Reuso estratégico com linhas de produtos de software

Paulo Borba
Centro de Informática
Universidade Federal de Pernambuco

Decision models for variability implementation

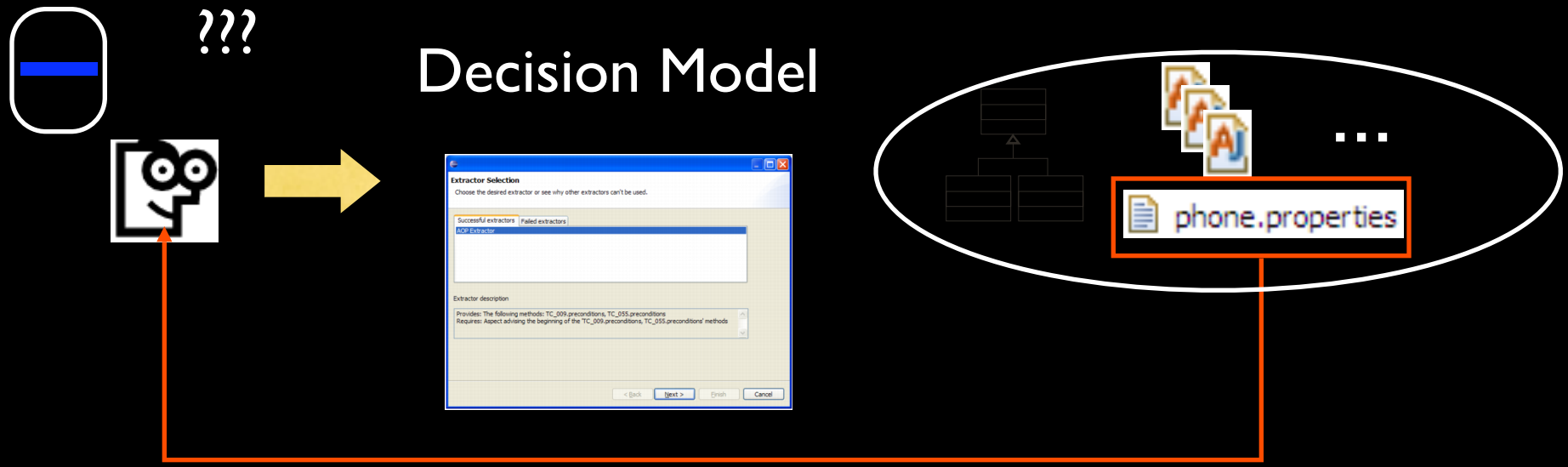
Paulo Borba
Centro de Informática
Universidade Federal de Pernambuco

Problem

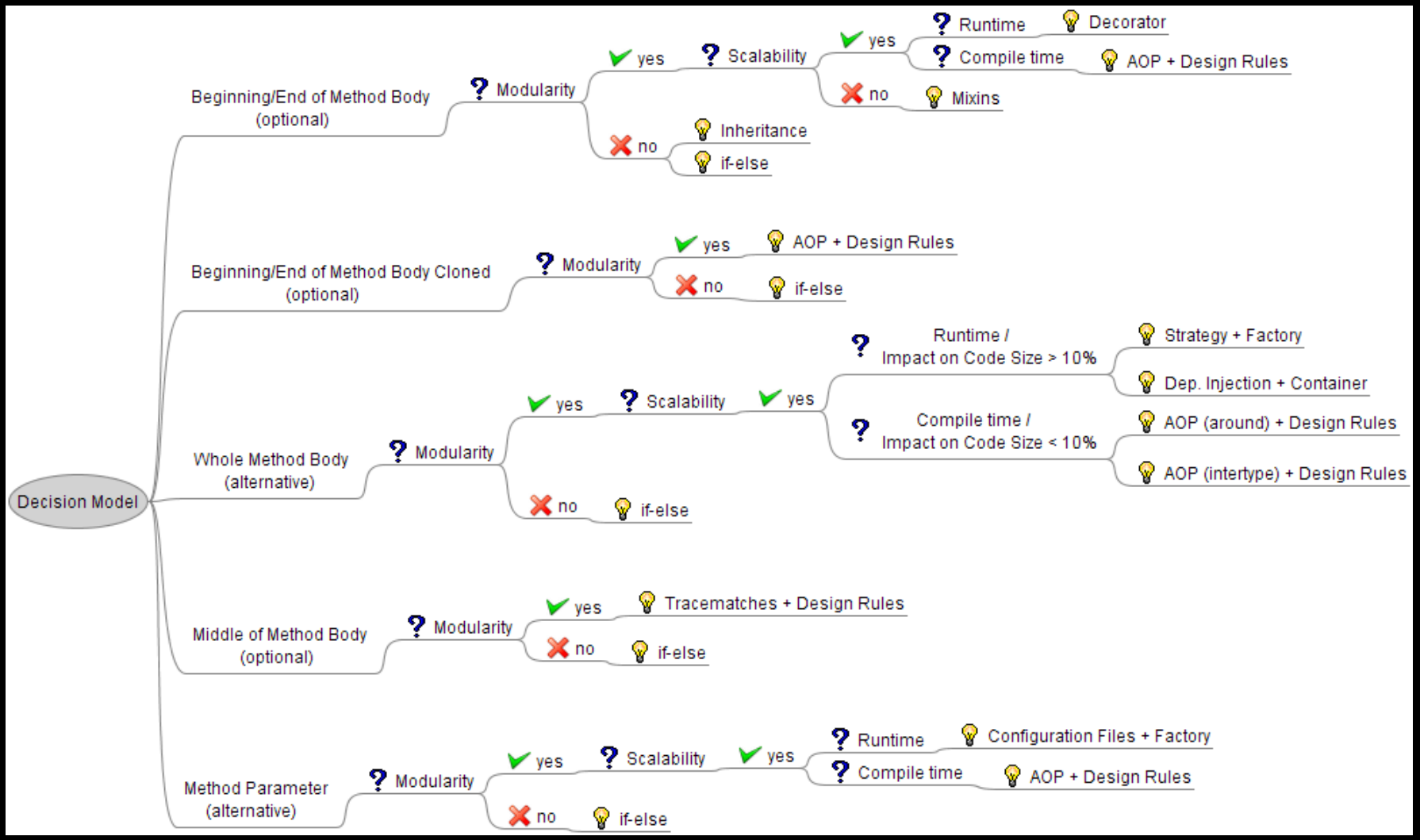
- Which mechanisms to use for **structuring** SPL variabilities?
  - Selecting the **correct** ones…
    - Difficult task due to the great variety of mechanisms
  - Selecting the **incorrect** ones…
    - Negative effects on the cost to maintain the SPL
    - Cloned code and no modularization
    - Increasing developer's effort
    - Decreasing productivity when evolving SPL

Proposed Solution

- Decision Model to help developers when choosing mechanisms to implement variabilities



Decision models



Criteria

Coplien	Anastasopoulos /Gacek	Patzke/ Muthig	CIn-MB	CIn-OB
What is variation and what is commonality ? Positive/ Negative Biding time	Positive/Negative Optional/ Alternative Binding time Traceability Scalability SoC (crosscutting)	Optional/ Alternative/ OR Market and tool support Complexity of the SPL	Location/ Structure Optional/ Alternative Modularity Scalability Binding time	Location/ Structure Optional/ Alternative Binding time Modularity Size SoC/SoV Efficiency

Techniques

Coplien	Anastasopoulos /Gacek	Patzke/ Muthig	CIn-MB	PedroCIn-OB
C++ (Data, Inheritance, Virtual Functions, Overloading, Templates, Preprocessor Directives)	Agregation / Delegation Inheritance Parameterization Properties Dynamic class loading Static libraries Dynamic link libraries Conditional Compilation Frames Reflection AOP	Condional compilation Polymorphism Defaults Refinements	Inheritance Configuration Files Conditional Compilation AOP Mixins Component Technology	Configuration Files AOP Conditional Compilation Inheritance Agregation/ Delegation Mixins

Anastasopoulos and Gacek

	Interface					*Implementation*					*Inititalization*					*Timing*				*Other*		
	Positive	Negative	Optional	Alternative	Multioptional	Positive	Negative	Optional	Alternative	Multioptional	Positive	Negative	Optional	Alternative	Multioptional	Compile	Link	Run	Postrun	Scalability	Traceability	SoC
Aggregation / Delegation						❍	❍	❍	●	●	❍	❍	❍	●	●	❍	❍	❍	❍	●	●	❍
Aspect-oriented programming	❍	●	❍	❍	❍	❍	●	❍	❍	❍	❍	●	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
Conditional Compilation	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍				●	●	❍
Dynamic Class Loading						❍	❍	❍	❍	❍	❍	❍	❍	❍	❍			❍		❍	❍	❍
Dynamic Link Libraries						❍		❍	❍	❍	❍		❍	❍	❍	❍		❍		❍	❍	❍
Frames	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍				●	❍	●
Inheritance	❍		❍		❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍		❍		●	❍	❍
Overloading	❍	❍		❍		❍	❍		❍		❍	❍		❍		❍				●	●	●
Parameterization						❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍		❍	❍	●	❍	●
(Delphi) Properties						❍	❍	❍	●	●	❍	❍	❍	●	●	❍	❍	❍	❍	●	●	●
Static Libraries						❍		❍	❍	❍	❍		❍	❍	❍		❍			●	●	❍

Legend: ❍: possible; ●: ineffective / difficult; ?: questionable; blank: not possible

Anastasopoulos and Gacek

	C++	Delphi	Java	Smalltalk
Aggregation / Delegation	❍	❍	❍	❍
Aspect-oriented programming			❍	❍
Conditional Compilation	❍	❍	❍	❍
Dynamic Class Loading			❍	?
Dynamic Link Libraries	❍	❍	(JNI*)	❍
Frames	❍	❍	❍	❍
Inheritance	❍	❍	❍	❍
Overloading	❍	❍		❍
Parameterization	❍	❍	❍	❍
(Delphi) Properties		❍		
Static Libraries	❍	❍	❍	❍

Legend:

❍: possible ●: ineffective / difficult ?: questionable

blank: not possible * possible with Java Native Interface

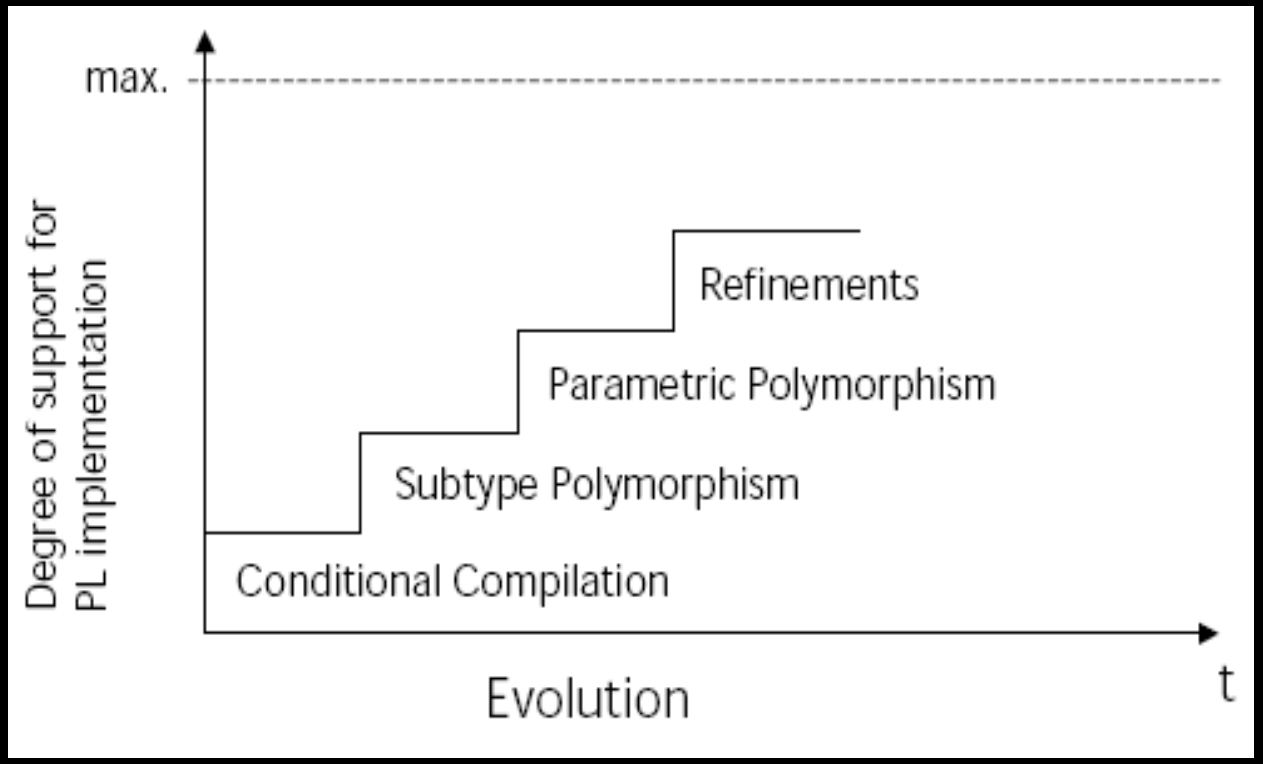
Patzke and Muthig

	Optional	XOR	OR
Conditional Compilation	X	x	
Subtype Polymorphism		X	
Parametric Polymorphism	x	X	X
Overloading		X	
Coercion	x	x	x
Defaults	X		
Collaborations	x	x	X
AOP	x	x	X

Patzke and Muthig

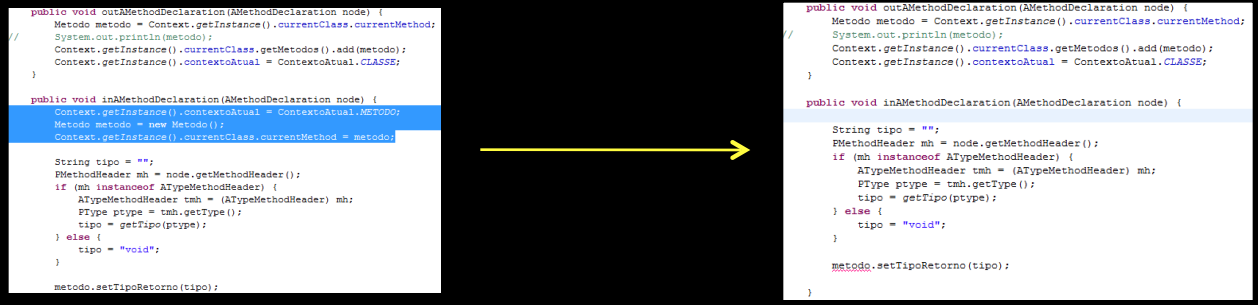
	Advantages	Disadvantages
Conditional Compilation	well-known, no space/ performance loss fine-grained	no direct language support, unscalable
Subtype Polymorphism	often well-known, dynamic	performance loss, OR-variability difficult to express
Parametric Polymorphism	no performance loss, all 3 kinds of variability expressible, built-in	less known, less supported
Ad-Hoc Polymorphism	often well-known	less important than universal polymorphism
Collaborations	good support for PL Implementation	generally not well-known, might require tools
AOP	good support for PL Implementation	often requires tools, market caution
Frame Processor	good support for PL Implementation	not well-known, additional tool

Techique support for SPL complexity



CIn

- Identification of Variation Types
  - Variations are classified by the location where they appear in the source code and the structure of the programming language that varies
    - Focus on the extractive and reactive SPL adoption approaches
    - Supports the development of variations extraction tools



```
public void outAMethodDeclaration(AMethodDeclaration node) {
    Metodo metodo = Context.getInstance().currentClass.currentMethod;
//  System.out.println(metodo);
    Context.getInstance().currentClass.getMetodos().add(metodo);
    Context.getInstance().contextoAtual = ContextoAtual.CLASSE;
}

public void inAMethodDeclaration(AMethodDeclaration node) {
    Context.getInstance().contextoAtual = ContextoAtual.METODO;
    Metodo metodo = new Metodo();
    Context.getInstance().currentClass.currentMethod = metodo;

    String tipo = "";
    PMethodHeader mh = node.getMethodHeader();
    if (mh instanceof ATypeMethodHeader) {
        ATypeMethodHeader tmh = (ATypeMethodHeader) mh;
        PType ptype = tmh.getType();
        tipo = getTipo(ptype);
    } else {
        tipo = "void";
    }

    metodo.setTipoRetorno(tipo);
```

```
public void outAMethodDeclaration(AMethodDeclaration node) {
    Metodo metodo = Context.getInstance().currentClass.currentMethod;
//  System.out.println(metodo);
    Context.getInstance().currentClass.getMetodos().add(metodo);
    Context.getInstance().contextoAtual = ContextoAtual.CLASSE;
}

public void inAMethodDeclaration(AMethodDeclaration node) {

    String tipo = "";
    PMethodHeader mh = node.getMethodHeader();
    if (mh instanceof ATypeMethodHeader) {
        ATypeMethodHeader tmh = (ATypeMethodHeader) mh;
        PType ptype = tmh.getType();
        tipo = getTipo(ptype);
    } else {
        tipo = "void";
    }

    metodo.setTipoRetorno(tipo);
}
```

Patterns

- Patterns for addressing each variation type
  - Patterns can use a single technique or a combination of techniques
  - The same technique can be used by more than one way to address a specific variation type
- Evaluation and comparison of the impact of each pattern on the quality attributes of a SPL
  - Definition evaluation criteria for comparing SPL implementations

VT 1: Variation in Class Attribute

- **Where**: The variation occurs at the declaration of an attribute
- **What**: The variation can be the initialization value for the attribute (alternative values) or the existence of the attribute (optional attributes)

```
public class MainCanvas {
    public static final int SCREEN_WIDTH = 128;
    public static final int SCREEN_HEIGHT = 128;
    [...]
}
```

```
public class MainCanvas {
    public static final int SCREEN_WIDTH = 176;
    public static final int SCREEN_HEIGHT = 220;
    [...]
}
```

```
public class Resources {

    public static final String CHEAT_CODE_FPS = [...];
    public static final String CHEAT_CODE_FPST = [...];
    public static final String CHEAT_CODE_UNLOCK_LEVELS = [...];

    [...]
}
```

Patterns for Addressing Variation in Class Attribute

- Attribute Defined by Conditional Compilation
- Attribute Defined by Inter-Type Declaration
- Attribute's Initial Value Defined by Configuration File
- Optional Attribute Defined by Inheritance

VT2: Variation in Class Hierarchy

- **Where**: Declaration of a class
- **What**: The superclass or the list of extended interfaces
- Patterns:
  - Class Hierarchy Defined by Conditional Compilation
  - Class Hierarchy Defined by Inter-Type Declaration

VT3: Variation in the Whole Body of a Method

- **Where**: In the body of a method
- **What**: The whole body of the method
- Patterns
  - Whole Body of a Method Defined by Conditional Compilation
  - Whole Body of a Method Defined by Inter-Type Declaration
  - Whole Body of a Method Defined by Around Advice
  - ...

VT4: Variation in the Beginning or End of Method Body

- **Where**: In the beginning or end of a method or constructor body
- **What**: A block of consecutive source code lines
- Patterns:
  - Variation in the beginning or End of Method Body Defined by Conditional Compilation
  - Variation in the Beginning or End of Method Body defined by After/Before advice
  - ...

VT5: Variation in a Structure Around Method Body

- **Where**: The body of a method
- **What**: A conditional or loop structure that surrounds the method body
- Patterns:
  - Structure Around Method Body Defined by Conditional Compilation
  - Structure Around Method Body Defined by Around Advice
  - Structure Around Method Body Defined by Inheritance

Completeness of Variation types and Patterns

- Some variation types have been identified in another domain
  - Evidence that they might be frequent
- Not every variation matches one of our variation types
  - Restrictions to the application of patterns can also arise
    - Refactoring!
- We do not prove that our set of variation types and patterns are complete
  - Our model can be extended to consider new variation types and patterns

Decision models for variability implementation

Paulo Borba
Centro de Informática
Universidade Federal de Pernambuco

Decision Models for Implementing SPL Variabilities

- Multi-Paradigm Design, 2000
- Implementing Product Line Variabilities, 2001
- Product Line Implementation Technologies, 2002
- Restructuring test variabilities in Software product lines, 2008
- Analysis of Techniques for Implementing Variabilities in Software Product Lines, 2008