



# Concepts and Artefacts

**Jan Bosch**

*Professor of Software Engineering  
University of Groningen, Netherlands*

*[Jan@JanBosch.com](mailto:Jan@JanBosch.com)*

*<http://www.janbosch.com>*



## Overview

- maturity levels
- reused artefacts
- organizational models
- software variability management
- variation point

# Maturity Model - Four dimensions

FOCUS

- Business
- **Architecture**
- Process
- Organization

Based on *J. Bosch, Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization, Proceedings SPLC2, August 2002*

and working paper with Frank v.d. Linden, Erik Kamsties, Kari Käsäla, Lech Krzanik and Henk Obbink

## Aspect - Business

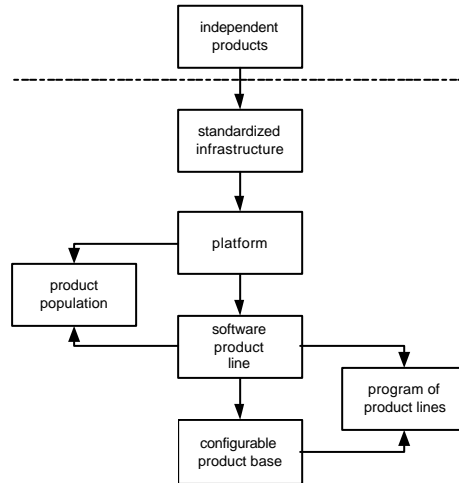
### Aspects

- Identity – integration between SPF and organization
- Vision – short, medium or long term perspective
- Objectives – none, qualitative, quantitative
- Strategic planning - e.g. roadmapping, ad-hoc to institutionalized process

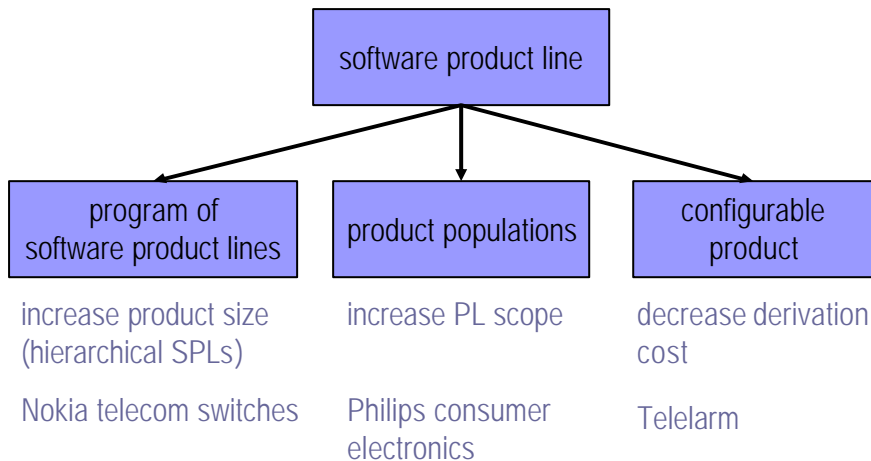
### Levels

- Reactive
- **Awareness**
- Extrapolate
- Proactive
- **Strategic**

# Architectural “Maturity” Levels



# Architectural “Maturity” Levels



# Independent Products

- *Product family architecture*: not established
- *Product quality*: ignored or managed in an ad-hoc fashion
- *Reuse level*: Although ad-hoc reuse may occur, there is no institutionalized reuse
- *Domain*: emerging
- *Software variability management*: absent
- *Example*: most SMEs and several large companies

# Standardized Infrastructure

- **description**
  - operating system
  - database
  - GUI
  - etc.

some additional proprietary glue code might be present



# Standardized Infrastructure

- *Product family architecture*: specified external components
- *Product quality*: infrastructure supports certain qualities, for the remaining qualities an over-engineering approach is used
- *Reuse level*: only external components
- *Domain*: later phases of emerging or the early phases of maturing domain
- *Software variability management*: limited variation points from the infrastructure components
- *Example*: Vertis Information Technology

# Platform

- **description:**
  - functionality common to all products in the product family is captured
  - infrastructure is typically also included

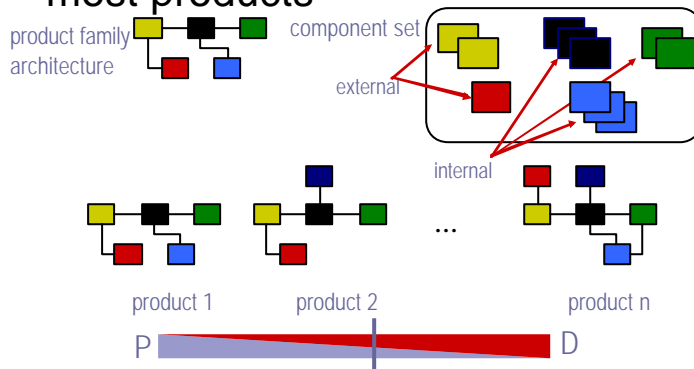


# Platform

- *Product family architecture*: only the features common to all products are captured
- *Product quality*: inherited from the platform
- *Reuse level*: reuse of internal platform components
- *Domain*: mature
- *Software variability management*: managed at platform level
- *Example*: Symbian OS

# Software Product Family

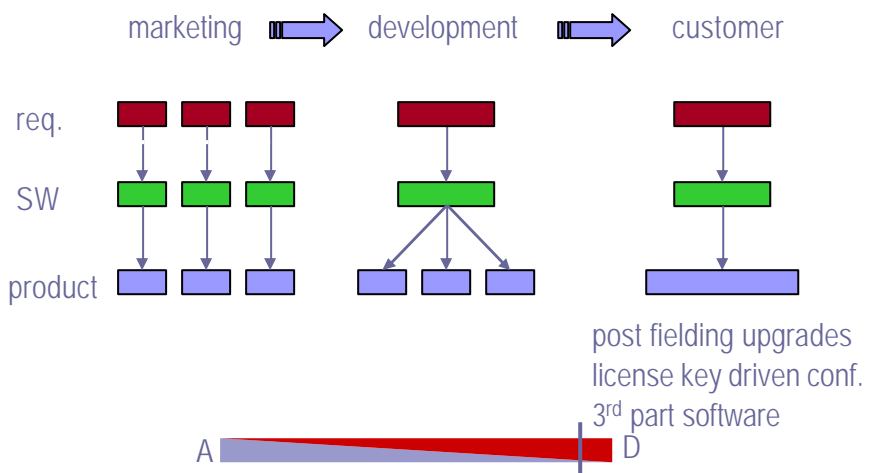
- **description**: domain assets capture functionality common to several or most products



# Software Product Family

- *Product family architecture*: fully specified
- *Product quality*: a key priority for development
- *Reuse level*: managed
- *Domain*: late stages of maturing or the early stages of the established phase
- *Software variability management*: many variation points and dependencies between them
- *Example*: Axis Communications AB

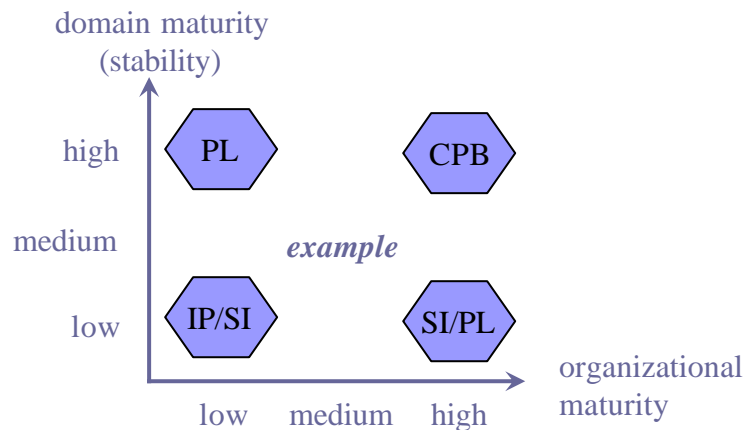
# Configurable Product (Base)



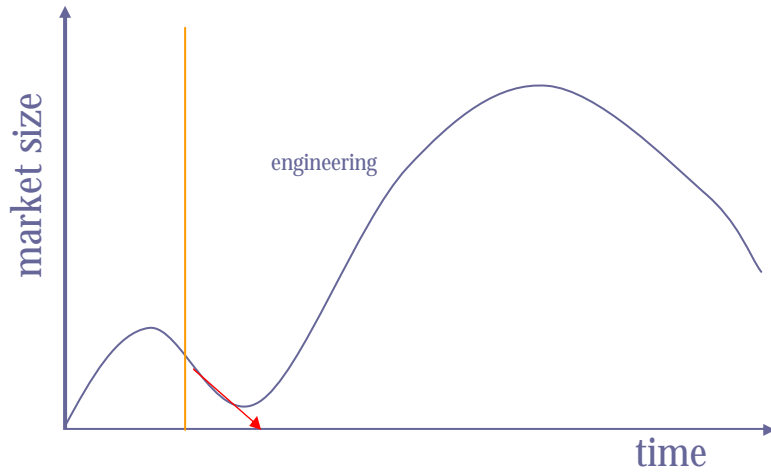
## Configurable Product (Base)

- *Product family architecture*: enforced
- *Product quality*: quality attributes are implemented as variation points in the architecture and components
- *Reuse level*: automatic generation of product family members
- *Domain*: established
- *Software variability management*: automated selection and verification of variants at variation points has been optimized
- *Example*: TeleLarm AB

## Two Dimensions of “Maturity”



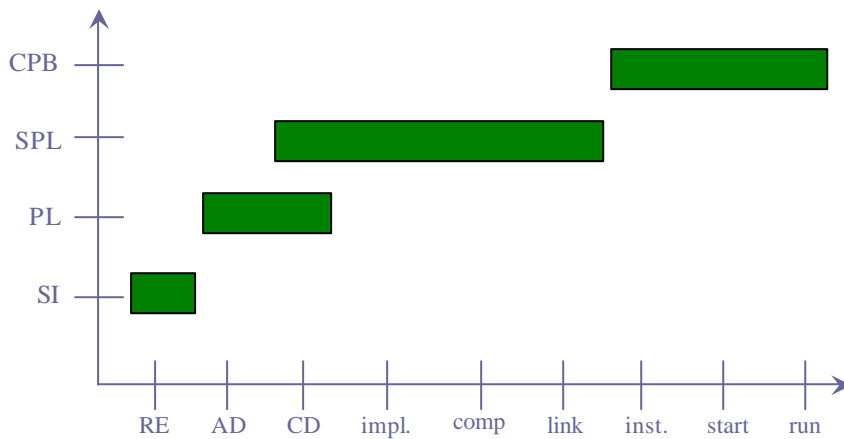
# Lifecycle



software variability management

17

# Variation Binding Times



software variability management

18

# Reused Artefacts

- architecture
  - under-specified architecture
  - specified architecture
  - enforced architecture

# Reused Artefacts

- components
  - specified component
  - multiple component implementations
  - configurable component implementation

# Reused Artefacts

- products
  - architecture conformance
  - platform-based product
  - configurable product base

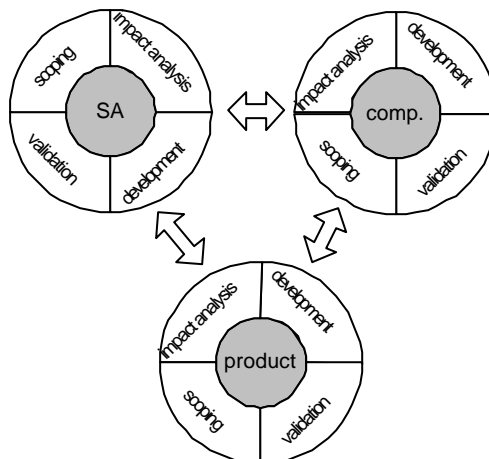
# Organizational Models

- development department
- business units
- domain engineering units
- hierarchical domain engineering units

# Relating levels and artefacts

	<i>standard infr.</i>	<i>platform</i>	<i>SPL</i>	<i>conf. prod. base</i>	<i>product popul.</i>	<i>program of SPLs</i>
u.spec. SA	+	+			+	
spec. SA			+		+	+
enf. SA				+		
spec. C.	+/-	+			+/-	
mult. C.I.		+/-	+		+	+/-
conf. C.			+/-	+		+
arch. conf.	+					
platf. b. P		+	+		+	
conf. P base				+		+
dev. dept.	+	+	+			
bus. units	+	+	+			
D.E. unit			+	+	+	
hier. DE units					+	+

# Evolution Processes



# Software Variability Management

Variability needs in software are constantly increasing

because

- variability moves from mechanics and hardware to software
- design decisions are delayed as long as economically feasible

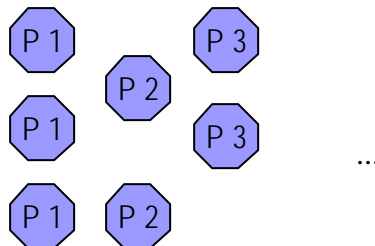
# Variability

- software variability is the ability of a software system or artefact to be extended, changed, customized or configured for use in a particular context
- software variability reflects problem domain variation, e.g. expressed through feature models
- software variability is expressed through variation points
- variation points delay design decisions

# Variation Points

- variation point in the lifecycle
  - implicit – implicitly present in system
  - designed – delayed design decision lead to VP
  - bound – variation has been connected to VP
    - rebindable – variant can be bound to different variants
    - permanent – variant has been bound permanently
  - times
  - open – possible to add new variants to set
  - closed – set of variants is closed
- can be introduced and bound at any level

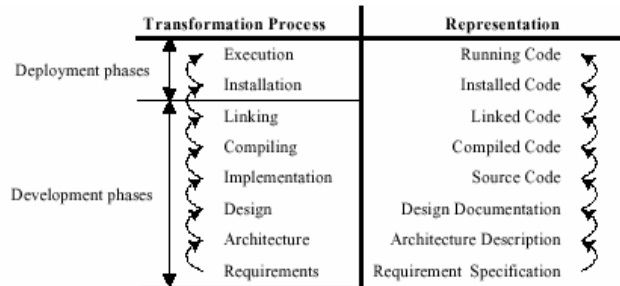
# Variability



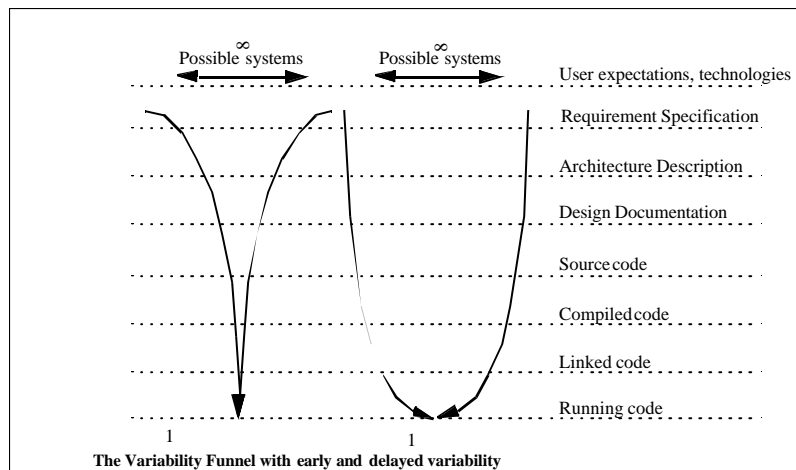
software product line should

- support variability in space (products)
- support 'variability' in time (evolution)
- not prohibit product specific variations

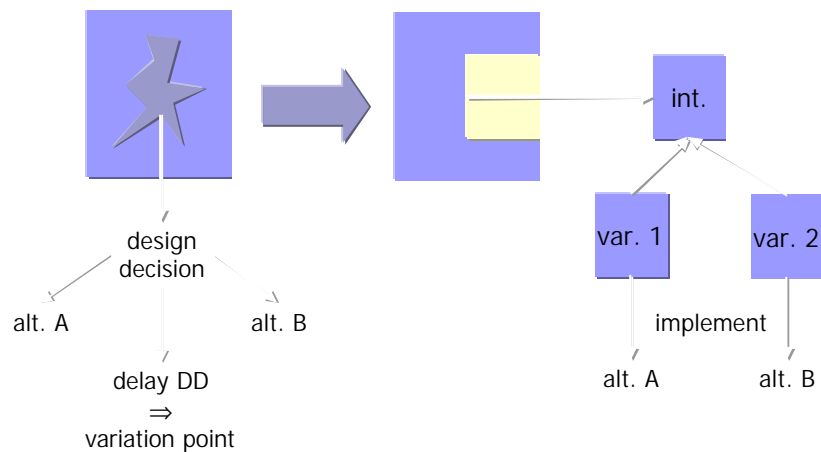
# Transformation Process



# Variability



# Variation Points: Basic Mechanism



# Variation Points

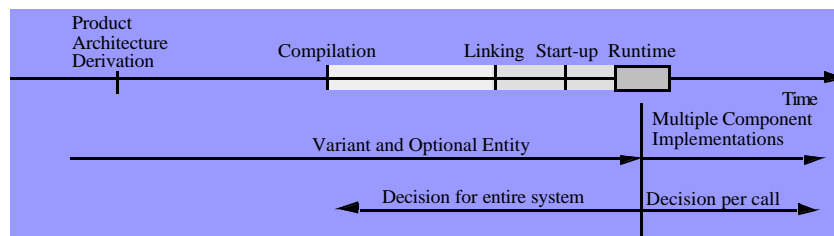
- variation point in the lifecycle
  - implicit – implicitly present in system
  - designed – delayed design decision lead to VP
  - bound – variation has been connected to VP
    - rebindable – variant can be bound to different variants
    - permanent – variant has been bound permanently
- times
- open – possible to add new variants to set
- closed – set of variants is closed
- can be introduced and bound at any level

# Examples of Mechanisms

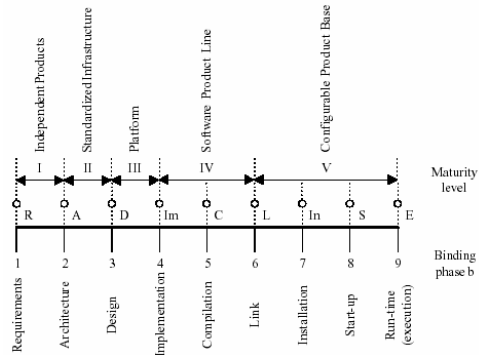
- architecture level  $\Rightarrow$  presence/absence architectural component
  - component level  $\Rightarrow$  selection of component implementation for architectural component  
 $\Rightarrow$  selection of plug-ins for component implementations
  - component implementation level  $\Rightarrow$  set appropriate parameters
- (see our paper for a taxonomy of mechanisms)

# Binding Times

- binding can be performed
  - before delivery – software product families
  - post delivery – configurable/dynamic products



# Binding Time



Maturity level	Minimum binding phase $b_{\min}$	Maximum binding phase $b_{\max}$	Minimum degree of dynamicity in per cent (approx.)	Maximum degree of dynamicity in per cent (approx.)	Difference in dynamicity in per cent (approx.)
Independent Products	1	2	0	11	11
Standardized Infrastructure	2	3	11	22	11
Platform	3	4	22	33	11
Software Product Line	4	6	33	55	22
Configurable Product Base	6	9	55	100	45

software variability management

35

# Research communities

who study variability?

- design community, e.g. design patterns, MDSOC
- configuration management community
- dynamic software architecture community

but only for part of the life cycle!

i.e. design, compilation, run-time

software variability management

36

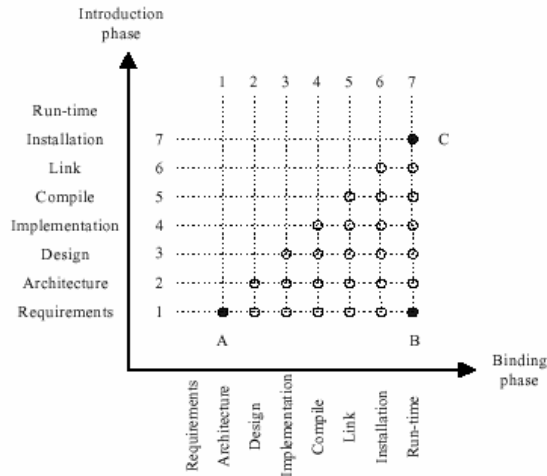
# SVM Research in Groningen

- conceptual framework
- visualizing variability and variability dependencies
- explicit modelling of architectural design decisions
- SVM in product derivation
- variability assessment
- case studies

# Comparison between patterns

Characteristic	Variant Entity	Optional Entity	Multiple Coexisting Entity
Management	Separate from Call	Separate from Call	Performed in Call
Scope of Binding	Valid for Entire System	Valid for Entire System	Valid for one Call
Collection	Implicit or Explicit	Not Applicable	Explicit
Binding	External or Internal	External or Internal	Internal
Open and Closed	Depends on runtime environment	Immediately closed	Depends on runtime environment

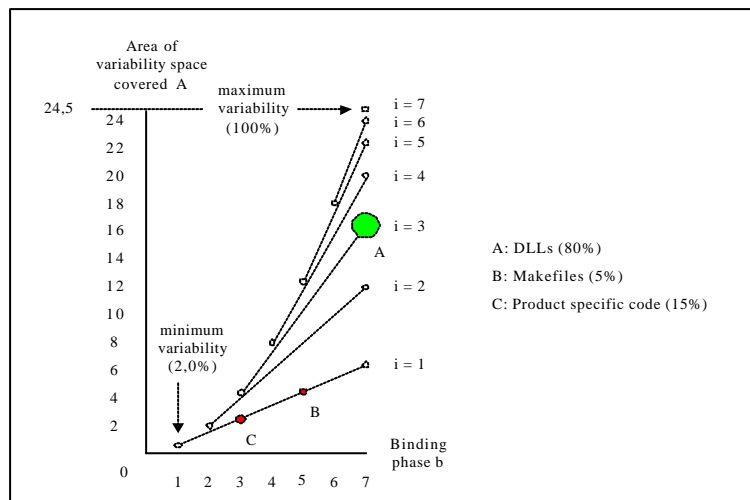
# Visualizing Variability



software variability management

39

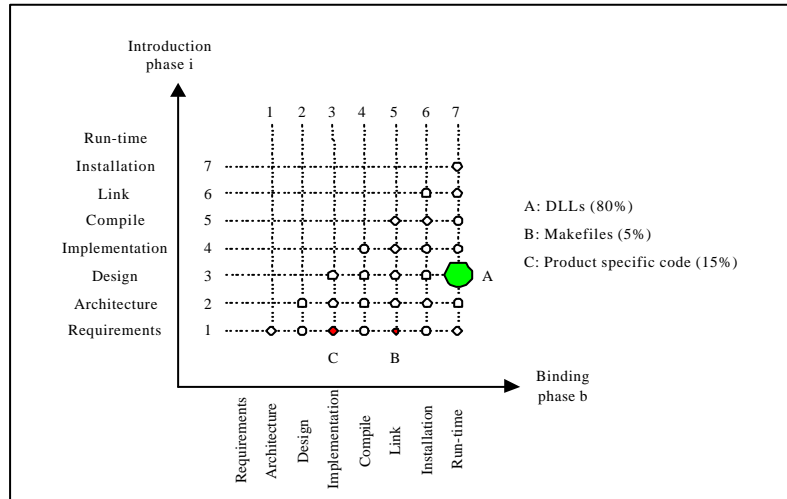
# Visualizing Variability



software variability management

40

# Rohill Technologies



software variability management

41

# Philips Medical Systems

Concern	Analysis
Variability identification	<ul style="list-style-type: none"> <li>Variability is identified in a functional decomposition of the MRI system resulting in different layers of abstraction, i.e., the so-called building block method, which has an emphasis on the construction of the software architecture. Each block provides a viewpoint on the system from an abstract (documentation), implementation (software) and project management perspective (tracking the development process).</li> <li>The building block structure generally follows a decomposition from a hardware perspective to identify hard- and software components (also for functionality that is mainly software-based).</li> <li>An implicit distinction is made between hardware neutral variability (e.g., multiple language support) and hardware enforced variability (e.g., hardware drivers and parts of the clinical viewing packages).</li> <li>Hardware neutral variability is identifiable in the form of distinct variants prior to hardware enforced variability in the development process, i.e., during design and implementation, respectively, but both types are bound simultaneously during system start-up.</li> </ul>

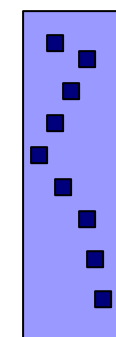
software variability management

42

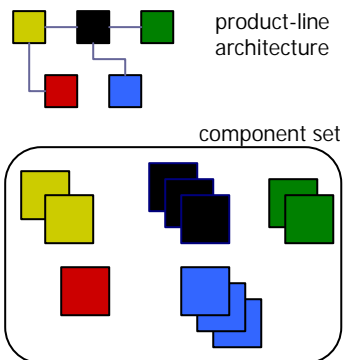
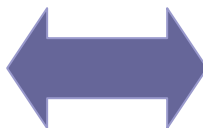
# Philips Medical Systems (cntd.)

Variability dependencies	<ul style="list-style-type: none"> <li>• Hierarchical (part-of relationships) and dependency aspects (use relationships) characterize each viewpoint in the building block method and are dealt with independently from each other.</li> <li>• As opposed to hardware enforced variability, the hierarchical and dependency aspects of hardware neutral variability are often less explicit.</li> <li>• Hardware neutral variability generally does not rely on hardware enforced variability and vice versa.</li> </ul>
Concern	<b>Analysis</b>
Tool support	<ul style="list-style-type: none"> <li>• Several tools and techniques have been developed in-house for supporting the building block method and for configuration validation (configuration validation is done independently from the building block structure).</li> <li>• Information from the perspective of each viewpoint is readily available for each building block by selecting the appropriate block from the tree hierarchy (department-wide through an intranet).</li> <li>• Variability mechanisms are used independently from each other, but sometimes act on the same variants.</li> </ul>

# Assessment



required variability



implemented variability



# Outline

- problems and issues
- COVAMOF - notation
- COSVAM - assessment
- design for variability