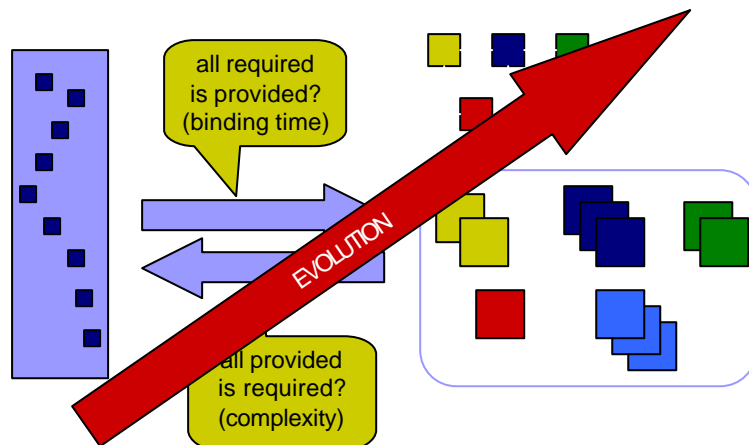


# COVAMOF Software Variability Assessment Method

**Jan Bosch**  
Professor of Software Engineering  
University of Groningen, Netherlands  
Jan@JanBosch.com  
<http://www.janbosch.com>

## Assessment

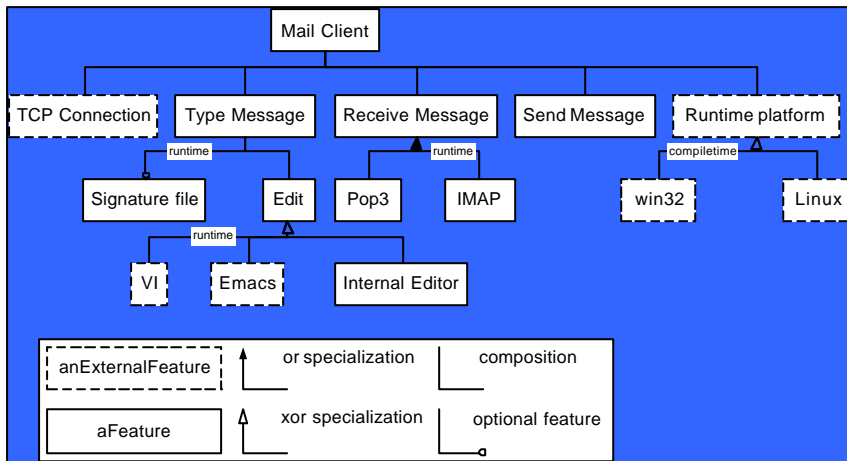


# Variability Assessment

- required variability
  - space dimension
  - time dimension
- provided variability

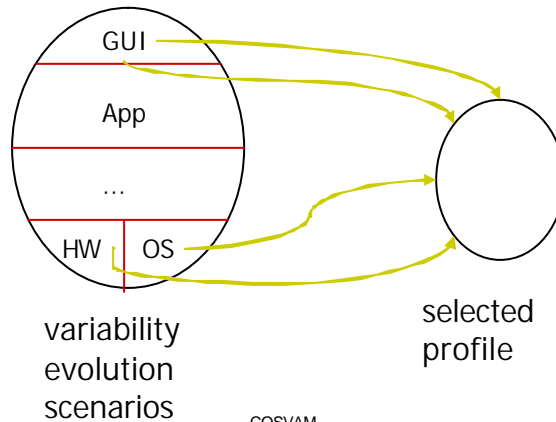
# Notation: required variability

- space dimension: feature models



# Required variability

- time dimension: scenario profiles



5

# Variability Scenario

- notion of equivalence class
- scenario categories
  - generic versus system specific QA specification
- generic – e.g. McCabe's cyclomatic complexity metric
- specific – scenario profiles

COSVAM

6

# Scenario Profiles

- top-down or bottom-up
- top-down profile development
  - (pre-)define scenario categories
  - selection and definition of scenarios for each category
  - each scenario is assigned a weight (either based on historical data or estimated)

# Scenario Profile Development

- bottom-up profile development
  - interview stakeholders
  - categorize scenarios
  - assign weights to scenarios
  - iterate until sufficient coverage
- stopping criterion
  - coverage

# Variability Profile - Space

variability profile combines

- currently required variability (space)
- variability expected required in future (time)

set of required variabilities

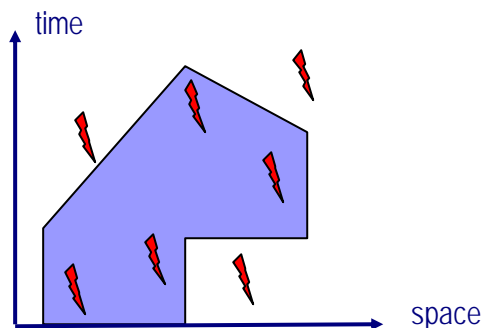
- complete set
- selected set
  - random
  - criticality
  - perceived implementation complexity

COSVAM

9

# Variability Profile - Time

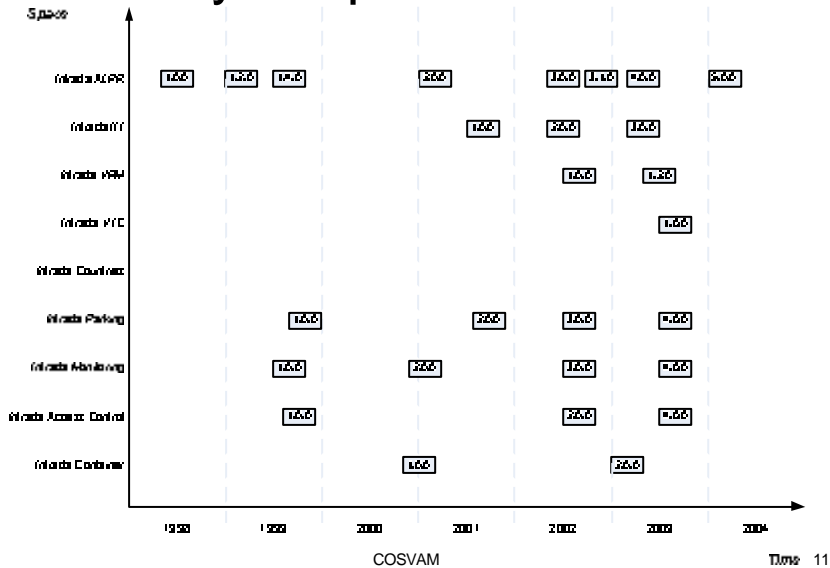
- notion of timeline: the closer the needed variability, the higher the chance.
- effort for a scenario:  
chance \* effort
- statistical assessment!



COSVAM

10

# Variability in space and time:



## Scenario Profiles - QAs

- performance: usage profile
- maintainability: maintenance profile
- variability: variation profile
- reliability: usage profile
- safety: hazard profile
- security: authorization profile

# Elicitation Techniques

- maintenance effort: scenarios that correspond to variability changes with high likelihood
- risk assessment: scenarios that search for difficult/impossible variability changes
- comparison: scenarios that highlight differences

# Provided Variability

notation

- configuration interface
- tabular representation

| Parameter           | Meaning                                   | Value Space | Binding Time               | Default |
|---------------------|---|-------------|----------------------------|---------|
| P5: MaxSensorPeriod | Maximum sensor period                     | [1..600]    | Translator<br>Construction | 600     |
| P6: MaxSensors      | Maximum number of sensors on board an FWS | [2..20]     | Translator<br>Construction | 20      |
| P7:                 | Maximum sensor period                     |             | Translator                 |         |

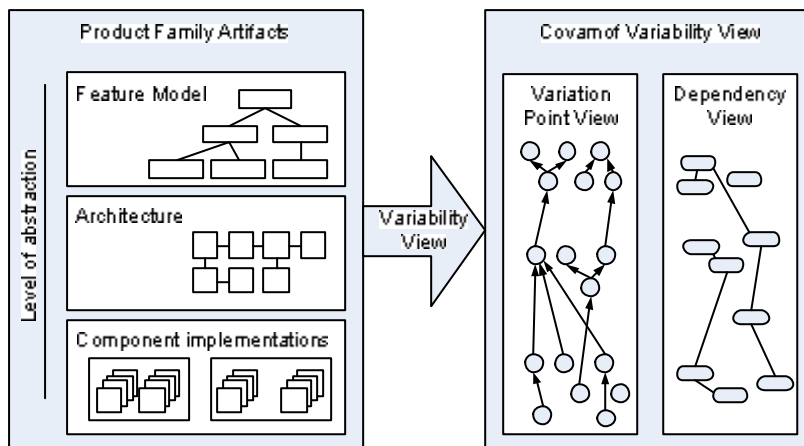
issues

- typically external to artefact
- often binding time left implicit

# Provided Variability

- making provided variability explicit
  - during design, variation points are designed that were implicit earlier
  - during software artefact inspection, previously unknown variation points are identified
  - during evolution, an implicit variation point is made explicit in response to new requirements

# COSVAM: Provided variability



# Variability Mismatch

- *Required variability*: New product family members demand certain variability as necessary or essential (e.g. certain combinations of features, including new features, different quality).
- Provided variability is influenced by:
  - Accuracy of predictions
  - Feasibility of changes
- Not all required variability can be predicted or included at the start of a product family
  - *Variability mismatch*: Mismatch between provided and required variability

# Evolving variability

- Reuse depends on variability
- Variability law (paraphrasing Lehman): *Variability has to undergo continual and timely change, or a product family will risk losing the ability to effectively exploit the similarities of its members.*
- Variability evolution patterns to solve mismatches, such as:
  - Add variant
  - Add new variation point
  - Change of binding time,
  - Change dependency
  - ...
- Challenge: Determine whether, how, and when mismatches should be solved => Variability assessment

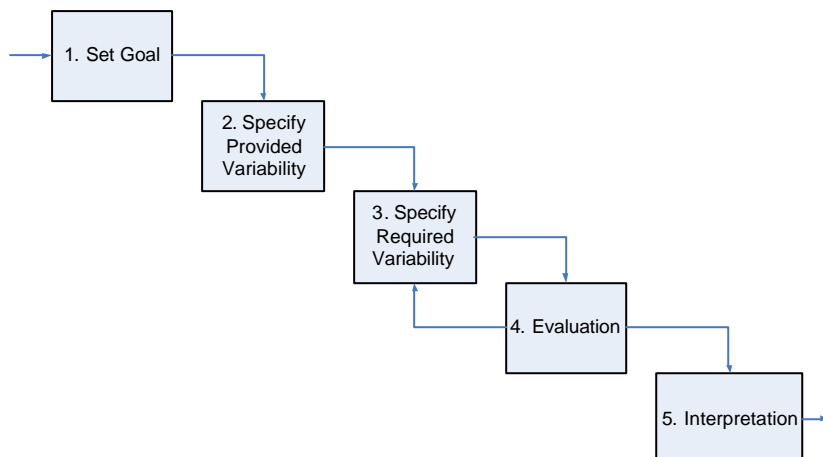
# COSVAM

- Adopts standard assessment steps
- Existing assessment methods do not address
  - Evolution in time and space
  - Product specific vs. product family
  - Uniform treatment of variability at different abstraction layers
- Addresses 4 basic questions
  - Do any mismatches exist?
  - What is the impact of these mismatches?
  - Should the mismatches be solved product specifically or in the product family?
  - Should they be solved now, or in the future?

COSVAM

19

## Standard assessment steps



COSVAM

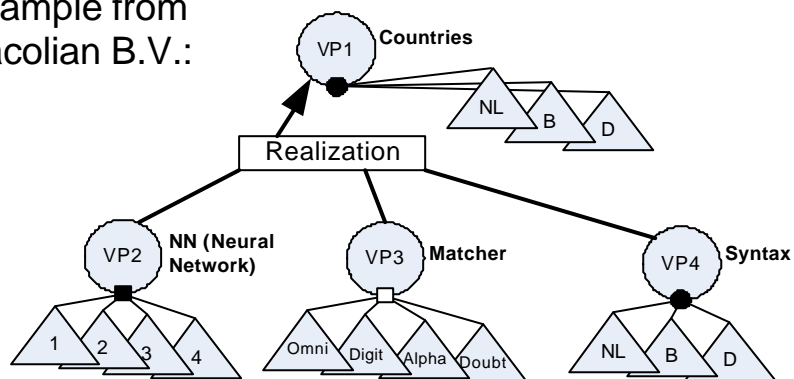
20

# 1. Goal Specification

- Expanding the product profile: *how well are the combinations of features and quality of the new product supported by the variability in the product family?*
- During product derivation: *Should the necessary changes be handled in a product specific manner or in the product family?*
- Collecting input data for release planning: *I have 250 change requests for each of my products, and cannot implement them all. What if I require this particular selection of features?*
- Cross-cutting features: *What is the impact of adding features that influence multiple products in the product family?*
- Evaluate necessity of provided variability: *Are all variants and variation points really necessary?*

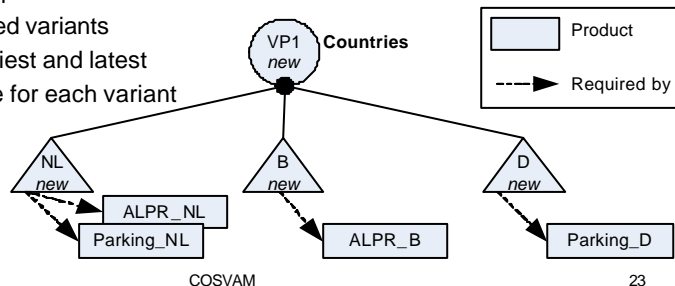
# 2. Specify provided variability

- COVAMOF
- Example from Dacolian B.V.:



## 3. Specify required variability

- Variability required on different levels e.g. *features, quality, components*
- Evolution ? revolution. Considerable parts will match provided variability or require changes to existing variation points
- In terms of delta to provided variability
  - Copy provided variability
  - Add new required variation points
  - Add new required variants
  - Mark required variants
  - Specify earliest and latest binding time for each variant

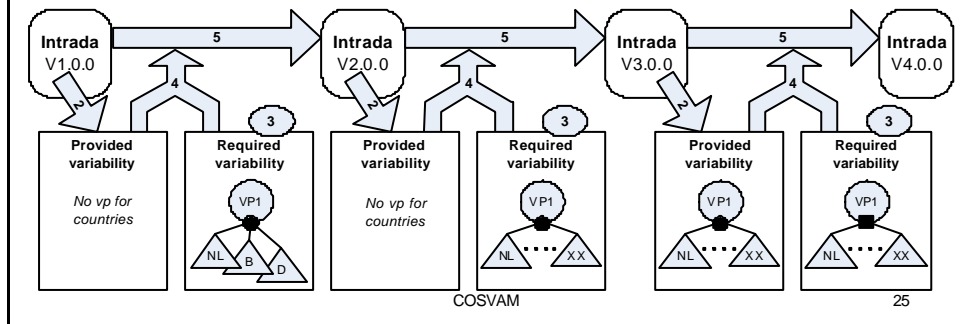


## 4. Evaluation

- Identify mismatches
  - New required variation points and variants
  - Conflicts due to dependencies
  - Conflicts in binding time
  - Whether required quality can be met
  - Which variation points and variants are not used
- Analyze impact
  - Determine what changes are required to solve mismatches
  - Determine conflicts between mismatches
  - Determine impact of various realization scenarios
- Amongst others, based on expert judgment and tracing the realization relations of the variability model
- Can lead to identification of new required variation points and variants

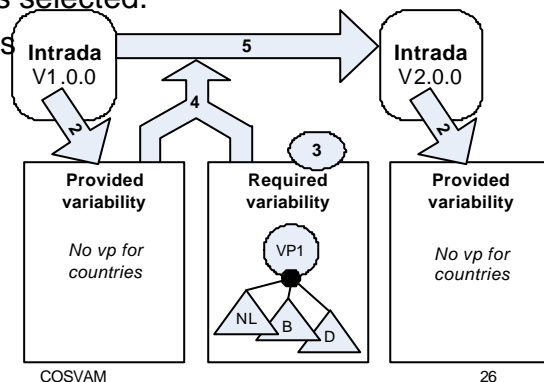
## 5. Interpretation

- Evaluation step provides information on support for required variability and describes the impact of various realization scenarios.
- During interpretation, one of these scenarios is selected.
- Example (Dacolian B.V.):

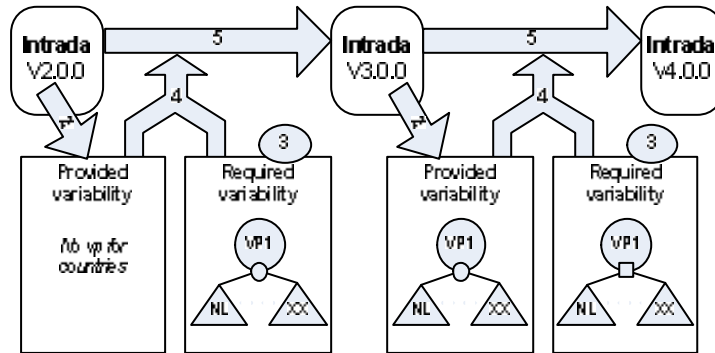


## 5. Interpretation (cntd.)

- Evaluation step provides information on support for required variability and describes the impact of various realization scenarios.
- During interpretation, one of these scenarios is selected.
- Based on factors such as
  - Likelihood
  - Cost
  - Risks
- Example: Dacolian B.V.



## Example: Dacolian B.V. (cntd.)



COSVAM

27

## Inverse Assessment

for each variation point

- verify whether it is associated with a required variability
- if not:
  - determine likelihood of future usage
  - determine need for legacy reasons
  - determine impact on conceptual complexity
- decide whether variation point should be removed

COSVAM

28

# Designing Variation Points

variability realization techniques

- architecture reorganization
- variant architecture component
- optional architecture component
- binary replacement - linker directives
- binary replacement - physical
- infrastructure-centered architecture
- variant component specializations
- optional component specializations
- runtime variant component specializations
- variant component implementations
- condition on constant
- condition on variable
- code fragment superimposition

# Designing Variation Points

process

- select variability requirement
- select implementation level, i.e. architecture, component, etc.
- identify binding and variant addition times
- identify number of expected variants
- select variability realization technique
- impose variability realization technique on software artefact

# Evolution of Variability

- new variation point
- change binding time
- change variant addition time
- remove variation point
- change variation point dependencies
- (add variants)

# Conclusion

- notation
  - required variability
  - provided variability
- assessment
  - time vs. space dimension
- design
  - variability realization techniques
- evolution