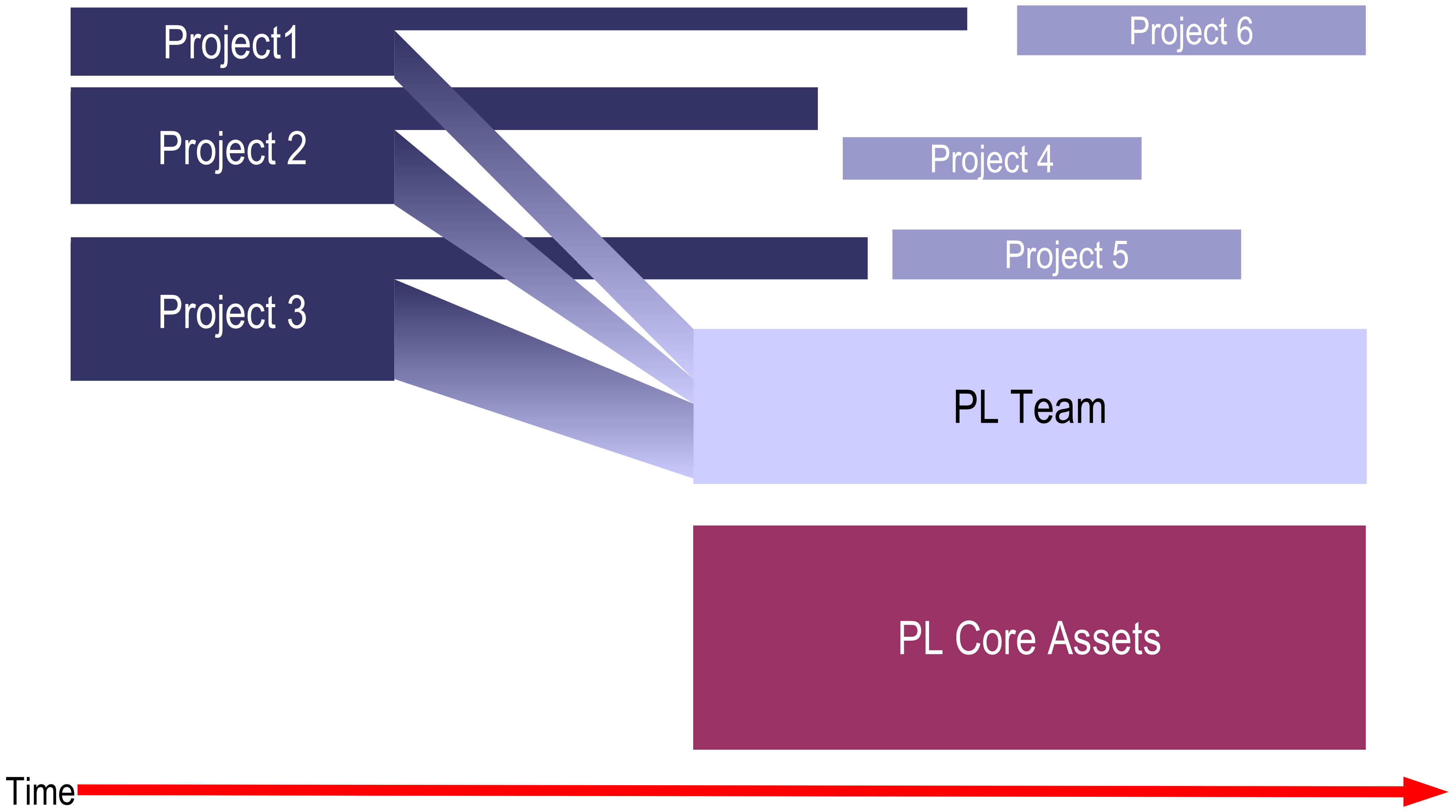
  pure::variants: explicit solution space modelling with integrated/external model transformation

Management Aspects of Software Product Lines

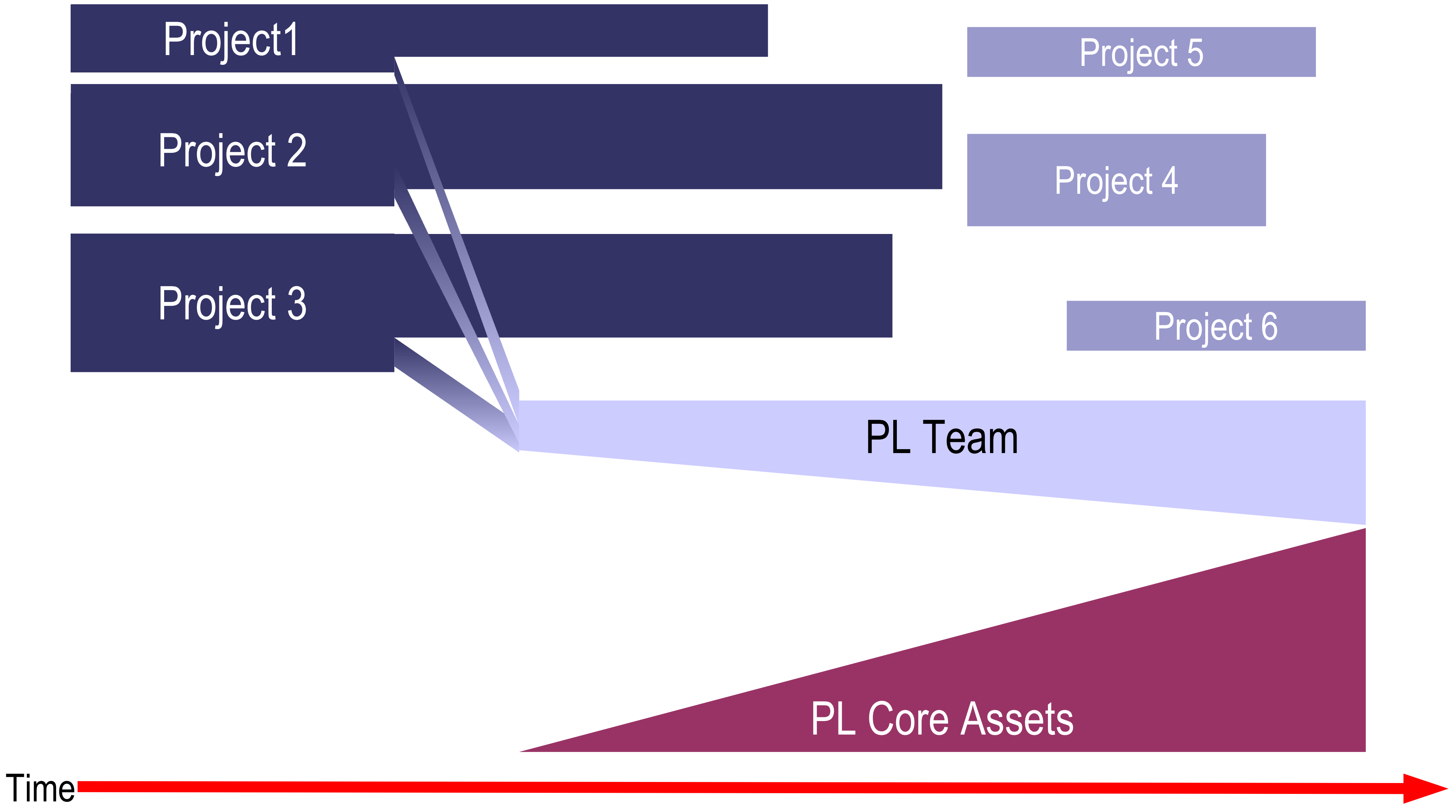
Transition to Variant Management

- Transistion does not happen in a single moment, its a process
- Choosing the right approach depends on
  - history and expected future
  - available funding
  - required time to market
  - skill level in the organisation

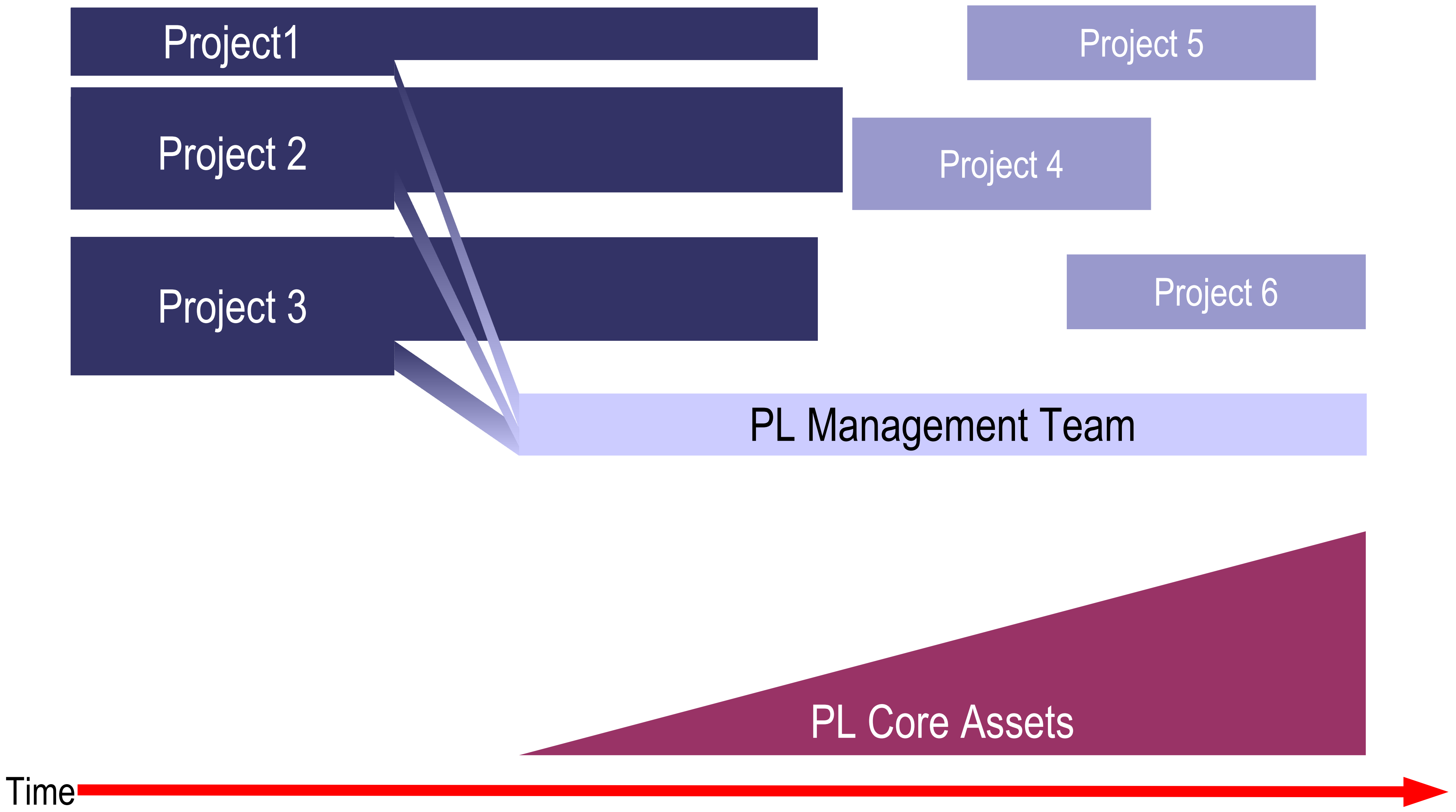
Approaches: Platform-Centric



Approaches: Incremental Platform-Centric



Approaches: Project-Centric



Approach Comparison

Platform-Centric

+ Pro:
  - lowest influence from projects
  - no legacy

- Con:
  - organisational impact
  - slow start
  - highest risk

Incremental Platform

+ Pro:
  - limited risk
  - incremental organisational changes

- Con:
  - slow reuse increase

Project-Centric

+ Pro:
  - lowest risk
  - only small organisational changes
  - instant ROI

- Con:
  - requires best reuse discipline
  - legacy system reuse

Summary

- Avoid variability, compare cost of variation to benefit
- If it is economically feasible/necessary to introduce variability make variation points and variants first level elements of your approach
- Explicit variant and variability management is the key to sucessful reuse
- Don't try to do all at once, incremental adoption lowers both effort and risk
- If you do it for the first time, ask someone who has done this before

Summary

- Transforming existing legacy software into product lines is possible
  - before starting a transition, careful risk assessment is necessary
- Results depend on:
  - amount and quality of extractable knowledge,
  - skills and management support of transition team.

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::variants

Linking Variants with Defects and Tests

The Challenge

In variant rich systems not every test or defect is relevant for all variants.

The following questions have to be answered:

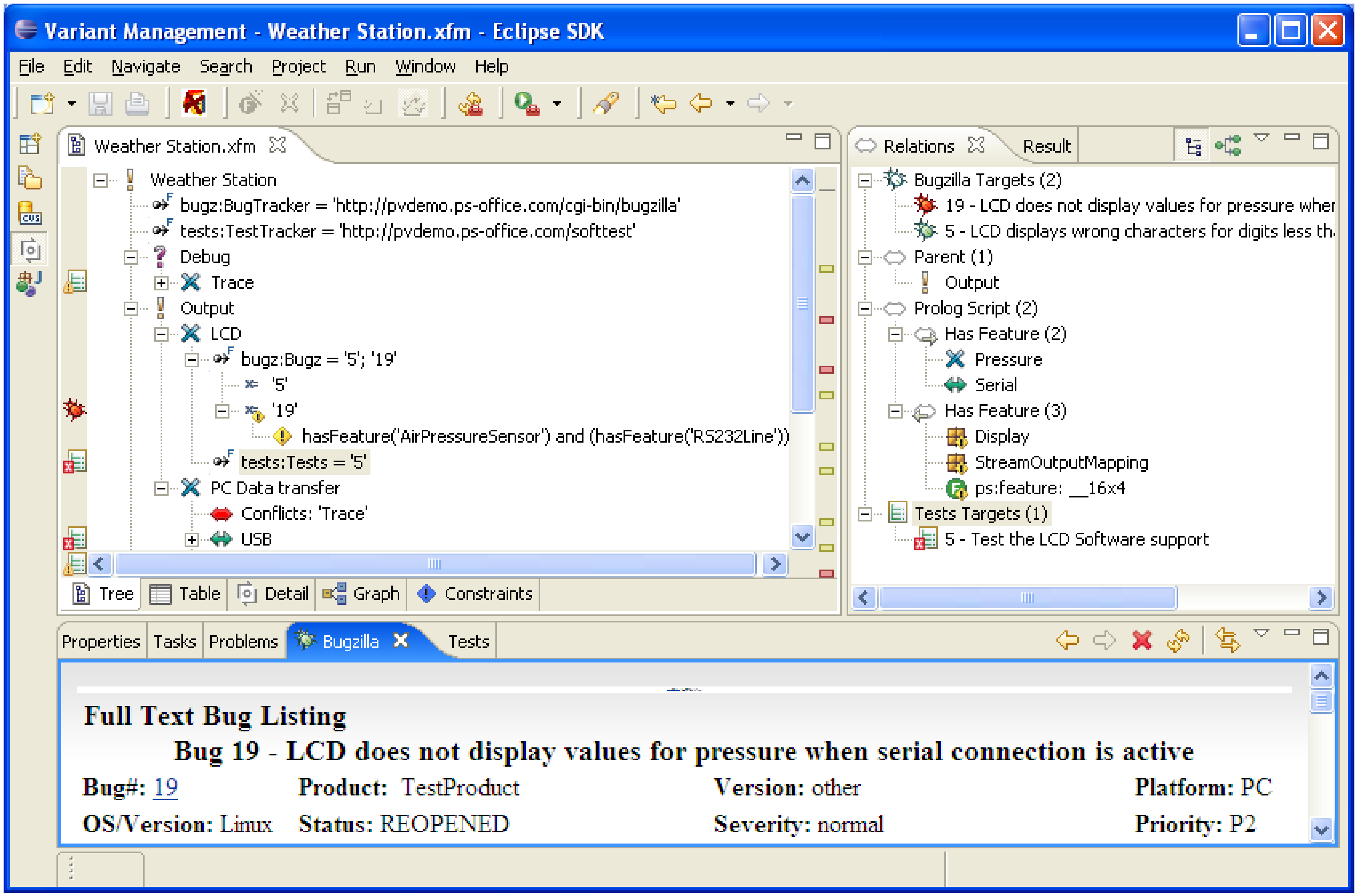
- Which tests, defects, change requests are relevant for which variants?
- What is the current development state regarding those items for the individual variants?

The approach should be applicable to existing databases and tools.

The pure::variants Approach

- pure::variants is used to manage the variability of the products. Each relevant product is described by a variant model.
- Each pure::variants element (feature, component, part, source) can be linked to external items using target list attributes.
- For each to be connected tool or database a pure::variants connector plugin resolves theses links on demand in real-time.
- Depending on the tool the connector can also provide actions like creation of a new defect or test case from within pure::variants.

pure::variants – Sample Workspace


Variant Management - Weather Station.xfm - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Weather Station.xfm
Relations
Result
Tree
Table
Detail
Graph
Constraints
Properties
Tasks
Problems
Bugzilla
Tests

```
Weather Station
    bugz:BugTracker = 'http://pvdemo.ps-office.com/cgi-bin/bugzilla'
    tests:TestTracker = 'http://pvdemo.ps-office.com/softtest'
    Debug
        Trace
    Output
        LCD
            bugz:Bugz = '5'; '19'
                '5'
                '19'
                    hasFeature('AirPressureSensor') and (hasFeature('RS232Line'))
            tests:Tests = '5'
        PC Data transfer
            Conflicts: 'Trace'
            USB
```

```
Bugzilla Targets (2)
    19 - LCD does not display values for pressure wher
    5 - LCD displays wrong characters for digits less th
Parent (1)
    Output
Prolog Script (2)
    Has Feature (2)
        Pressure
        Serial
    Has Feature (3)
        Display
        StreamOutputMapping
        ps:feature: __16x4
Tests Targets (1)
    5 - Test the LCD Software support
```

Full Text Bug Listing

Bug 19 - LCD does not display values for pressure when serial connection is active

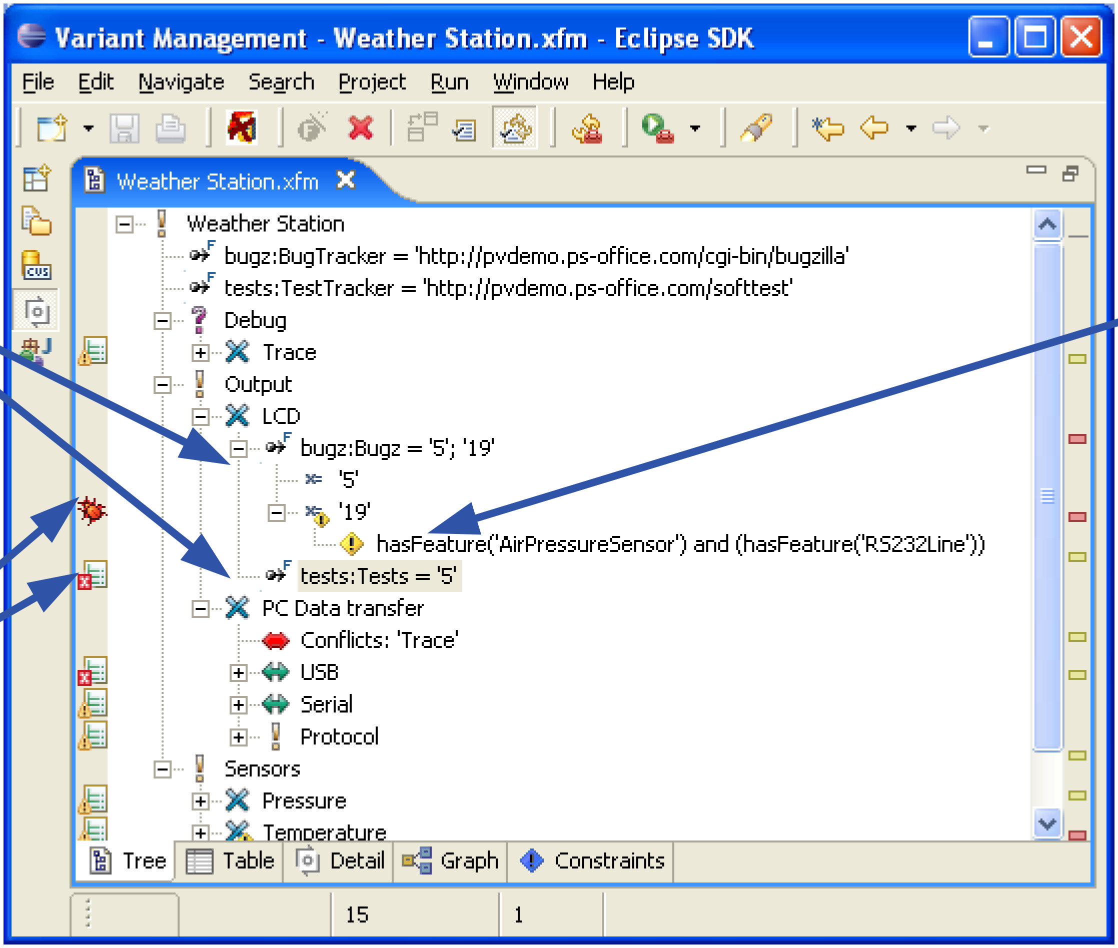
Bug#: 19	**Product:** TestProduct	**Version:** other	**Platform:** PC
OS/Version: Linux	**Status:** REOPENED	**Severity:** normal	**Priority:** P2

Model with Test and Defect Information

Elements are linked to test case or defects using special list attributes

Markers show current defect/test state of elements (here an open bug and a failing test case). Succeeded tests and resolved defects are not shown

Restrictions can be used to make a defect or test link valid only under defined conditions (here the selection of two features)



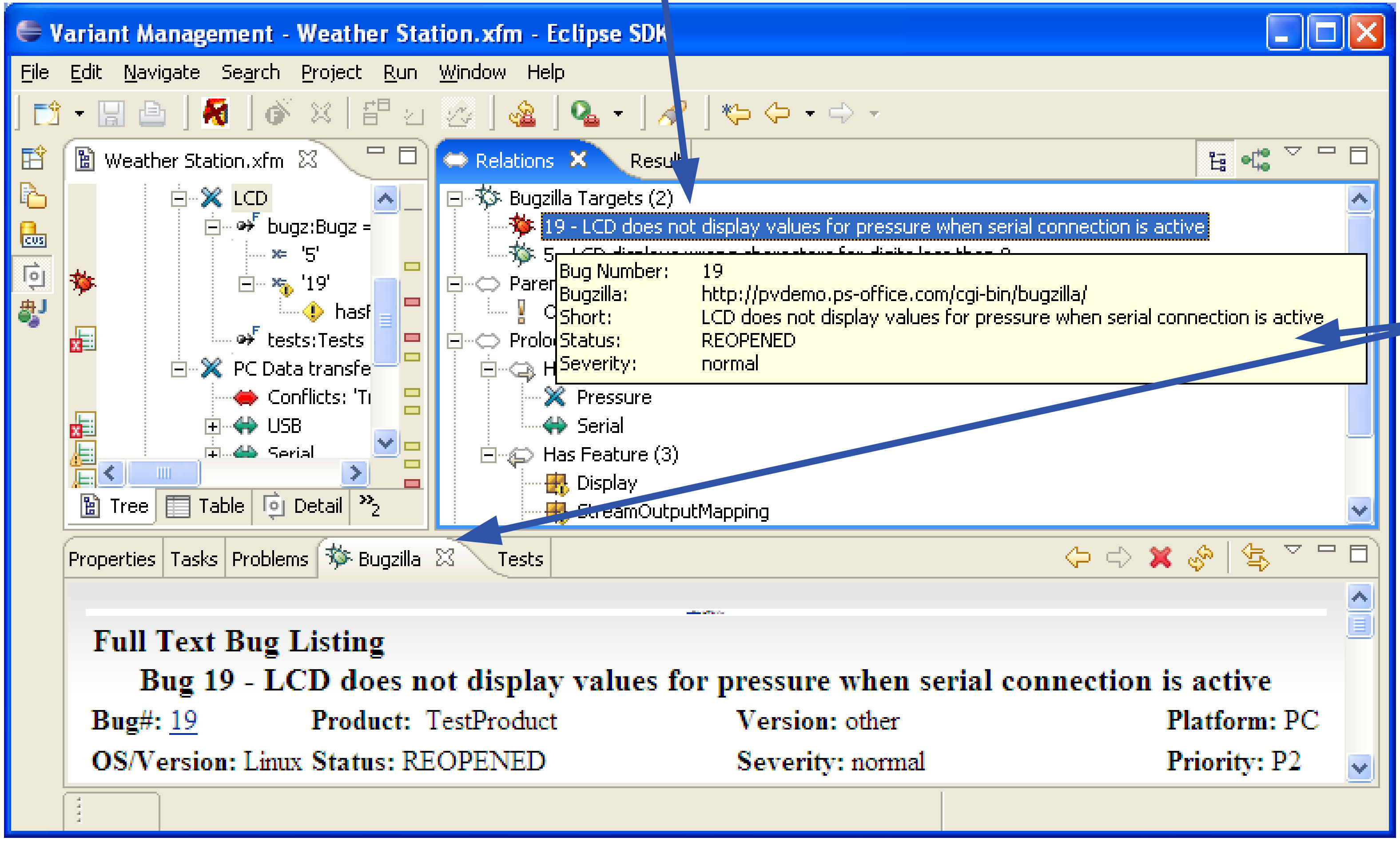
Variant Management - Weather Station.xfm - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Weather Station.xfm
Tree
Table
Detail
Graph
Constraints
15
1

Weather Station
bugz:BugTracker = 'http://pvdemo.ps-office.com/cgi-bin/bugzilla'
tests:TestTracker = 'http://pvdemo.ps-office.com/softtest'
Debug
Trace
Output
LCD
bugz:Bugz = '5'; '19'
'5'
'19'
hasFeature('AirPressureSensor') and (hasFeature('RS232Line'))
tests:Tests = '5'
PC Data transfer
Conflicts: 'Trace'
USB
Serial
Protocol
Sensors
Pressure
Temperature

Relations View – Quick Target Info Access

Relations view lists related relation targets for the selected element. For defects and tests the summary is shown

Additional information is accessible via the tool tip and optionally in specialized views.


Variant Management - Weather Station.xfm - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Weather Station.xfm
Relations
Result
Bugzilla Targets (2)
Tree Table Detail
Properties Tasks Problems Bugzilla Tests

19 - LCD does not display values for pressure when serial connection is active

```
Bug Number:   19
Bugzilla:     http://pvdemo.ps-office.com/cgi-bin/bugzilla/
Short:        LCD does not display values for pressure when serial connection is active
Status:       REOPENED
Severity:     normal
```

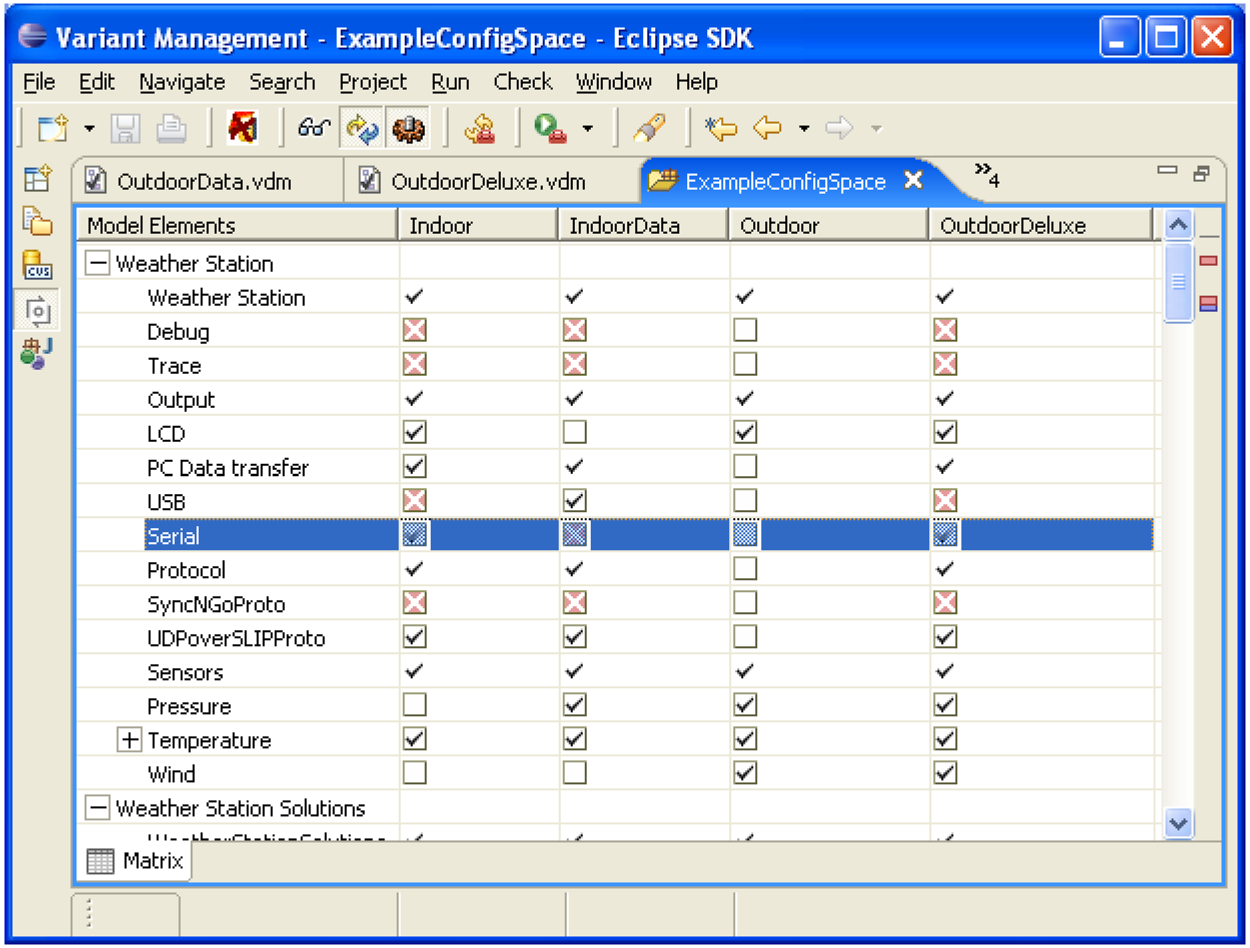
Full Text Bug Listing

Bug 19 - LCD does not display values for pressure when serial connection is active

Bug#: 19 Product: TestProduct Version: other Platform: PC
OS/Version: Linux Status: REOPENED Severity: normal Priority: P2

Matrix View – Element Selection States

- The Matrix view enables quick overview for multiple variants. Here the selection is shown (default visualization).
- Each column represents a variant, rows are selectable elements like features
- Other visualizations can show variant specific defect and test states.

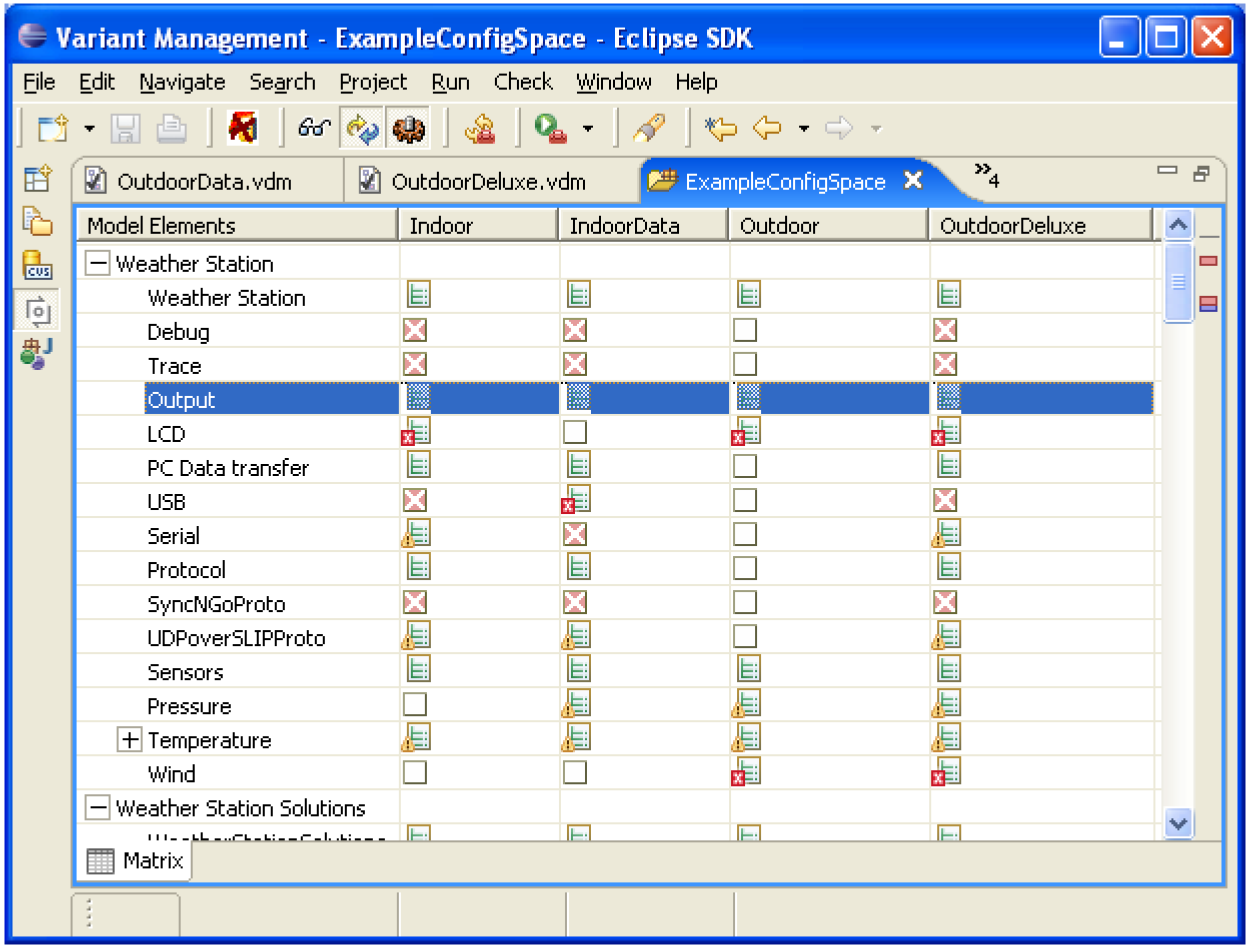


Variant Management - ExampleConfigSpace - Eclipse SDK
File Edit Navigate Search Project Run Check Window Help
OutdoorData.vdm
OutdoorDeluxe.vdm
ExampleConfigSpace
Matrix

Model Elements	Indoor	IndoorData	Outdoor	OutdoorDeluxe
⊟ Weather Station				
Weather Station	✓	✓	✓	✓
Debug	☒	☒	☐	☒
Trace	☒	☒	☐	☒
Output	✓	✓	✓	✓
LCD	☑	☐	☑	☑
PC Data transfer	☑	✓	☐	✓
USB	☒	☑	☐	☒
Serial	▨	▨	▨	▨
Protocol	✓	✓	☐	✓
SyncNGoProto	☒	☒	☐	☒
UDPoverSLIPProto	☑	☑	☐	☑
Sensors	✓	✓	✓	✓
Pressure	☐	☑	☑	☑
⊞ Temperature	☑	☑	☑	☑
Wind	☐	☐	☑	☑
⊟ Weather Station Solutions				

Matrix View – Element Test States

- The test state visualization shows for all selected features the success of the related tests (indicated by different icons).
- It is visible here that not all variants have the same set of tests.

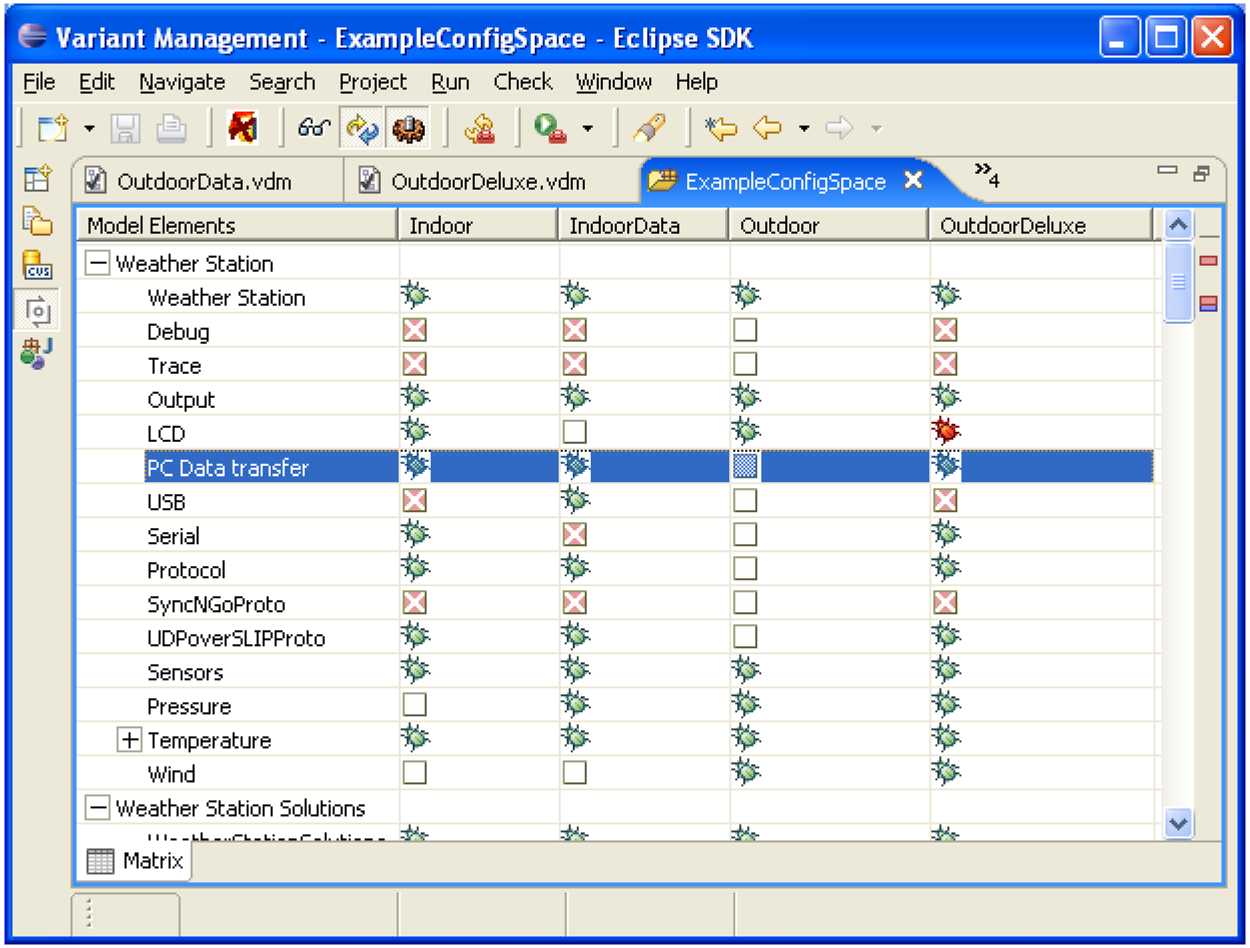


Variant Management - ExampleConfigSpace - Eclipse SDK
File Edit Navigate Search Project Run Check Window Help
OutdoorData.vdm
OutdoorDeluxe.vdm
ExampleConfigSpace
Matrix

Model Elements	Indoor	IndoorData	Outdoor	OutdoorDeluxe
− Weather Station				
Weather Station	▤	▤	▤	▤
Debug	☒	☒	☐	☒
Trace	☒	☒	☐	☒
Output	▨	▨	▨	▨
LCD	▤☒	☐	▤☒	▤☒
PC Data transfer	▤	▤	☐	▤
USB	☒	▤☒	☐	☒
Serial	▤⚠	☒	☐	▤⚠
Protocol	▤	▤	☐	▤
SyncNGoProto	☒	☒	☐	☒
UDPoverSLIPProto	▤⚠	▤⚠	☐	▤⚠
Sensors	▤	▤	▤	▤
Pressure	☐	▤⚠	▤⚠	▤⚠
+ Temperature	▤⚠	▤⚠	▤⚠	▤⚠
Wind	☐	☐	▤☒	▤☒
− Weather Station Solutions				

Matrix View – Element Defect States

- The defect state visualization shows for all selected features the existence of the related open defects (indicated by different icons).
- It is visible here that not all variants have the same set of defects.



Model Elements	Indoor	IndoorData	Outdoor	OutdoorDeluxe
⊟ Weather Station				
Weather Station	🐛	🐛	🐛	🐛
Debug	☒	☒	☐	☒
Trace	☒	☒	☐	☒
Output	🐛	🐛	🐛	🐛
LCD	🐛	☐	🐛	🐞
PC Data transfer	🐛	🐛	▨	🐛
USB	☒	🐛	☐	☒
Serial	🐛	☒	☐	🐛
Protocol	🐛	🐛	☐	🐛
SyncNGoProto	☒	☒	☐	☒
UDPoverSLIPProto	🐛	🐛	☐	🐛
Sensors	🐛	🐛	🐛	🐛
Pressure	☐	🐛	🐛	🐛
⊞ Temperature	🐛	🐛	🐛	🐛
Wind	☐	☐	🐛	🐛
⊟ Weather Station Solutions				

pure::variants – Extension Availability

- The Connector API has been published with release 2.4
  - The API is available for public use by third parties
- pure-systems provides the following Connectors as commercial products for release 2.4:
  - pure::variants Connector for ClearQuest
  - pure::variants Connector for Bugzilla

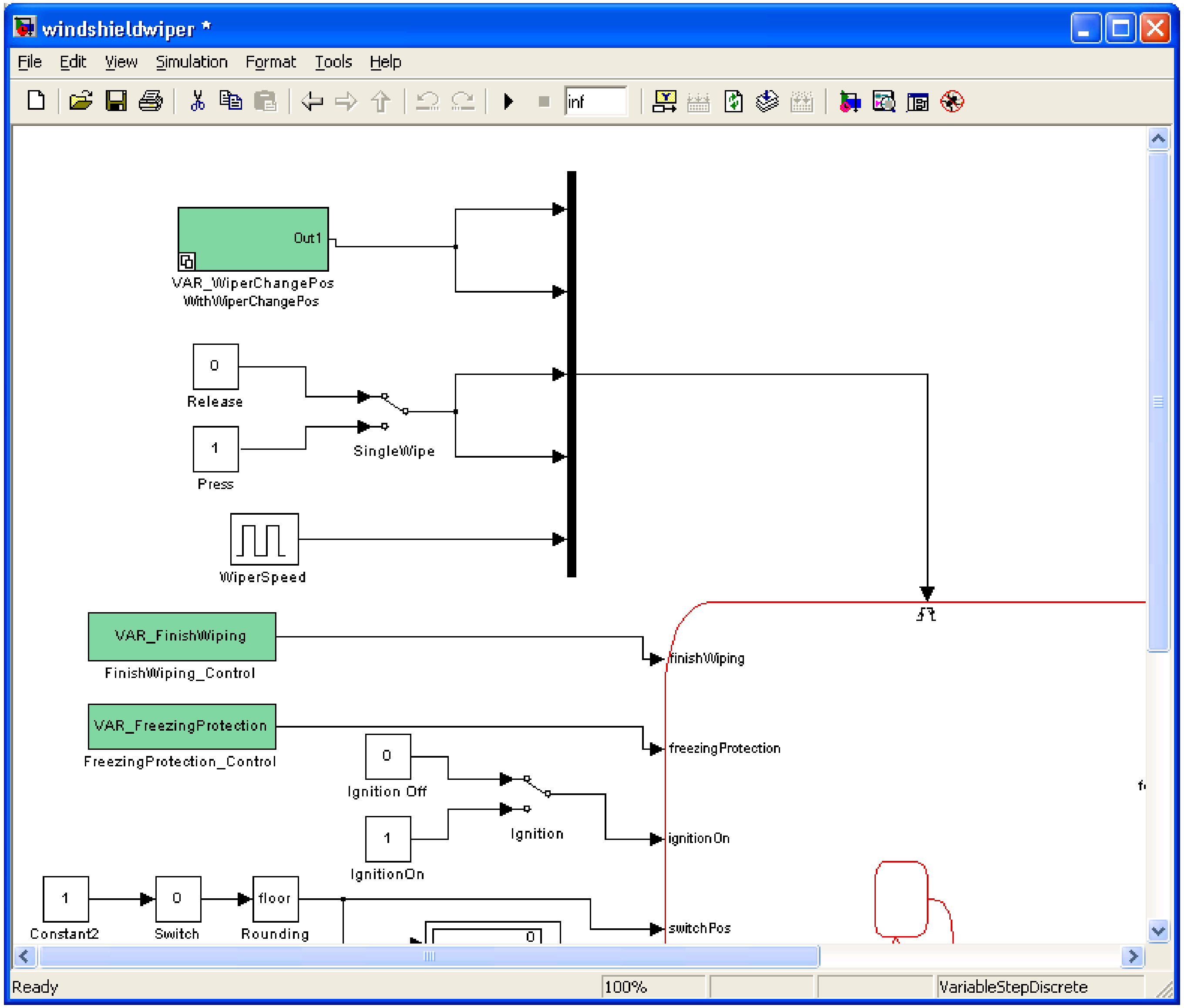
pure::variants

Connecting pure::variants with Model-based Tools

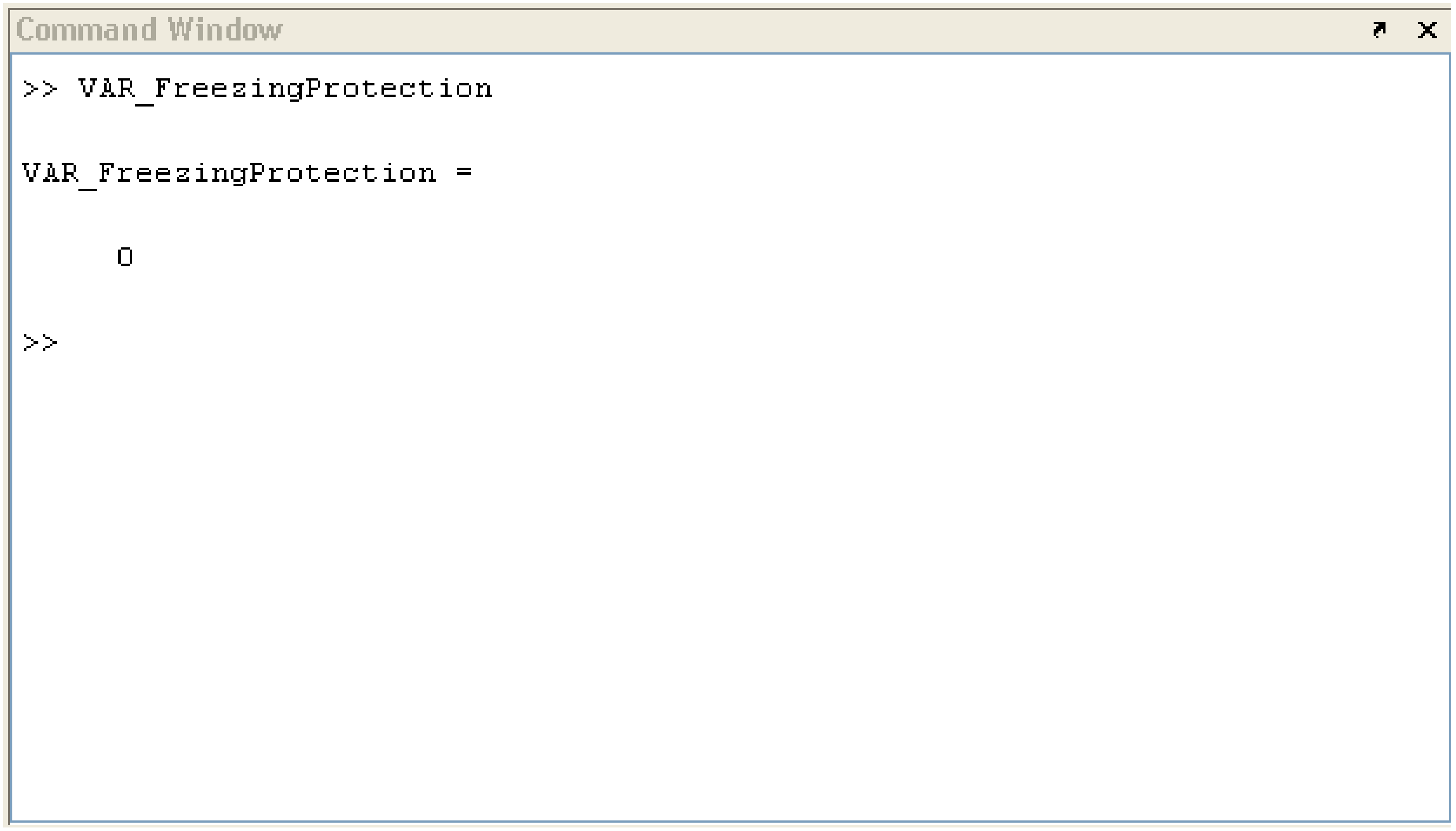
Connection Approaches

- Read and Generate
  - External model is fully read into pure::variants model
  - pure::variants allows for modification and addition of rules etc.
  - pure::variants generates variant-specific external model from p::v model
  - Examples: Connector for Simulink
- Link and Communicate
  - only variation point related information is extracted from external source
    - possibly most information is entered in external tool
  - pure::variants actuates variation points during configuration
  - relevant configuration for tool is communicate to tool (online/offline)
  - Examples: Connector for Doors, new Simulink Configurator

Simulink Model with Variation Points



Simulink Variation Point Setting



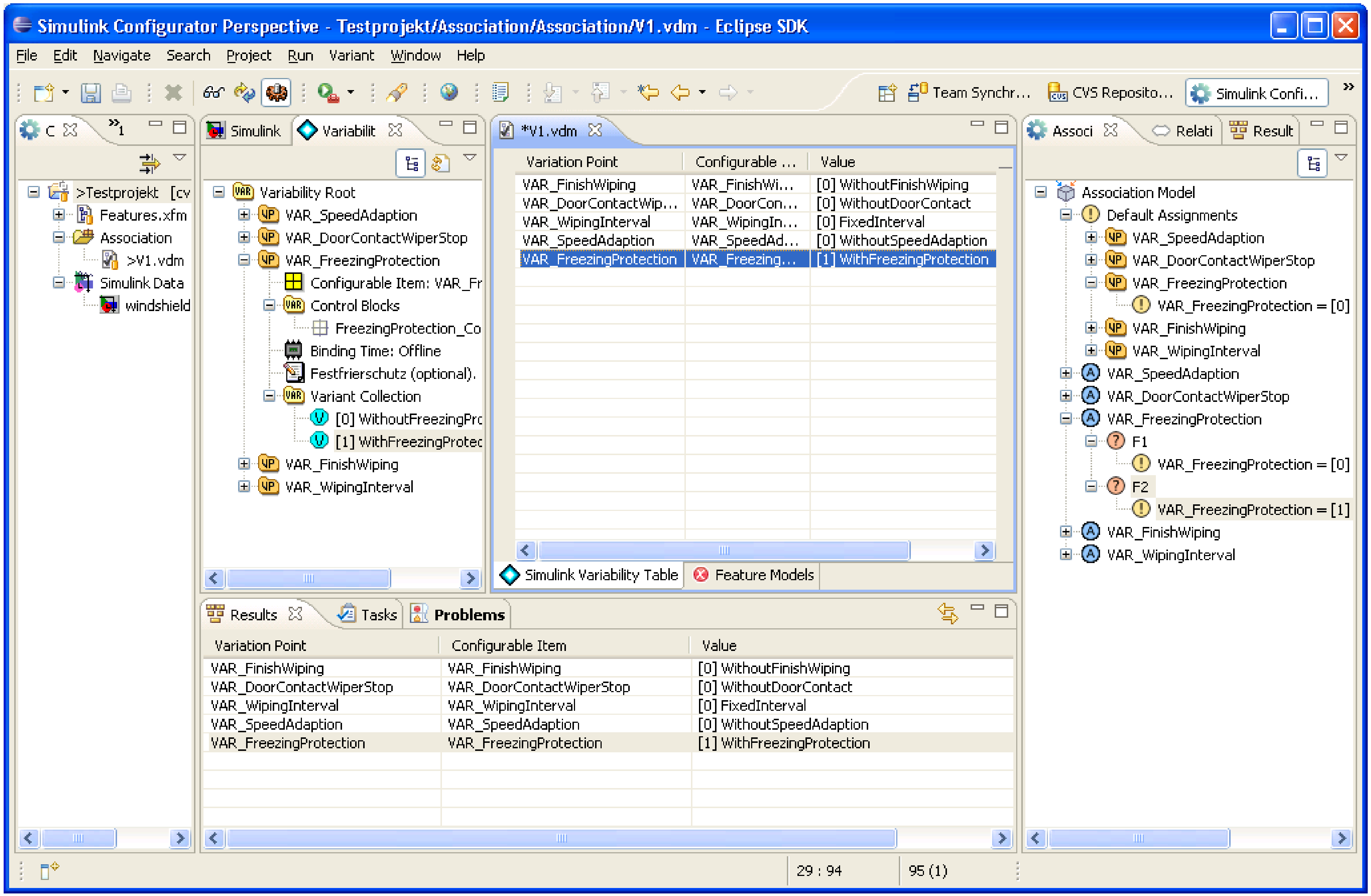
```
>> VAR_FreezingProtection

VAR_FreezingProtection =

     0

>>
```

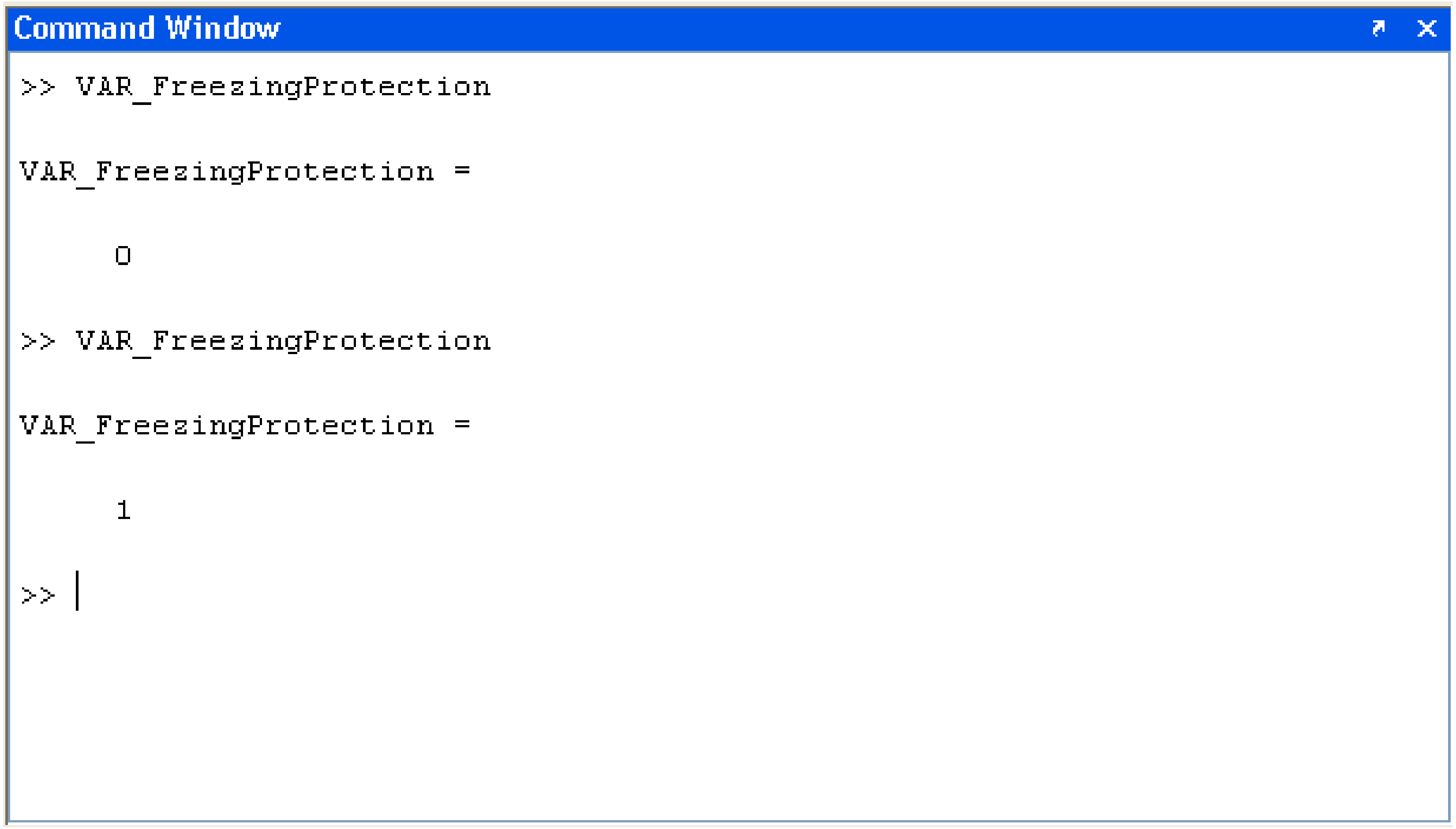
pure::variants based Simulink Configurator



Variation Point	Configurable ...	Value
VAR_FinishWiping	VAR_FinishWi...	[0] WithoutFinishWiping
VAR_DoorContactWip...	VAR_DoorCon...	[0] WithoutDoorContact
VAR_WipingInterval	VAR_WipingIn...	[0] FixedInterval
VAR_SpeedAdaption	VAR_SpeedAd...	[0] WithoutSpeedAdaption
VAR_FreezingProtection	VAR_Freezing...	[1] WithFreezingProtection

Variation Point	Configurable Item	Value
VAR_FinishWiping	VAR_FinishWiping	[0] WithoutFinishWiping
VAR_DoorContactWiperStop	VAR_DoorContactWiperStop	[0] WithoutDoorContact
VAR_WipingInterval	VAR_WipingInterval	[0] FixedInterval
VAR_SpeedAdaption	VAR_SpeedAdaption	[0] WithoutSpeedAdaption
VAR_FreezingProtection	VAR_FreezingProtection	[1] WithFreezingProtection

Simulink Variation Point Setting



```
>> VAR_FreezingProtection

VAR_FreezingProtection =

     0

>> VAR_FreezingProtection

VAR_FreezingProtection =

     1

>> 
```