Lecture 24

Software Visualization and Metrics
Polymetric Views

Today’s Agenda (1)

- Discussion on Practical Applications of Software Evolution Research
  - Concern Graph
  - Delta Debugging
  - Regression Test Selection

Today’s Agenda (2)

- Class presentations
  - Meiru (skeptic)
- Polymetric Views, Lanza et al. TSE 2003
  - Some slides are borrowed from Dr. Michele Lanza at the University of Lugano, Switzerland

Recap of Software Reflexion Model (1)

- Software Reflexion Model allows programmers to iteratively refine their high-level mental model and compare it to source implementation

Recap of Software Reflexion Model (2)

1. Enable a software engineer to produce a reasonable first-cut of a high-level model.
2. Extract a low-level model using a static analysis or a dynamic analysis
3. Enable the engineer to map the high-level model and source code.
4. Then the reflexion model tool computes agreement and disagreement between the high-level model and the source code.

Discussion - Concern Graph

- What are key ideas of FEAT?
-

Discussion - Concern Graph

- How can you use or adopt the key ideas of Concern Graph when you do not have the FEAT tool?s
-

Discussion - Delta Debugging

- What are key ideas of Delta Debugging?
-

Discussion - Delta Debugging

- How can you use the key ideas of Delta Debugging when you need to identify code that is responsible for faulty behavior? (Imagine that you do not have DD tool that can run on your codebase.)
-

Discussion - Regression Testing

- What are key ideas of Orso et al.'s regression test selection?
-

Discussion - Delta Debugging

- How can you use the key ideas of RTS when you need to run regression tests with the RTS tool?

Information Visualization

- Human eyes and brain interpret visual information in order to “react to the world.”

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”

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]

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…

Polymetric Views

- Polymetric Views - A lightweight Visual Approach to Reverse Engineering, Michaele Lanza and Stephane Ducasse, TSE 2003
  - Lightweight software ***visualization*** enriched software ***metrics***
  - Some slides are borrowed from Dr. Lanza.

Motivation

- Large software systems are difficult to understand.
  - ***When*** can you use this system for doing ***what***?
  - ***Why this*** particular ***approach*** of combining visualization and metrics?

When to use this system?

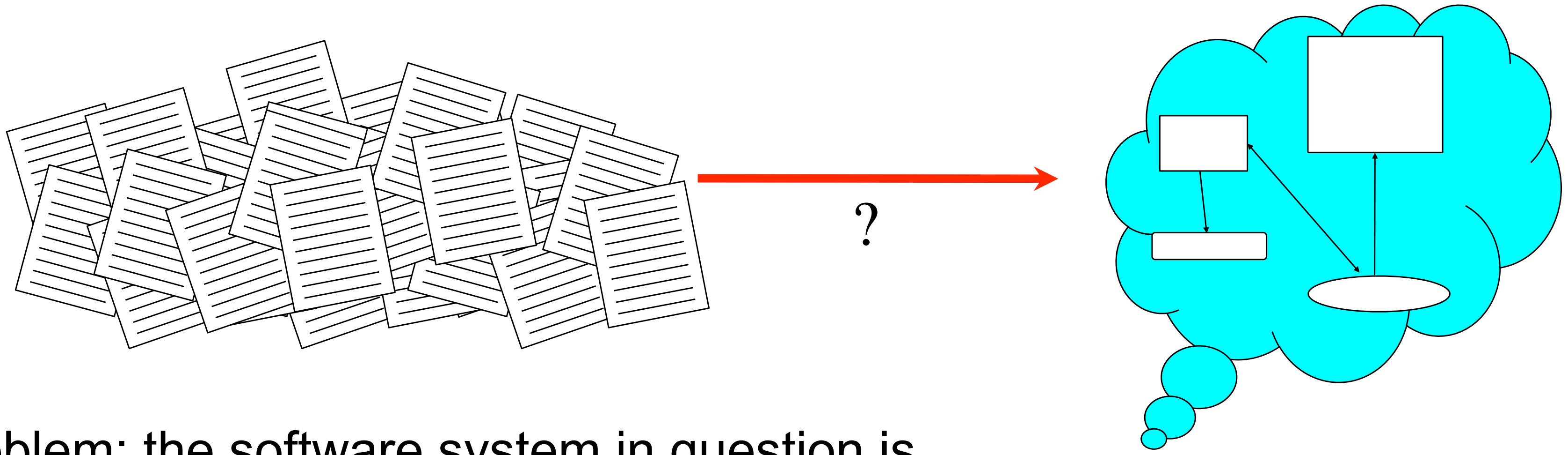
- ***Reengineering*** a ***legacy code***
  - It is not age that turns a piece of software into a legacy system, but the rate at which it has developed and adapted.
  - => Programmers do not know the problem area of this system or where to get started.

For doing ***What***?

- Assess the overall quality of the system and gain an overview of the system in terms of size, complexity, and structure
- Locate and understand the ***most important classes***
- Identify ***exceptional classes*** in terms of size and / or complexity
  - e.g., ***a god class***: a class that has grown over the years ending up with too many responsibilities
  - e.g., ***a long method*** that should be split up into smaller, more reusable methods

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”

Object-Oriented Reverse Engineering (II)

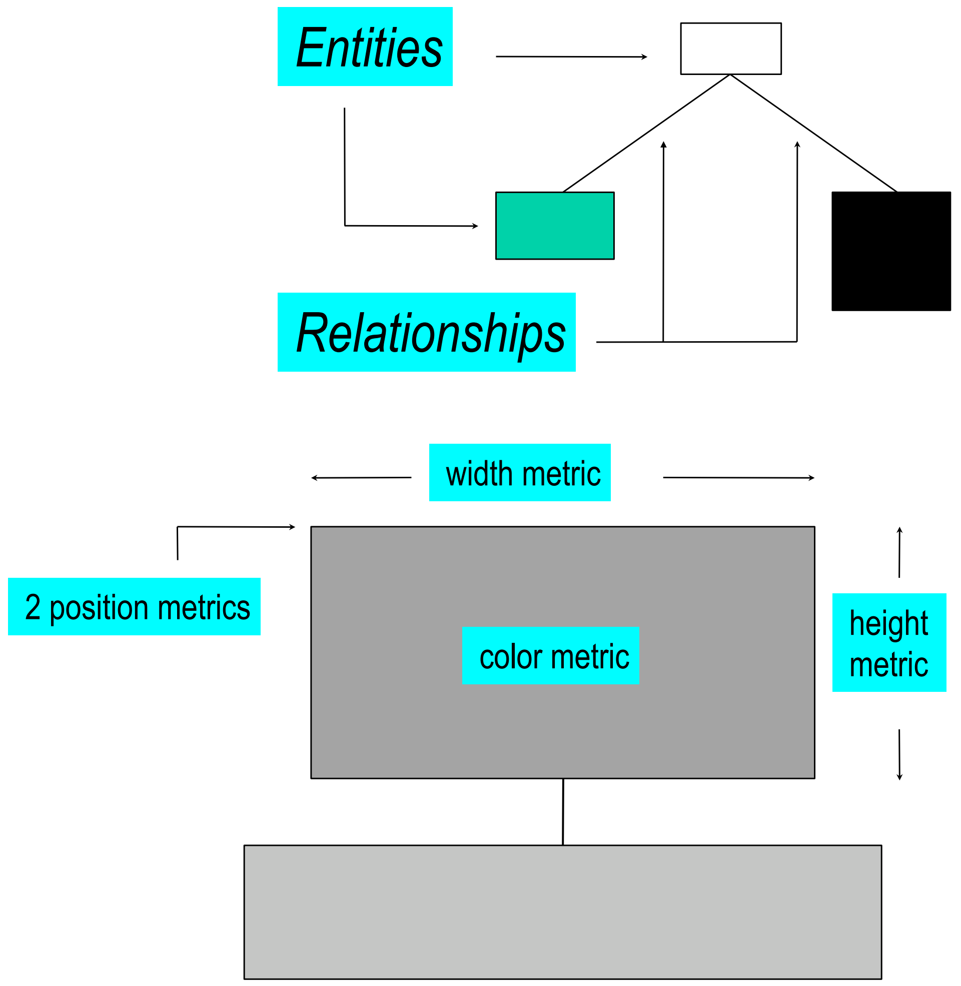
- Constructing a mental model requires *information* about the system:
  - Top-down approaches ⬇
  - Bottom-up approaches ⬆
  - *Mixed Approaches* ⬆⬇
- There is no “silver bullet” methodology
- Every reverse engineering situation is unique
- Need for **flexibility**, **customizability**, **scalability**,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

A simple Solution - The Polymetric View

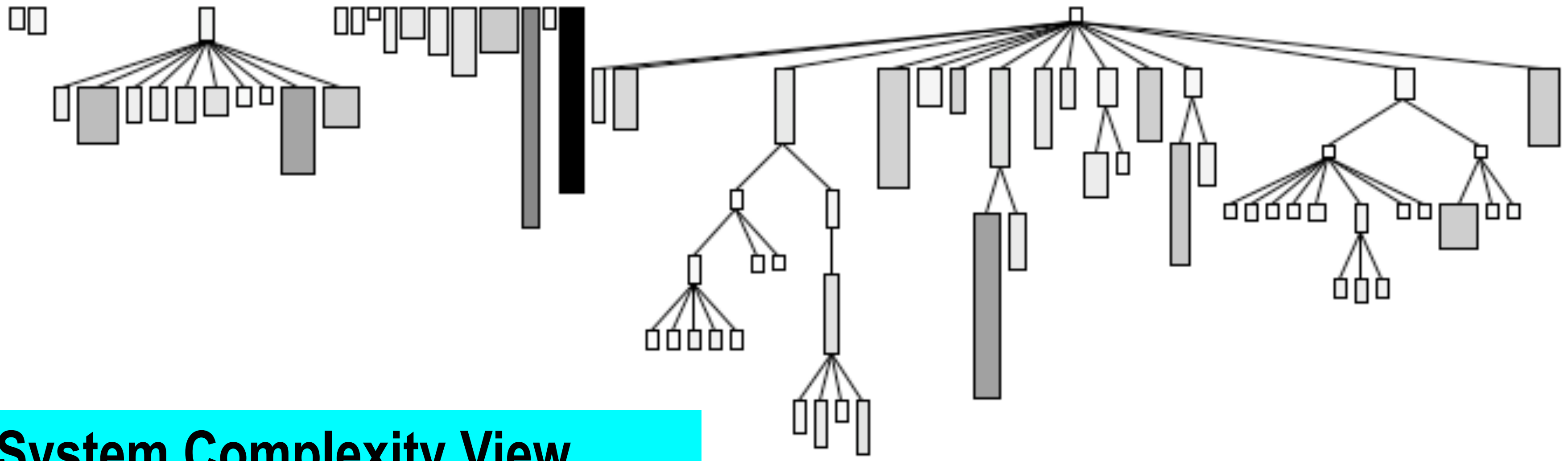
- A lightweight combination of two approaches:
  - Software visualization (reduction of complexity, intuitive)
  - Software metrics (scalability, assessment)
- Interactivity (iterative process, silver bullet impossible)
- Does not replace other techniques, it complements them:
  - “Opportunistic code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

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



The Polymetric View - Example

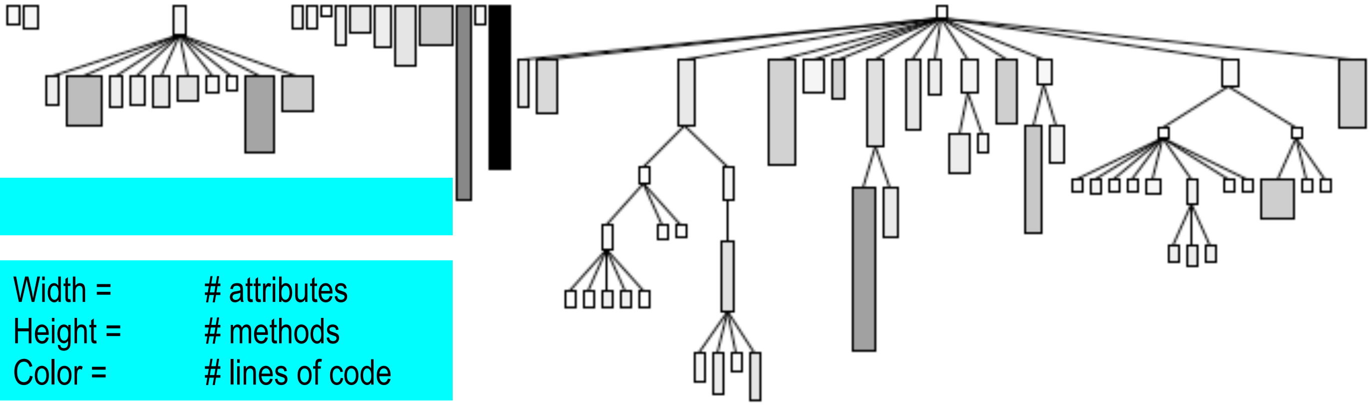


System Complexity View

Nodes = Classes
Edges = Inheritance Relationships

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

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

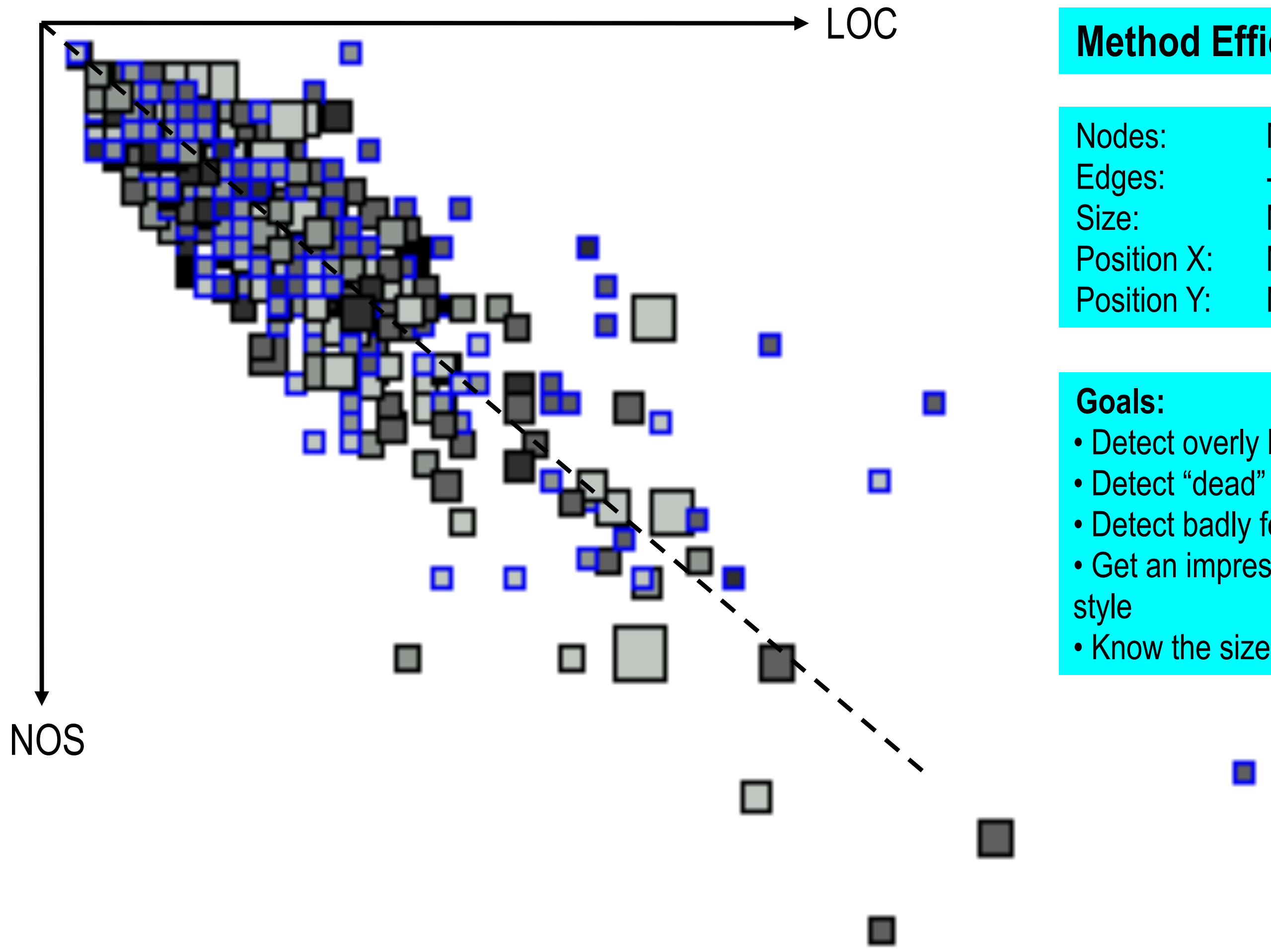
• Count the classes, look at the displayed nodes, count the hierarchies

• 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

Shanghai 2006 ICSE

Coarse-grained Polymetric Views - Example

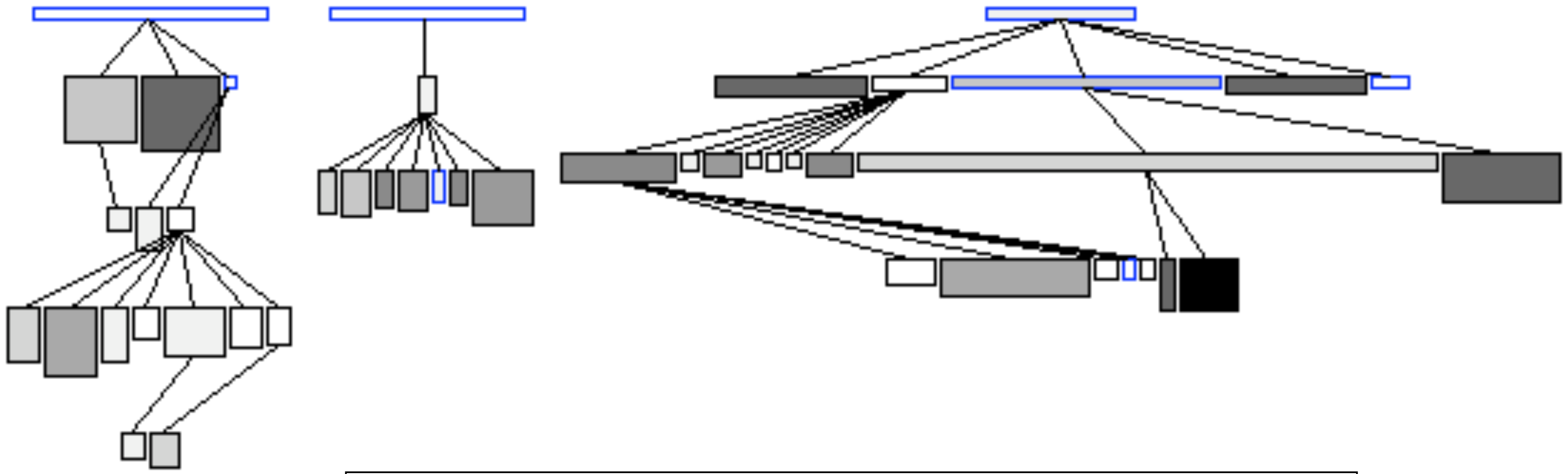


Method Efficiency Correlation View

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

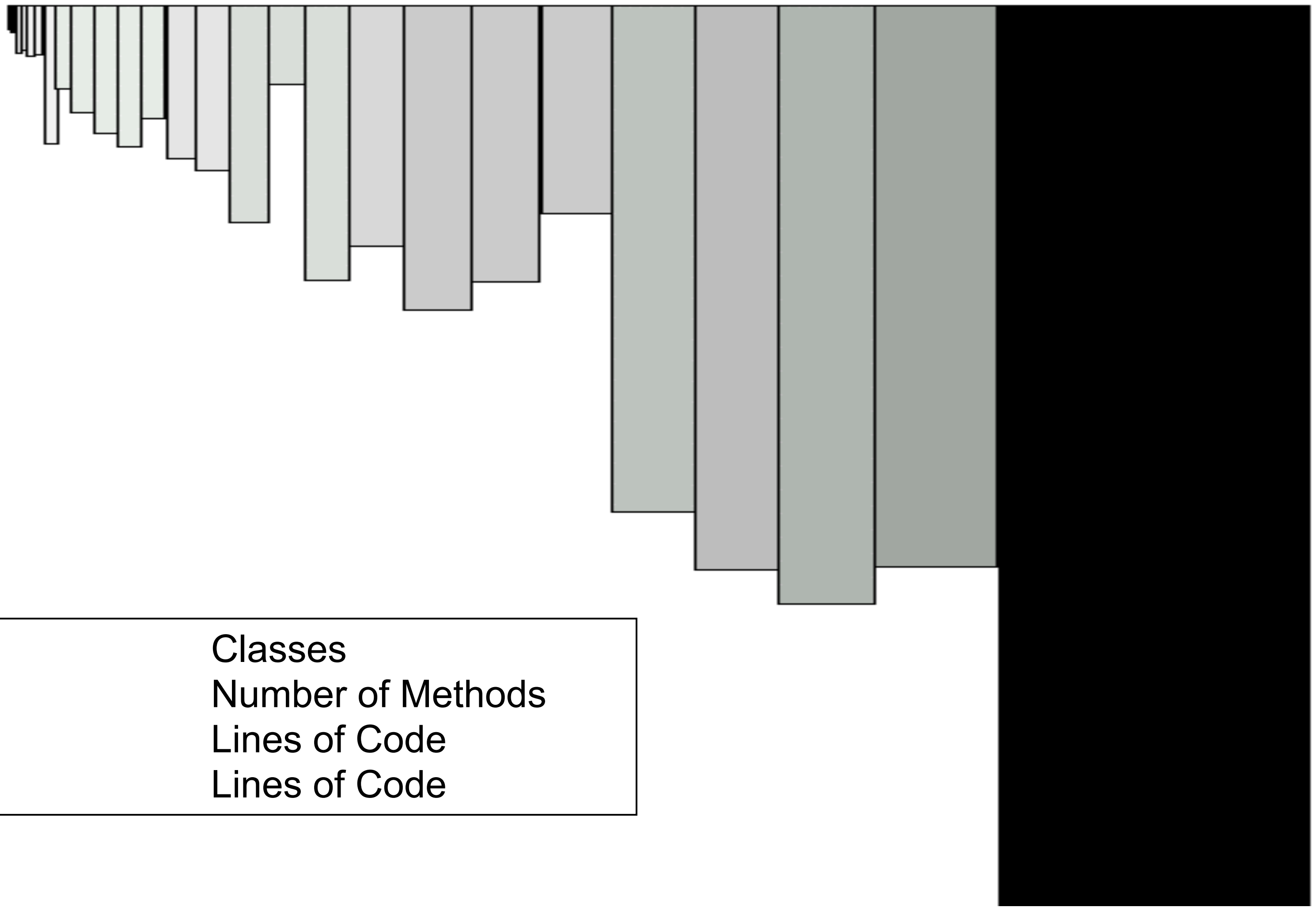
Goals:
- Detect overly long methods
- Detect “dead” code
- Detect badly formatted methods
- Get an impression of the system in terms of coding style
- Know the size of the system in # methods

Inheritance Classification View



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

Data Storage Class Detection View



Boxes:	Classes
Width:	Number of Methods
Height:	Lines of Code
Color:	Lines of Code

Evaluation: Industrial Case Study

- 1.2 Million lines in C++
- "The---often initially skeptical---developers were surprised that we had not only gained an overview over such large systems, but had also uncovered many design falws in such a short time."
- "Even though they were aware of at least half of the problems we found, many developers *saw* the complete software system for the first time."

My general thoughts on Polymetric Views

- Pros:
  - Views are customizable
  - Simple approach yet powerful
- Cons:
  - Visual language must be learned

Recap

- Polymetric view is a customizable software visualization tool enriched software metrics.
- This tool targets initial understanding of a legacy system.
- This tool can help programmers develop a high-level mental model.
- It is simple, powerful, scalable, and customizable, but it requires some training to parse these generated views.

Preview for This Wednesday

- We will move onto research on code duplication.
  - Automatic clone detection: CCFinder, Kamiya et al. TSE 2002
  - Empirical studies of clones: Clone genealogies, Kim et al. FSE 2005

Announcement (1)

- Your final report draft is due tomorrow, 9 PM.
- Your grade review period ends on Apr 27th 11:50 PM.
  - If your grade is missing or incorrect, please talk to TA.
  - After 11:50 PM on Apr 28th, everything will be finalized except ones that require grading.

Announcement (2)

- You ***must*** come to class this wednesday **to receive your assignments for peer reviews.**
- **Your assignment** (discussion on practical uses of software evolution research, part 2) is handed out today and **will be due on Monday, April 27th.**