


Software Product Line Engineering

modeling and managing variability of software intensive systems

Dr. Mathieu Acher

email: macher@fundp.ac.be

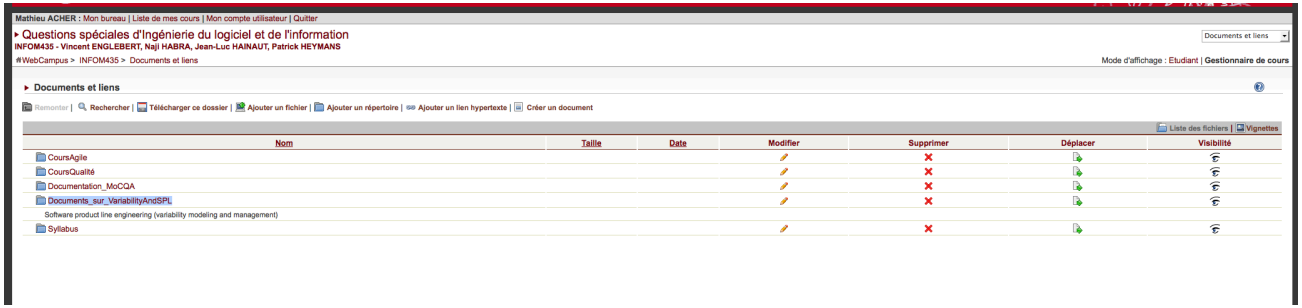
Prof. Patrick Heymans

University of Namur

PReCISE Research Centre

Material

- http://www.fundp.ac.be/etudes/cours/page_view/INFOM435/
  - Folder: “Documents_sur_VariabilityAndSPL”
  - Slides, exercises, evaluation



- Email: macher@fundp.ac.be

I reuse some material from other colleagues (Czarnecki, Kästner, etc.)

I try to mention references as much as possible

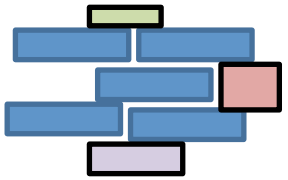
(you can find/read them on the web or simply ask me!)

[Czarnecki et al. (GPCE'05)]

Previously

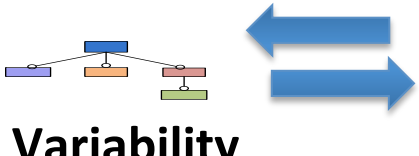
- **Software product line engineering**
  - Mass customization
  - Family of software intensive systems
  - Systematic reuse
  - Domain engineering
  - Variability management

- **Variability** everywhere
  - Applied and applicable to many industries and domains



- **Modeling** and **implement** variability: an overview
  - More to come!

- Running **project**
  - Re-engineering a car configurator



Variability Model

Have you made your homework?

- Forming **groups**
  - same until the end of the course!

- Exercice
  - Identify three projects that look like software product lines
  - Justify it!
    - Identify **commonality**
    - Identify **configuration options** or **features**
    - Report **how variability is implemented**
      - Preprocessors? Design patterns?

Software product lines

~=

Variability intensive systems

Today

- Software product line engineering: a generic **framework**
  - conceptual framework that serves as guiding principles
- **Domain engineering** pays off
- **Domain** and **Application** engineering
- **Variability** Management
  - Structuring the modeling space
  - An overview of generative techniques

Software Product Line Engineering

The development of a

family of software systems

differs from the development of

a **single** software system

THANKS CAPTAIN
OBVIOUS

« The development of a **family** of software systems differs from the development of a **single** software system »

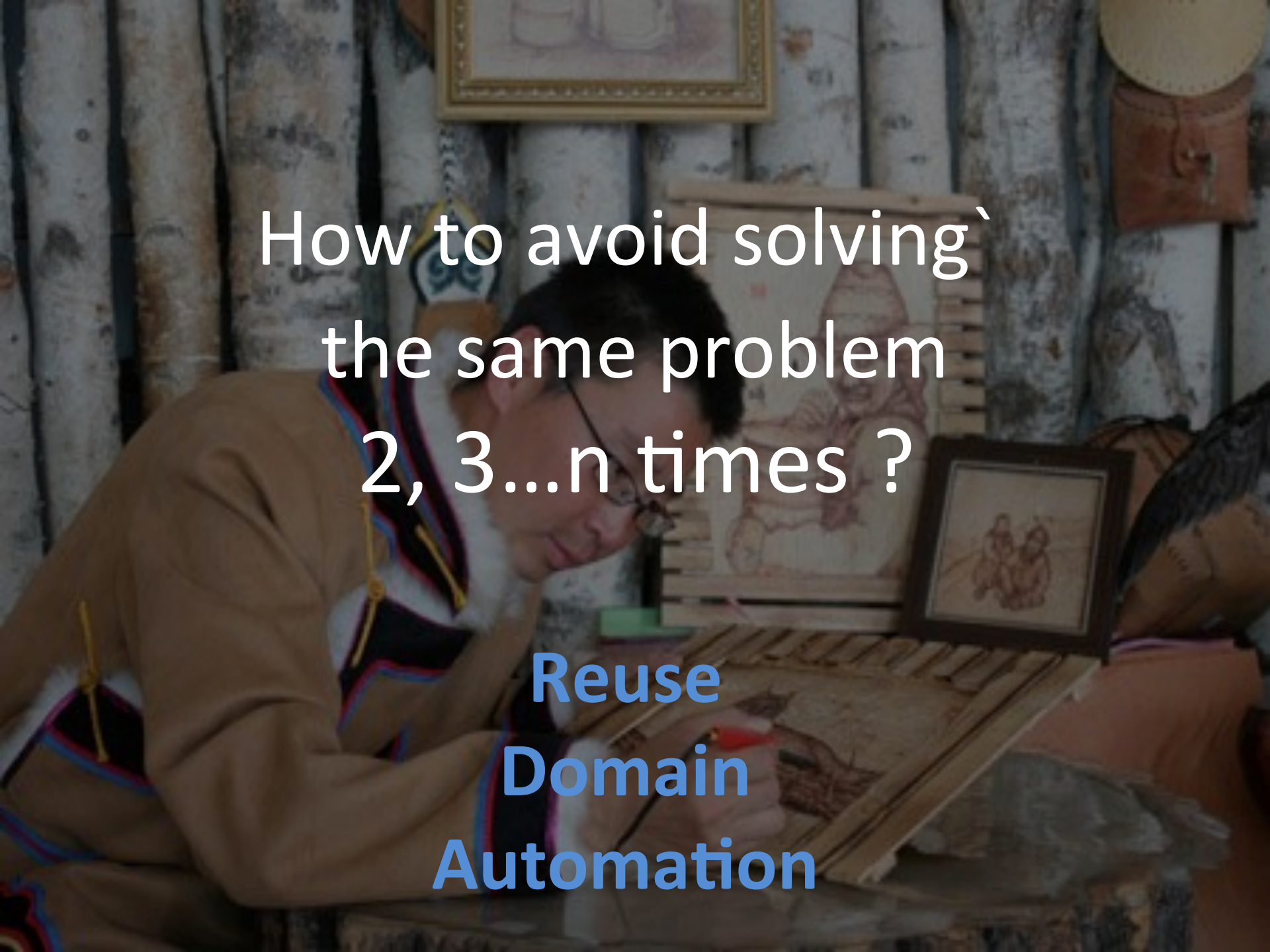
Reuse

Customization

Automation

Commonality

Variability


How to avoid solving`
the same problem
2, 3…n times ?
Reuse
Domain
Automation

Assembly Line and Mass Customization

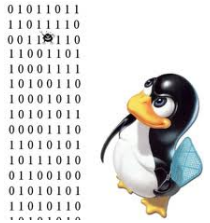
Reuse and Mass Customization


Starting from scratch?

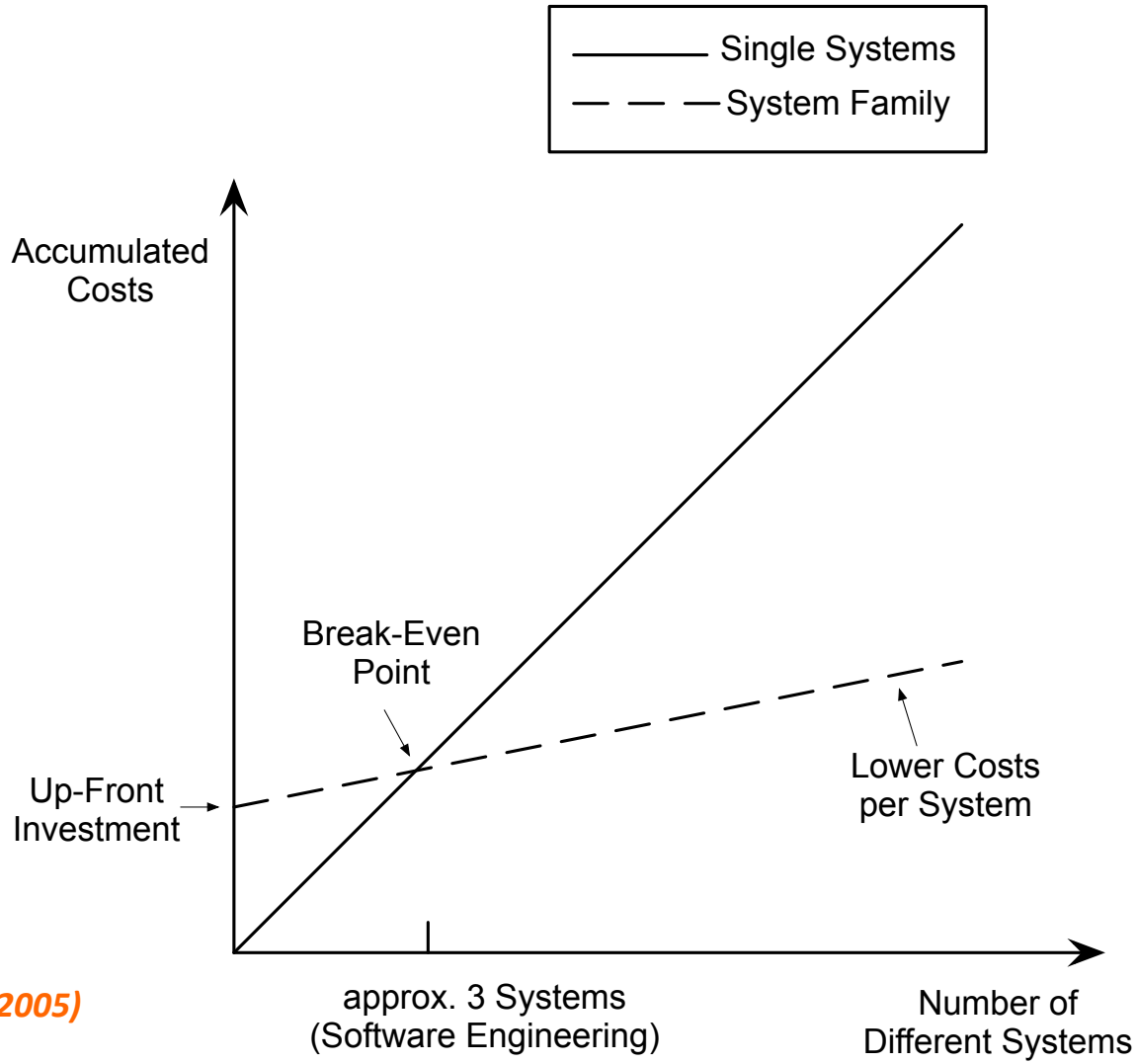

You cannot start from scratch

"a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [Clements et al., 2001]

Software Product Lines

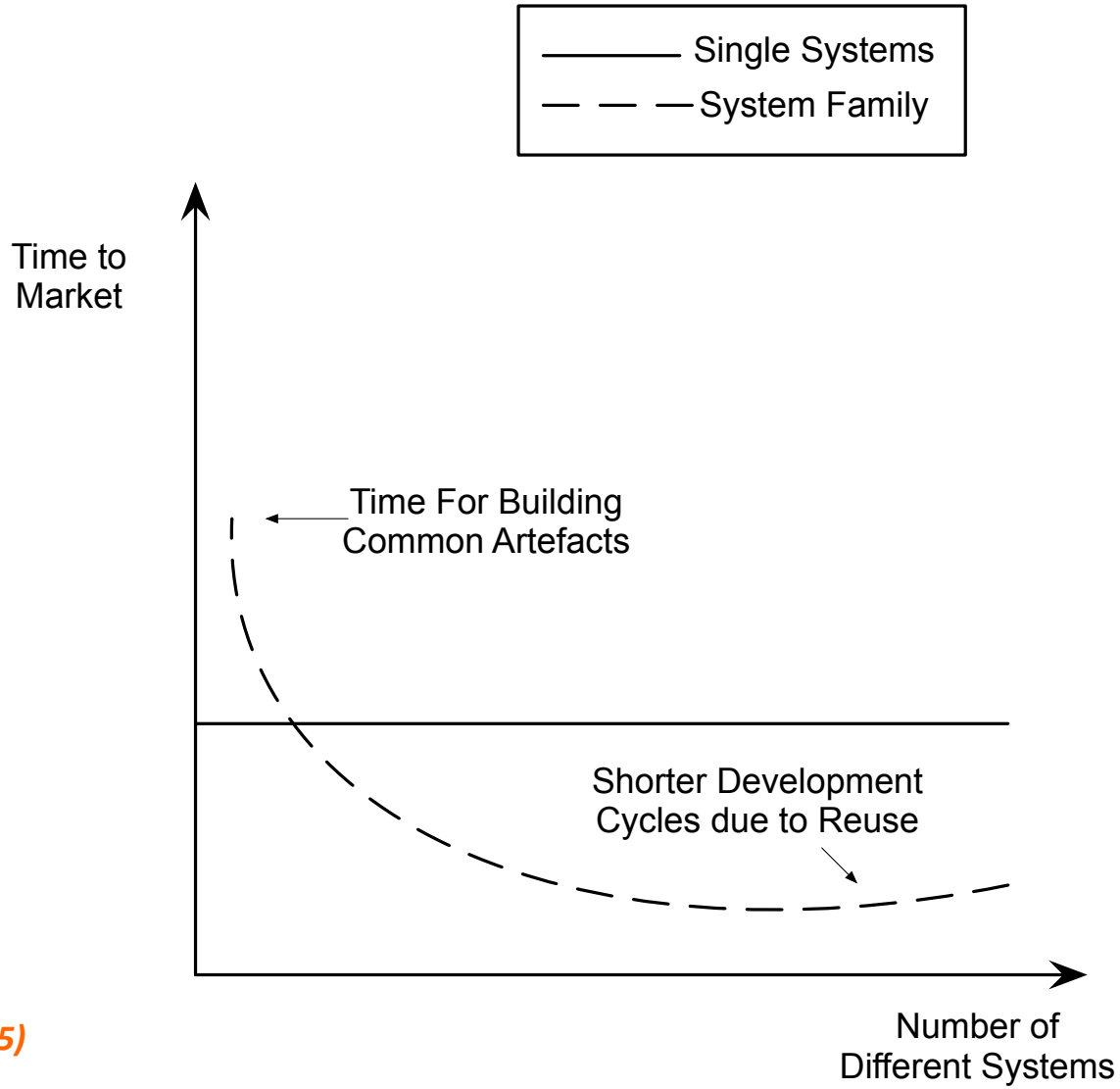


Promises of Software Product Line Engineering



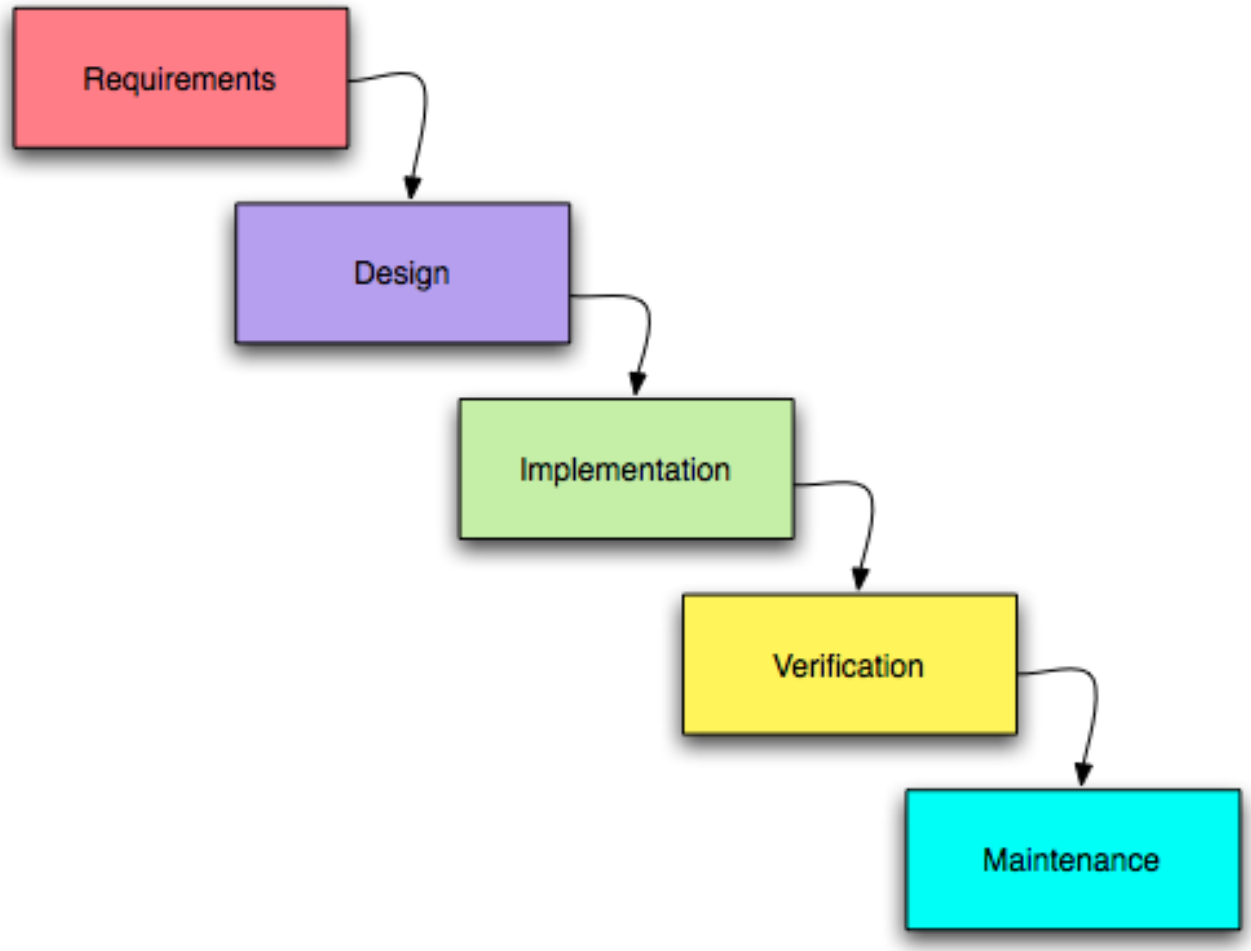
Klaus Pohl (2005)

Promises of Software Product Line Engineering

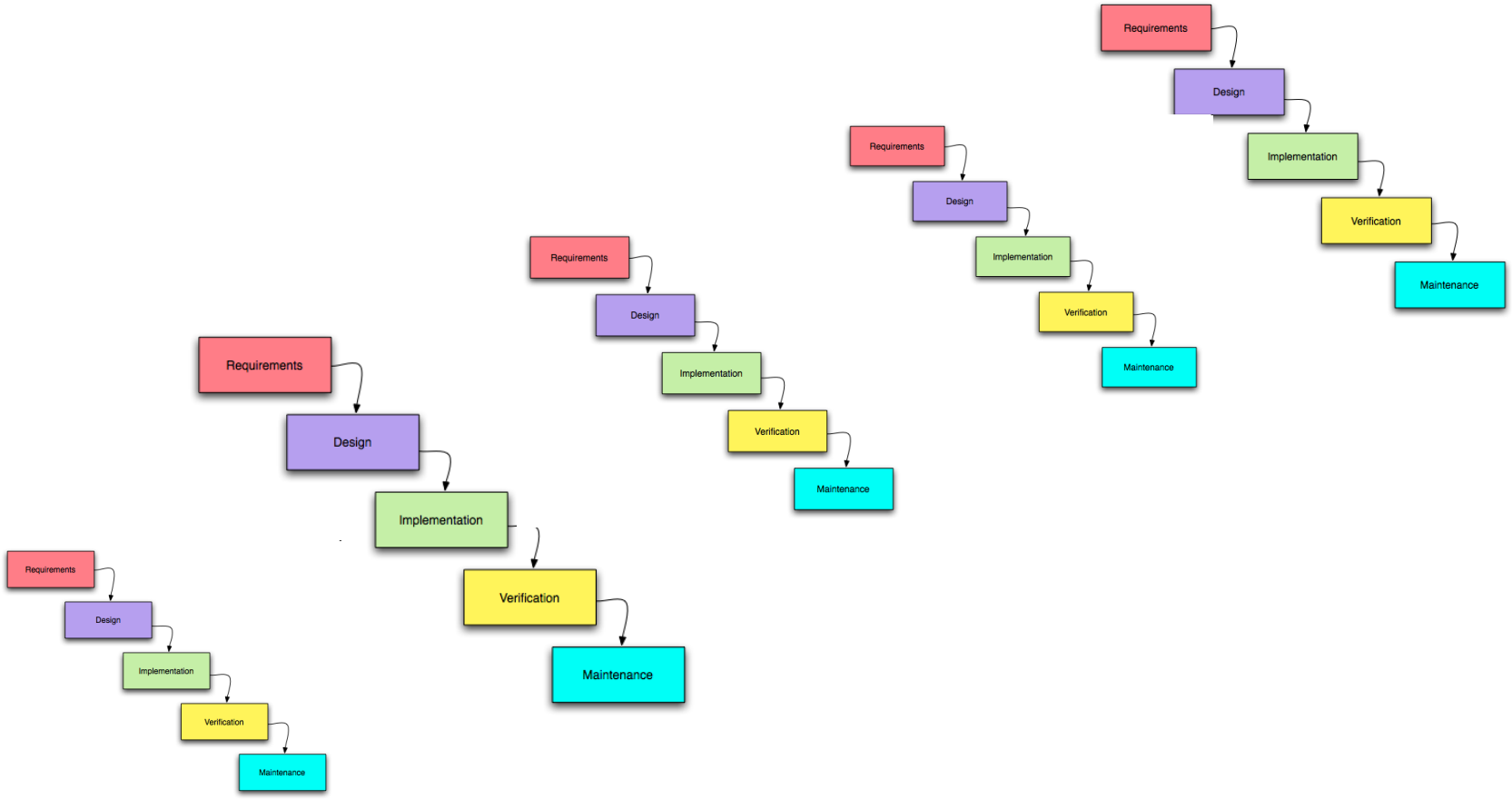


Klaus Pohl (2005)

Single Software Development



Software Product Line Development?



Time and Effort: not scalable!

We need an **engineering process specific** to **software product lines**

Observation: "Reuse-in-the-large works best in families of related systems, and thus is domain dependent." [Glass, 2001]

Domain Engineering

[...] is the activity of collecting, organizing, and storing past experience in building systems [...] in a particular domain in the form of reusable assets [...], as well as providing an adequate means for reusing these assets (i.e., retrieval, qualification, dissemination, adaptation, assembly, and so on) when building new systems.

K. Czarnecki and U. Eisenecker

Domain Engineering

~=

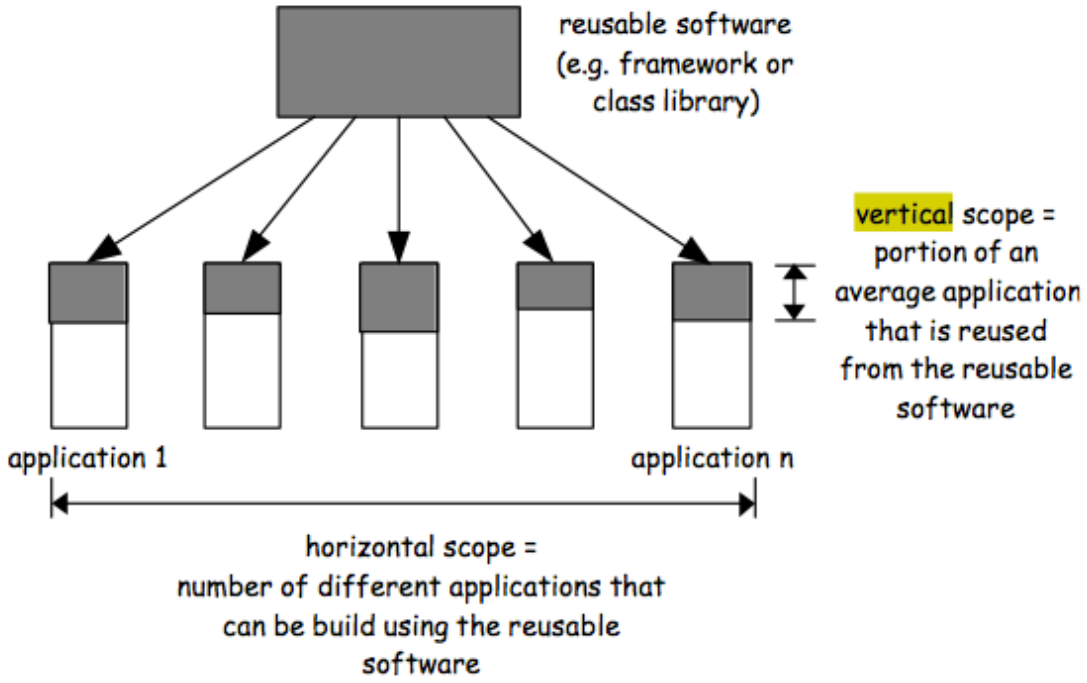
Product Line Engineering

The conventional software engineering concentrates on satisfying the requirements for a **single** system

Domain Engineering concentrates on providing **reusable** solutions for **families** of systems.

Domains of systems vs subsystems

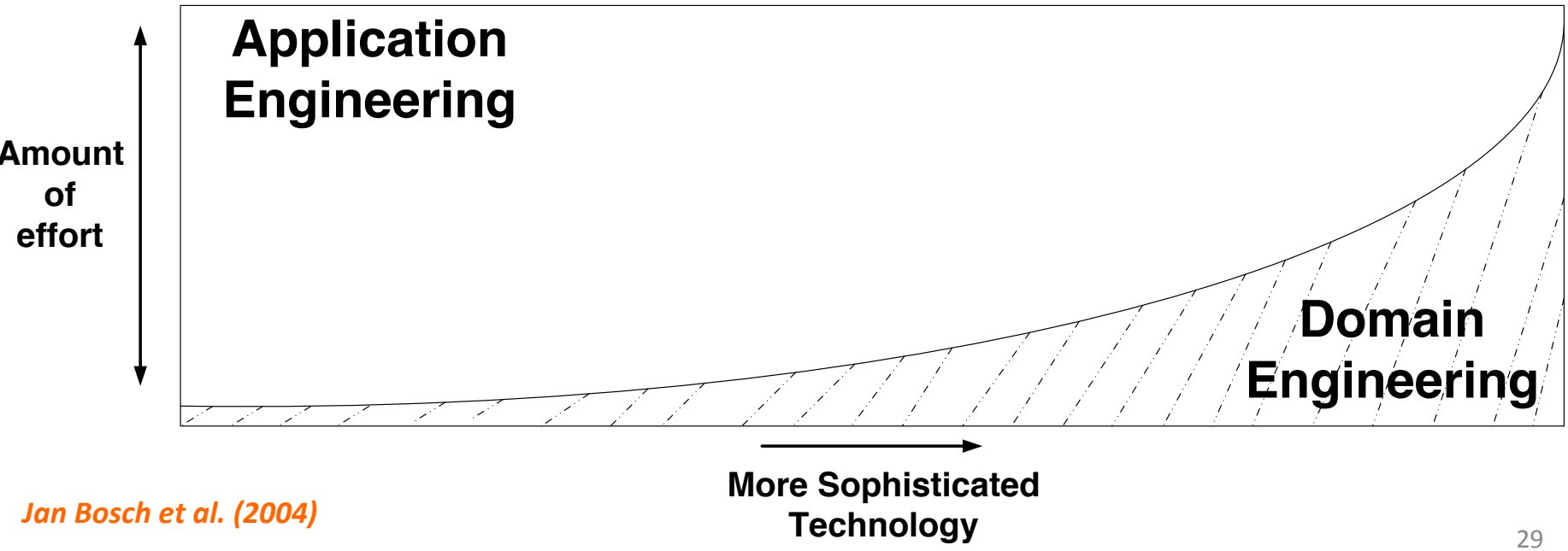
- **vertical domains**
  - e.g. domain of medical record systems, domain of portfolio management systems, etc.
- **horizontal domains**
  - e.g. database systems, numerical code libraries, financial components library, etc.



Czarnecki (PhD thesis)

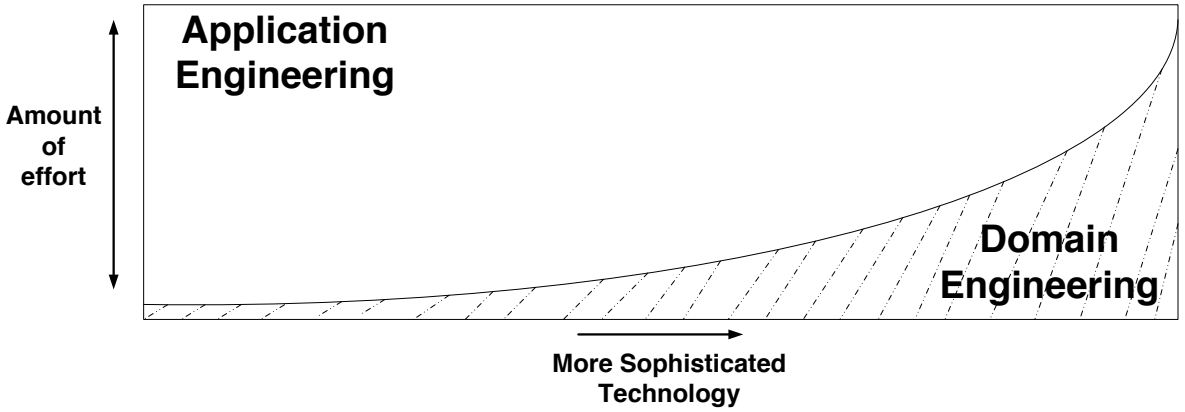
Key idea: building a reusable platform during domain engineering

Key idea: *« the investments required to develop the reusable artifacts during **domain engineering**, are outweighed by the benefits of deriving the individual products during **application engineering** »*



Jan Bosch et al. (2004)

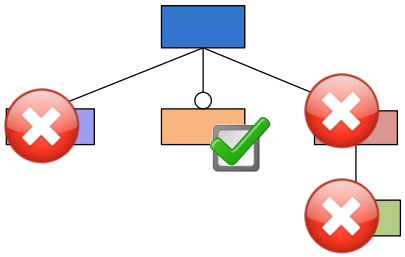
The dream for an SPL practitioner

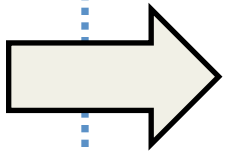


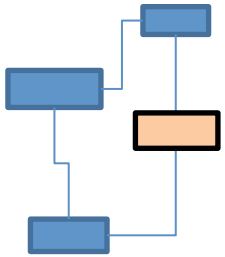
99% domain engineering, 1% application engineering

- specifies what you want (click, click, click) a customized product is automatically built for you
- Iterate the process for n products

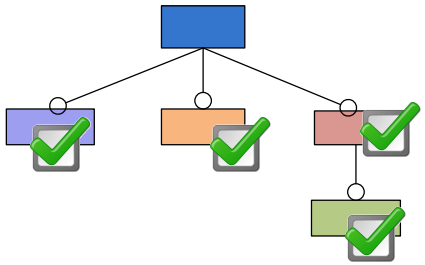
Specific requirements

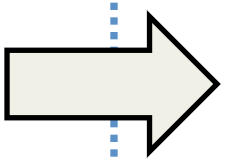


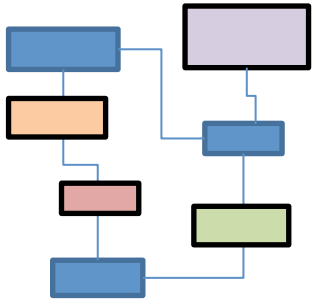




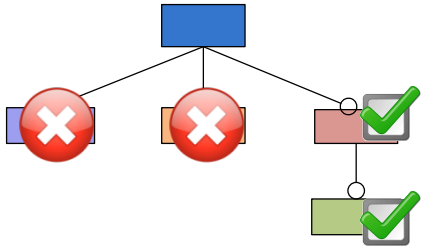
product$_2$

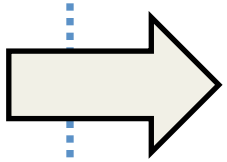


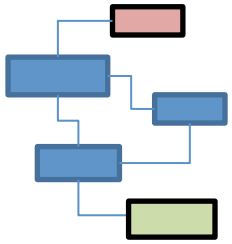




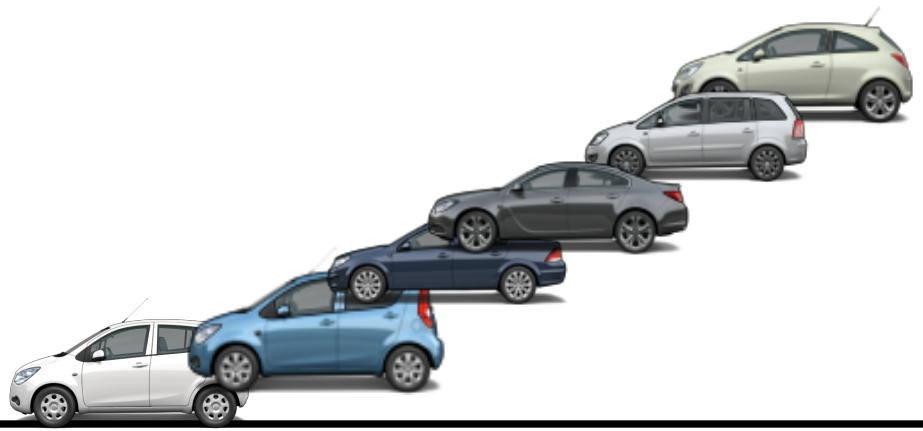
product$_n$







product$_1$



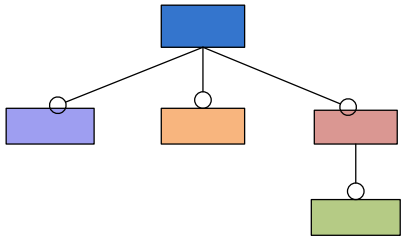
"Reuse-in-the-large works best in families of related systems, and thus is domain dependent." [Glass, 2001]

Domain engineering

Domain Analysis

(problem)

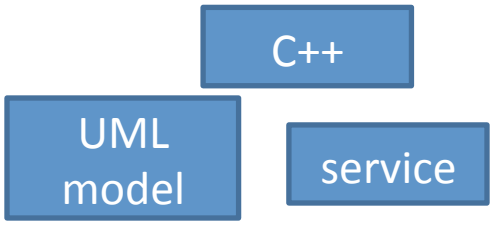
- elicitate requirements and scope the line
- variability modeling: determine commonalities and variabilities usually in terms of features



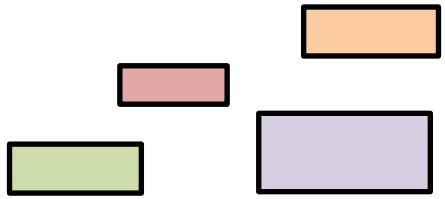
Variability Model (Feature Model)

Domain Implementation

(solution)



Common assets

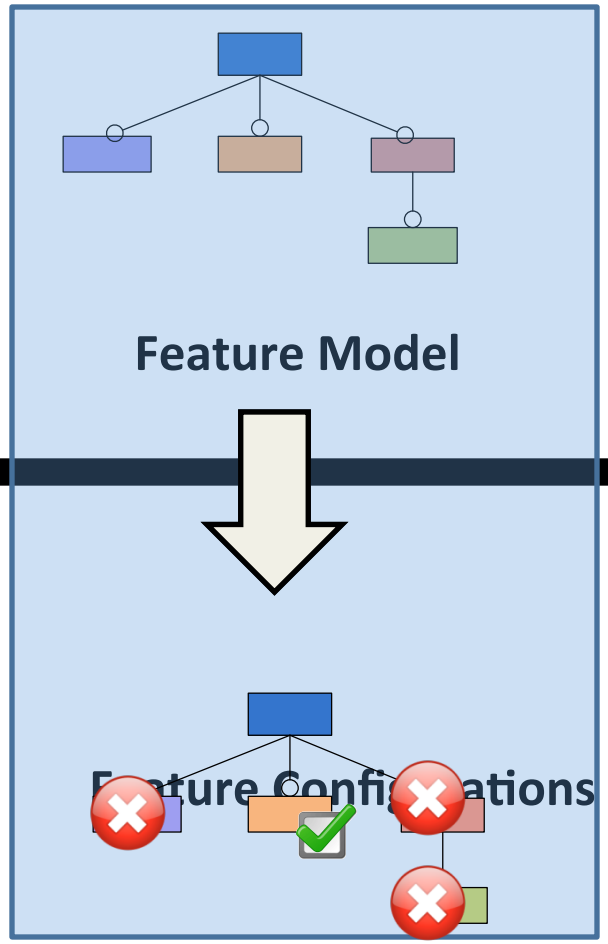


Variants

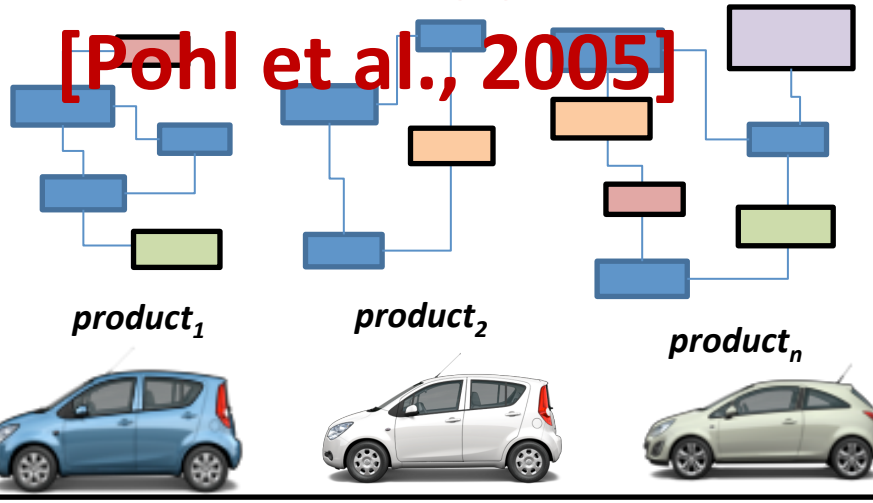
Reusable Assets

(e.g., models or source code)

Domain engineering (development for reuse)



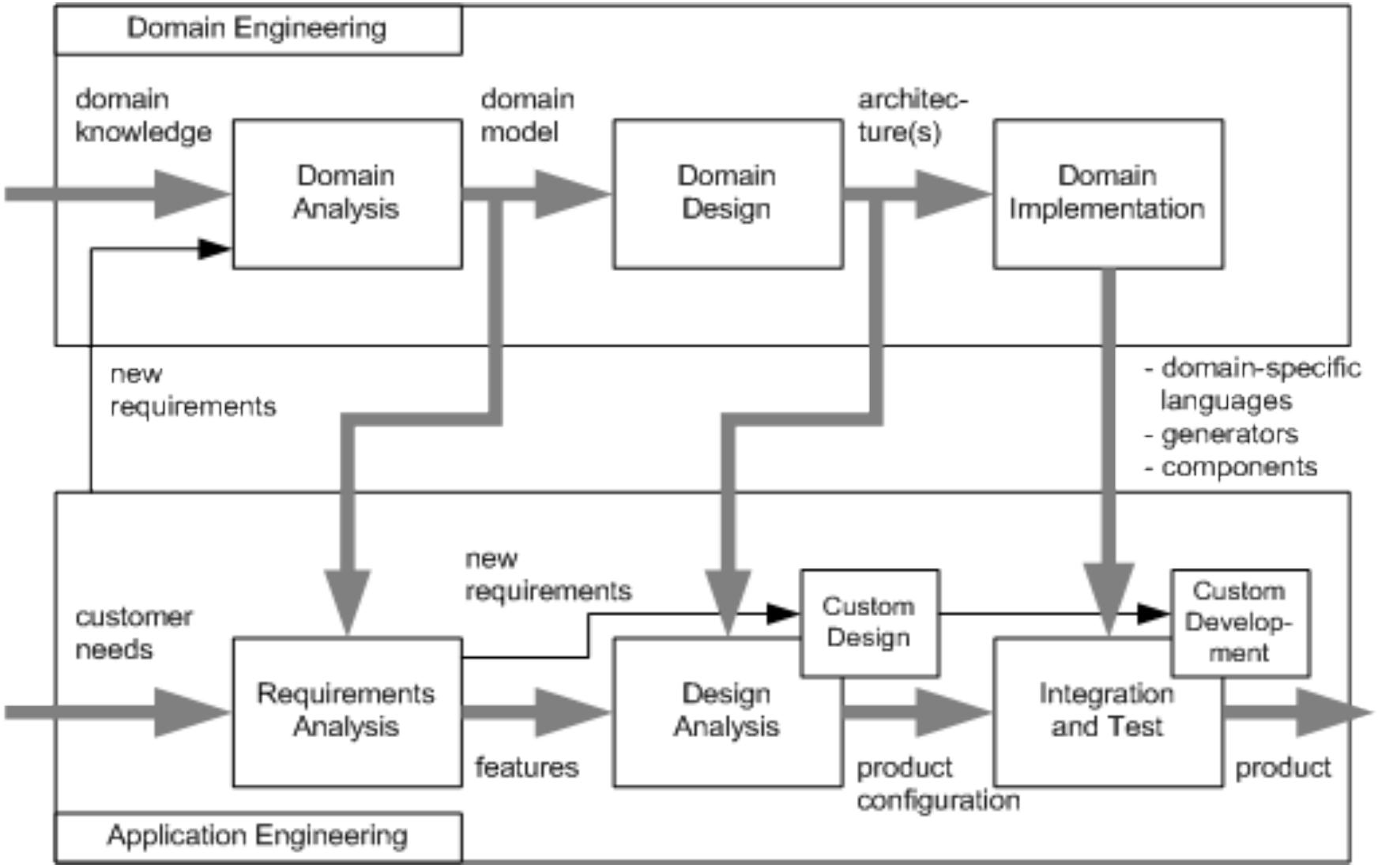
Reusable Assets
(e.g., models or source code)



"central to the software product line paradigm is the modeling and management of variability, that is, the commonalities and differences in the applications" [Pohl et al., 2005]

Application engineering (development with reuse)

Activities related to domain engineering and application engineering



Klaus Pohl (2005)

Domain Analysis

- Collect relevant domain information
  - domain experts (interviews, workshops)
  - system handbooks, textbooks, prototyping, experiments,
  - already known requirements on future systems
  - Creative activity
- Domain Definition
  - examples of systems in a domain,
  - counterexamples (i.e. systems outside the domain),
  - generic rules of inclusion or exclusion (e.g. “Any system having the capability X belongs to the domain.”).
- Domain vocabulary
- Domain concepts
- and integrate it into a coherent *domain model*
  - more or less formal

Czarnecki and Eisenecker (2000)

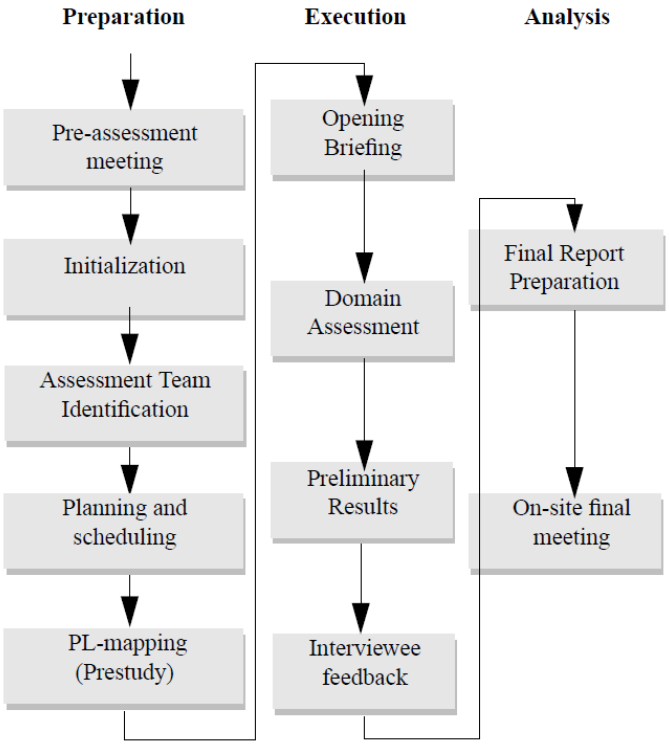
Domain Model

- Ontology, ER, UML, Feature Model

- Analysis of similarity
  - Analyze similarities between entities, activities, events, relationships, structures, etc.
- Analysis of variations
  - Analyze variations between entities, activities, events, relationships, structures, etc.

- Clustering
- Abstraction
- Classification
- Generalization
- Vocabulary construction

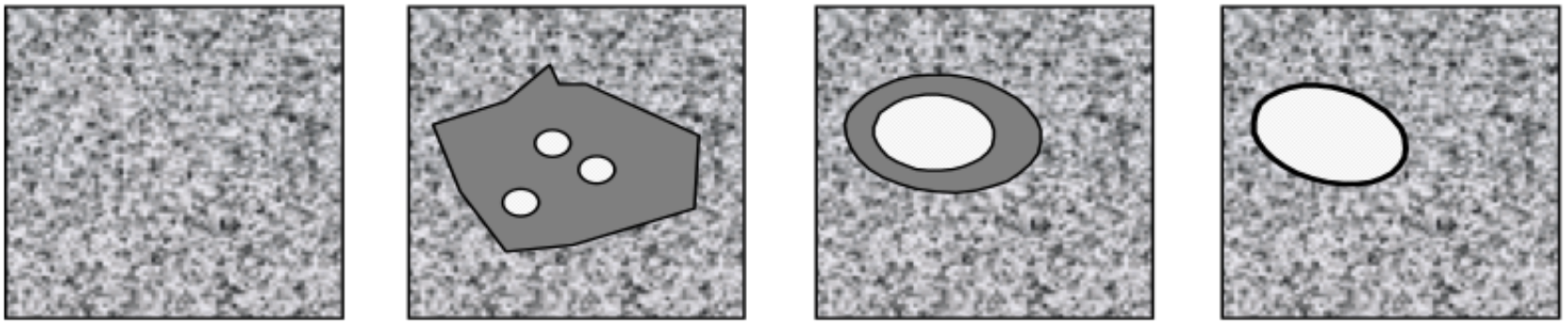
Domain/Product Line **Scoping**



Schmid 2002

			exist.	planned		potent.
			P1	P2	P3	P4
Domain 1	Sub-Domain 1.1	Feature 1.1.1	X	X	X	X
		Feature 1.1.2	—	X	X	X
		Feature 1.1.3	X	X	—	X
	...	...	...	...	...	...
	Sub-Domain 1.n	Feature 1.n.1	X	—	X	X
		...	...	...	...	...
Domain 2	Sub-Domain 2.1	Feature 2.1.1	—	X	X	—
	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	Feature m.1.1	—	X	—	X

Scope

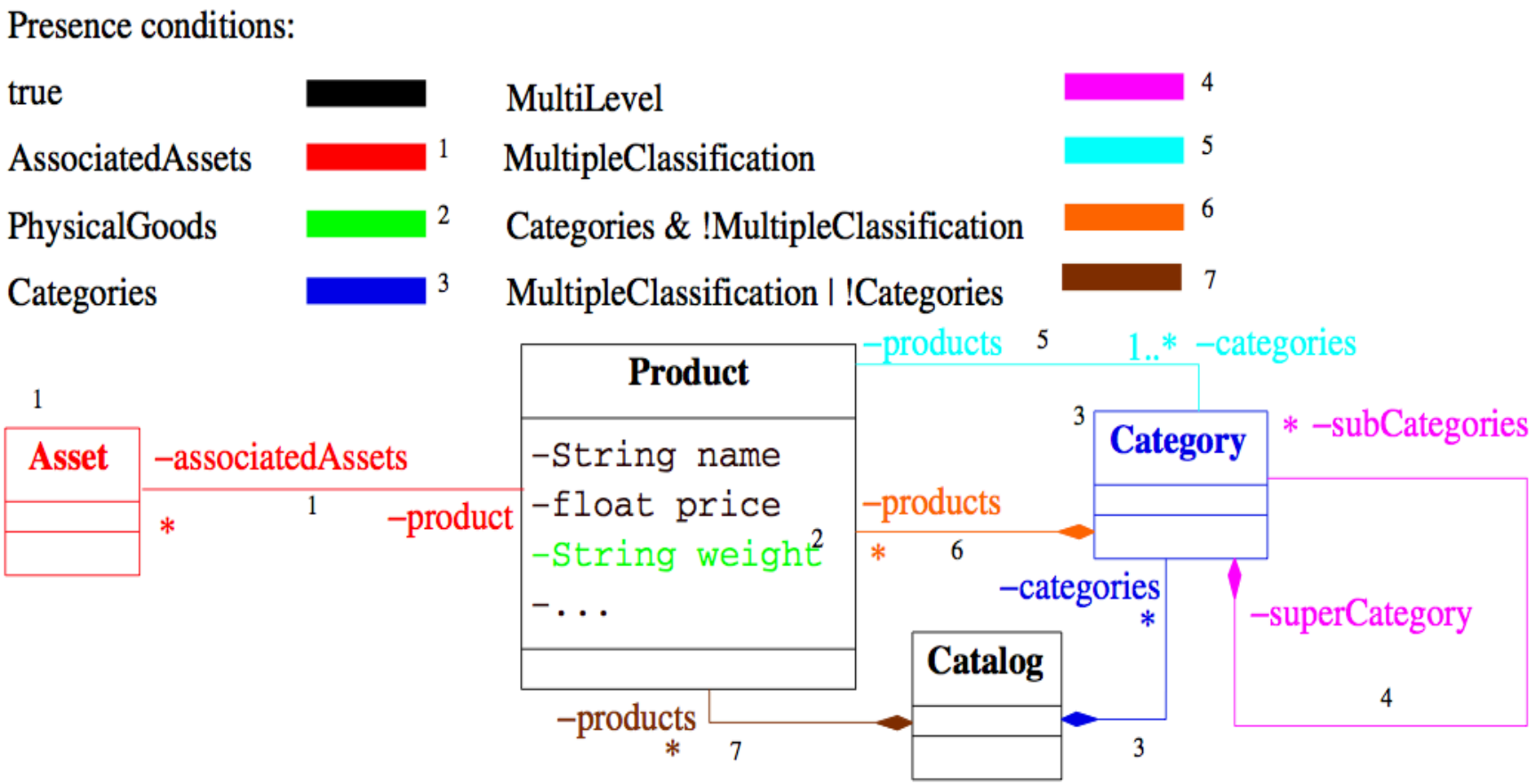


Clement et al. (2001)


Unused flexibility

Illegal Variant

Domain Design



Czarnecki et al. (GPCE'05)

Domain Implementation

- Reusable assets
  - e.g. reusable services
- Domain-specific languages
- Generators
- Reuse infrastructure

- Many implementation techniques

Domain-specific language (DSL)

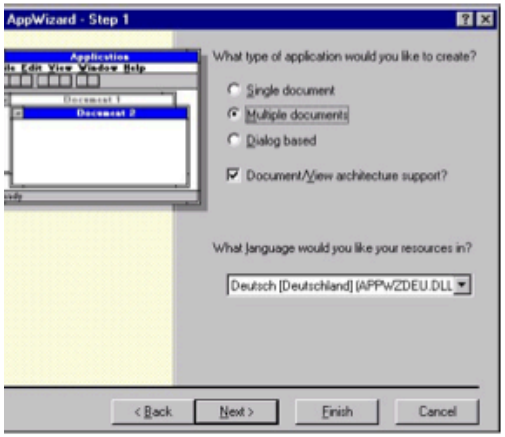
- Domain-specific **abstractions**
  - a DSL provides pre-defined abstractions to directly represent concepts from the domain.
- Domain-specific **concrete syntax**
  - a DSL offers a natural notation for a given domain and avoids syntactic clutter that often results when using a general- purpose language.
- Domain-**specific** error checking
- Domain-**specific** optimizations
- Domain-**specific** tool support

DSLs

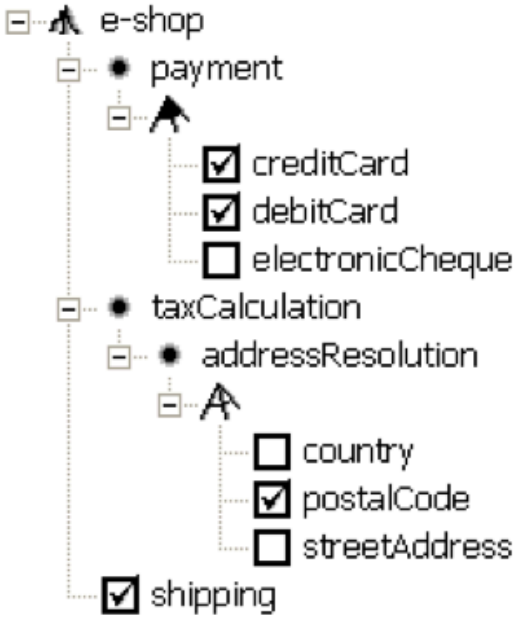
- textual languages (stand-alone or embedded in a general-purpose programming language),
- visual, diagrammatic languages
- form-based languages

Czarnecki et al. (2005)

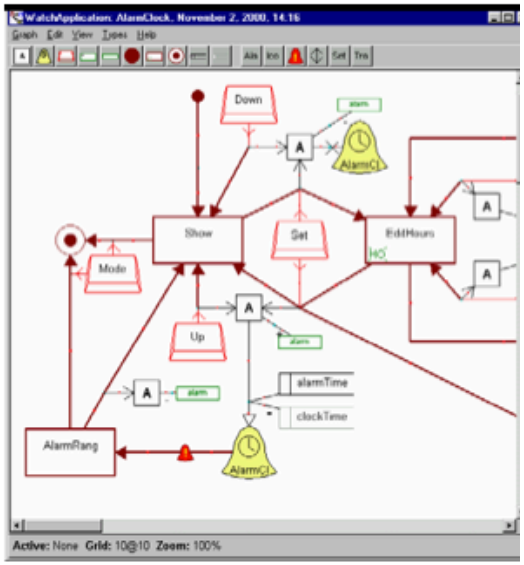
Routine configuration ⟷ *Creative construction*



Wizard



Feature-based configuration



Graph-like language

Dummy Feature Model

```
feature runtimeCalibration : false
feature bumper : true
feature sonar : false
feature debugOutput : true
```

Voelter (SPLC'11)

Configuring Models and Code

```
{sonar} task sonartask cyclic prio = 2 every = 100 {
          int s = ecrobot_get_sonar_sensor(SENSOR_PORT_T::NXT_PORT_S2);
          sonarHistory[sonarIndex] = s;
          sonarIndex = sonarIndex + 1;
          if ( sonarIndex == 10 ) {
            sonarIndex = 0;
          }
          int ss = 0;
          for ( int i = 0; i < 10; i = i + 1; ) {
            ss = ss + sonarHistory[i];
          }
          currentSonar = ss / 10;
          { {debugOutput} { debugInt(2, "sonar:", currentSonar); } }
        }
```

```
doc This is the cyclic task that is called every 1ms to do the actual control of th
task run cyclic prio = 2 every = 2 {
  stateswitch linefollower
    state running
      {bumper} int bump = ecrobot_get_touch_sensor(SENSOR_PORT_T::NXT_PORT_S3);
      {bumper} if ( bump == 1 ) {
        {debugOutput} { debugString(3, "bump:", "BUMP!"); }
        event linefollower:bumped
        terminate;
      }
      {sonar} if ( currentSonar < 150 ) {
        event linefollower:blocked
        terminate;
      }
      int light = ecrobot_get_light_sensor(SENSOR_PORT_T::NXT_PORT_S1);
      if ( light < ( WHITE + BLACK ) / 2 ) {
        updateMotorSettings(SLOW, FAST);
      } else {
        updateMotorSettings(FAST, SLOW);
      }
      {debugOutput} { debugInt(4, "light:", light); }
    {sonar} state paused
              updateMotorSettings(0, 0);
              if ( currentSonar < 255 ) {
                event linefollower:unblocked
              }
    {bumper} state crash
               updateMotorSettings(0, 0);
    default
      <noop>;
}
```

Preprocessor for Java code (Munge)

```
class Example {
        void main() {
                System.out.println("immer");
                /*if[DEBUG]*/
                System.out.println("debug info");
                /*end[DEBUG]*/
        }
}
```

java Munge –DDEBUG –DFEATURE2 Example.java

configuration option

Kastner's slide

http://weblogs.java.net/blog/tball/archive/2006/09/munge_swings_se.html

```
class Graph {
  Vector nv = new Vector(); Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++) {
      ((Edge)ev.get(i)).print();
    }
  }
}
```

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
    Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;= new Weight();
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    Color.setDisplayColor(color);
    a.print(); b.print();
    weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

Kastner's slide

```
class Graph {
  Vector nv = new Vector(); Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    /*if[WEIGHT]*/
    e.weight = new Weight();
    /*end[WEIGHT]*/
    return e;
  }
  /*if[WEIGHT]*/
  Edge add(Node n, Node m, Weight w)
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  /*end[WEIGHT]*/
  void print() {
    for(int i = 0; i < ev.size(); i++) {
      ((Edge)ev.get(i)).print();
    }
  }
}
```

```
/*if[WEIGHT]*/
class Weight { void print() { ... } }
/*end[WEIGHT]*/
```

```
class Edge {
  Node a, b;
  /*if[COLOR]*/
  Color color = new Color();
  /*end[COLOR]*/
  /*if[WEIGHT]*/
  Weight weight;
  /*end[WEIGHT]*/
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    /*if[COLOR]*/
    Color.setDisplayColor(color);
    /*end[COLOR]*/
    a.print(); b.print();
    /*if[WEIGHT]*/
    weight.print();
    /*end[WEIGHT]*/
  }
}
```

```
/*if[COLOR]*/
class Color {
  static void setDisplayColor(Color c) { ... }
}
/*end[COLOR]*/
```

```
class Node {
  int id = 0;
  /*if[COLOR]*/
```

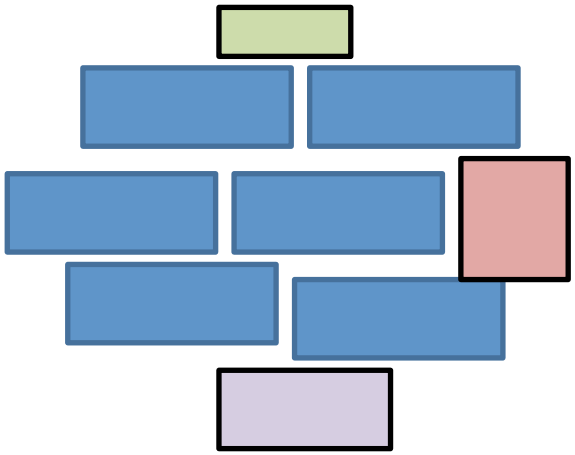
Adoption and Strategies

- **Proactive**
- **Reactive**
- **Extractive**

Software Product Line Engineering

Factoring out **commonalities**

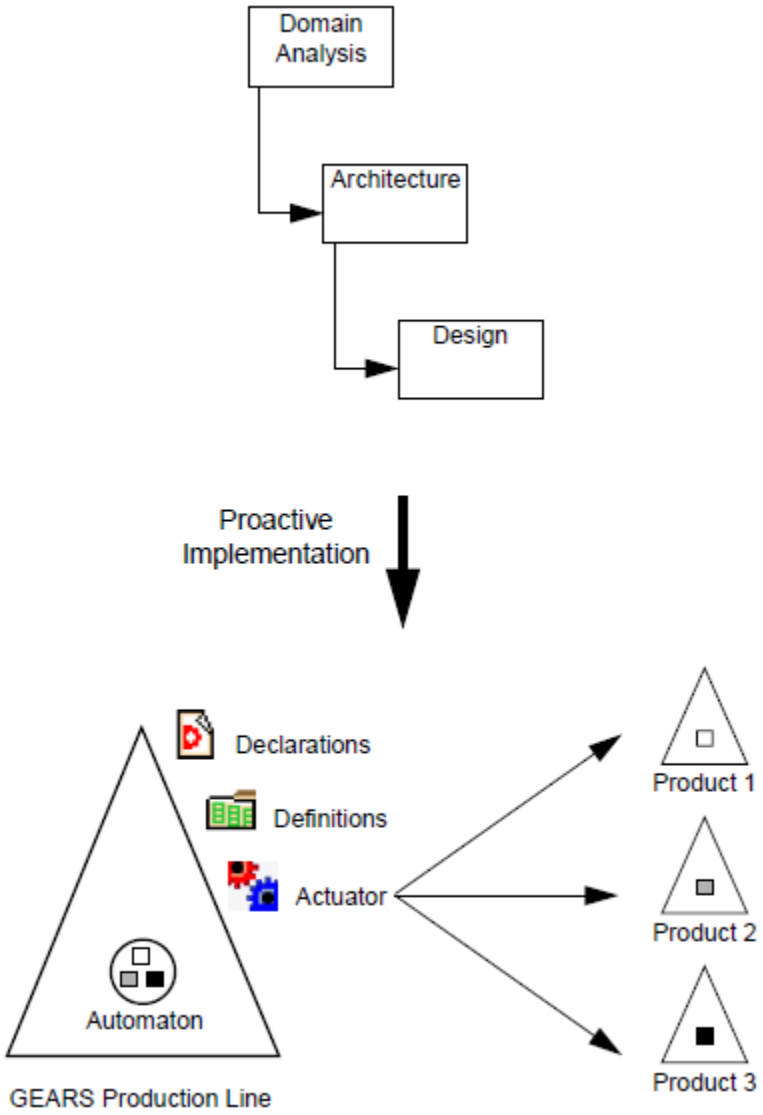
for **Reuse** [Krueger et al., 1992] [Jacobson et al., 1997]



Managing **variabilities**

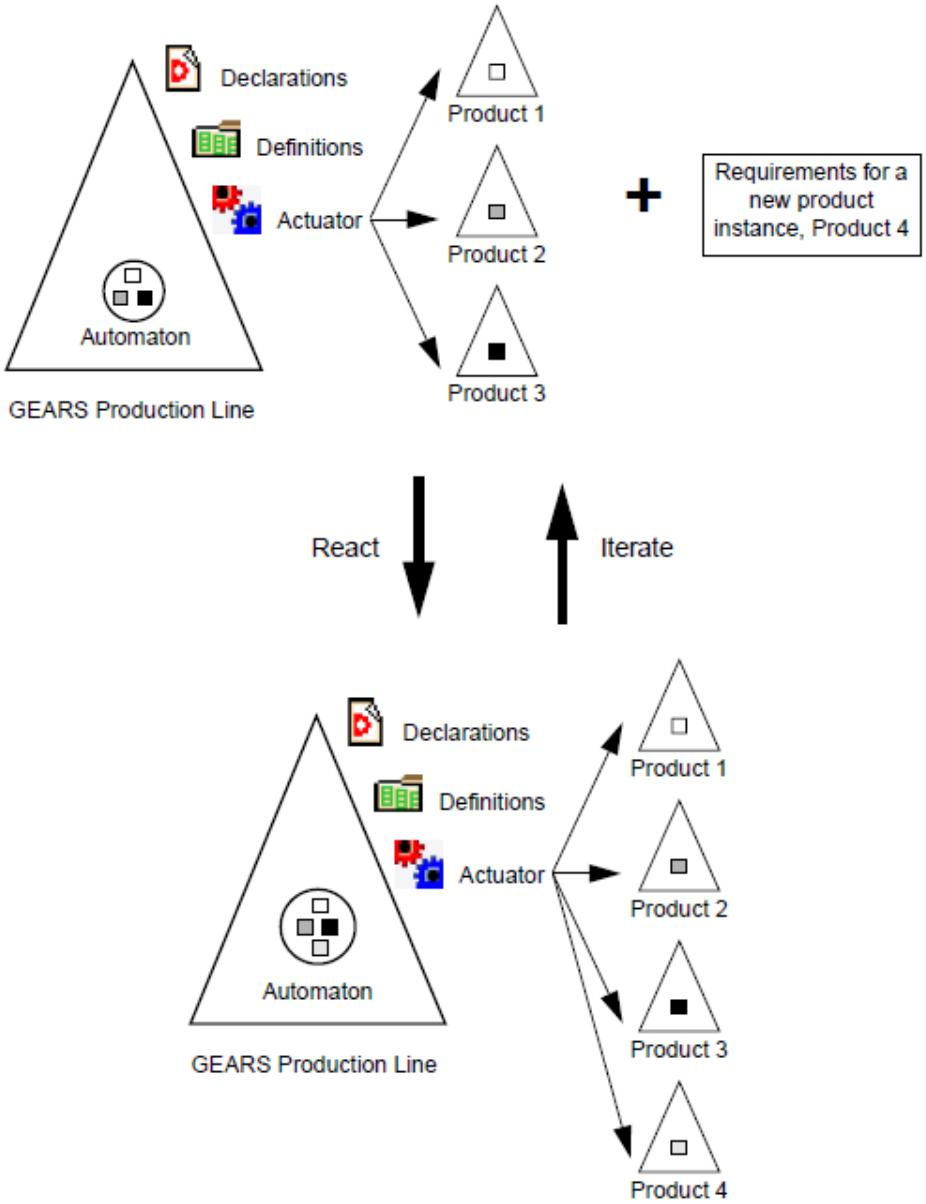
for Software **Mass Customization** [Bass et al., 1998] [Krueger et al., 2001], [Pohl et al., 2005]

Proactive



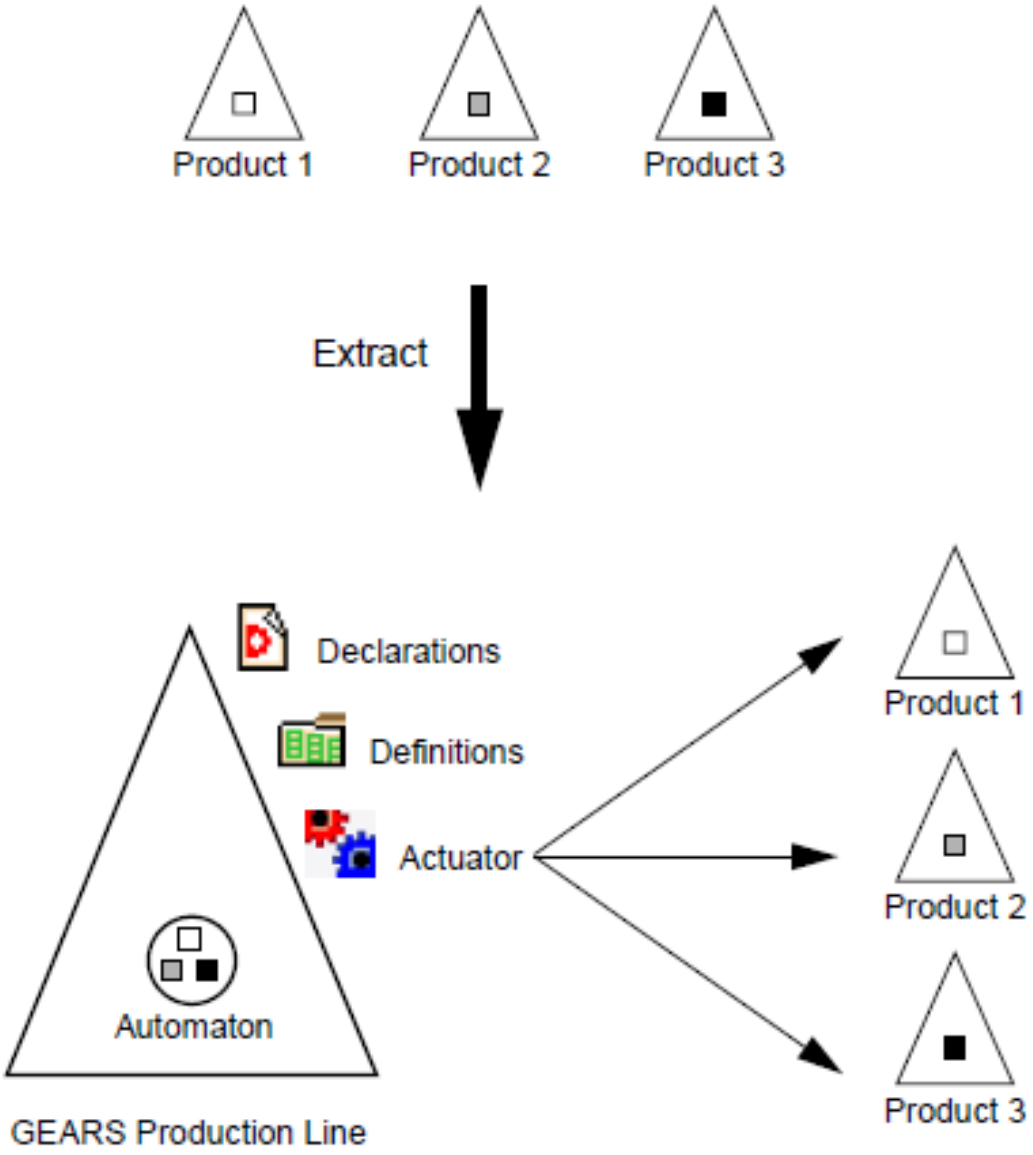
[Krueger 2002]

Reactive



[Krueger 2002]

Extractive



[Krueger 2002]

Software Product Line Engineering

Generative perspective

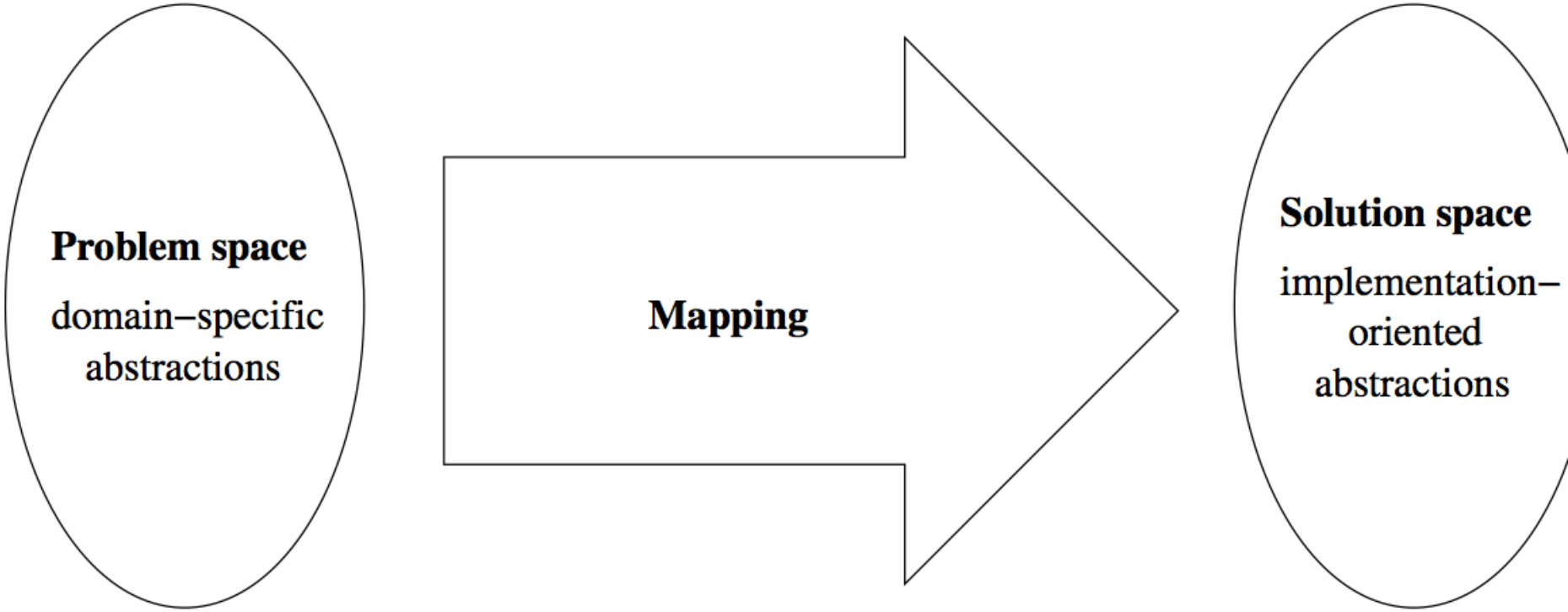
Generative software development aims at modeling and implementing system families such that a desired system can be automatically generated from a specification written in one or more textual or graphical domain-specific languages.

[GSD lab, webiste, Waterloo]

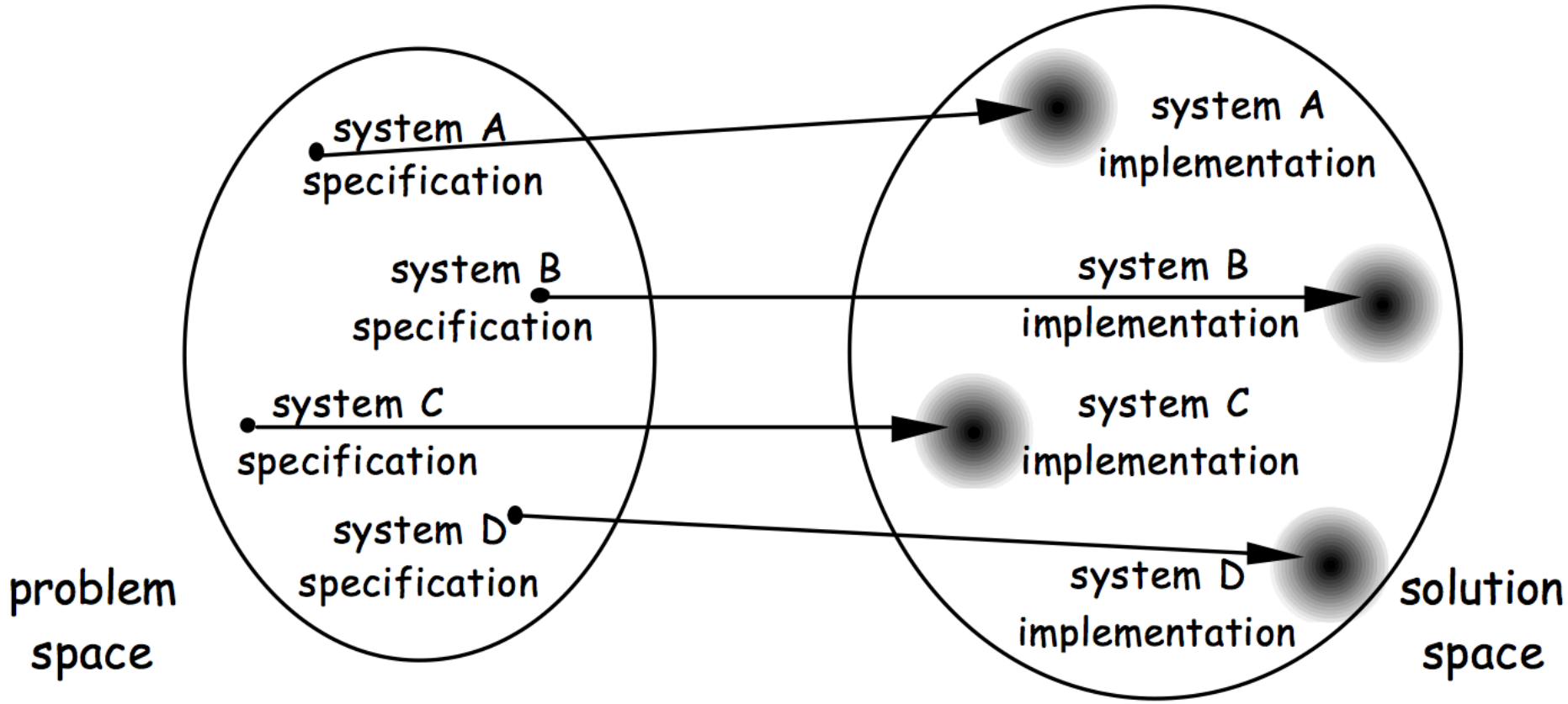
Generative approach

- Generative programming
  - Programming the generation of programs
    - Very old practice
    - Metaprogramming: generative language and target language are the same
      - Reflection capabilities

- Generalization of this idea:
  - from a specification written in one or more textual or graphical domain-specific languages
  - you generate customized variants

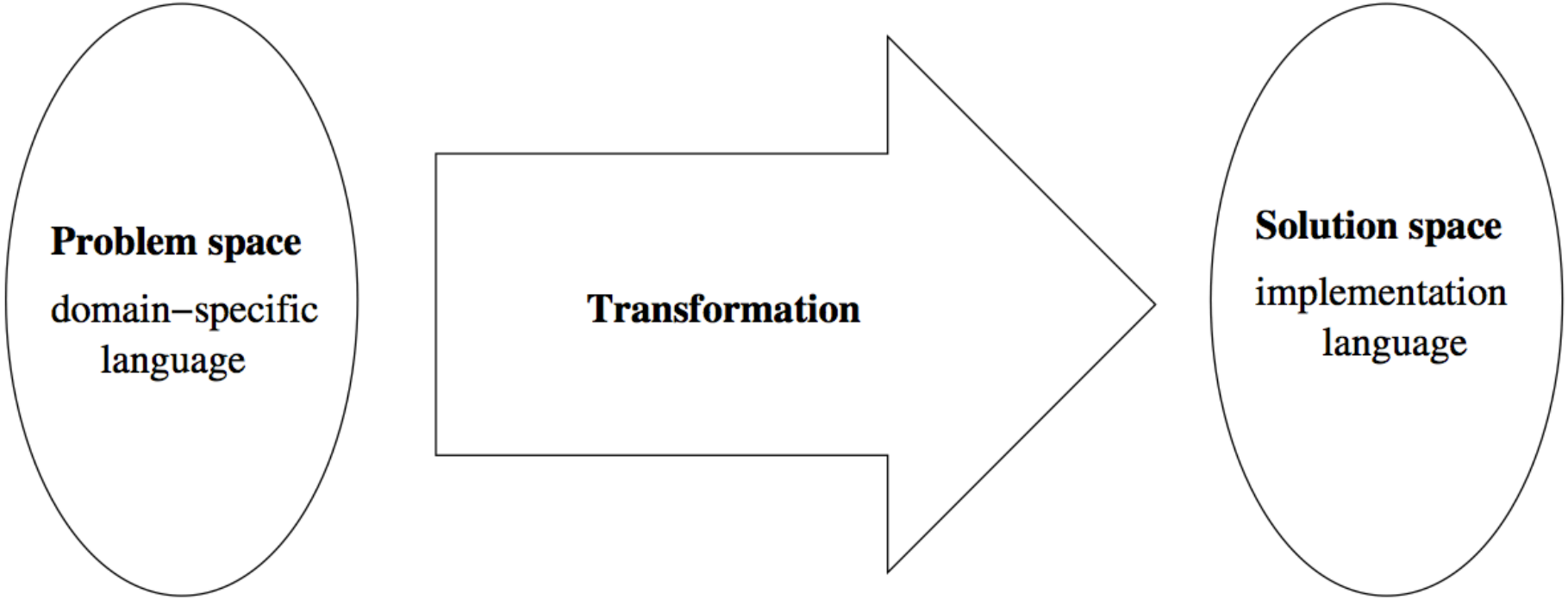


[Czarnecki and Eisenecker 2000]

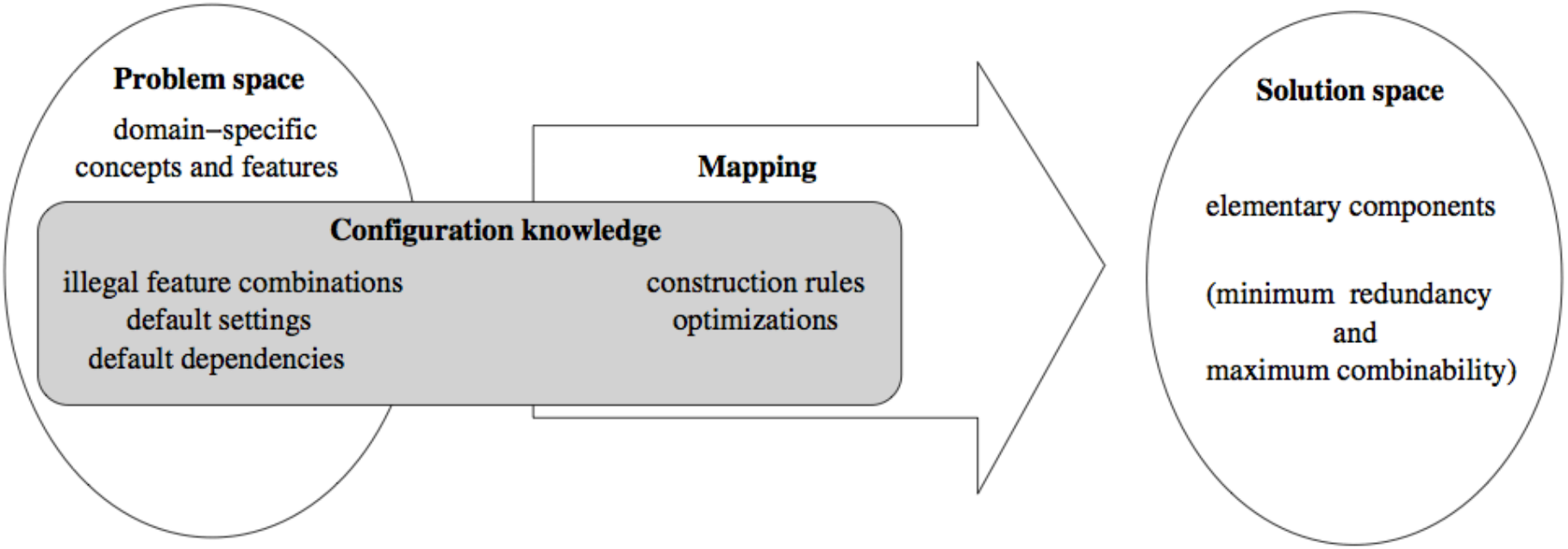


[Czarnecki, PhD thesis]

S



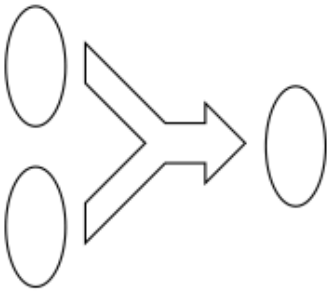
[Czarnecki and Eisenecker 2000]



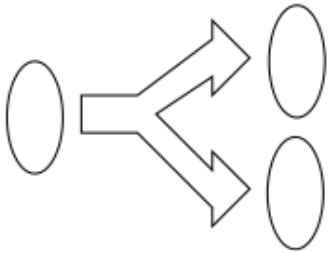
[Czarnecki and Eisenecker 2000]

[Czarnecki et al. 2005]

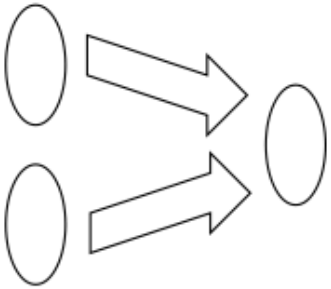
a. Chaining of mappings



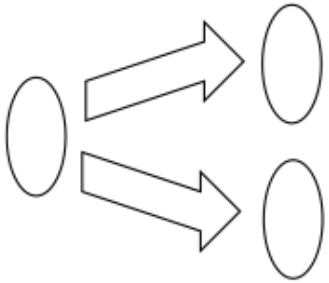
b. Multiple problem spaces



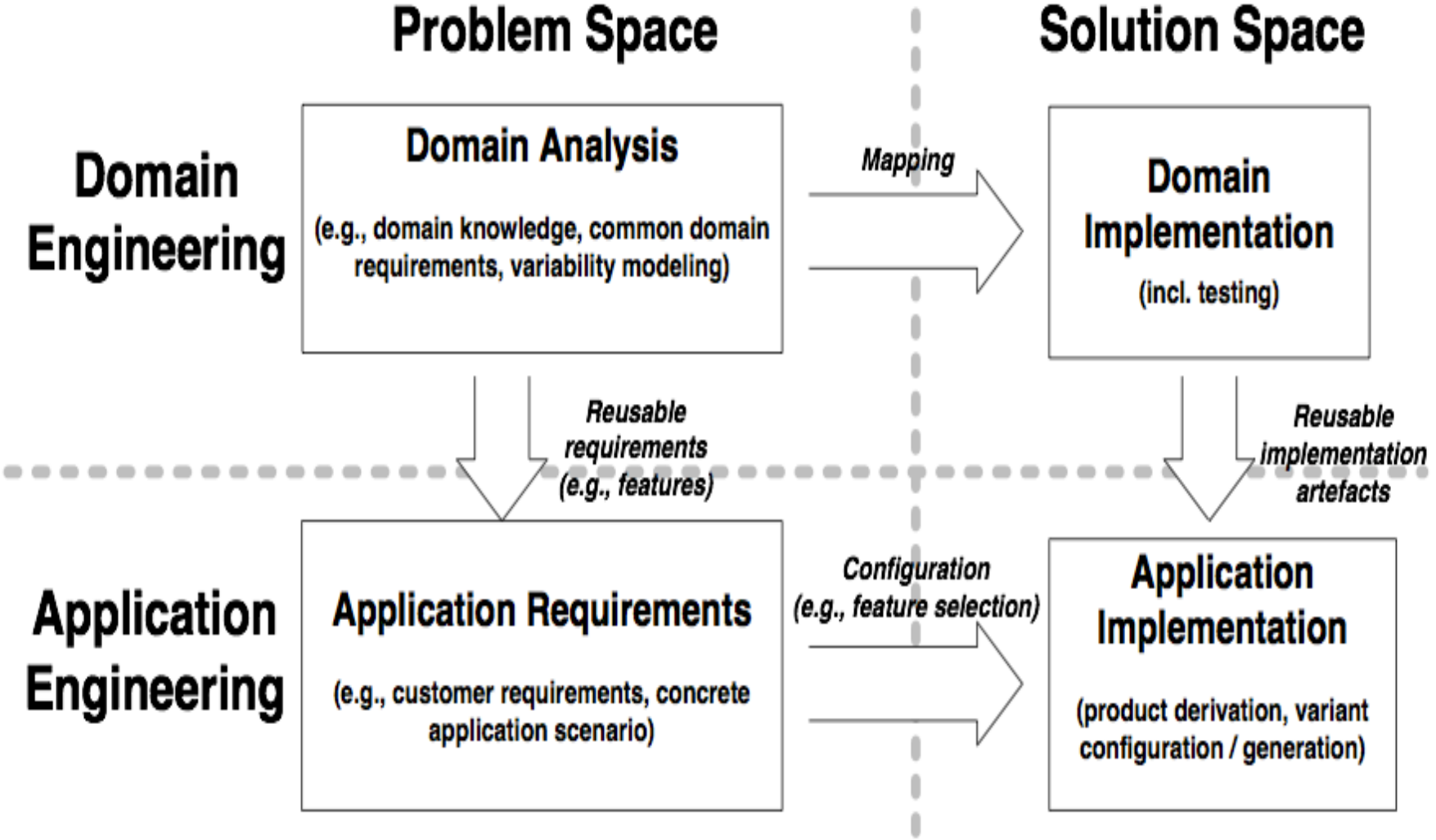
c. Multiple solution spaces



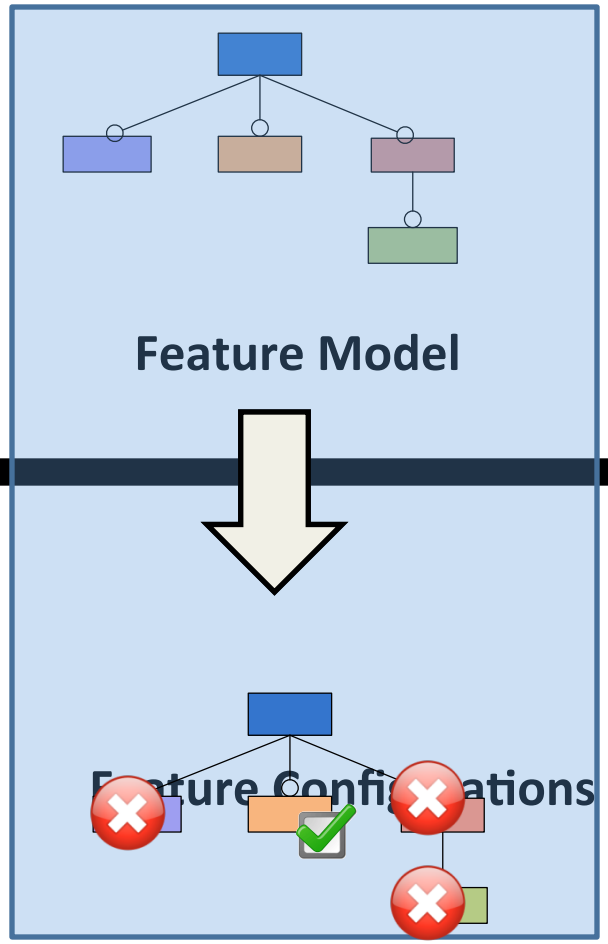
d. Alternative problem spaces



e. Alternative solution spaces

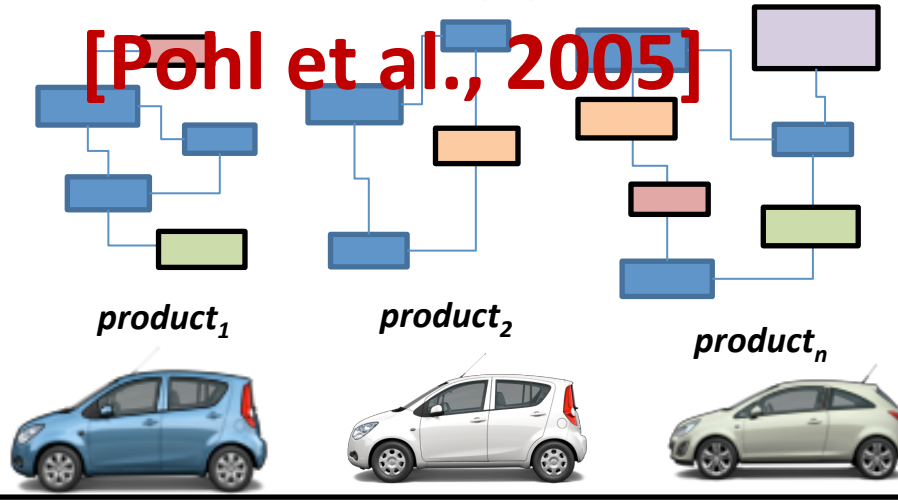

Problem Space
Solution Space
Domain Engineering
Domain Analysis
(e.g., domain knowledge, common domain requirements, variability modeling)
Mapping
Domain Implementation
(incl. testing)
Reusable requirements (e.g., features)
Reusable implementation artefacts
Application Engineering
Application Requirements
(e.g., customer requirements, concrete application scenario)
Configuration (e.g., feature selection)
Application Implementation
(product derivation, variant configuration / generation)

Domain engineering (development for reuse)



Reusable Assets
(e.g., models or source code)

"central to the software product line paradigm is the modeling and management of variability, that is, the commonalities and differences in the applications" [Pohl et al., 2005]



Application engineering (development with reuse)

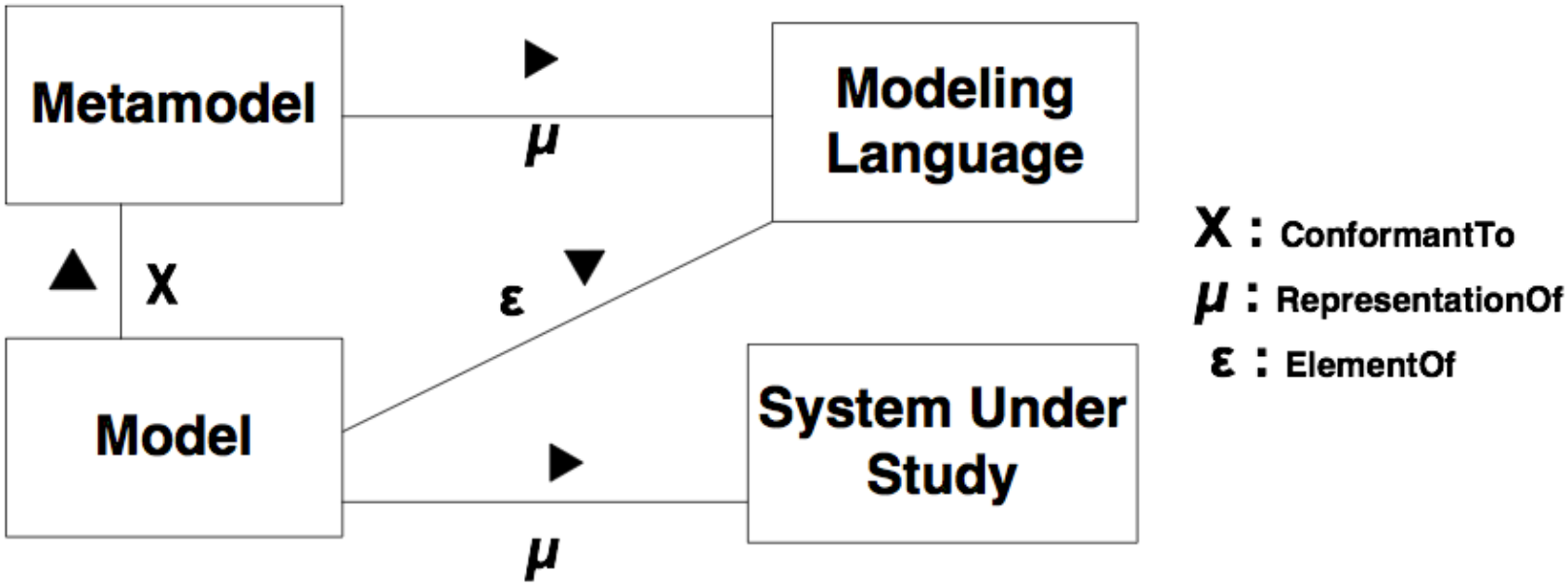
Software Product Line Engineering

Model-based perspective

Possible Roles of

Model-Driven Engineering

Model


Metamodel
Modeling Language
Model
System Under Study
μ
χ
ε
μ
X : ConformantTo
μ : RepresentationOf
ε : ElementOf

Model in a nutshell

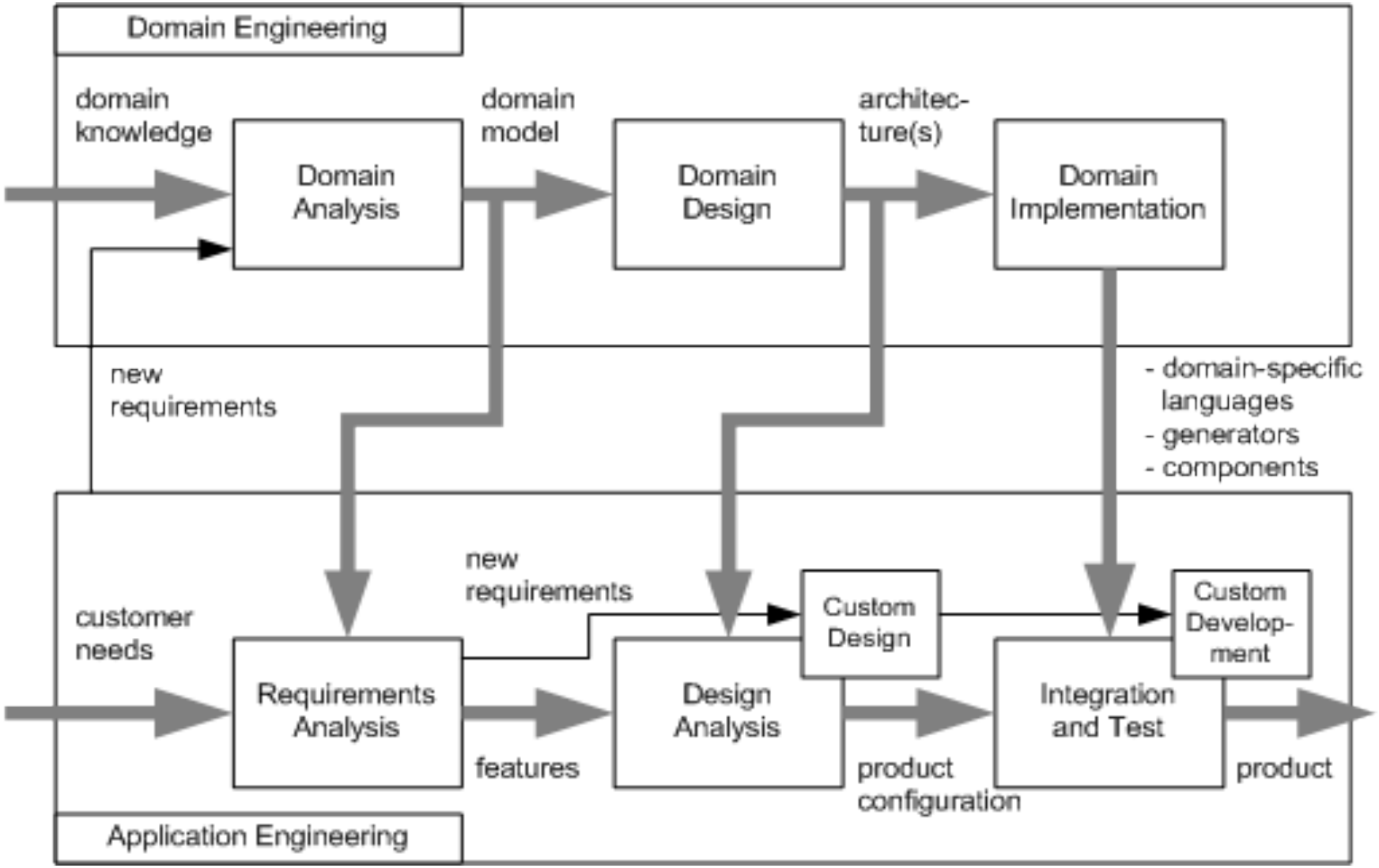
- In essence, a model is an **abstraction** of some aspect of a system under study.
- Some details are hidden or removed to **simplify** and focus attention.
- A model is an abstraction since **general** concepts can be formulated by abstracting common properties of instances or by extracting common features from specific examples.

Models in a nutshell

- Models are created to serve particular *purposes*
  - not necessarily for the purpose of comprehensively represent a whole system
  - some “aspects”
  - multiple models are generally used
- In a form that can be *mechanically* analyzed.
- Models can processed by computer-based tools in order to derive other useful artefacts (models): ***model transformation***

Promises of Model-driven Engineering

- Reducing the gap between the problem space and the solution space
- Raising the level of abstraction
- Avoiding accidental complexity
- **Abstraction & Transformations**

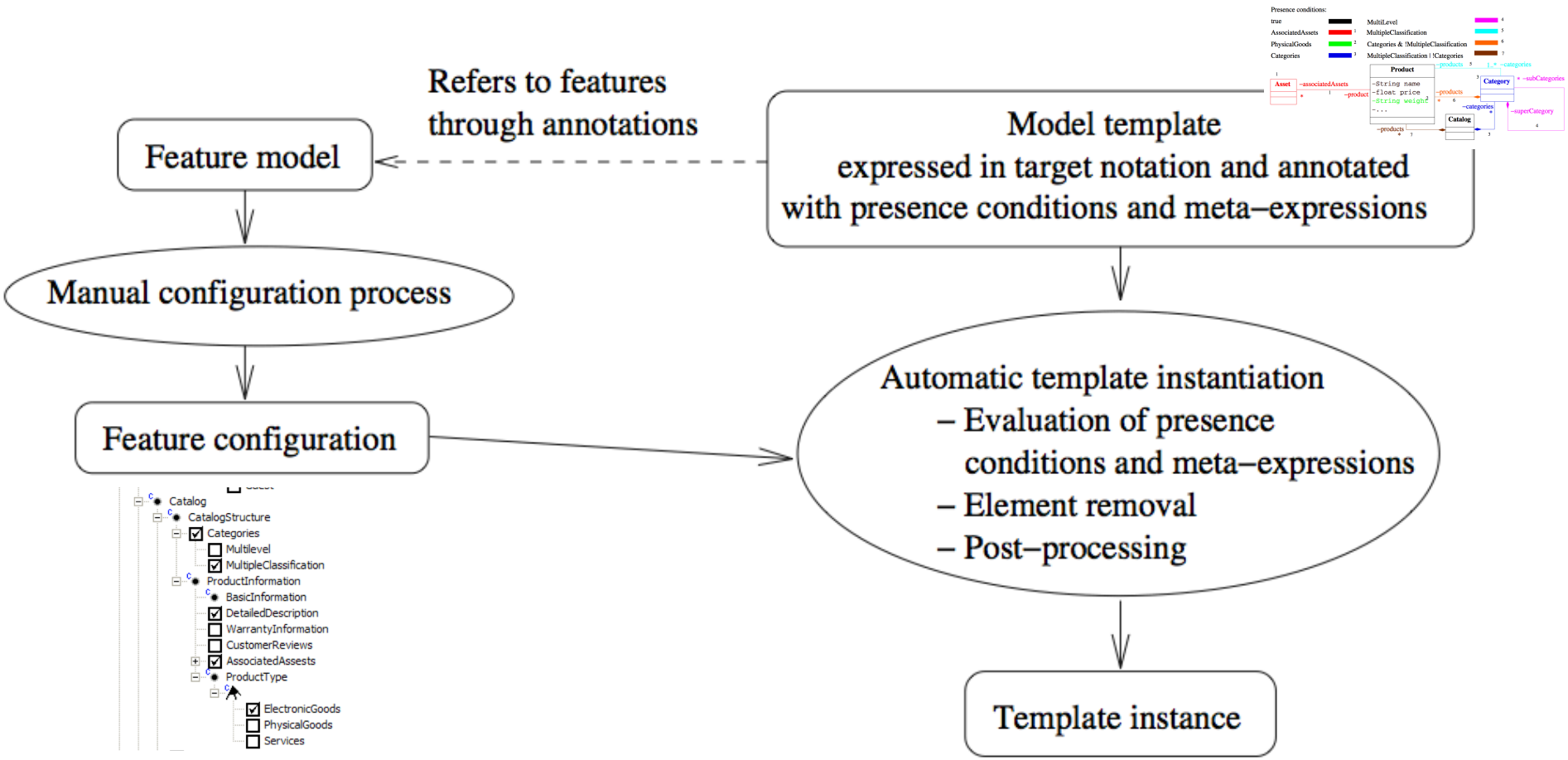

Domain Engineering
domain knowledge
Domain Analysis
domain model
Domain Design
architec-ture(s)
Domain Implementation
new requirements
- domain-specific languages
- generators
- components
customer needs
Requirements Analysis
new requirements
Custom Design
Design Analysis
Custom Develop-ment
Integration and Test
features
product configuration
product
Application Engineering

Model-driven Engineering in the SPL framework

- Domain Engineering
  - Domain Models
  - Level of abstraction
  - Domain-specific modeling languages
    - (visual or textual) syntacs, precise semantics
    - analyzed (verification)
  - Traceability between the artefacts
- Application Engineering
  - Model transformations (automation)
- Reduce the gap

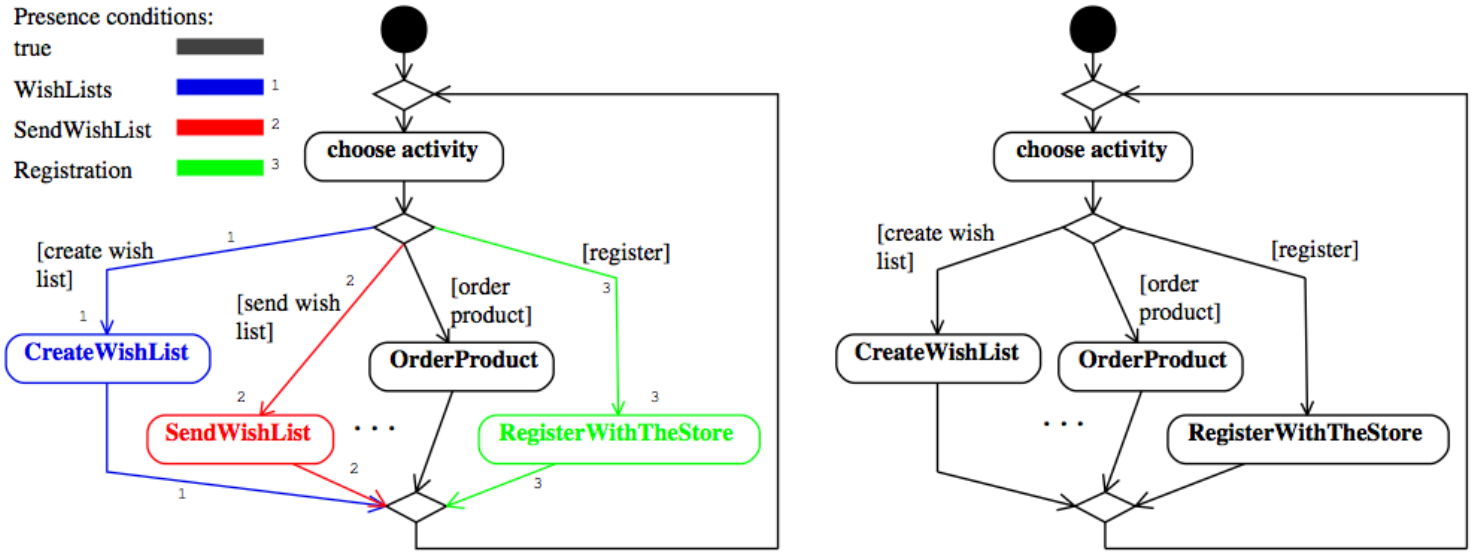
[Czarnecki et al. (GPCE'05)]

Realizing variability: derivation of models



[Czarnecki et al. (GPCE'05)]

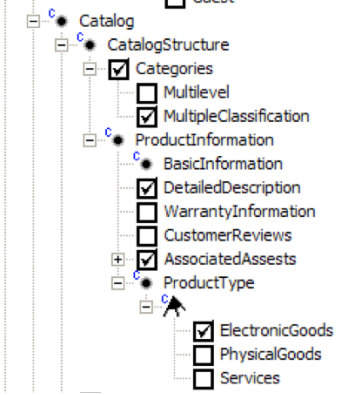
Example



(a) Storefront template

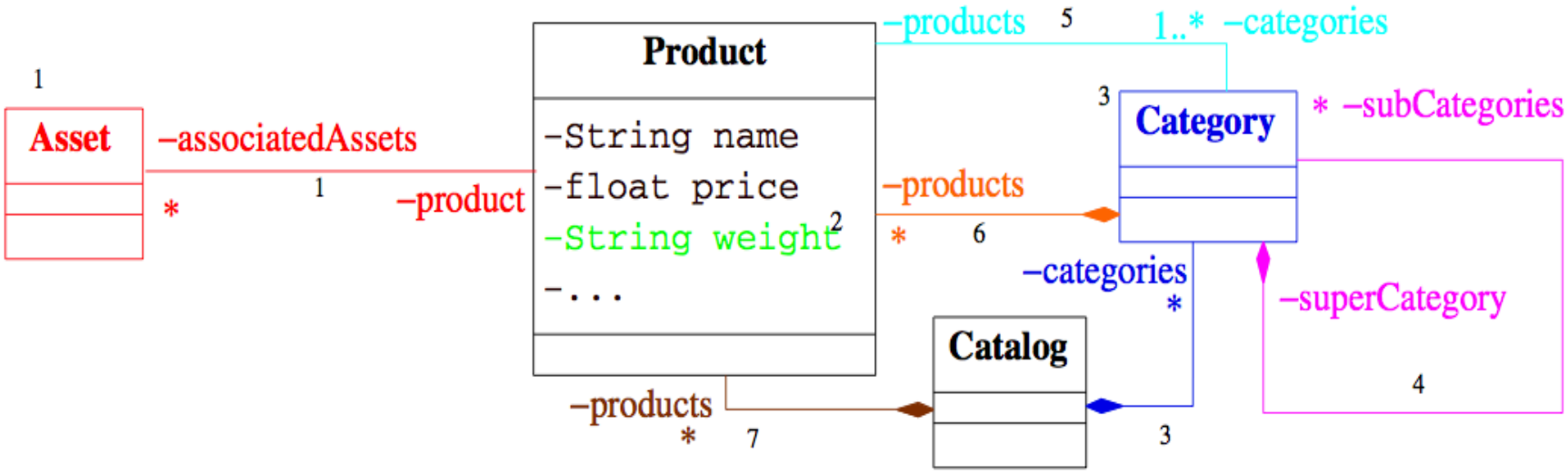
(b) Storefront instance

[Czarnecki et al. (GPCE'05)]



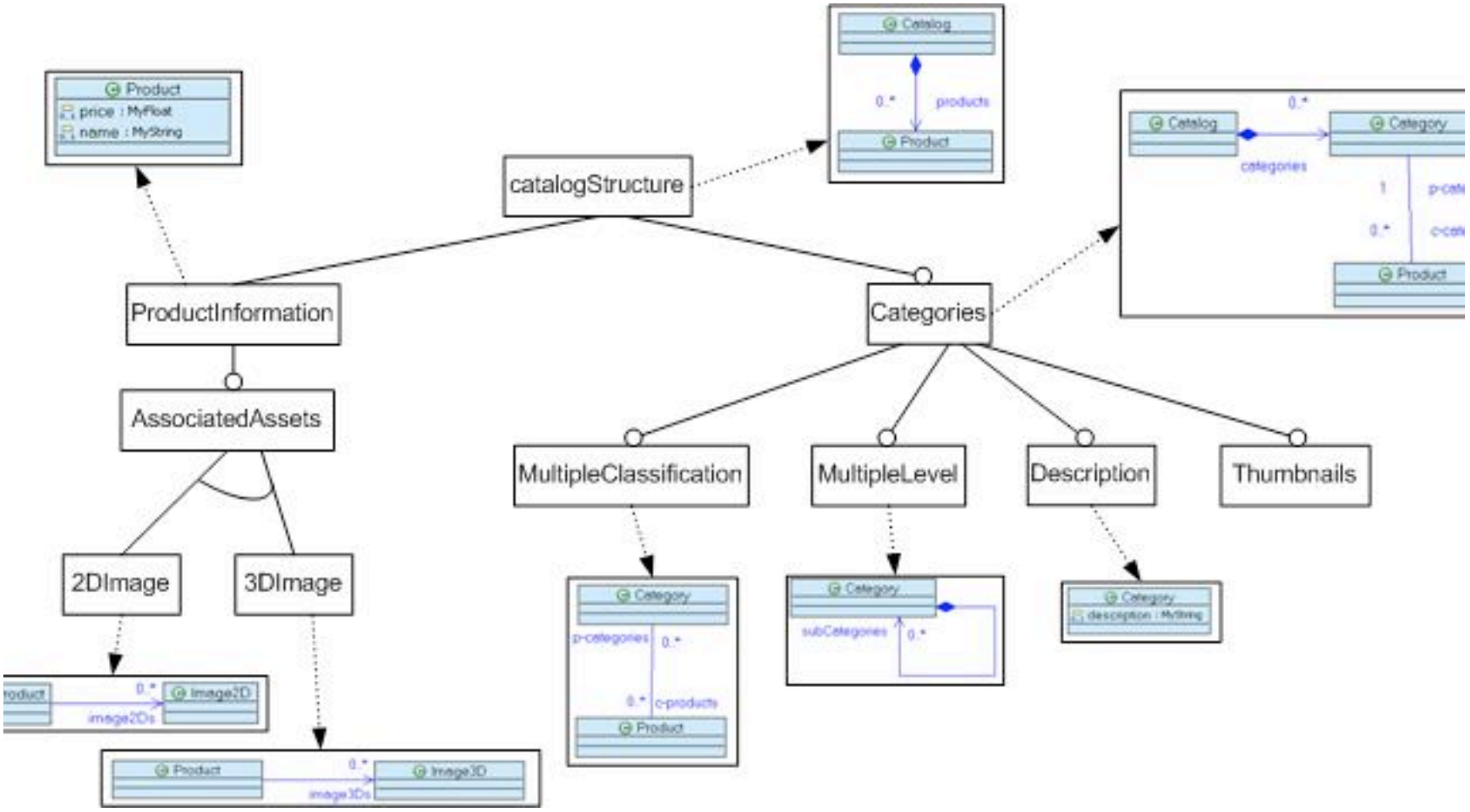
Presence conditions:

true		MultiLevel	4
AssociatedAssets	1	MultipleClassification	5
PhysicalGoods	2	Categories & !MultipleClassification	6
Categories	3	MultipleClassification | !Categories	7

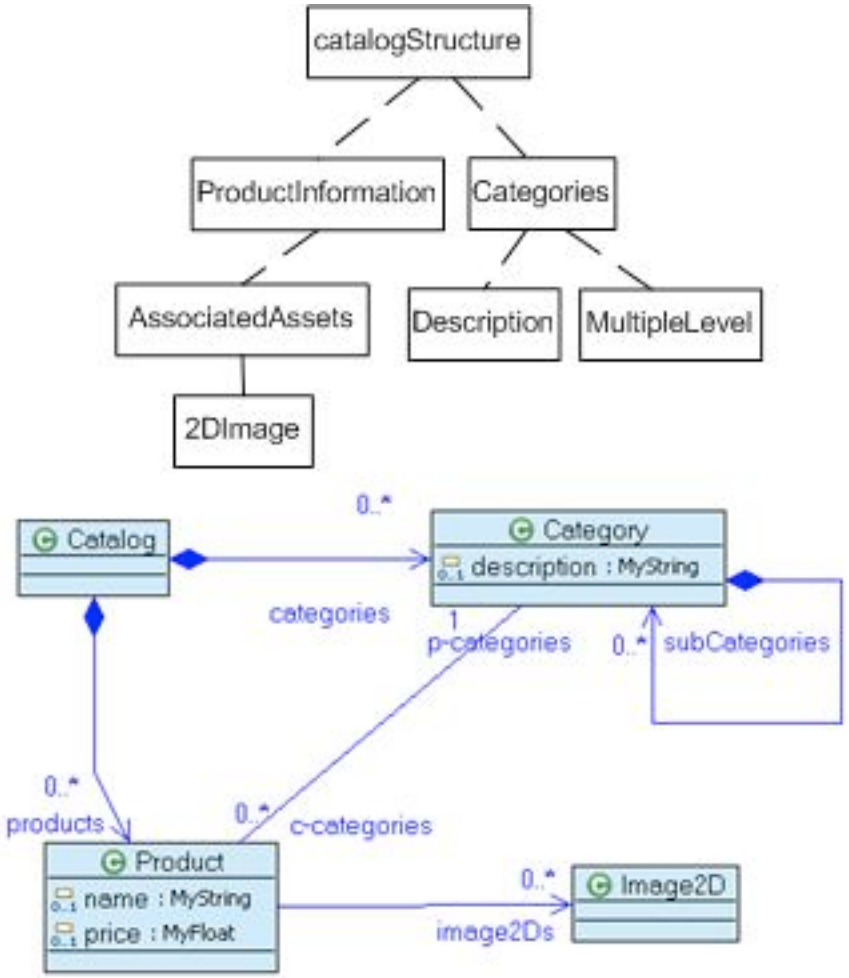


[Perrouin et al. (SPLC'08)]

Another approach



Composition

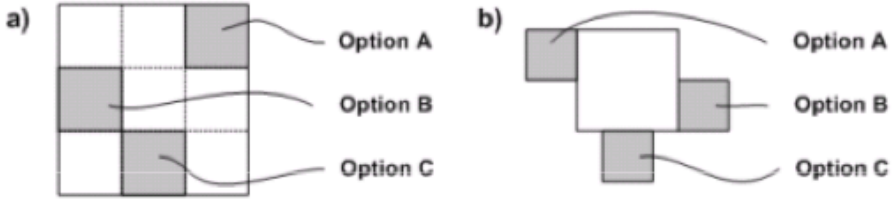


[Voelter et al., SPLC'07)]

Realizing variability

- Negative Variability (pruning, annotative)
  - **takes optional parts away** from an „overall whole“
- Positive Variability (merging, compositional)
  - **adds optional parts** to a minimal core

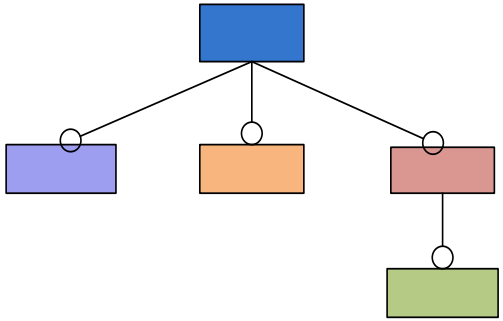
Negative vs. Positive Variability



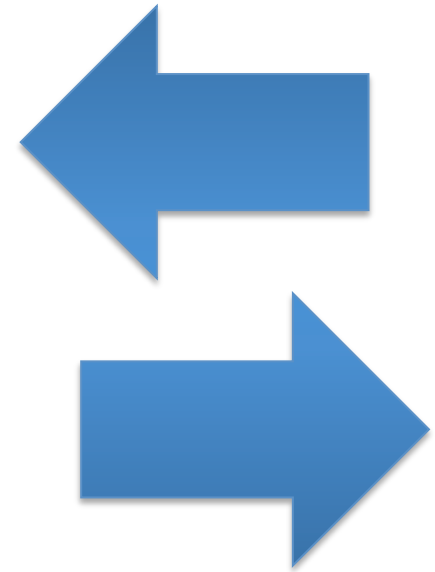
- We need both!

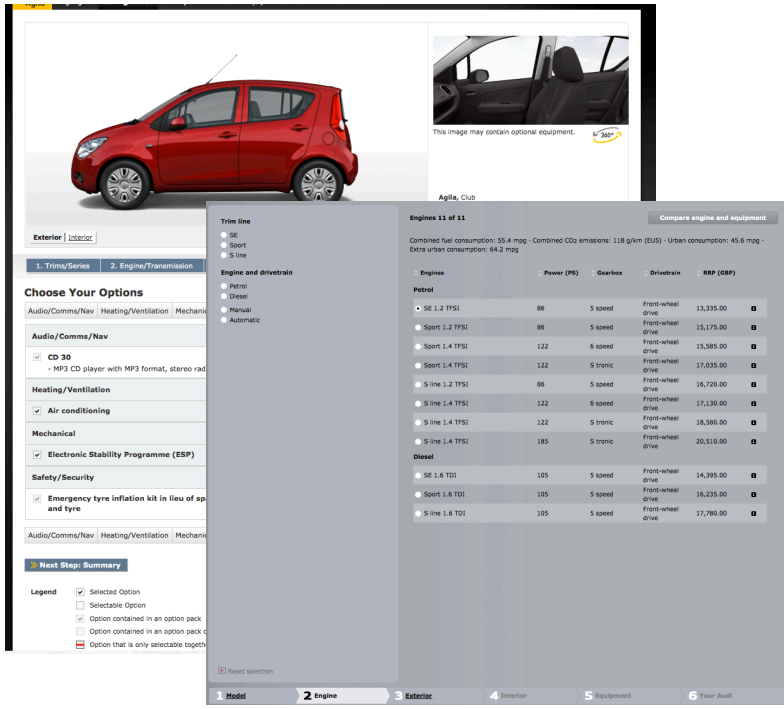
Running project

- Re-engineer a car configurator



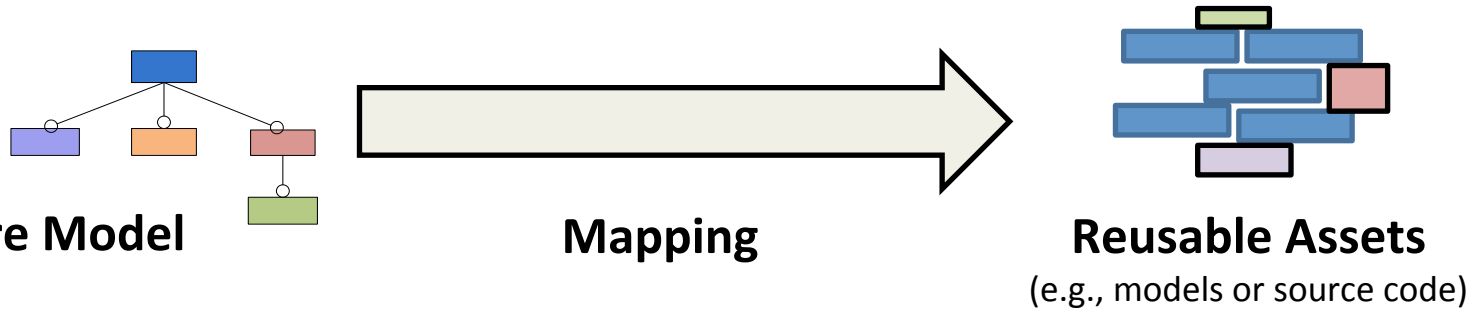
Variability Model (Feature Model)





Summary

- **Software product line engineering**
  - Domain engineering pays off
  - Domain and application engineering
  - Variability management
    - Problem space vs solution space
    - Generative and model-based approaches



Homework for the next course

- Exercice
  - present techniques and tools in the software product line engineering framework
    - notions you have learned during your cursus
    - revisit your requirements and software engineering knowledge