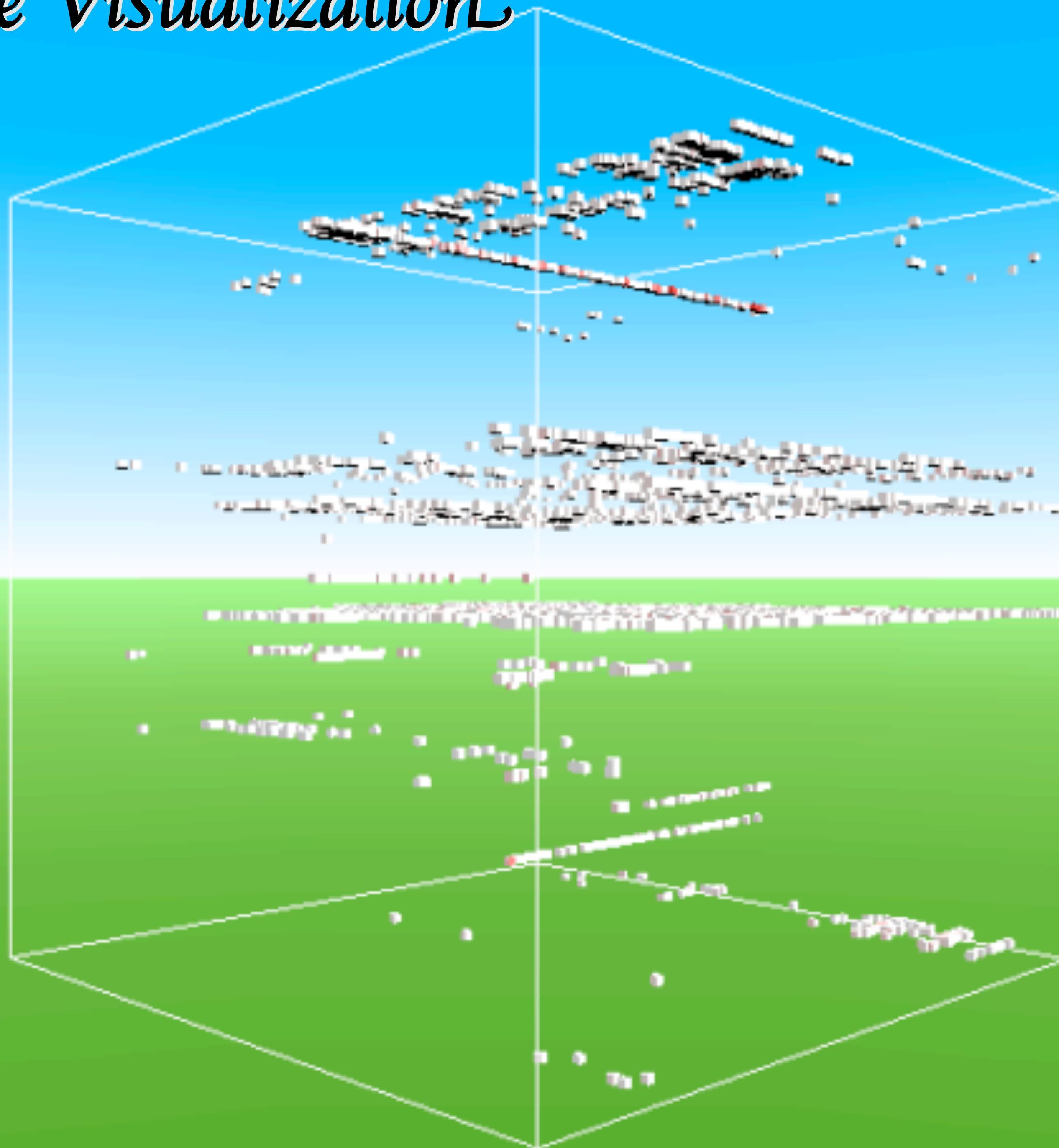
28th International Conference
on Software Engineering

Software Analysis Visualization

Harald Gall and Michele Lanza

Software Visualization

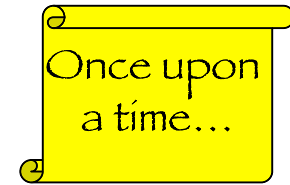


Software Visualization - Outline

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

© 2006 Harald C. Gall & Michele Lanza

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?

© 2006 Harald C. Gall & Michele Lanza

Introduction

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

© 2006 Harald C. Gall & Michele Lanza

Information Visualization

- The human eye and brain interpret visual information in order to “react to the world”
- We want to answer questions on what we perceive
- J. Bertin inferred three levels of questions
  - Lower perception (one element)
  - Medium perception (several elements)
  - Upper perception (all elements/the complete picture)
- Information Visualization is about
  - how to display information
  - how to reduce its complexity

 © 2006 Harald C. Gall & Michele Lanza

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

© 2006 Harald C. Gall & Michele Lanza

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]

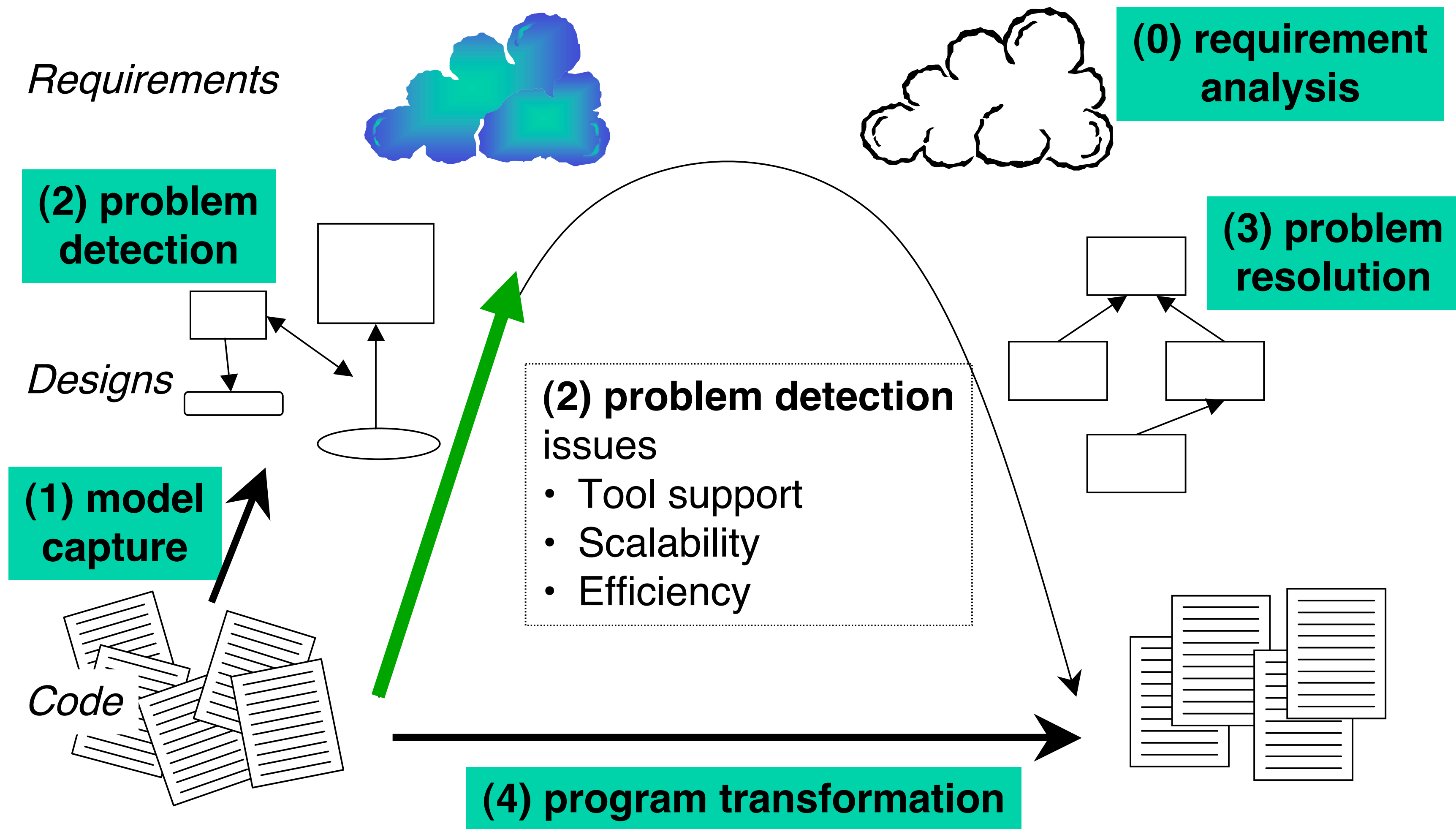
 © 2006 Harald C. Gall & Michele Lanza

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

© 2006 Harald C. Gall & Michele Lanza

The Reengineering Life-cycle



 © 2006 Harald C. Gall & Michele Lanza

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, …)

© 2006 Harald C. Gall & Michele Lanza

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?

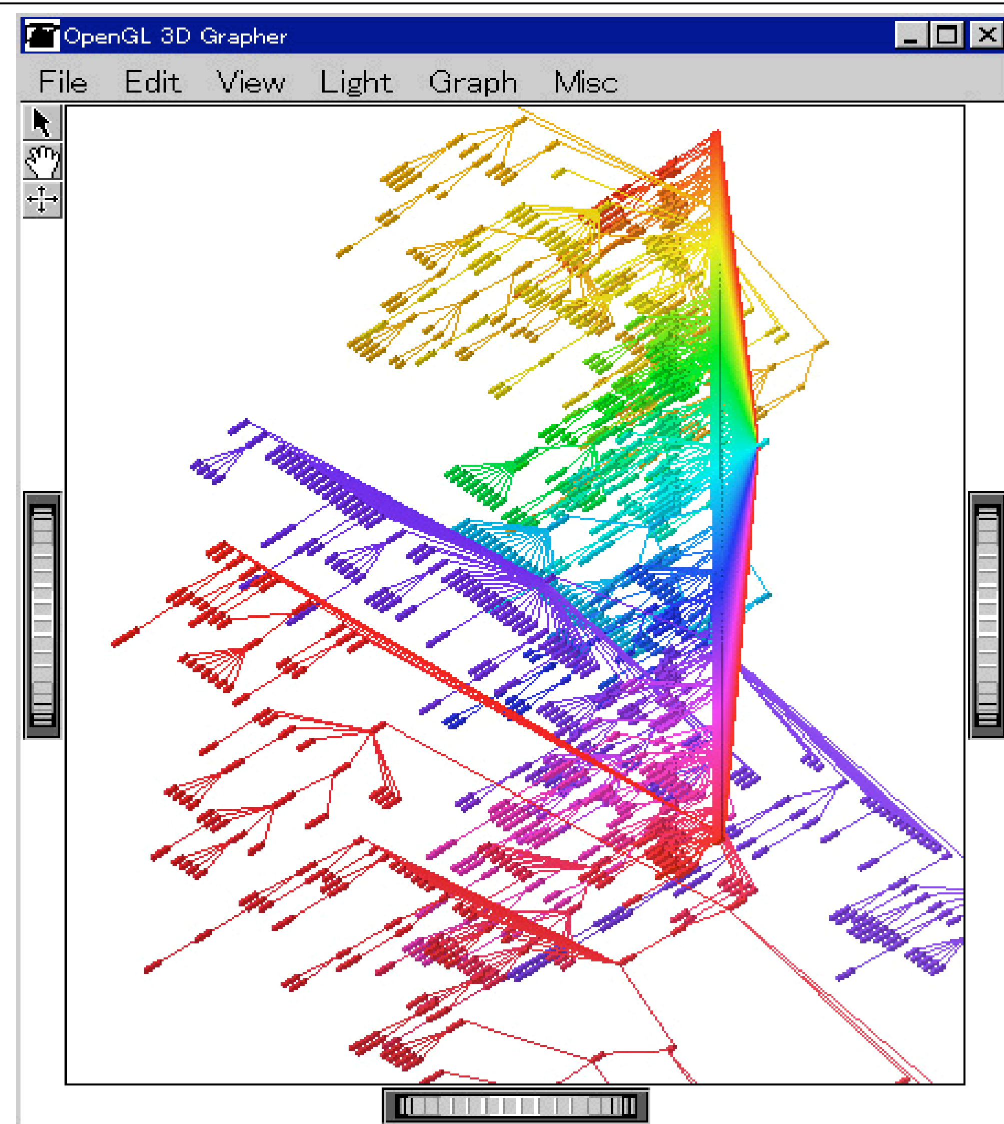
© 2006 Harald C. Gall & Michele Lanza

Static Code Visualization

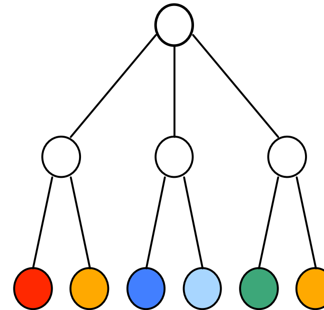
- The Visualization of information that can be extracted from the static structure of a software system
- Depends on the programming language and paradigm:
  - Object-Oriented PL:
    - classes, methods, attributes, inheritance, …
  - Procedural PL:
    - procedures, invocations, …
  - Functional PL: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

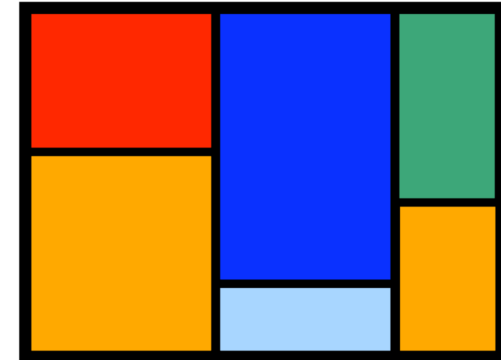
Example 1: Class Hierarchies

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

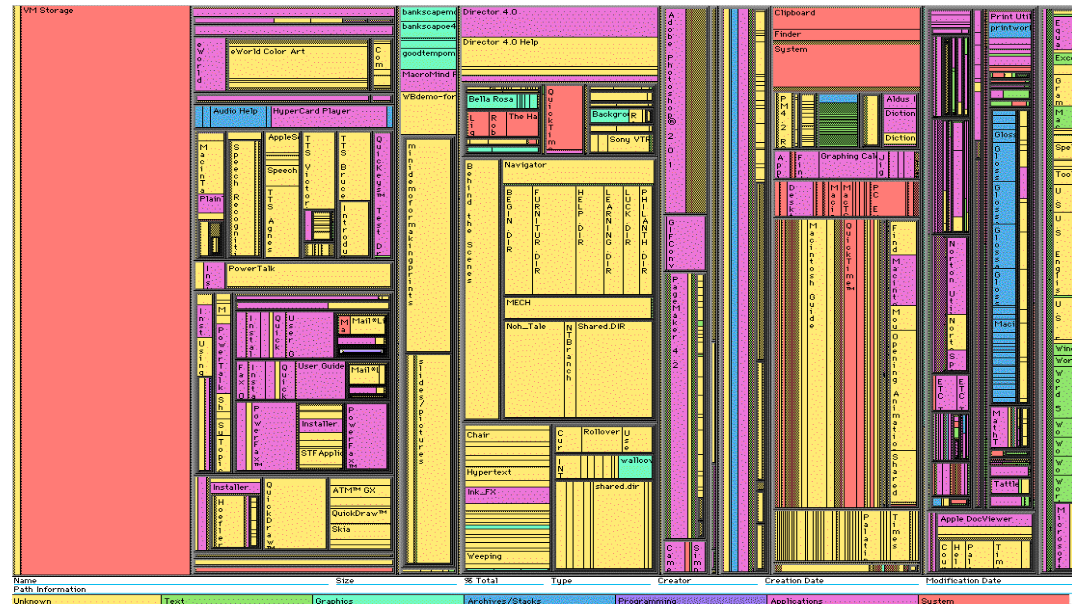


Example 2: Tree Maps





- 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

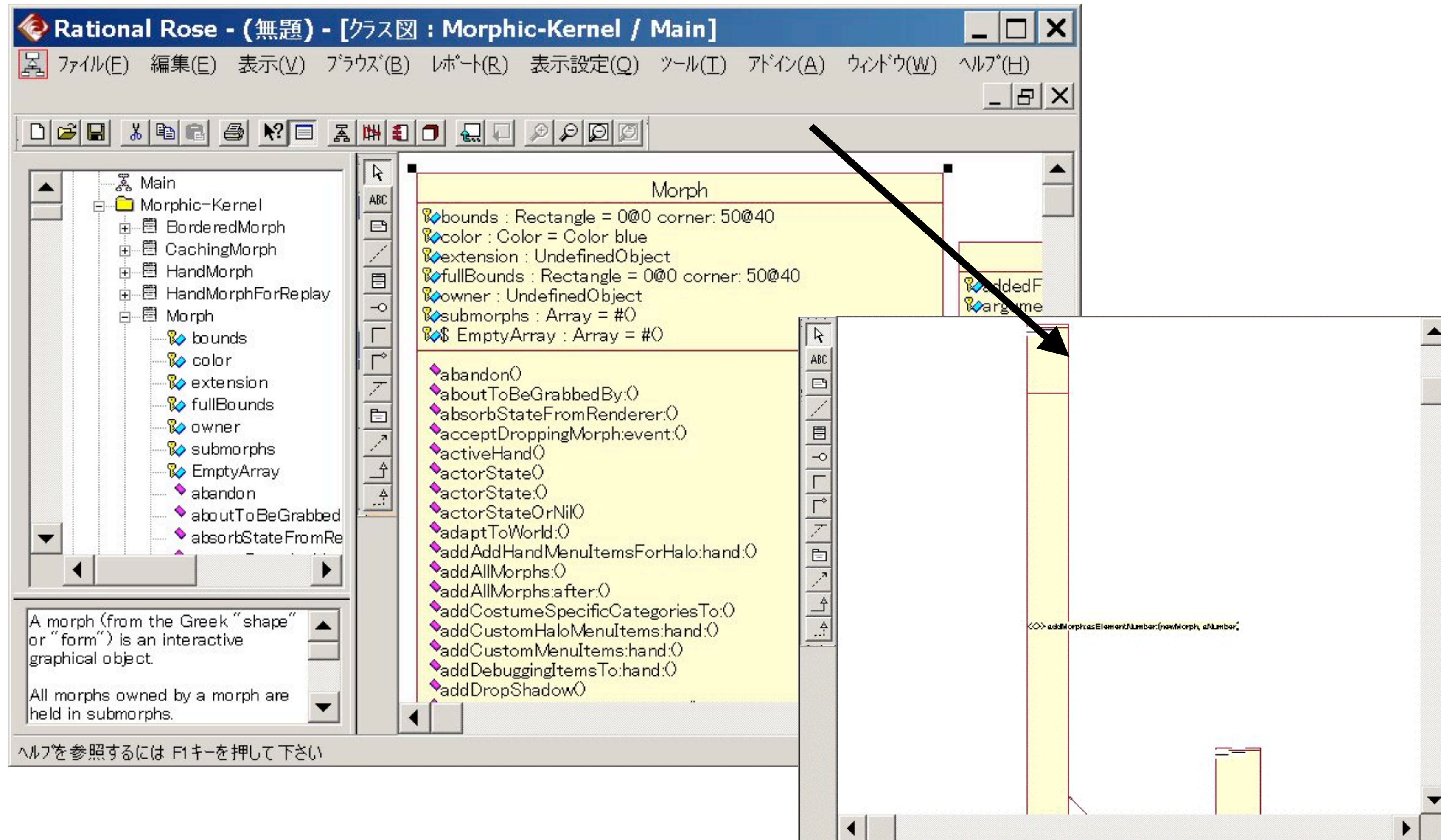
© 2006 Harald C. Gall & Michele Lanza

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

© 2006 Harald C. Gall & Michele Lanza

Class Diagram Examples



Example 5: MetricView

- UML & 3D

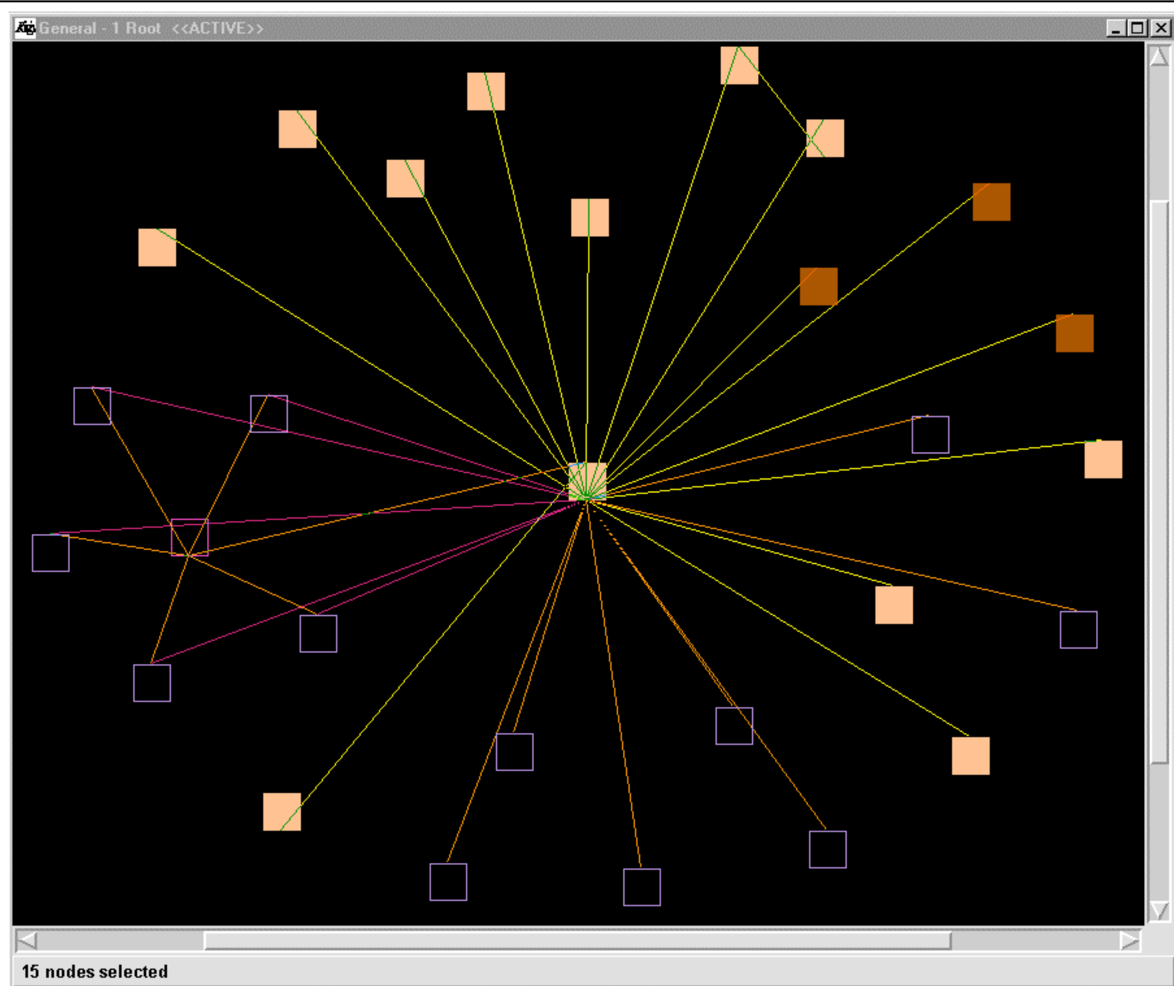
© 2006 Harald C. Gall & Michele Lanza

Example 6a: Rigi

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

Example 6b: 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

© 2006 Harald C. Gall & Michele Lanza

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
    - …

© 2006 Harald C. Gall & Michele Lanza

Example 1: JInsight

- Visualization of execution trace

variant button

bright blue stripe

© 2006 Harald C. Gall & Michele Lanza

Example 2: Inter-class call matrix

- Simple
- Scales quite well
- Reproducible

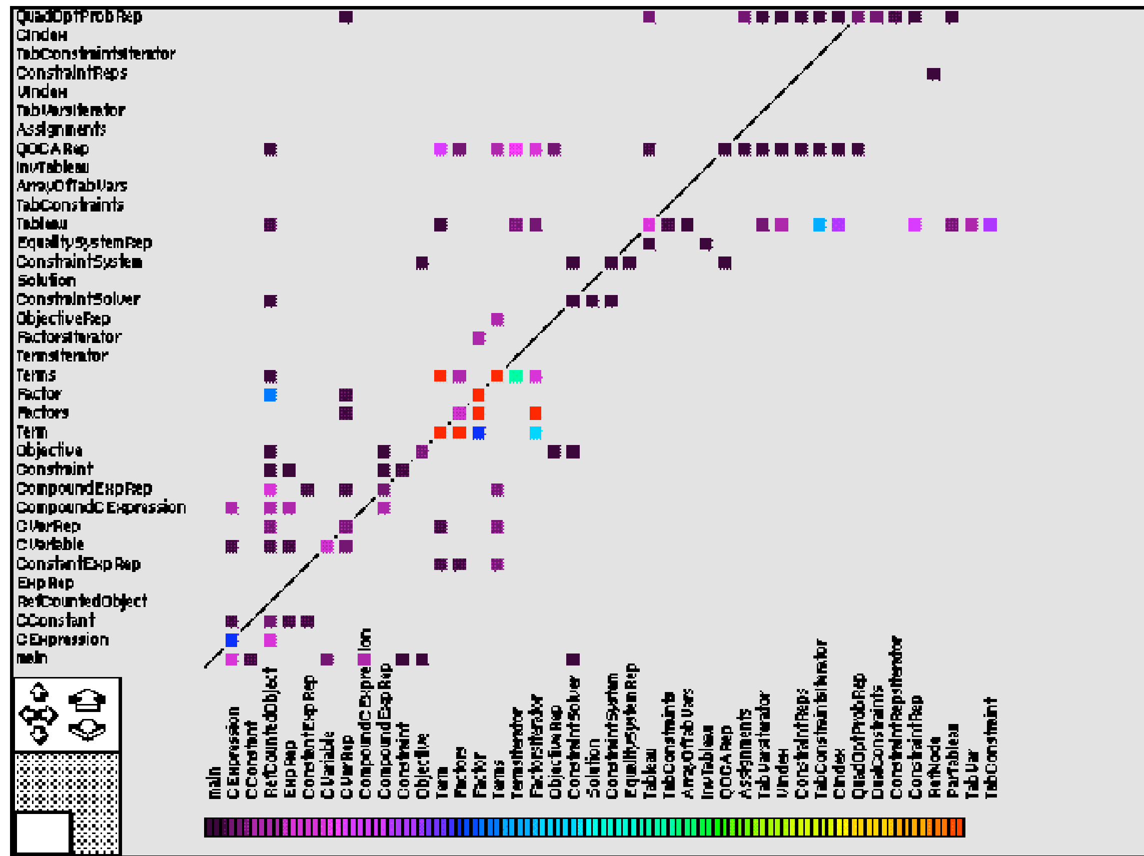
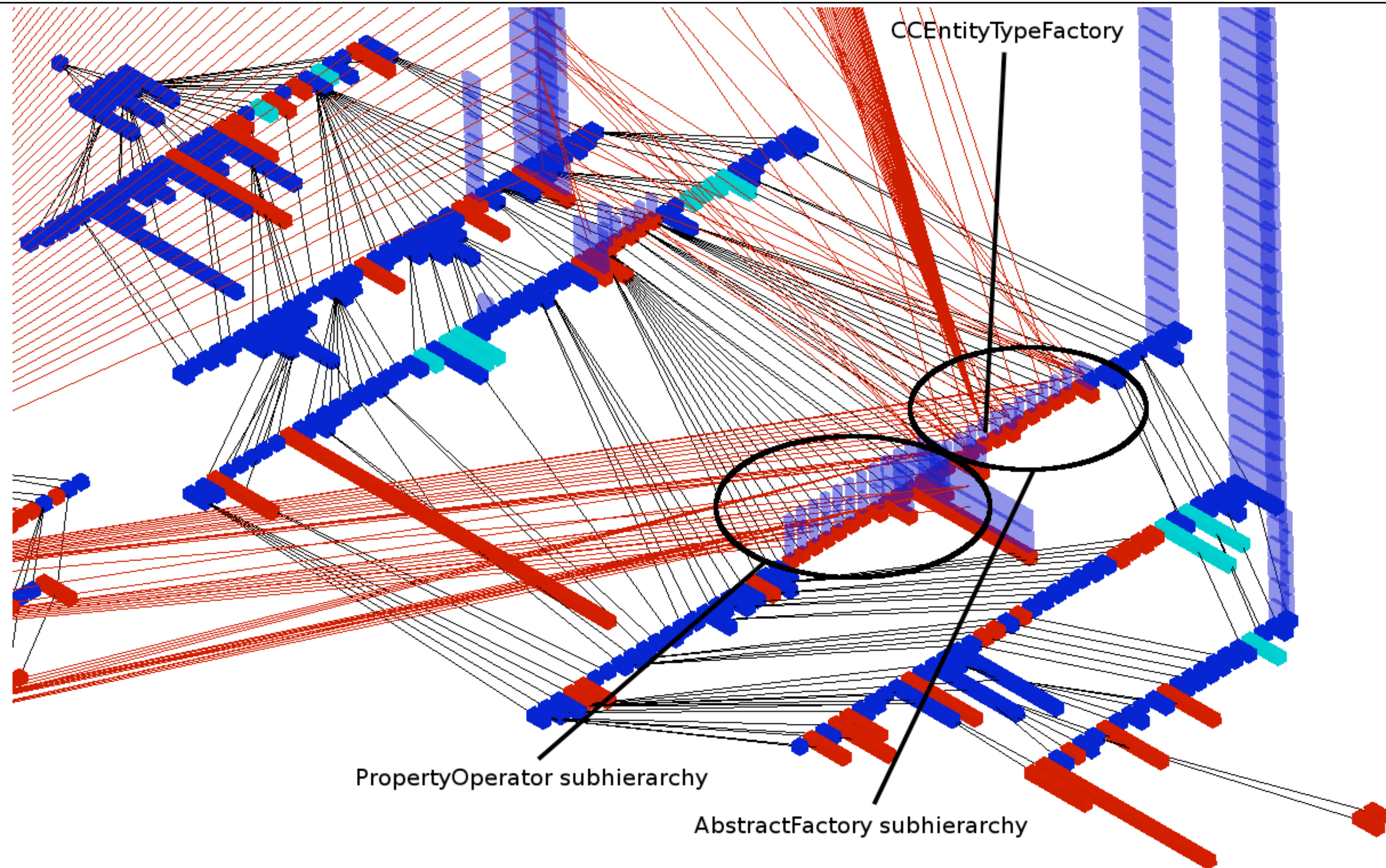


Figure 6: Inter-class call matrix

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

 © 2006 Harald C. Gall & Michele Lanza

Taking a step back…

- Why is visualization important at all?
- Is it actually useful?
  - No, visualization is only a means, not the end…
  - Yes, visualization is only a means, not the end!!!
- The question is: “What is the end?”
  - We want to understand systems…

© 2006 Harald C. Gall & Michele Lanza

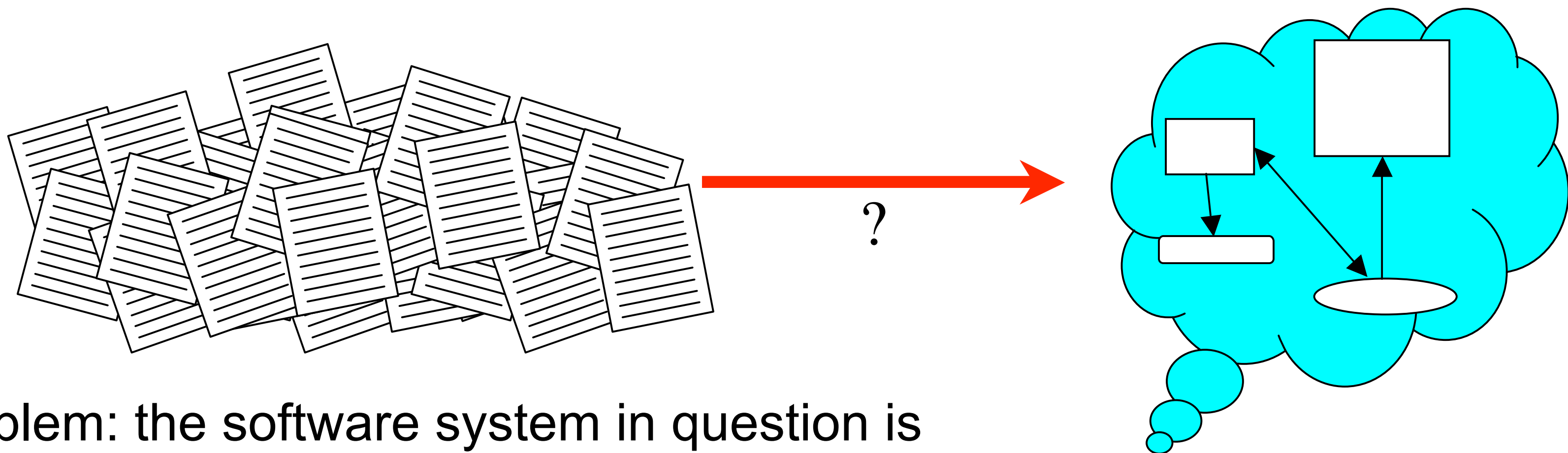
Lightweight Approaches

- Already existing approaches and tools exist:
  - hyperbolic views, fish-eye views, spring layouts, …
  - Rigi, ShrimpView, Creole, Gsee, …
  - Some of them are even copyrighted and/or commercial tools!
- Why are they not widely used?
- The reengineering context does not permit heavy-weight approaches
  - Let’s do it lightweight then…

© 2006 Harald C. Gall & Michele Lanza

Object-Oriented Reverse Engineering

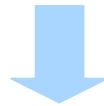
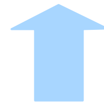
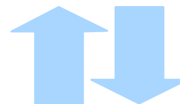
- Goal: take a (*large legacy)* software system and "understand" it, *i.e., construct a mental model* of the system



- Problem: the software system in question is
  - Unknown, very large, and complex
  - Domain- and language-specific
  - Seldom documented or commented
  - "In bad shape"

© 2006 Harald C. Gall & Michele Lanza

Object-Oriented Reverse Engineering (II)

- Constructing a mental model requires *information* about the system:
  - Top-down approaches 
  - Bottom-up approaches 
  - *Mixed Approaches* 
- There is no “silver bullet” methodology
- Every reverse engineering situation is unique
- Need for flexibility, customizability, scalability, and simplicity

© 2006 Harald C. Gall & Michele Lanza

Reverse Engineering Approaches

- Reading (source code, documentation, UML diagrams, comments)
- Running the SW and analyze its execution trace
- Interview users and developers (if available)
- Clustering
- Concept Analysis
- Software Visualization
- Software Metrics
- Slicing and Dicing
- Querying (Database)
- Data Mining
- Logic Reasoning
- …

© 2006 Harald C. Gall & Michele Lanza

The "Information Crystallization" Problem

- Many approaches generate too much or not enough information

- The reverse engineer must make sense of this information by himself

- We need the *right* information at the *right* time

What is the actual problem?

- The information needed to reverse engineer a legacy software system resides at various levels
- We need to obtain and combine
  - Coarse-grained information about the whole system
  - Fine-grained information about specific parts
  - Evolutionary information about the past of the system

© 2006 Harald C. Gall & Michele Lanza

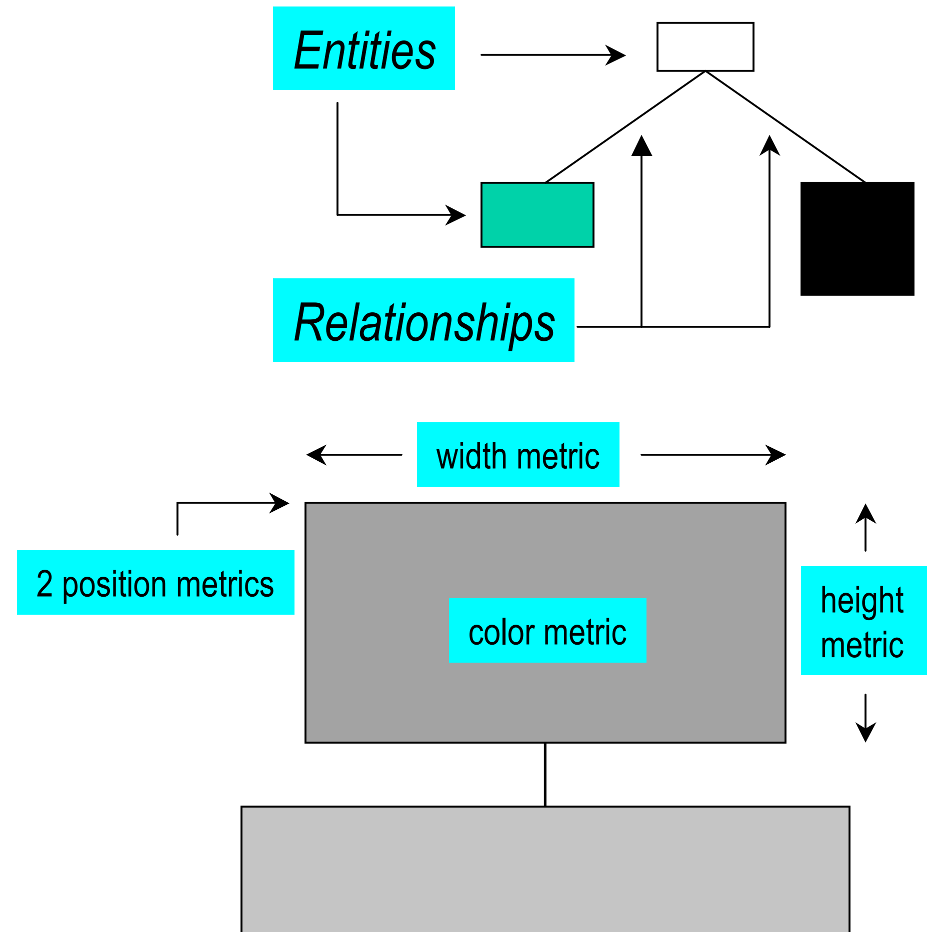
A simple Solution - The Polymetric View

- A lightweight combination of two approaches:
  - Software visualization (reduction of complexity, intuitive)
  - Software metrics (scalability, assessment)
- Interactivity (iterative process, silver bullet impossible)
- Does not replace other techniques, it complements them:
  - “Opportunistic code reading”

 © 2006 Harald C. Gall & Michele Lanza

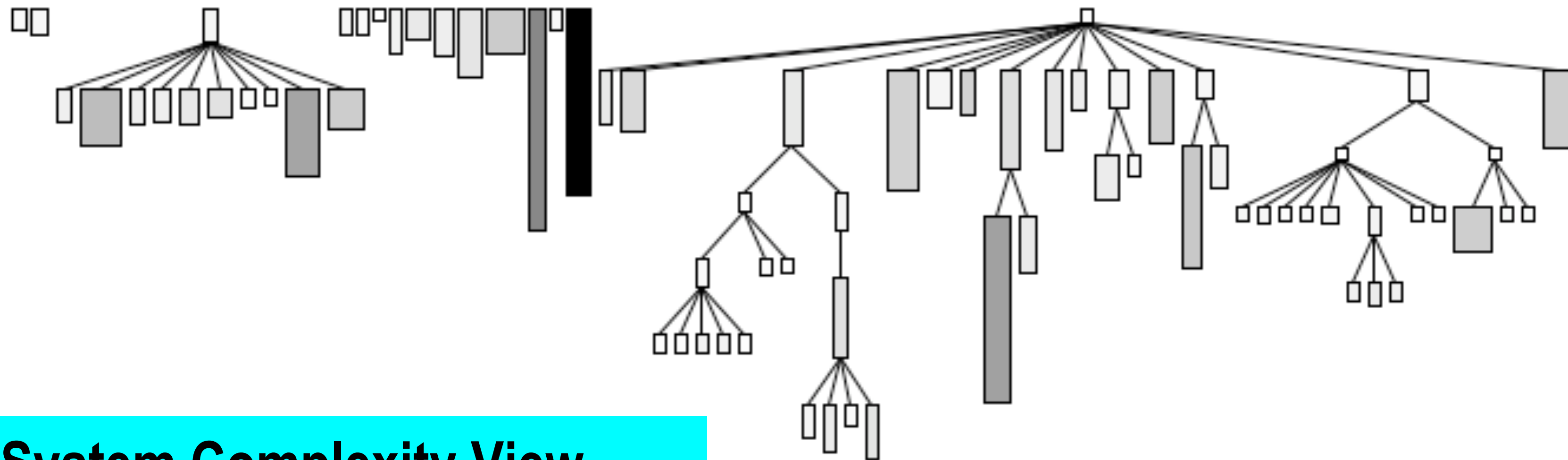
The Polymetric View - Principles

- Visualize software:
  - entities as rectangles
  - relationships as edges
- Enrich these visualizations:
  - Map up to 5 software metrics on a 2D figure
  - Map other kinds of semantic information on *nominal* colors



© 2006 Harald C. Gall & Michele Lanza

The Polymetric View - Example



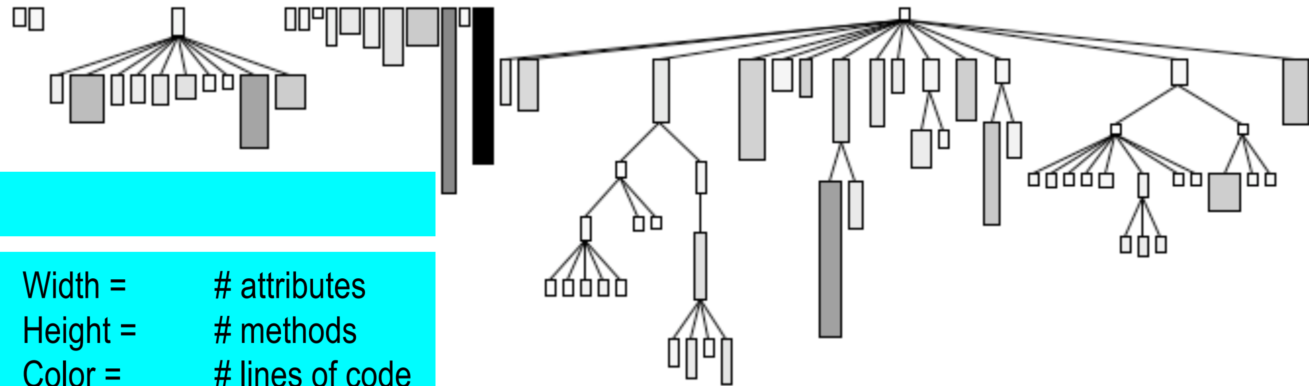
System Complexity View

Nodes = Classes
Edges = Inheritance Relationships

Width = Number of Attributes
Height = Number of Methods
Color = Number of Lines of Code

© 2006 Harald C. Gall & Michele Lanza

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

The Polymetric View - Description

- Every polymetric view is described according to a common pattern
- Every view targets specific reverse engineering goals
- The polymetric views are implemented in CodeCrawler

System Complexity View

Structural Specification

Target ......

Scope ....

Metrics ....
....

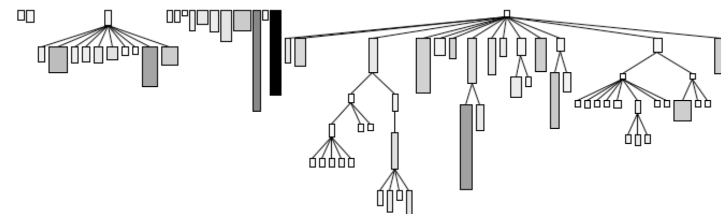
Layout ....

Description ....
....

Goals ....
....

Symptoms ....
....

Scenario



Case Study ....
....

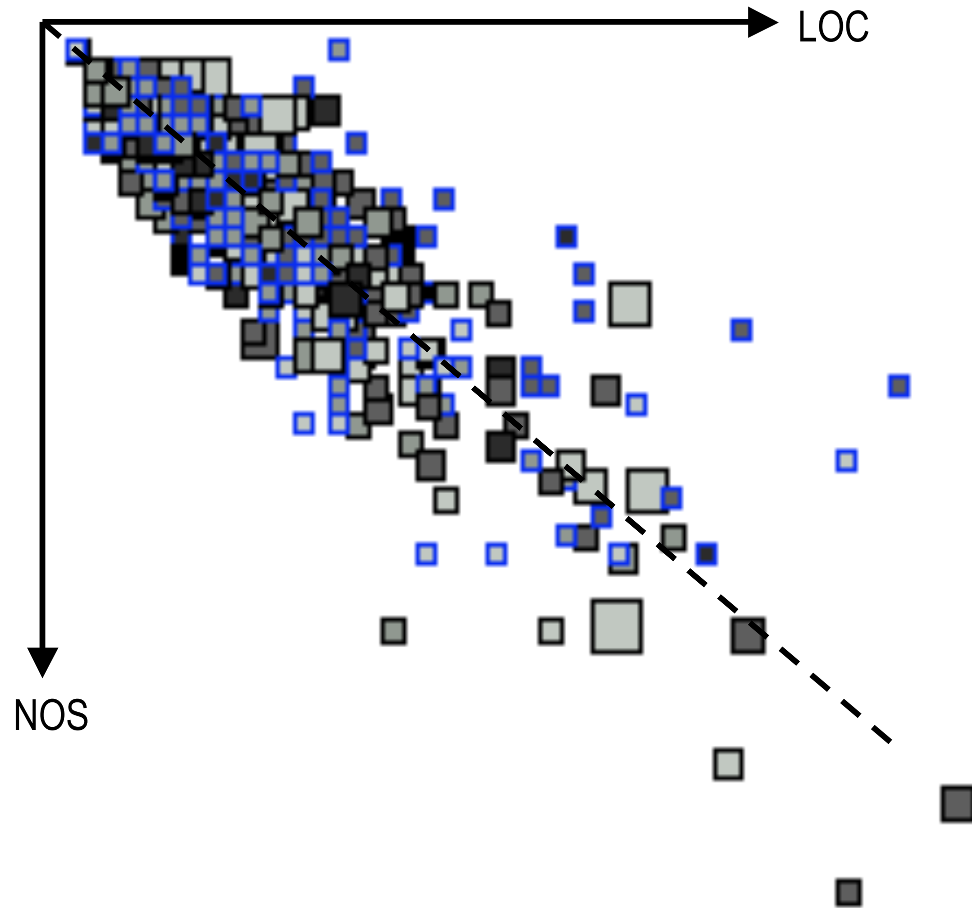
© 2006 Harald C. Gall & Michele Lanza

Coarse-grained Software Visualization

- Reverse engineering question:
  - What is the size and the overall structure of the system?
- Coarse-grained reverse engineering goals:
  - Gain an overview in terms of size, complexity, and structure
  - Asses the overall quality of the system
  - Locate and understand important (domain model) hierarchies
  - Identify large classes, exceptional methods, dead code, etc.
  - …

 © 2006 Harald C. Gall & Michele Lanza

Coarse-grained Polymetric Views - Example



Method Efficiency Correlation View

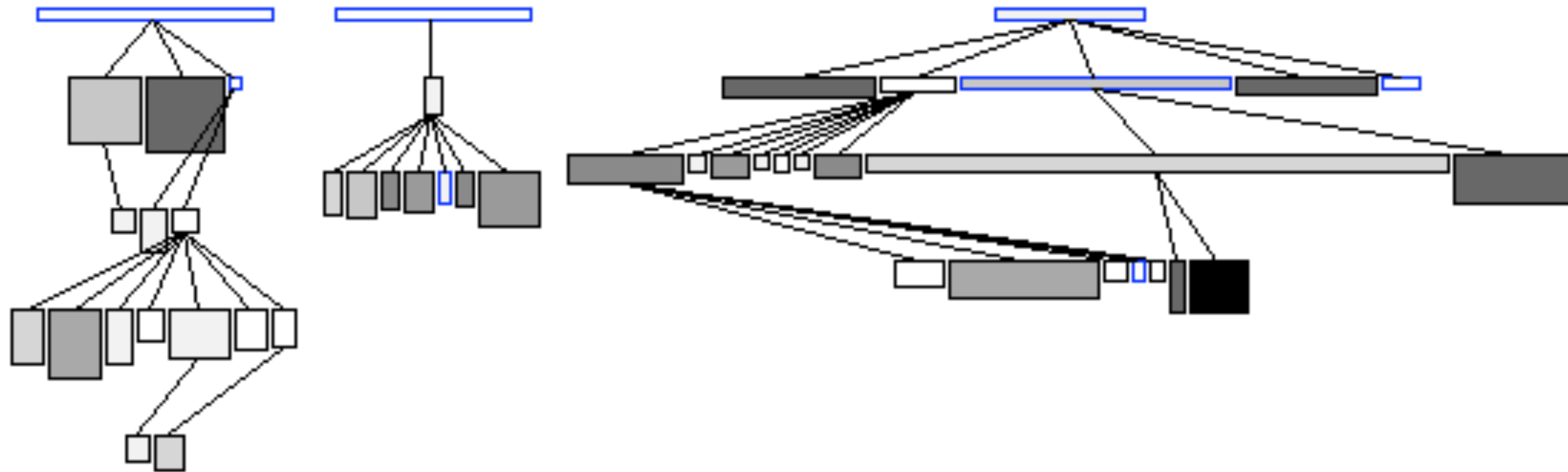
Nodes: Methods
Edges: -
Size: Number of method parameters
Position X: Number of lines of code
Position Y: Number of statements

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

© 2006 Harald C. Gall & Michele Lanza

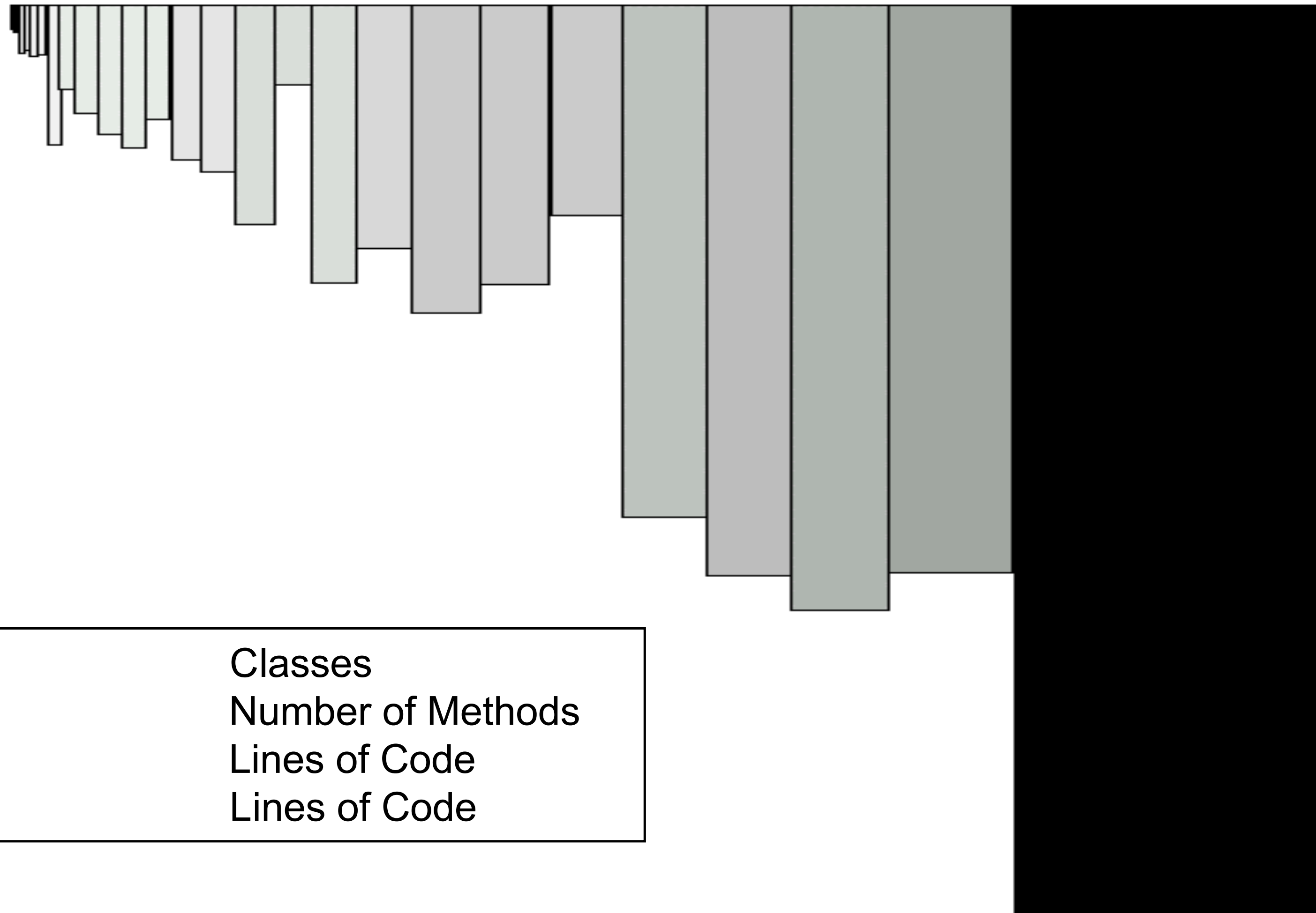
Inheritance Classification View



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

© 2006 Harald C. Gall & Michele Lanza

Data Storage Class Detection View



Quiz: Where would you start looking?

Nodes:	Classes
Edges:	Inheritance Relationships
Width:	Number of attributes
Height:	Number of methods
Color:	Number of lines of code

© 2006 Harald C. Gall & Michele Lanza

CodeCrawler Demo

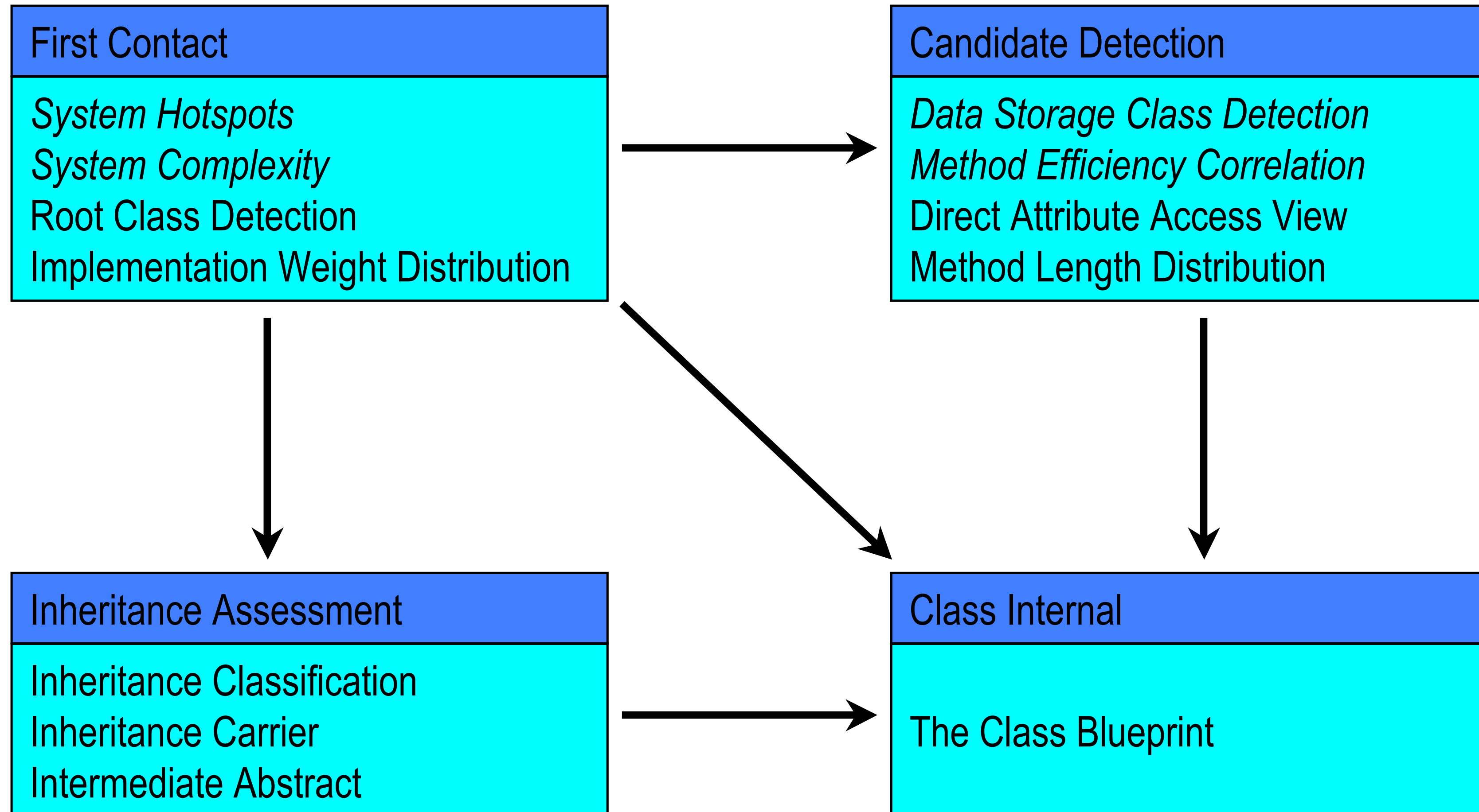
CodeCrawler – System Complexity View

CodeCrawler Moose Apply View Spawn View Selection Transformation Colors Layout

Item | Class Root::Smalltalk::CodeCrawler::CodeCrawler II NOA: 5 I NOM: 74 I WLOC: 548 I -: 0 I -: 0 I

77 Nodes, 61 Edges - 0 selected Nodes

Clustering the Polymetric Views



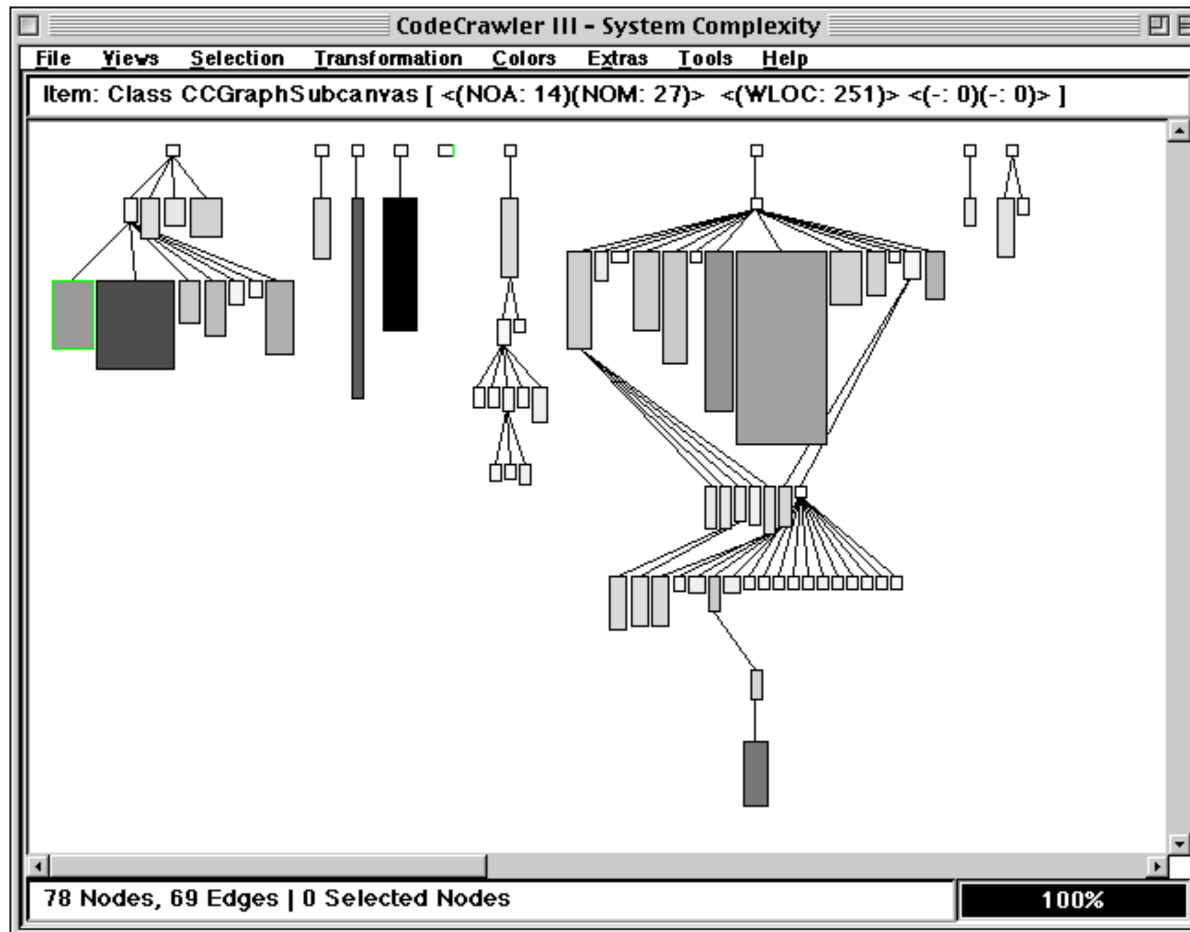
© 2006 Harald C. Gall & Michele Lanza

Coarse-grained SV - Conclusions

- Benefits
  - Views are customizable (context…) and easily modifiable
  - Simple approach, yet powerful
  - Scalability
- Limits
  - Visual language must be learned

© 2006 Harald C. Gall & Michele Lanza

Granularity level problem: It looks nice, but...what's inside?



© 2006 Harald C. Gall & Michele Lanza

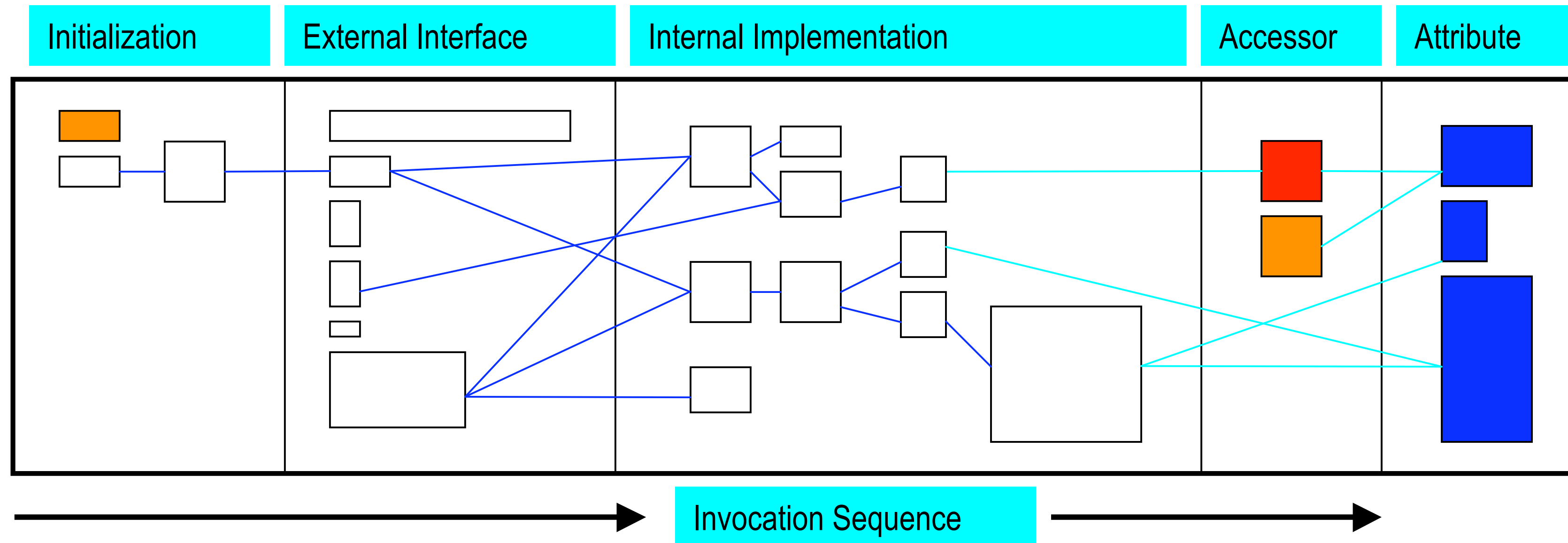
Fine-grained Software Visualization

- Reverse engineering question:
  - What is the internal structure of the system and its elements?

- Fine-grained reverse engineering goals:
  - Understand the internal implementation of classes and class hierarchies
  - Detect coding patterns and inconsistencies
  - Understand class/subclass roles
  - Identify key methods in a class
  - …

© 2006 Harald C. Gall & Michele Lanza

The Class Blueprint - Principles

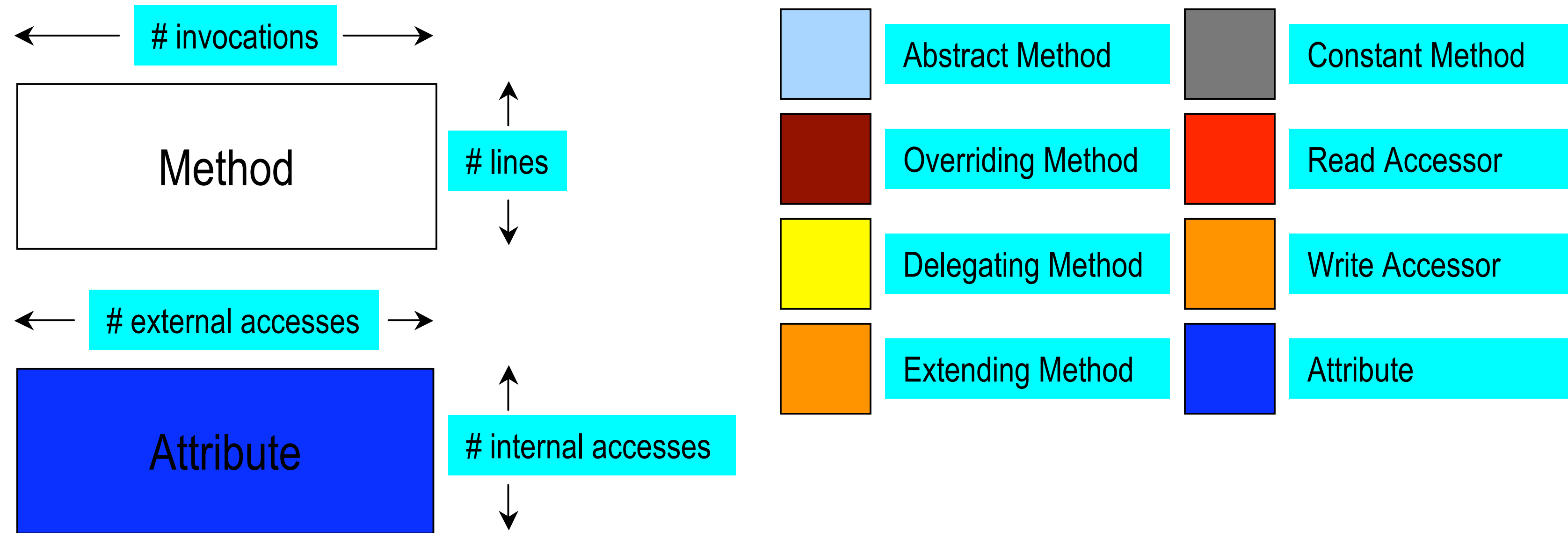


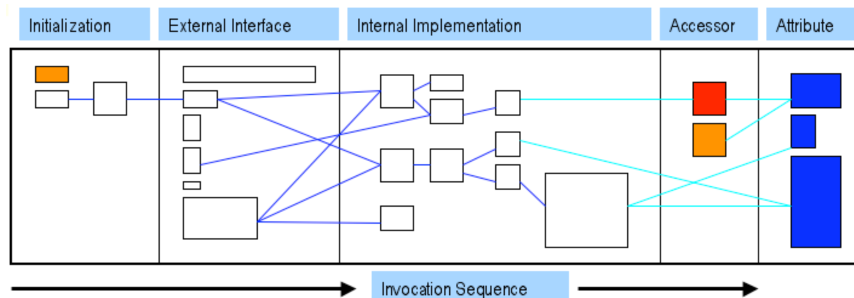
- The class is divided into 5 layers
- Nodes
    - Methods, Attributes, Classes
- Edges
    - Invocation, Access, Inheritance

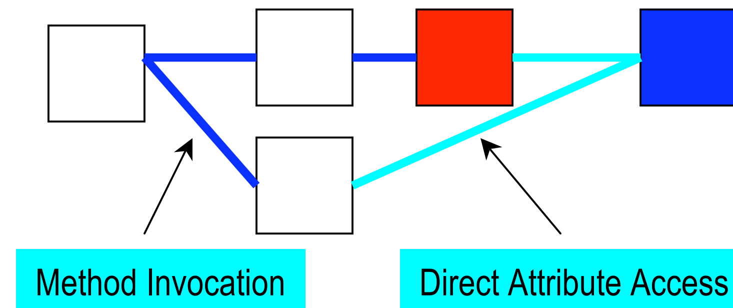
- The method nodes are positioned according to
    - Layer
    - Invocation sequence

© 2006 Harald C. Gall & Michele Lanza

The Class Blueprint - Principles (II)

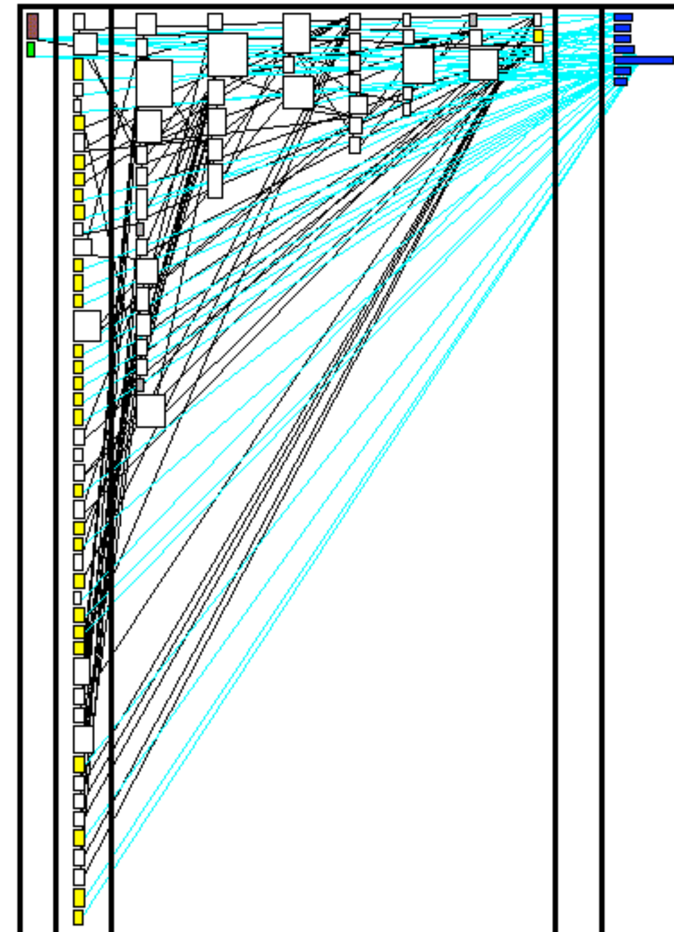






The Class Blueprint - Example

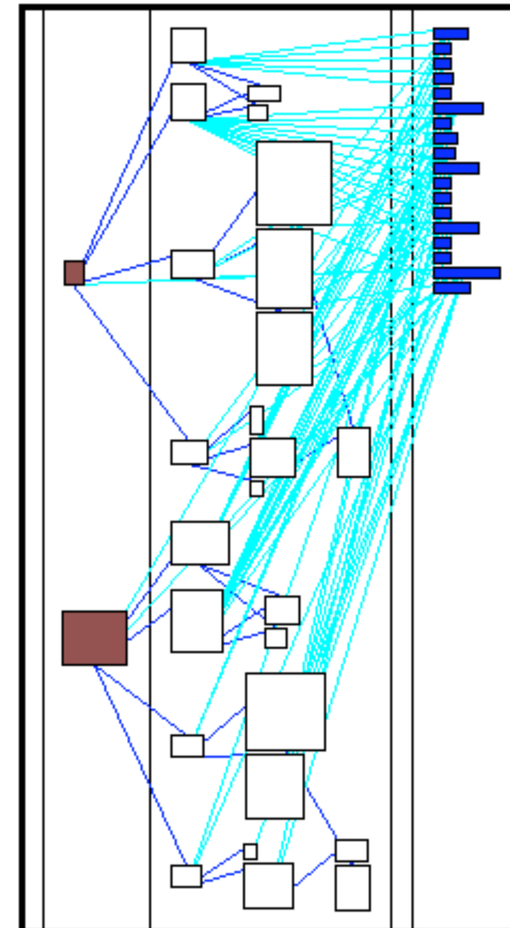
- Delegate:
  - Delegates functionality to other classes
  - May act as a “Façade” (DP)
- Large Implementation:
  - Deep invocation structure
  - Several methods
  - High decomposition
- Wide Interface
- Direct Access
- Sharing Entries



© 2006 Harald C. Gall & Michele Lanza

The Class Blueprint - Example (II)

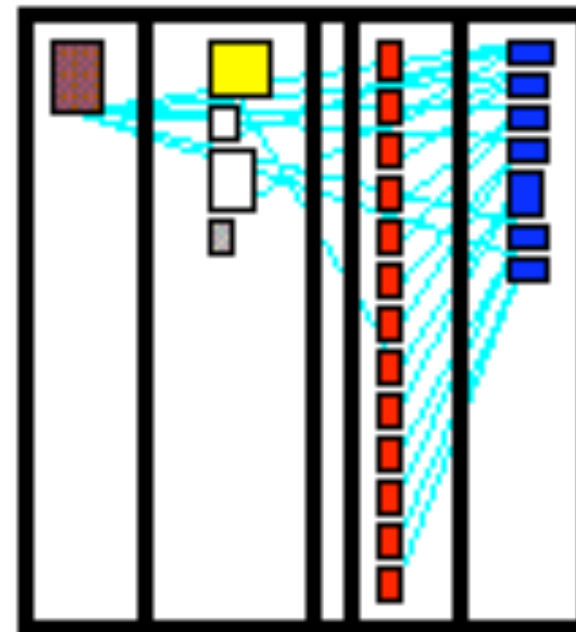
- Call-flow
  - Double Single Entry
  - (=> split class?)
- Inheritance
  - Adder
  - Interface overriders
- Semantics
  - Direct Access
- State Usage
  - Sharing Entries



 © 2006 Harald C. Gall & Michele Lanza

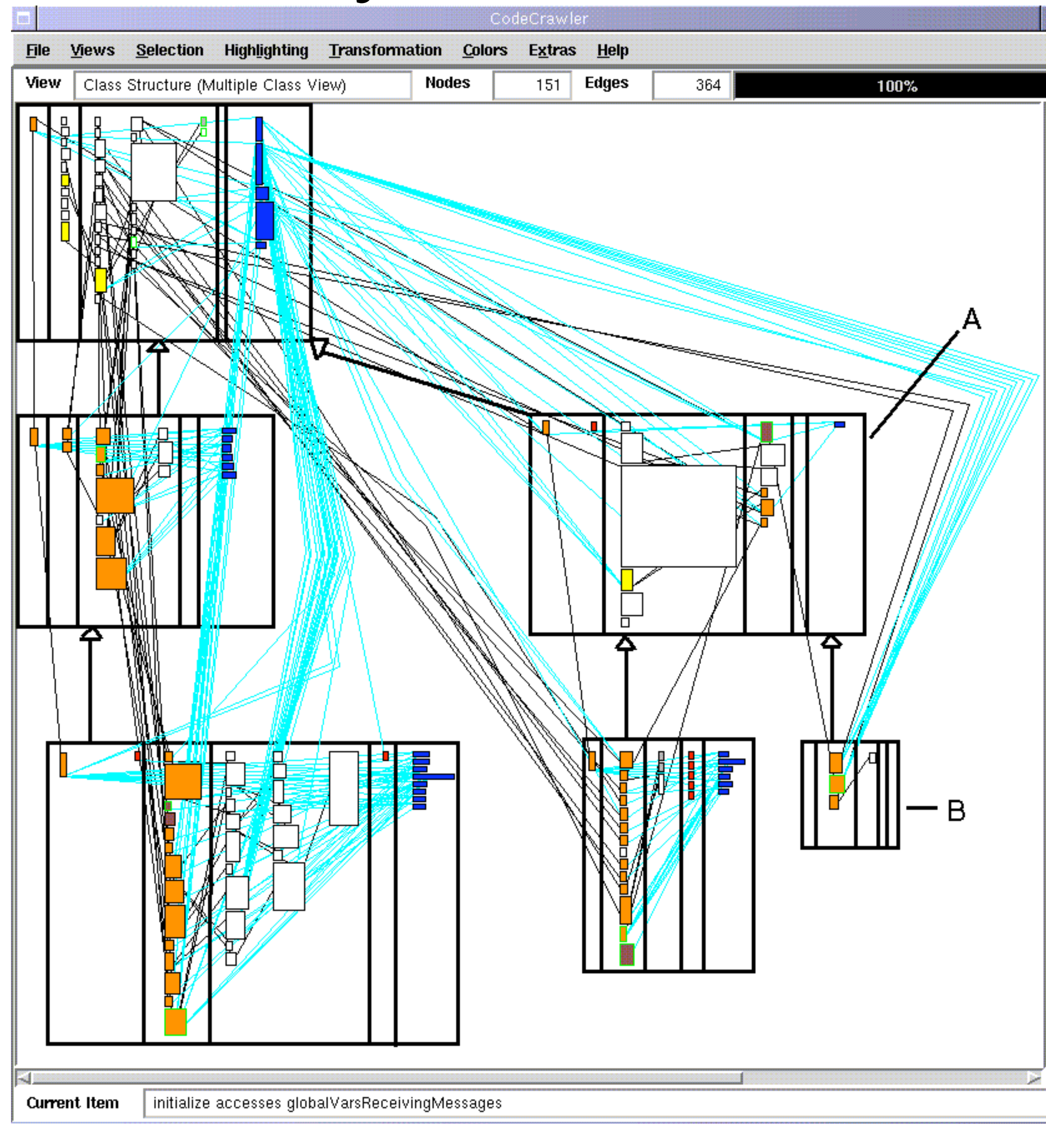
Class Blueprint: Data Storage

- Has many attributes
- May have many accessor methods
- No complex behavior
  - No internal implementation!



© 2006 Harald C. Gall & Michele Lanza

Class Blueprint: Inheritance Policy Breach

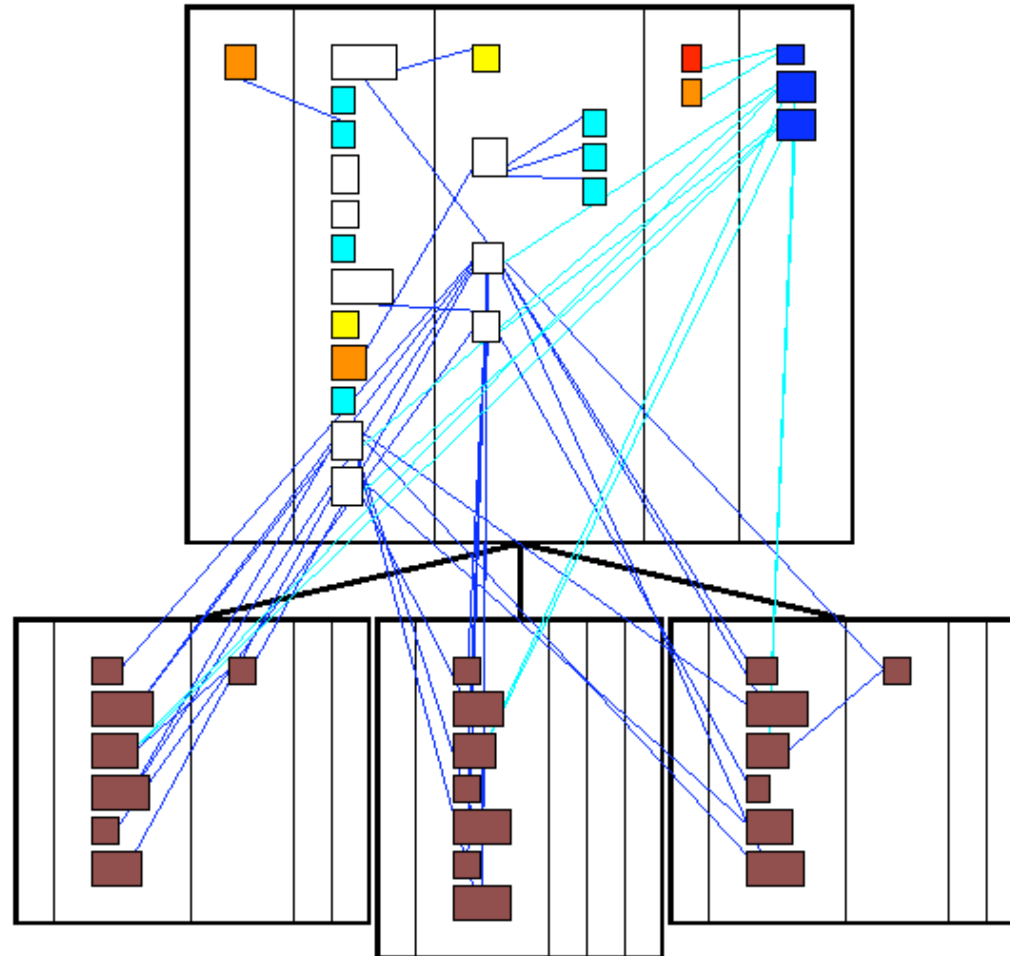


The Class Blueprint - A Pattern Language?

- The patterns reveal information about
  - Coding style
  - Coding policies
  - Particularities
- We grouped them according to
  - Size
  - Layer distribution
  - Semantics
  - Call-flow
  - State usage
- Moreover…
  - Inheritance Context
  - Frequent pattern combinations
  - Rare pattern combinations
- They are all part of a *pattern language*

© 2006 Harald C. Gall & Michele Lanza

The Class Blueprint - What do we see?



© 2006 Harald C. Gall & Michele Lanza

CodeCrawler Demo

CodeCrawler – System Complexity View

CodeCrawler Moose Apply View Spawn View Selection Transformation Colors Layout

Item Class Root::Smalltalk::CodeCrawler::CodeCrawler II NOA: 5 I NOM: 74 I WLOC: 548 I -: 0 I -: 0 I

77 Nodes, 61 Edges - 0 selected Nodes

Fine-grained SV - Conclusions

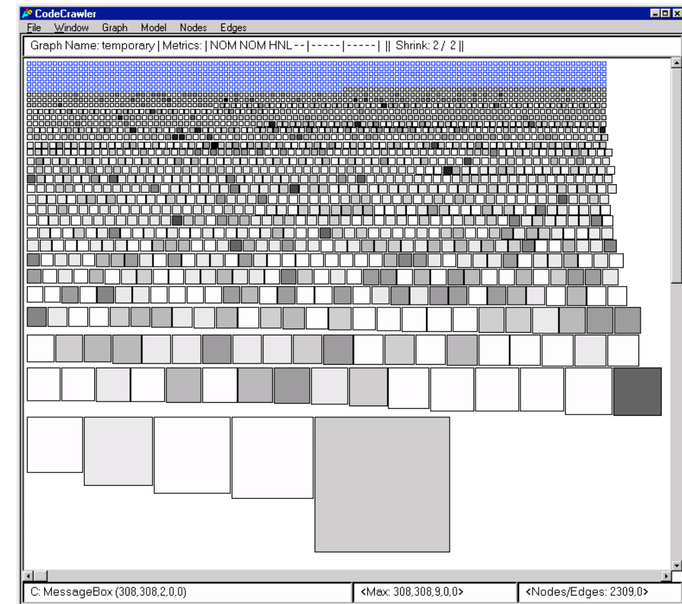
- Benefits
  - Complexity reduction
  - Visual code inspection technique
  - Complements the coarse-grained views
- Limits
  - Visual language must be learned
  - Good object-oriented knowledge required
  - No information about actual functionality => opportunistic code reading necessary

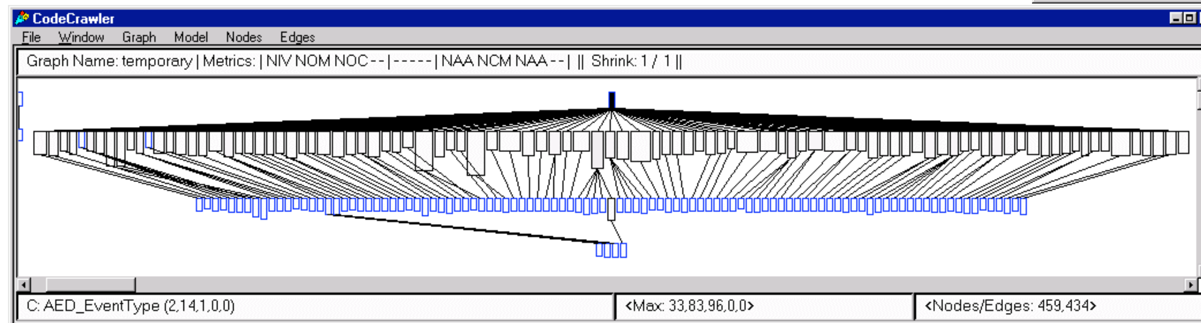
© 2006 Harald C. Gall & Michele Lanza

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++





© 2006 Harald C. Gall & Michele Lanza

Industrial Validation - The Acid Test

- Several large, industrial case studies (NDA)
- Different implementation languages
- Severe time constraints

System	Language	Lines of Code	Classes
Z	C++	1’200’000	~2300
Y	C++/Java	120’000	~400
X	Smalltalk	600’000	~2500
W	COBOL	40’000	-
Sortie	C/C++	28’000	~70
Duploc	Smalltalk	32’000	~230
Jun	Smalltalk	135’000	~700

© 2006 Harald C. Gall & Michele Lanza

Software Visualization: Conclusions

- SV 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

 © 2006 Harald C. Gall & Michele Lanza