IDEs as Ecosystems

Andrew P. Black
joint work with Daniel Vainsencher

Portland State University

Context

Context

- IDE ≠ fancy text editor

Context

- IDE ≠ fancy text editor
- IDE is a collection of tools

Context

- IDE ≠ fancy text editor
- IDE is a collection of tools
  - operating on the *same* program

Context

- IDE ≠ fancy text editor
- IDE is a collection of tools
  - operating on the *same* program
  - reveling *different* aspects of that program

Context

- IDE ≠ fancy text editor
- IDE is a collection of tools
  - operating on the *same* program
  - reveling *different* aspects of that program
  - probably targeted at different programming tasks

Context

- IDE ≠ fancy text editor
- IDE is a collection of tools
  - operating on the *same* program
  - reveling *different* aspects of that program
  - probably targeted at different programming tasks
  - having a deep knowledge of the language, and

Context

- IDE ≠ fancy text editor
- IDE is a collection of tools
  - operating on the *same* program
  - reveling *different* aspects of that program
  - probably targeted at different programming tasks
  - having a deep knowledge of the language, and
  - able to support semantic transformations

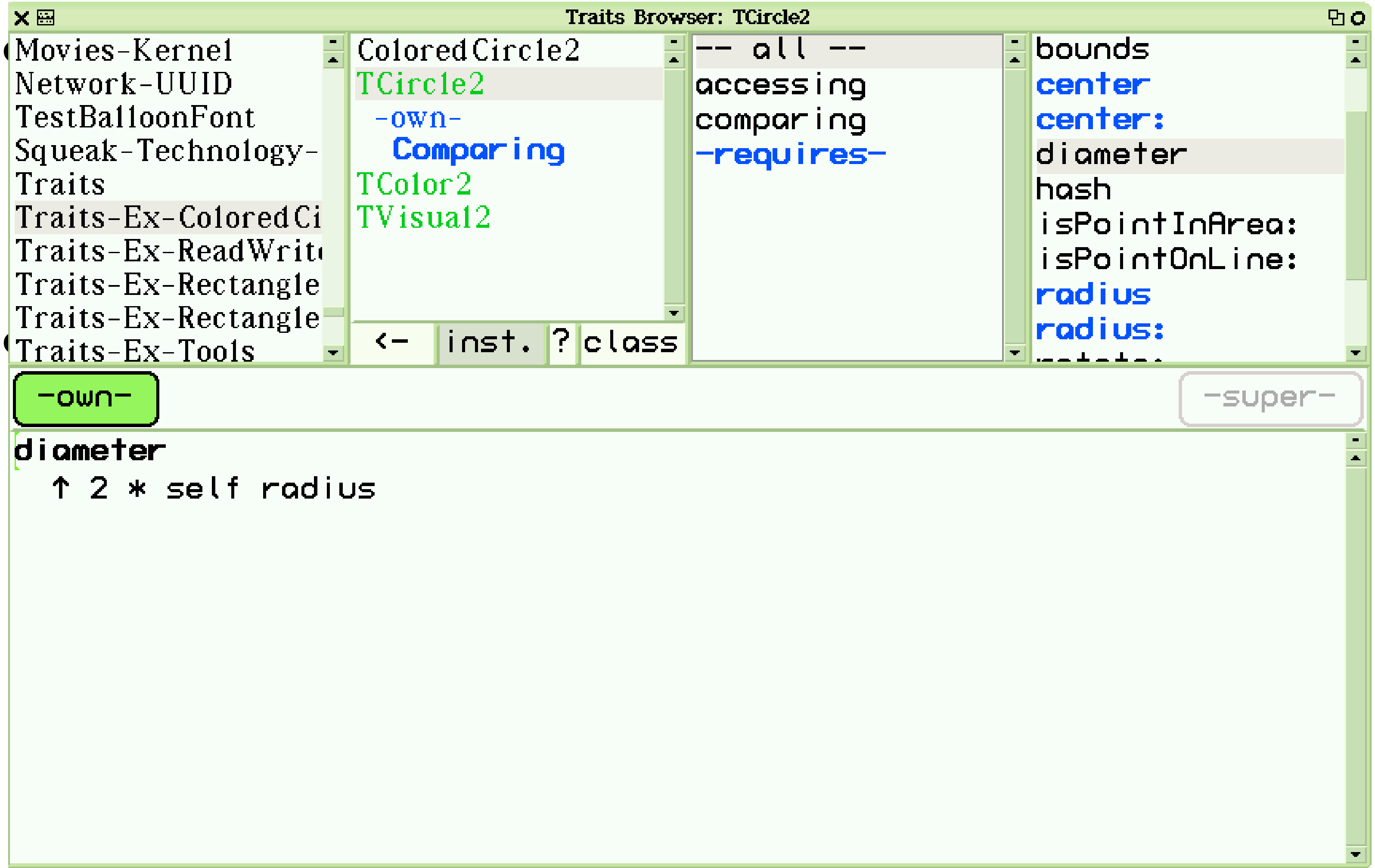
Context

- IDE ≠ fancy text editor
- IDE is a collection of tools
  - operating on the *same* program
  - reveling *different* aspects ~~TM~~ of that program
  - probably targeted at different programming tasks
  - having a deep knowledge of the language, and
  - able to support semantic transformations

In the beginning…

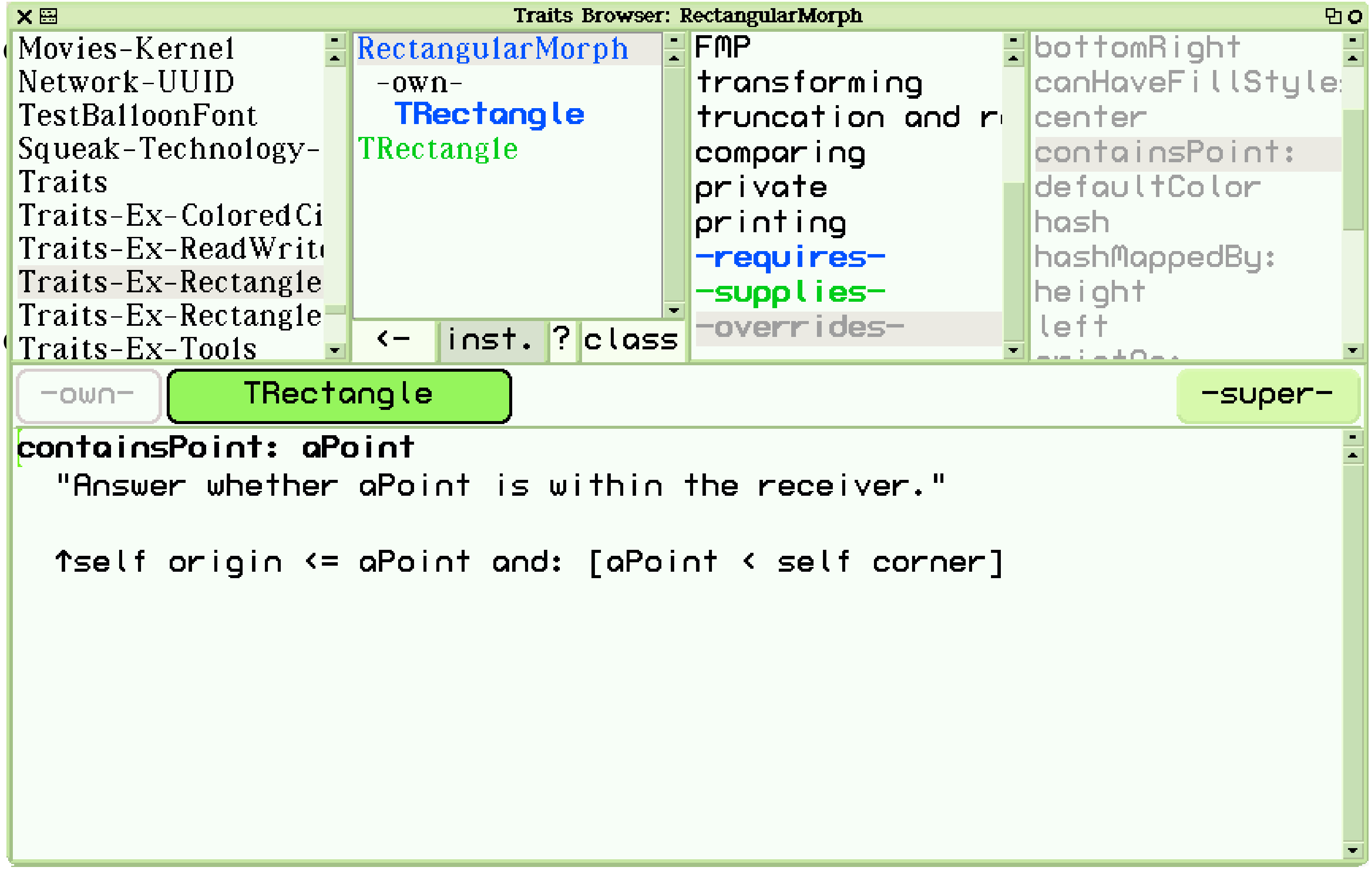
- There were traits.
  - And traits were good, but really hard to program with
- So Nathaniel wrote the “green browser”

In the beginning…



Portland State UNIVERSITY

In the beginning…



Portland State UNIVERSITY

Fast forward several years:

- New:
  - Traits kernel
  - Browser Platform
- Need to re-implement all of the cleverness of the "green browser"
  - Why? The code model, the code analyses and the tools that let us view them were inextricably bound up together
  - "That's just the way IDEs are"

Portland State University

IDEs are Ecosystems

IDEs are Ecosystems

IDEs are Ecosystems

IDE Architects

IDEs are Ecosystems

IDE Architects

IDEs are Ecosystems

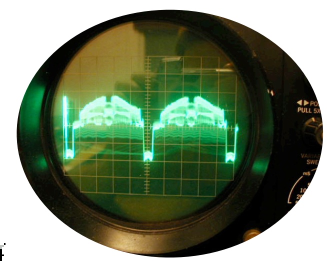
Toolsmiths

IDE Architects

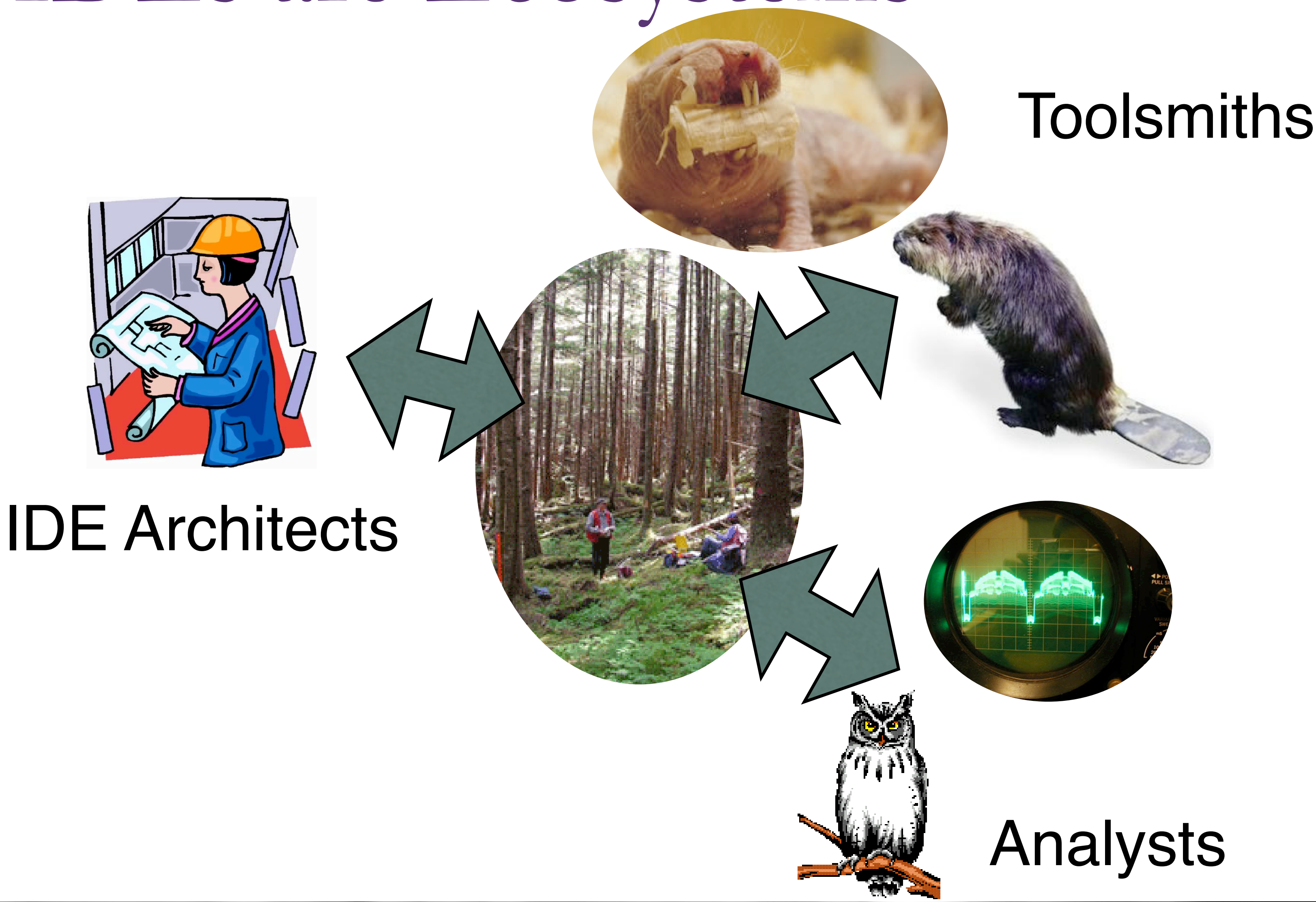
IDEs are Ecosystems

Toolsmiths

IDE Architects



IDEs are Ecosystems



IDEs are Ecosystems

An IDE should be a home for all species

We were not the first

- Brown University, late 1980s:
  - Steve Reiss and his students:
    - The Pecan, FIELD and GARDEN environments
- Scott Meyers [IEEE Softw. 1991]:

  many problems…would be solved if all the tools in a development environment shared a single representation… Unfortunately, no representation has yet been devised that is suitable for all possible tools.

What's the problem?

What's the problem?

- Five "obvious" solutions:
  - Shared File System
  - Selective Broadcast
  - Simple Database
  - View-oriented Database
  - Canonical Representation

What's the problem?

- Five "obvious" solutions:
  - Shared File System
  - Selective Broadcast
  - Simple Database
  - View-oriented Database
  - Canonical Representation
- None of them works

What's the problem?

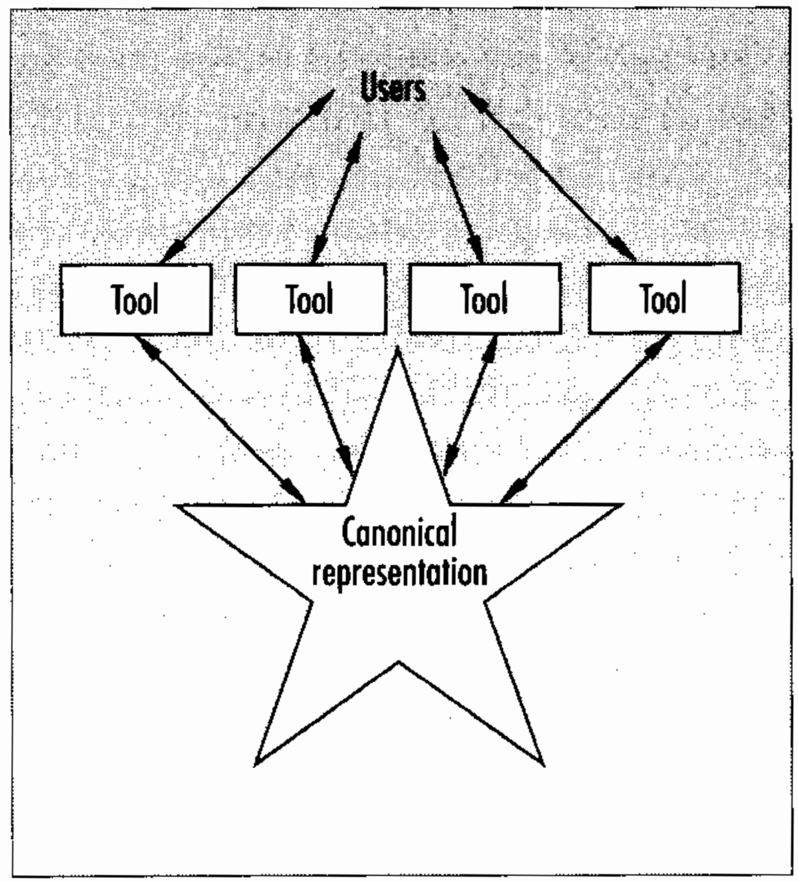
- Five "obvious" solutions:

TABLE 1.
COMPARISON OF FIVE APPROACHES TO ENVIRONMENT INTEGRATION.

Criterion	Shared file system	Selective broadcasting	Simple database	Database with views	Canonical representation
Writing new tools	Good	Fair	Poor	Poor	Fair
Adding new tools	Good	Fair	Poor	Poor	Fair
Simultaneous views	Poor	Fair	Poor	Good	Good
Consistency maintenance	Poor	Fair	Good	Good	Good
Redundancy avoidance	Poor	Poor	Fair	Good	Good

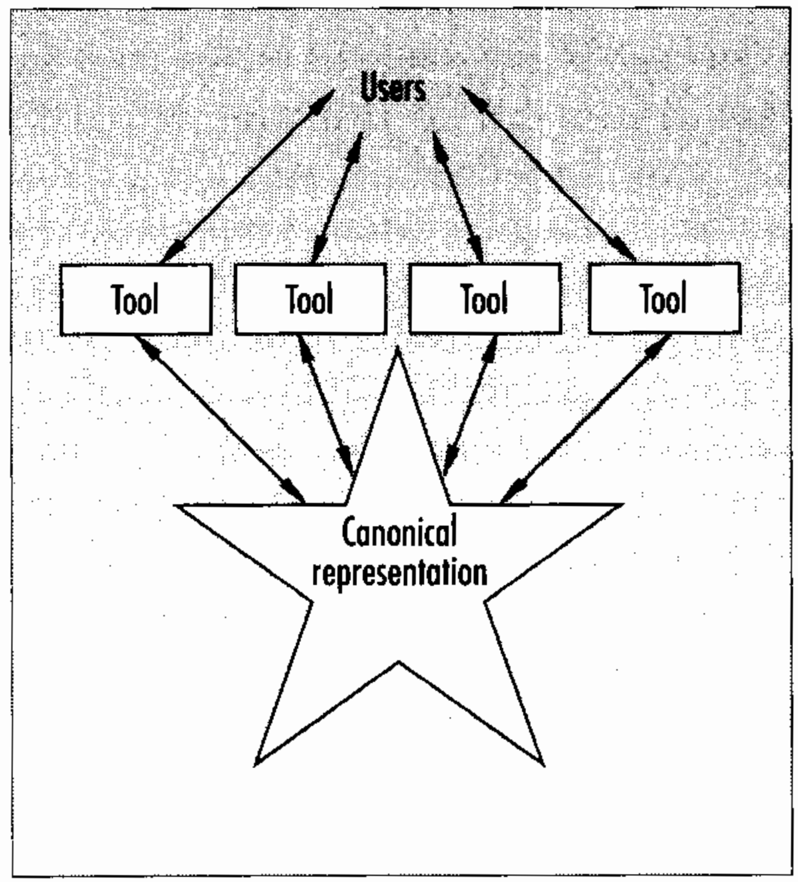
- None of them works

Start with "Canonical Representation"



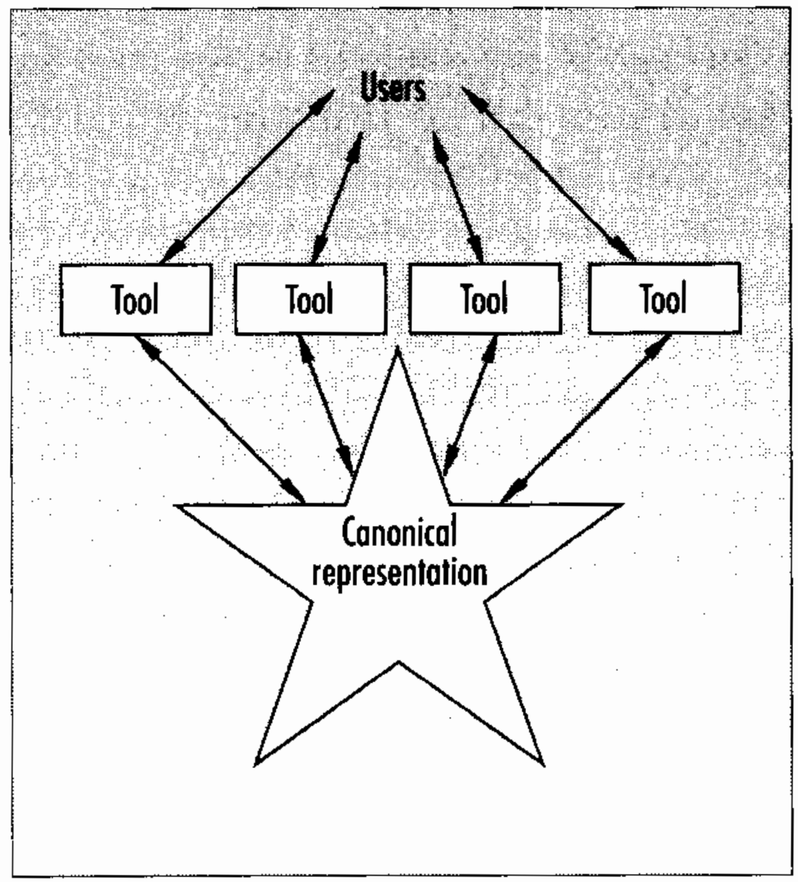
Portland State University

Start with "Shared Code Model"



Start with "Shared Code Model"

- The data needed by every tool must be easily accessible
- Tools must be able to get notifications of changes made by other tools
- "Shared code model" is our first pattern



It's not just about being smarter!

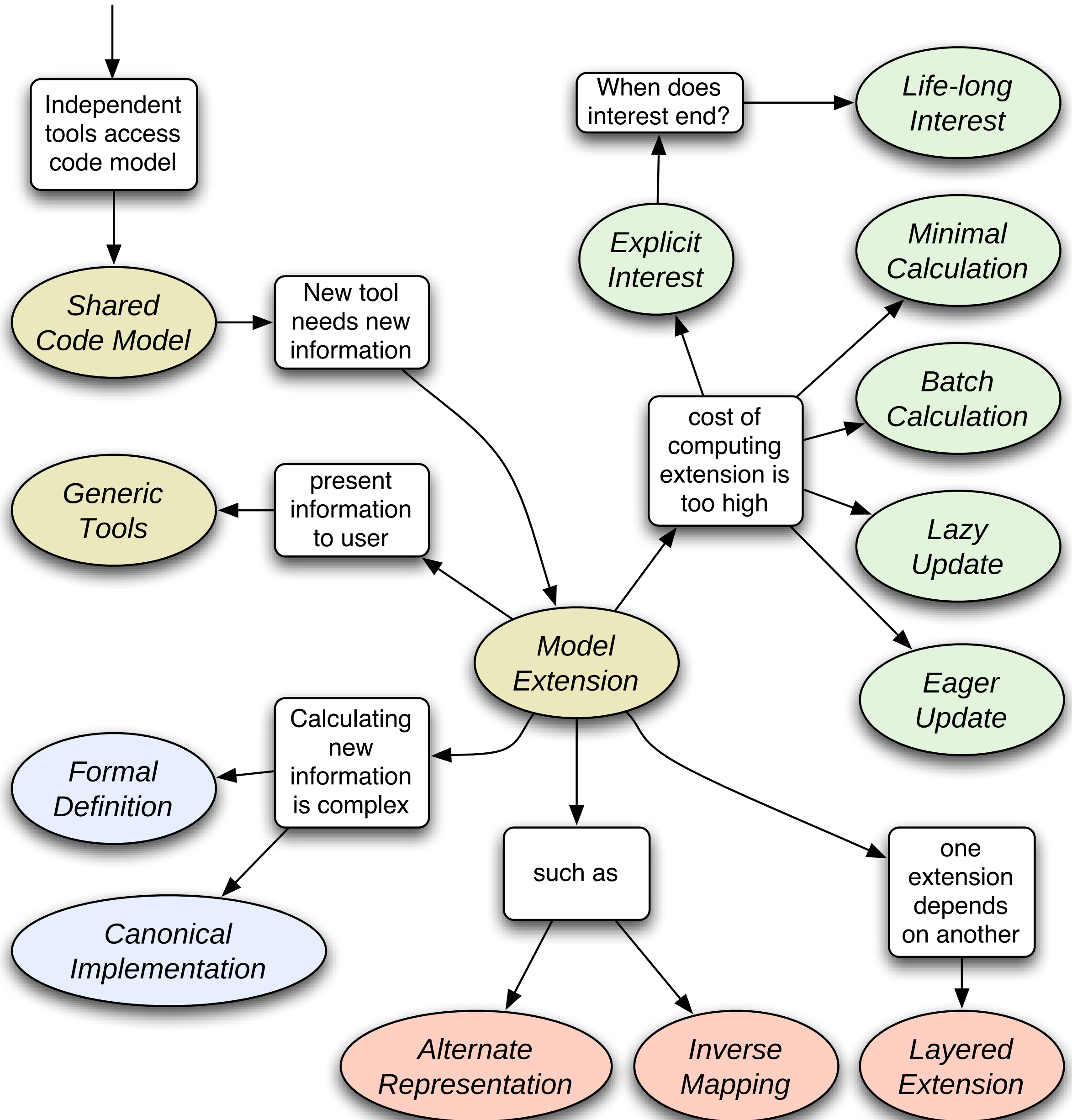
- There *is* no "holy grail"
- No representation can be guaranteed to support the new tool that I'll be building next month or next year

It's not just about being smarter!

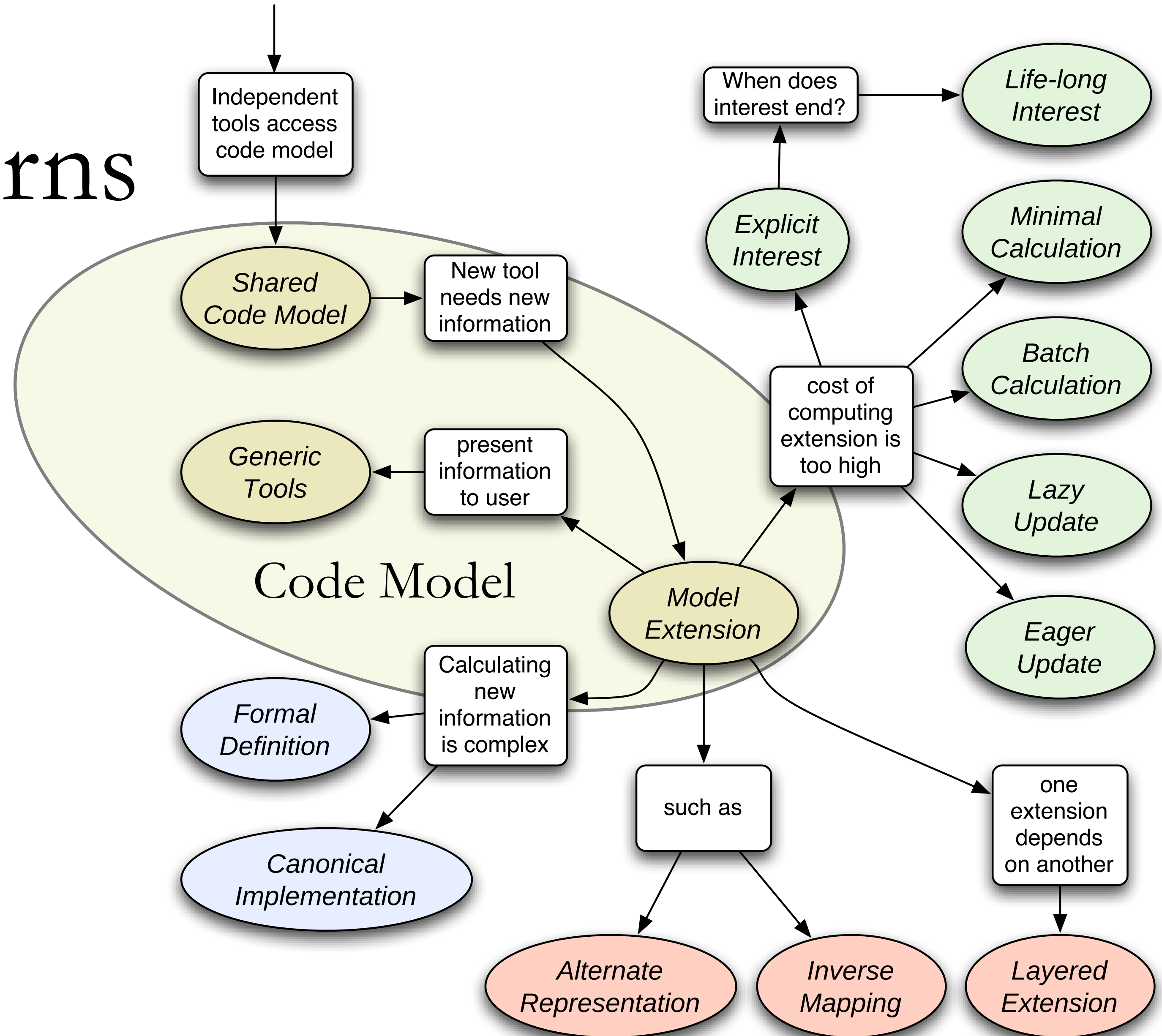
- There *is* no "holy grail"
- No representation can be guaranteed to support the new tool that I'll be building next month or next year

We need an *extensible* representation

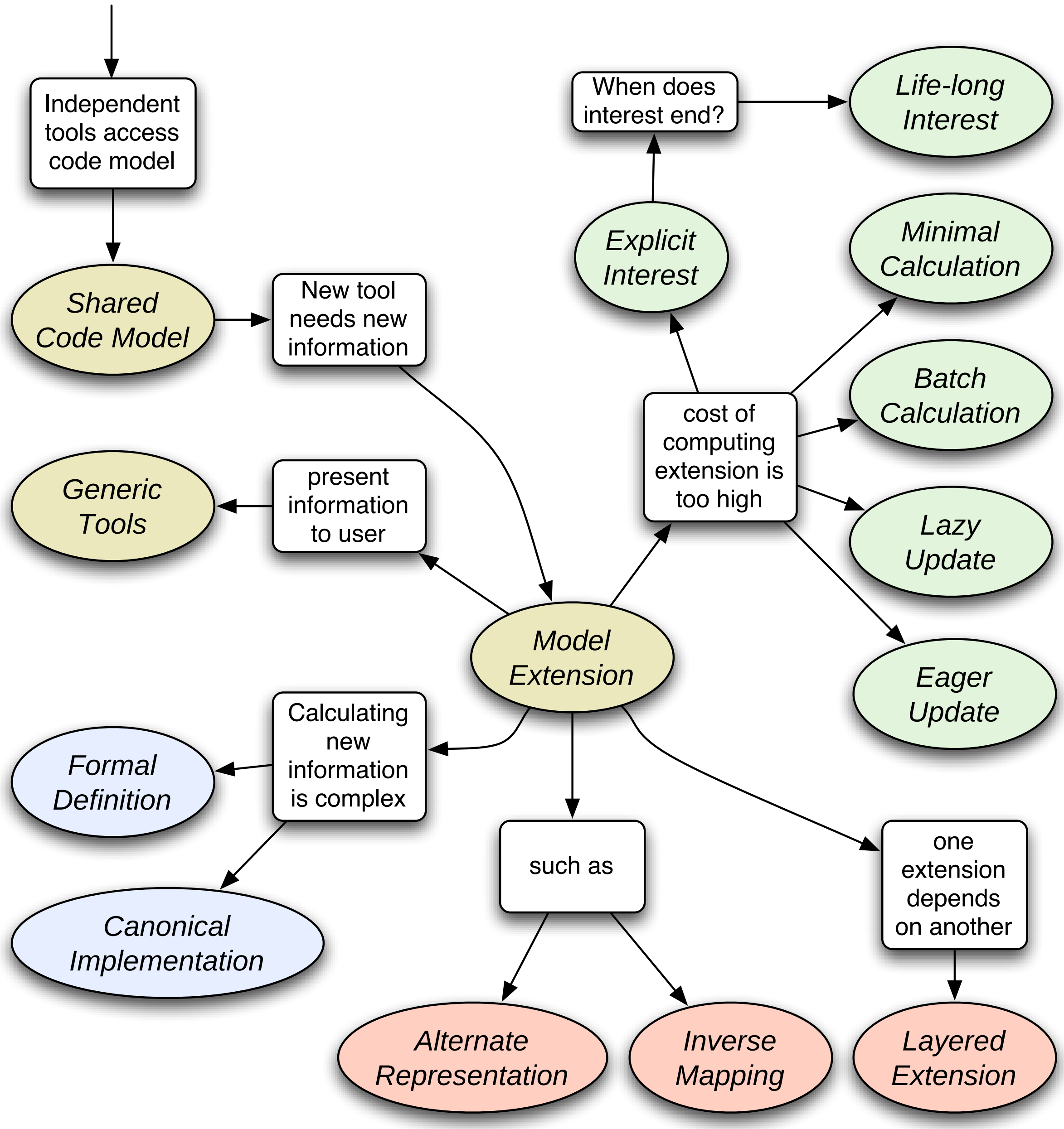
The Patterns



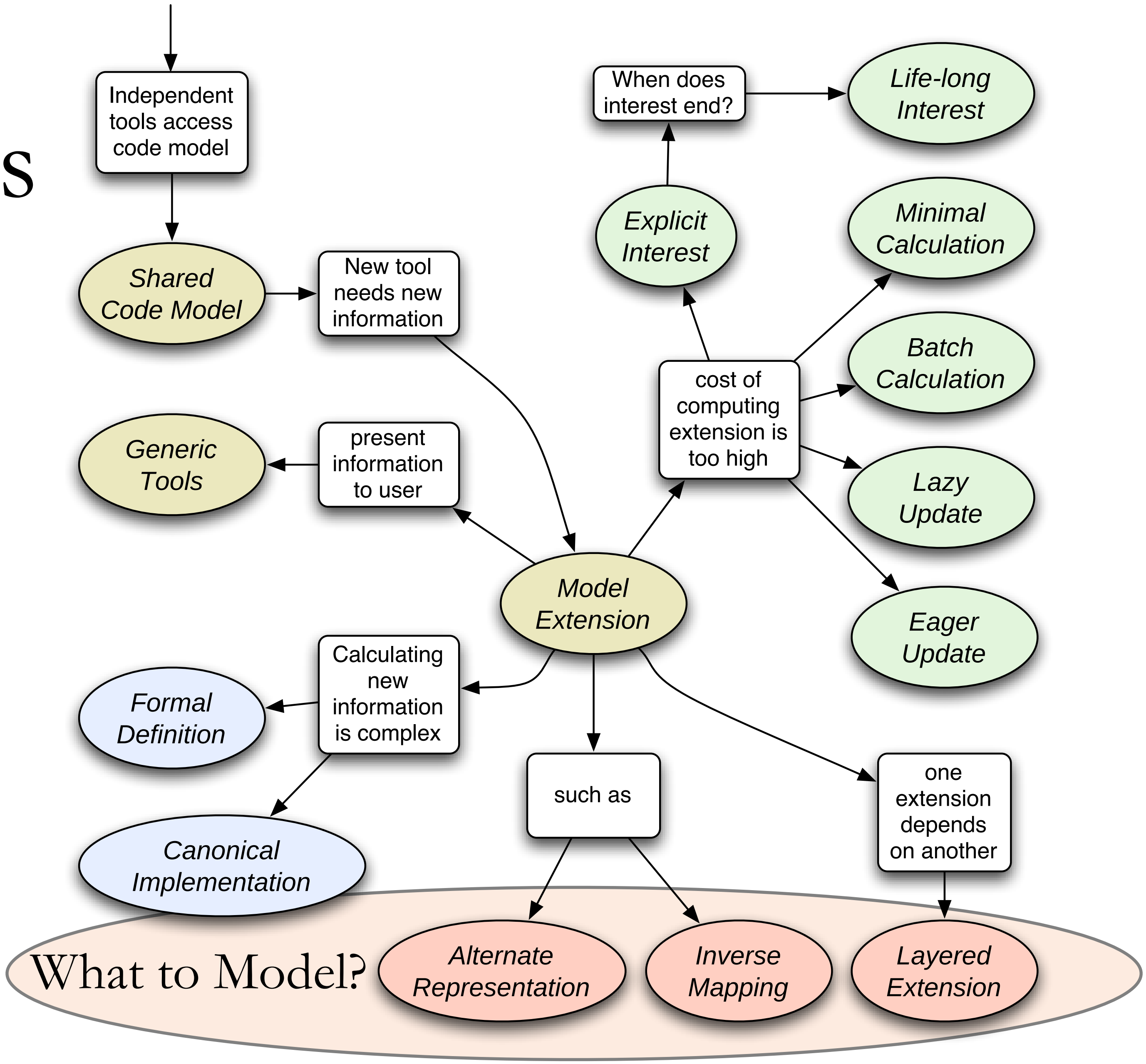
The Patterns



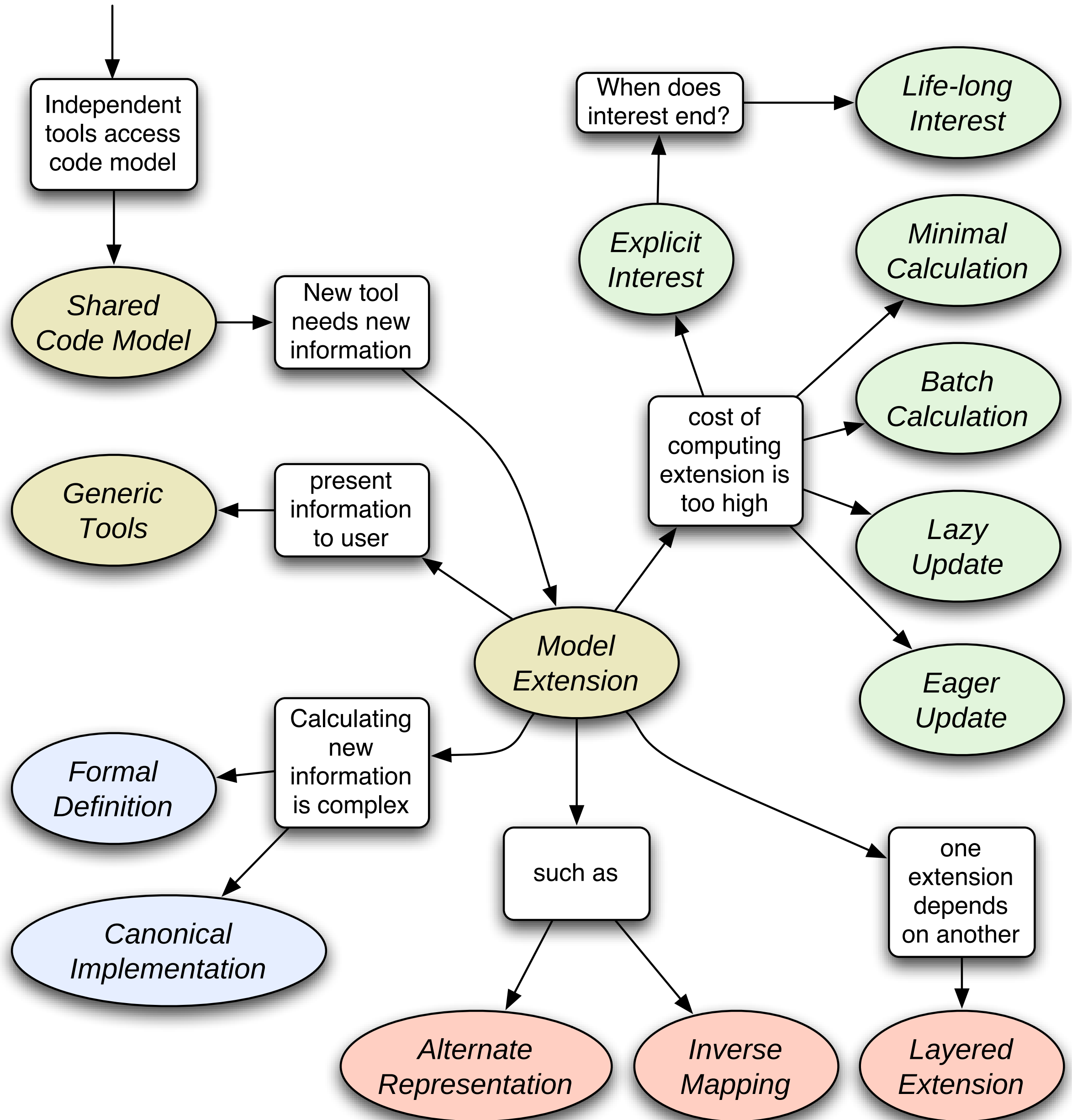
The Patterns



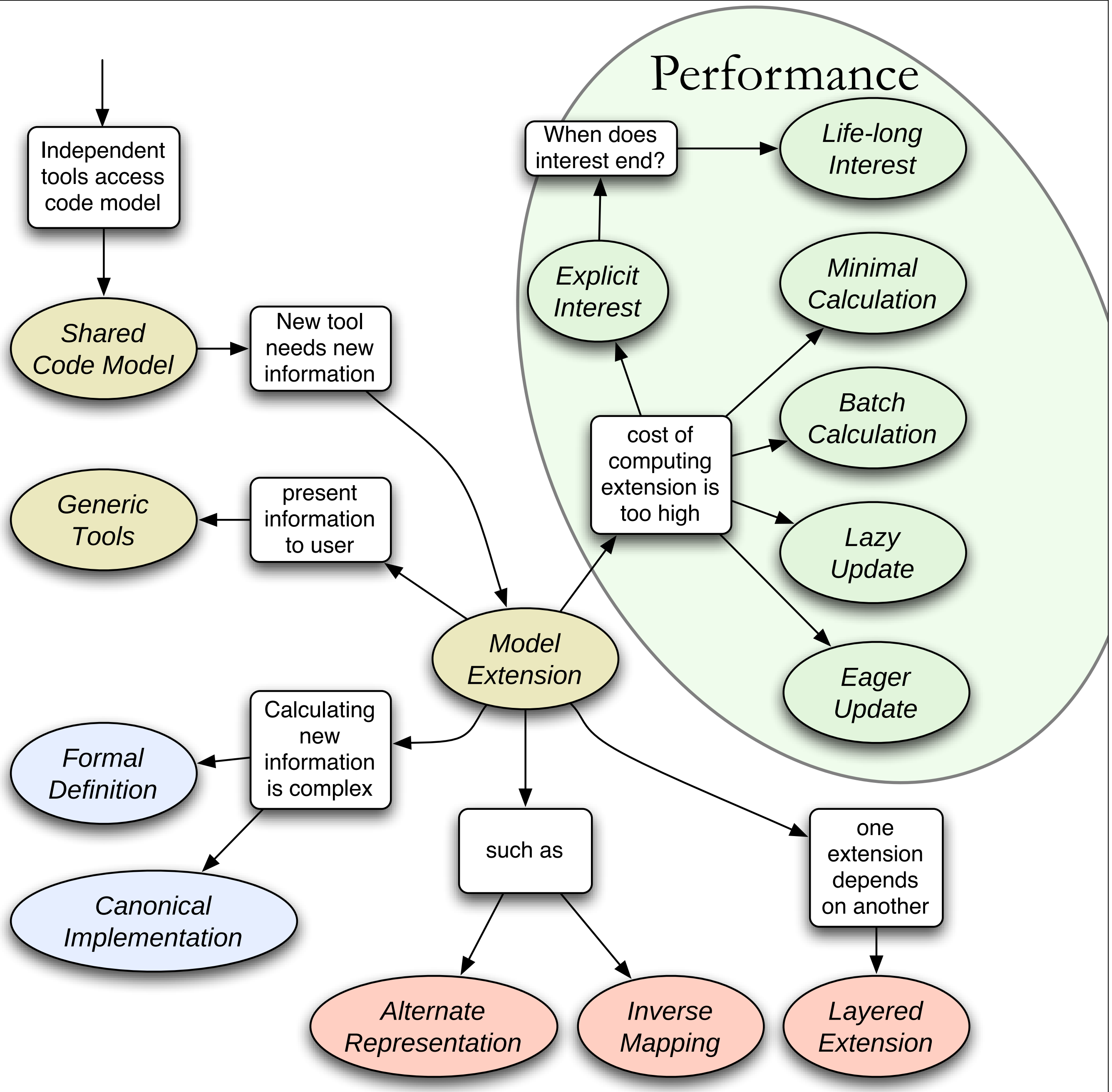
The Patterns



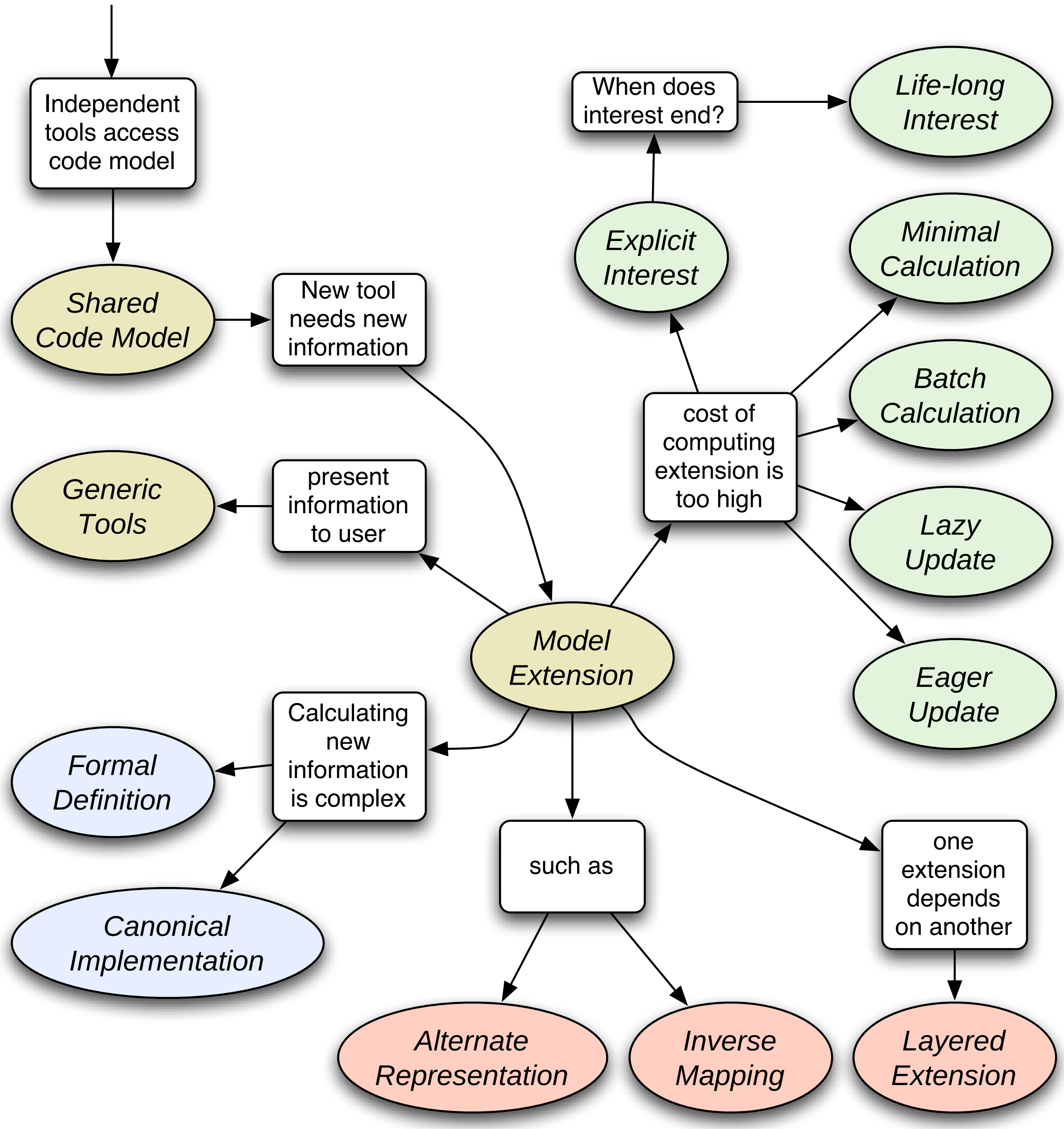
The Patterns



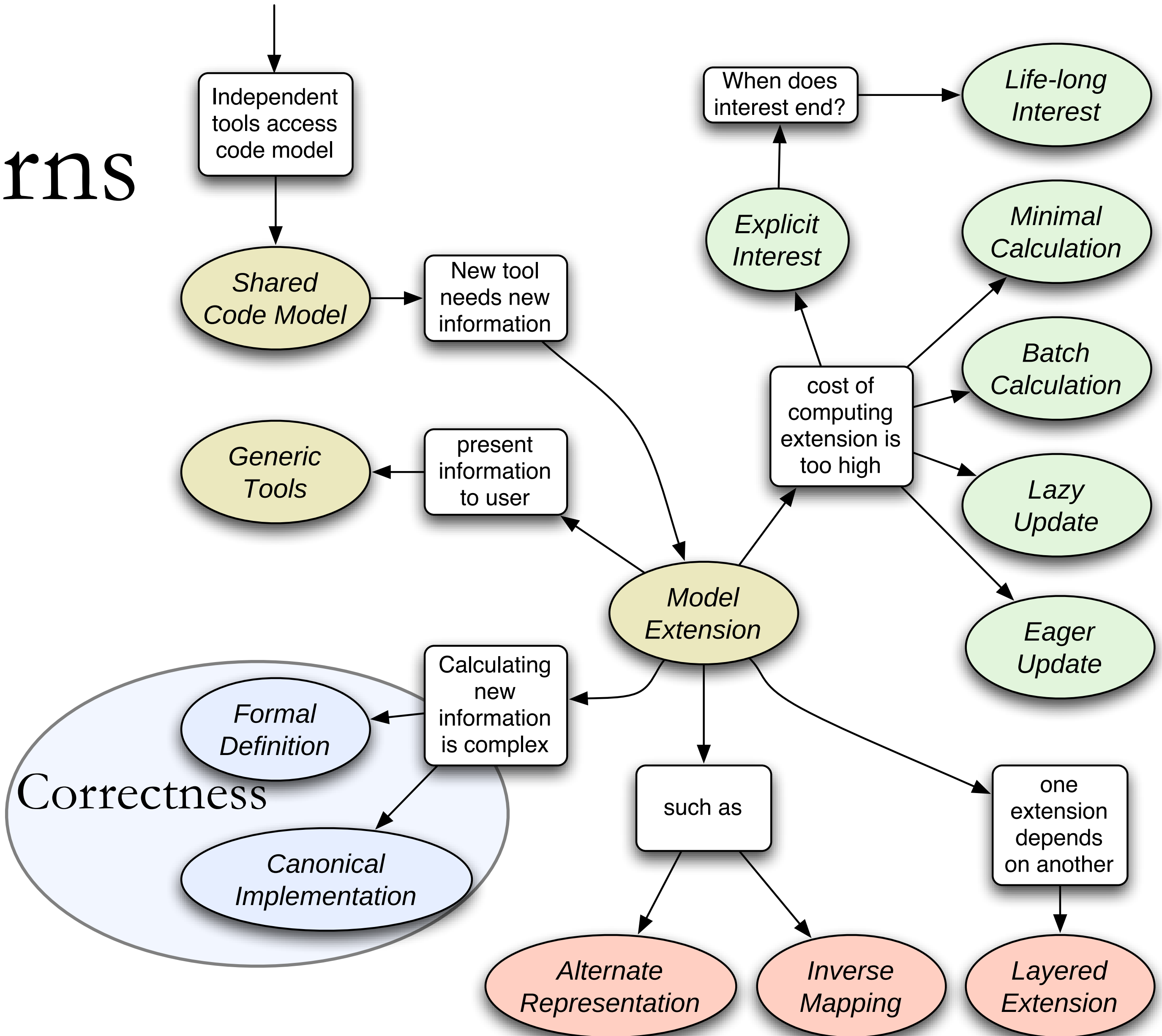
The Patterns



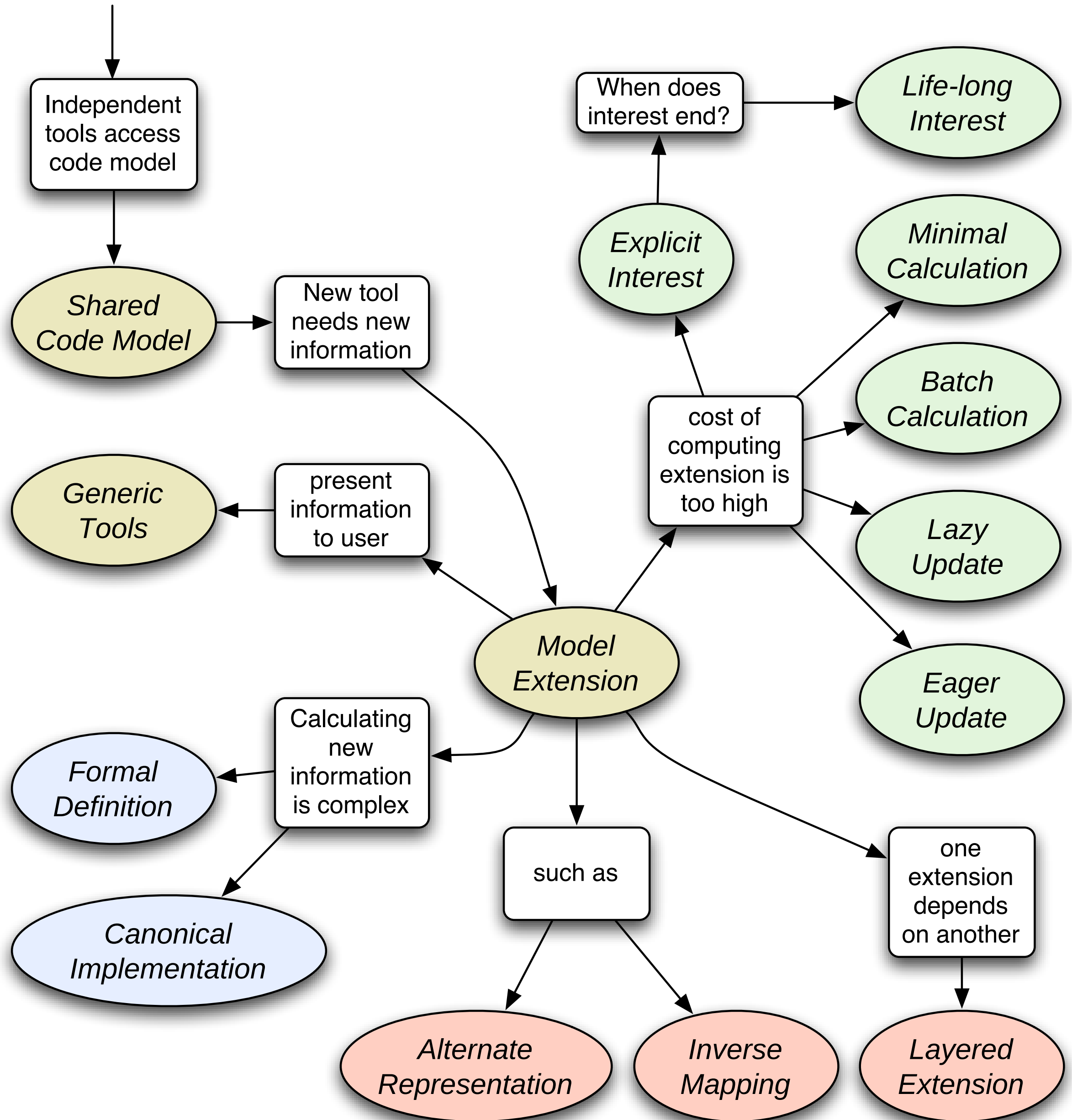
The Patterns



The Patterns

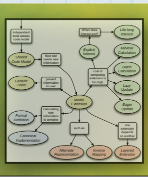


The Patterns



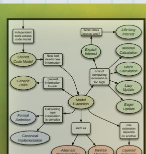
Shared Coded Model

- Single Representation of the program
  - … as a graph of objects
  - organized to enable browsing and searching
  - details can be kept as text
  - keep it simple
  - avoid redundancy



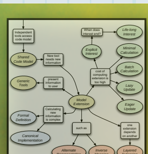
Portland State UNIVERSITY

- Not a new idea
  - Used in Cornell Program Synthesizer
  - Smalltalk
  - Eclipse
    - v 2.1 did not have a shared code model
    - v 3.2 “Java Model” is in memory
  - Cadillac



Consequences:

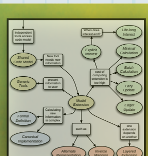
- Need observer pattern
  - shared model + observer ensures that all clients are synchronized
- Navigation and query are quick and easy
- objects representing details can be stored as part of the model, synthesized on demand, or implemented as an Alternative Representation
- Shared Code Model may not be complete
  - use Model Extension



Portland State UNIVERSITY

Model Extension

- What do you do when tools need properties that aren't in the code model?
  - Add them — as *extensions*
- Put the *implementation* in its own class/module, but add the *interface* to the appropriate class of the shared code model



Option a: Non-uniform interface

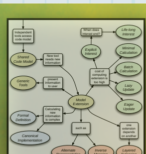
Class

superclass
name
classVarNames
instanceVarNames
…

RequiredMethods

computeFor: aClass
computerForAllsubcla
ssesOf: aClass
    helperMethod1
    helperMethod2

(a) put new property in its own class. There may be multiple interfaces for performance or convenience



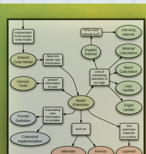
Option b: Unencapsulated implementation

Class

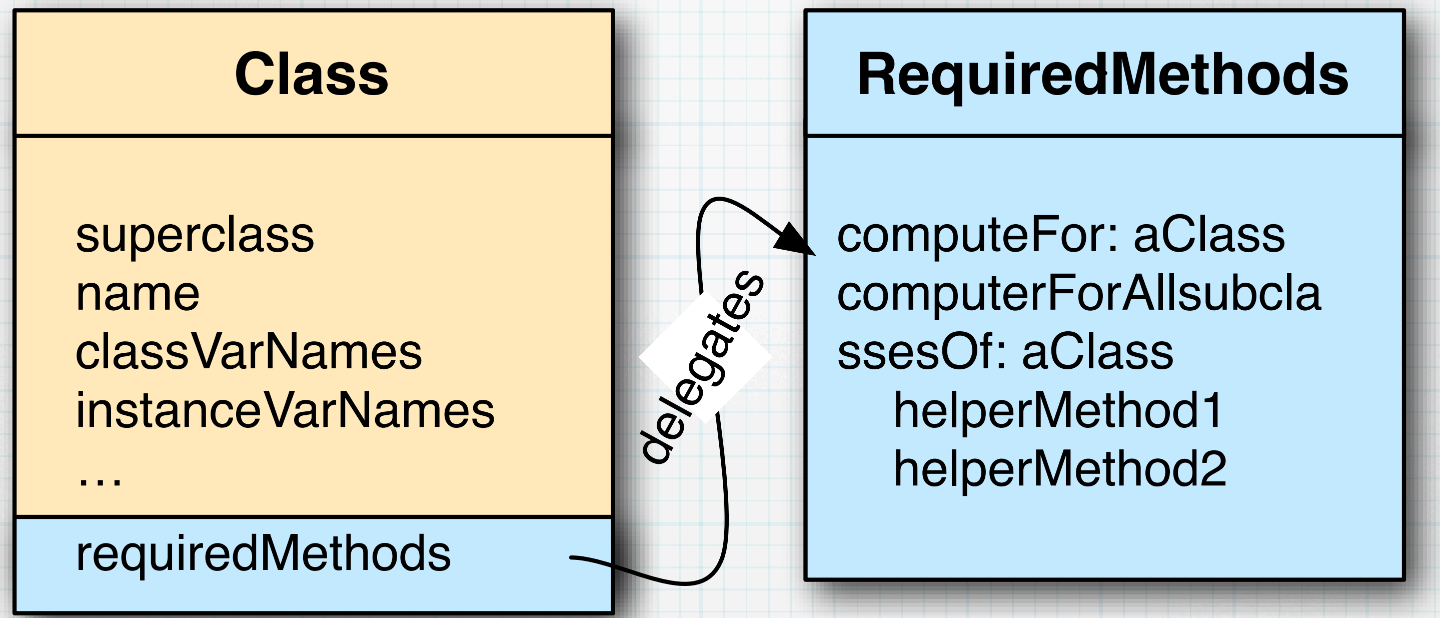
superclass
name
classVarNames
instanceVarNames
…

requiredMethods
helperMethod1
helperMethod2

(b) extend model by adding the whole implementation of the new property to an appropriate class in the model



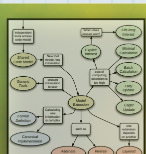
Option c: Model Extension



(c) Model Extension: put interface to the new property in the appropriate class in the model, but put calculation of new property in its own class

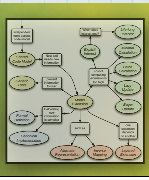
Consequences

Tool and analysis

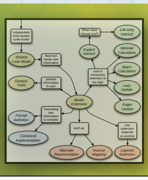


Consequences

~~Tool and analysis~~



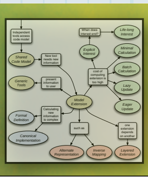
Consequences



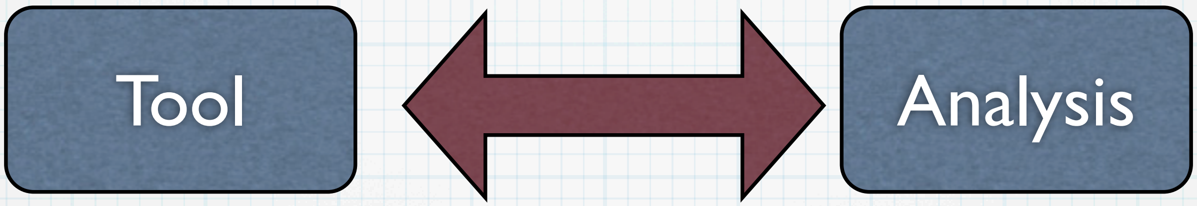
Consequences

Tool

Analysis



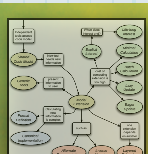
Consequences



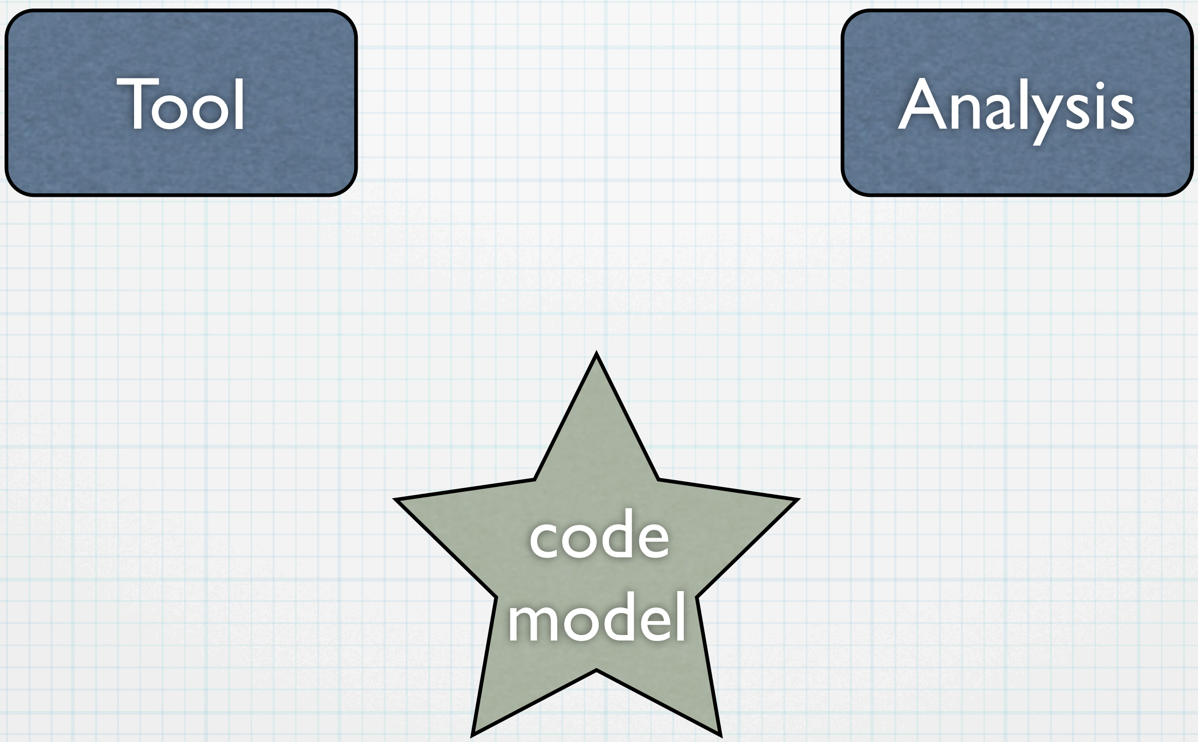
Consequences

Tool

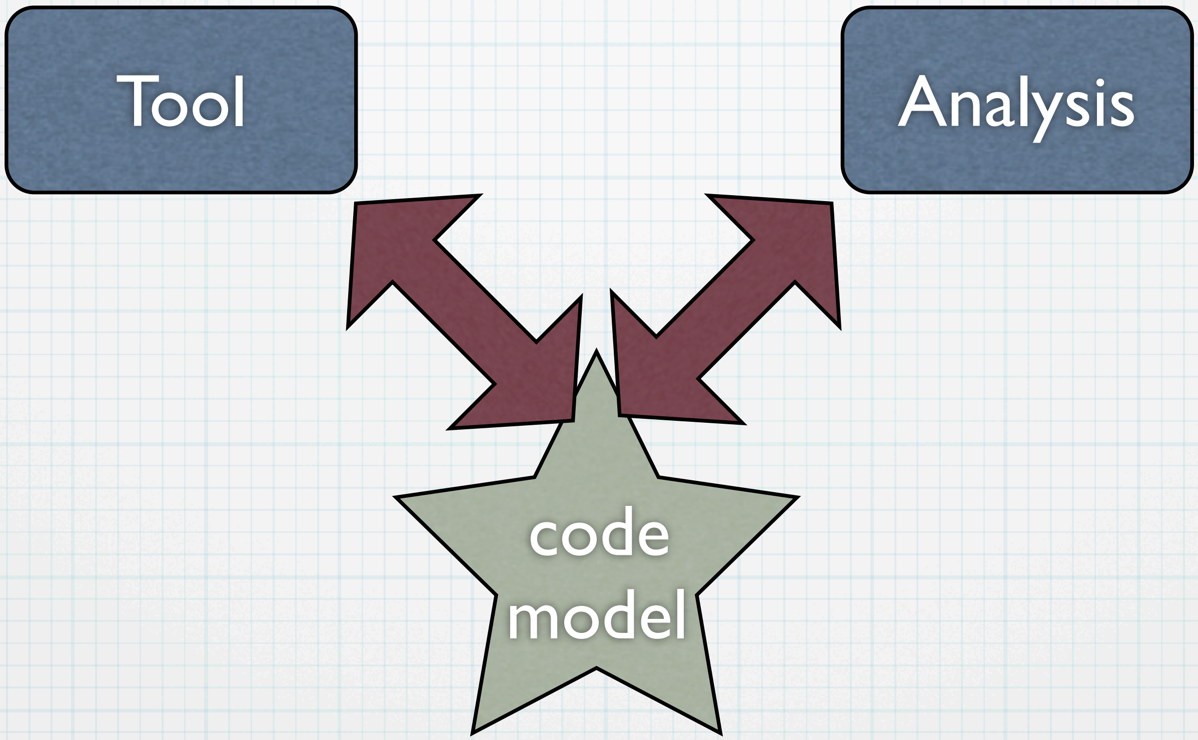
Analysis



Consequences



Consequences

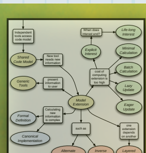


Generic Tools

- Tools that allow the display of arbitrary metrics and predicates
  - ```
    (MyPackage allClasses) do: [ :each |
        self deny: (each subclasses isEmpty and:
                        [each isAbstract])]
    ```
- Starbrowser
    ```

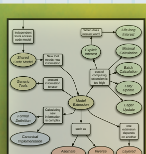
Alternative Representation

- Shared code model represents everything…
  - but not necessarily in a way that helps!
- Define a better representation as a Model Extension
  - efficiency may demand caching
- The alternative representation can be shared by multiple tools



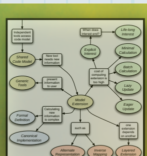
Example:

- Bytecode representation of methods in Smalltalk
- The “primary” representation for methods is text
  - This obviously doesn't work well for execution
- Bytecodes are cached, and recomputed eagerly whenever the text is changed.



Inverse Mapping

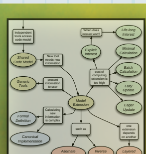
- Shared code model & alternative representation provide a set of navigation links,
  - e.g., superclass
- How can we navigate in the opposite direction?
  - searching is expensive
- Provide an Inverse Mapping as a Model Extension



Portland State UNIVERSITY

Layered Extension

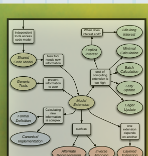
- You have a complex model extension
- How do you implement it efficiently, yet readably?
  - it's inputs may be expensive to compute
  - … and might be valuable in themselves
- Define each complex model extension in terms of lower-level, simpler model extensions
  - higher-level extensions express *Explicit Interest* in the lower levels



Portland State UNIVERSITY

Explicit Interest

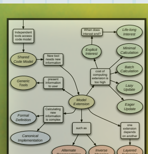
- You have a model extension that depends on heavyweight calculations
- Clearly can't re-calculate it globally at frequent intervals
- Could calculate it only on demand
  - but then *Observer* doesn't work on the model extension
  - ⇒ model extension behaves differently from the core model
- Could cache it
  - but not over the whole code model



Portland State UNIVERSITY

Solution:

- Add a new interface that allows clients of a property to declare interest in it explicitly, for some part of the code model
  - implementation can assume that it won't be asked for the property on "uninteresting" code elements
  - property is cached for "interesting" code elements
- Assumption: only a small part of the program is "interesting" at any one time

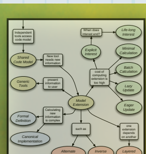


Explicit Interest vs. Observer

- Duals?
  - Explicit Interest: no concern with who expresses interest, only in what is interesting
  - Observer: no concern with what is being observed, only with who is observing
- Explicit Interest gives the model more choices — non-architectural
- Observer gives the model more responsibilities — architectural

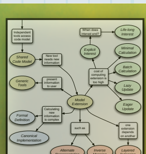
Life-long Interest

- an Explicit Interest:
  - declared when a tool object comes into existence
  - and retracted when the object is garbage-collected.



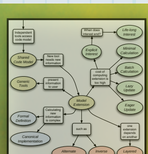
Minimal Calculation

- You are maintaining a cache over a model extension in which clients express explicit interest
- When the model changes, how do you avoid unnecessary re-computation of the cache?
- Update only those elements of the extension that are both interesting and dependent on the chnages



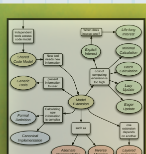
Eager Update

- You have defined a Model Extension or a Layered Extension on a Shared Code Model
- When do you re-calculate the Extension?
- If re-calculation is local and fast, then
  - update the Model Extension eagerly, as soon you are notified of a change in the model or lower-level extension
- Simple, supports Observer



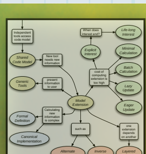
Lazy Update

- You have defined a Model Extension or a Layered Extension on a Shared Code Model
  - You Extension depends on multiple properties, and caches them
  - You are an observer of them all, but notification of changes arrive in an indeterminate order
- In what order should the cached properties be re-calculated?



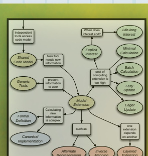
Solution:

- If re-calculation is local and fast, then
  - update the Model Extension lazily.
  - when you receive a change notification, *invalidate* the appropriate cache, but don't recompute it.
  - recompute the cache as a side-effect of answering client queries.
- Does not support Observer on the extension



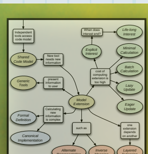
Batch Calculation

- The calculation of Model Extension depends on non-local properties of the model.
- How can you get "economies of scale" in re-computing it?
  - neither lazy nor eager update help!



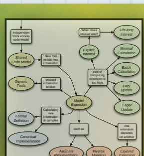
Solution:

- Extension tracks changes to the model, but defers acting on them.
- When calculation of the extension eventually happens, all changes are dealt with at once, and the property is calculated for all *interesting* code elements.
- When is "eventually"?
  - Lazy update can tell us
    - the combination is *partially* lazy.



Canonical Implementation

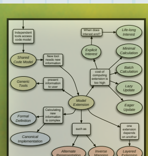
- You have a Model Extension
  - it's complex
  - the simple implementation is too slow
- How do you improve performance while remaining confident of correctness?



Portland State UNIVERSITY

Solution:

- Encapsulate the simple implementation as the “Canonical Implementation”
- Create an independent, efficient implementation for client use
- Write tests that compare the Efficient Implementation with the Canonical Implementation



Formal Definition

- You have thought of a useful, but complex property
- How can you decide:
  - whether it is well-defined in all cases
  - what implementation-shortcuts are possible?
  - when it needs to be re-calculated
- Define the property formally using mathematical language

Conclusion

- Canonical Representation lives
  - 2GB of main memory certainly help
  - but we can't get a canonical representation by calling a standards meeting!
  - Any representation that hopes to be—and stay—canonical must be extensible
- These patterns help us to build efficient extensible representations

Open Questions

- How universal are these patterns?
- Certainly there are more to be discovered…
- but are these useful beyond a particular implementation (ours) of a particular tool (requirmenst browser) in a particular language (Smalltalk)
- Over to you!