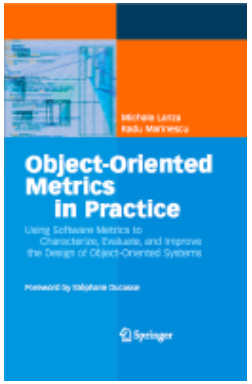
Using Metrics in SQA

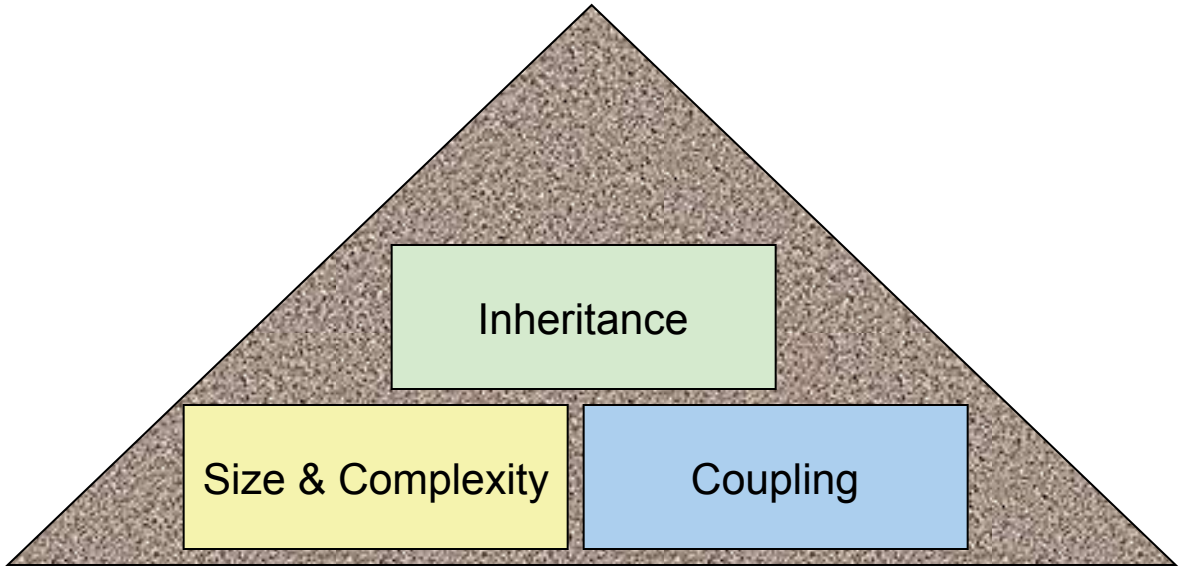
Using Software Metrics to analyze the implementation and design of object-oriented systems



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

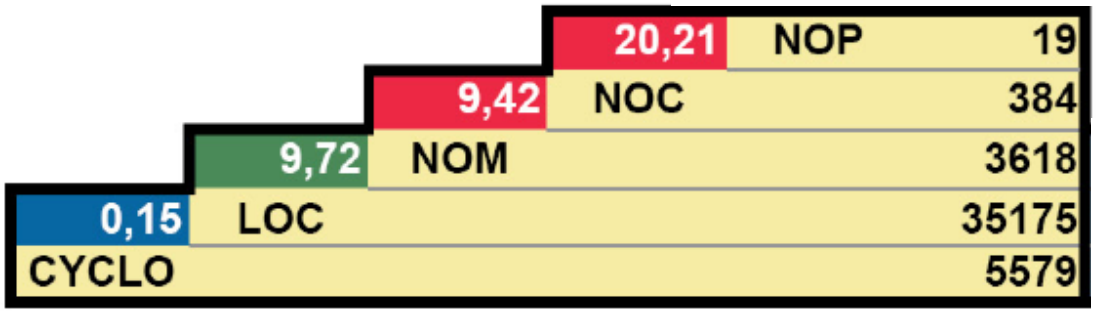
The Metrics Pyramid

- A metrics-based means to both describe and characterize the structure of an object-oriented system by quantifying its ***complexity***, ***coupling*** and ***usage of inheritance***
- Measuring these 3 aspects at system level provides a comprehensive ***characterization*** of an entire system



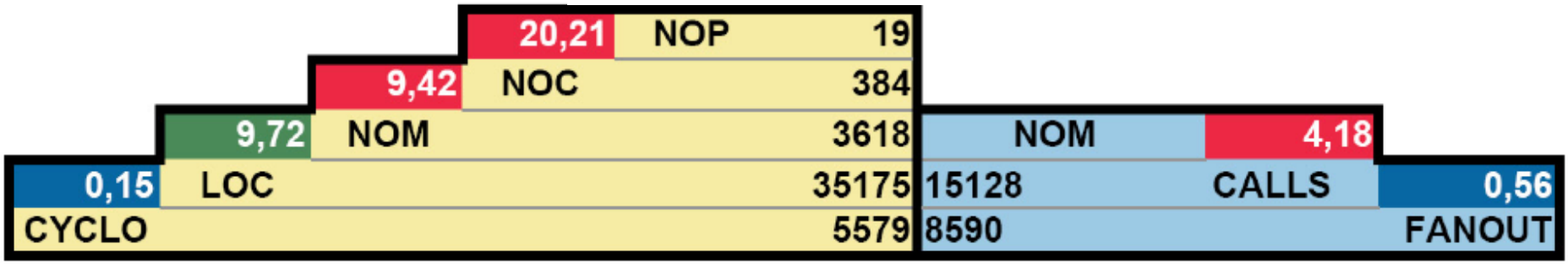
The Metrics Pyramid in Detail

- The left side: System Size & Complexity
  - Direct metrics: NOP, NOC, NOM, LOC, CYCLO
  - Derived metrics: NOC/P, NOM/C, LOC/M, CYCLO/LOC



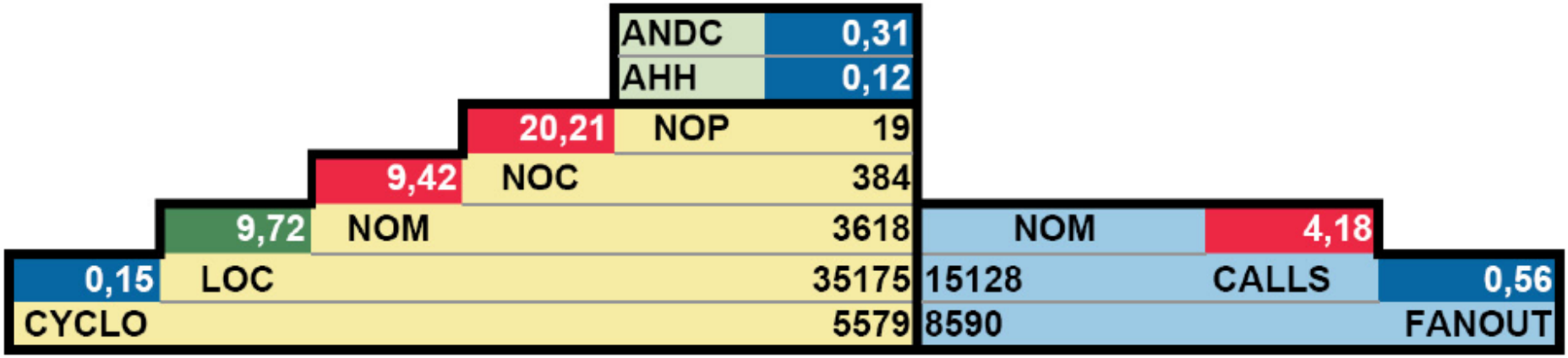
The Overview Pyramid in Detail

- The left side: System Size & Complexity
  - Direct metrics: NOP, NOC, NOM, LOC, CYCLO
  - Derived metrics: NOC/P, NOM/C, LOC/M, CYCLO/LOC
- The right side: System Coupling
  - Direct metrics: CALLS, FANOUT
  - Derived metrics: CALLS/M, FANOUT/CALL



The Overview Pyramid in Detail

- The left side: System Size & Complexity
  - Direct metrics: NOP, NOC, NOM, LOC, CYCLO
  - Derived metrics: NOC/P, NOM/C, LOC/M, CYCLO/LOC
- The right side: System Coupling
  - Direct metrics: CALLS, FANOUT
  - Derived metrics: CALLS/M, FANOUT/CALL
- The top: System Inheritance
  - Direct metrics: ANDC, AHH

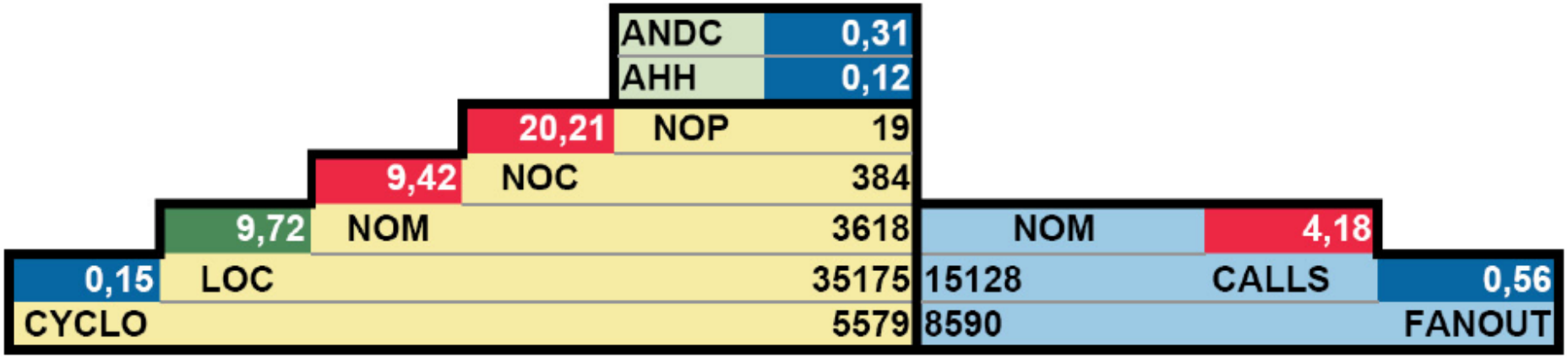


Metrics listed

- NOP - *Number Of Packages*
- *NOC - Number Of Classes*
- *NOM - Number Of Methods*
- *LOC – Lines of Code*
- CYCLO - *Cyclomatic complexity*
- CALLS - number of distinct function- and method-calls
- ANDC - Average Number of Derived Classes
- AHH - Average Hierarchy Height

Interpreting the Overview Pyramid

- The pyramid characterizes a system in terms of size&complexity, coupling, and inheritance; based on 8 computed proportions:
  - They are independent of the size of the system!
  - This enables an objective assessment…
    - Wait a second…objective? Where is the reference point?



Putting things in a real-world context

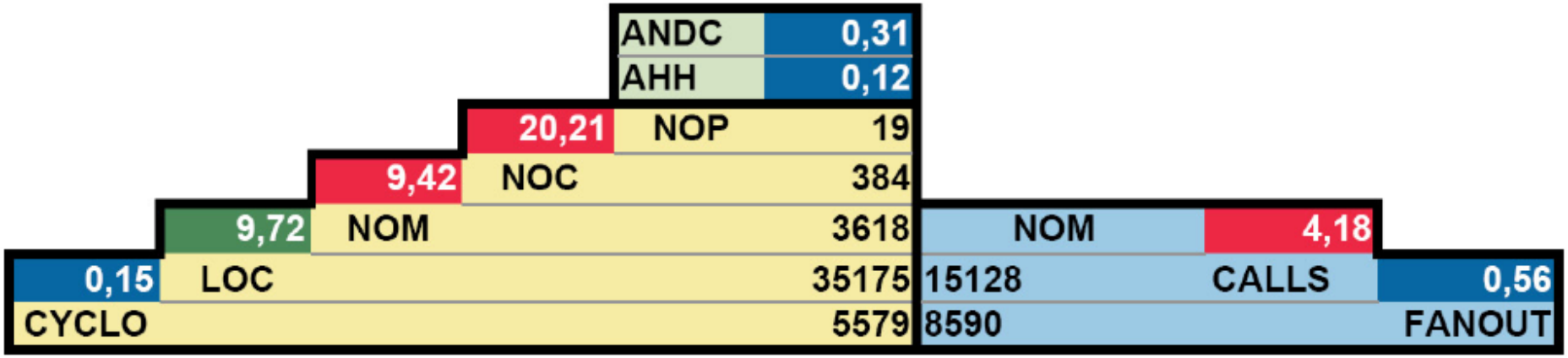
- We measured 80+ systems written in Java and C++
- Based on the obtained measurements we can now statistically assess the design of a system

	Java			C++		
Metric	Low	Average	High	Low	Average	High
CYCLO/Line of code	0.16	0.20	0.24	0.20	0.25	0.30
LOC/Operation	7	10	13	5	10	16
NOM/Class	4	7	10	4	9	15
NOC /Package	6	17	26	3	19	35
CALLS/Operation	2.01	2.62	3.2	1.17	1.58	2
FANOUT /Call	0.56	0.62	0.68	0.20	0.34	0.48
ANDC	0.25	0.41	0.57	0.19	0.28	0.37
AHH	0.09	0.21	0.32	0.05	0.13	0.21

High

Average

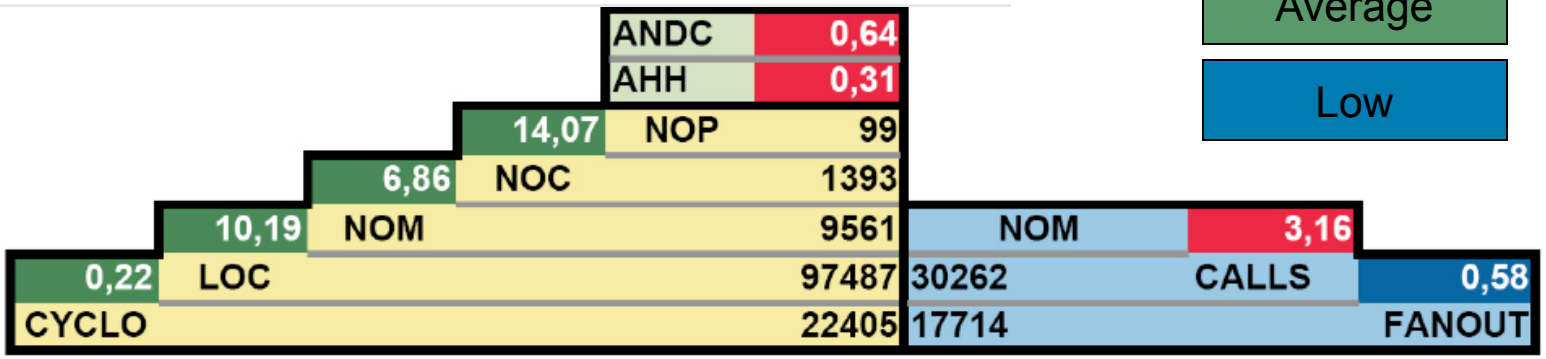
Low



Overview Pyramid Example: ArgoUML

Metric	Value	Remarks
No. of Lines of Code	223,068	including comments
No. of Source Files	1,209	*.java files
No. of Packages	99	–
No. of Classes	1,393	including 140 inner classes
No. of Methods	9,561	including accessor methods
No. of Attributes	3,358	all variables including static and local variables

	Java			C++		
Metric	Low	Average	High	Low	Average	High
CYCLO/Line of code	0.16	0.20	0.24	0.20	0.25	0.30
LOC/Operation	7	10	13	5	10	16
NOM/Class	4	7	10	4	9	15
NOC /Package	6	17	26	3	19	35
CALLS/Operation	2.01	2.62	3.2	1.17	1.58	2
FANOUT /Call	0.56	0.62	0.68	0.20	0.34	0.48
ANDC	0.25	0.41	0.57	0.19	0.28	0.37
AHH	0.09	0.21	0.32	0.05	0.13	0.21



Pattern: Study the Exceptional Entities

Problem

- **How can you quickly gain insight into complex software?**

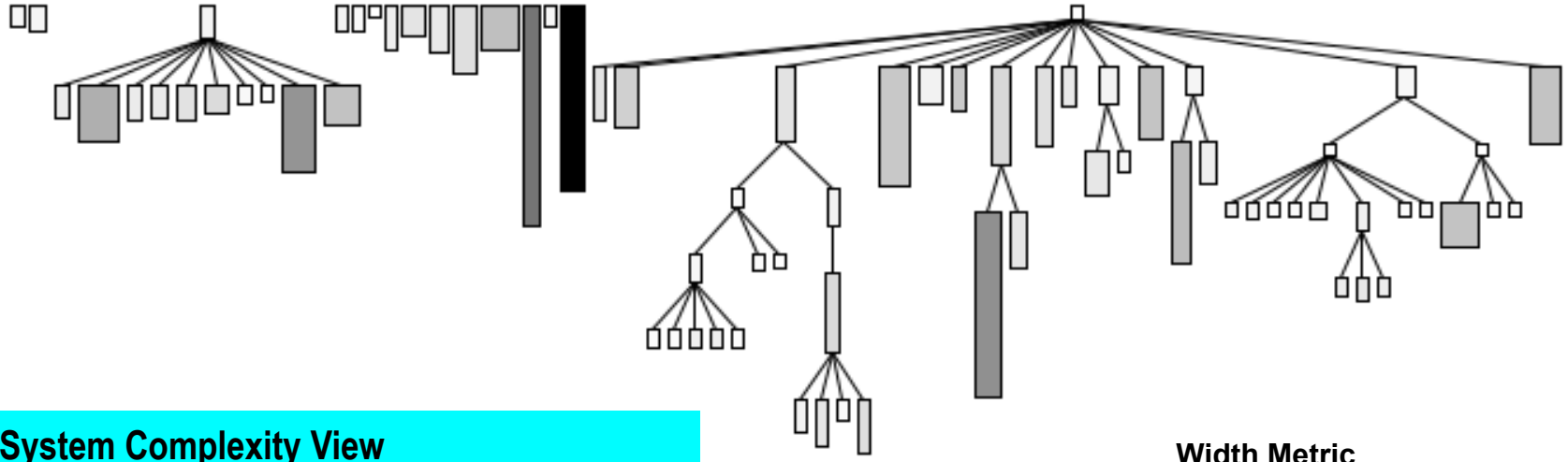
Solution

- *Measure* software entities and *study the anomalous ones*

Steps

- Use simple metrics
- Visualize metrics to get an overview
- Browse the code to get insight into the anomalies

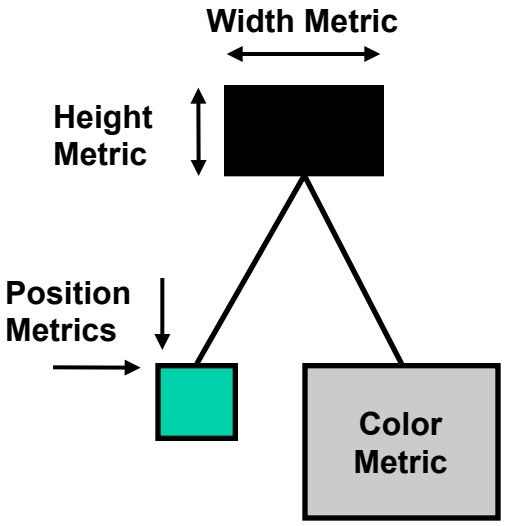
System Complexity View



System Complexity View

Nodes = Classes
Edges = Inheritance Relationships

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



Detection strategy

- A detection strategy is a *metrics-based predicate* to identify *candidate* software artifacts that *conform to* (or violate) a particular *design rule*

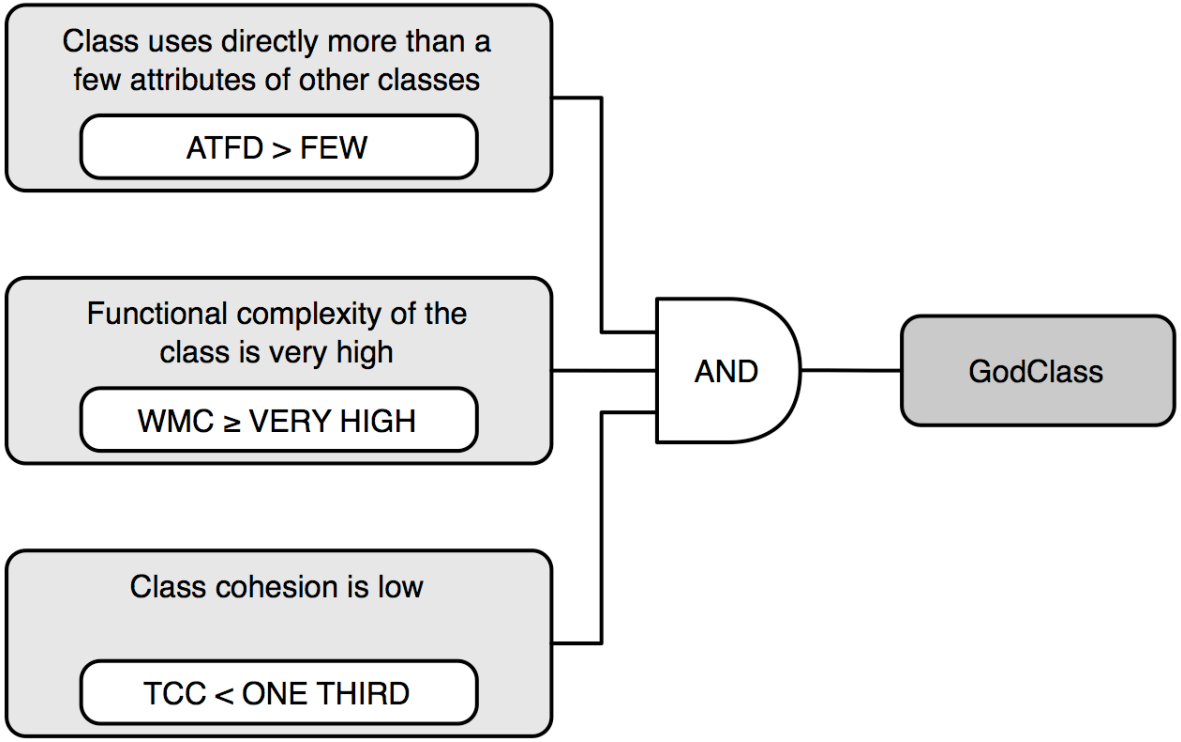
Filters and composition

- A data filter is a predicate used to focus attention on a *subset of interest* of a larger data set
  - Statistical filters
    - I.e., top and bottom 25% are considered outliers
  - Other relative thresholds
    - I.e., other percentages to identify outliers (e.g., top 10%)
  - Absolute thresholds
    - I.e., fixed criteria, independent of the data set

- A useful detection strategy can often be expressed as a *composition* of data filters

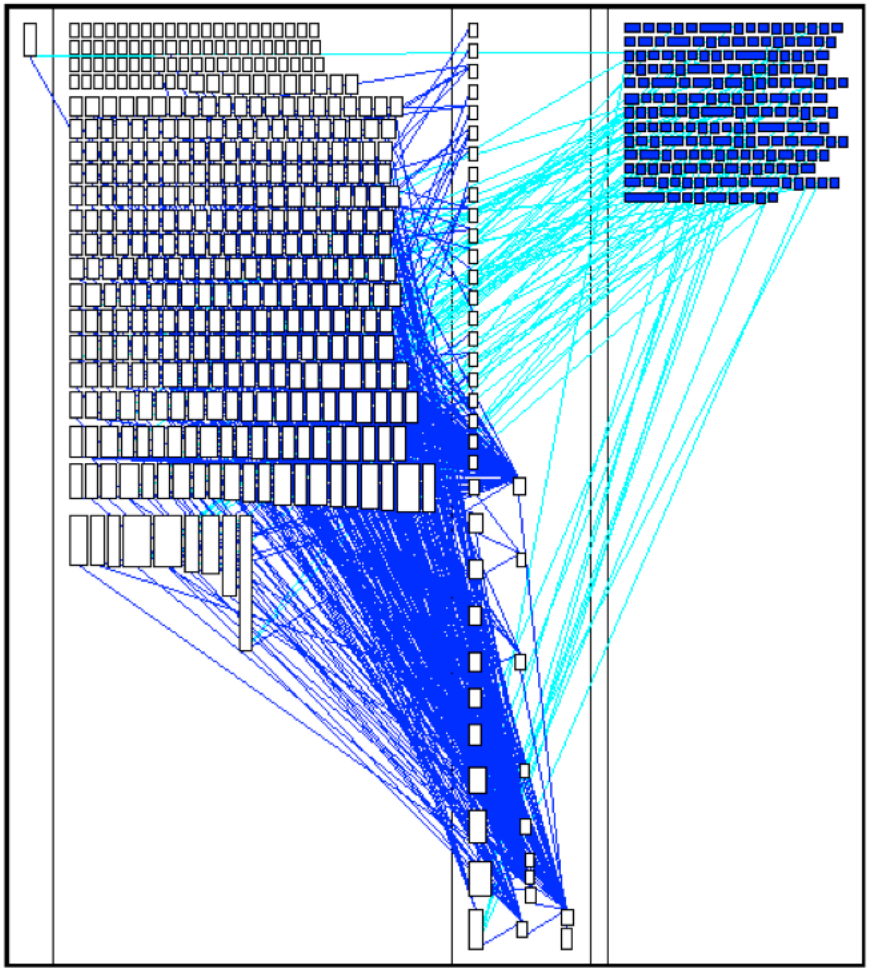
God Class

- A *God Class* centralizes intelligence in the system
  - Impacts understandibility
  - Increases system fragility

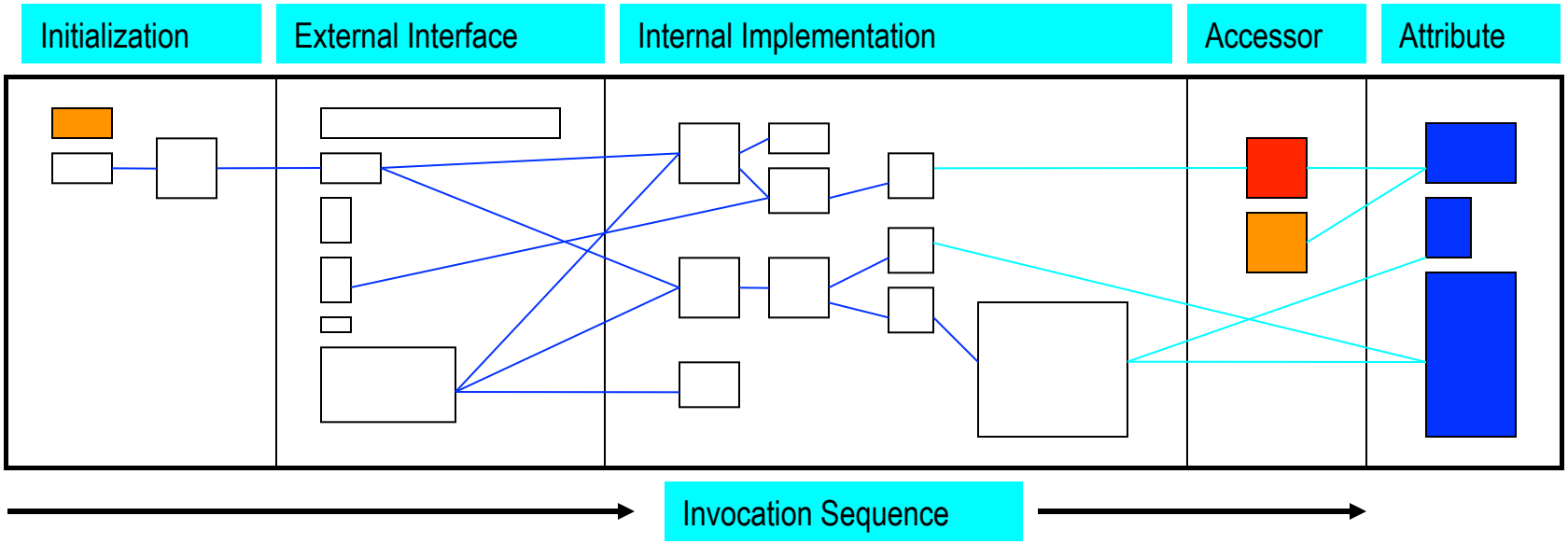


ModelFacade (ArgoUML)

- 453 methods
- 114 attributes
- over 3500 LOC
- all methods and all attributes are static



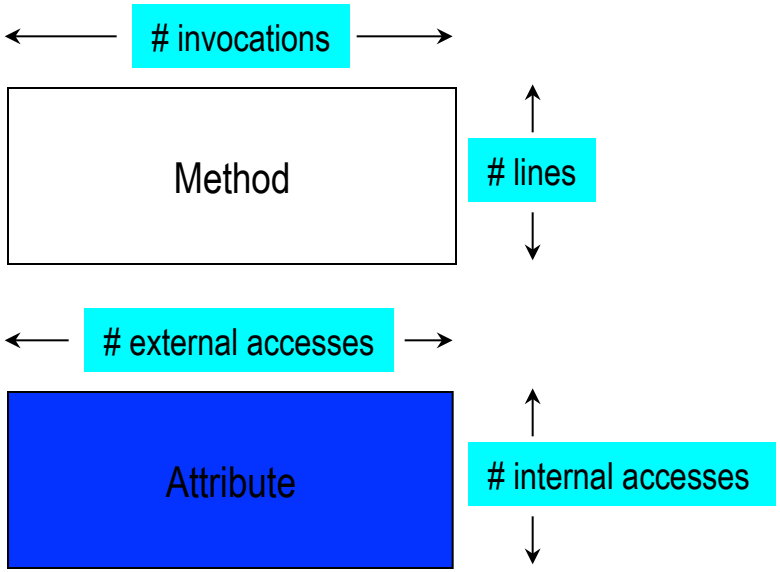
The Class Blueprint - Principles

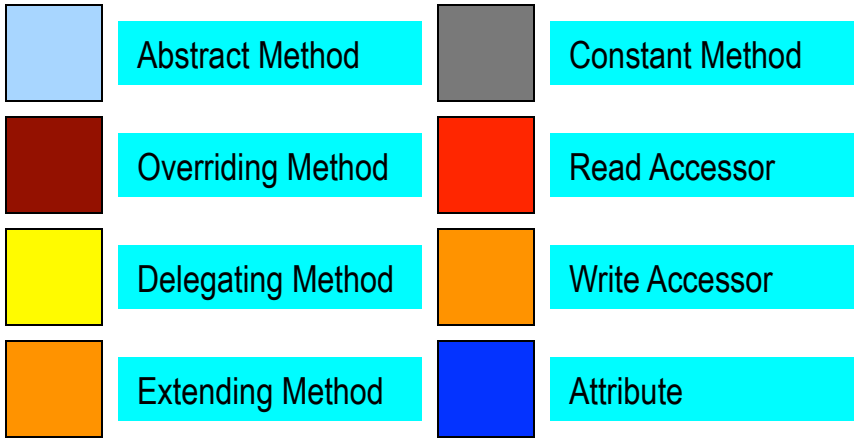


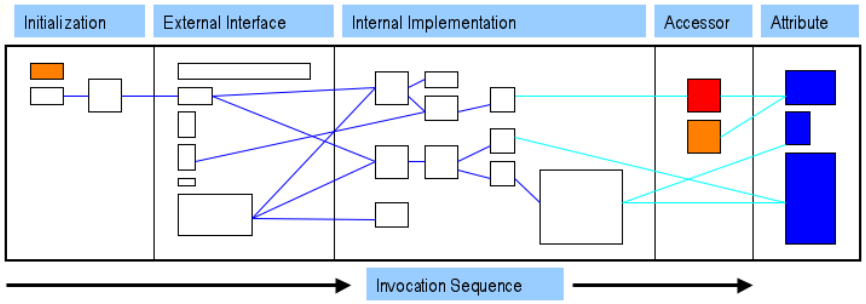
- The class is divided into 5 layers
- Nodes
  - Methods, Attributes, Classes
- Edges
  - Invocation, Access, Inheritance

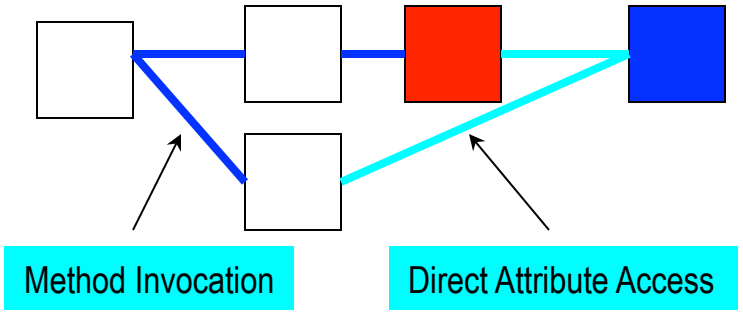
- The method nodes are positioned according to
  - Layer
  - Invocation sequence

The Class Blueprint - Principles (II)



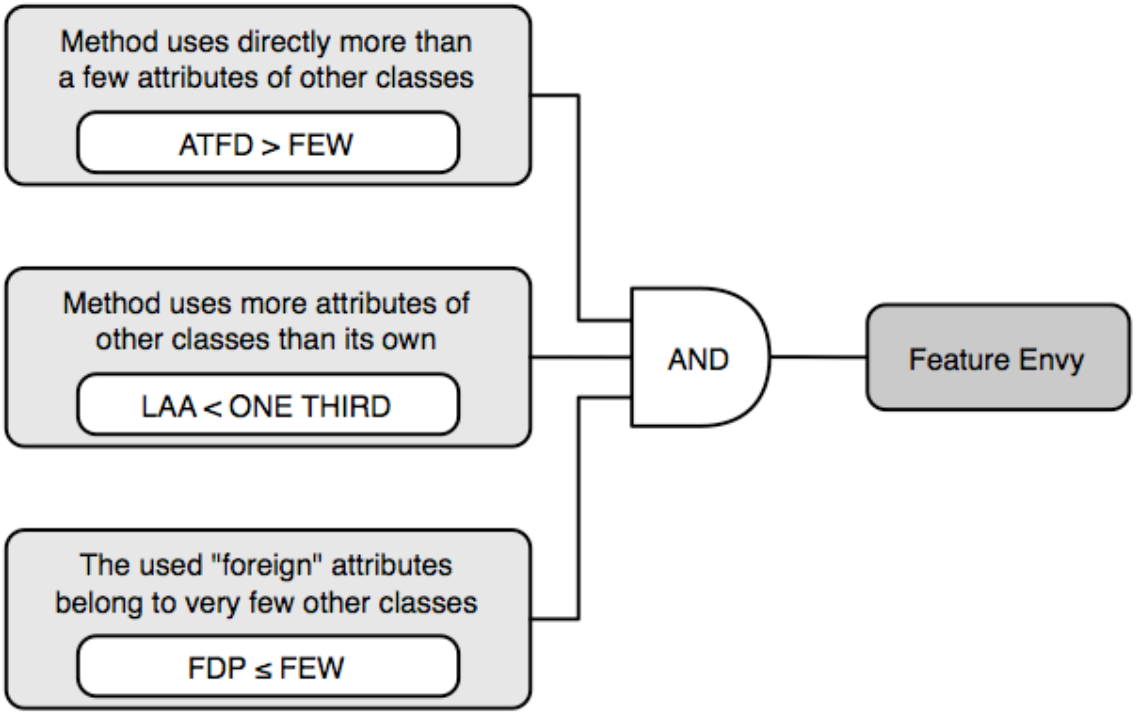




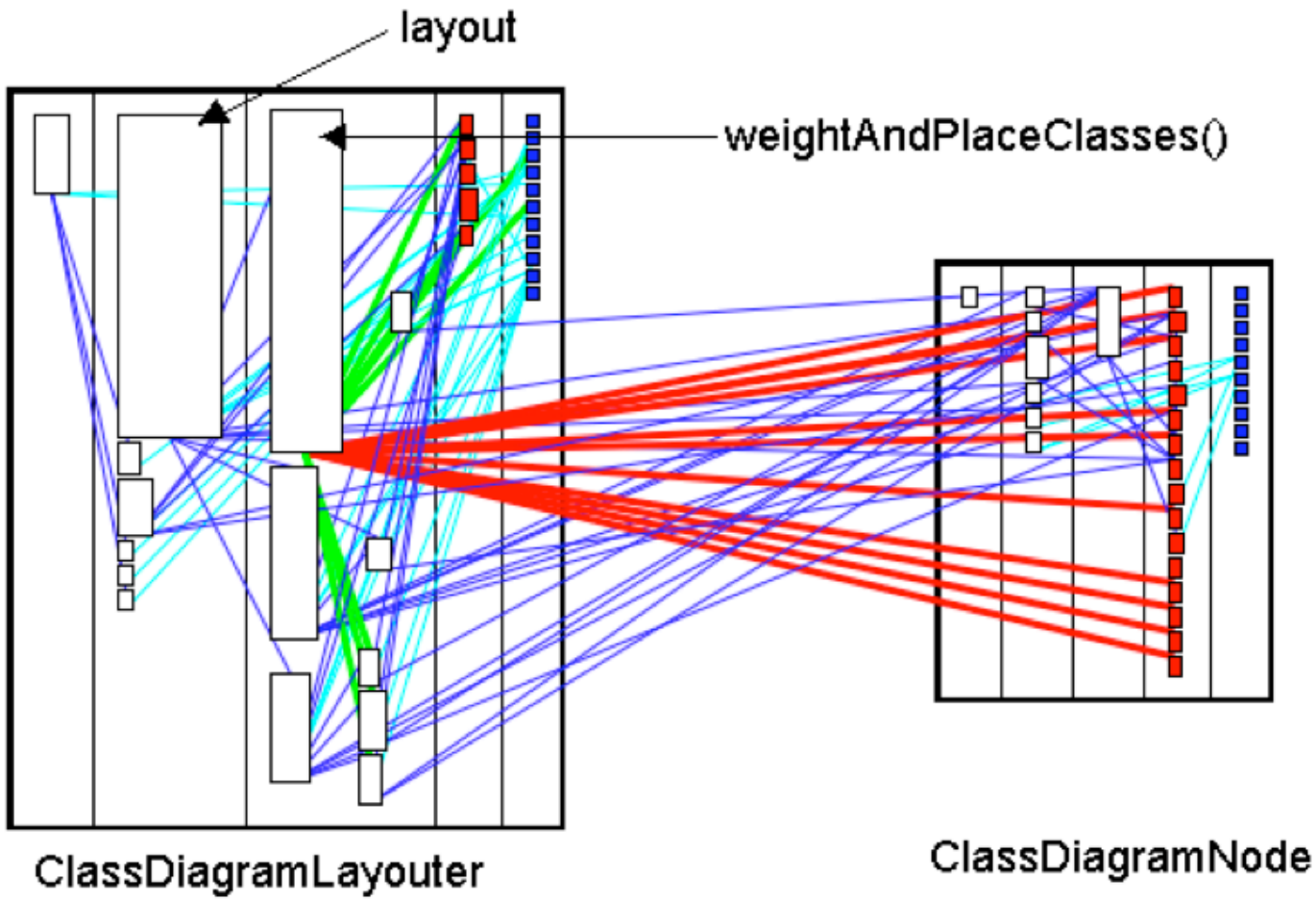


Feature Envy

- Methods that are more interested in data of other classes than their own [Fowler et al. 99]

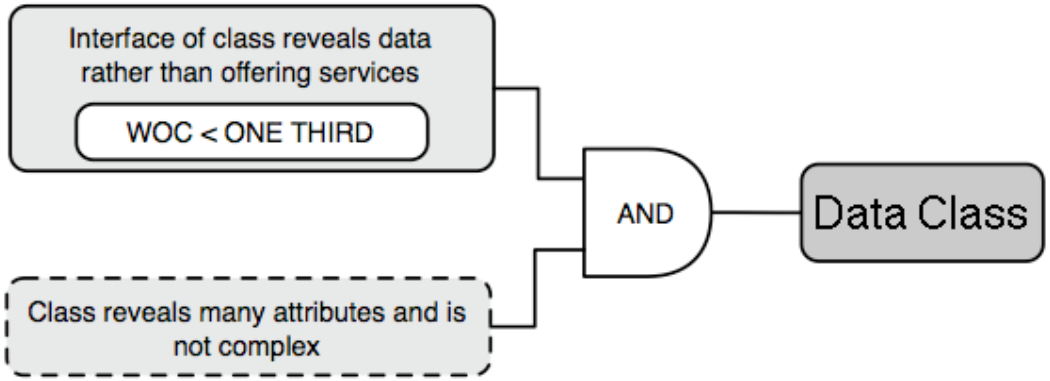


ClassDiagramLayouter

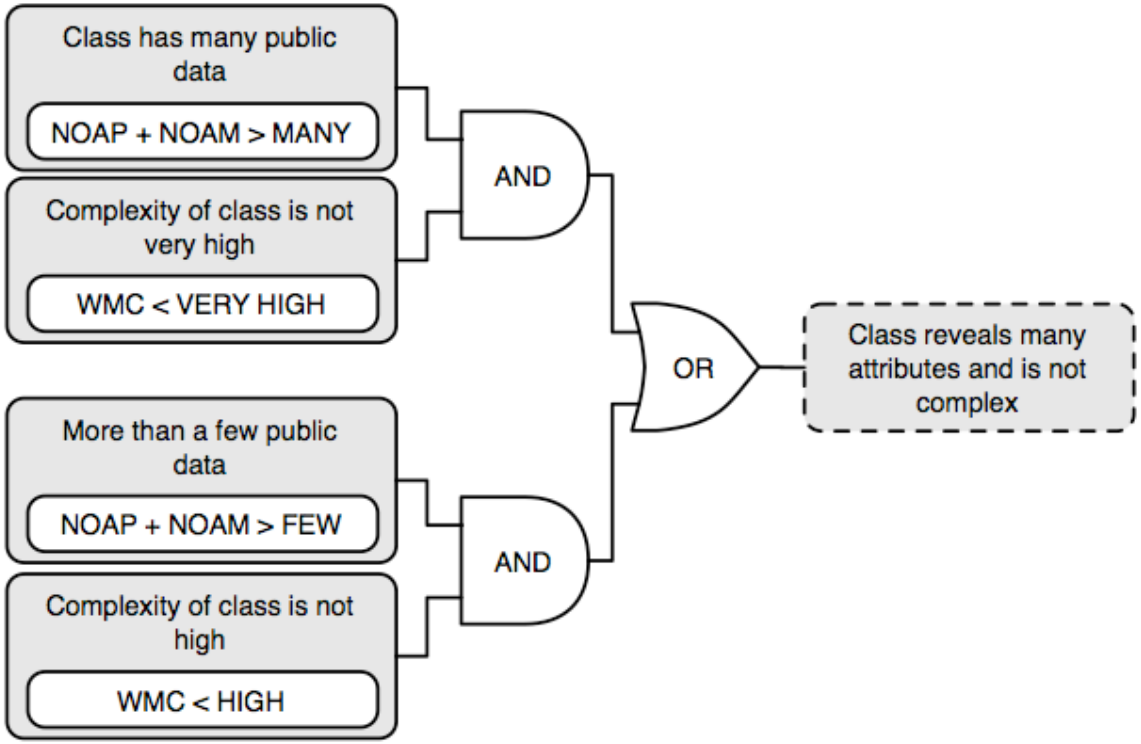


Data Class

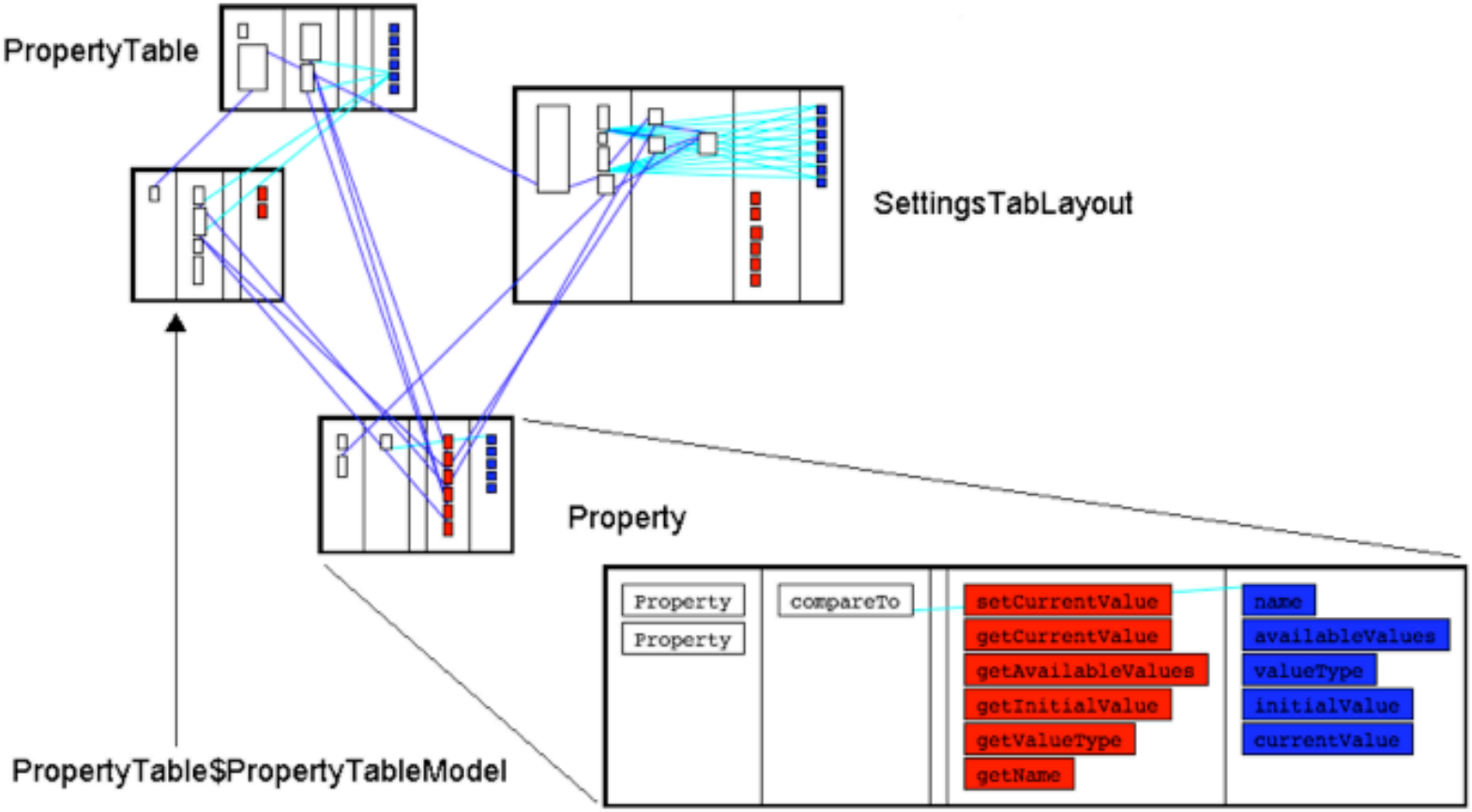
- A Data Class provides data to other classes but little or no functionality of its own



Data Class (2)

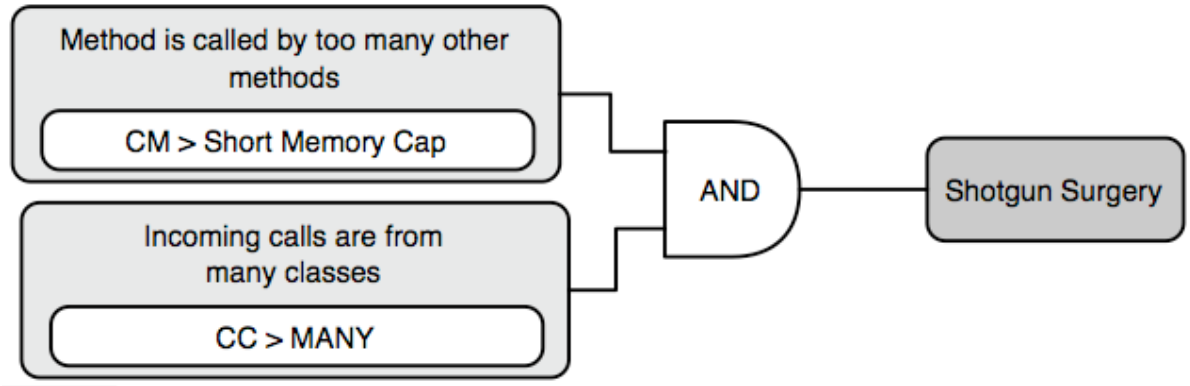

Class has many public data
NOAP + NOAM > MANY
Complexity of class is not very high
WMC < VERY HIGH
AND
More than a few public data
NOAP + NOAM > FEW
Complexity of class is not high
WMC < HIGH
AND
OR
Class reveals many attributes and is not complex

Property


PropertyTable
SettingsTabLayout
Property
PropertyTable$PropertyTableModel
Property
Property
compareTo
setCurrentValue
getCurrentValue
getAvailableValues
getInitialValue
getValueType
getName
name
availableValues
valueType
initialValue
currentValue

Shotgun Surgery

- A change in an operation implies many (small) changes to a lot of different operations and classes



Code Duplication

Code Duplication

a.k.a. Software Cloning, Copy&Paste Programming

- Code Duplication
  - What is it?
  - Why is it harmful?
- Detecting Code Duplication
- Approaches
- A Lightweight Approach
- Visualization (dotplots)
- Duploc

Code is Copied

Small Example from the Mozilla Distribution (Milestone 9)
Extract from /dom/src/base/nsLocation.cpp

```
[432]  NS_IMETHODIMP
[433]  LocationImpl::GetPathname(nsString
[434]  {
[435]    nsAutoString href;
[436]    nsIURI *url;
[437]    nsresult result = NS_OK;
[438]
[439]    result = GetHref(href);
[440]    if (NS_OK == result) {
[441]  #ifndef NECKO
[442]      result = NS_NewURL(&url, href);
[443]  #else
[444]      result = NS_NewURI(&url, href);
[445]  #endif // NECKO
[446]      if (NS_OK == result) {
[447]  #ifdef NECKO
[448]        char* file;
[449]        result = url->GetPath(&file);
[450]  #else
[451]        const char* file;
[452]        result = url->GetFile(&file);
[453]  #endif
[454]        if (result == NS_OK) {
[455]          aPathname.SetString(file);
[456]  #ifdef NECKO
[457]          nsCRT::free(file);
[458]  #endif
[459]        }
[460]        NS_IF_RELEASE(url);
[461]      }
[462]    }
[463]
[464]    return result;
[465]  }
[466]
```

```
[467]  NS_IMETHODIMP
[468]  LocationImpl::SetPathname(const nsString
[469]  {
[470]    nsAutoString href;
[471]    nsIURI *url;
[472]    nsresult result = NS_OK;
[473]
[474]    result = GetHref(href);
[475]    if (NS_OK == result) {
[476]  #ifndef NECKO
[477]      result = NS_NewURL(&url, href);
[478]  #else
[479]      result = NS_NewURI(&url, href);
[480]  #endif // NECKO
[481]      if (NS_OK == result) {
[482]        char *buf = aPathname.ToNewCString();
[483]  #ifdef NECKO
[484]        url->SetPath(buf);
[485]  #else
[486]        url->SetFile(buf);
[487]  #endif
[488]        SetURL(url);
[489]        delete[] buf;
[490]        NS_RELEASE(url);
[491]      }
[492]    }
[493]
[494]    return result;
[495]  }
[496]
```

```
[497]  NS_IMETHODIMP
[498]  LocationImpl::GetPort(nsString& aPort)
[499]  {
[500]    nsAutoString href;
[501]    nsIURI *url;
[502]    nsresult result = NS_OK;
[503]
[504]    result = GetHref(href);
[505]    if (NS_OK == result) {
[506]  #ifndef NECKO
[507]      result = NS_NewURL(&url, href);
[508]  #else
[509]      result = NS_NewURI(&url, href);
[510]  #endif // NECKO
[511]      if (NS_OK == result) {
[512]        aPort.SetLength(0);
[513]  #ifdef NECKO
[514]        PRInt32 port;
[515]        (void)url->GetPort(&port);
[516]  #else
[517]        PRUint32 port;
[518]        (void)url->GetHostPort(&port);
[519]  #endif
[520]        if (-1 != port) {
[521]          aPort.Append(port, 10);
[522]        }
[523]        NS_RELEASE(url);
[524]      }
[525]    }
[526]
[527]    return result;
[528]  }
[529]
```

How Much Code is Duplicated?

Usual estimates: 8 to 12% in normal industrial code
15 to 25 % is already a lot!

Case Study	***LOC***	***Duplication without comments***	***with comments***
gcc	460'000	8.7%	5.6%
Database Server	245'000	36.4%	23.3%
Payroll	40'000	59.3%	25.4%
Message Board	6'500	29.4%	17.4%

What Is Considered To Be Copied Code?

Duplicated Code = *Source code segments that are found in different places of a system*.

- in different files
- in the same file but in different functions
- in the same function

The segments must contain some *logic or structure* that can be abstracted, i.e.,

```
...
computeIt(a,b,c,d);
...
```

```
...
computeIt(w,x,y,z);
...
```

is not considered duplicated code.

```
...
getIt(hash(tail(z)));
...
```

```
...
getIt(hash(tail(a)));
...
```

could be abstracted to a new function

Copied artifacts range from expressions, to functions, to data structures, and to entire subsystems.

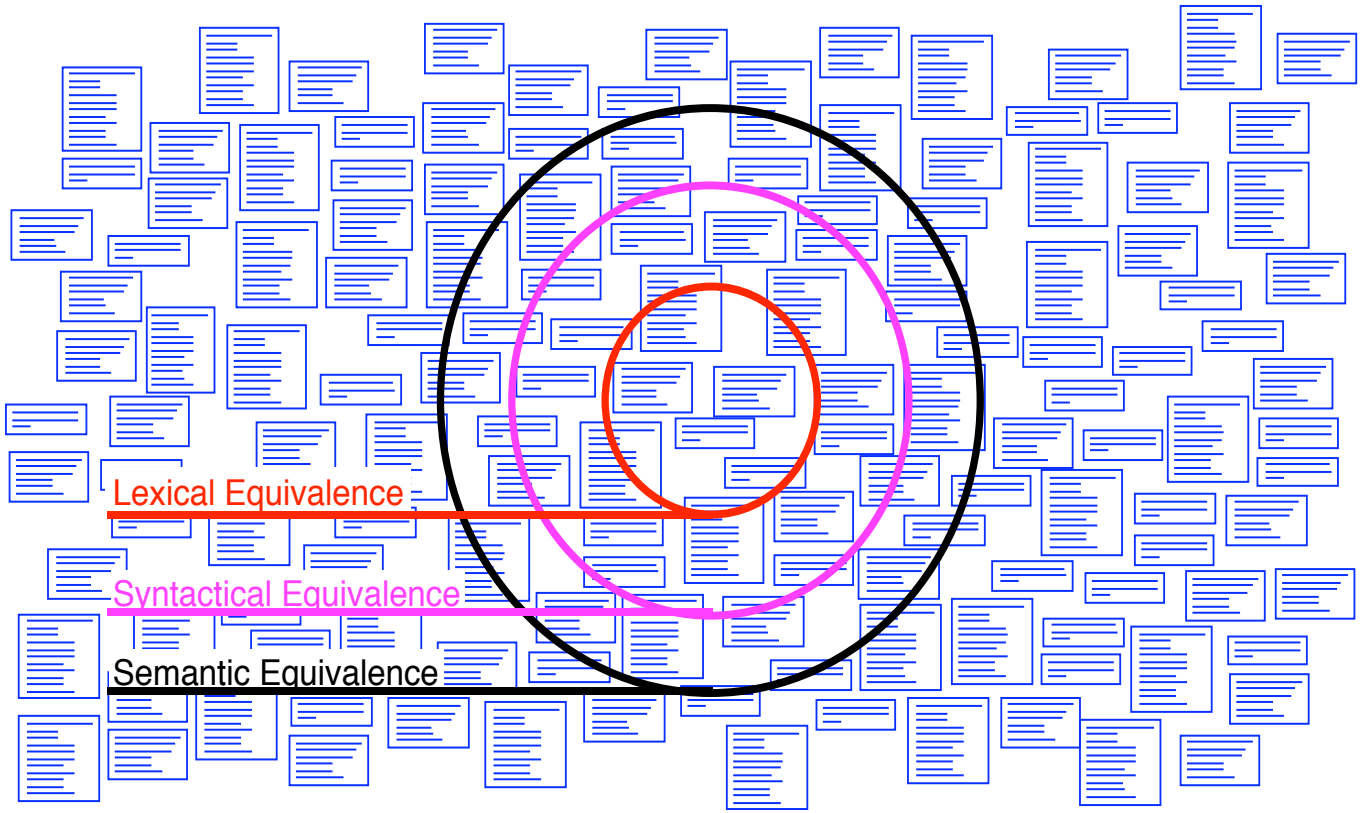
Copied Code Problems

- General negative effect:
  - Code bloat
- Negative effects on *Software Maintenance*
  - Copied Defects
  - Changes take double, triple, quadruple, ... Work
  - Dead code
  - Add to the cognitive load of future maintainers
- Copying as additional source of defects
  - Errors in the systematic renaming produce unintended aliasing
- Metaphorically speaking:
  - Software Aging, “hardening of the arteries”,
  - “Software Entropy” increases even small design changes become very difficult to effect

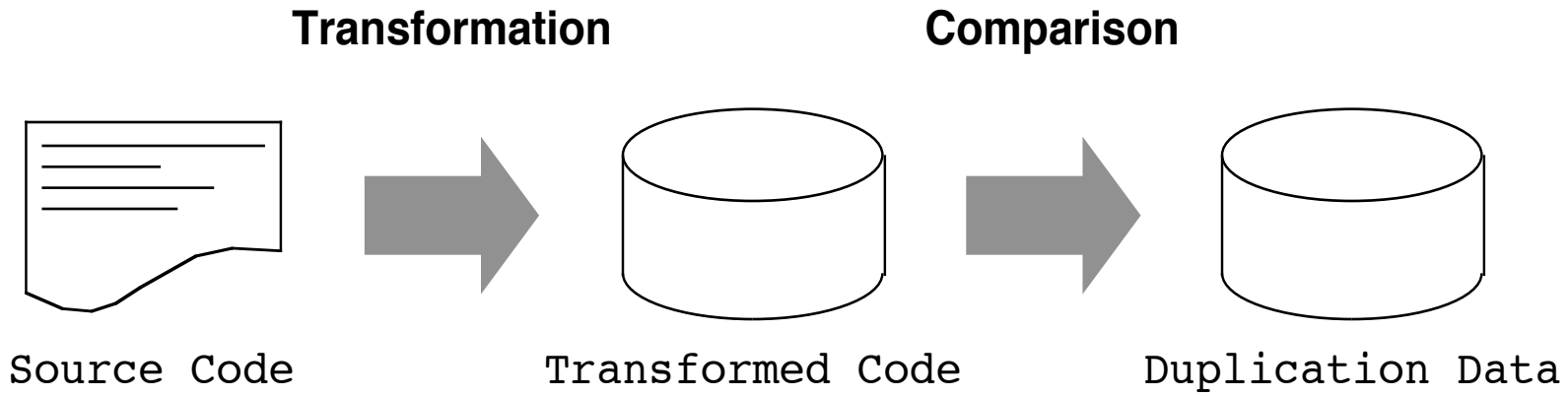
Code Duplication Detection

Nontrivial problem:

- No a priori knowledge about which code has been copied
- How to find all clone pairs among all possible pairs of segments?



General Schema of Detection Process

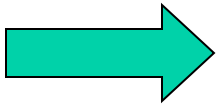


Author	*Level*	*Transformed Code*	*Comparison Technique*
[John94a]	Lexical	Substrings	String-Matching
[Duca99a]	Lexical	Normalized Strings	String-Matching
[Bake95a]	Syntactical	Parameterized Strings	String-Matching
[Mayr96a]	Syntactical	Metric Tuples	Discrete comparison
[Kont97a]	Syntactical	Metric Tuples	Euclidean distance
[Baxt98a]	Syntactical	AST	Tree-Matching

A Lightweight Approach (1)

- Assumption
  - Code segments are just copied and changed at a few places
- Code Transformation Step
  - remove white space, comments
  - remove lines that contain uninteresting code elements (e.g., just 'else' or '}')

```
...
//assign same fastid as container
fastid = NULL;
const char* fidptr = get_fastid();
if(fidptr != NULL) {
 int l = strlen(fidptr);
 fastid = newchar[ l + 1 ];
```



```
...
fastid=NULL;
constchar*fidptr=get_fastid();
if(fidptr!=NULL)
intl=strlen(fidptr)
fastid = newchar[l+1]
```

A Lightweight Approach (2)

- Code Comparison Step
  - Line based comparison (Assumption: Layout did not change during copying)
  - Compare each line with each other line.
  - Reduce search space by hashing:
    - Preprocessing: Compute the hash value for each line
    - Actual Comparison: Compare all lines in the same hash bucket
- Evaluation of the Approach
  - Advantages: Simple, language independent
  - Disadvantages: Difficult interpretation

Enhanced Simple Detection Approach

- Code Comparison Step

  Same as before +

  - Collect consecutive matching lines into match sequences
  - Allow holes in the match sequence

- Evaluation of the Approach

  Advantages

  - Identifies more real duplication, language independent

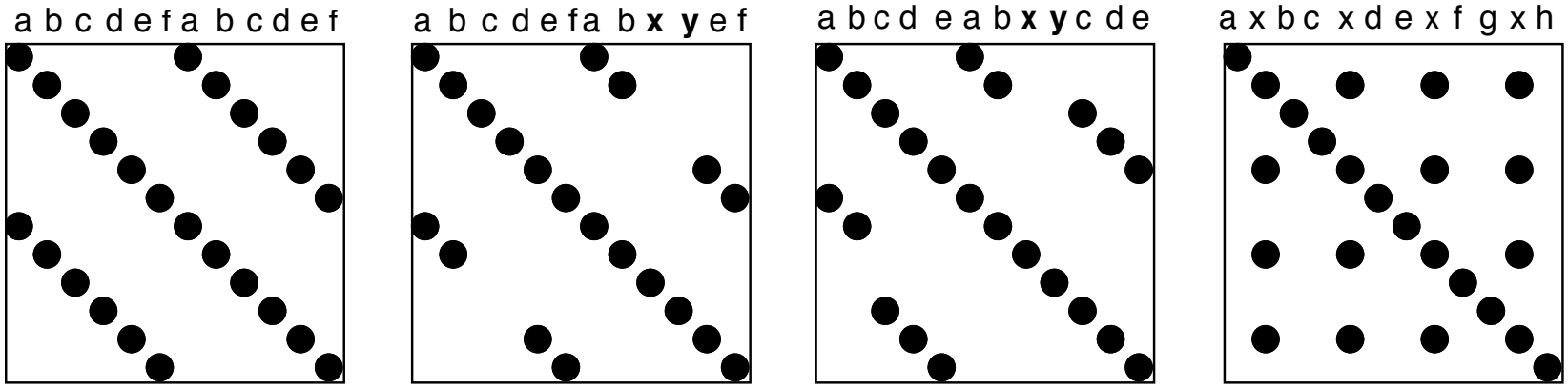
  Disadvantages

  - Less simple
  - Misses copies with (small) changes on every line

Visualization of Duplicated Code

- Visualization provides insights into the duplication situation
- A simple version can be implemented in three days
- Scalability issue

- Dotplots — Technique from DNA Analysis
  - Code is put on vertical as well as horizontal axis
  - A match between two elements is a dot in the matrix



Visualization of Copied Code Sequences

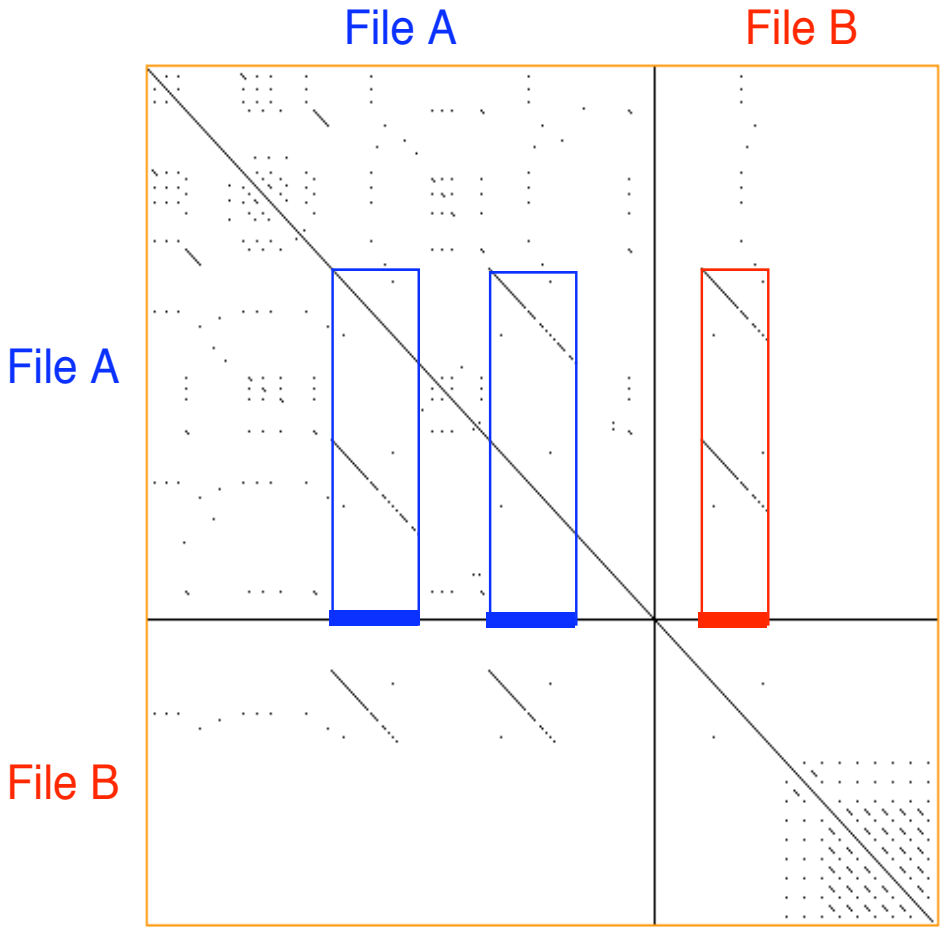
Detected Problem

File A contains two copies of a piece of code

File B contains another copy of this code

Possible Solution

Extract Method



All examples are made using Duploc from an industrial case study (1 Mio LOC C++ System)

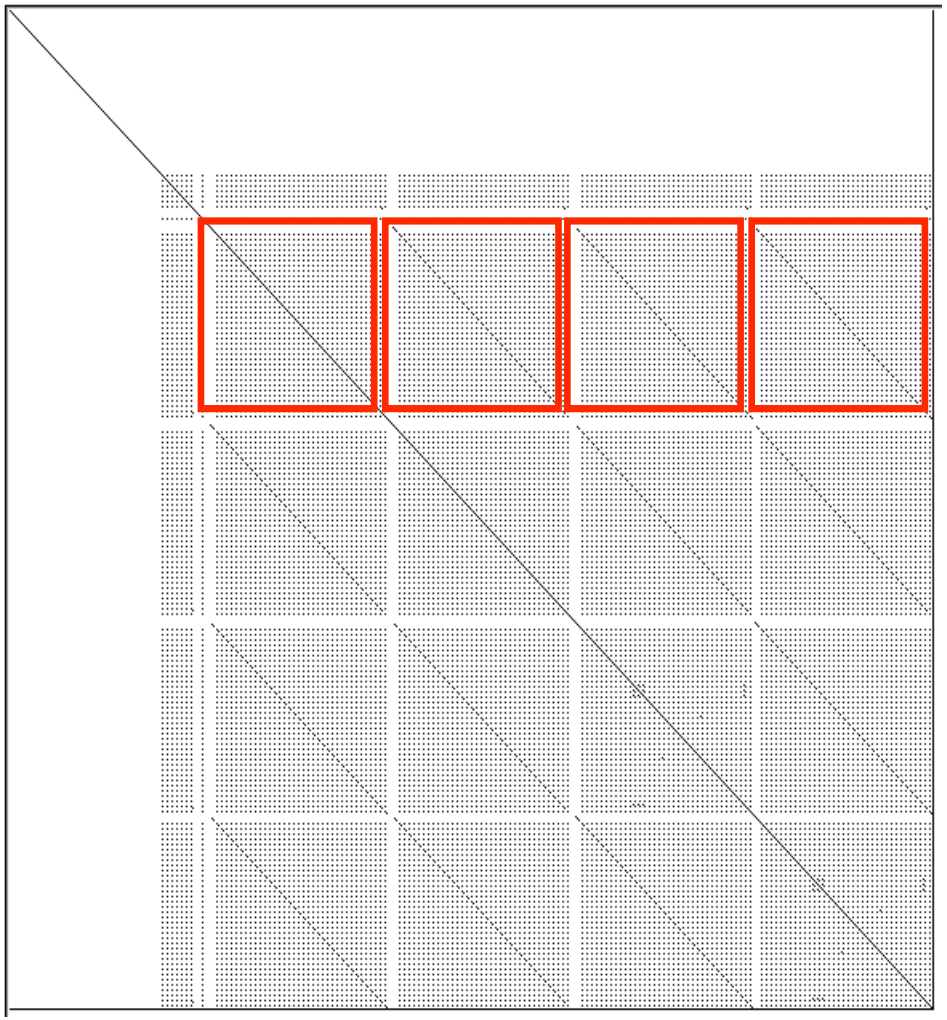
Visualization of Repetitive Structures

Detected Problem

4 Object factory clones: a switch statement over a type variable is used to call individual construction code

Possible Solution

Strategy Method



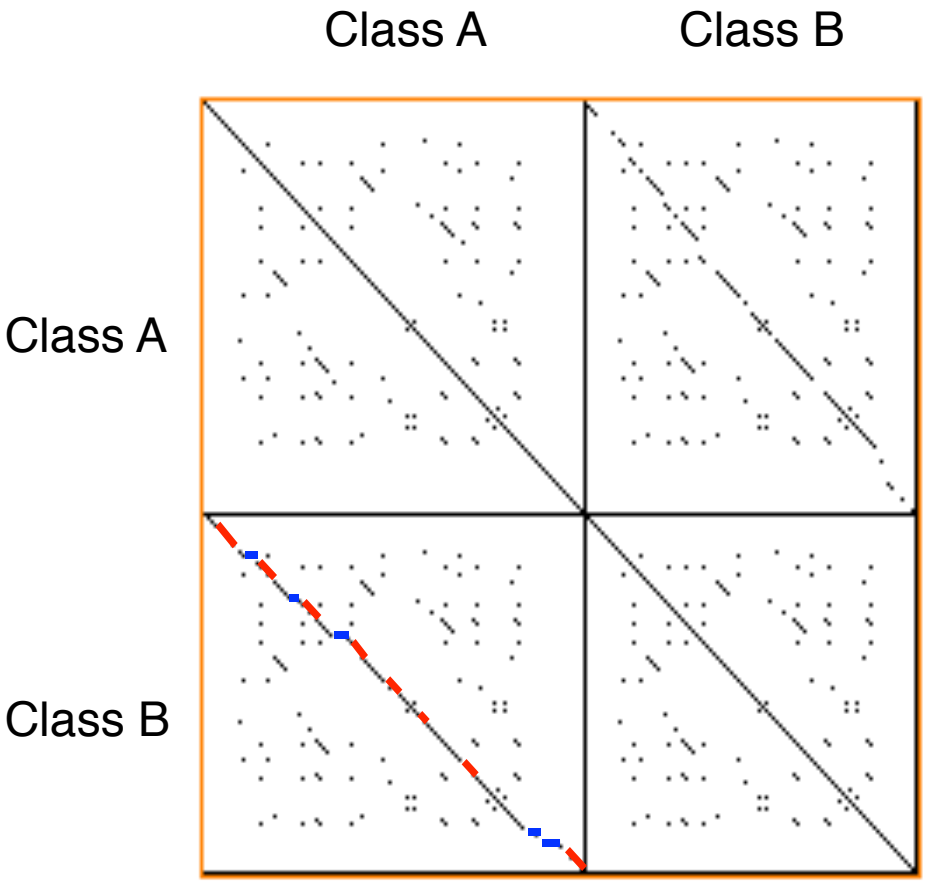
Visualization of Cloned Classes

Detected Problem

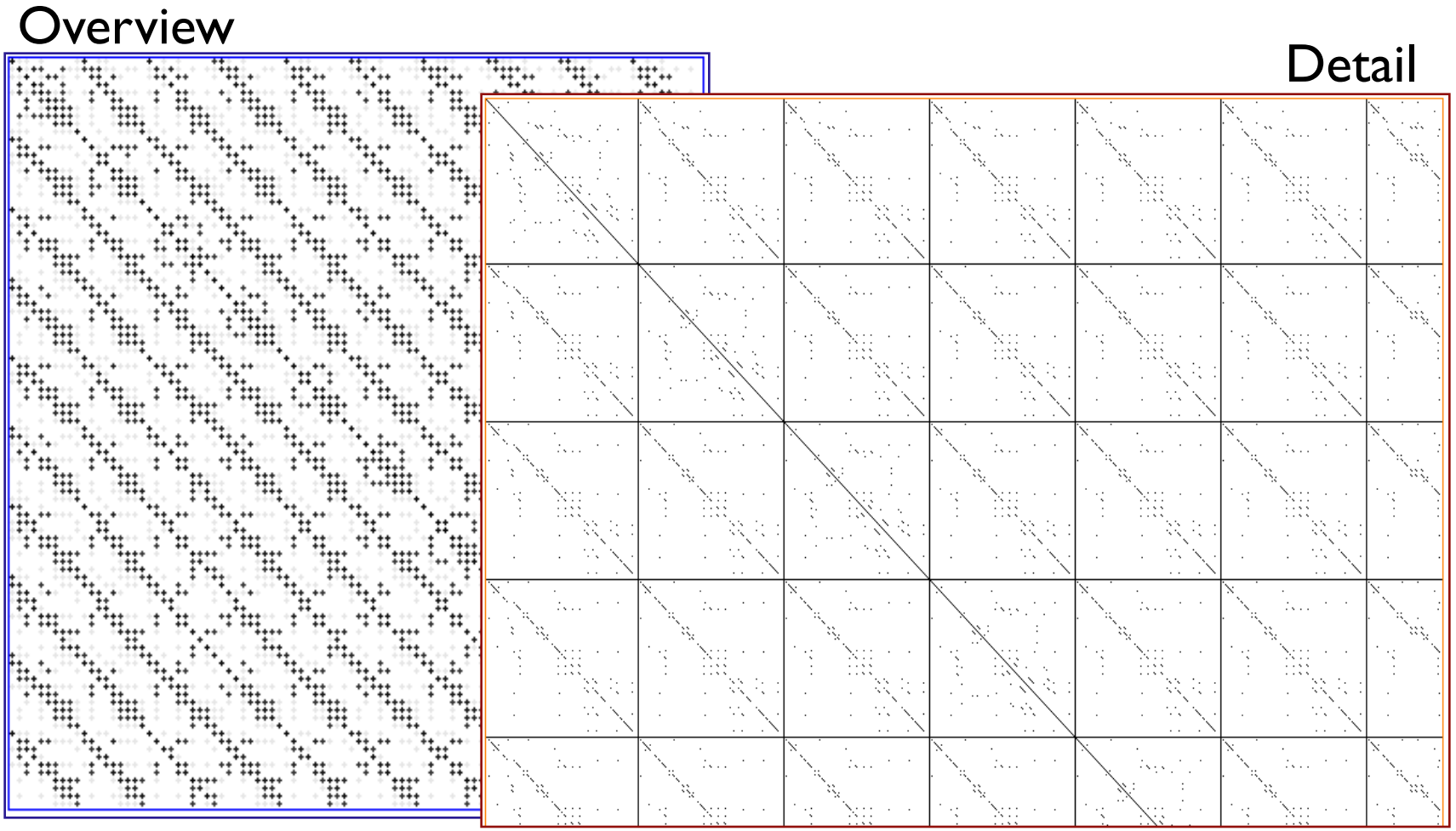
Class A is an edited copy of class B. Editing & Insertion

Possible Solution

Subclassing …



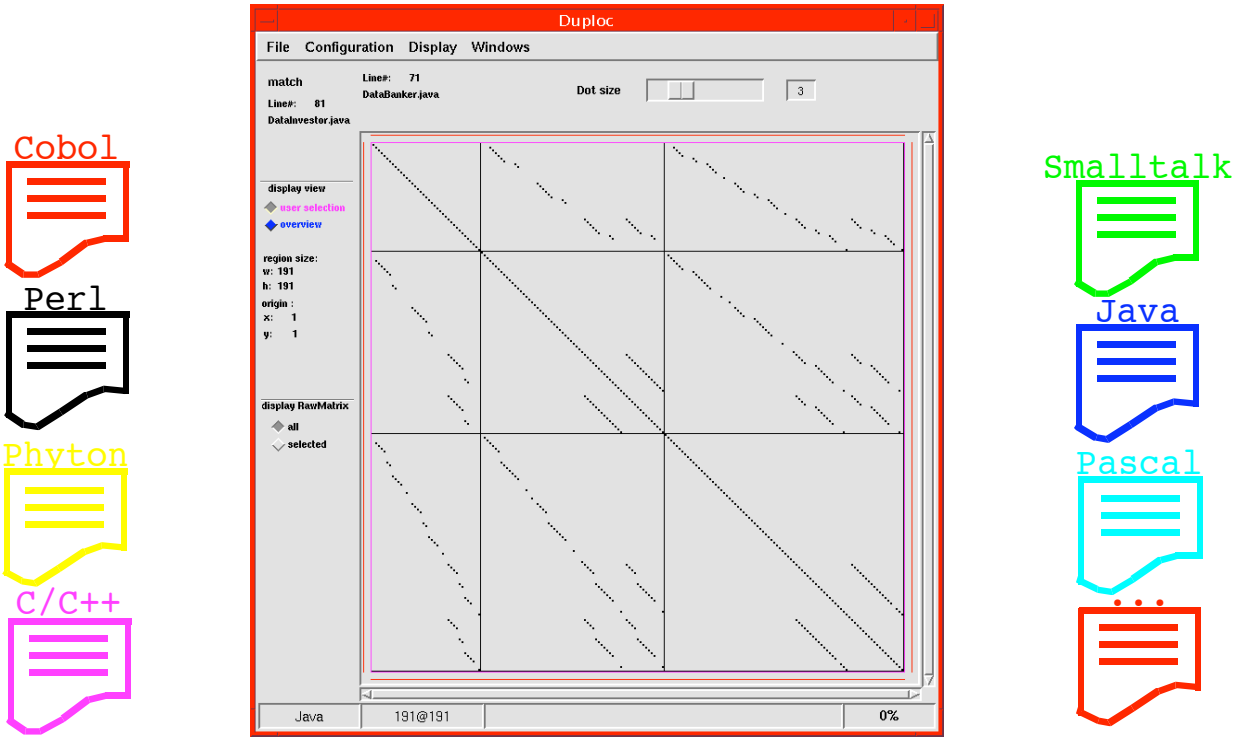
Visualization of Clone Families



20 Classes implementing lists for different data types

Lightweight is sometimes not enough

Duploc is scalable, integrates detection and visualization



It runs really everywhere (Smalltalk inside)

More Clone Detection

Tool	Author	Supported Languages	Domain	Approach Category	Background
CCFinder	T.Kamiya	C, C++, COBOL, Java, Emacs Lisp, Plain Text	Clone Detection	Transformation followed by token matching	Academic
CloneDr	I. Baxter	C, C++, COBOL, Java, Progress	Clone Detection	Abstract Syntax Tree comparison	Commercial
Covet	J. Bailey J. Mayrand	Java	Clone Detection	Comparison of Function Metrics	Academic
JPlag	G. Malpohl	C, C++, Java, Scheme	Plagiarism Detection	Transformation followed by token matching	Academic
Moss	A. Aiken	Ada, C, C++, Java, Lisp, ML, Pascal, Scheme	Plagiarism Detection	Unpublished	Academic

[Burd02]

Résumé

- Duplicated code is a real problem
  - makes a system progressively harder to change
- Detecting duplicated code is a hard problem
  - some simple technique can help
  - tool support is needed
- Visualization of code duplication is useful
  - some basic support are easy to build
  - one student build a simple visualization tool in three days
- Curing duplicated code is an active research area