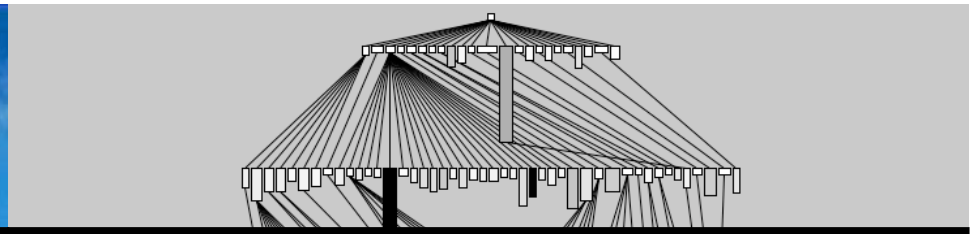
Software Analysis & Visualization

Harald Gall
Institut für Informatik
Universität Zürich
http://seal.ifi.uzh.ch

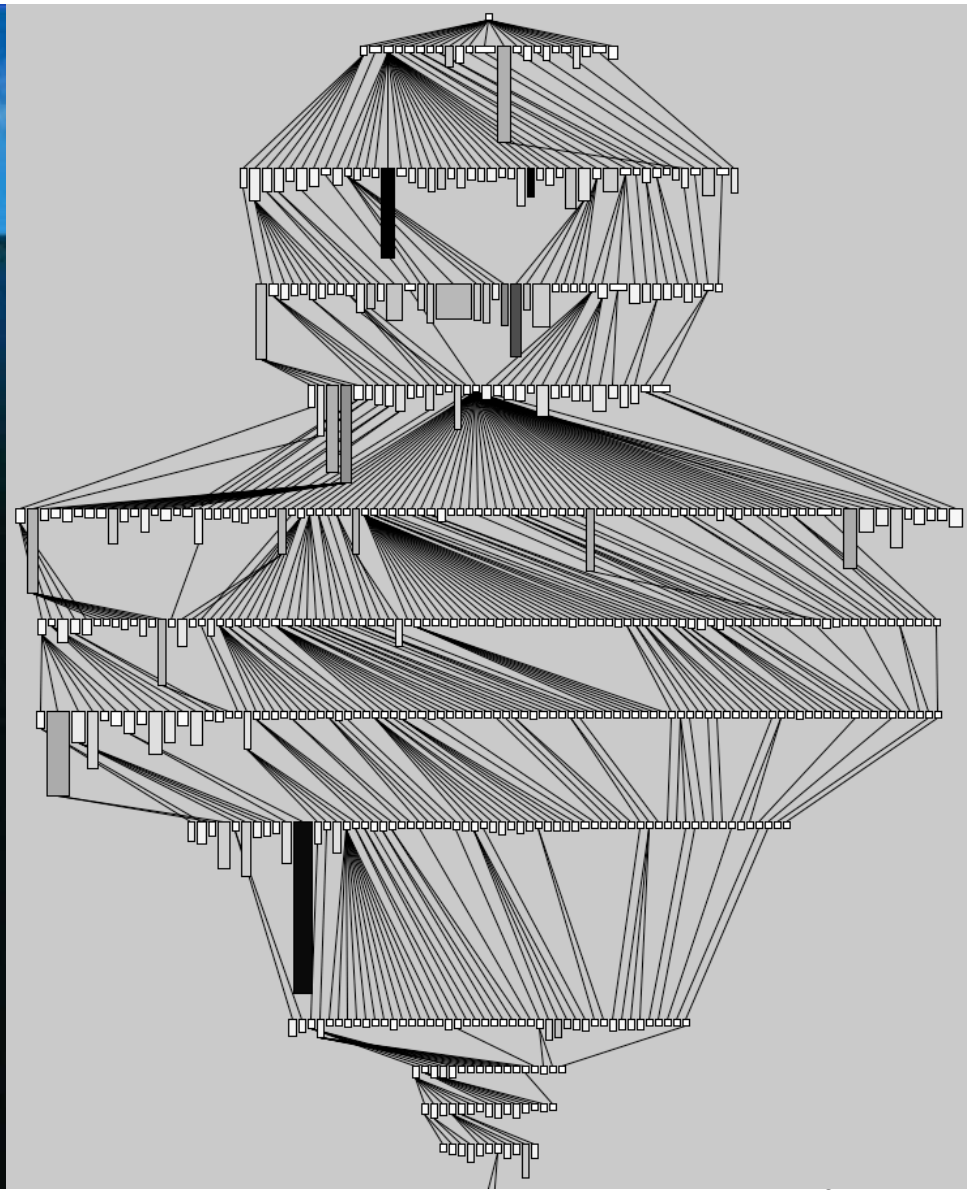
Universität Zürich

s.e.a.l.
software evolution & architecture lab

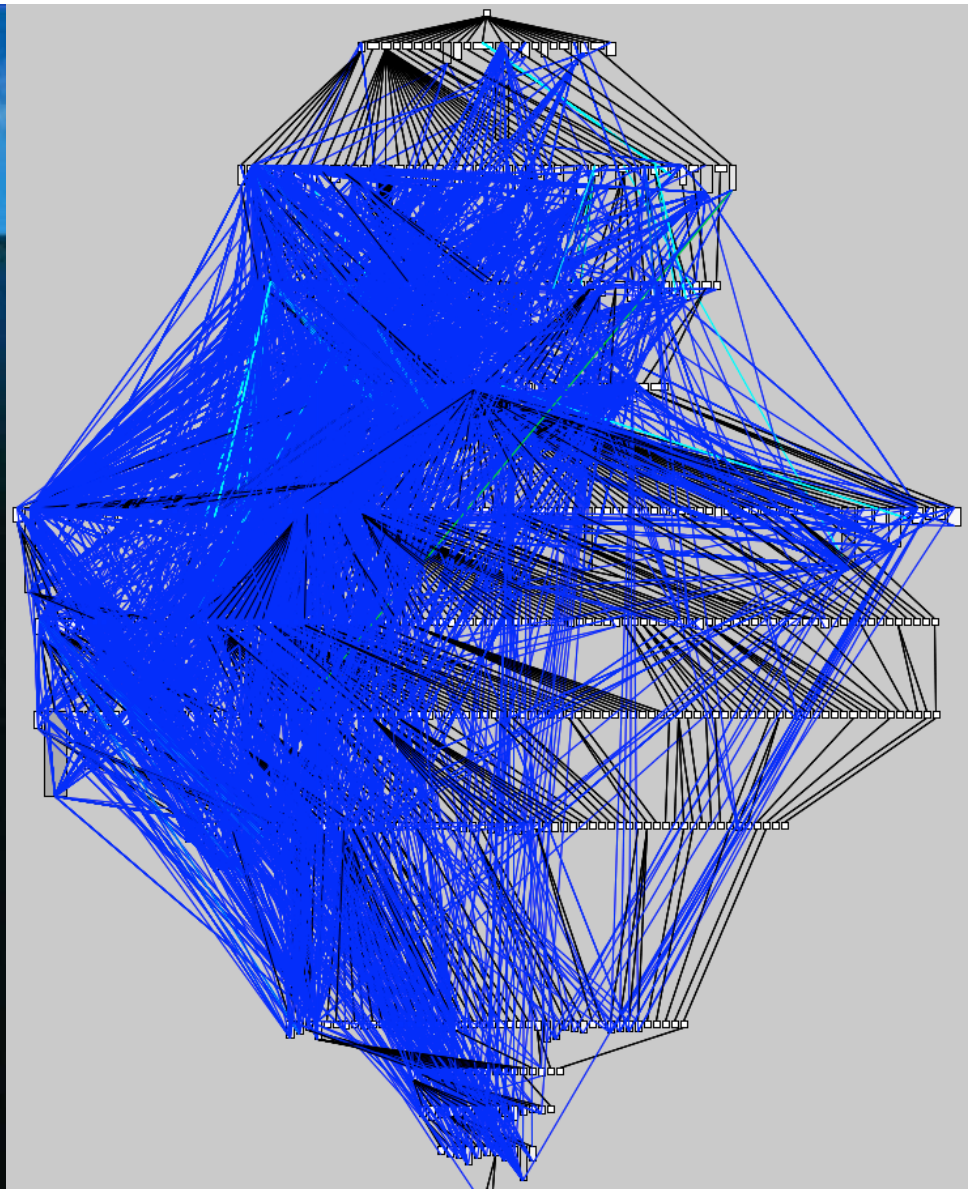
…software is intangible, having no physical shape or size…



…software is intangible, having no physical shape or size…



…software is intangible, having no physical shape or size…



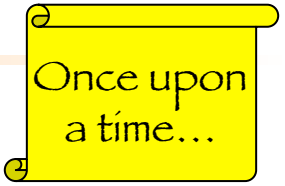
Software Visualization in Context

- There are many good-looking visualization techniques, but..when it comes to software maintenance & evolution, there are several problems:
  - Scalability
  - Information Retrieval
  - What to visualize
  - How to visualize
  - Reengineering context constraints
    - Limited time
    - Limited resources

Software Visualization - Outline

- Introduction
- Software Visualization in a Reengineering Context
- Static Code Visualization
  - Examples
- Dynamic Code Visualization
  - Examples
- Lightweight Approaches
  - Combining Metrics and Visualization
  - Demonstration
- Conclusion

Prologue



- Reverse engineer 1.2 MLOC C++ system of ca. 2300 classes
- * 2 = 2'400'000 seconds
- / 3600 = 667 hours / 8 = 83 days / 5 = 16 weeks & 3 days
- ~ 4 months to read the system
- Questions:
  - What is the size and the overall structure of the system?
  - What is the internal structure of the system and its elements?
  - How did the software system become like that?

Introduction

- Visualization
  - Information Visualization
- Software Visualization
  - Algorithm Visualization
  - Program Visualization
    - Static Code Visualization
    - Dynamic Code Visualization
- The overall goal is to *reduce complexity*

Software Visualization

"Software Visualization is the use of the crafts of typography, graphic design, animation, and cinematography with modern human-computer interaction and computer graphics technology to facilitate both the human understanding and effective use of computer software."

Price, Baecker and Small, "Introduction to Software Visualization"

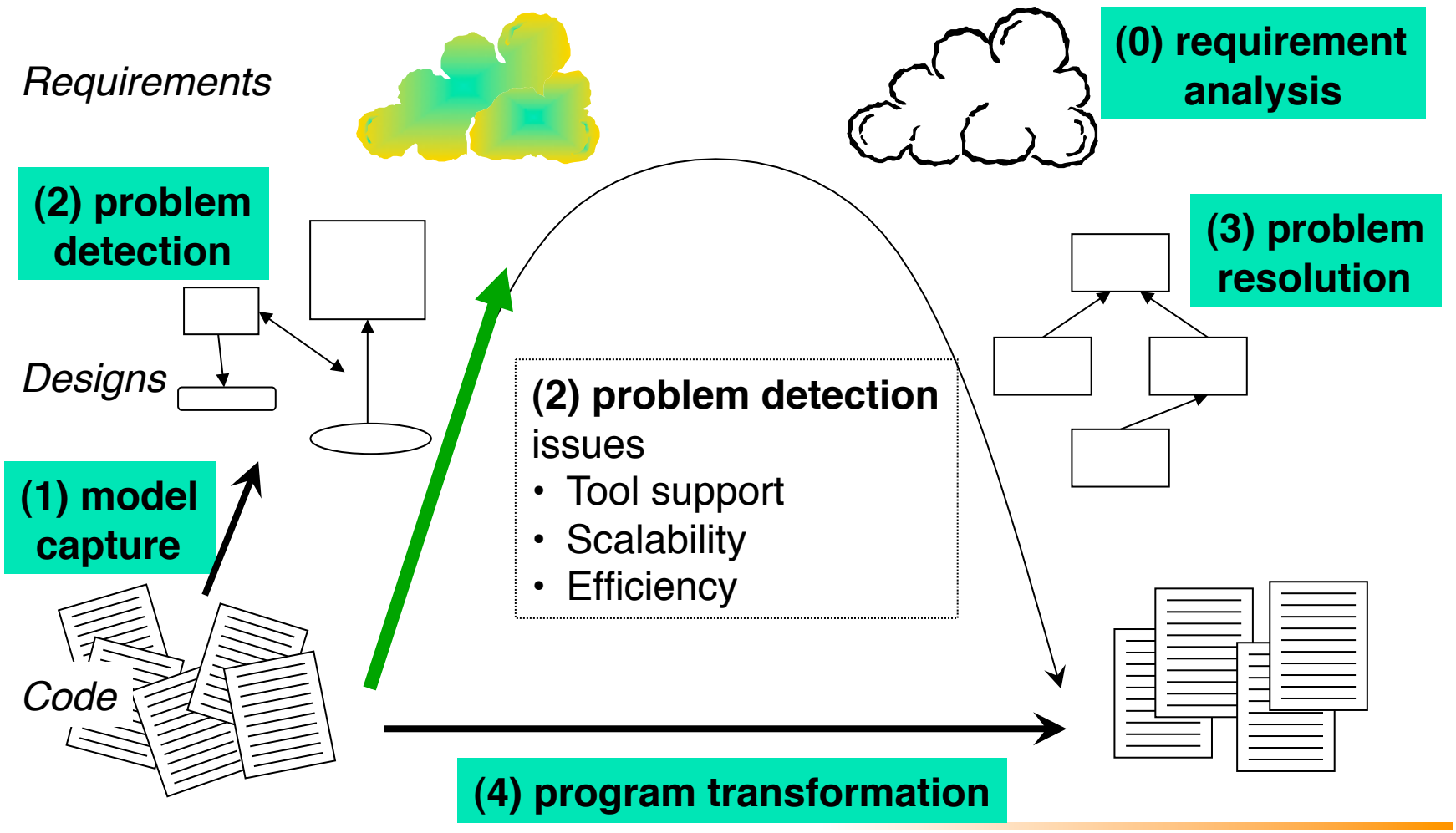
- 2 main fields:
  - Algorithm Visualization
  - Program Visualization

Conceptual Problem

"Software is intangible, having no physical shape or size. Software visualization tools use graphical techniques to make software visible by displaying programs, program artifacts and program behavior."

Thomas Ball

The Reengineering Life-cycle



Program Visualization

"Program visualization is the visualization of the actual program code or data structures in either static or dynamic form"

[Price, Baecker and Small]

- Static code visualization
- Dynamic code visualization
- Generate different views of a system and infer knowledge based on the views
- Complex problem domain (current research area)
  - Efficient space use, edge crossing problem, layout problem, focus, HCI issues, GUI issues, …
  - Lack of conventions (colors, symbols, interpretation, …)

Program Visualization II

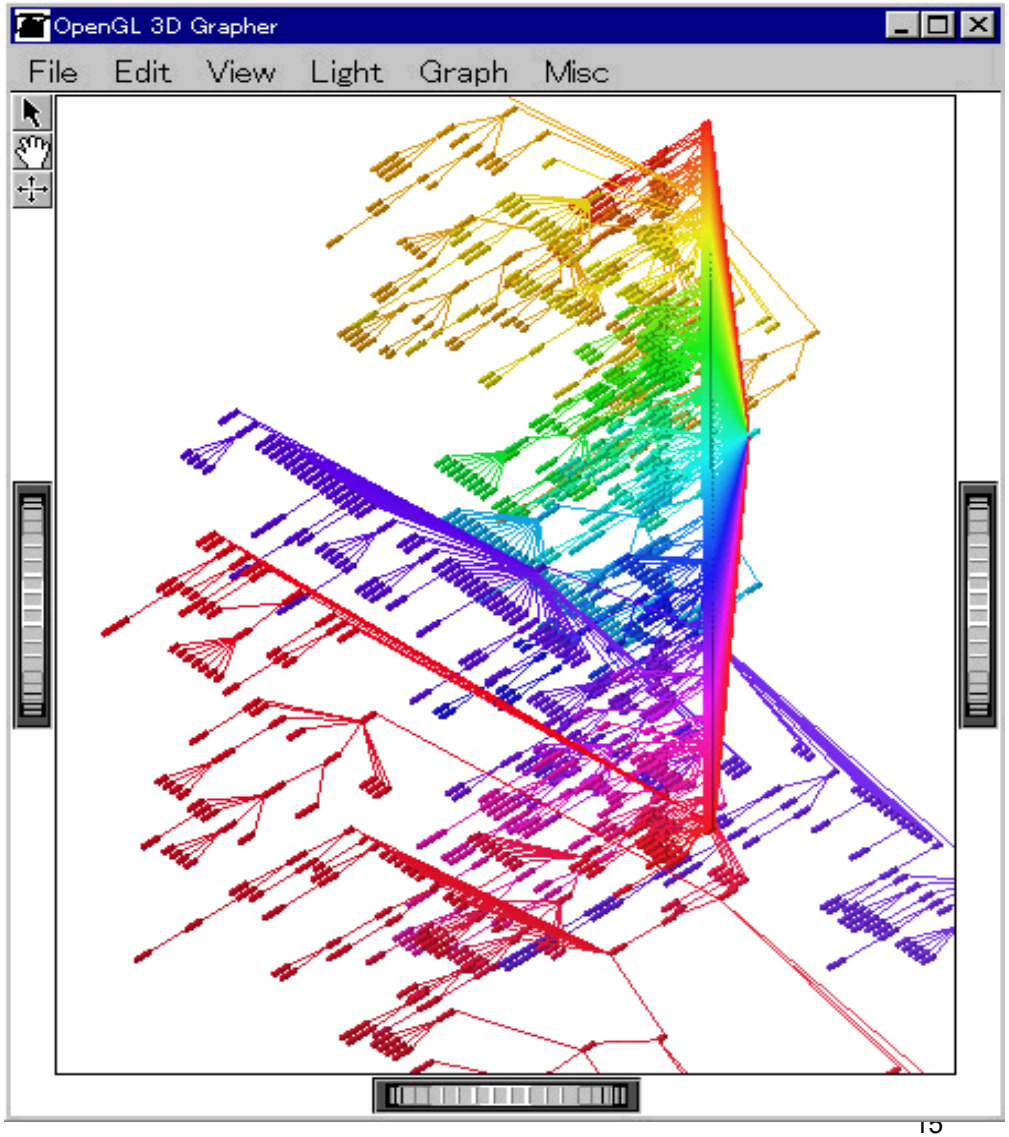
- Level of granularity?
  - Complete systems, subsystems, modules, classes, hierarchies,...
- When to apply?
  - First contact with an unknown system
  - Known/unknown parts?
  - Forward engineering?
- Methodology?

Static Code Visualization

- The Visualization of information that can be extracted from the static structure of a software system
- Depends on the programming language and paradigm:
  - Object-Oriented PL:
    - classes, methods, attributes, inheritance, …
  - Procedural PL:
    - procedures, invocations, …
  - Functional PL:
    - functions, function calls, …

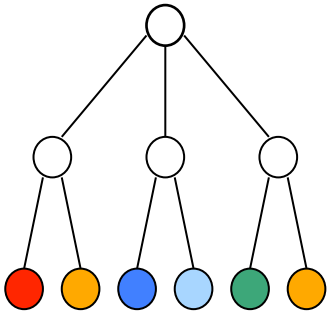
Example 1: Class Hierarchies

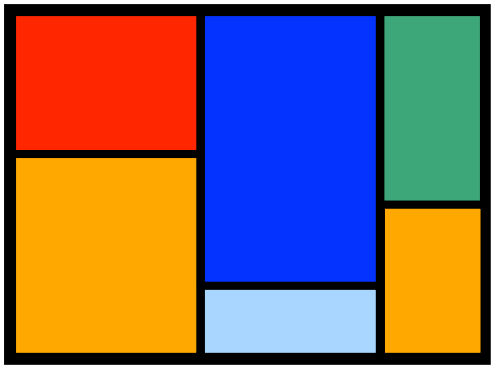
- Jun/OpenGL
- The Smalltalk Class Hierarchy
- Problems:
  - Colors are meaningless
  - Visual Overload
  - Navigation

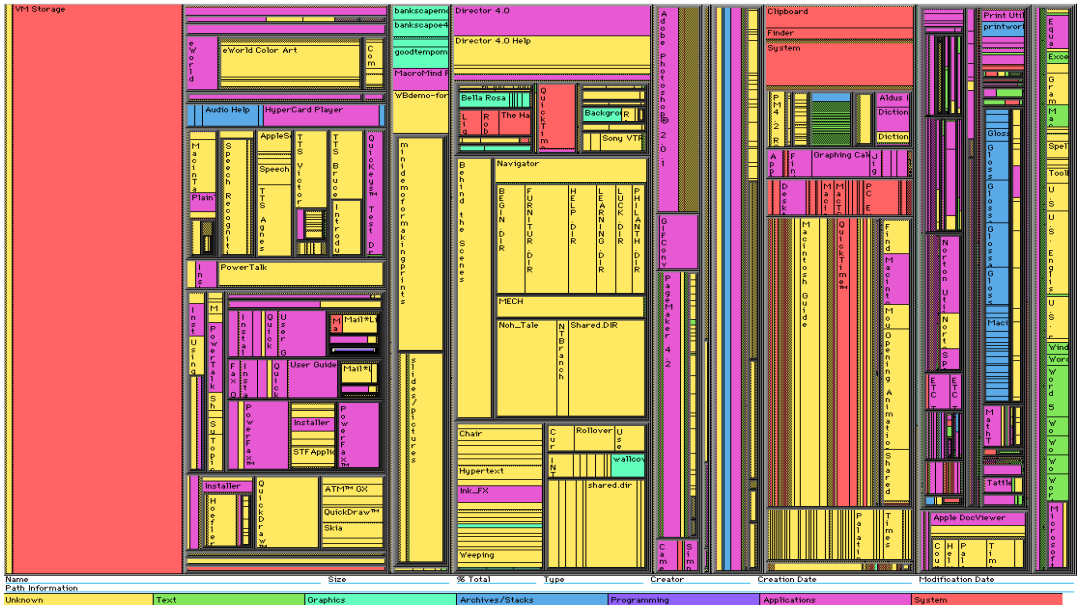


Example 2: Tree Maps

- Pros
  - 100% screen
  - Large data
  - Scales well
- Cons
  - Boundaries
  - Cluttered display
  - Interpretation
  - Leaves only
- Useful for the display of Hard Disks

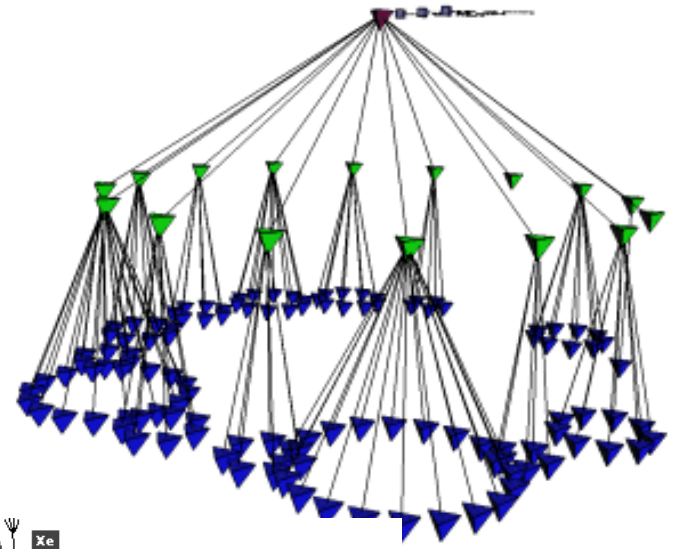


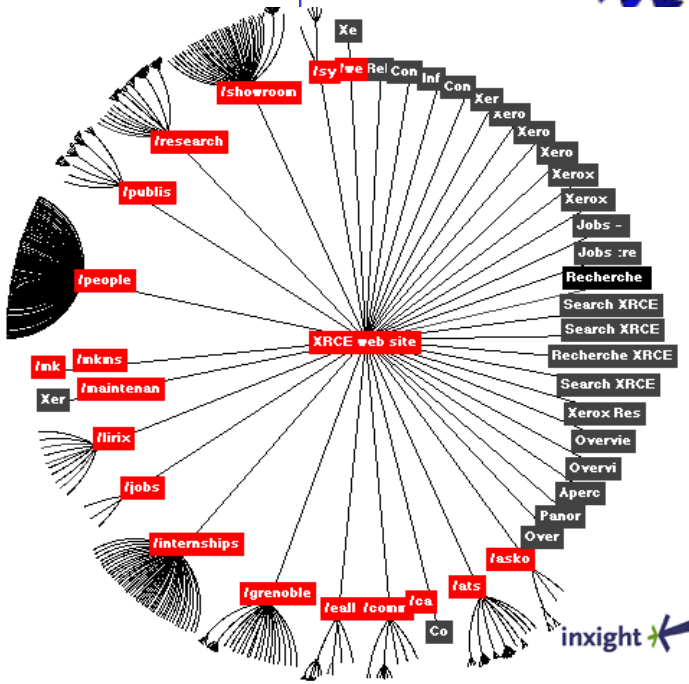




Examples 3 & 4

- Euclidean cones
  - Pros:
    - More info than 2D
  - Cons:
    - Lack of depth
    - Navigation
- Hyperbolic trees
  - Pros:
    - Good focus
    - Dynamic
  - Cons:
    - Copyright



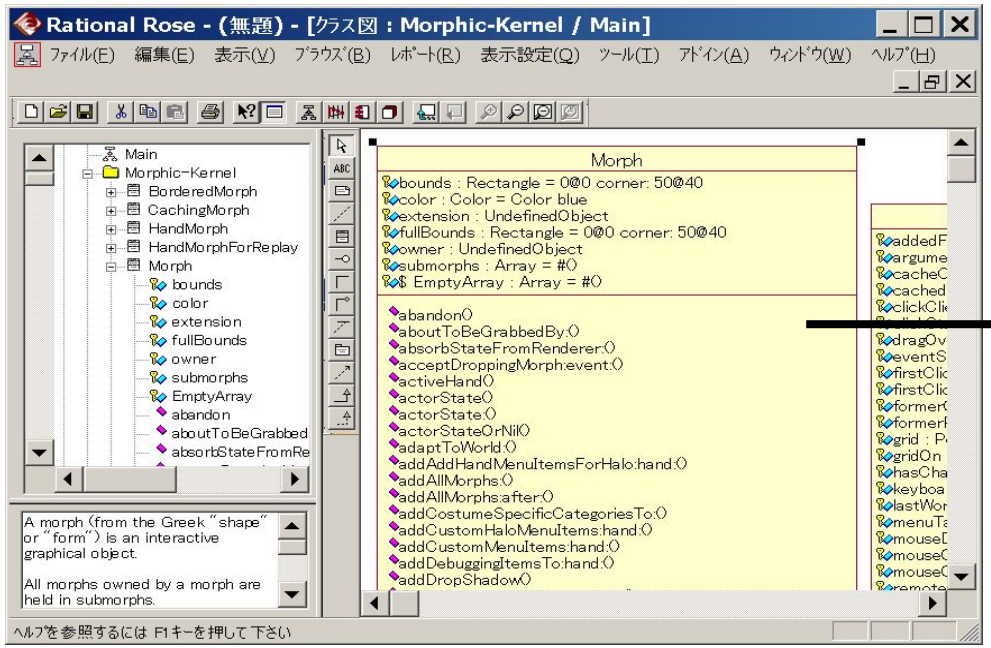


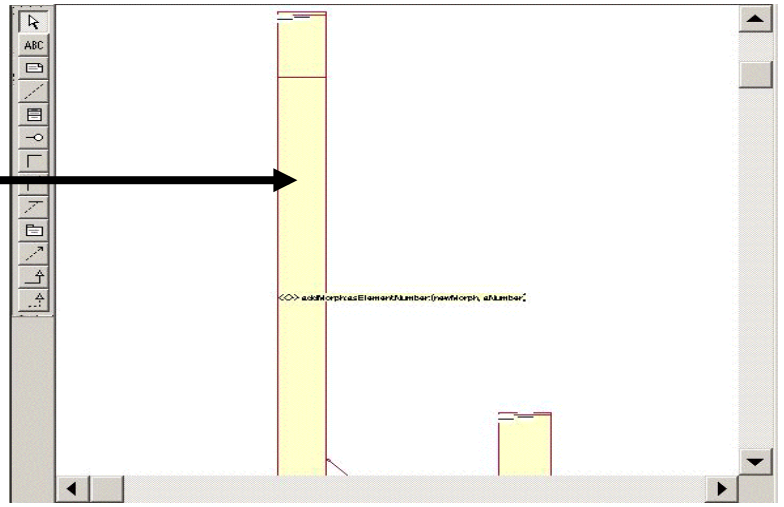
Class Diagram Approaches

- For example UML diagrams…
- Pros:
  - OO Concepts
  - Good for small parts
- Cons:
  - Lack of scalability
  - Require tool support
  - Requires mapping rules to reduce noise
  - Preconceived views

Example 5: UML and derivates

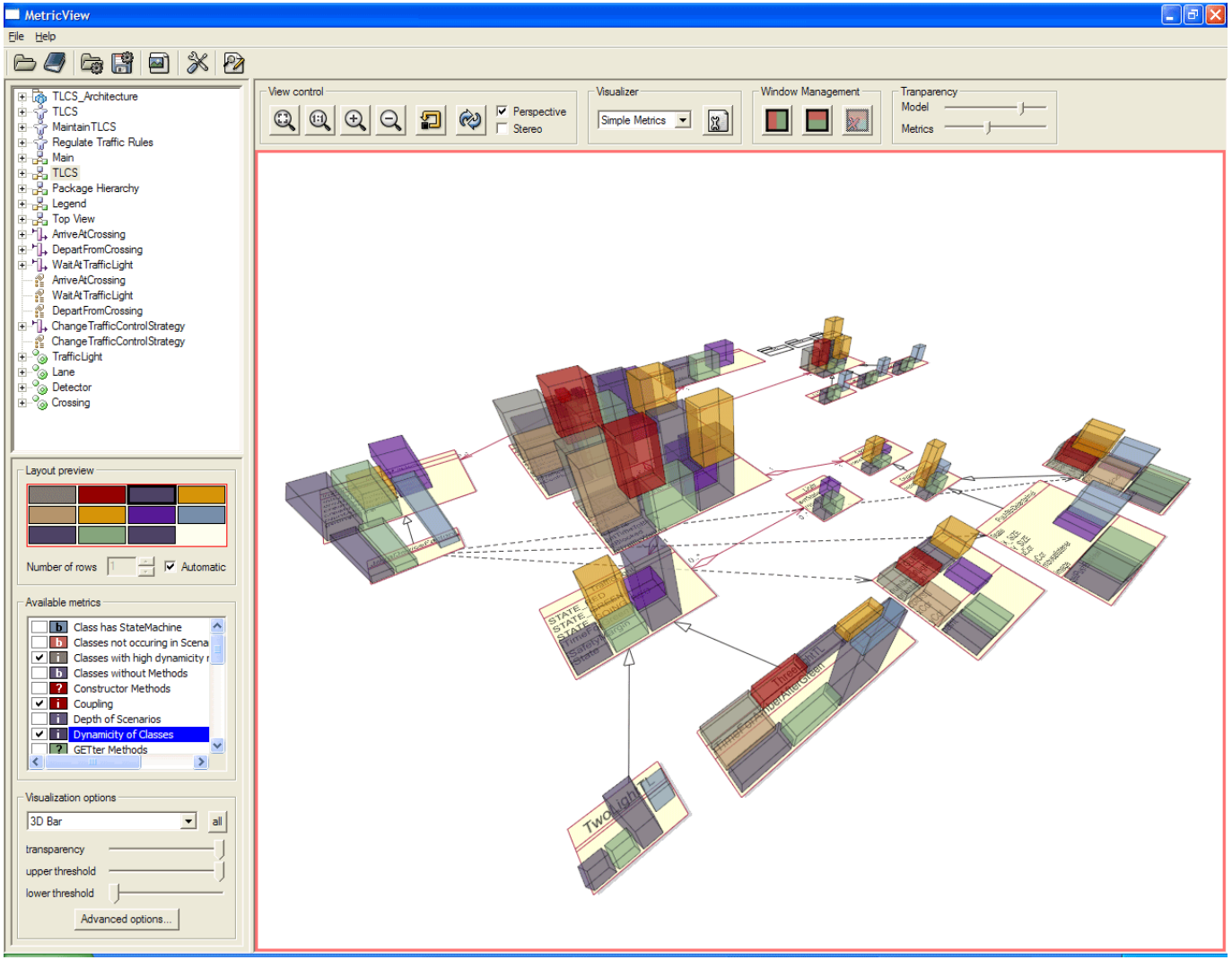
- Pros
  - OO concepts
  - Works very well for small parts
- Cons
  - Lack of scalability
  - Requires tool support
  - Requires mapping rules to reduce noise
  - Hardly extensible





Example 6: MetricView

- UML & 3D

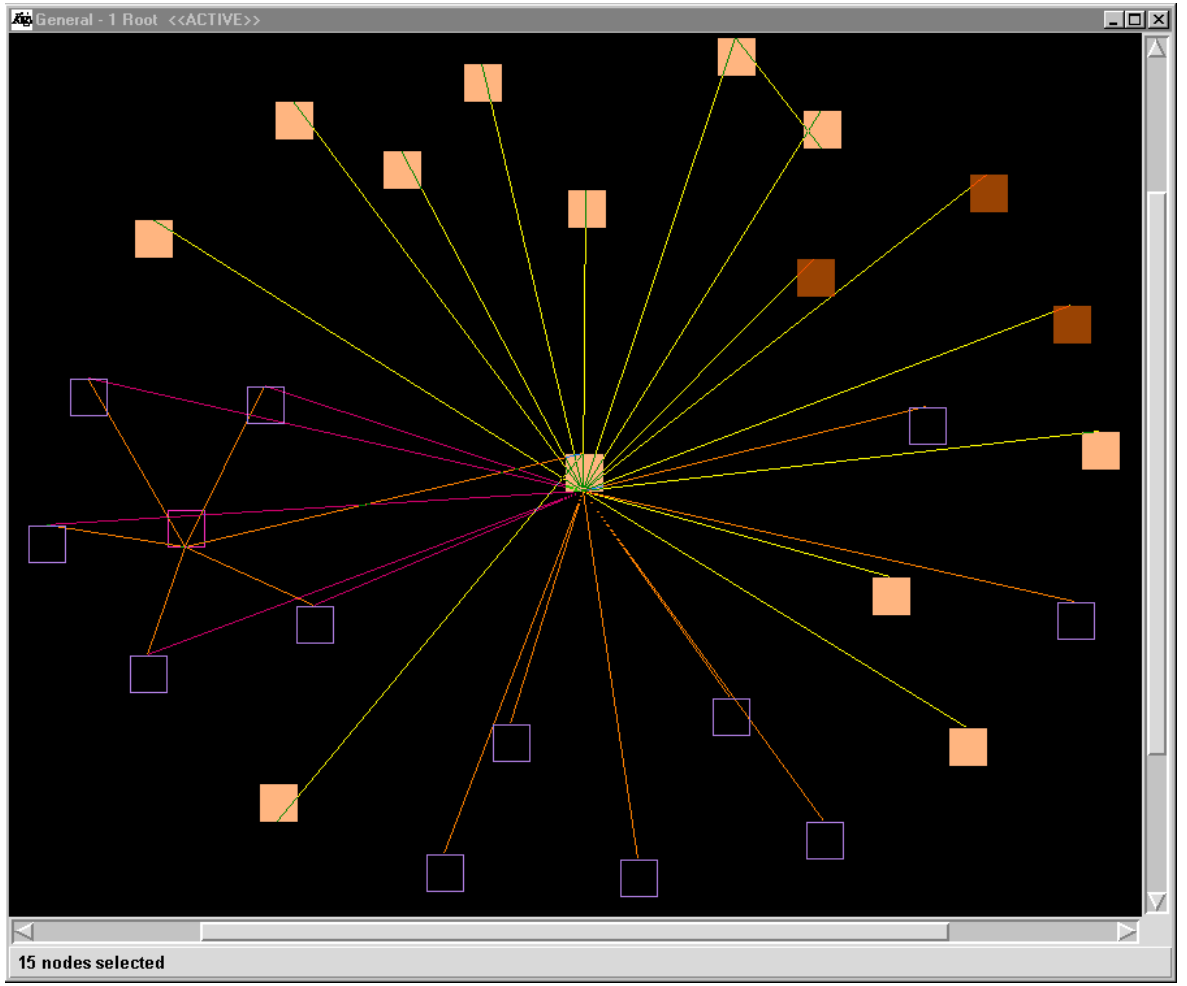


Example 7a: Rigi

- Scalability problem
- Entity-Relationship visualization
- Problems:
  - Filtering
  - Navigation

Example 7b: Rigi

- Entities can be grouped
- Pros:
  - Scales well
  - Applicable in other domains
- Cons:
  - Not enough code semantics



Evaluation

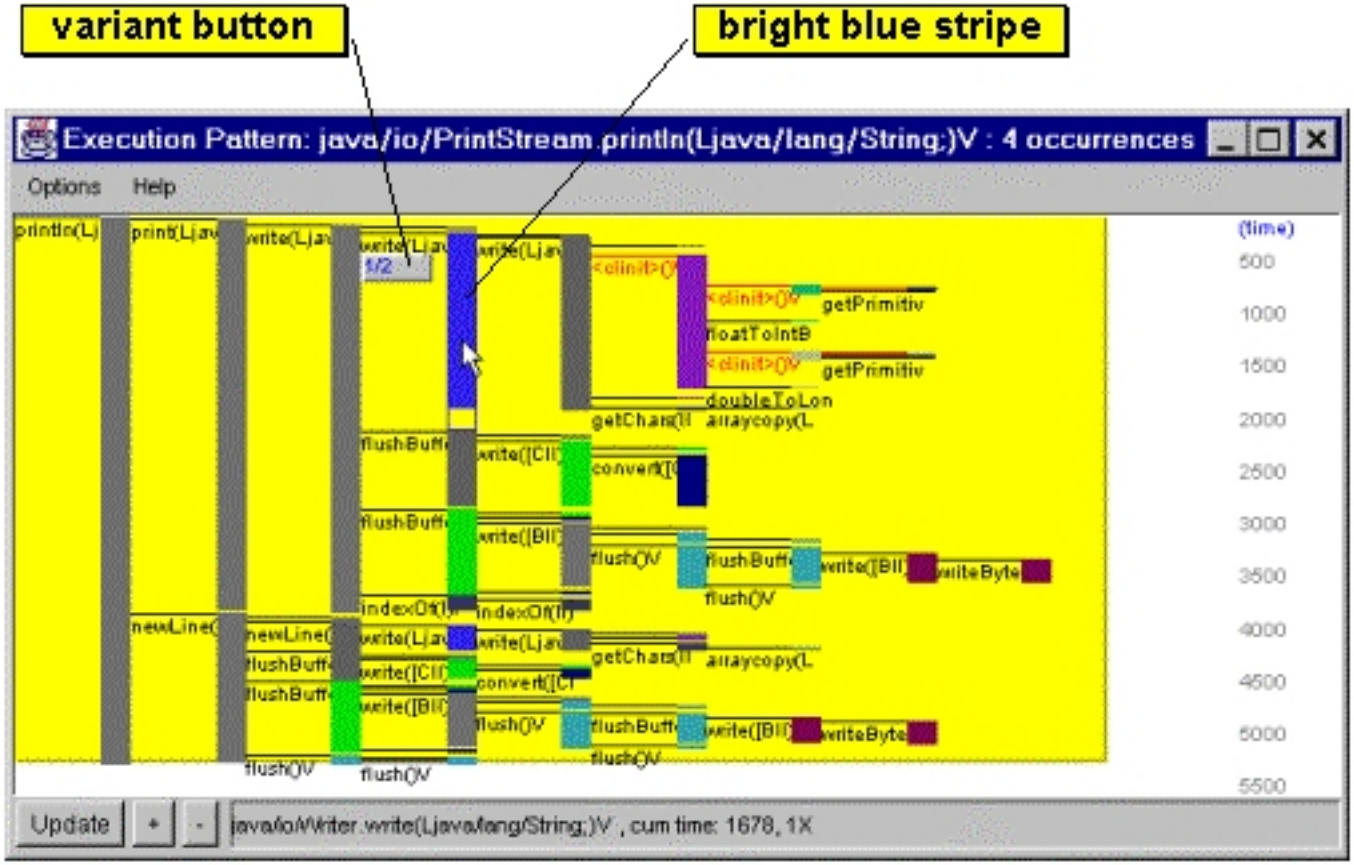
- Pros
  - Intuitive approaches
  - Aesthetically pleasing results
- Cons
  - Several approaches are orthogonal to each other
  - Too easy to produce meaningless results
  - Scaling up is sometimes possible, but at the expense of semantics

Dynamic Code Visualization

- Visualization of dynamic behavior of a software system
  - Code instrumentation
  - Trace collection
  - Trace evaluation
  - What to visualize
    - Execution trace
    - Memory consumption
    - Object interaction
    - …

Example 1: JInsight

- Visualization of execution trace



Example 2: Inter-class call matrix

- Simple
- Scales quite well
- Reproducible

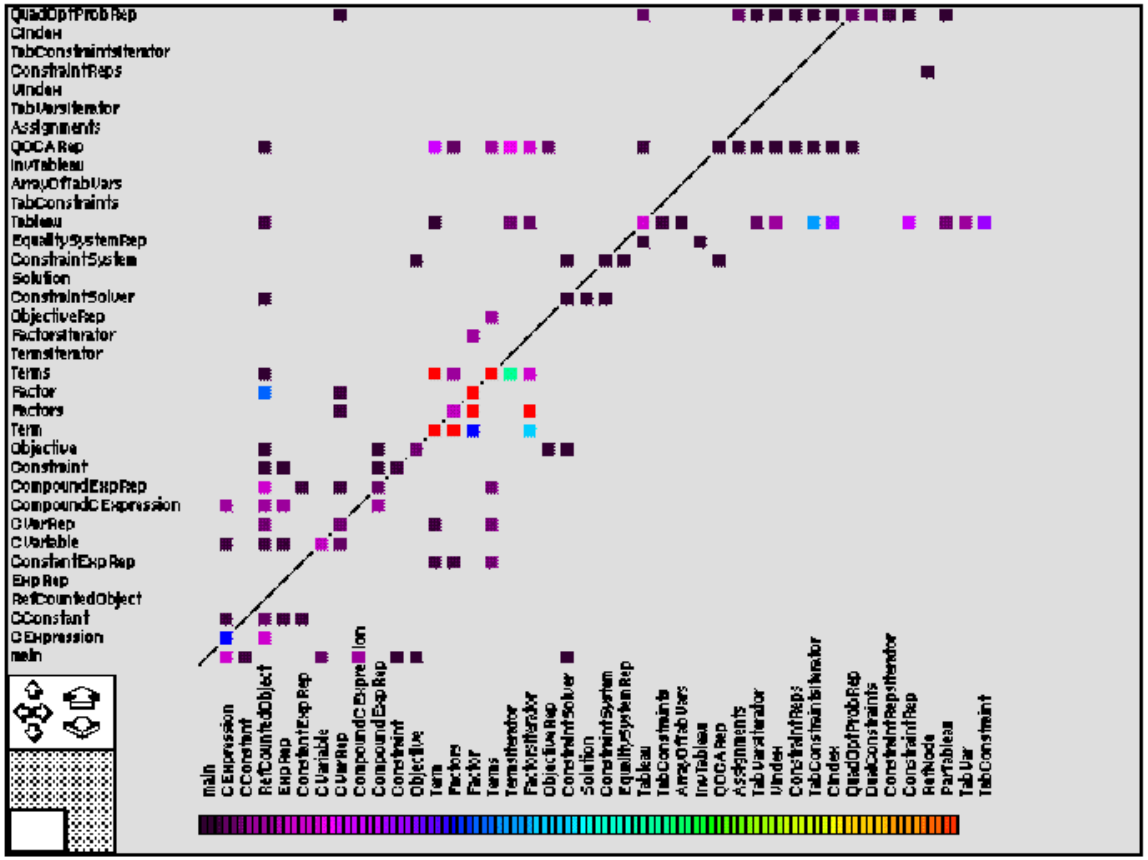
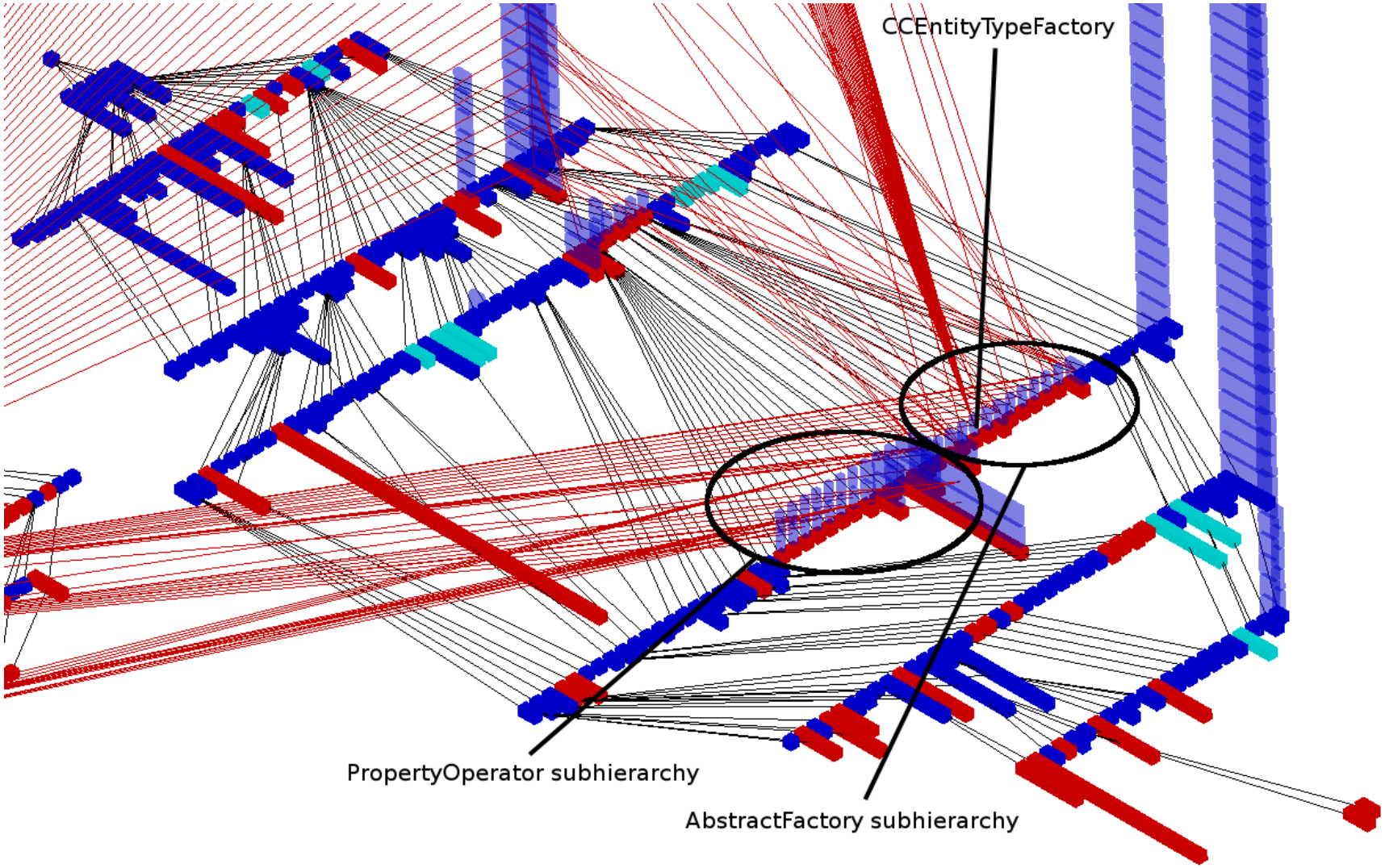


Figure 6: Inter-class call matrix

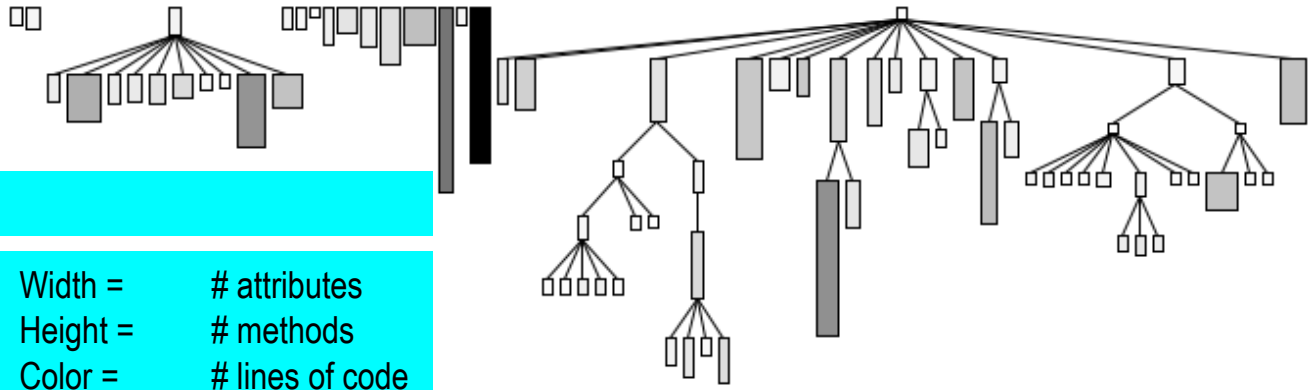
Example 3: TraceCrawler



Dynamic SV: Evaluation

- Code instrumentation problem
  - Logging, Extended VMs, Method Wrapping
- Scalability problem
  - Traces quickly become very big
- Completeness problem
  - Scenario driven
- Pros:
  - Good for fine-tuning, problem detection
- Cons:
  - Tool support *crucial*
  - Lack of abstraction without tool support

The Polymetric View - Example (II)



System Complexity View

Nodes = Classes	Width =	# attributes
Edges = Inheritance Relationships	Height =	# methods
	Color =	# lines of code

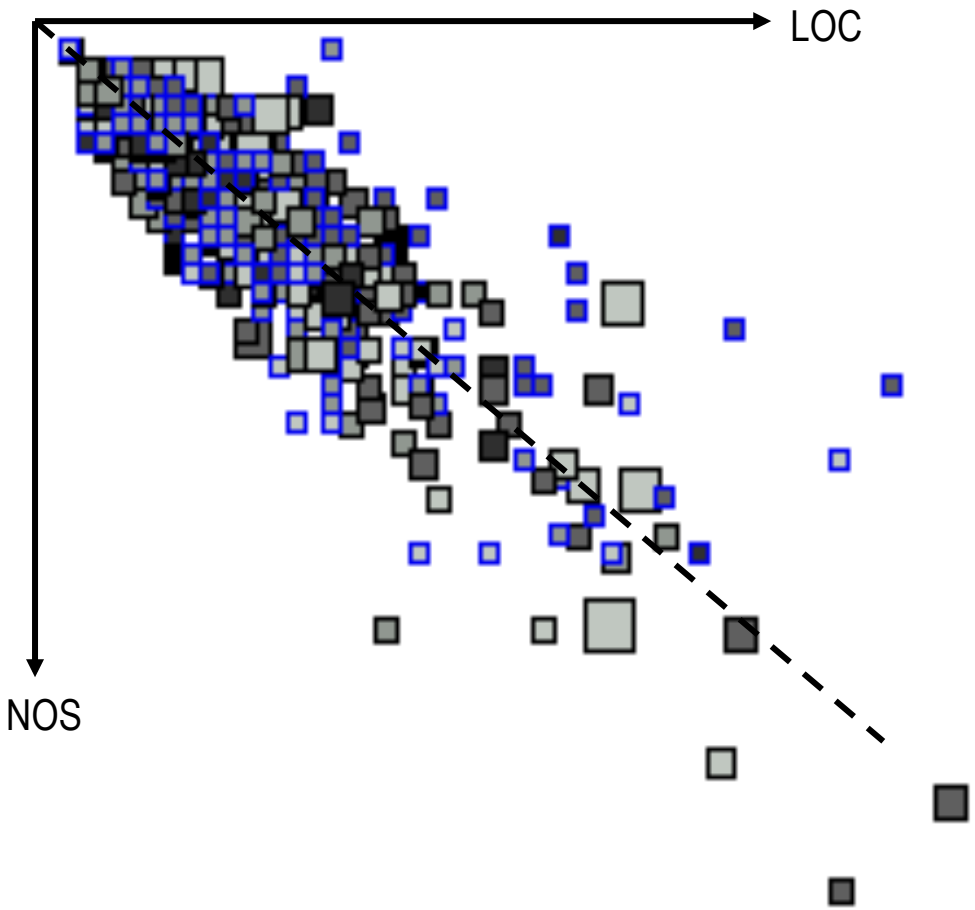
Reverse engineering goals

- Get an impression (build a first raw mental model) of the system, know the size, structure, and complexity of the system in terms of classes and inheritance hierarchies
- Locate important (domain model) hierarchies, see if there are any deep, nested hierarchies
- Locate large classes (standalone, within inheritance hierarchy), locate stateful classes and classes with behaviour

View-supported tasks

- Count the classes, look at the displayed nodes, count the hierarchies
- Search for node hierarchies, look at the size and shape of hierarchies, examine the structure of hierarchies
- Search big nodes, note their position, look for tall nodes, look for wide nodes, look for dark nodes, compare their size and shape, “read” their name => opportunistic code reading

Coarse-grained Polymetric Views



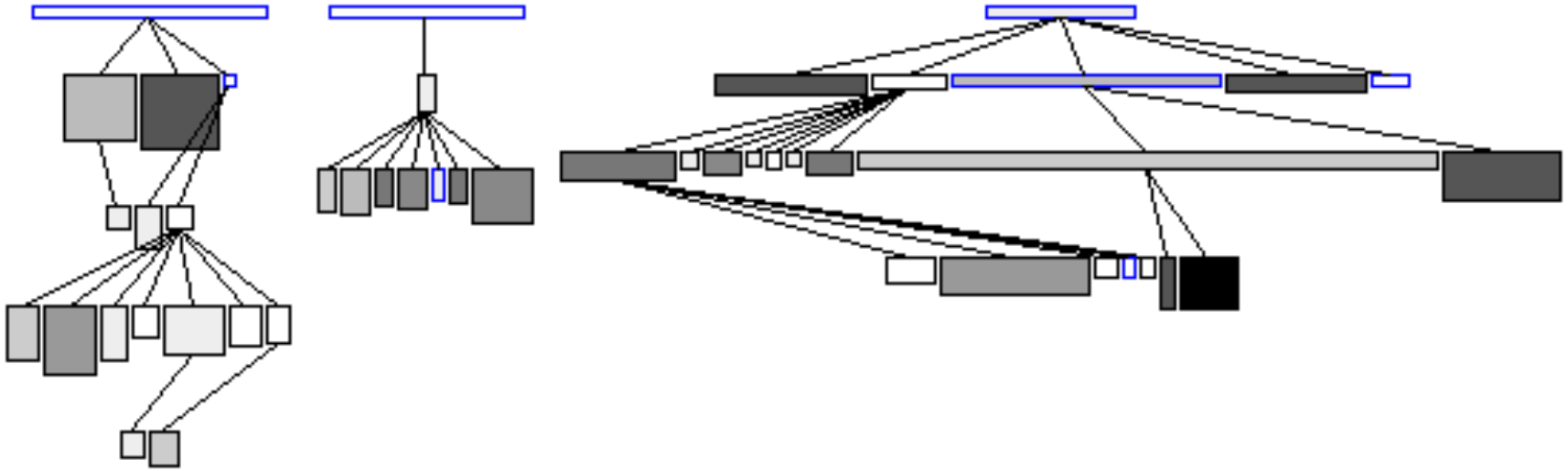
Method Efficiency Correlation View

Nodes:	Methods
Edges:	-
Size:	Number of method parameters
Position X:	Number of lines of code
Position Y:	Number of statements

Goals:

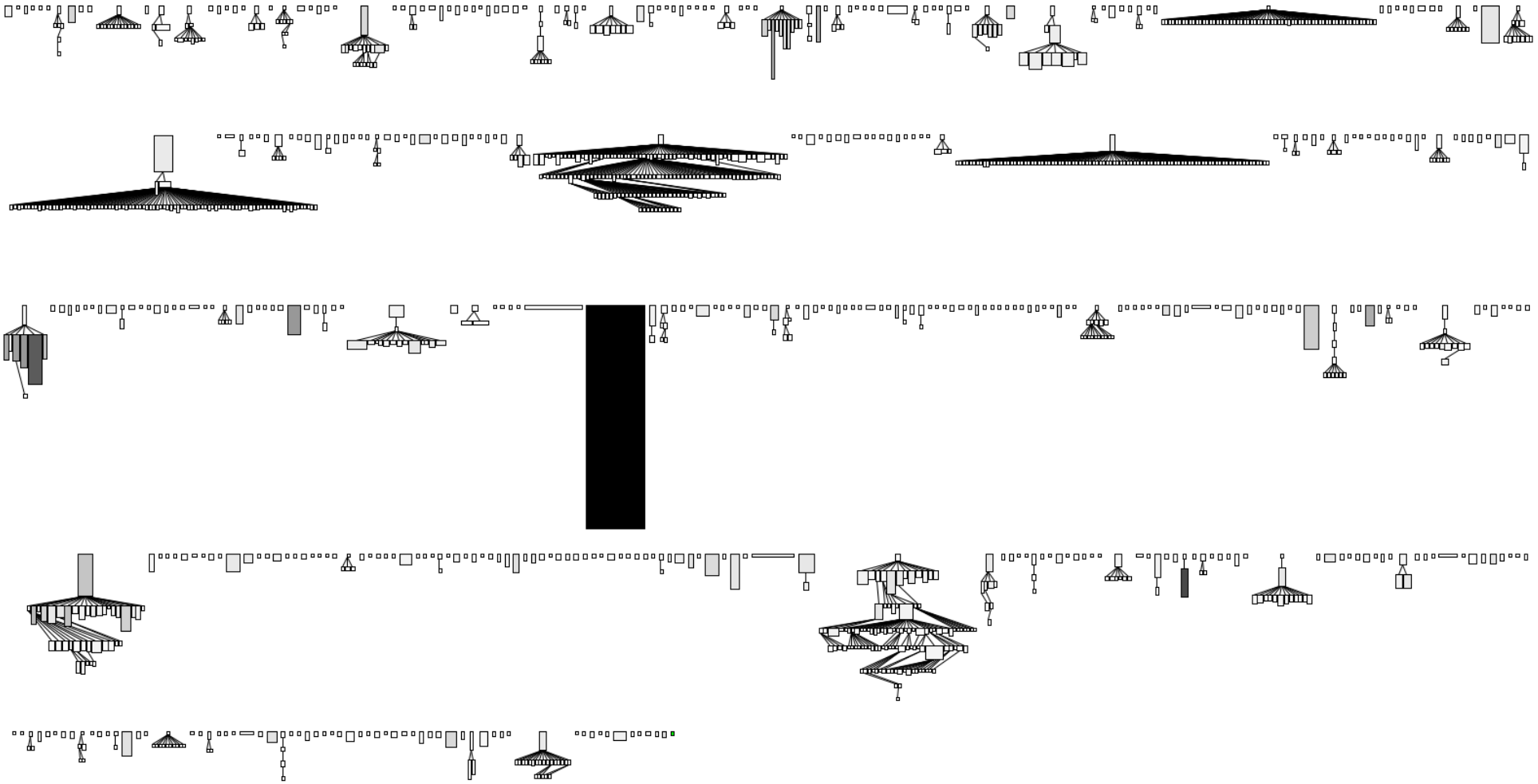
- Detect overly long methods
- Detect “dead” code
- Detect badly formatted methods
- Get an impression of the system in terms of coding style
- Know the size of the system in # methods

Inheritance Classification View

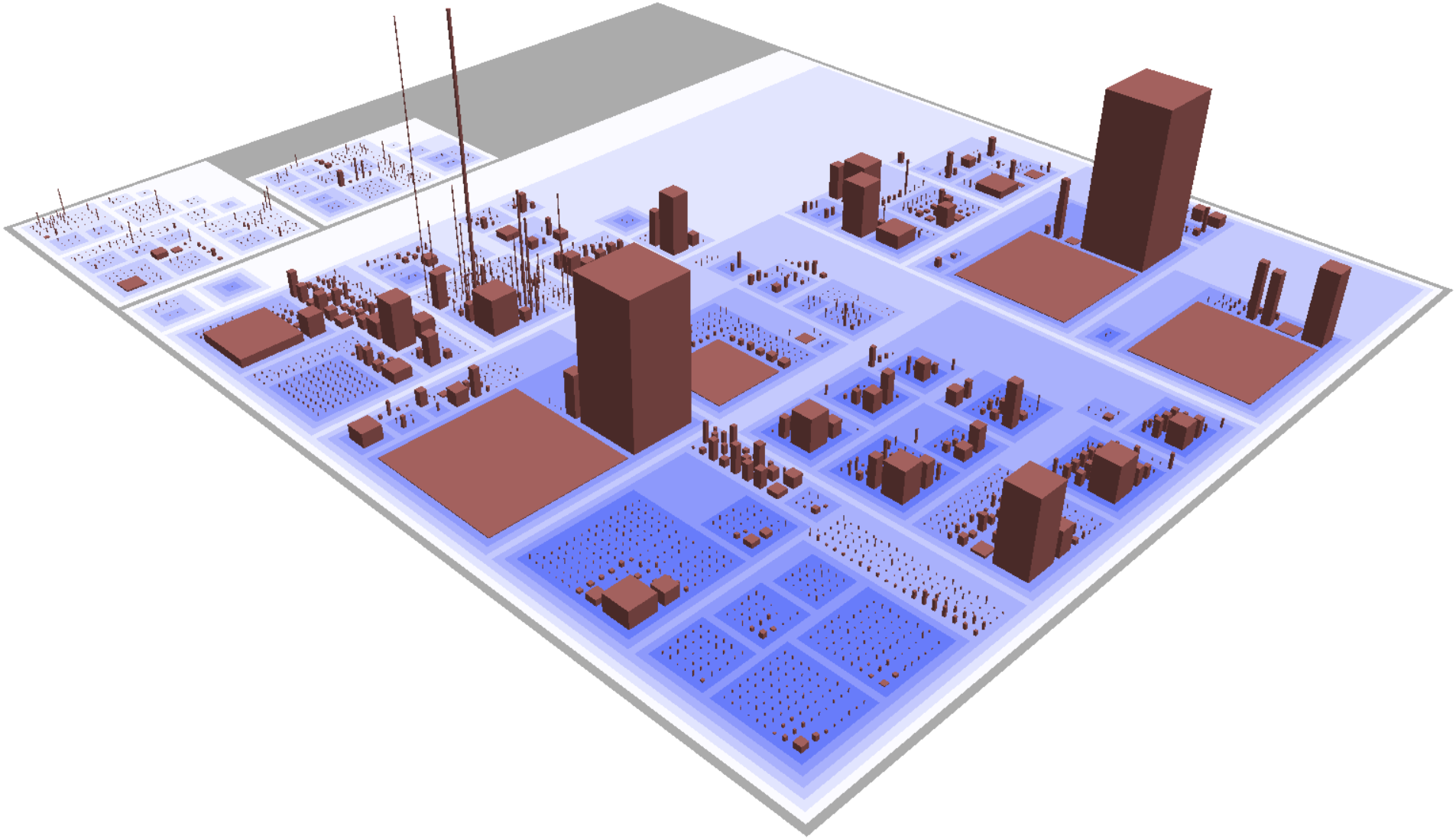


Boxes:	Classes
Edges:	Inheritance
Width:	Number of Methods Added
Height:	Number of Methods Overridden
Color:	Number of Method Extended

Polymetric View Example: ArgoUML



ArgoUML City

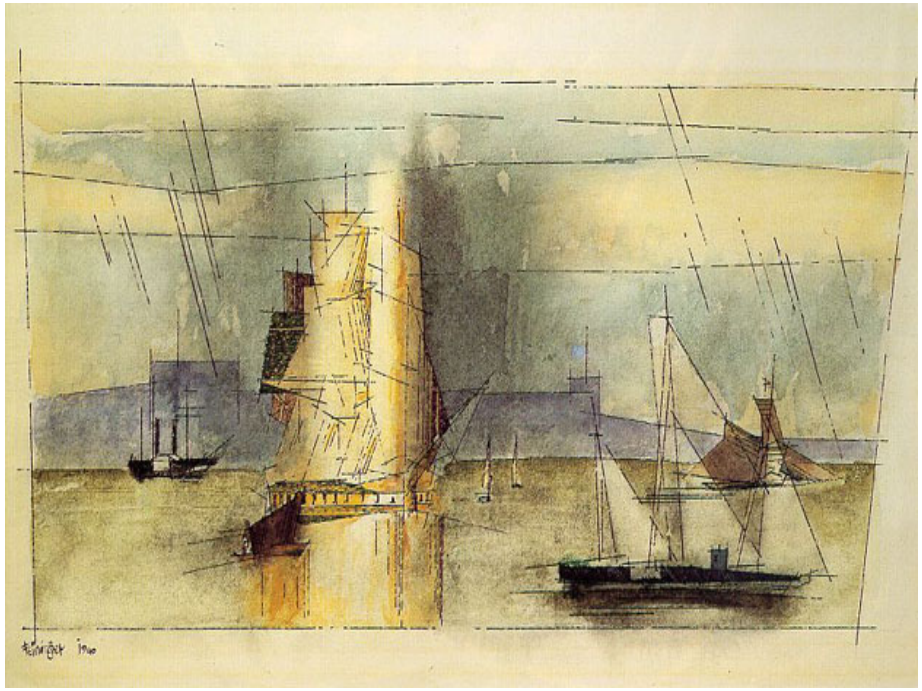


Reflections on Visualization

- Visualizations are useless…
  - …as pictures: Polymetric views are navigable & interactive
  - …if not accessible: Polymetric views are implemented in…
    - CodeCrawler, Mondrian, Sotograph, Jsee, etc.
- It will take some time and a lot of work for them to be accepted - time will tell
- “Everything must change to remain the same”
  [Giuseppe Lanza Tomasi di Lampedusa, “Il Gattopardo”]

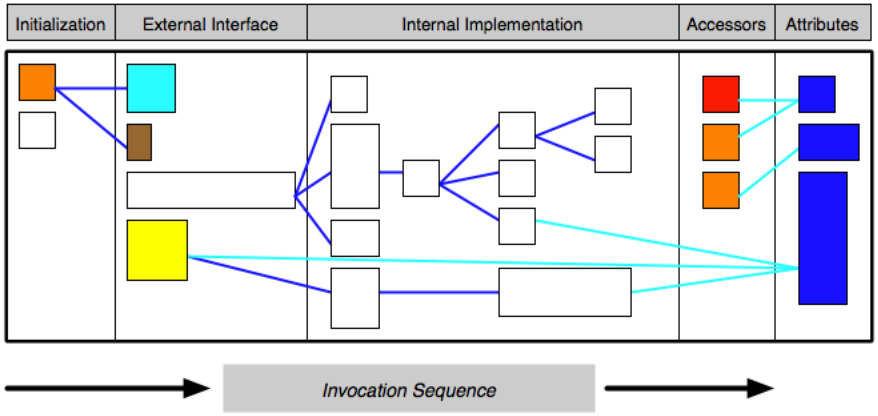
Evaluating the Design of a System

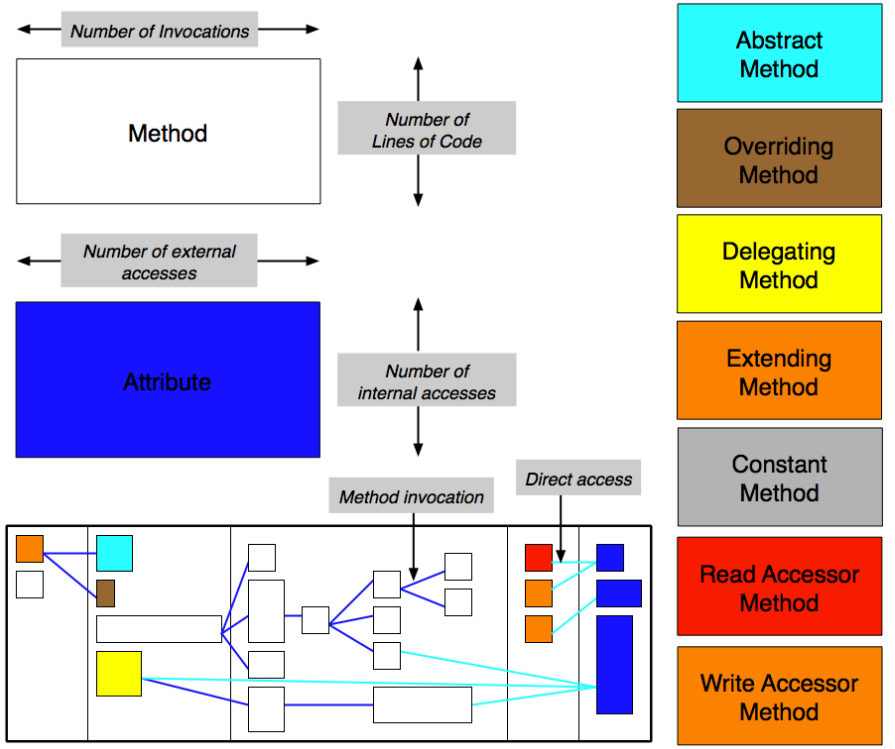
- What entities do we measure in object-oriented design?
  - It depends…on the language
- What metrics do we use?
  - It depends…on our measurement goals
- What can we do with the information obtained?
  - It depends…on our objectives
- Simple metrics are not enough to understand and evaluate design
  - Can you understand the beauty of a painting by measuring its frame?



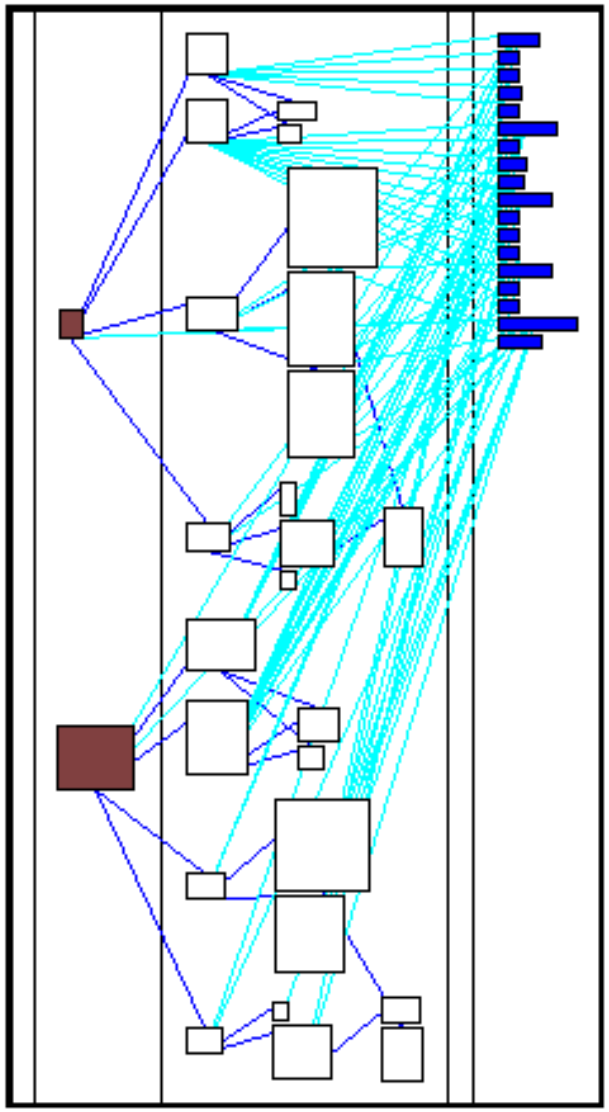
The Class Blueprint

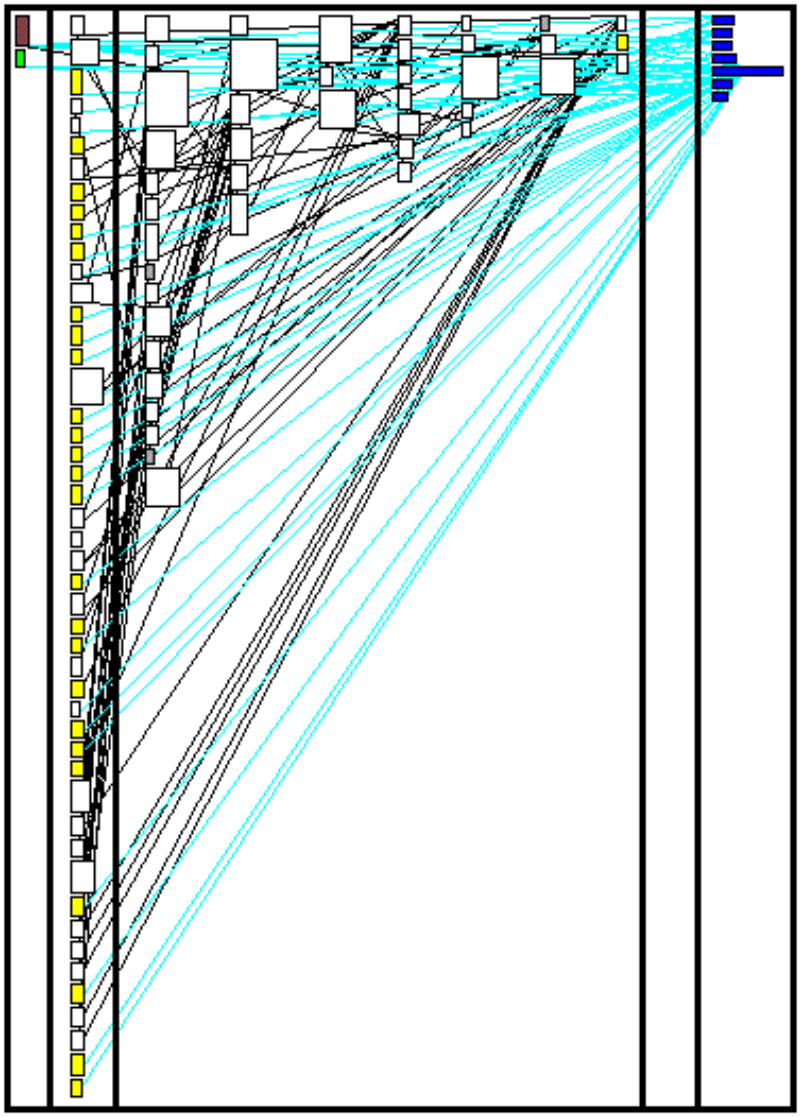
- A semantically rich visualization of the internal structure of classes and class hierarchies
  - Useful for inspecting source code, and detecting visual anomalies which point to desi



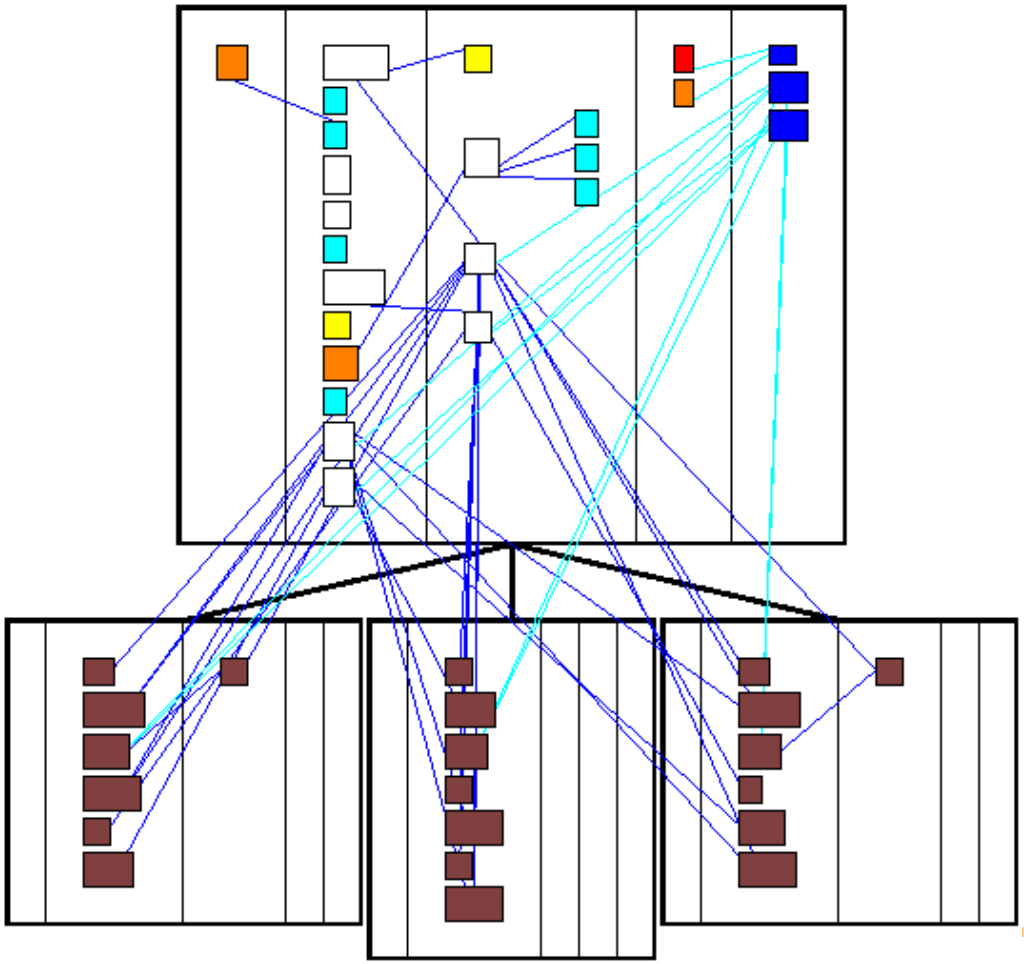


The Class Blueprint: Seeing Code & Design





The Class Blueprint - What do we see?



Nice! …but, what about the practice?

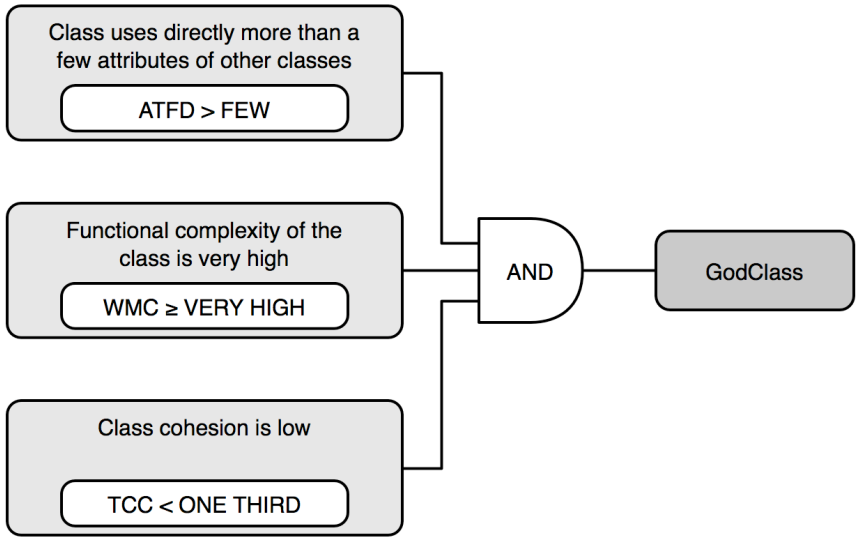
- In practice the key question is ***where to start***
- We have devised a methodology to ***characterize***, ***evaluate*** and ***improve*** the design of object-oriented systems
- It is based on:
  - The Overview Pyramid
  - The System Complexity View
  - Detection Strategies
  - Class Blueprints

Design Harmony

- Software is a human artifact
- There are several ways to implement things
- The point is to find the ***appropriate*** way!
- Appropriate to what?
  - Identity Harmony
    - How do I define myself?
  - Collaboration Harmony
    - How do I interact with others?
  - Classification Harmony
    - How do I define myself with respect to my ancestors and descendants?
- Let's see some examples

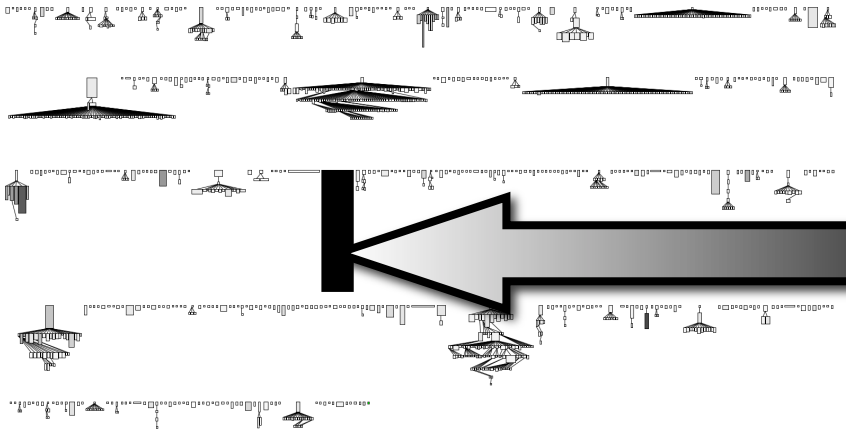
Identity Disharmony: God Class

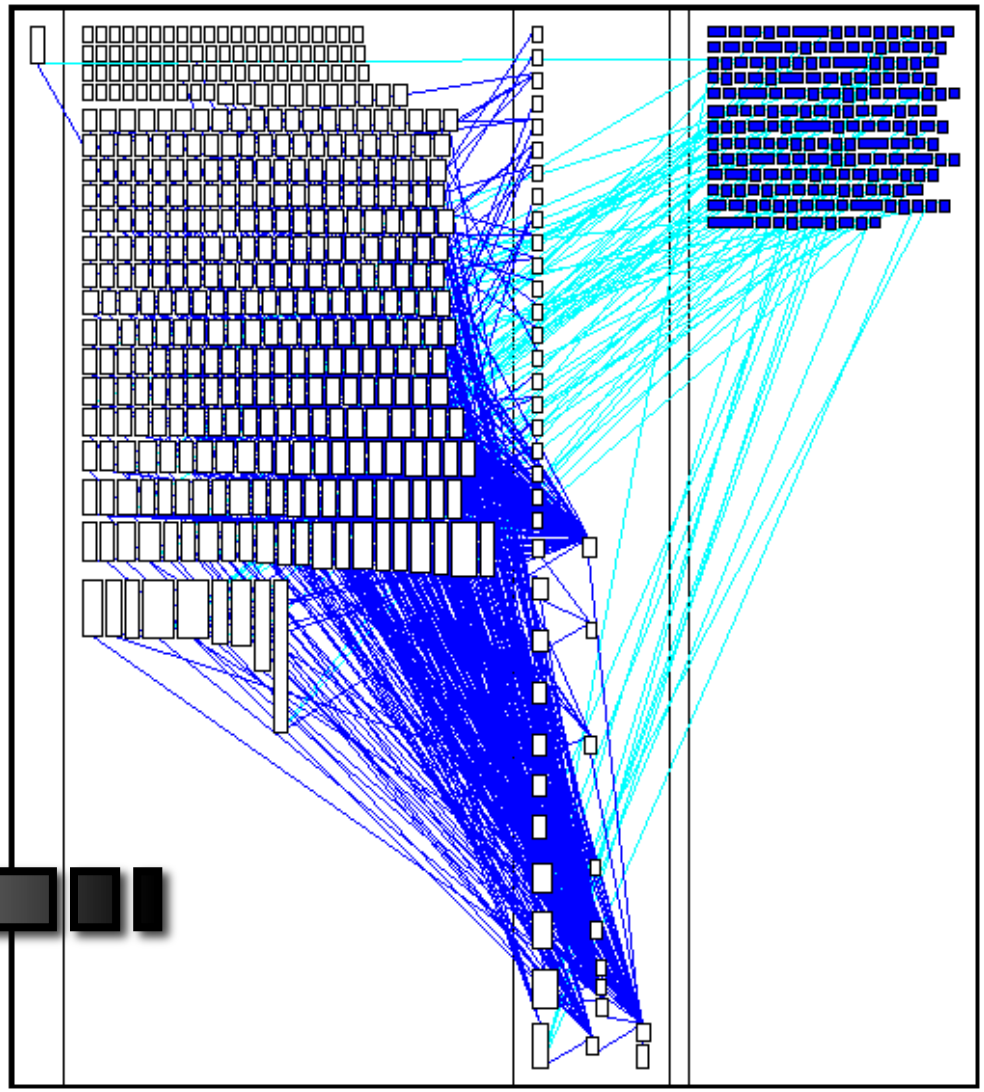
- An aggregation of different abstractions which (mis) uses other classes to perform its functionality
  - The “other” classes are usually dumb data holders
  - Difficult to cure: only do it if it hampers evolution
- Detection: Find large and complex classes on which many other classes depend



Oh my God…it's the ModelFacade

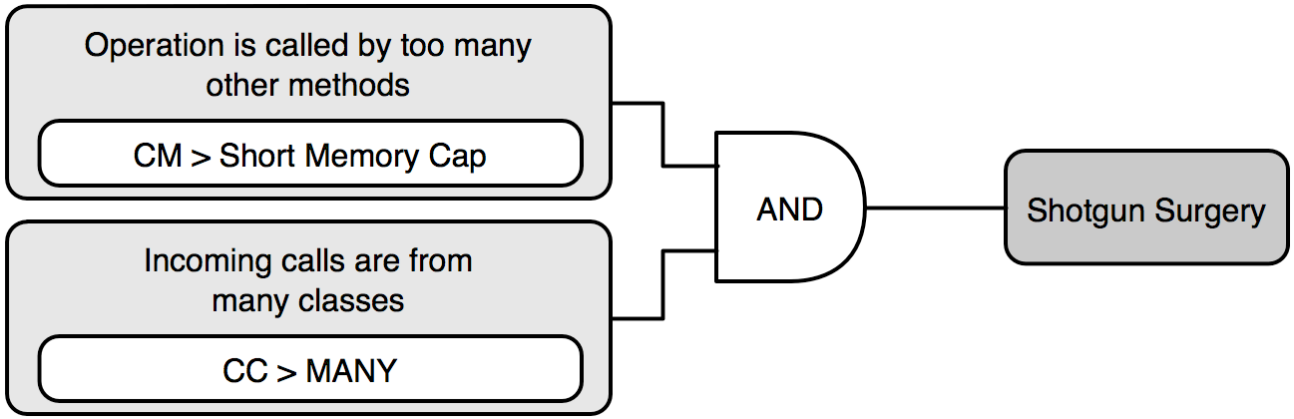
- ModelFacade: The Black Hole
  - 453 methods
  - 114 attributes
  - 3500 lines of code
- Coupled to hundreds of ArgoUML classes



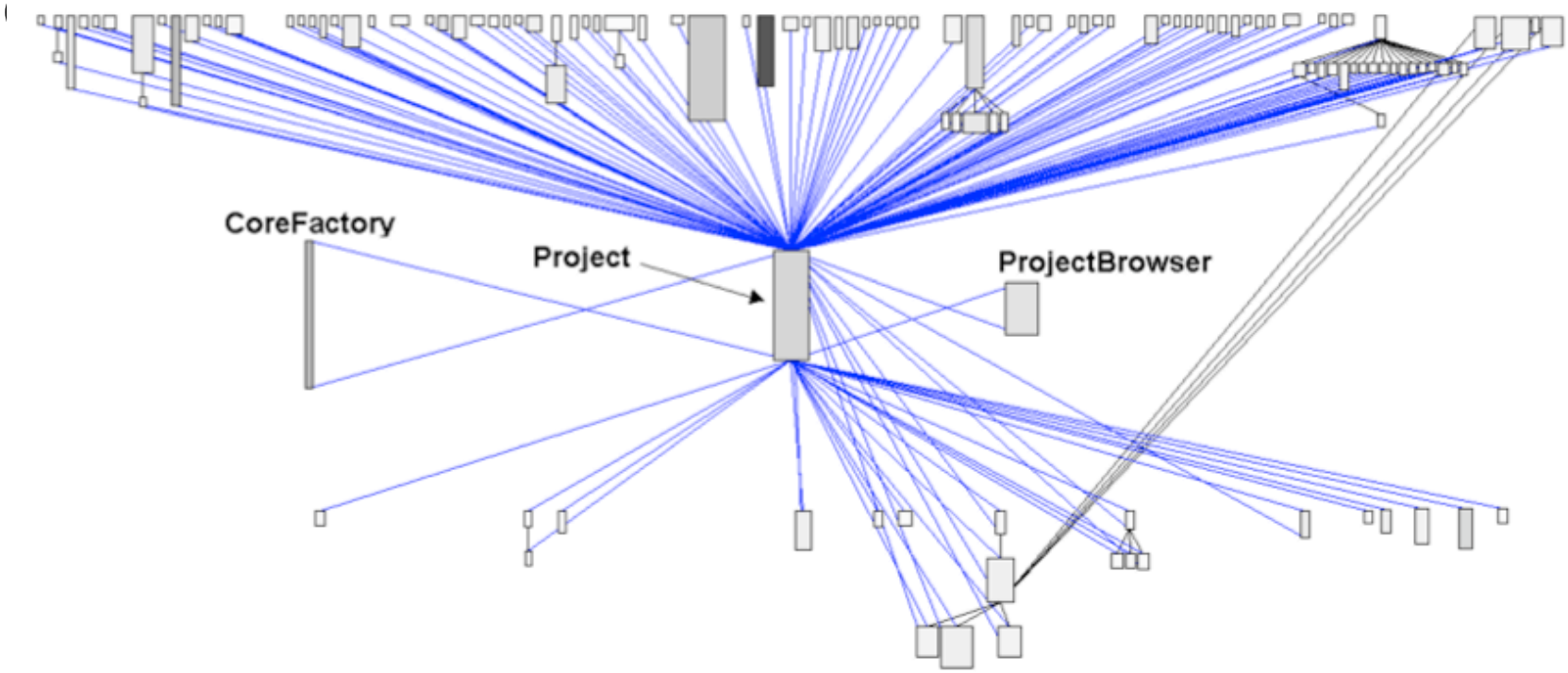


Collaboration Disharmony: Shotgun Surgery

- A change in a method may imply changes in many places
- Detection: Find the classes in which a change would significantly affect many other places in the system
  - We have to consider both the *strength* and the *dispersion* of the coupling
  - We focus on *incoming* coupling

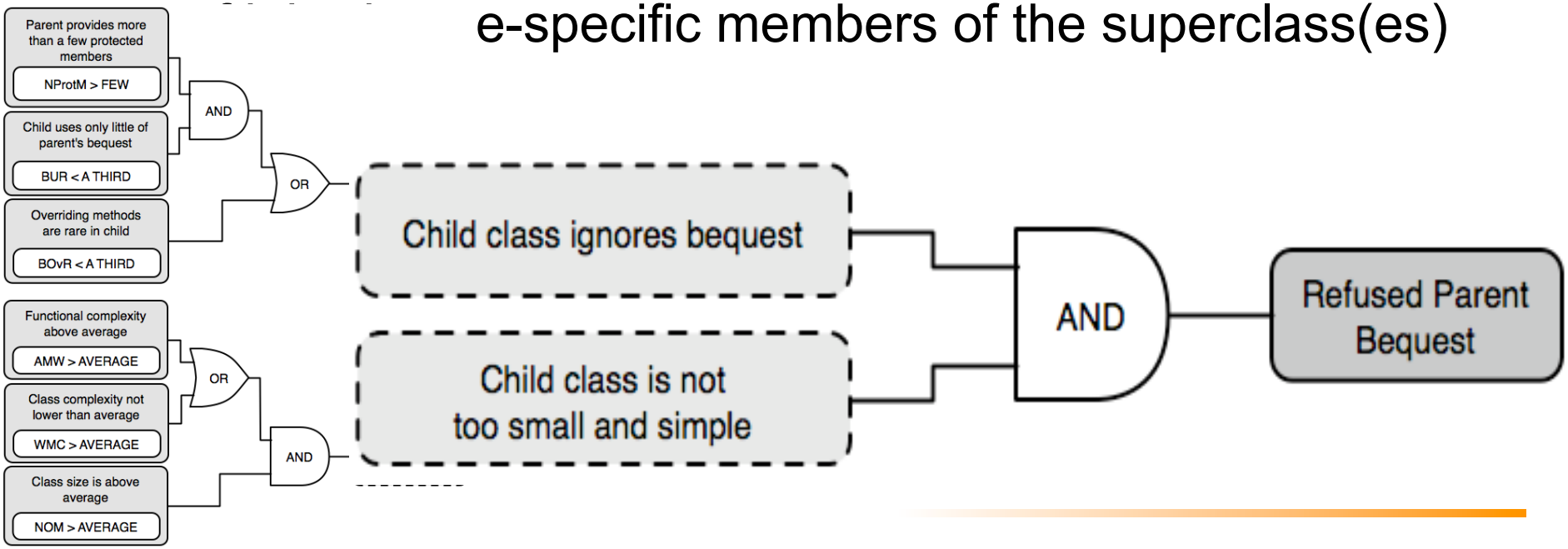


I shot…the Project…

- Project has several methods affected by SS
  - Coupled with 131 classes (ModelFacade not shown here)
  - Cyclic Dependencies with CoreFactory & ProjectBrowser
- 

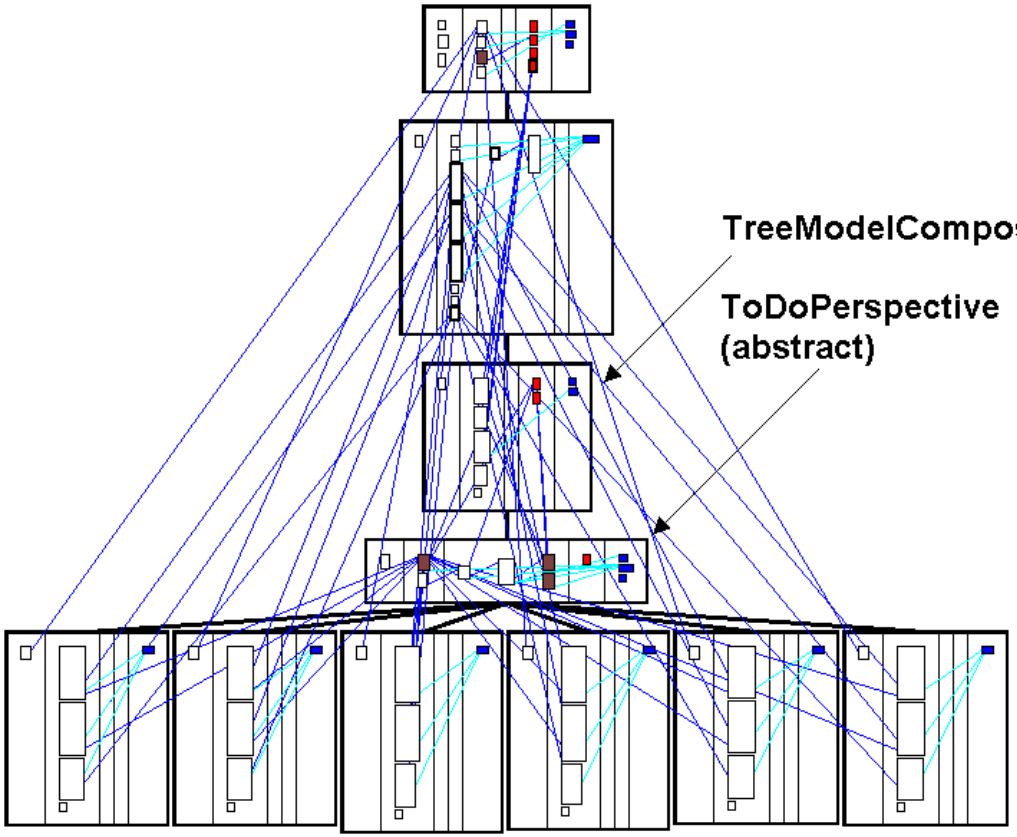
Classification Disharmony: Refused Parent Bequest

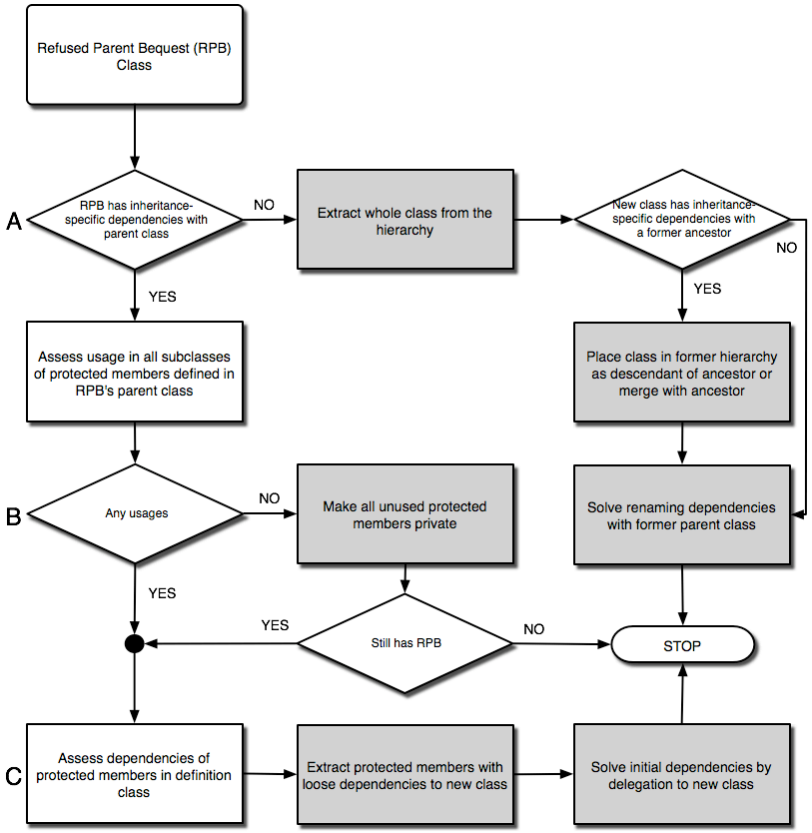
- The primary goal of inheritance: code reuse
  - When you add a subclass you should look at what is “already there”: add/extend-abstract-change cycle
- Detection: Find fairly complex classes with low usage
  e-specific members of the superclass(es)



Kids never listen: The PerspectiveSupport Hierarchy

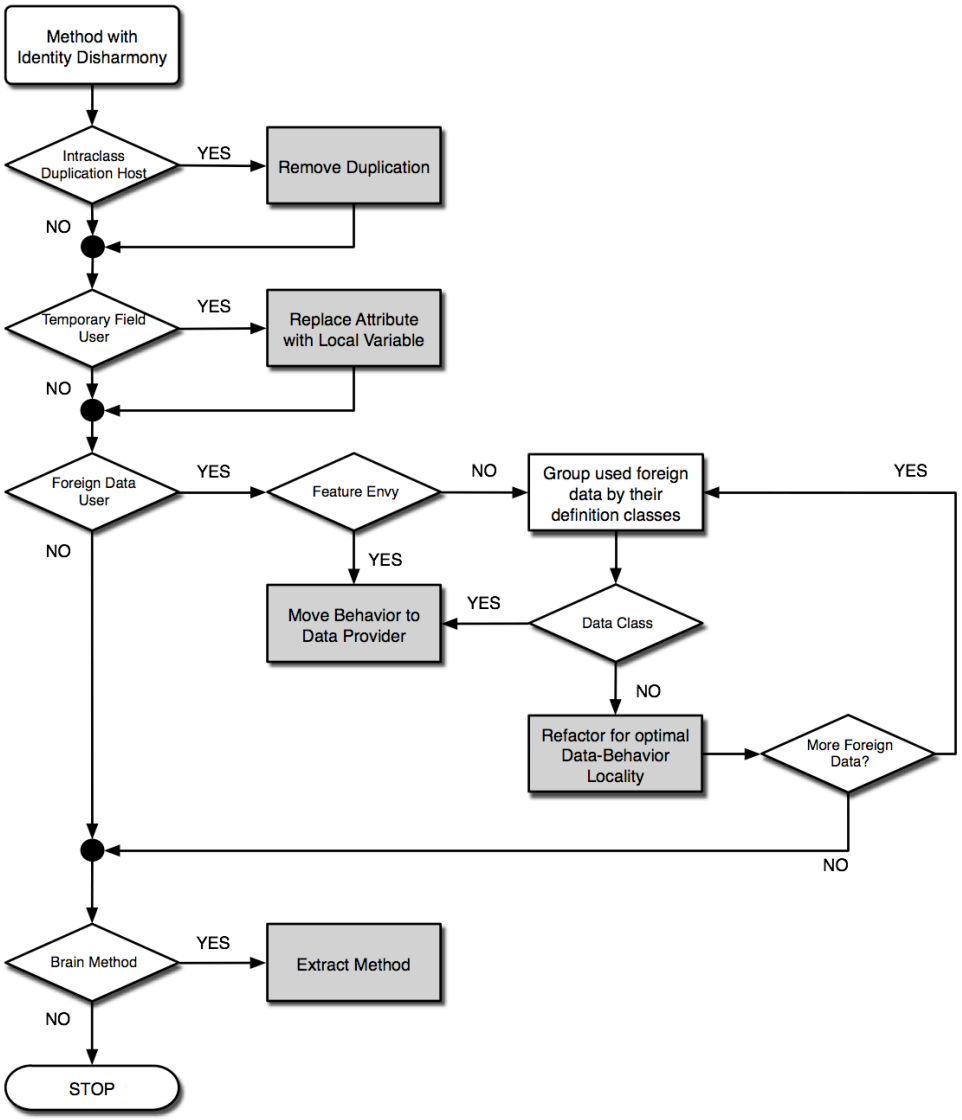
- “Pipeline”-Inheritance with funky usage of abstract classes
- Suspicious regularity in the leaf classes: duplicated code





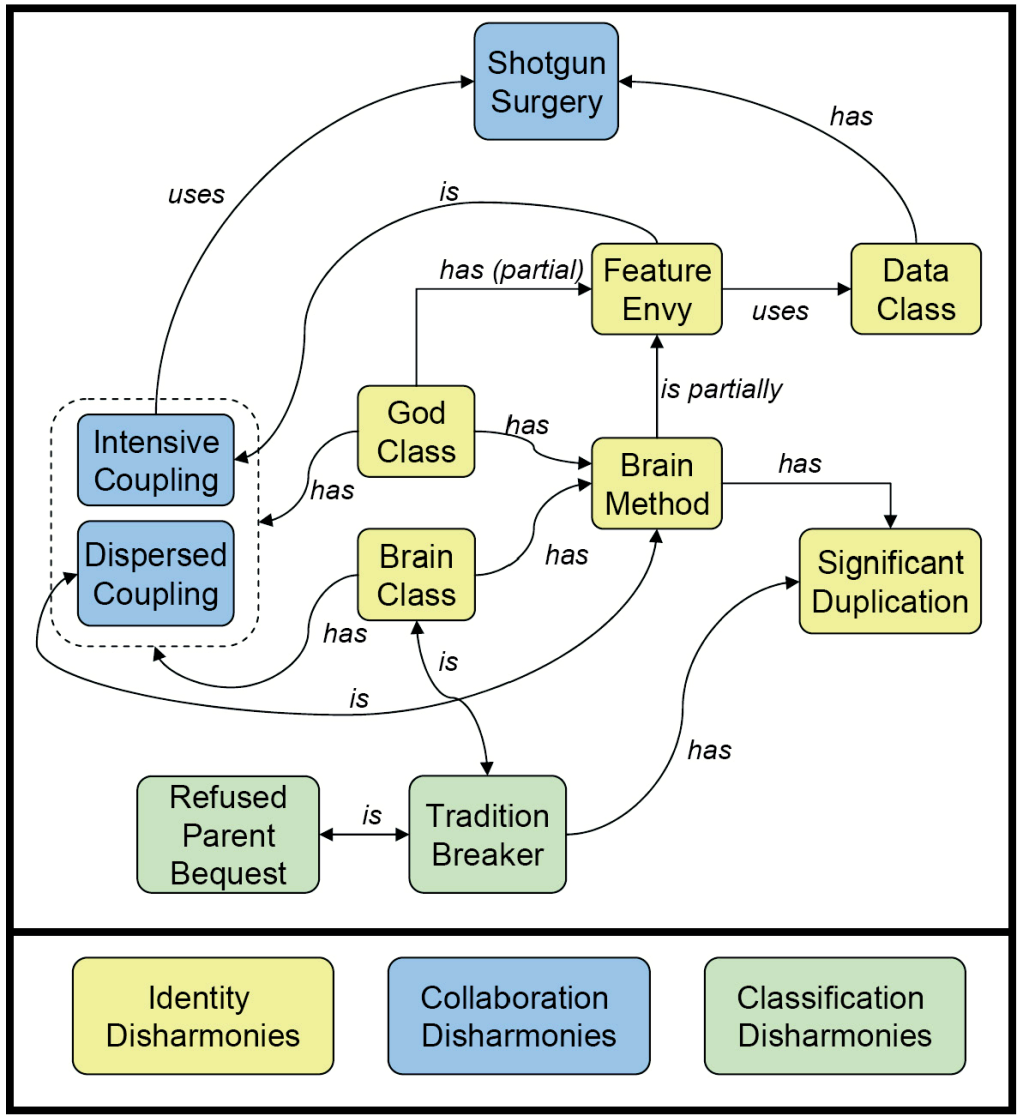
Recovering from a Design Disharmony

- Misery loves company:
  - The Design Disharmonies do not exist alone, they are correlated
- Where to start?
- How to start?
- Recovering can be a lengthy process and must be evaluated in terms of effort/benefit



A Catalogue of Design Disharmonies

- For each Design Disharmony, we provide
  - Description
  - Context
  - Impact
  - Detection Strategy
  - Examples
  - Refactoring



Tools

- “A fool with a tool is still a fool”, but…
- Better a fool with a tool than just a fool…
- Everything presented is based on extensive tooling
  - Moose
  - CodeCrawler
  - iPlasma
  - Free and open source - take it or leave it
- (Parts of) these tools are now making it into industry
  - The Disharmonies are now part of “Borland Together”

Software Visualization: Conclusions

- Software Visualization is very useful when used correctly
- An integrated approach is needed, just having nice pictures is not enough
- Most tools still at prototype level
- In general: only people that know what they see can react on that: SV is for expert/advanced developers
- The future of software development is coming…and SV is part of it

Epilogue

…happily everafter.

- Did we succeed after all?
- Not completely, but…
  - System Hotspots View on 1.200’000 LOC of C++
  - System Complexity View on ca. 200 classes of C++