Variability: What’s new?

Jim Coplien

Gertrud&Cope

Mørdrup, Denmark

Background

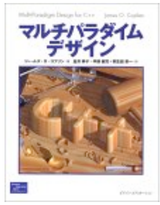
- ECE / CS Background, University of Wisconsin
- Ph.D. @ VUB, 2000
- Bell Labs: development, tech transfer, 1979 – 1990
- Bell Labs Research, 1990 – 2000
  - Domain Engineering, Multiparadigm Design, Architecture Patterns, Organizational Patterns
- Academics at NCC, UMIST, Adelaide

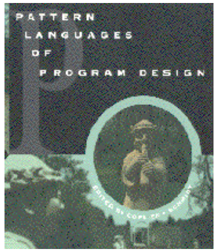
Today…

- Independent researcher & consultant
- Gertrud&Cope, Mørdrup, Denmark
  - http://www.gertrudandcope.com
  - Large variety of in-house research programs with partners
- ScrumHouse
  - Research with DKU on cultural mappings
  - Lean Architecture, anti-TDD
  - http://www.scrumorgpatterns.com
- Joint research with Trygve Reenskaug on DCI architecture
- Pattern research with Aalborg University
- Working on a new book

Variability stuff

- Early work in object-oriented design
- Commonality /variability correspondences in problem/ solution domain
- Patterns — software and real architecture





What does a programming language express?

- Programming languages have “features”
- Features express semantics important to model building
- These features are:
  - Logical (the logic of problem solving)
  - Structural (the structure of systems)
- They express design models
  - Discovery is 30% – 50%
  - Coding is only 5%

The basic cognitive models

- Human minds see patterns
- Patterns can be characterized as:
  - The same thing again and again
  - Recurring commonality
  - Recurring variability
  - e.g. writing out a check

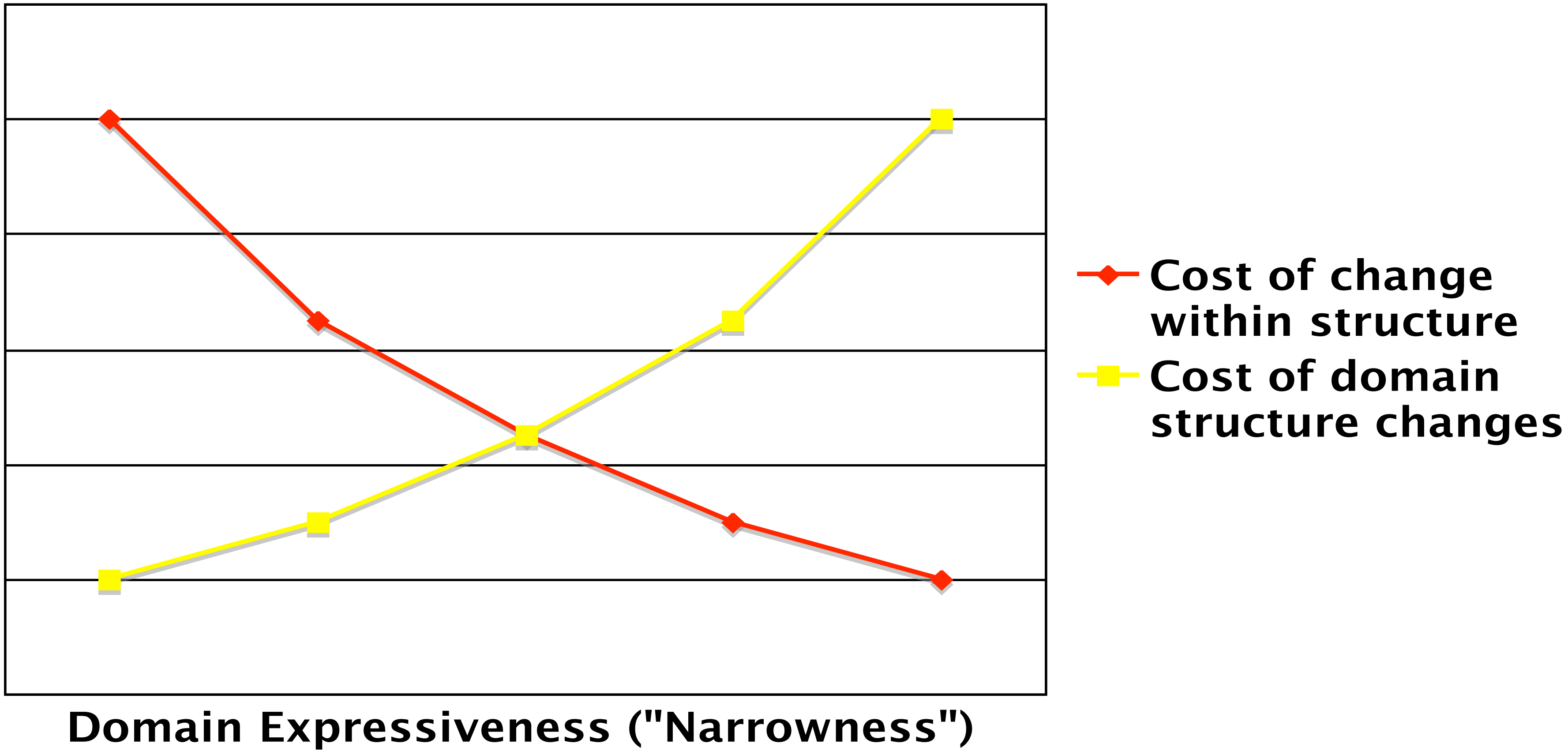
What is programming?

1. Model building
   - Most of a program doesn't solve a problem but models the environment
   - The model is a context for problem solving
2. Problem solving
   - The goal: To turn around solutions fast

Levels of Purposefulness

- A checkbook programming language
  - Structure: Like my checkbook
  - Problems: writing checks, reconciliation
- Excel
  - Structure: Ledger accounting
  - Problems: many, including checks/reconciliation
- OOPLs
  - Structure: Many, including ledger accounting
  - Problems: many…
- FORTRAN
  - Structure: Algorithms
  - Problems: Algorithm problems

For any language



The only constant is change

- We can predict very mature domains
- Experience suggests that we’re bad at this
- Why?
  - Good domain analyses take >6 months
  - Today’s agile markets expect >2 releases every six months
  - There is rarely enough time to design a language that captures the domain just right

Horrors! Going to a general–purpose language?

- Domain specific languages express commonalities and variations, too
- Concept starter sets [Simos1996]
- Remarkably small!
  - Structure
  - Behavior
  - Name
  - . . . .

Text Buffer Variability Table

TextBuffer: Common Structure and Behavior

Parameters of Variability	Meaning	Domain	Binding	Default / Technique
Output Type	The formatting of text lines is sensitive to the output medium	Database, RCS, TTY, UNIX file	Run	UNIX File
Character Set	Different buffer types support different character sets	ASCII, EBCDIC, FIELDATA	Compile	ASCII
Working Set Management	Different applications need to cache different amounts of memory	Whole file, whole page, LRU, fixed	Compile	Whole file
Debugging Code	Debug in-house only, but keep tests in source code	Debug, production	Compile	None

Text Buffer Transformational Analysis

TextBuffer: Common Structure and Behavior

Parameters of Variability	Meaning	Domain	Binding	Default / Technique
Output Type *Structure, Algorithm*	The formatting of text lines is sensitive to the output medium	Database, RCS, TTY, UNIX file	Run	UNIX File *Virtual Functions*
Character Set *Non-structural*	Different buffer types support different character sets	ASCII, EBCDIC, FIELDATA	Compile	ASCII *Templates*
Working Set Management *Algorithm*	Different applications need to cache different amounts of memory	Whole file, whole page, LRU fixed	Compile	Whole file *Inheritance*
Debugging Code *Code Fragments*	Debug in-house only, but keep tests in source code	Debug, production	Compile	None *#ifdef* *(from Negative variability Table)*

Transformational Analysis Table

Commonality	Variability	Binding	Instantiation	C++ Feature
Function Name and Semantics	Anything other than algorithm structure	Source	N/a	Template
	Fine algorithm	Compile	N/a	#ifdef
	Fine or gross algorithm	Compile	N/a	Overloading
Data Structure	Value of State	Run Time	Yes	Struct, simple types
	A small set of values	Run time	Yes	Enum
	Types, values and state	Source	Yes	Template
Related Operations and Some Structure	Value of State	Source	No	Module
	Value of State	Source	Yes	struct, class
	Data Structure and State	Compile	Optional	Inheritance
	Algorithm, Data Structure and State	Compile	Optional	Inheritance
		Run	Optional	Virtual Functions

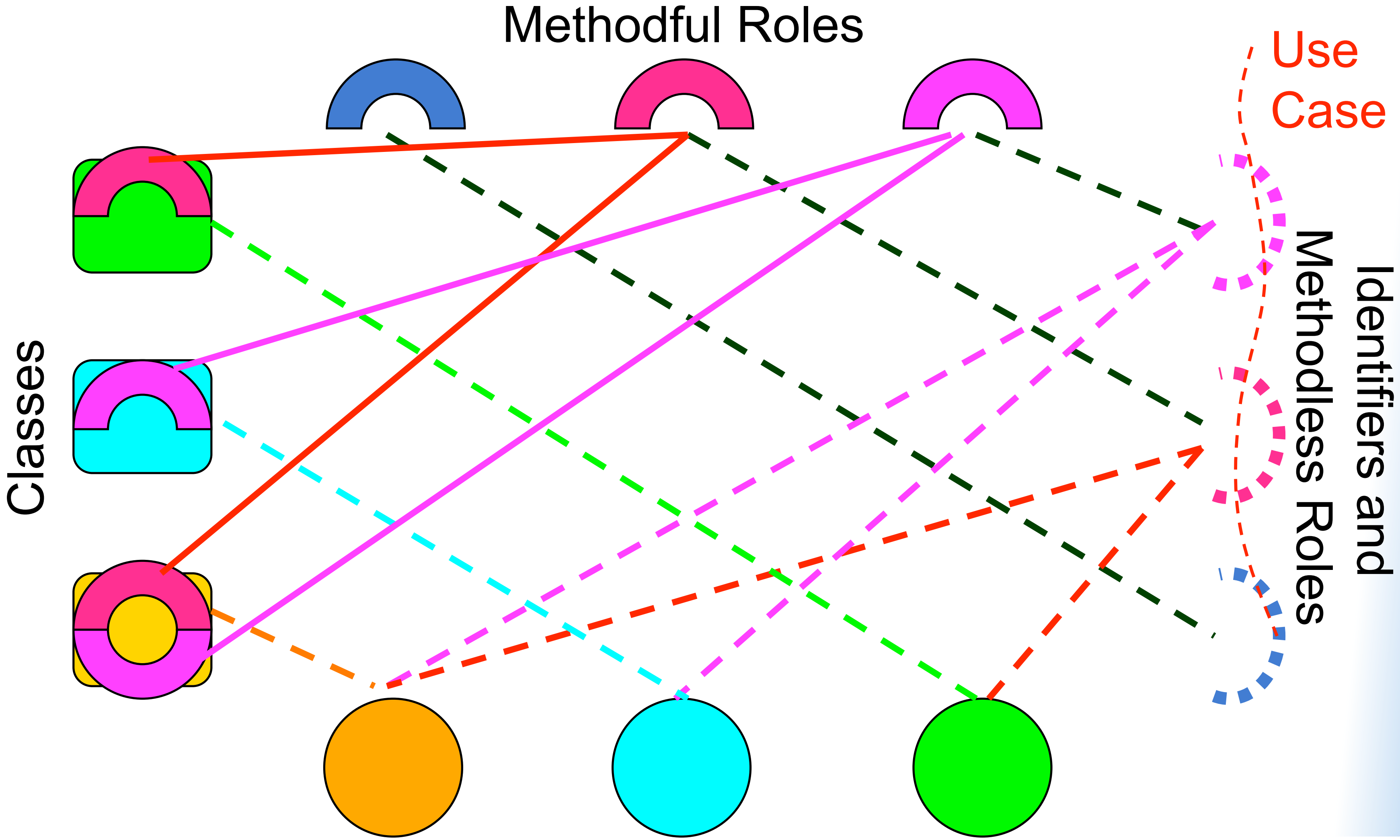
For Java

Commonality	Variability	Binding	Instantiation	Java Feature
Function Name and Semantics (forced to be within a class scope)	Anything other than algorithm structure	Source	N/a	Generic
	Fine algorithm	Compile	N/a	#ifdef
	Fine or gross algorithm	Compile	N/a	Overloading (restricted to non-built-in operations)
Data Structure	Value of State	Run Time	Yes	struct, simple types
	A small set of values	Run time	Yes	enum
	(class) Types, values and state	Source	Yes	Generic
Related Operations and Some Structure	Value of State	Source	No	Module
	Value of State	Source	Yes	struct, class
	Data Structure and State	Compile	Optional	Inheritance
	Algorithm, Data Structure and State	Compile	Optional	Inheritance
		Run	Optional	Virtual Functions

For C#

Commonality	Variability	Binding	Instant-iation	C# Feature
Function Name and Semantics (forced to be within a class scope)	Anything other than algorithm structure	Source	N/a	Generic
	Fine algorithm	Compile	N/a	Tag parameters
	Fine or gross algorithm	Compile	N/a	Overloading
Data Structure	Value of State	Run Time	Yes	struct, simple types
	A small set of values	Run time	Yes	enum
	(class) Types, values and state	Source	Yes	Generic (but no operators)
Related Operations and Some Structure	Value of State	Source	No	static class
	Value of State	Source	Yes	struct, class
	Data Structure and State	Compile	Optional	Inheritance
	Algorithm, Data Structure and State	Compile	Optional	Inheritance
		Run	Optional	Virtual Functions

Reenskaug’s DCI demonstrates that standard OO captures behavior variability



Industrial experience

- Good languages take time
- A compiler/translator is the trivial part
  - Uniform debugger that maintains intentionality at run time
  - Configuration management / impact-of-change analysis tools
  - Documentation support tools (as Rational Rose does to link Java with UML)
  - Compatible/uniform type system (CLR equivalent)
  - Re-factoring tools, source browsers, code optimizers…
  - Field update tools / strategies
  - Language training materials, language reference documentation
  - Data persistence framework for language data elements
  - Line coverage testing tools
  - Unit testing frameworks (à la xUnit)
  - Language-oriented editor (in the sense that most modern editors “understand” Java)
  - Reusable (!) libraries of code written *in* the DSL (?!!)
- Learning curve rises with number of languages
- DSLs are brittle unless very-well designed.

A recent client

… but they have architecture rot, loss of conceptual integrity, 15-layer Java inheritance, and training latencies

Client conclusions

- DSLs help coding tremendously
  - Reduce turnaround cycles from hours to seconds
- DSLs increase the discovery costs
  - Lack of inter-domain reasoning: too many DSLs
  - Lack of architectural vision — *even though all DSLs share a common, rich type system analogous to the CLR*

DSLs that survive

- AuditDraw
  - … but long-term experience was questionable
- Voice XML
  - W3C standard for ACDs
  - thriving, but took ten years to refine
- yacc, bison, excel
  - culturally universal

Other important findings

- Domain analysis is good, but vulgar programming languages are enough for implementation (down to C!)
- Leveling continues to be a crucial problem
- Heterogeneous environments struggle to thrive
- DSLs are a cynical form of employee retention

In conclusion

- The future belongs to well-designed low-level general-purpose languages
- A handful of DSLs will still find a place
- DSL creation is a discipline
- You still need good architecture, and that addresses the lion’s share of development cost
- Don’t trust a language hacked together in a few weeks