# Software product lines, feature modelling, analysis and configuration

David Benavides
benavides@us.es
Evolución y Gestión de la Configuración

The main goal of this lesson is to give an overview of “software product lines” from a practical and research point of view

SPL

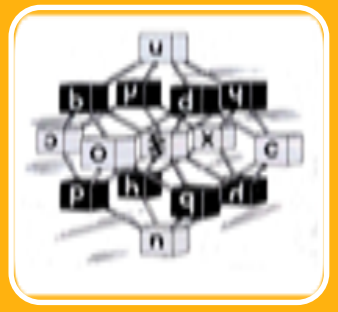
FM

AAFM

FOP

# Part I

# Software Product Lines



# Variability Modelling

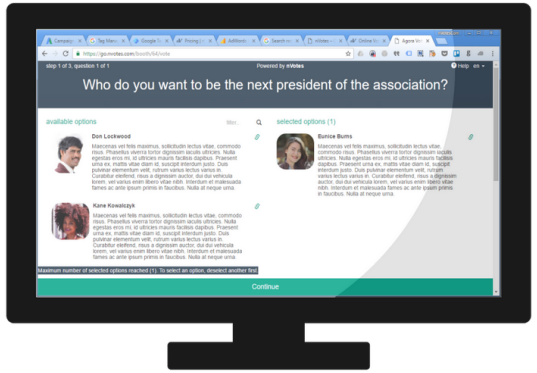
# Some real cases

# Real cases

nVotes

Features & Plans  Contact us

## Secure & easy online voting




Secure, robust and affordable internet election management solution that makes it easy for voters to cast their vote online

UNIVERSIDAD Ð SEVILLA

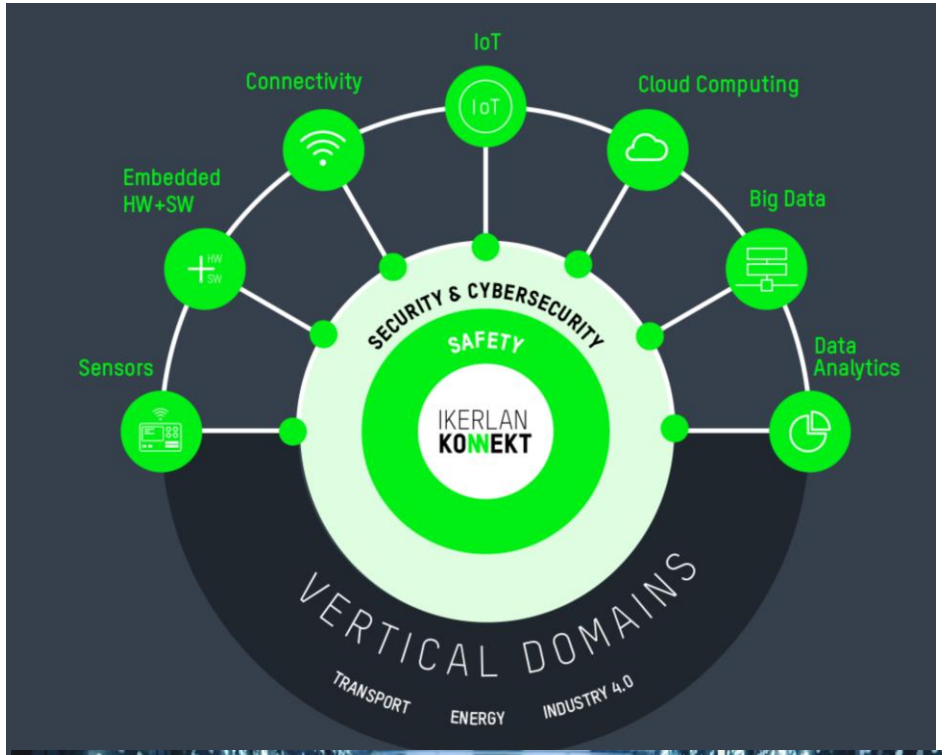
# Real cases


DIPUTACION DE SEVILLA
INPRO
DIPUTACION
buscar...
Zona Empleados
RSS | Directorio | Mapa Web | Contacta
Sociedad de Informática Provincial
INPRO
Inicio
La Empresa
Noticias
Novedades
Nuestros Usuarios
Productos y Servicios
I+D

# Real cases


Equipo
Carpeta personal
Instalar Guadalinex v9
Todas las aplicaciones
Accesorios
Educación
Juegos
Gráficos
Internet
Oficina
Sonido y vídeo
Acceso universal
Wine
Preferencias
Administración
Lugares
Archivos recientes
Actualización de software
Analizador de uso de disco
Aplicaciones preferidas
Applets
Archivos
Ayuda
Bloc de Notas
Bloqueador de pantalla
Bluetooth
Calculadora
Capturar pantalla
Cliente de correo Thunderbird
Color
Conexiones de red
Menú
22 de nov, 00:45

# Real cases


IoT
Connectivity
Cloud Computing
Embedded
HW+SW
Big Data
Sensors
Data
Analytics
SECURITY & CYBERSECURITY
SAFETY
IKERLAN
KONNEKT
VERTICAL DOMAINS
TRANSPORT
ENERGY
INDUSTRY 4.0


IKERLAN KONNEKT
CONOCE LAS SOLUCIONES DE DIGITALIZACIÓN QUE SE ADAPTAN A TU EMPRESA

# Real cases


**What configurations should I test to defend my users from bugs?**

# Software product lines

# Industrial Trends

| | |
|---|---|
| Organizations are evolving | • *Project* Centric Software Engineering<br>• *Product* Centric Software Engineering |
| Software ***variability*** constantly increasing: | • Variability goes from hardware to software<br>• Variations points grows by thousands |
| Assets' ***Reuse*** is shifting | • from ad-hoc to ***systematic*** |

# What is a software product line?

# Real example

# Real example

# Mass production

producing efficiently a large amount of standardized products

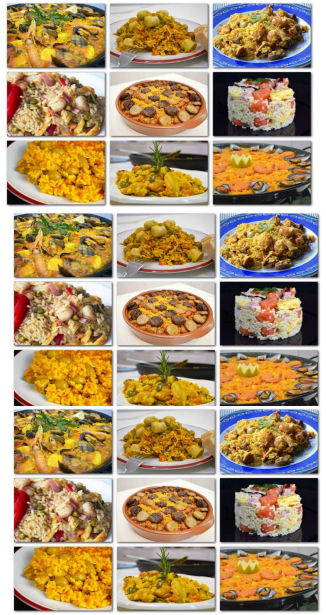
# Software product lines

No customization
-
one product

## oduction

acing efficiently a large amount of
standardized products

# Software product lines

# Mass customization

“a paradigm shift for the enterprise to offer products and services best catering to individual customer's needs whereas keeping near-mass production efficiency “

[Tseng, M.M., Jiao, J. (2001)]

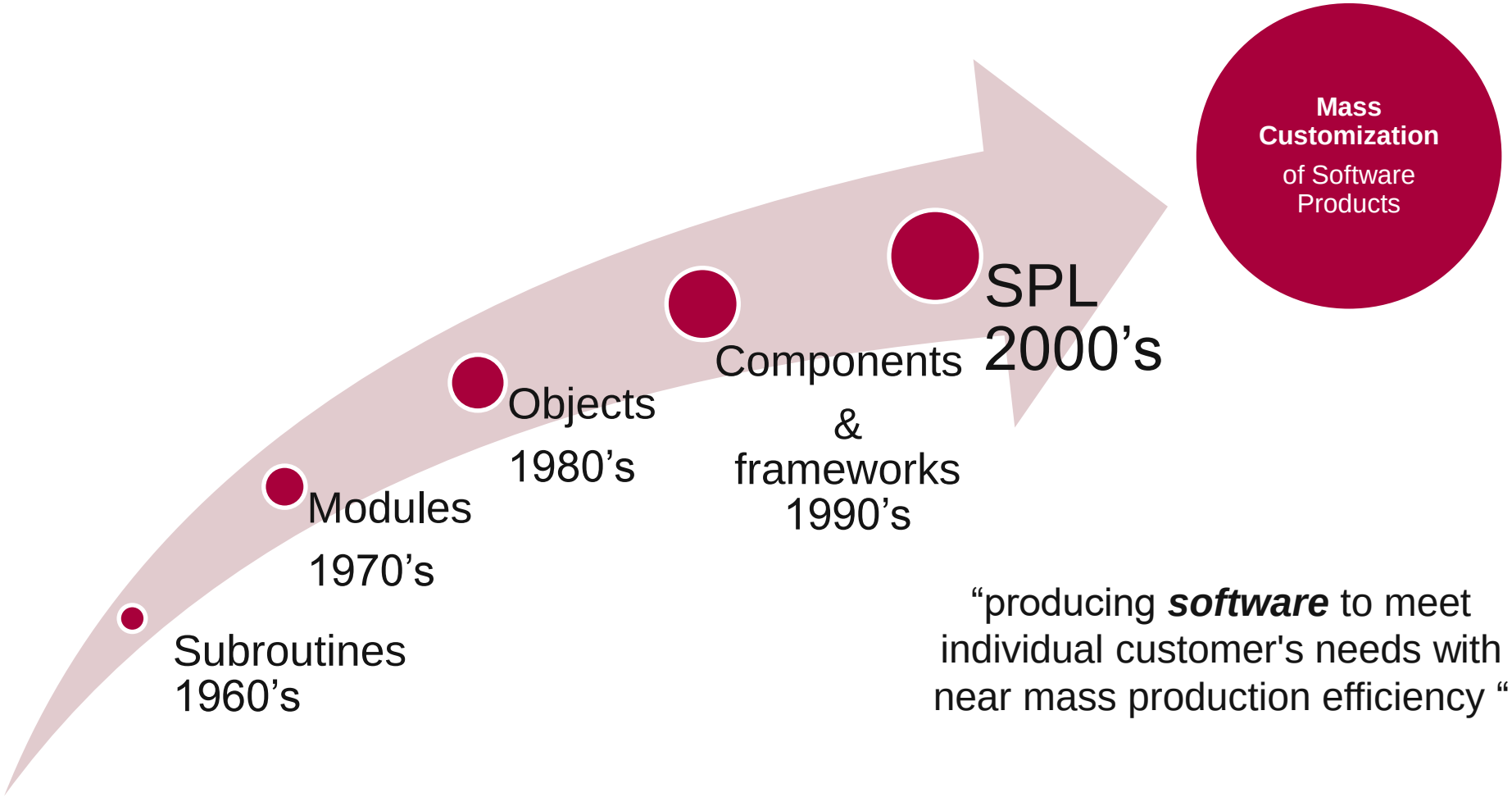

Customization
-
A set of products

# Mass customization

“a paradigm shift for the enterprise to offer products and services best catering to individual customer's needs whereas keeping near-mass production efficiency “

[Tseng, M.M., Jiao, J. (2001)]

# Software product lines




"producing ***software*** to meet individual customer's needs with near mass production efficiency "

# Software product lines

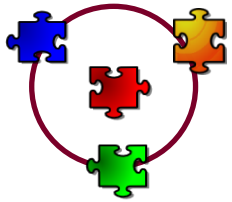
Common features

- Alarm clock
- Calls
- Messaging

Variable features

- Media
- Games
- Connectivity

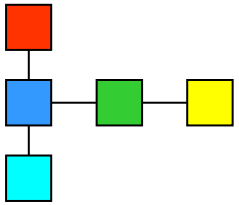
## Variability Model



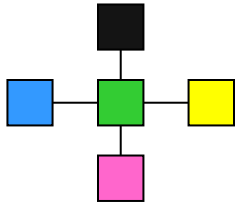
- Documents the variability of SPL
- Enable managing the variability

# Software product lines

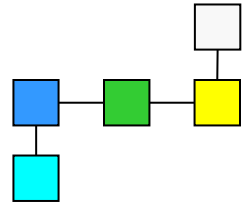
Traditional Approach (*mass production*)



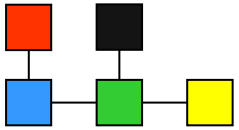
Product 1



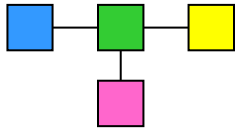
Product 2



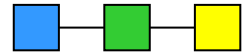
Product 3



Product 4



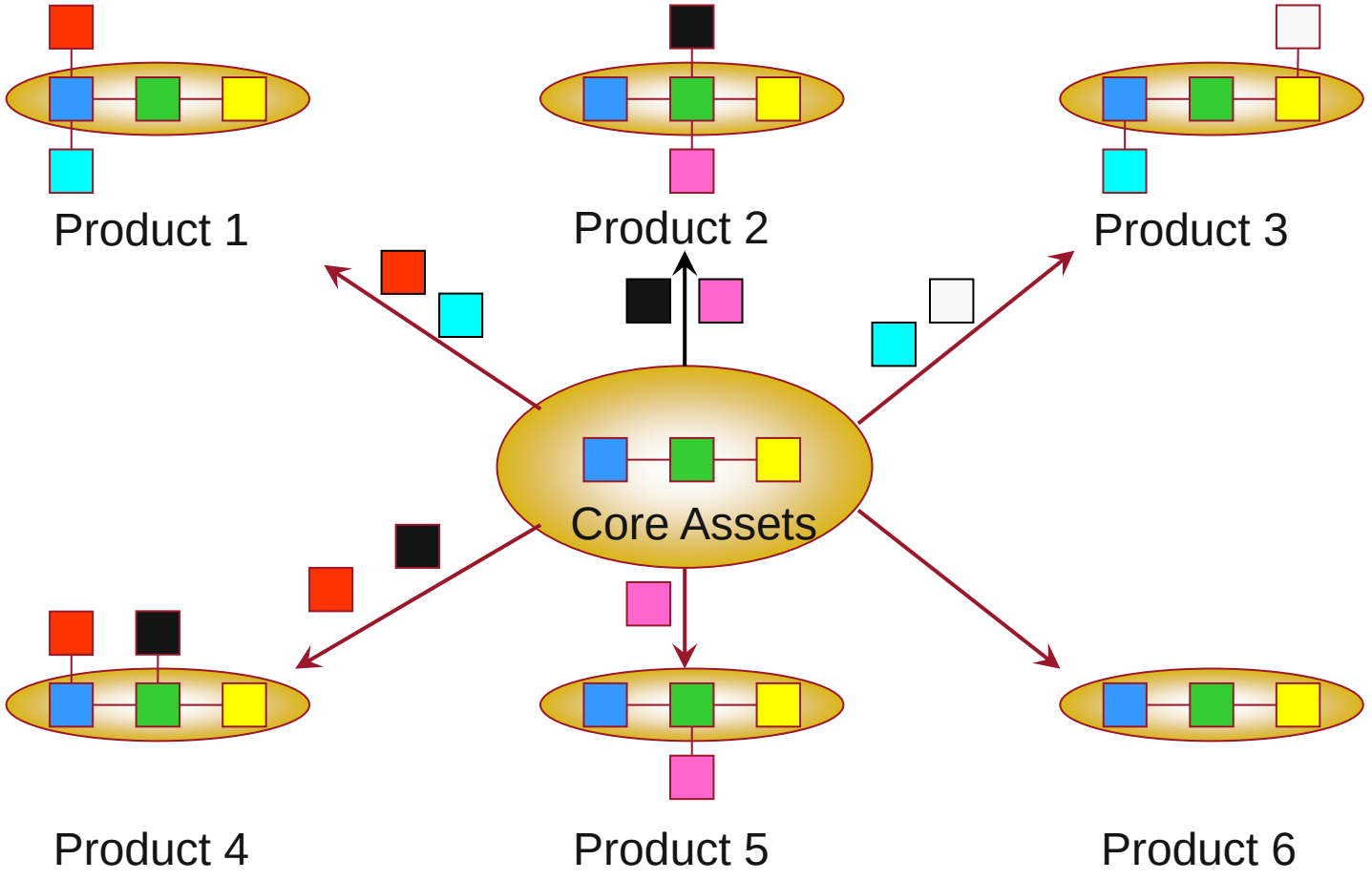
Product 5



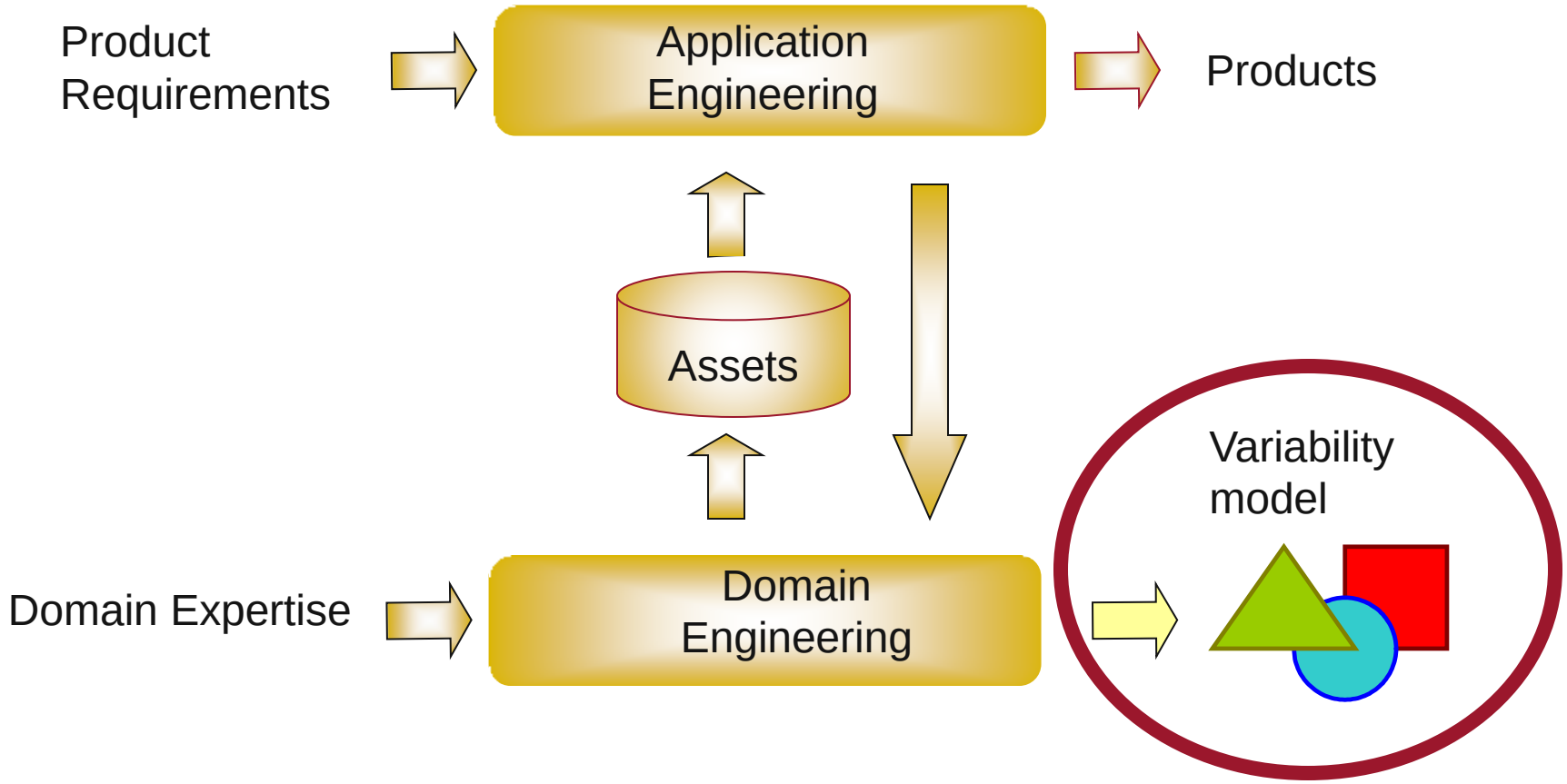
Product 6

# Software product lines

Product Lines Approach (*mass customization*)



# SPL: Activities



# SPL framework

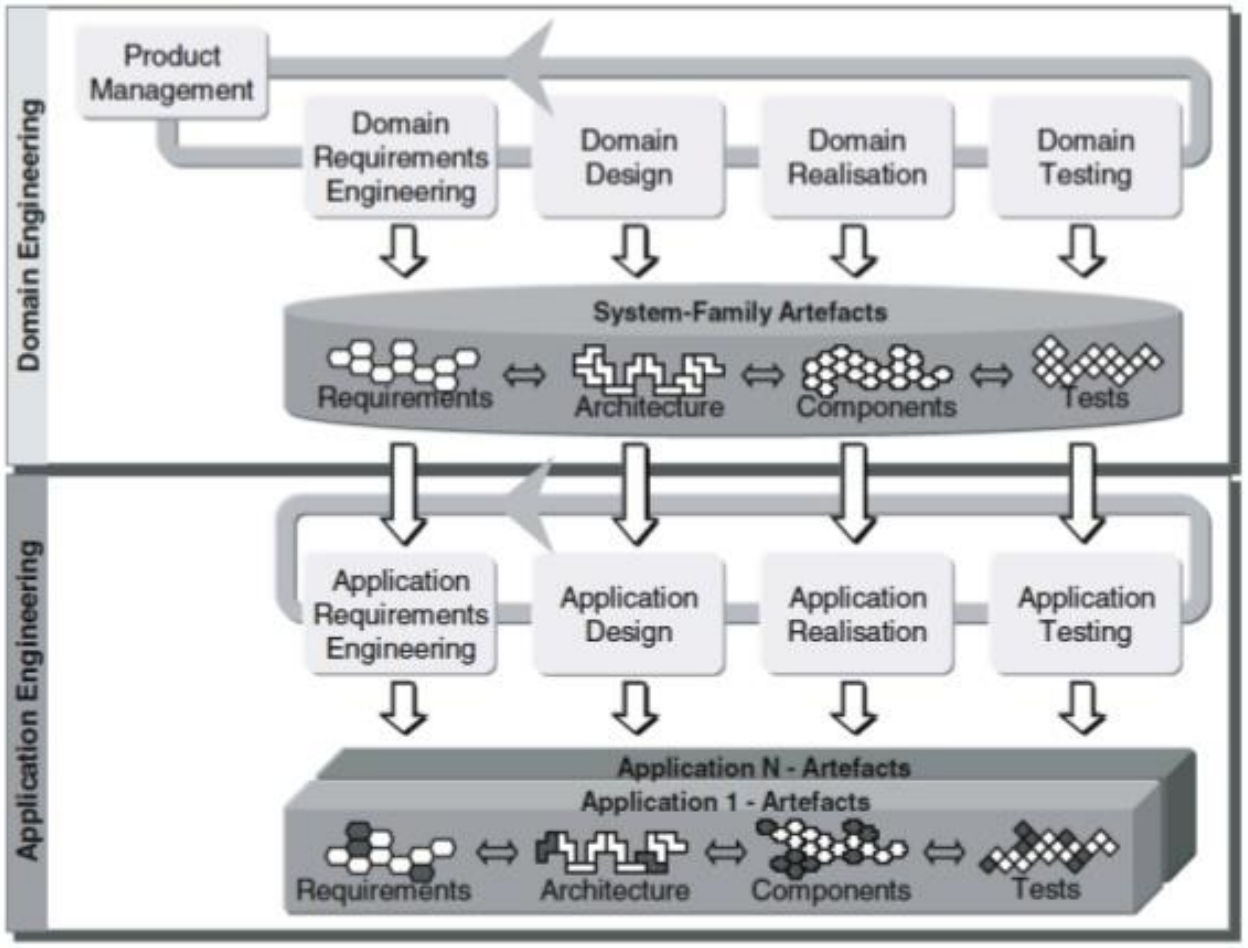



**Fig. 1.2.** The two-life-cycle model of software product line engineering

From "Software Product Line Engineering" by Phol et al.

# A more practical view of the SPL framework

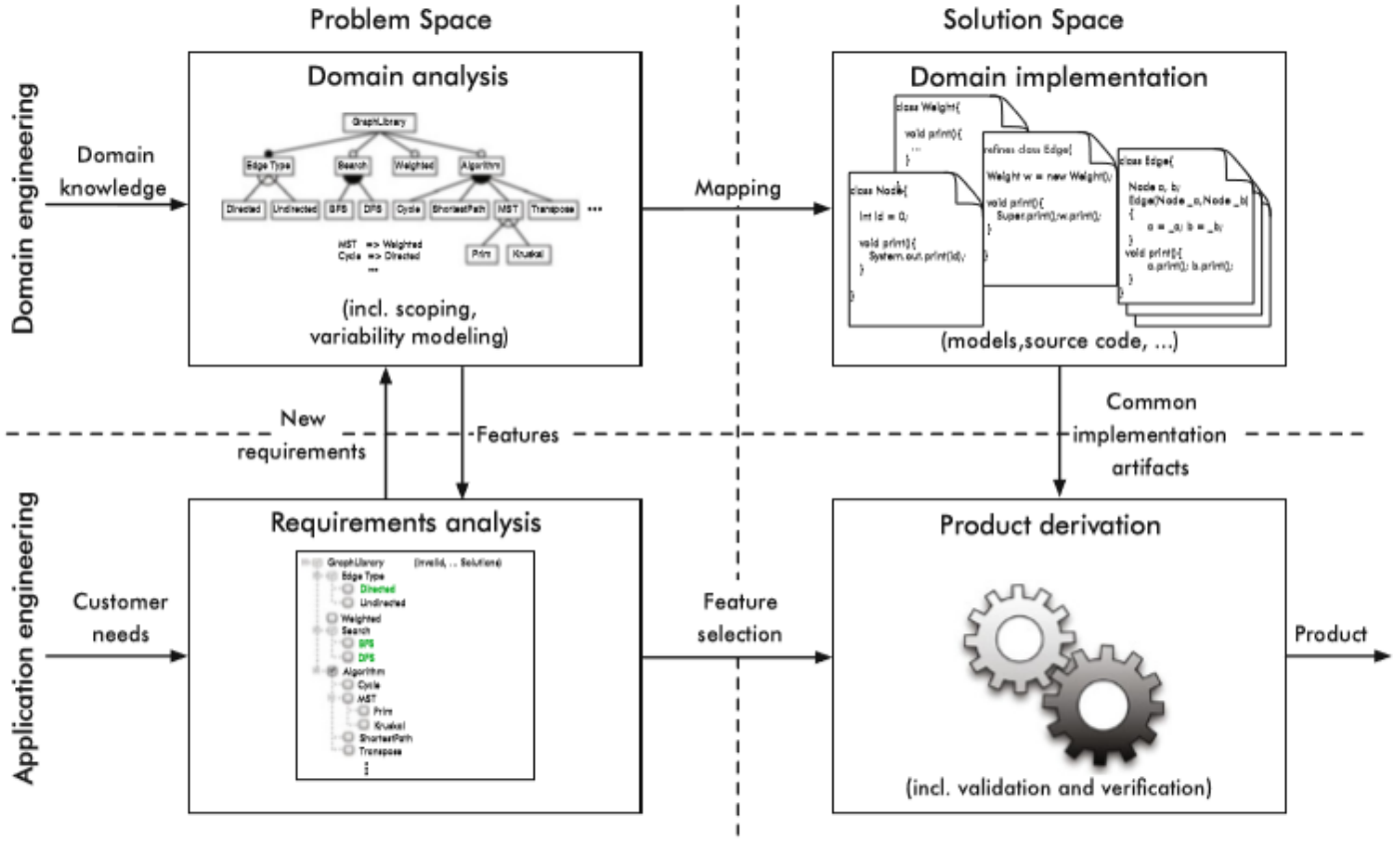



**Fig. 1.1** An overview on software product-line engineering

From “Mastering Software Variability with FeatureIDE”

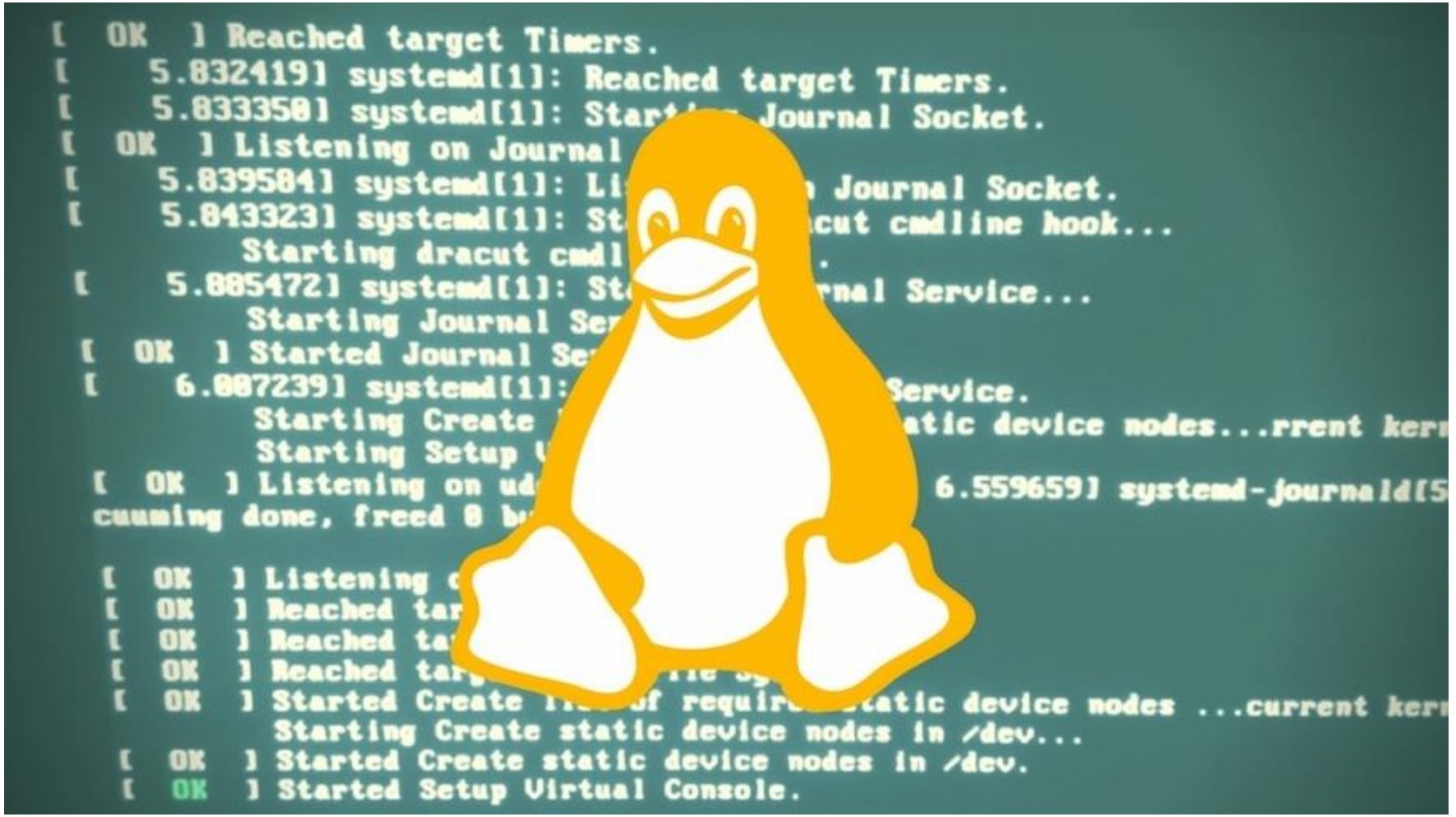
# What are the reasons for SPLE “tentations”?

# Product explosion

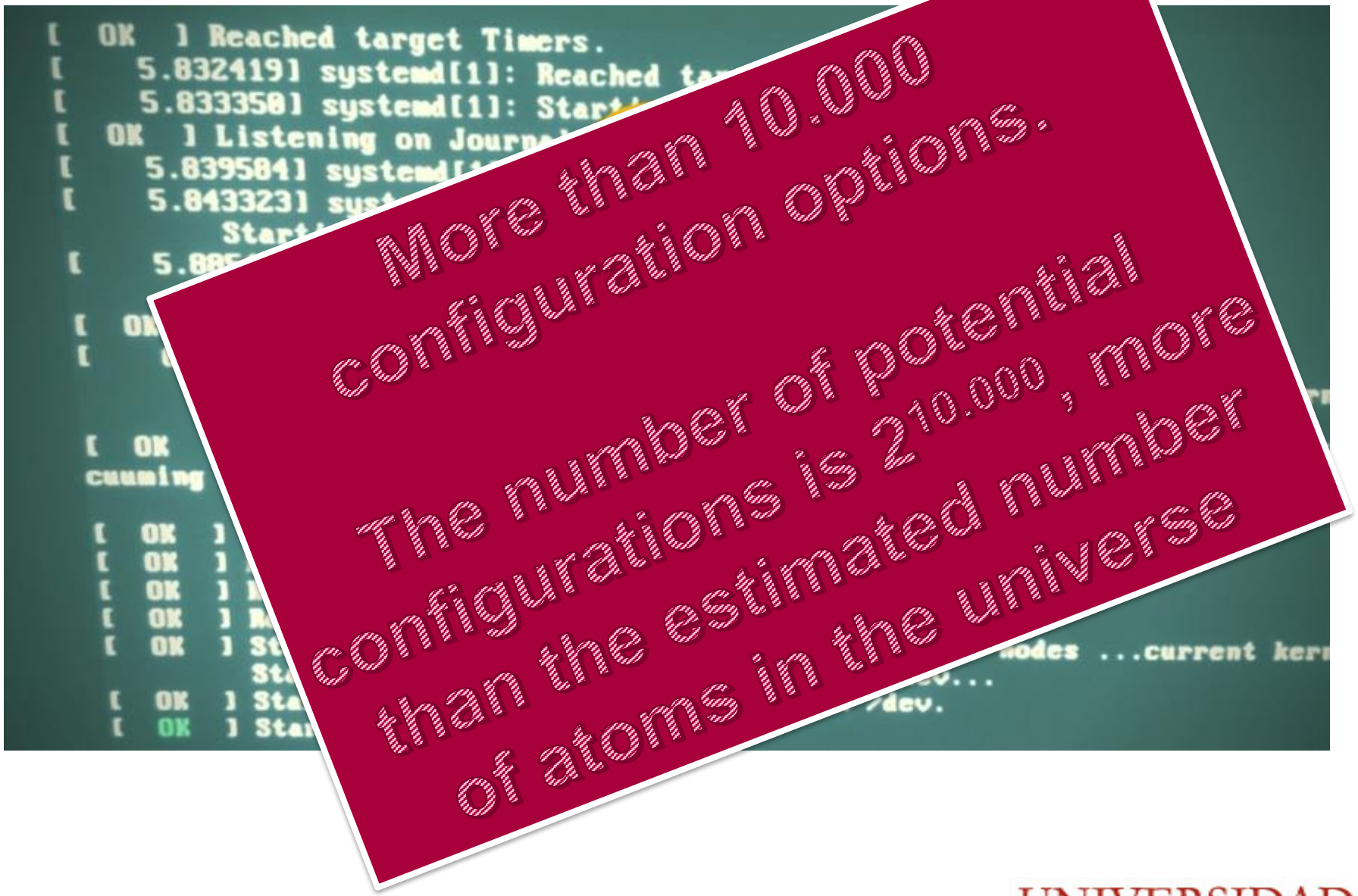
# Customers explosion

# Technology explosion

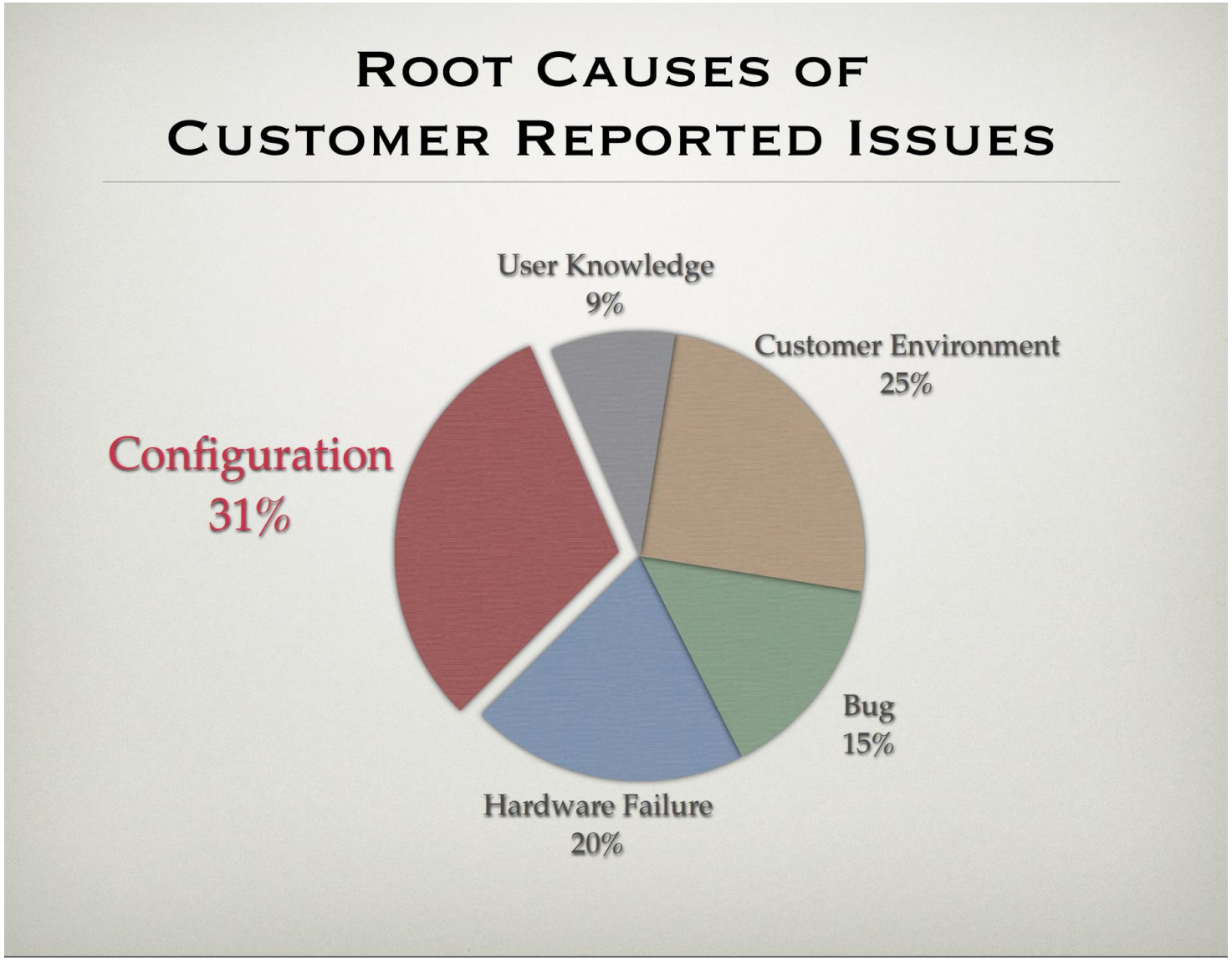
# Configuration explosion



# Configuration explosion



# Configuration explosion




Taken from http://sigops.org/sosp/sosp11/current/2011-Cascais/12-yin-slides.pdf

# Explosions consequences

- Product oriented development
- Fire-fighting mode
- Opportunistic reuse

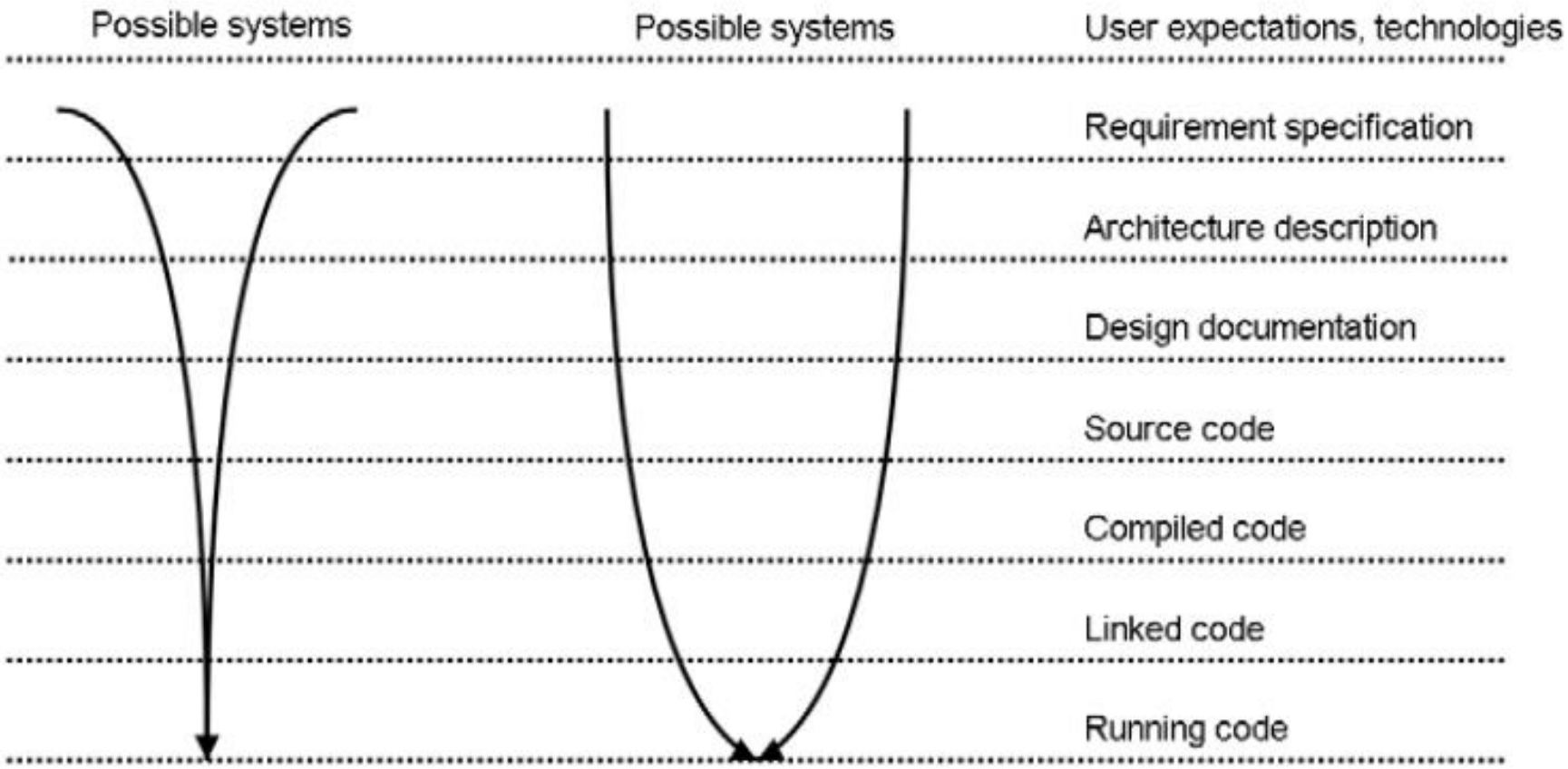
- Lack of innovation
- Quality degradation
- Knowledge lost

## Some “tentations”

- Product portfolio diversity
- Common user experience for product in the portfolio
- Customization of products

# What are the goals?

# SPL metaphors




Svahnberg M., van Gurp J., Bosch J., *On the Notion of Variability in Software Product Lines*. Proceedings of IEEE/IFIP Conference on Software Architectures, 2001.

# SPL metaphors

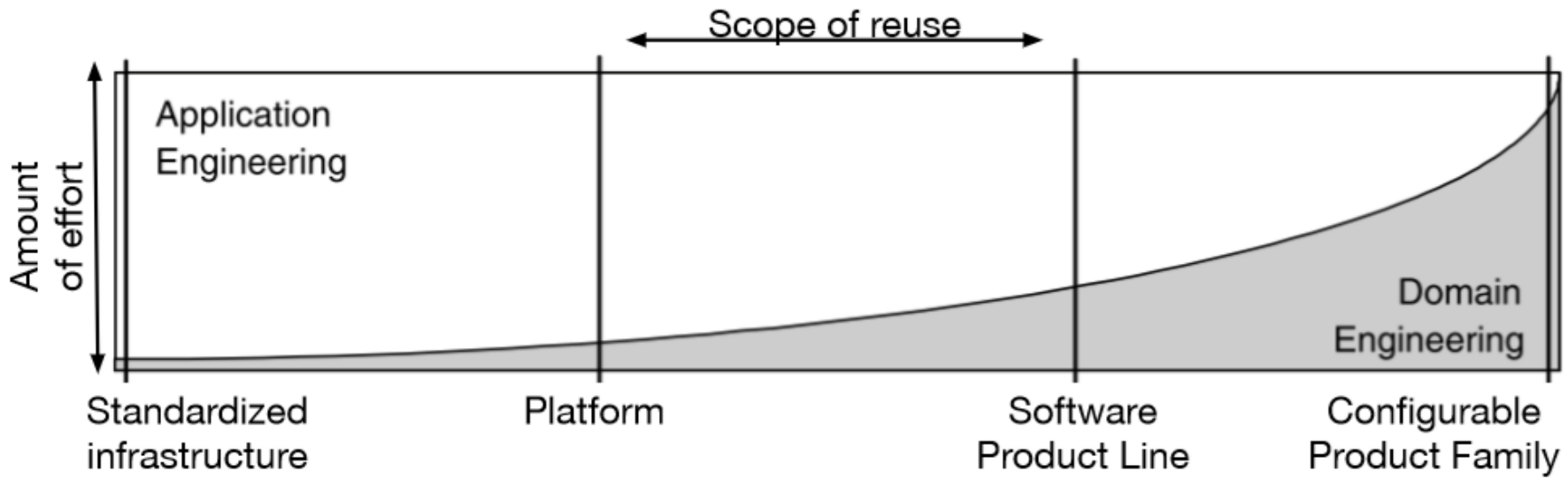



Figure 1.1: SPL maturity stages: from less mature (left) to more mature (right)[DSB05]).

# Evolution of an SPL



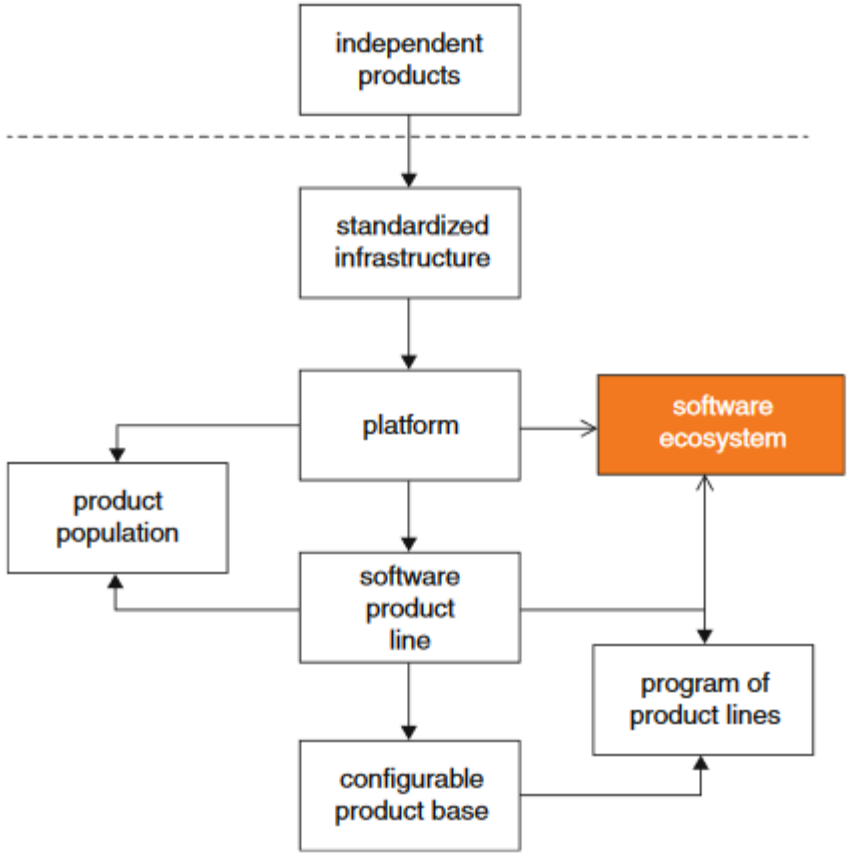
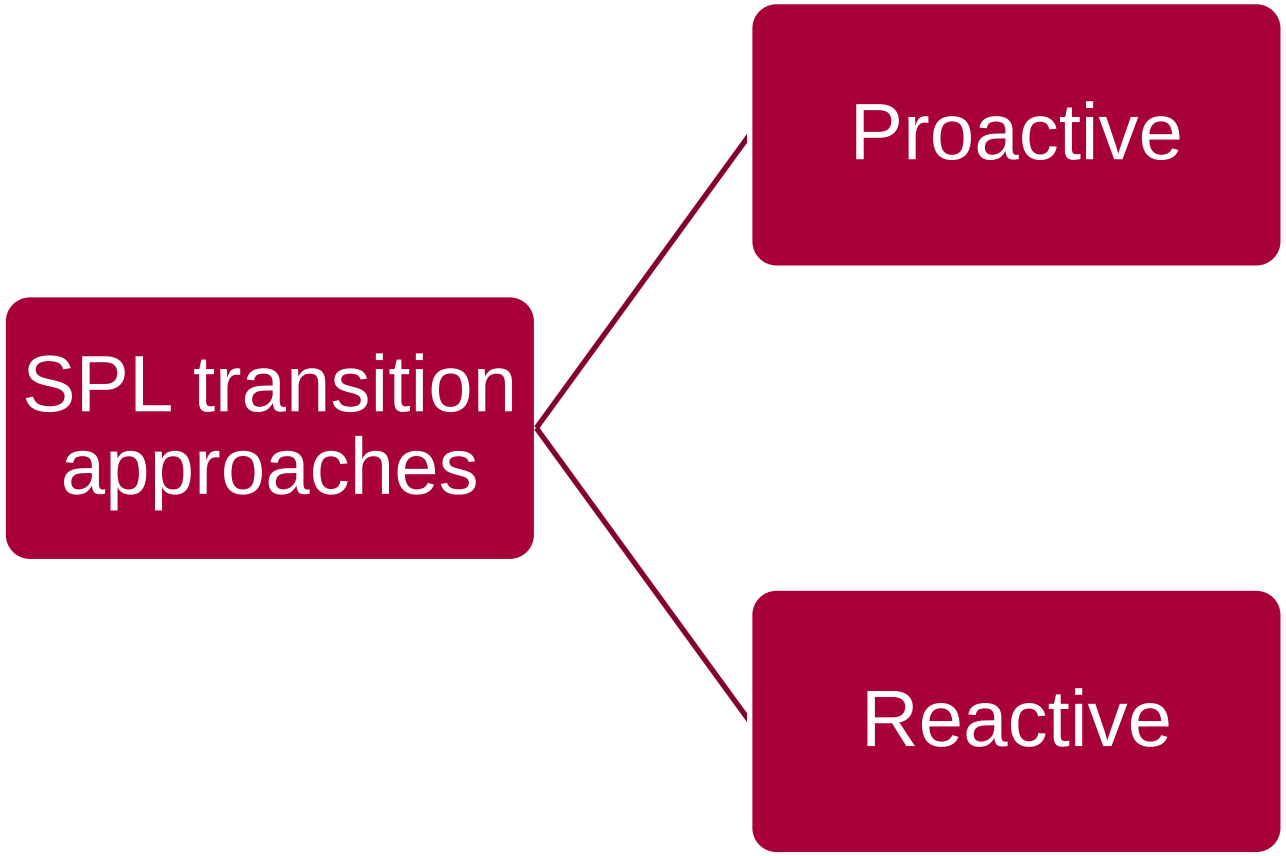



**Fig. 1.1** Evolution of a software product line

Taken from “Systems and Software Variability Management”

# OK, I trust you but... how shall I transition?

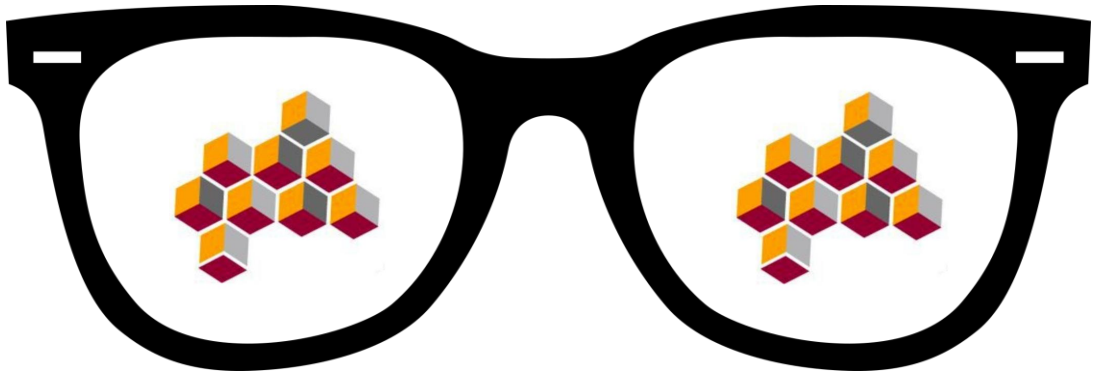

SPL transition approaches
Proactive
Reactive

# Some barriers

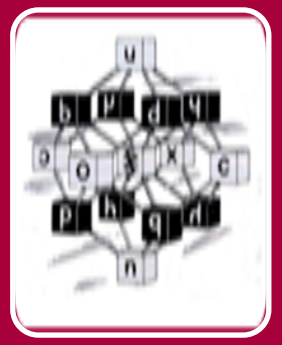
# Business strategy

# Variability, a new degree of complexity


DISCUSSION
QUESTIONS



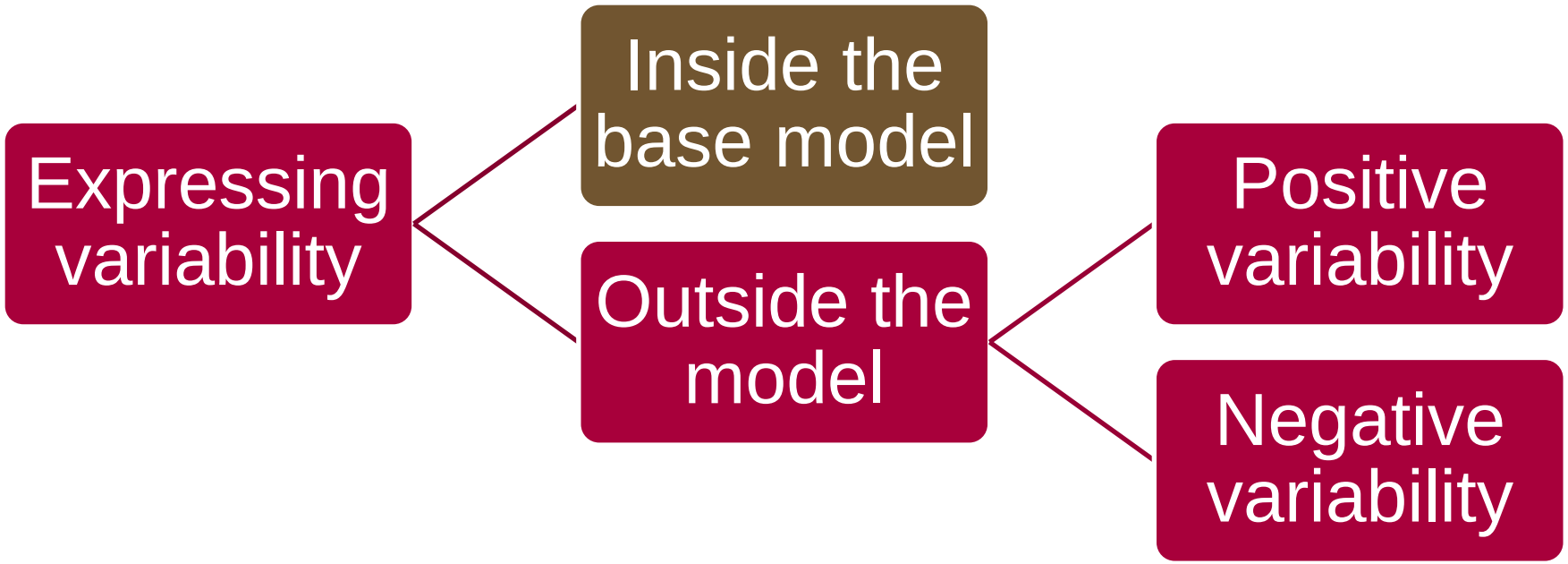
# Software Product Lines



# Variability modelling

# How to model variability?

# How to model variability



# Inside the model

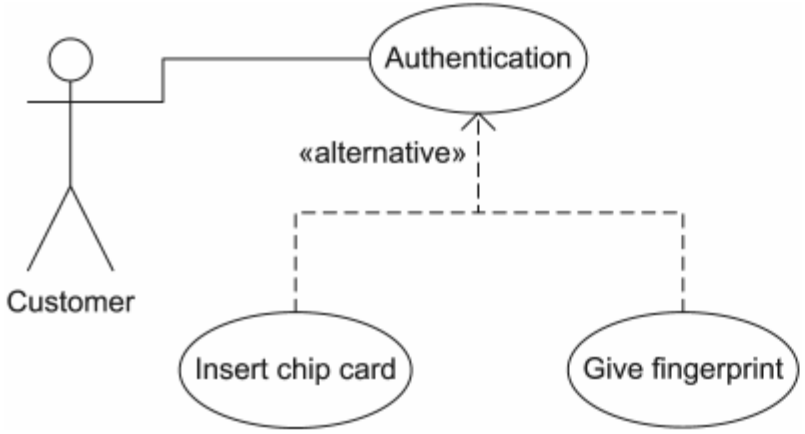



**Figure 5: Example of an alternative relationship**

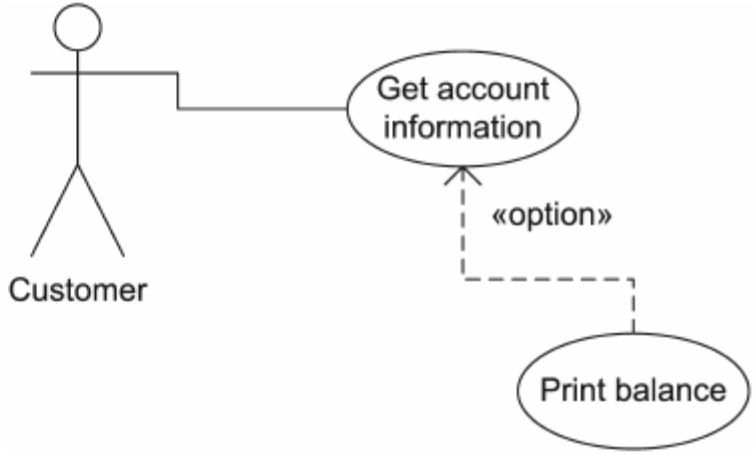
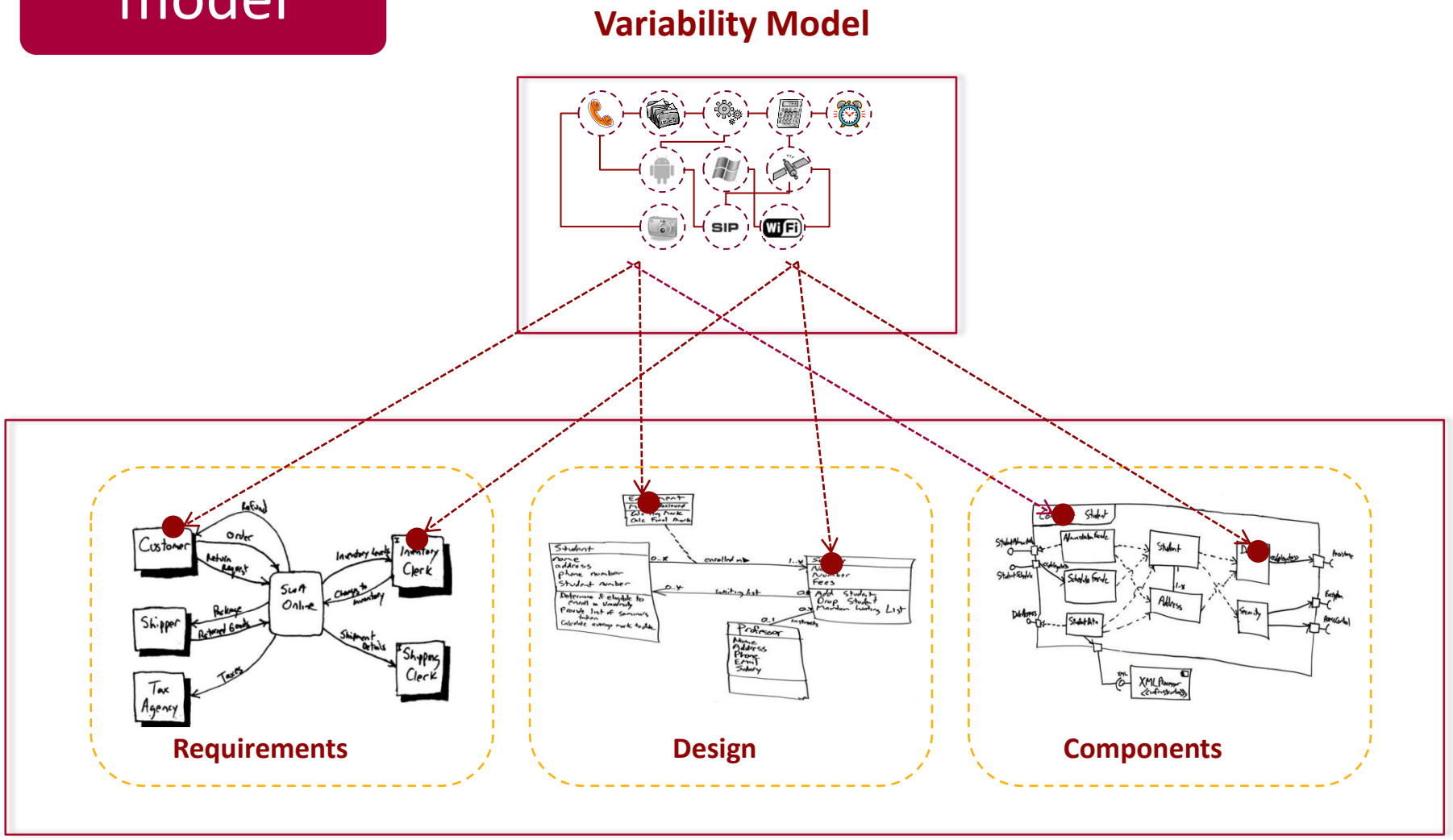


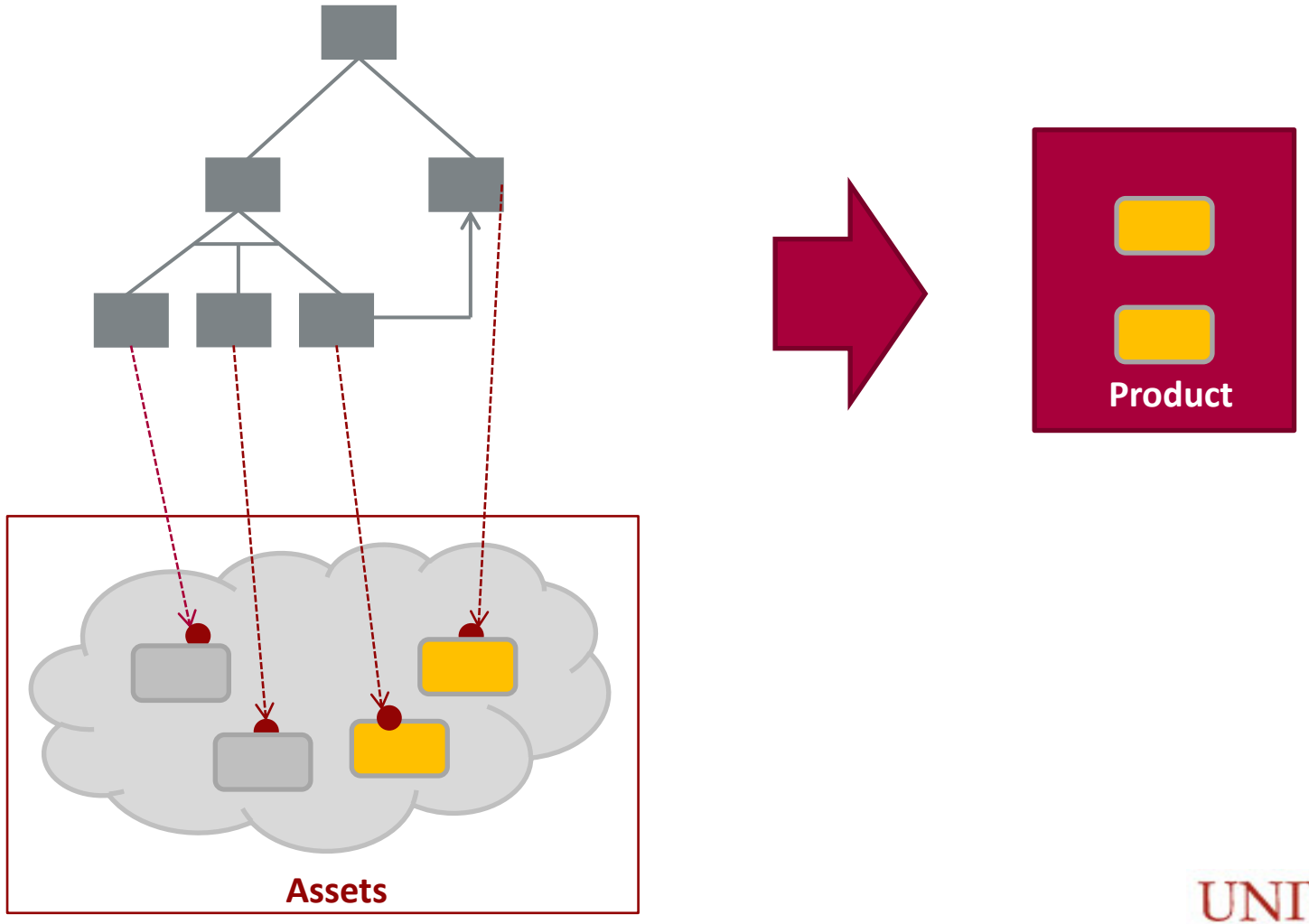

**Figure 6: Example of an optional relationship**

Taken from Modeling variability by UML use case diagrams by Von der Massen et al.

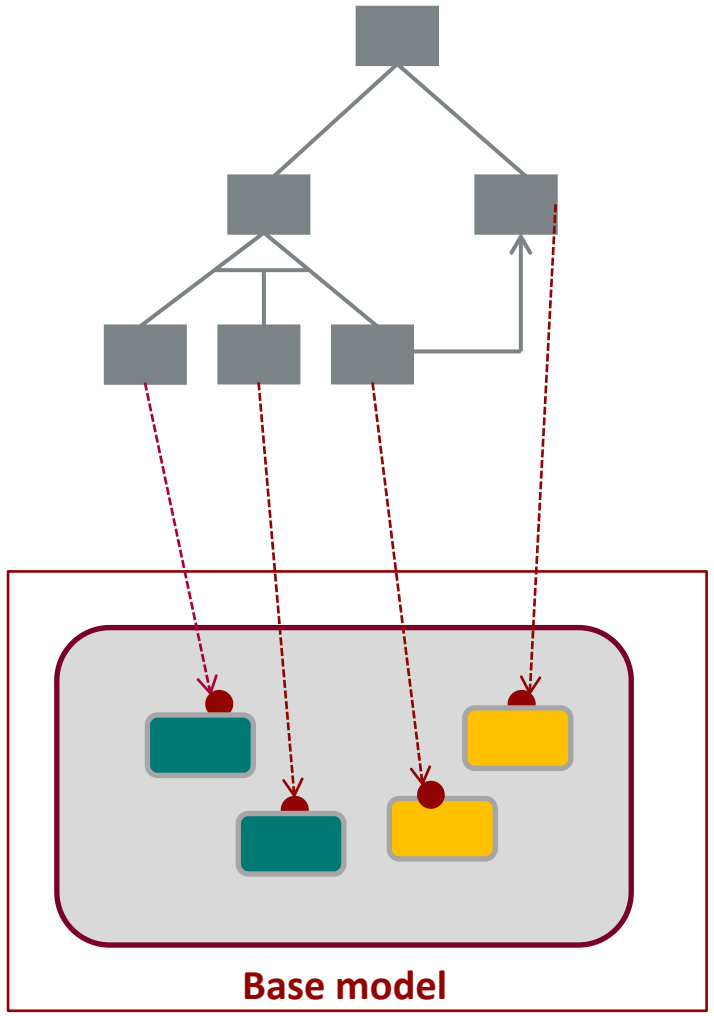
# Outside the model

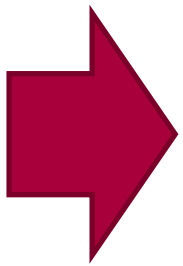


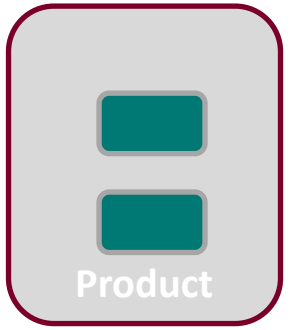
# Positive variability



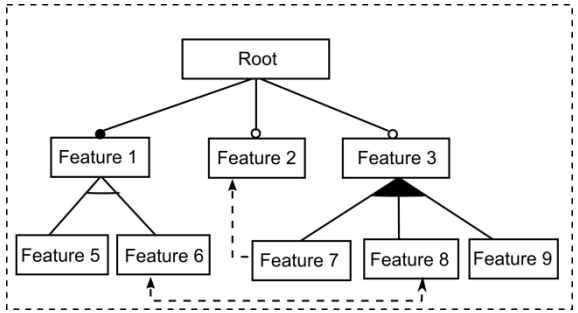
# Negative variability



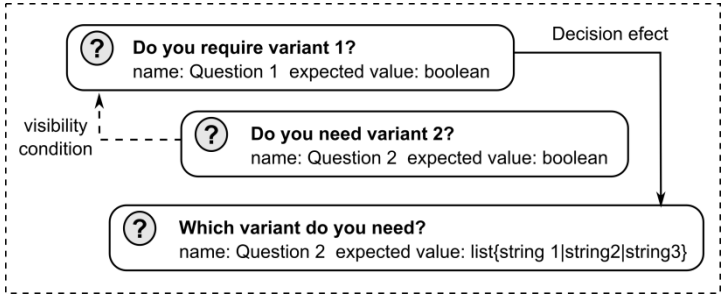





# How to model variability




**Feature modelling**

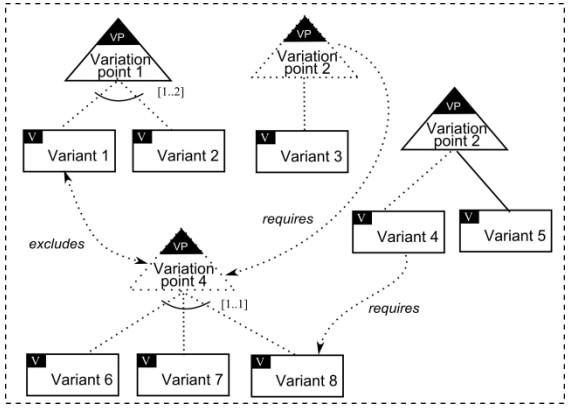



**Decision modelling**

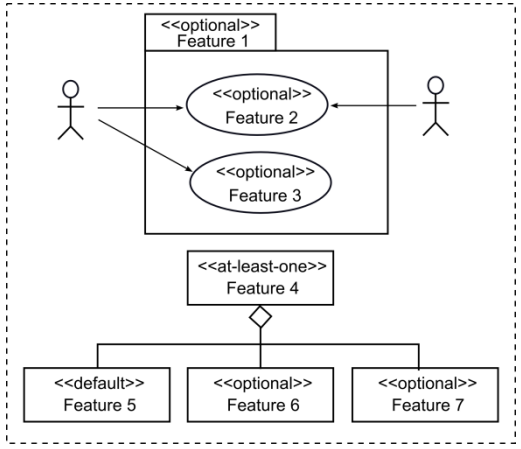
**Techniques**

**Ad-hoc solutions: tables, textual docs, ...**

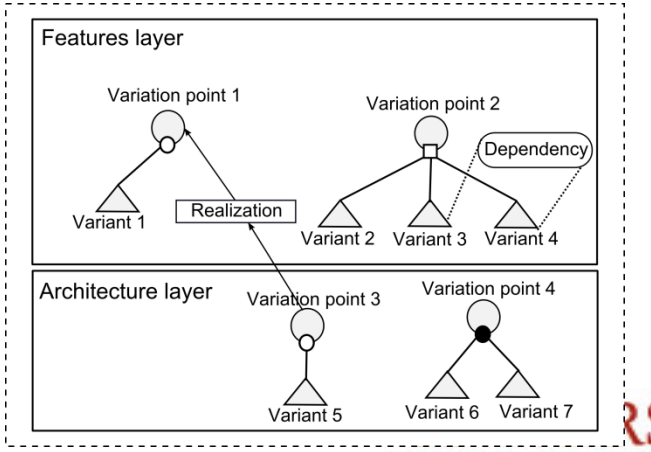
**Orthogonal variability modelling**



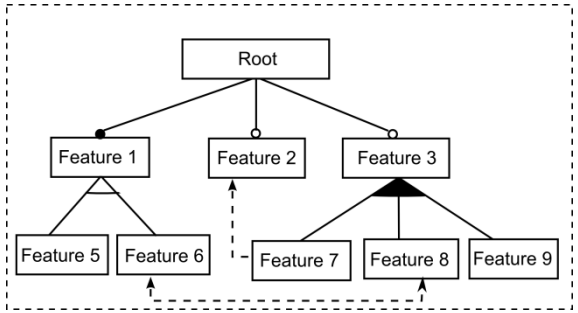

**UML-based**



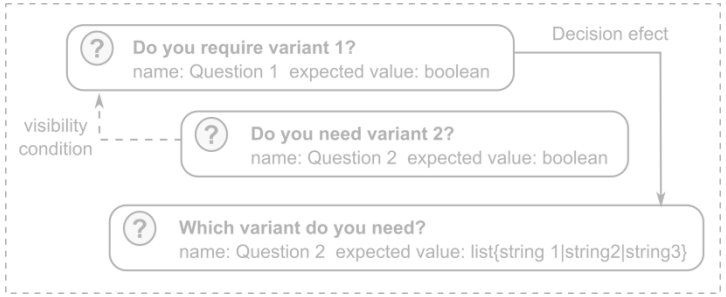

**COVAMOF**



# How to model variability



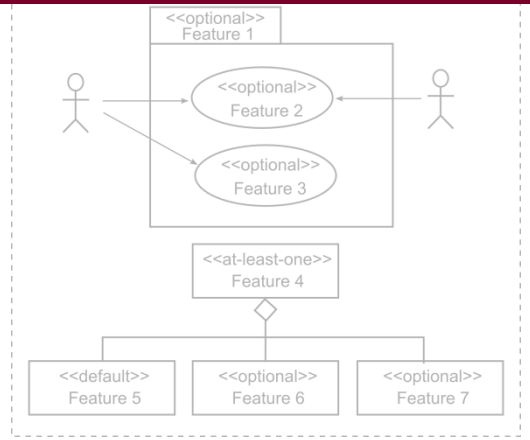

**Feature modelling**

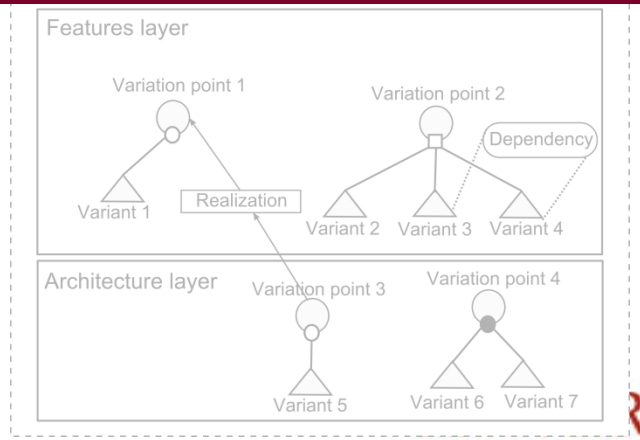



Decision modelling

Ad-hoc
tables,
...

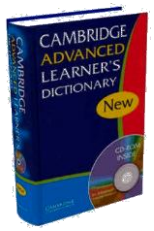
Feature models were first introduced by Kang et al. in 1990



# How to specify a particular product?

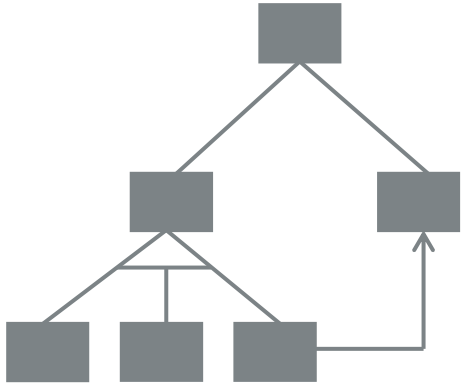
FEATURE




*"An important part of something"*

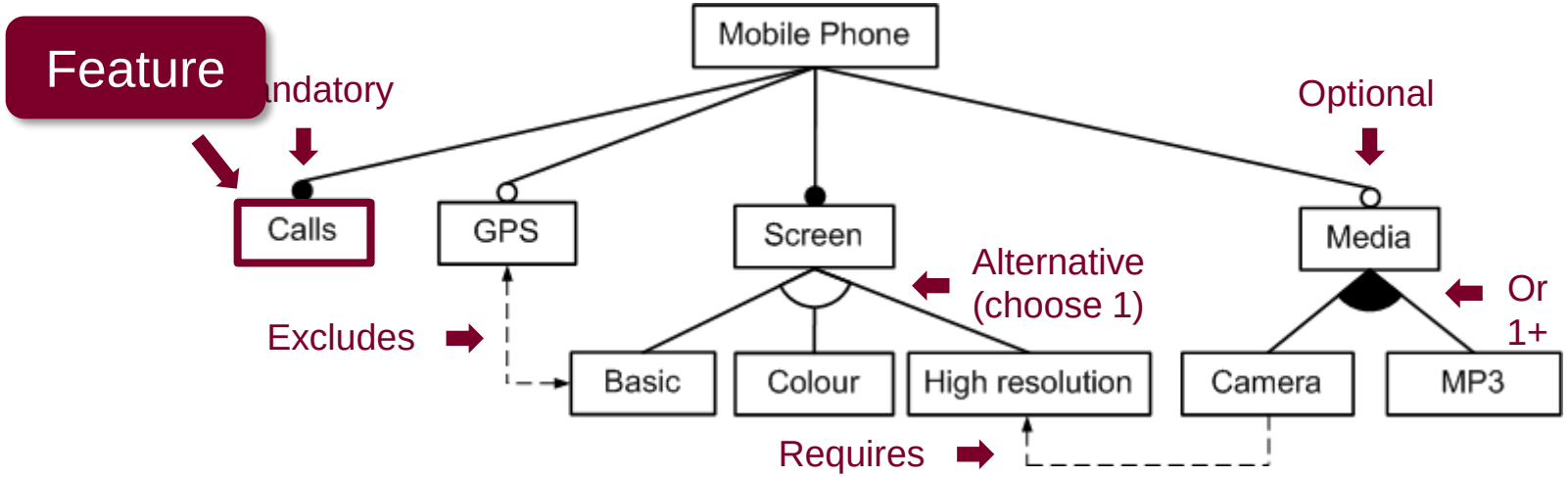

*"A prominent or distinctive characteristic of a software system"*

# How to specify an SPL?



*“Feature Model: A hierarchically arranged set of features to represent all possible products of an SPL”*

# Feature models

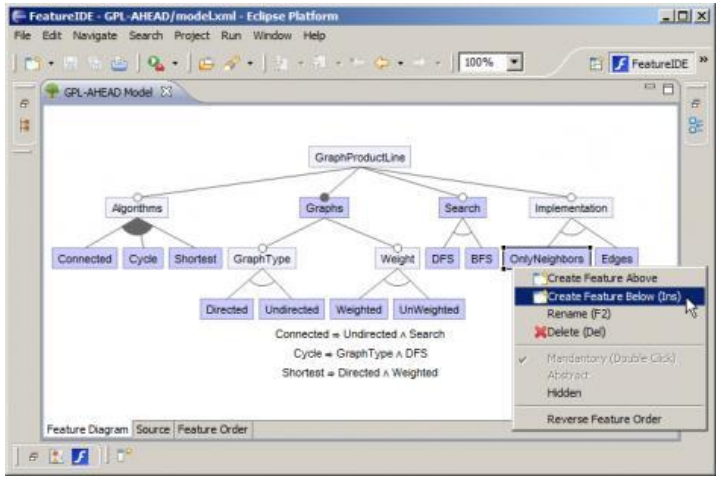


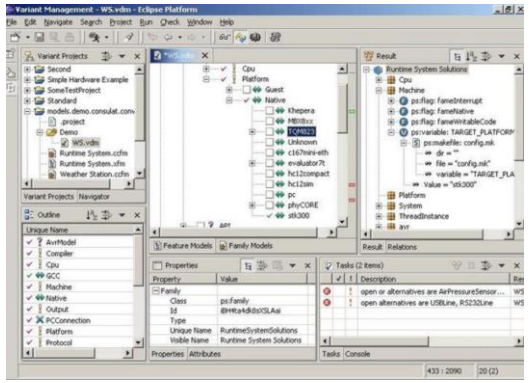
Design a feature model for your own SPL!

pure·systems

IDE Feature

BigLever Software, Inc.



UNIVERSIDAD D SEVILLA

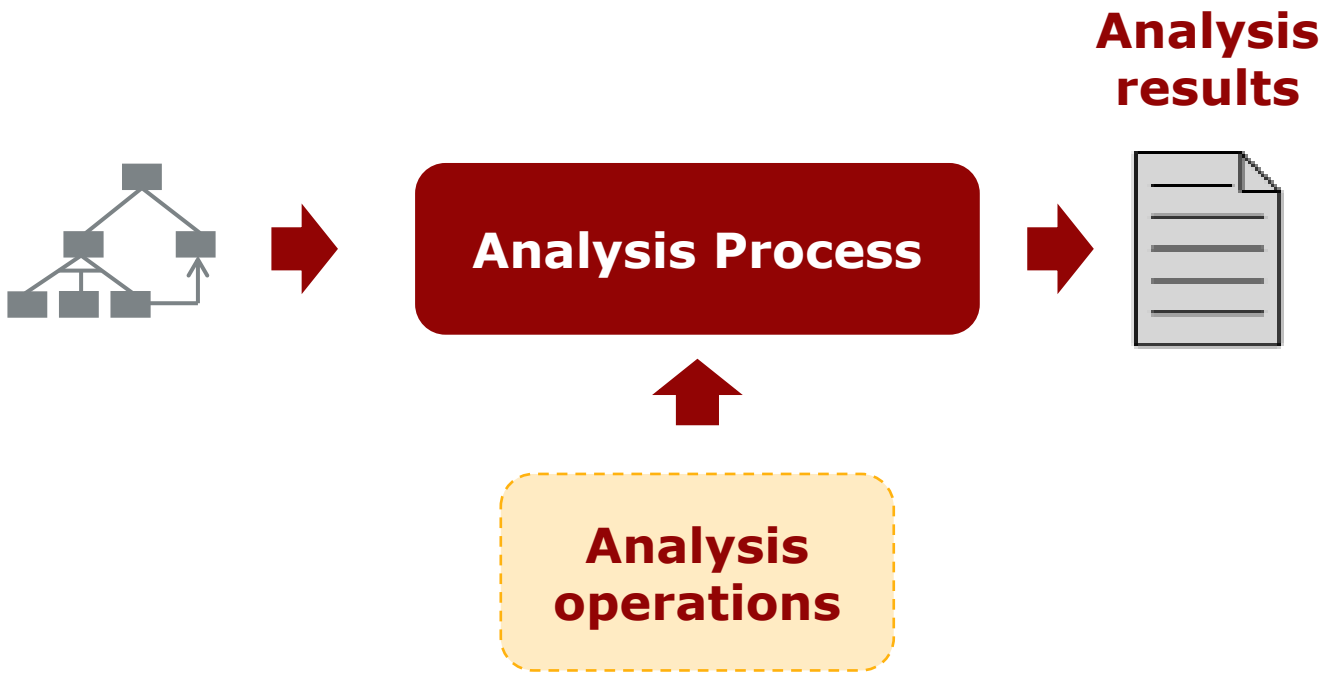
# Conclusions

- SPL is a *new* software production paradigm
- Variability management is essential

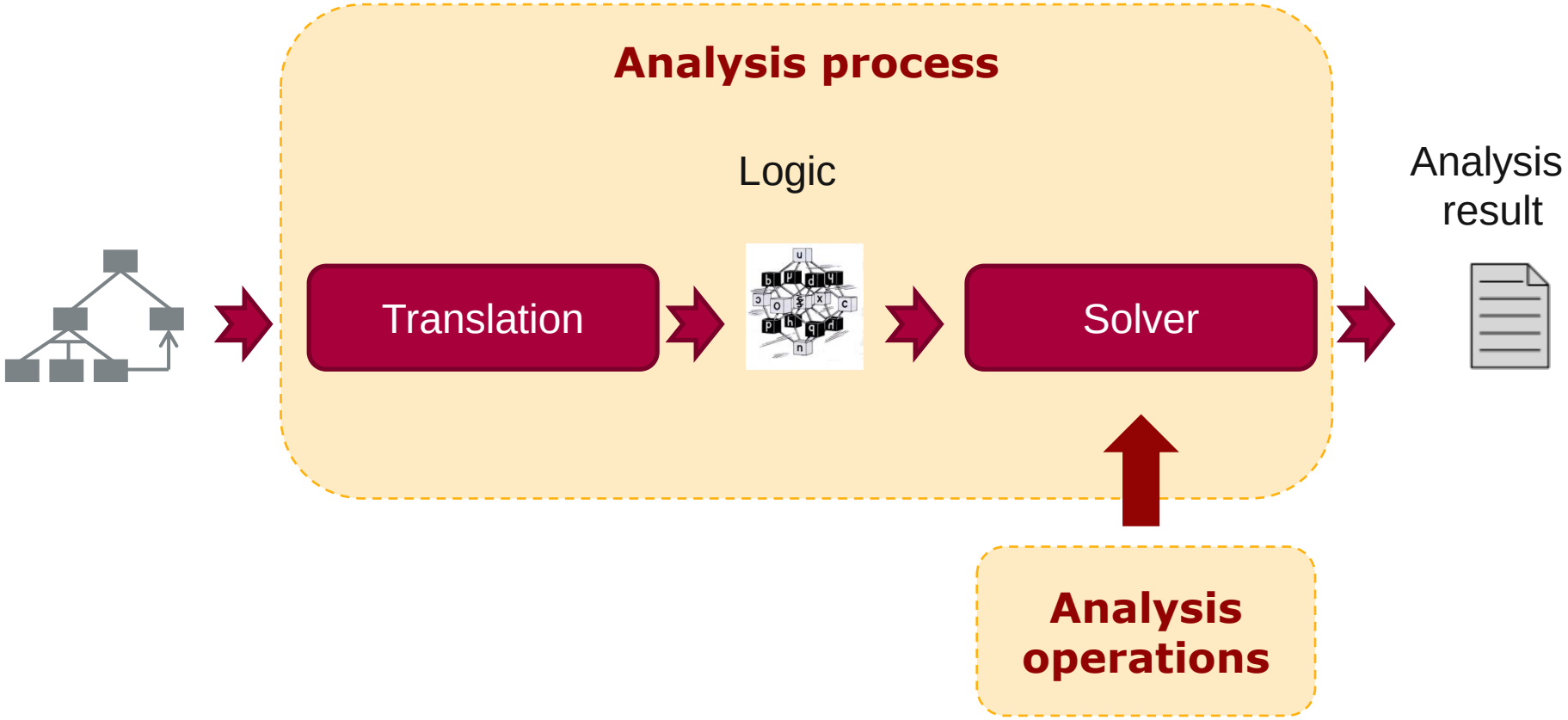
# Part II

# Challenge 1: Automated analysis of Feature Models

**Computer-aided, extraction of useful information from feature models**

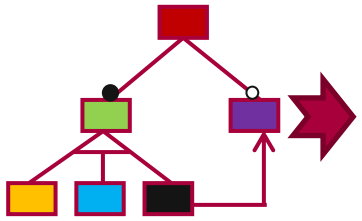


# Analysis process



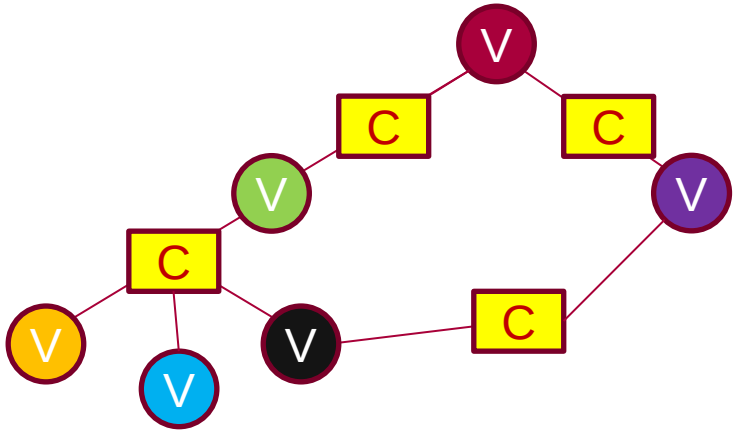
# Feature models as CSPs

Feature Model



Translation

Constraint Satisfaction Problem



# Feature models as CSPs

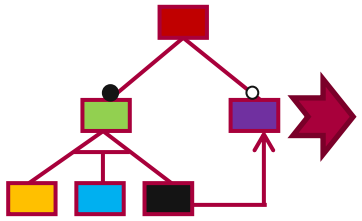
- A CSP is defined as:
  - A set of variables
  - A set of domains for those variables
  - A set of constraints restricting the values of the variables
- A CSP solvers is defined as:
  - A software tool that takes a CSP and find possible assignments of variables, if any, taking into account the constraints.

# CSP example

- CSP:
  - A set of variables: X, Y, Z
  - A set of domains for those variables:
    - X in {2,3}, Y in {4,6}, Z in {1,10}
  - A set of constraints restricting the values of the variables:
    - X + Y < Z
- Solutions for the CSP:
  - X = 2; Y = 4; Z = 6 
  - X = 3; Y = 5; Z = 8 
  - X = 3; Y = 6; Z = 9 

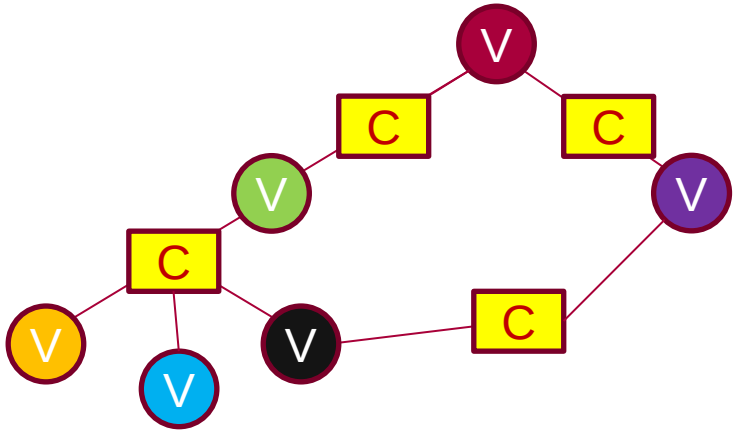
# Feature models as CSPs

Feature Model

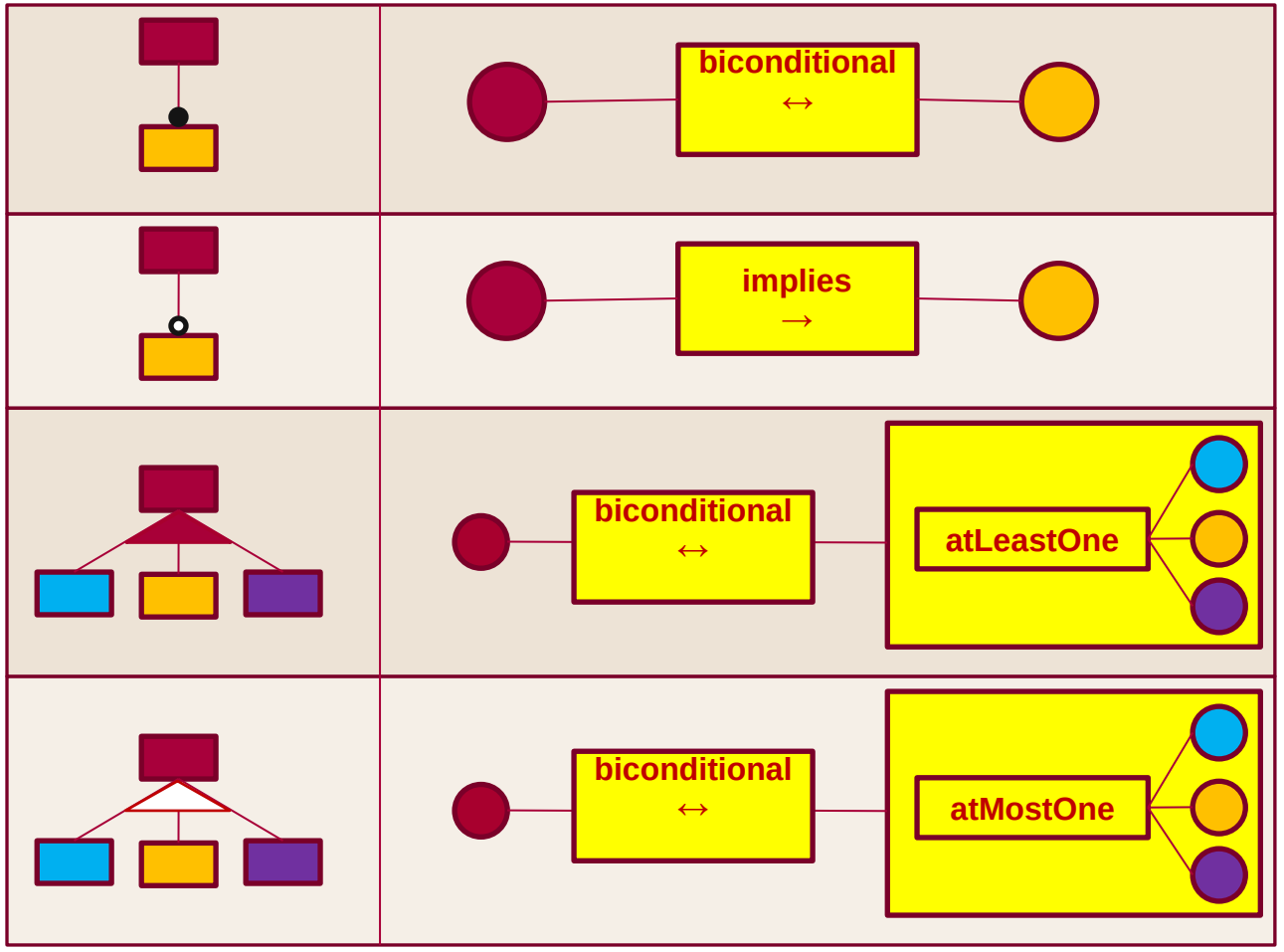


Translation

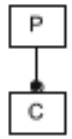
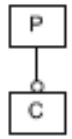
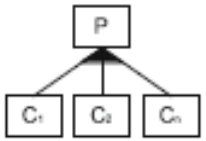
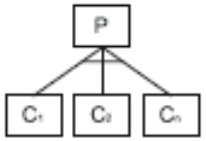
CSP



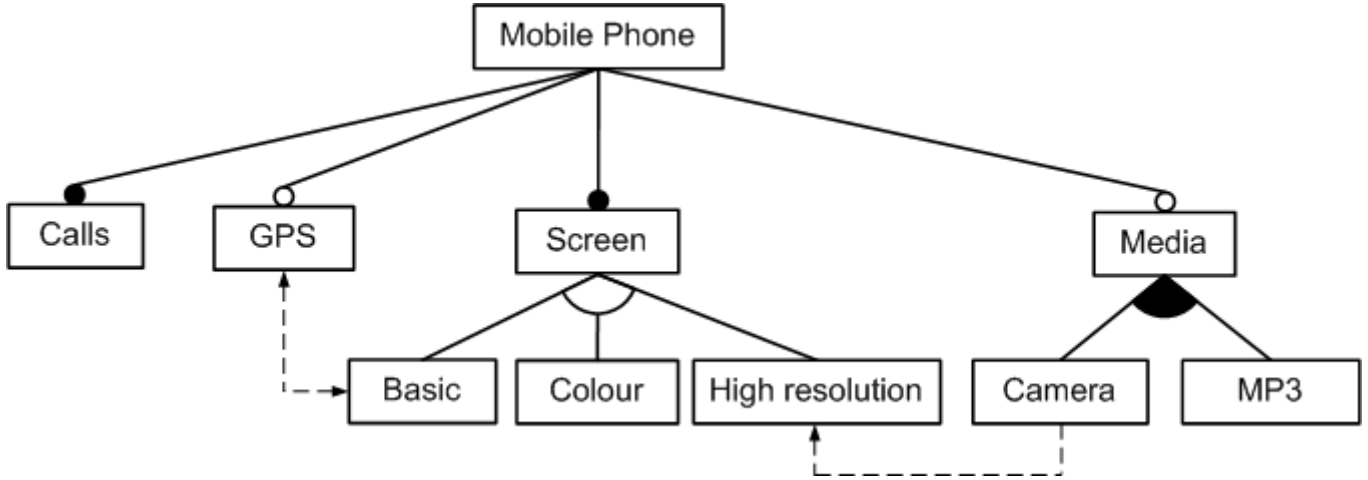
# Feature models as CSPs



# Feature models as Propositional formulas

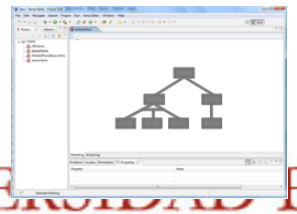
| Relationship | | PL Mapping |
|---|---|---|
| MANDATORY |   | $P \leftrightarrow C$ |
| OPTIONAL |   | $C \rightarrow P$ |
| OR |   | $P \leftrightarrow (C_1 \vee C_2 \vee ... \vee C_n)$ |
| ALTERNATIVE |   | $(C_1 \leftrightarrow (\neg C_2 \wedge ... \wedge \neg C_n \wedge P)) \wedge$<br>$(C_2 \leftrightarrow (\neg C_1 \wedge ... \wedge \neg C_n \wedge P)) \wedge$<br>$(C_n \leftrightarrow (\neg C_1 \wedge \neg C_2 \wedge ... \wedge \neg C_{n-1} \wedge P))$ |
| IMPLIES |   | $A \rightarrow B$ |
| EXCLUDES |   | $\neg(A \wedge B)$ |

# Automated analysis of feature models:
Computer-aided extraction of information from FMs

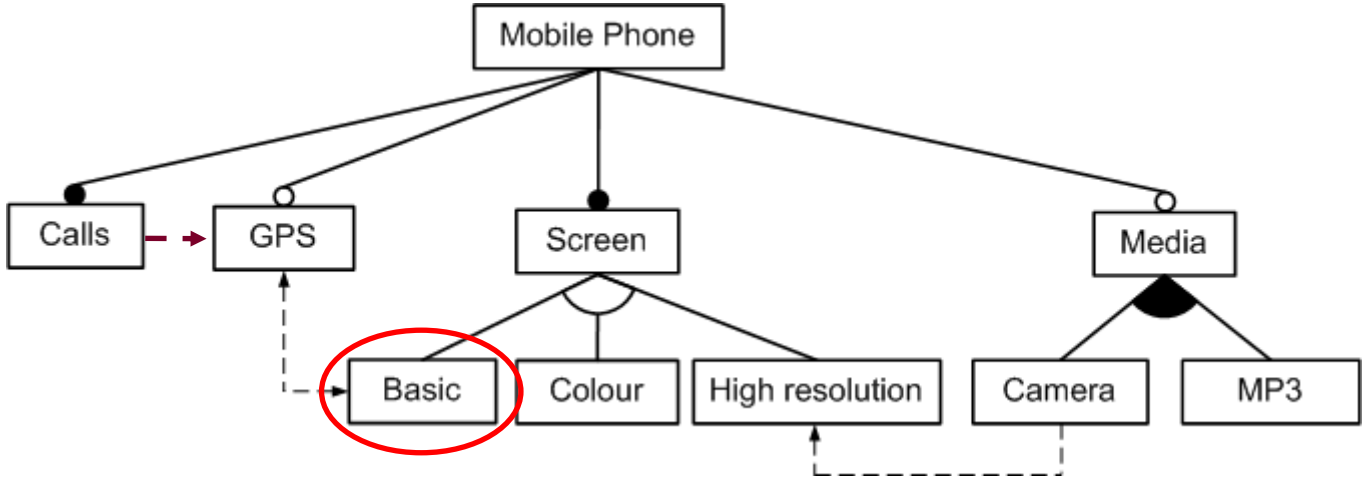



How many products?

14

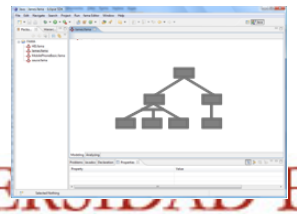


# Automated analysis of feature models:
Computer-aided extraction of information from FMs

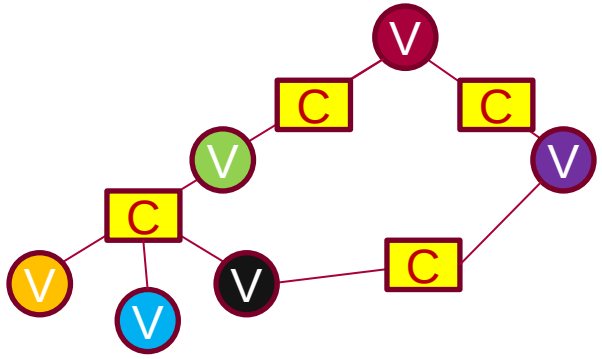



Any error?

Yes, feature "Basic" is dead



# Analysis implementations




- SAT
  - SAT4J
- BDD
  - JavaBDD
- General CSP
  - JaCoP
  - Choco

## Automated analysis of SPL

### Why it's an important problem?

Doing this by hand is an error prone task in large-scale feature models

---

Detecting properties at early stage of development and along all the life cycle

---

It's the base for other more complicated tasks, i.e. product *configuration*

# Challenge 1: Automated analysis of SPL:

## Computer-aided, extraction of useful information from SPL models

| | PL | | | | | | | | | | | | CP | | | | | | DL | | | Multi | | | Others | | | | | | | | | | | | | | | | No support | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Batory [5] | Czarnecki *et al.* [30] | Gheyi *et al.* [37] | Mannion *et al.* [51, 52] | Mendonca *et al.* [57] | Mendonca *et al.* [56] | Sun *et al.* [74] | Thüm *et al.* [75] | van der Storm [86, 87] | Zhang *et al.* [102, 101] | Zhang *et al.* [103] | Yan *et al.* [100] | Benavides *et al.* [10, 11, 12] | Benavides *et al.* [15] | Djebi *et al.* [34] | Trinidad *et al.* [78, 76] | White *et al.* [99] | White *et al.* [97] | Abo Zaid *et al.* [1] | Fan *et al.* [35] | Wang *et al.* [92, 93] | Benavides *et al.* [14] | Benavides *et al.* [16] | Segura [70] | Bachmeyer *et al.* [4] | Cao *et al.* [20] | Fernandez *et al.* [36] | Hemakumar [41] | Gheyi *et al.* [38] | Kang *et al.* [43] | Mendonca *et al.* [55] | Osman *et al.* [59, 60] | Salinesi *et al.* [66] | Van den Broek *et al.* [84] | Van Deursen *et al.* [88] | Von der Massen *et al.* [90] | Von der Massen *et al.* [91] | White *et al.* [98, 96] | Batory *et al.* [7] | Schobbens *et al.* [42, 68, 69] | Trinidad *et al.* [80] | Von der Massen *et al.* [89] |
| Void feature model | + | + | | + | | + | + | | + | + | + | + | + | + | + | + | | | | + | + | + | + | | | + | | + | ⊕ | | + | + | + | | | | | ~ | ~ | |
| #Products | | + | | ⊕ | | | | | | | | | + | + | + | | | | | | | + | + | + | | | + | | | ⊕ | | | | + | ⊕ | | + | | | ~ | ~ | |
| Dead features | | ~ | | | | + | | | | + | + | + | | | | + | | | + | | | | | | | | | | ⊕ | | + | + | + | | | | | ~ | ~ | ~ |
| Valid product | + | + | + | + | | | + | | | | | | | | | | + | | | | + | | | | + | | | | | ⊕ | | | | | | + | | ~ | ~ | ~ | |
| All products | + | | + | ⊕ | | | + | | | | | | + | | | | | | | | | | + | | | + | | | | ⊕ | | | | + | ⊕ | | | | ~ | ~ | |
| Explanations | + | ~ | | | | | + | | | | | | | | | + | | | + | | + | | | | | | | | ⊕ | | + | | + | | | | | ~ | ~ | |
| Refactoring | | | + | | | | ⊕ | + | | | | | | | | | | | | | + | | | | | | | + | ⊕ | | | | | | | | | ~ | ~ | ~ | |
| Optimization | | | | | | | | | | | | | ⊕ | | + | | | | | | | | | | | | | | | | + | | | | | | + | ~ | | ~ | |
| Commonality | | | | | | | | | | | | | ⊕ | | | + | | | | | | | + | | | | + | | | | | | | | | | | | | ~ | |
| Filter | | + | | | | | | | | | | | ⊕ | | + | | | | | | | | | | | | | | | | | | | + | | | | | | ~ | |
| Valid partial configuration | + | + | | | | | | | + | | | | | | | | | | | | | | | | | | | | | ⊕ | | | | | | | | | | ~ | |
| Atomic sets | | | | | + | | | | | ⊕ | | | | | | | | | | | | | | + | | | | | | | + | | | | | | | | | | |
| False optional features | | | | | | | | | | ⊕ | + | | | | | + | | | | | | | | | | | | | | | | | | | | | | | | | ~ | ⊖ |
| Corrective explanations | | | | | | | | | | | | | | | | | + | | | | | | | | | | | | | | + | | | | | | | | | | ⊖ |
| Dependency analysis | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ⊕ | + | | | | | | | | | | | |
| ECR | | | | ⊕ | + | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Generalization | | | ⊕ | | | | | + | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Core features | | | | | + | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ⊖ | |
| Variability factor | | | | | | | | | | | | | ⊕ | | | | | | | | | | | | | | | | | | | | | | | | | | | | ~ | |
| Arbitrary edit | | | | | | | | + | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Conditional dead features | | | | | | | | | | | | | | | | | | | | | | | | | | | + | | | | | | | | | | | | | | |
| Homogeneity | | | | | | | | | | | | | | | | | | | | | | | | | | + | | | | | | | | | | | | | | | |
| LCA | | | | | + | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Muti–step configuration | | | | | | | | | | | | | | | | | | + | | | | | | | | | | | | | | | | | | | | | | | | |
| Roots features | | | | | + | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Specialization | | | | | | | | + | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Degree of orthogonality | | ~ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Redundancies | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ~ |
| Variant features | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ~ | |
| Wrong cardinalities | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ~ | |
| Feature model notation | B | C | B | B | B | B | B | B | B | B | C | B | B | C | C | B | B | B | B | B | B | B | C | B | B | B | C | B | B | B | C | C | B | B | B | B | B | B | C | C | B |
| Extended feature model | | | | | | | | | | | | | + | | + | | + | + | + | | + | | | | | | | | + | | + | | | | | | | + | | + | | |
| Formalization | | | + | | + | + | + | + | | + | + | | + | | | + | | + | | + | | | | | + | | + | | | | + | | | | | + | | + | | | |

+ Supported ~ No support ⊕ Supported(first reference) ⊖ No support (first reference) B Basic feature model C Cardinality–based feature models

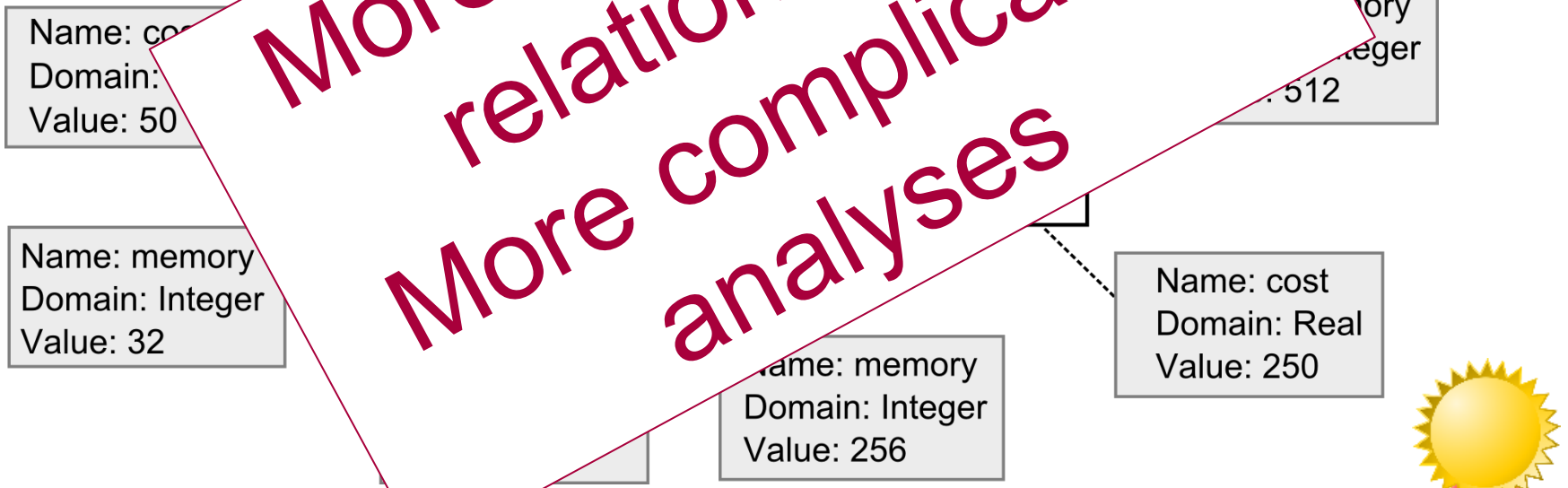
Table 3: Summary of operations and support

David Benavides, Sergio Segura, Antonio Ruiz Cortés: Automated analysis of feature models 20 years later: A literature review. Inf. Syst. 35(6): 615-636 (2010)

# Are boolean feature models enough?

# Challenge 1.1: Automat… Feature Model…

More complicated relationships
More complicated analyses







• David Benavides, Pablo Trinidad Martín-Arroyo, Antonio Ruiz Cortés: Automated Reasoning on Feature Models. CAiSE 2005: 491-503

• F Roos-Frantz, D Benavides, A Ruiz-Cortés, A Heuer, K Lauenroth Quality-aware analysis in product line engineering with the orthogonal variability model. Software Quality Journal

# Parte III

# Variability implementation

- Templates
- Compilación
- Preprocesamiento
- Cargadores de clases
- Modularización

# Variabilidad

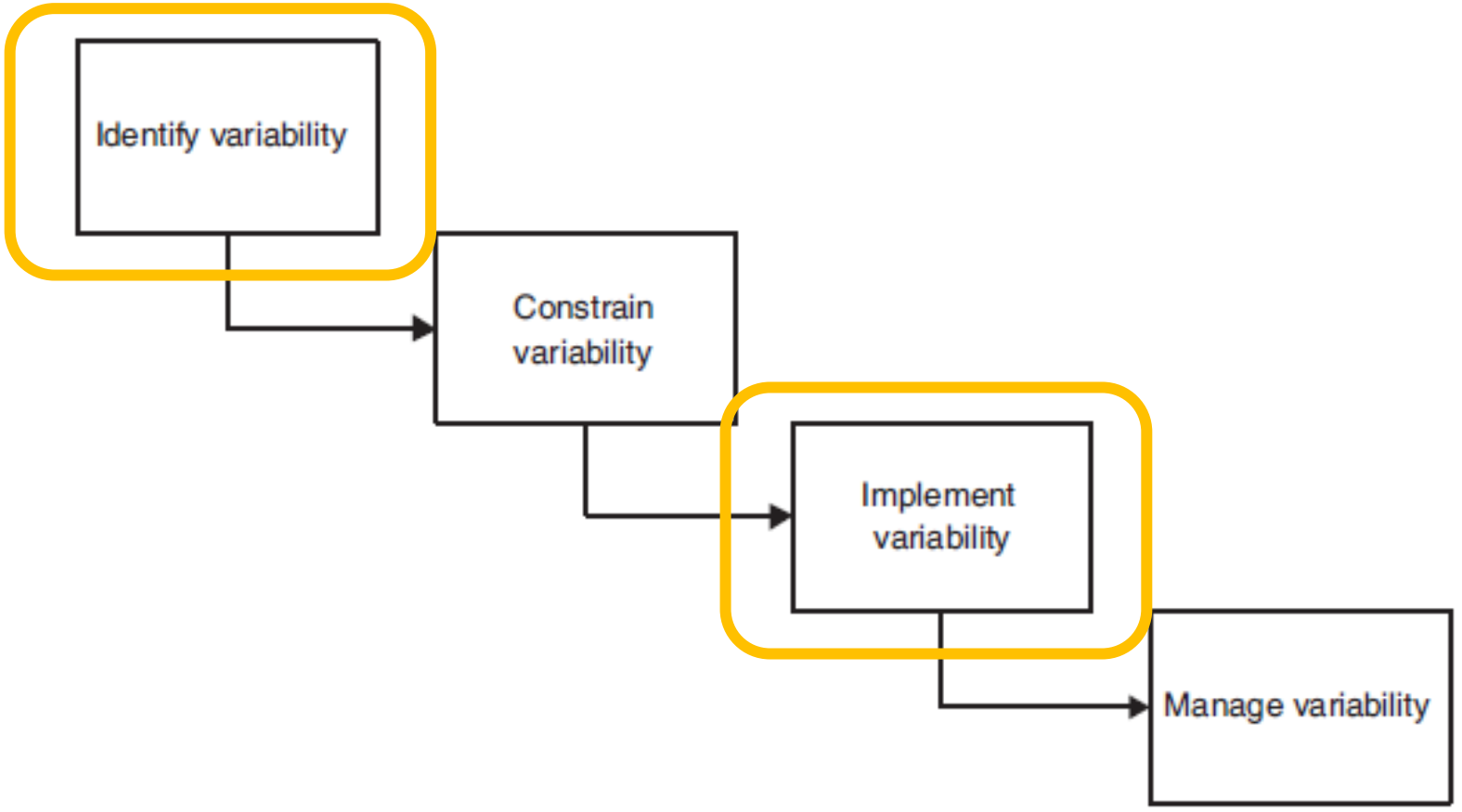
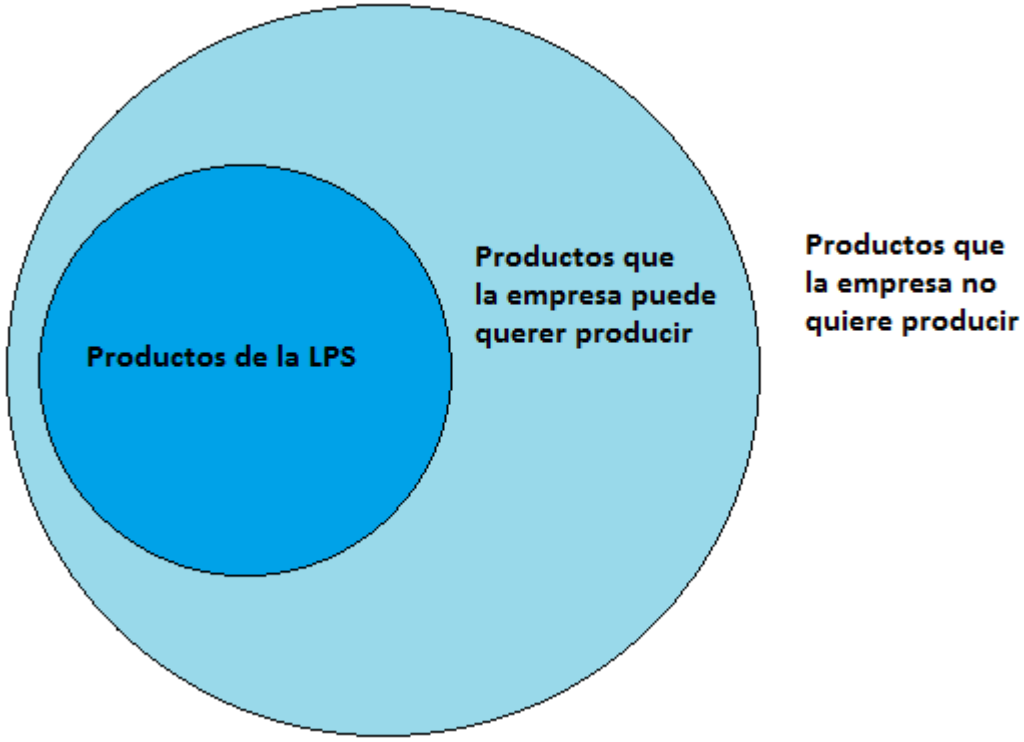



Figure 1. Steps for introducing variability.

# El alcance de una SPL



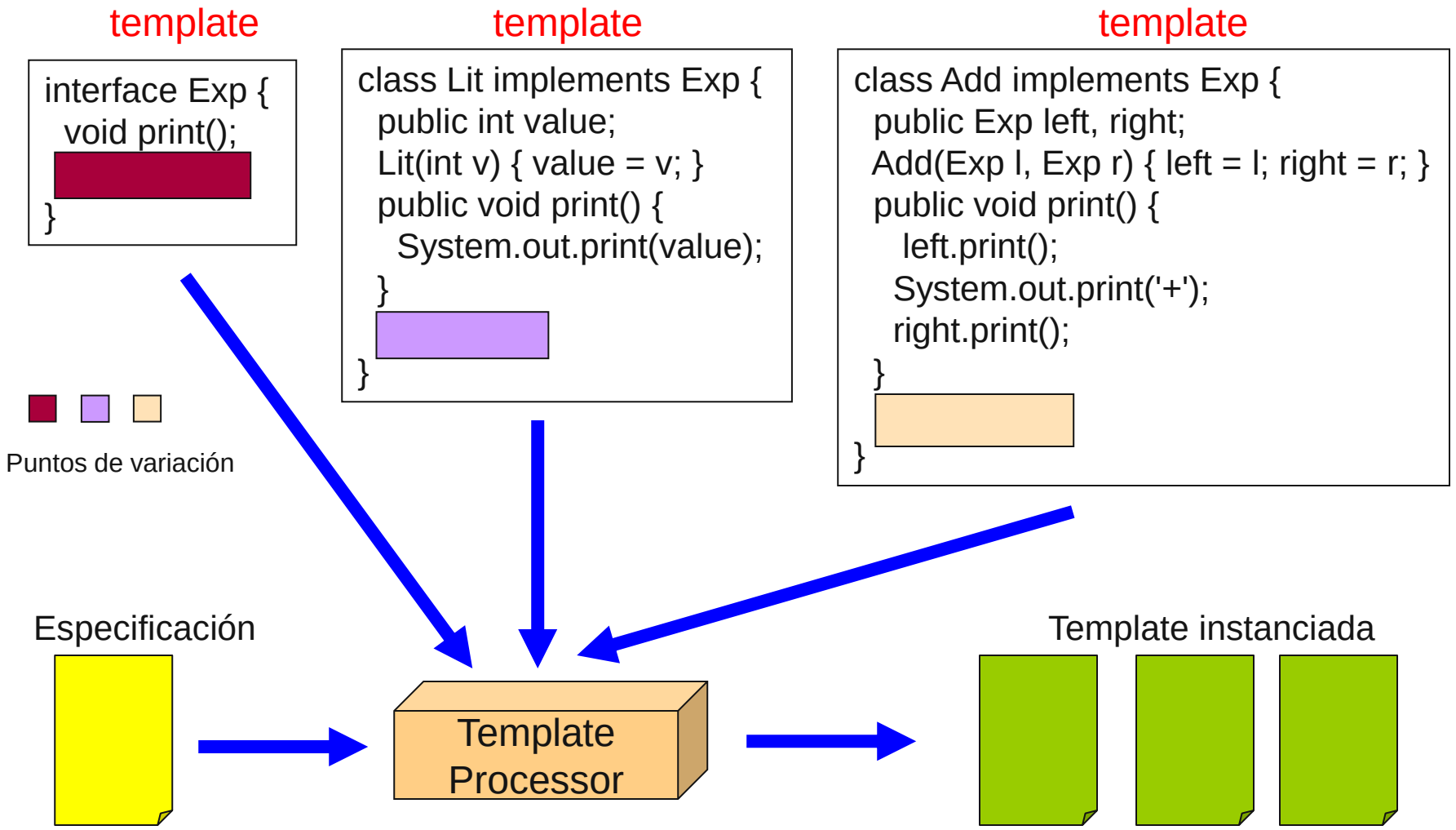
# Implementación de la variabilidad

- **Plantillas**
- Compilación
- Preprocesamiento
- Cargadores de clases
- Modularización
- DSL

# ¿Cómo implementar la variabilidad?

- Técnicas basadas en motores de plantillas (*template engines, frames*)
- Se pueden definir como motores de “copy & paste”
- Distintas alternativas


# Proceso general



# XVCL

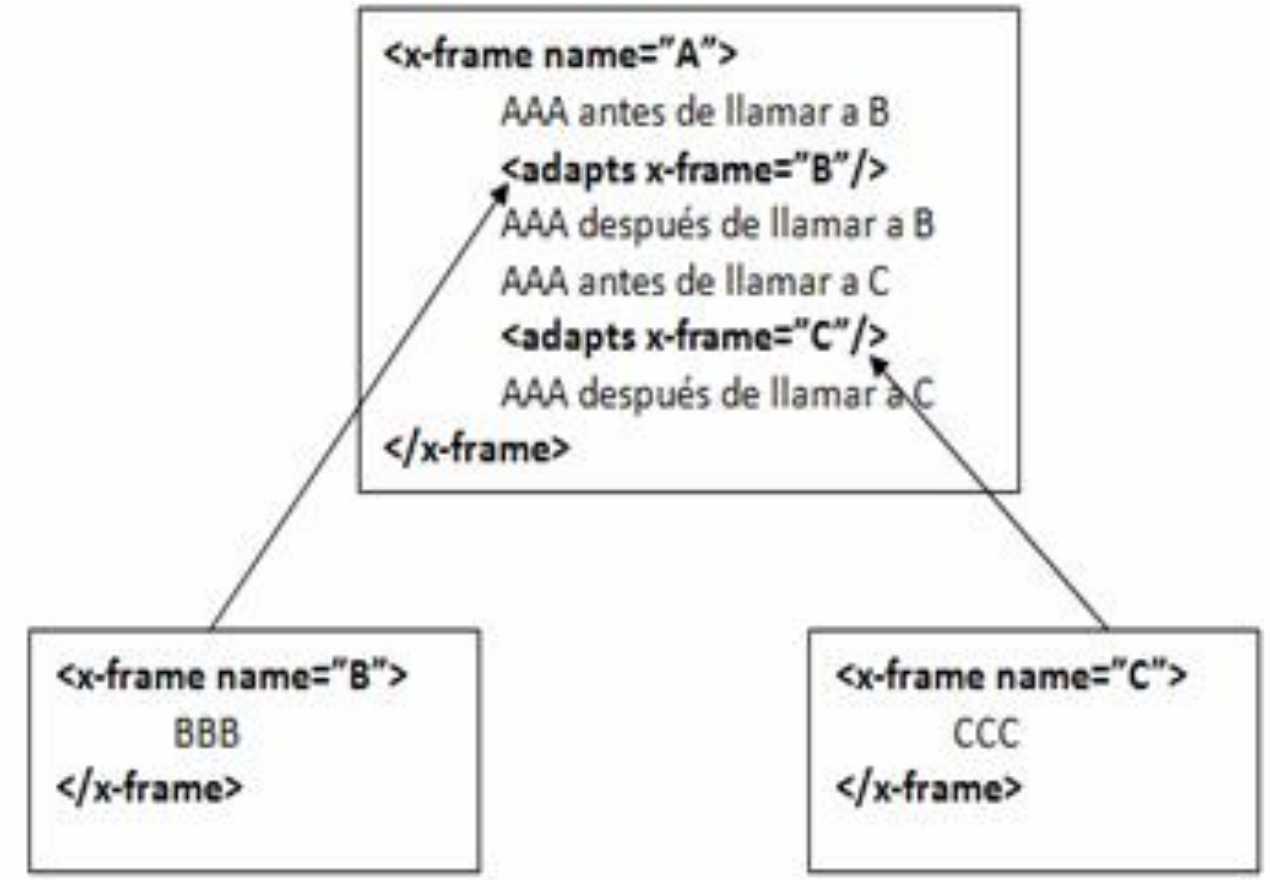
- XML-based Variant Configuration Language
- Stanislaw Jarzabek (Universidad de Singapur)
- Basado en tecnología de marcos (*frames*)
- Complemento a lenguaje de programación convencional

# XVCL

- Funcionamiento:
  - X-frames
  - X-framework
  - Puntos de variación
  - Comandos

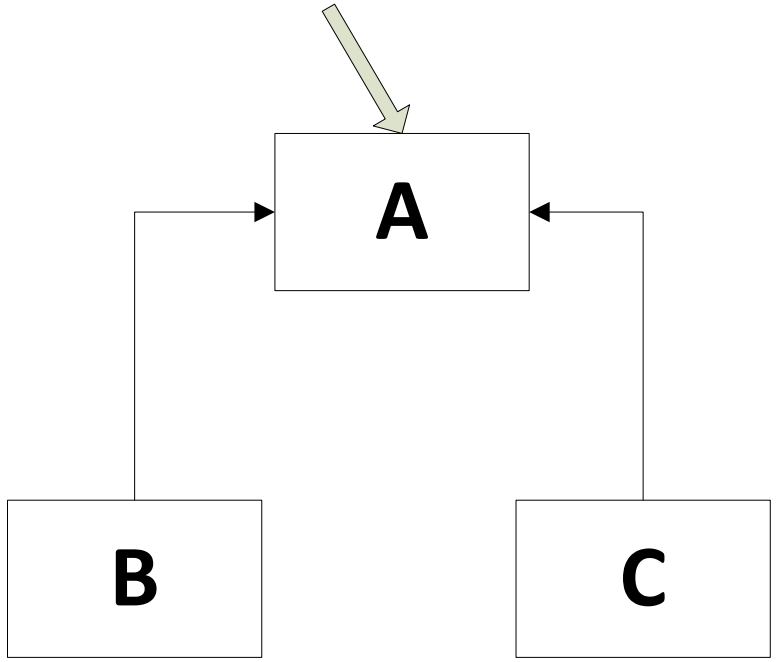
# XVCL

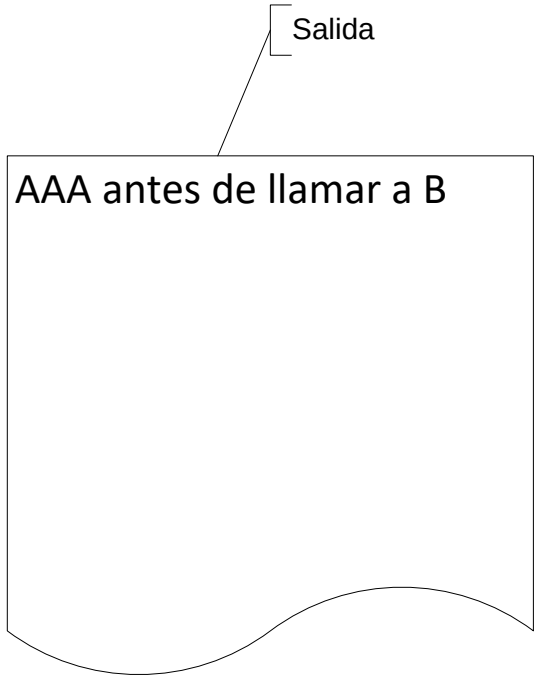
- Tenemos el siguiente x-framework:



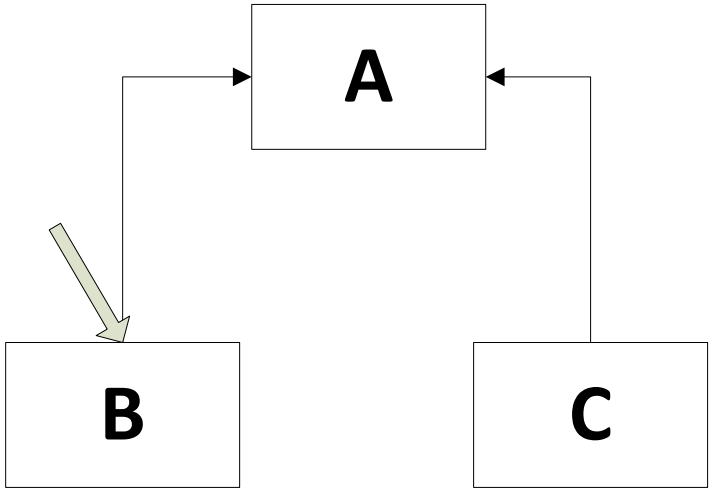
# XVCL

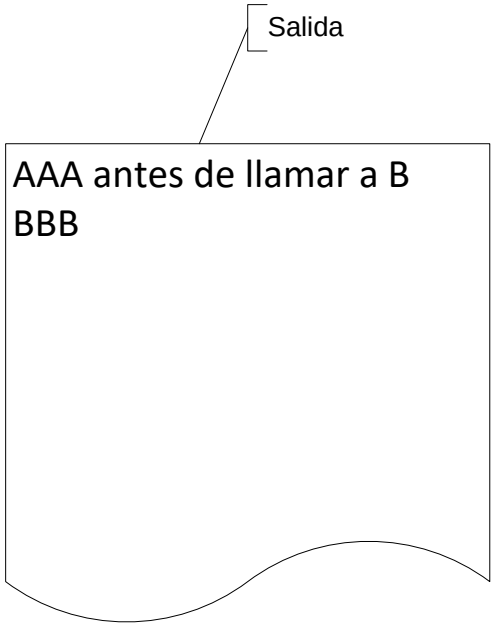
- Ejecución – paso 1



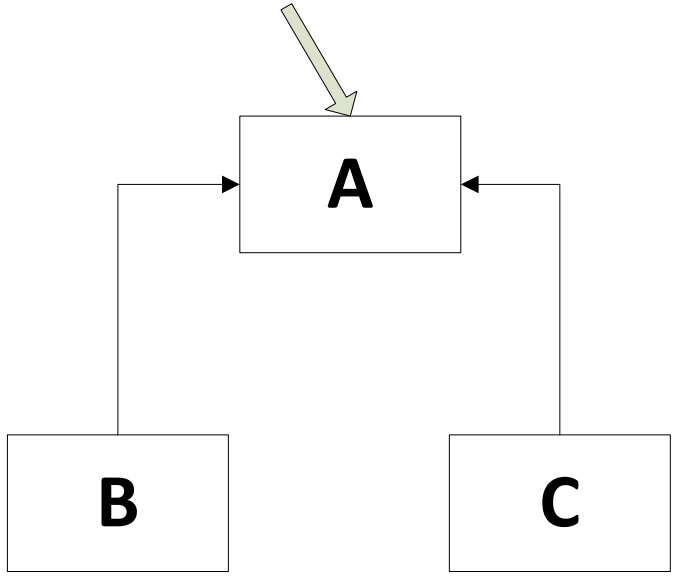
- Ejecución – paso 2

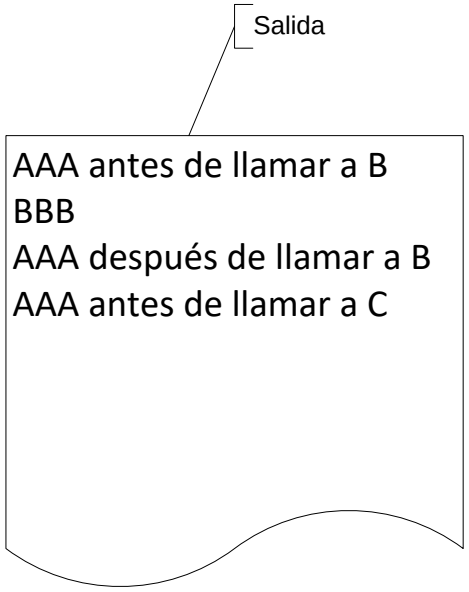


# XVCL

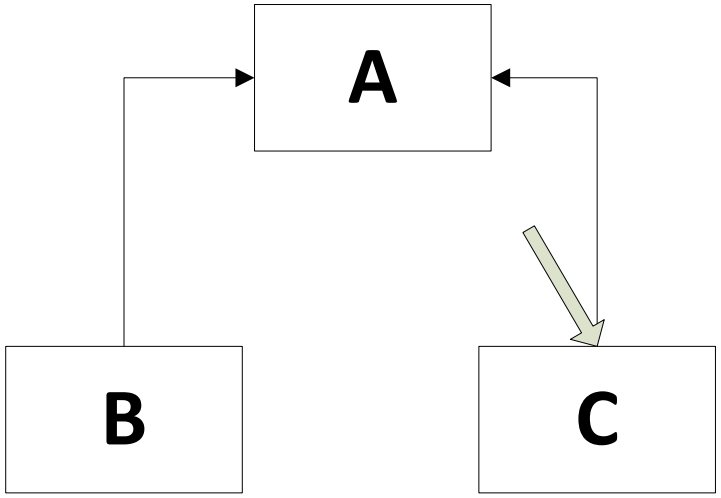
- Ejecución – paso 3

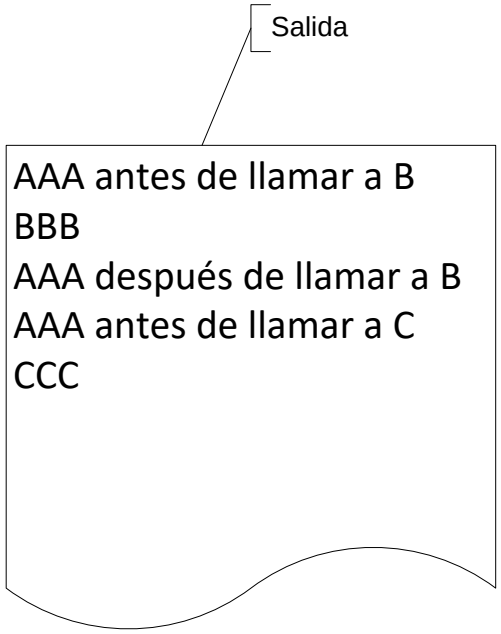


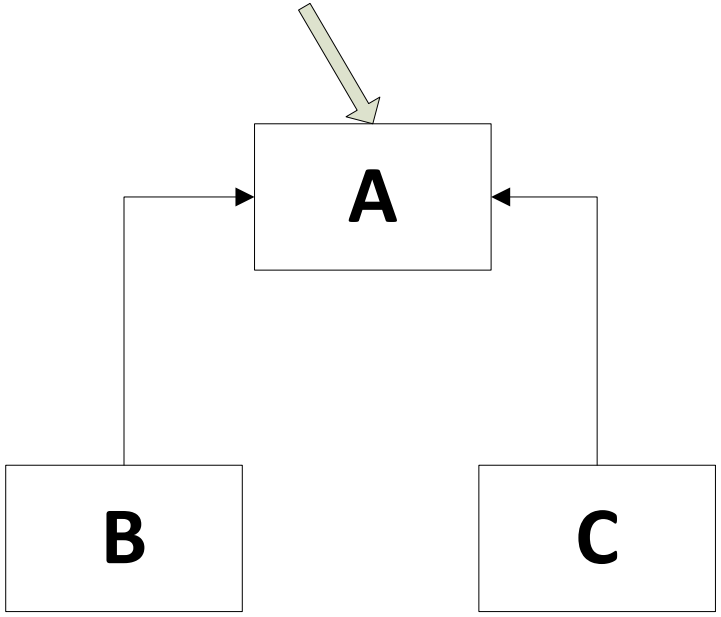
# XVCL

- Ejecución – paso 4



- Ejecución – paso 5




Salida

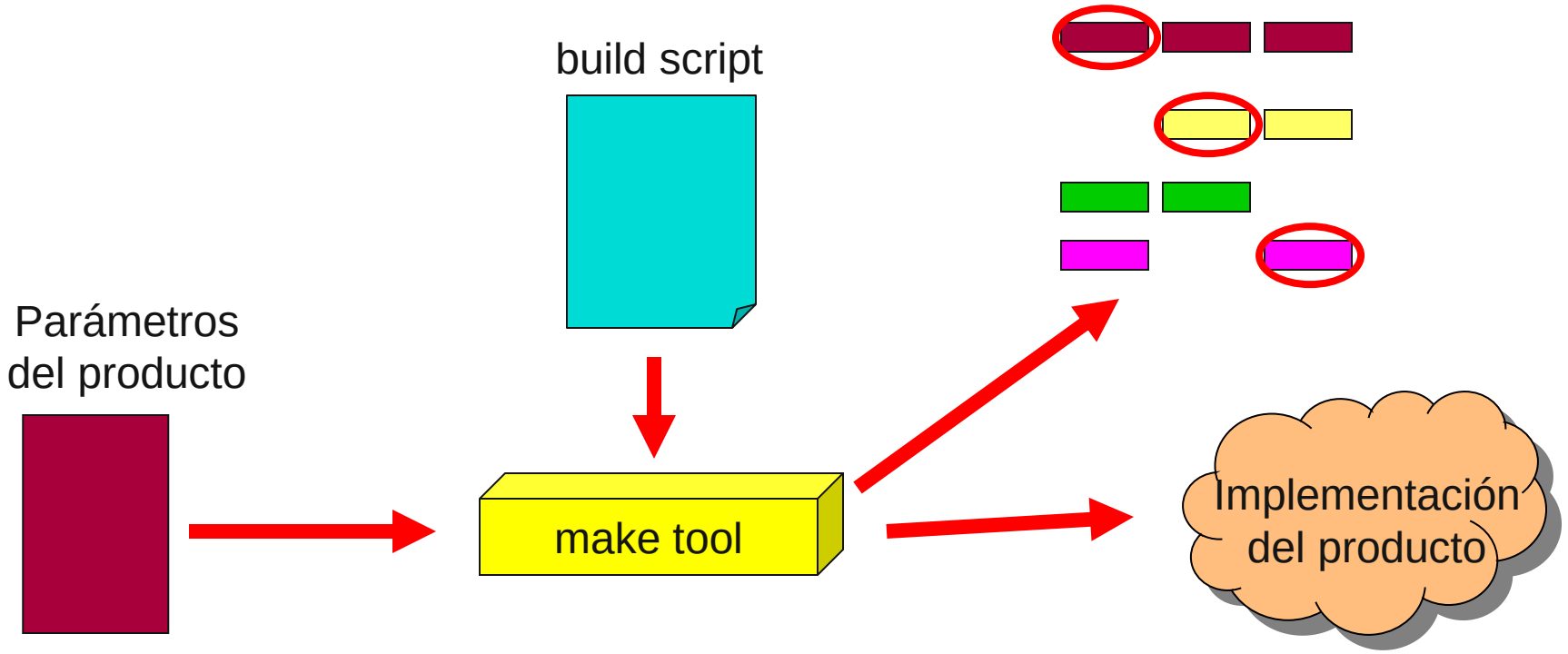
```
AAA antes de llamar a B
BBB
AAA después de llamar a B
AAA antes de llamar a C
CCC
AAA después de llamar a C
```

# Implementación de la variabilidad

- Plantillas
- **Compilación**
- Preprocesamiento
- Cargadores de clases
- Modularización

# Técnicas de compilación

- Algunas herramienta como make, ant, maven,... se pueden usar para implementar la variabilidad en tiempo de compilación



# Técnicas de compilación

- Ventajas
  - Son fáciles de implementar y útiles cuándo hay puntos simples de variación

- Desventajas
  - Implementar todas las variantes de una SPL en un solo script puede resultar muy tedioso si no directamente imposible, especialmente cuándo haya muchas variantes

# Implementación de la variabilidad

- Plantillas
- Compilación
- **Preprocesamiento**
- Cargadores de clases
- Modularización

# Preprocesamiento

- Directivas del compilador para incluir o no ciertos fragmentos de código

- Es la primera y más antigua técnica para soportar variabilidad

- Está soportada por lenguajes como C, pero también se pueden introducir en Java
- Ventajas
  - Simple, viene por defecto en algunos lenguajes y compiladores.
- Desventajas
  - Parecida a las desventajas de las plantillas, incluso más graves.

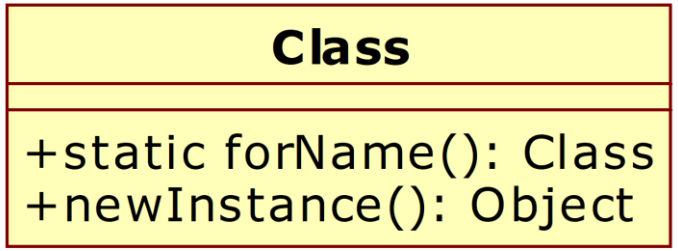
# Implementación de la variabilidad

- Plantillas
- Compilación
- Preprocesamiento
- **Cargadores de clases**
- Modularización

# Cargadores de clases

- Classloader
  - Forma parte del JRE estándar de Java
  - Se cargan de manera dinámica clases en la JVM

- Se proporciona un cargador de clases por defecto
  - Este cargador se puede refinar y/o extender.




```
Driver dBDriver = (Driver) Class.forName(driverName)
    .newInstance();
DriverManager.registerDriver(dBDriver);
```

# Cargadores de clases

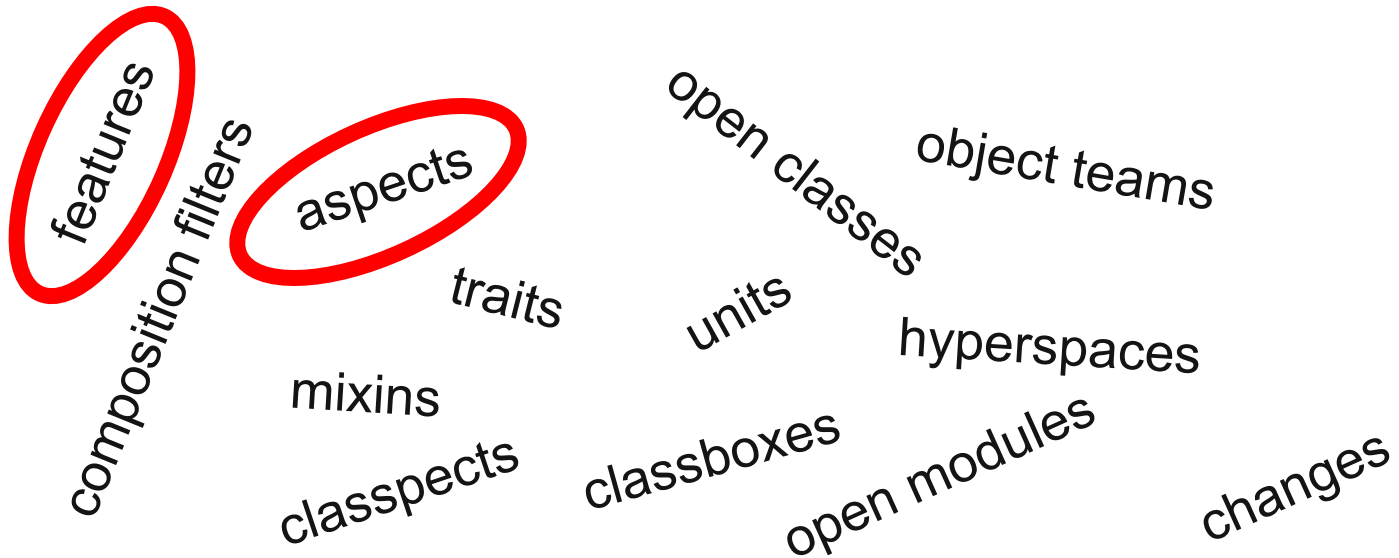
- Ventajas
  - Son una manera flexible de implementar variabilidad en tiempo de ejecución
- Desventajas
  - Implementar un cargador de clases no es una tarea trivial
  - Solo funciona para programas escritos en lenguajes que soporten reflexión
  - La legibilidad del código no es trivial

# Implementación de la variabilidad

- Plantillas
- Compilación
- Preprocesamiento
- Cargadores de clases
- **Modularización**

# Técnicas de modularización

- Hay muchos lenguajes de programación que se proponen para solventar los problemas de los lenguajes OO
- El punto clave es la granularidad
  - Se parte de la constatanción de que una clase es muy pequeña para construir un sistema si a este sistema hay que añadirle incrementos de funcionalidad.



# Bibliografía

JOURNAL OF OBJECT TECHNOLOGY
Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering. ©JOT, 2009
Vol. 8, No. 6, September–October 2009

## Virtual Separation of Concerns – A Second Chance for Preprocessors

**Christian Kästner,** School of Computer Science, University of Magdeburg, Germany
**Sven Apel,** Department of Informatics and Mathematics, University of Passau,

SOFTWARE—PRACTICE AND EXPERIENCE
*Softw. Pract. Exper.* 2005; **35**:705–754
Published online 1 April 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.652

## A taxonomy of variability realization techniques‡

Mikael Svahnberg[1,*,†], Jilles van Gurp[2] and Jan Bosch[3]

UNIVERSIDAD D SEVILLA

# Bibliografía