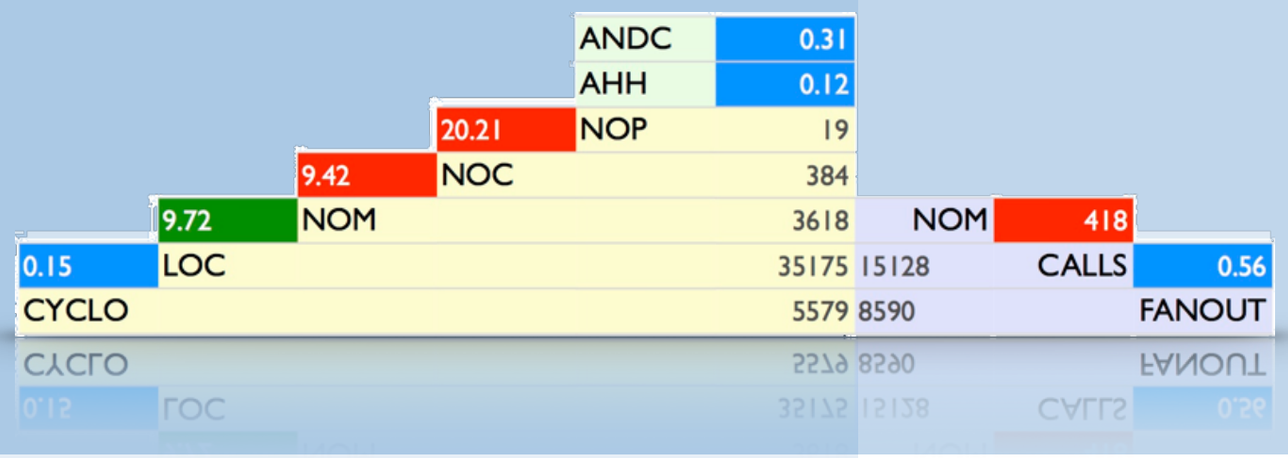
b
UNIVERSITÄT
BERN

Software Metrics and Problem Detection

Oscar Nierstrasz

Selected material by Mircea Lungu



Roadmap

- **Software Metrics**
  - Size / Complexity Metrics
  - Quality Metrics
- Metric-Based Problem Detection
  - Detecting Outliers
  - Encoding Design Problems
- Moose

Measurements

A measurement is a *mapping* from a *domain* to a *range* using mapping *rules*

A measure is a *numerical value* or a symbol assigned during mapping

In Software:
measures = *“metrics”*

**Measurement owes its existence to Earth;
Estimation of quantity to Measurement;
Calculation to Estimation of quantity;
Balancing of chances to Calculation;
and Victory to Balancing of chances.**

Measurements help you to reason about things using a simplified model rather than the things themselves. (Will these boxes fit in my car?)

https://en.wikipedia.org/wiki/Software_metric

The quotation is from Sun Tzu's "Art of War." Sun Tzu claims that measurement is the first step towards victory.

https://en.wikipedia.org/wiki/The_Art_of_War

The Measurement Process

The Goal-Question-Metric model proposes three steps to finding the correct metrics.

(Victor Basili)

1) Establish the **goals** of your maintenance or development project.

2) Derive, for each goal, **questions** that allow you to verify its accomplishment.

3) Find what should be **measured** in order to quantify the answer to the questions.

Targets without clear goals will not achieve their goals clearly.

Gilb's Principle

The Goal-Question-Metric method was introduced by Victor Basili. It proposes three steps for finding the correct metrics that should be collected during a program.

The GQM model seems too simple to deserve to be called a model. Still, its usefulness becomes clear when realizing that many metrics programs start not with a goal in mind, but with measuring what is easy to measure and end up with bunch of unrelated and non-conclusive measurements.

This situation was addressed also by Kybourg: “If you have no viable theory in which X enters, you have very little motivation to generate a measure of X”.

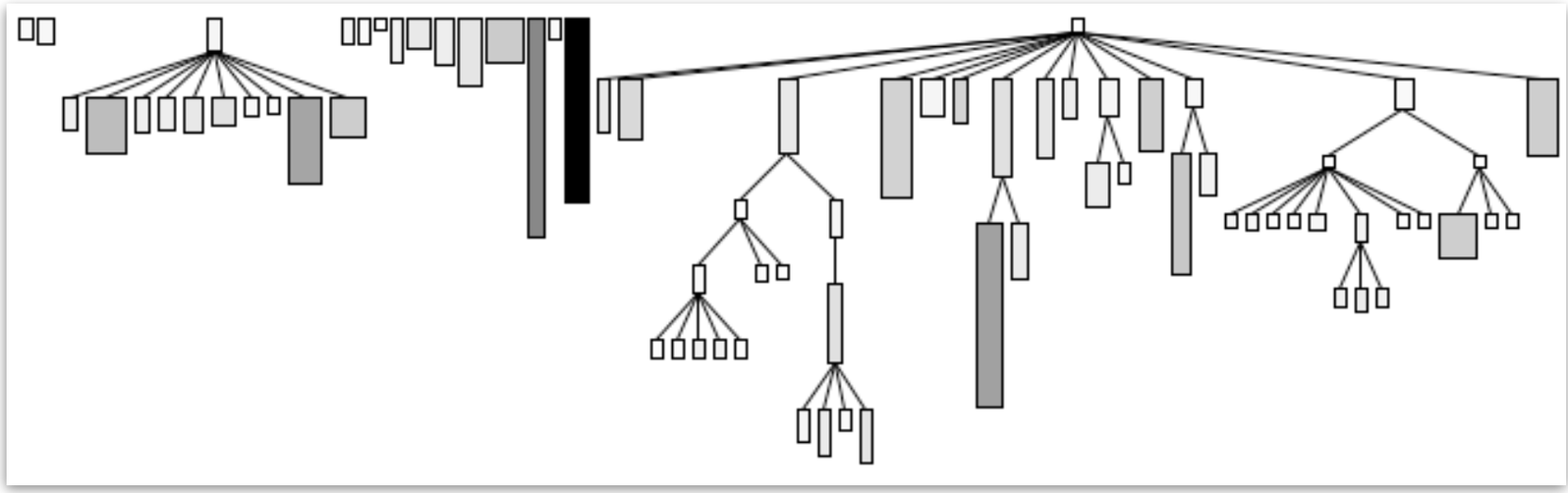
https://en.wikipedia.org/wiki/GQM

Roadmap

- Software Metrics
  - **Size / Complexity Metrics**
  - Quality Metrics
- Metric-Based Problem Detection
  - Detecting Outliers
  - Encoding Design Problems
- Moose

Size Measures

- LOC
- NOM
- NOA
- NOC
- NOP
- ... etc.



The graphic is a *polymetric view* that maps metrics to simple visualizations. This is a “System Complexity View” showing an inheritance hierarchy with NOM (# of methods) mapped to the height of each class node, NOA (# attributes) to the width, and LOC to the colour (black indicates the most LOC).

Many object-oriented metrics were first defined in the early 90s. See, for example: Chidamber, Kemerer, *A Metrics Suite for Object Oriented Design*. IEEE TSE, 1994.

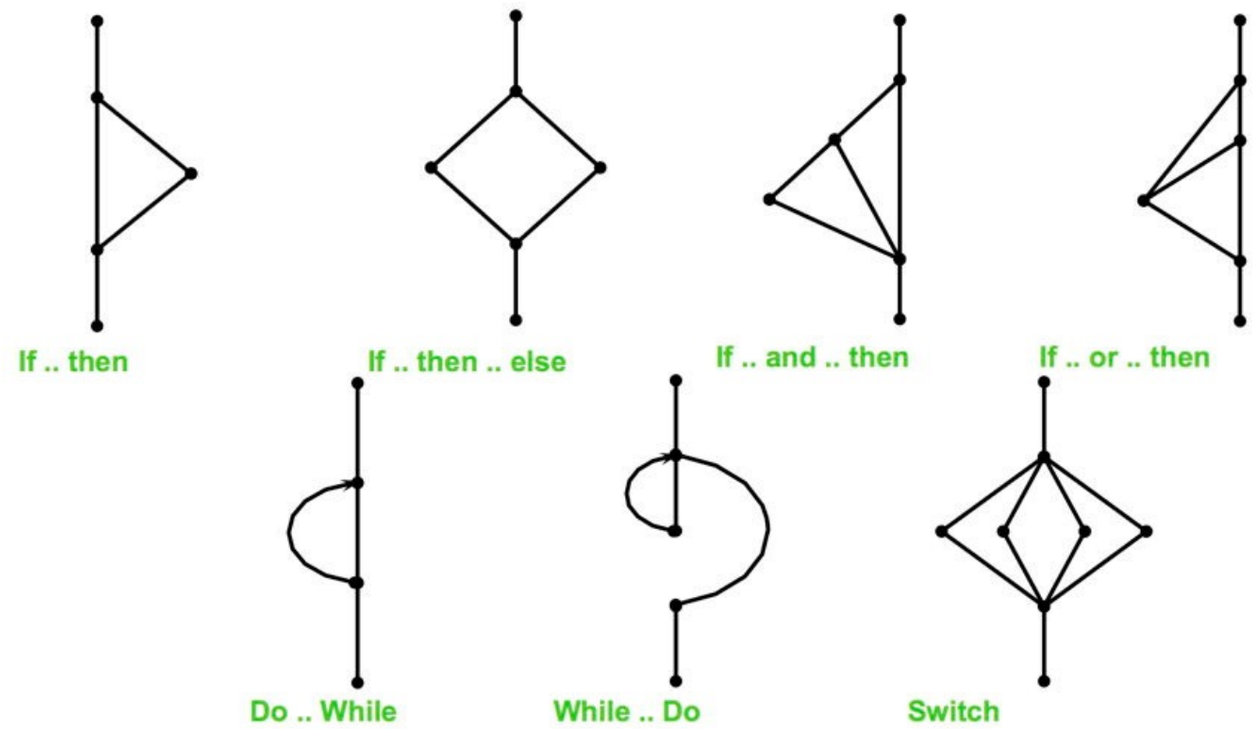
http://dx.doi.org/10.1109/32.295895

http://scgresources.unibe.ch/Literature/SMA/Chid94a-OOMetrics.pdf

Cyclomatic Complexity (CYCLO)

The number of independent linear paths through a program.
(McCabe ’77)

+ Measures minimum effort for testing



Cyclomatic Complexity is one of the best-known measures of complexity, which equates complexity with control flow: the more paths through a piece of code, the more complex it is considered to be. This is especially useful for measuring test coverage. (Tests should cover all paths through a program.)

CC can be computed by representing control flow as a directed graph, and computing:

CC = E - N + 2

where E = # edges and N = # nodes

https://en.wikipedia.org/wiki/Cyclomatic_complexity

Weighted Methods per Class (WMC)

The complexity of a class by summing the complexity of its methods, usually using CYCLO.
(Chidamber & Kemerer ’94)

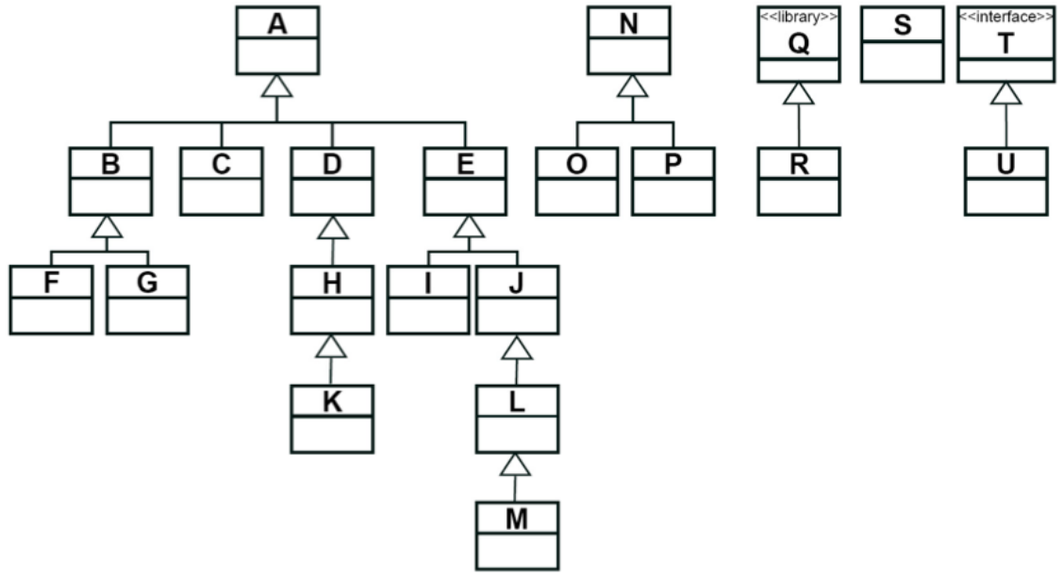
+ A proxy for the time and effort required to maintain a class

Intuition: the larger the complexity of a class, the more difficult its maintenance.

Depth of Inheritance Tree (DIT)

The maximum depth level of a class in a hierarchy. (Chidamber & Kemerer ’94)

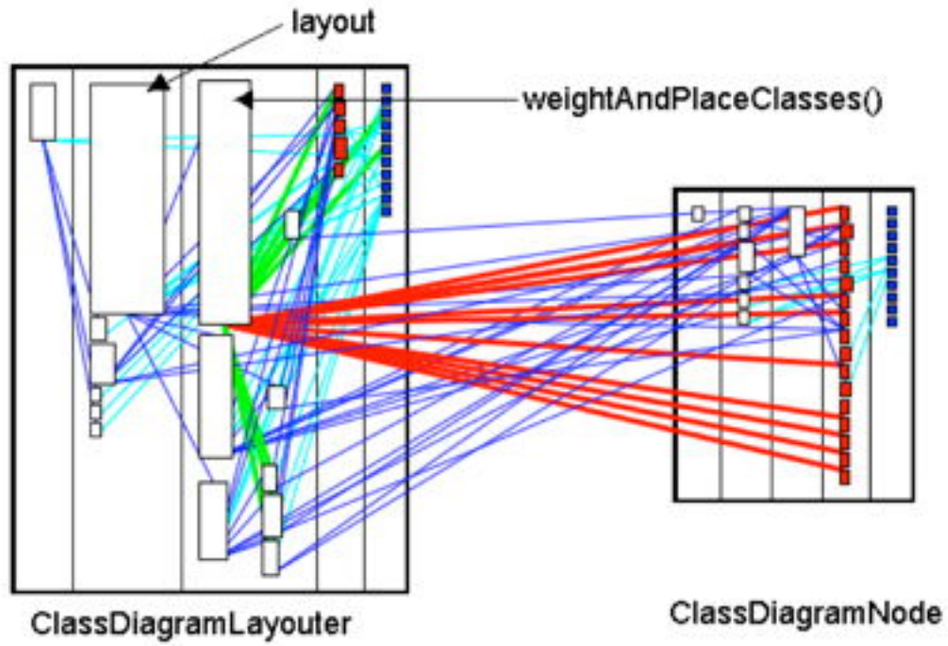
+ Inheritance depth is a good proxy for complexity

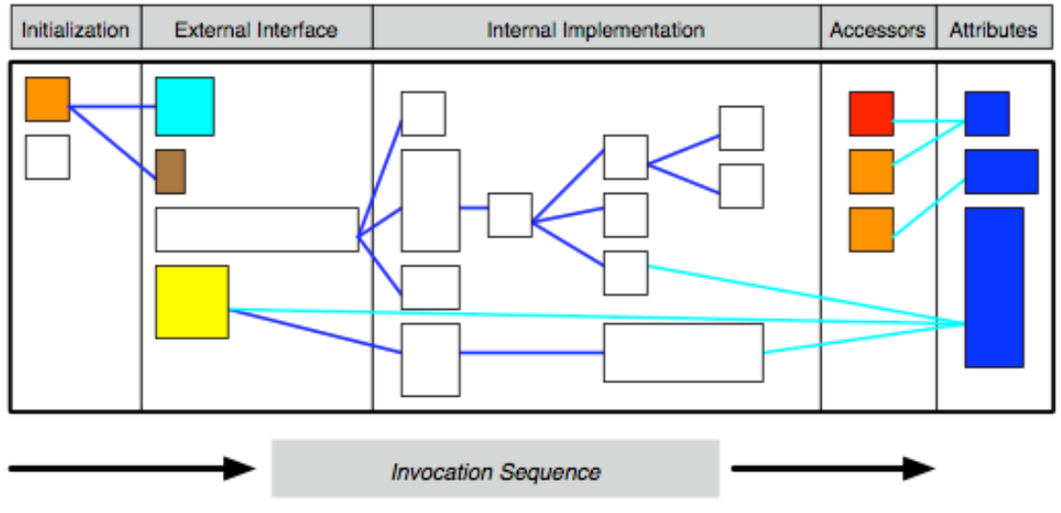


Access To Foreign Data (ATFD)

ATFD counts how many attributes from other classes are accessed directly from a given class.
(Lanza & Marinescu ’06)

+ ATFD summarizes the interaction of a class with its environment





The visualizations are *class blueprints*. They show five categories of methods and attributes of a single class and their calling/ accessing relations. In the first column at the left are *constructors*, followed by *public methods*, *internal methods* (protected or private), *accessors*, and finally *attributes*.

Here we see that the `weightAndPlaceClasses` and `layout` methods of `ClassDiagramLayouter` are very large, and use many accessors and attributes of `ClassDiagramNode`. The latter has little behavior of its own.

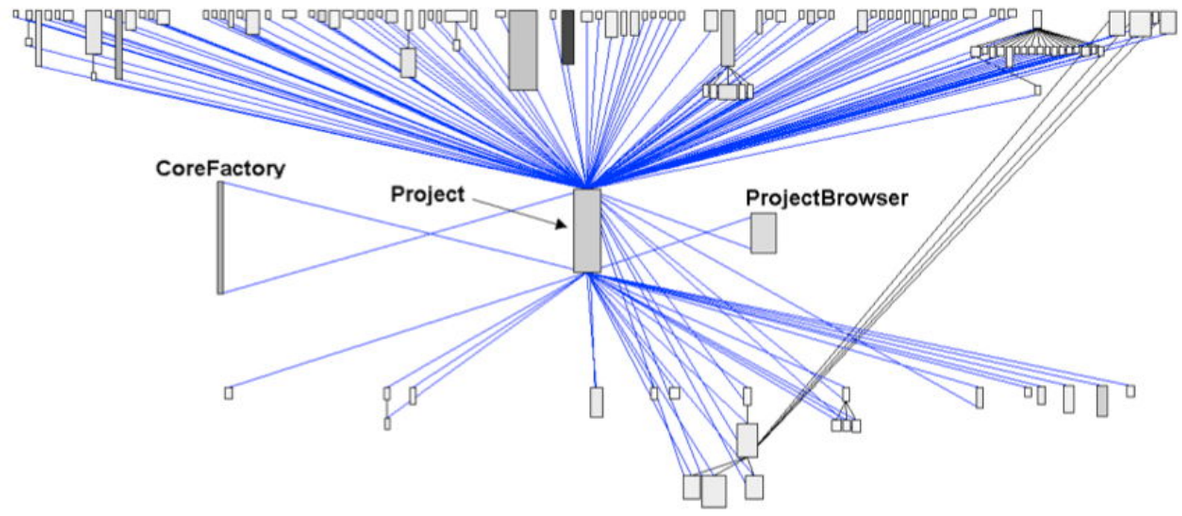
Roadmap

- Software Metrics
  - Size / Complexity Metrics
  - **Quality Metrics**
- Metric-Based Problem Detection
  - Detecting Outliers
  - Encoding Design Problems
- Moose

Coupling Between Object Classes (CBO)

CBO for a class is the number of other classes to which it is coupled. (Chidamber & Kemerer ’94)

+ Meant to assess modular design and reuse



The concept of *coupling* was introduced in a 1974 article by Stevens, Myers, and Constantine. *Structured Design*. IBM Systems Journal, 1974

http://scgresources.unibe.ch/Literature/SMA/Stev74a-StructuredDesign.pdf

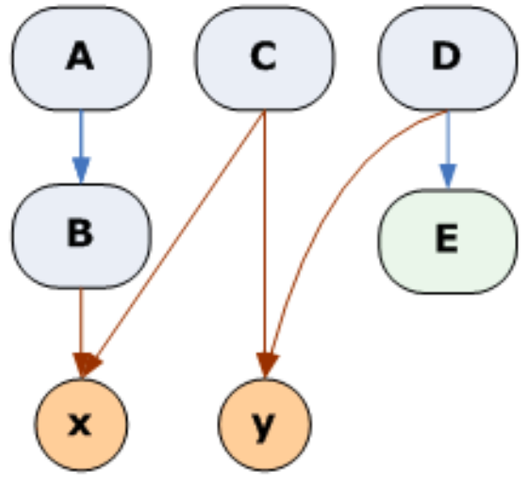
https://en.wikipedia.org/wiki/Coupling_(computer_programming)

Here we see that the class `Project` has both large *fan-in* (clients using it) and *fan-out* (classes it is a client of), so it is highly coupled to much of the system.

Tight Class Cohesion (TCC)

TCC counts the relative number of method-pairs that access attributes of the class in common. (Bieman & Kang, 95)

+ Can lead to improvement action



TCC = 4 / 10 = 0.4

A class is considered to be cohesive using TCC if its methods access many of the same attributes. TCC counts the pairs of methods that access a common attribute.

Roadmap

- Software Metrics
  - Size / Complexity Metrics
  - Quality Metrics
- **Metric-based Problem Detection**
  - Detecting Outliers
  - Encoding Design Problems
- Moose

Roadmap

- Software Metrics
  - Size / Complexity Metrics
  - Quality Metrics
- Metric-based Problem Detection
  - **Detecting Outliers**
  - Encoding Design Problems
- Moose

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

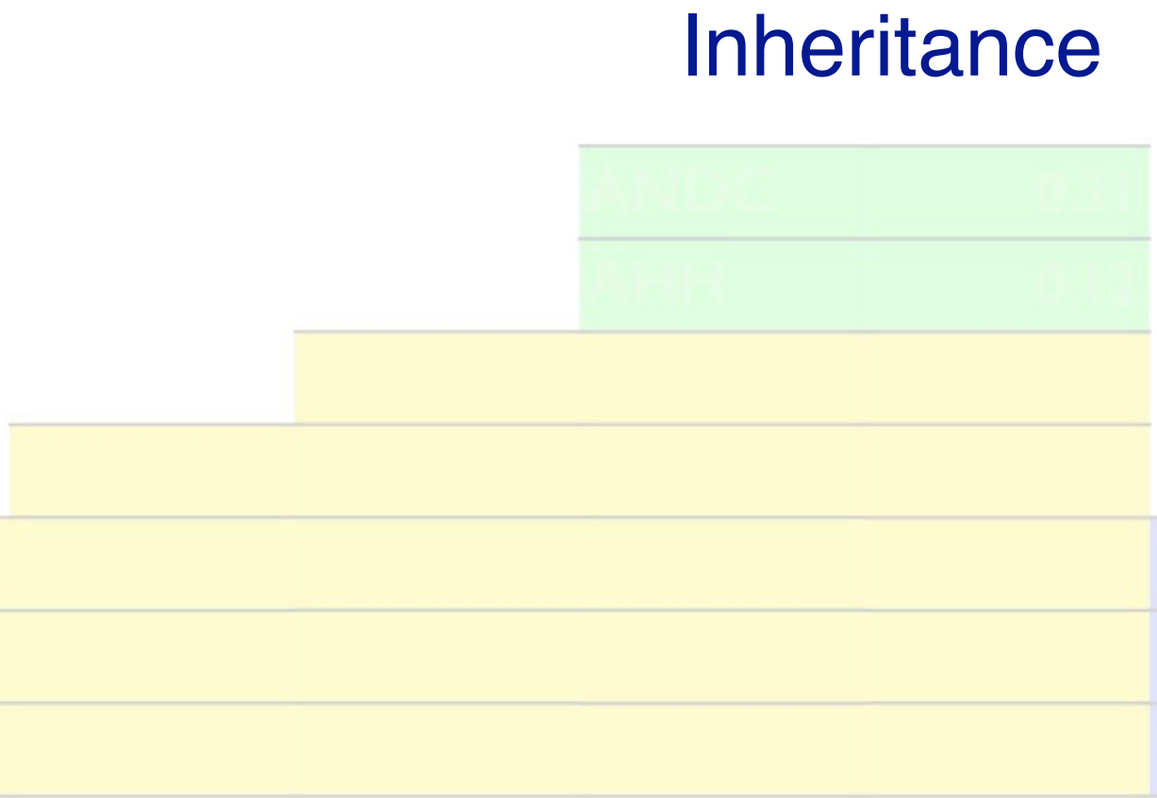
—Browse the code to get insight into the anomalies

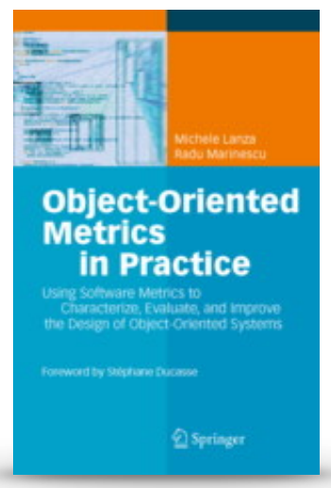
This pattern is from the open-source book, "*Object-Oriented Reengineering Patterns*". Like design patterns, reengineering patterns encode knowledge mined from experience with practical problems in real software systems. Instead of encoding design experience, however, these patterns express *how to reverse engineer and reengineer* legacy software systems.

This particular pattern is useful when you want to obtain an initial overview of a complex object-oriented software system. Since the code base may be very large, it is not feasible to read even a small portion of the code. Instead, by applying simple metrics and visualizing the results, your attention may be drawn to anomalous outliers (i.e., exceptional entities) that will tell you interesting things about the system.

http://scg.unibe.ch/download/oorp/

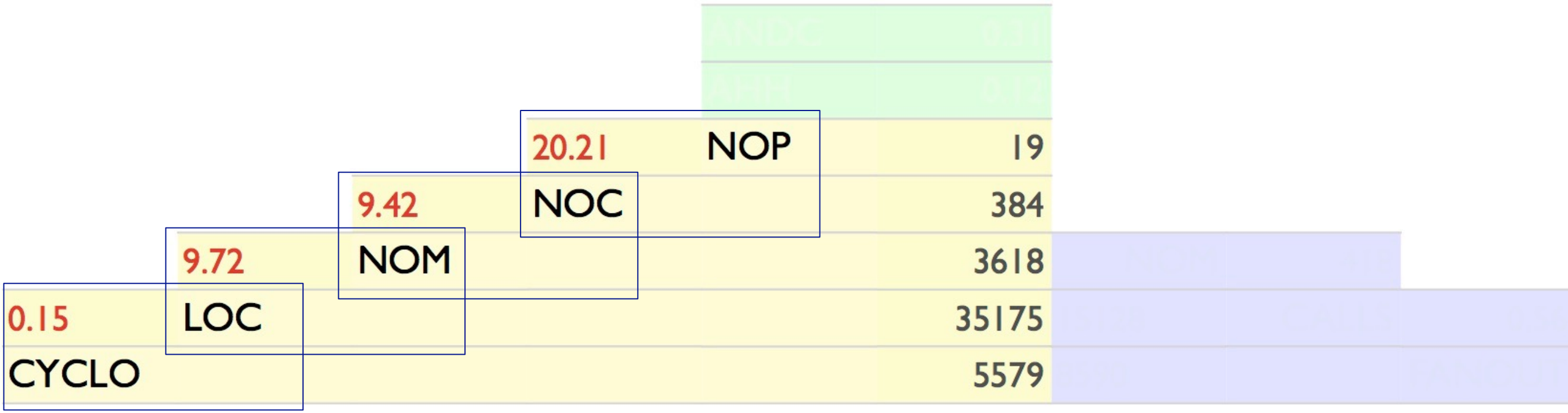
The Overview Pyramid provides a metrics overview.





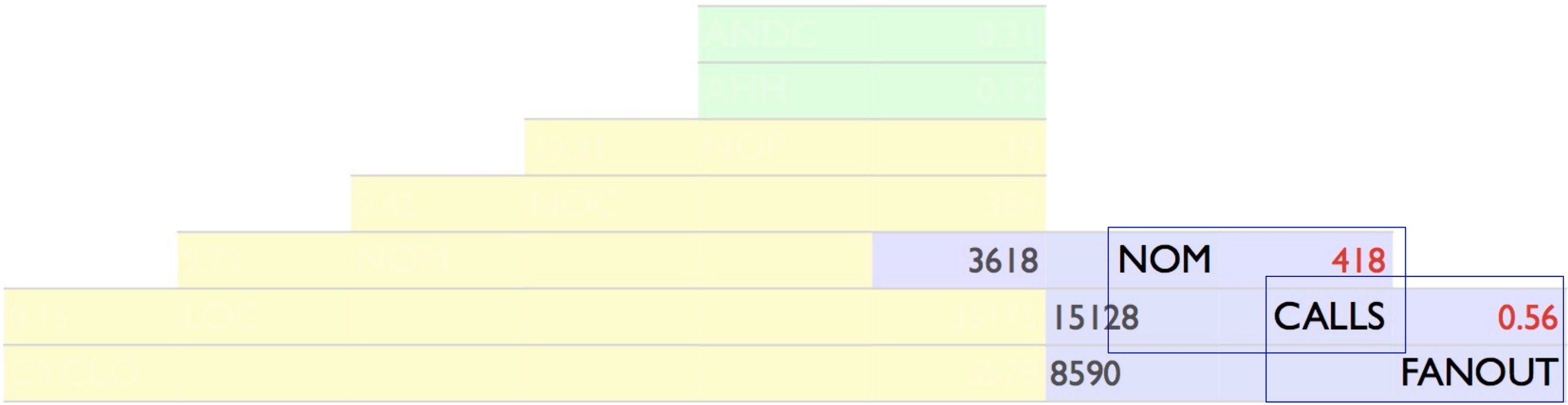
The Overview Pyramid summarizes several common metrics in a single diagram, indicating whether metric values are outliers are not.

The Overview Pyramid provides a metrics overview.



Size

The Overview Pyramid provides a metrics overview.



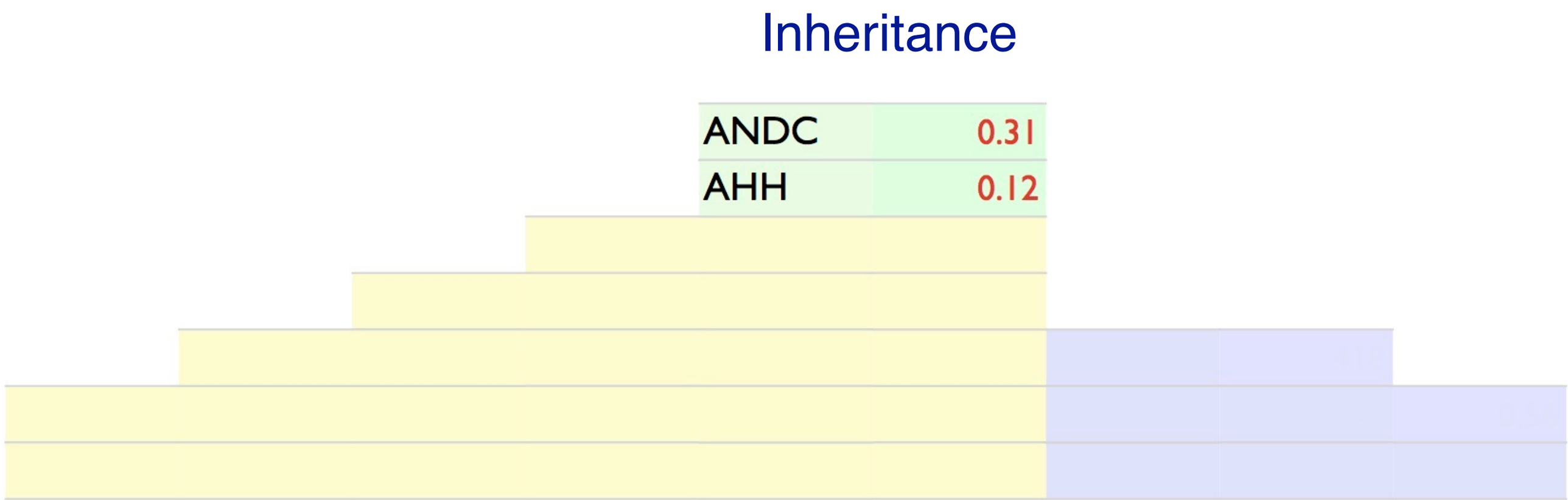
Communication

CALLS = Number of call sites

FANOUT = Sum of all FANOUTS

Coupling Intensity

The Overview Pyramid provides a metrics overview.



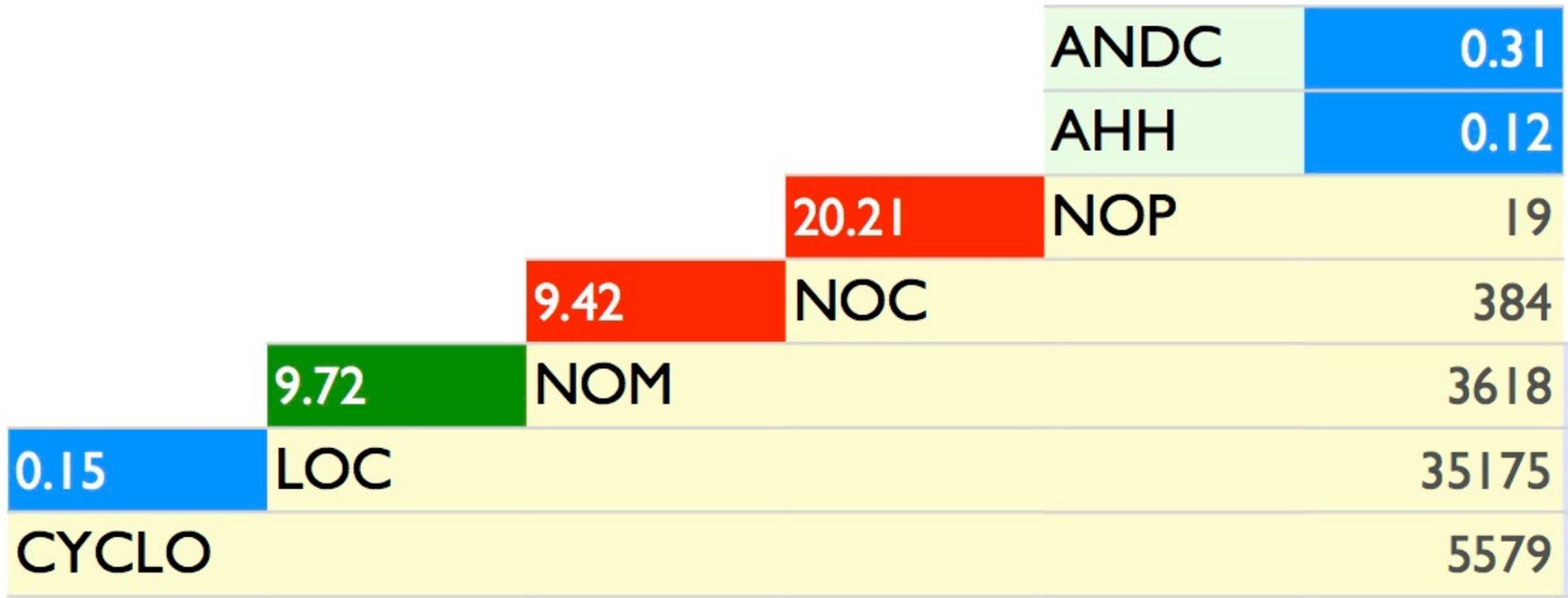
ANDC = Average Number of Derived Classes

AHH = Average Hierarchy Height

The Overview Pyramid provides a metrics overview.

			ANDC	0.31		
			AHH	0.12		
		20.21	NOP	19		
	9.42	NOC		384		
	9.72	NOM	3618	NOM	418	
0.15	LOC		35175	15128	CALLS	0.56
CYCLO			5579	8590		FANOUT

The Overview Pyramid provides a metrics overview.



Outlier Detection

close to high

close to average

close to low

How to obtain the thresholds?

	Java			C++		
	LOW	AVG	HIGH	LOW	AVG	HIGH
CYCLO/LOC	0.16	0.20	0.24	0.20	0.25	0.30
LOC/NOM	7	10	13	5	10	16
NOM/NOC	4	7	10	4	9	15
...						

Statistical static analysis of reference systems

Context is important (e.g. programming language)

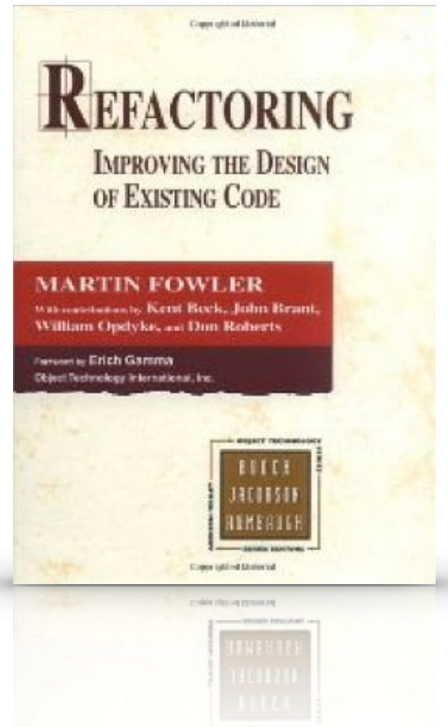
Lanza and Marinescu advise that you study many software systems and extract statistical rules. The book extracts statistics based on 45 Java systems.

Outliers may be signs of poor quality …

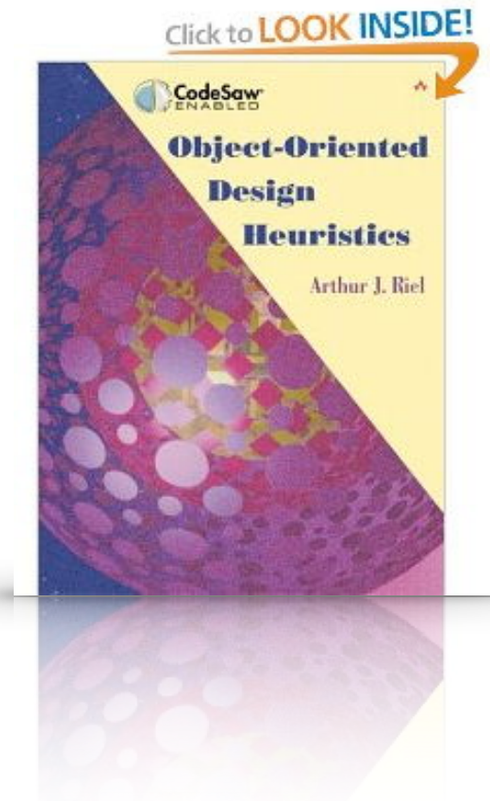
Roadmap

- Software Metrics
  - Size / Complexity Metrics
  - Quality Metrics
- Metric-based Problem Detection
  - Detecting Outliers
  - **Encoding Design Problems**
- Moose

Design Problems and Principles



Bad Smells
Comments
Switch Statement
Shotgun Surgery
...



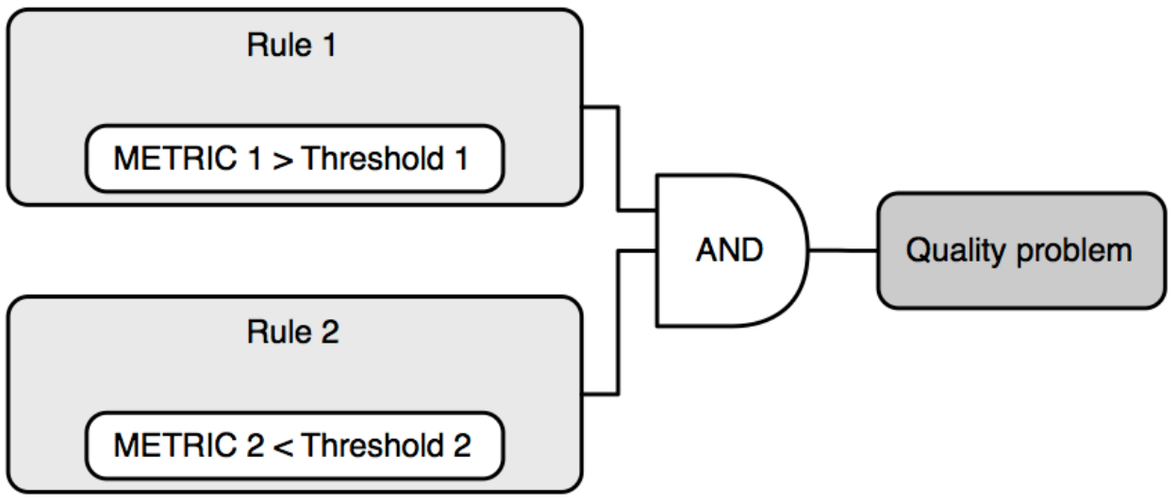
Design Heuristics
Encapsulation
Minimize Coupling
Class Coherence
Inheritance Depth
...

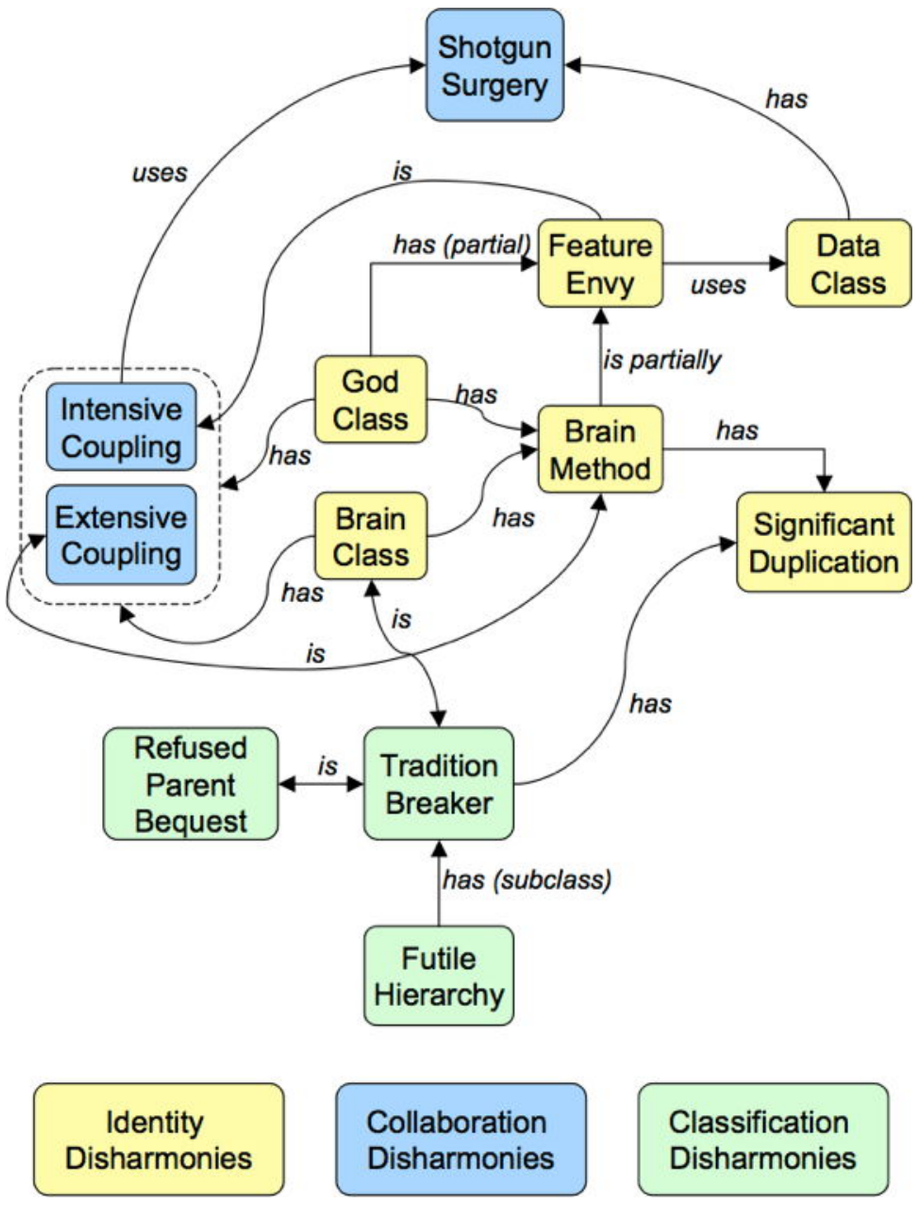
Design principles come in prose - how to measure them?

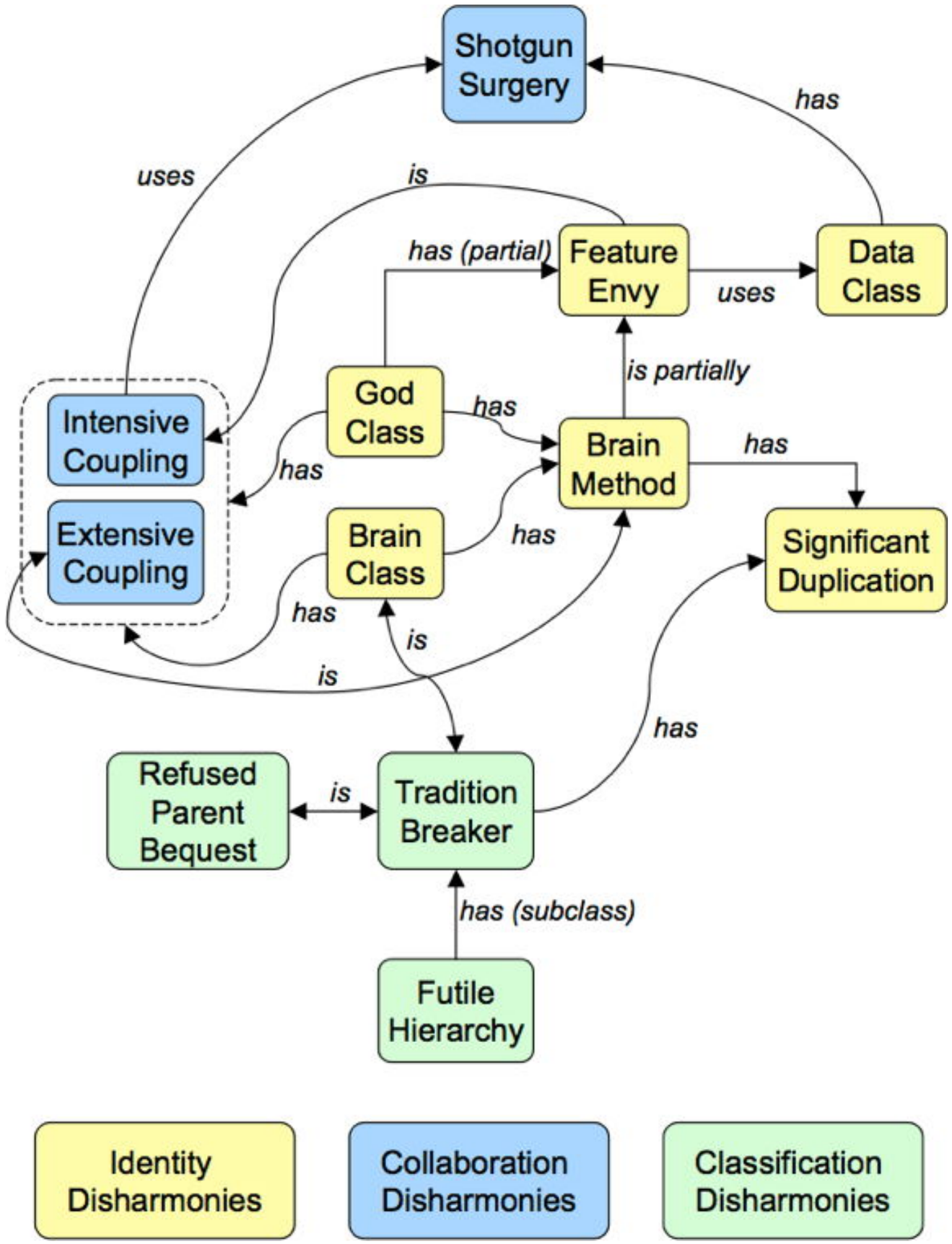
Rarely a single metric is sufficient >>> Detection Strategies

Detection Strategies...

- ... are metric based queries for detecting design problems
- (Lanza & Marinescu 2002)






Shotgun Surgery
has
uses
is
has (partial)
Feature Envy
uses
Data Class
is partially
God Class
has
Brain Method
has
Intensive Coupling
has
Extensive Coupling
Brain Class
has
Significant Duplication
has
is
is
has
Refused Parent Bequest
is
Tradition Breaker
has (subclass)
Futile Hierarchy
Identity Disharmonies
Collaboration Disharmonies
Classification Disharmonies

God Classes ...

... tend to **centralize the intelligence** of the system, to **do everything**, and to **use data** from small data-classes

God Classes ...

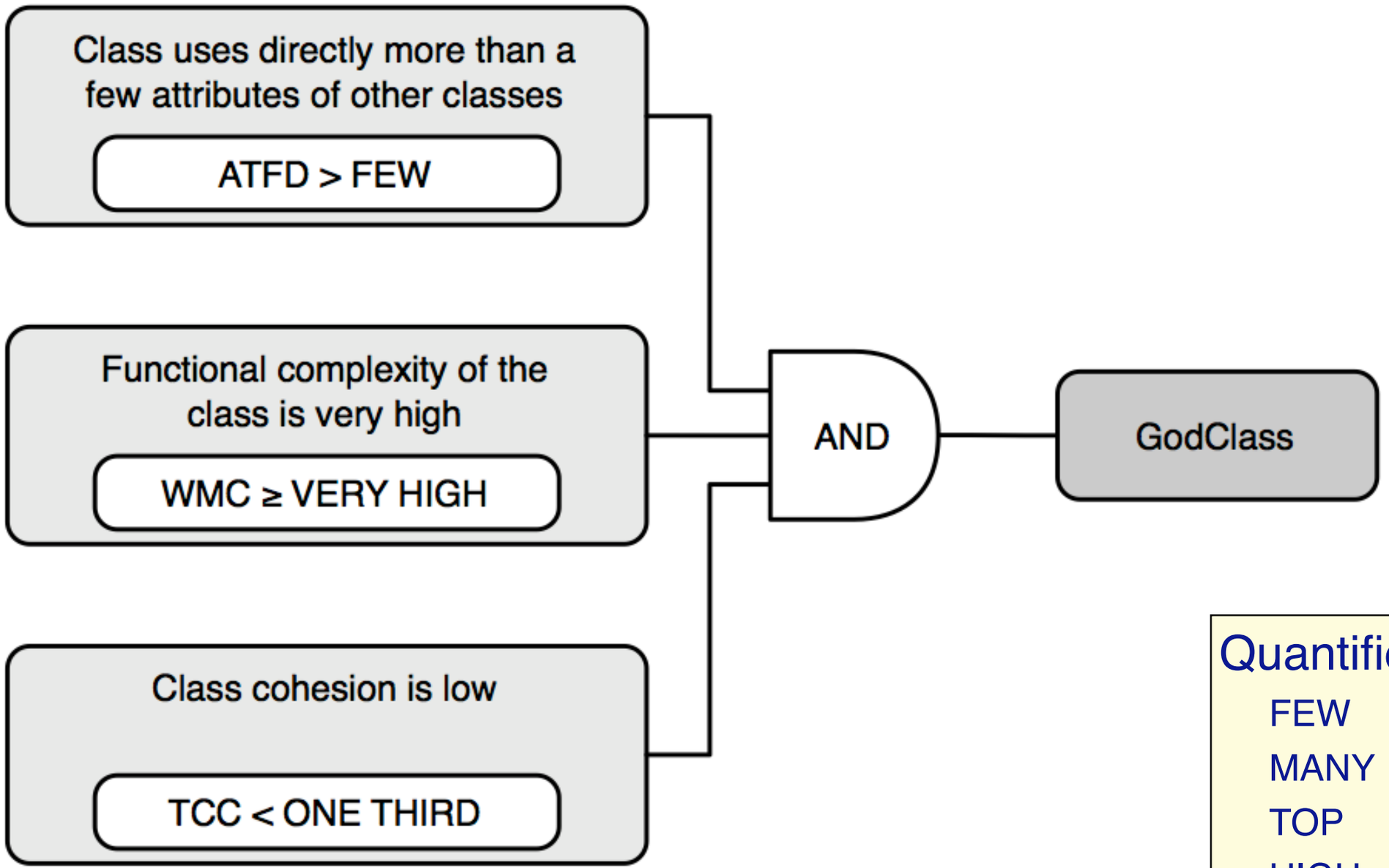
Complexity (WMC)

... tend to **centralize the intelligence** of the system, to **do everything**, and to **use data** from small data-classes

Lack of cohesion (TCC)

Foreign data usage (ATFD)

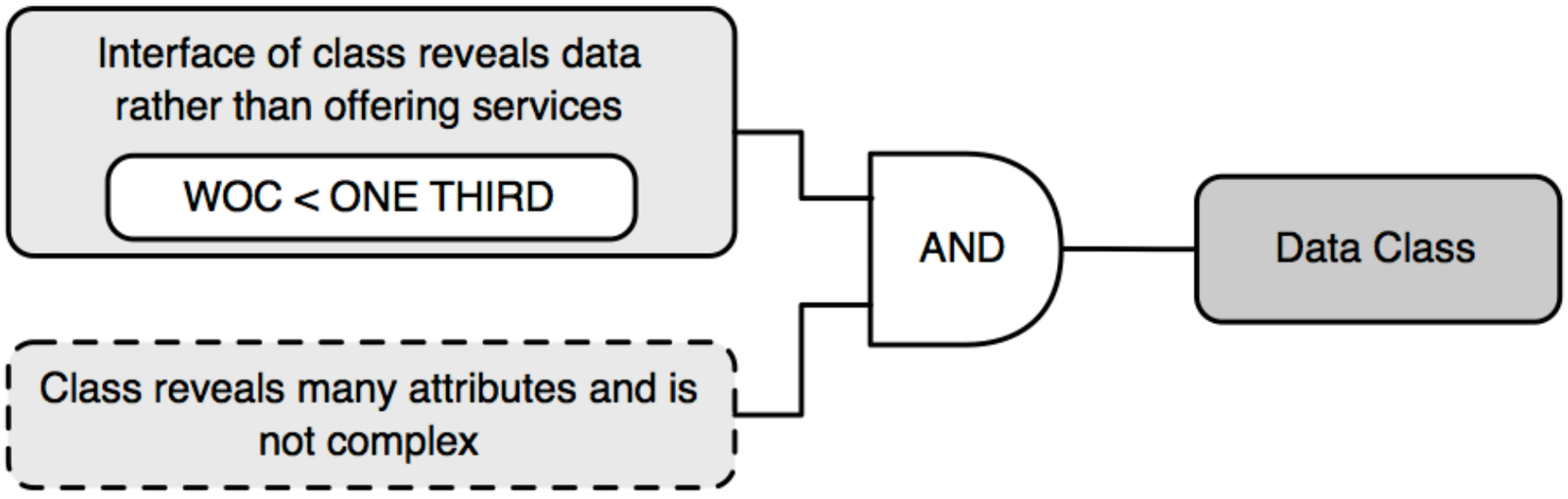
God Classes



Quantifiers

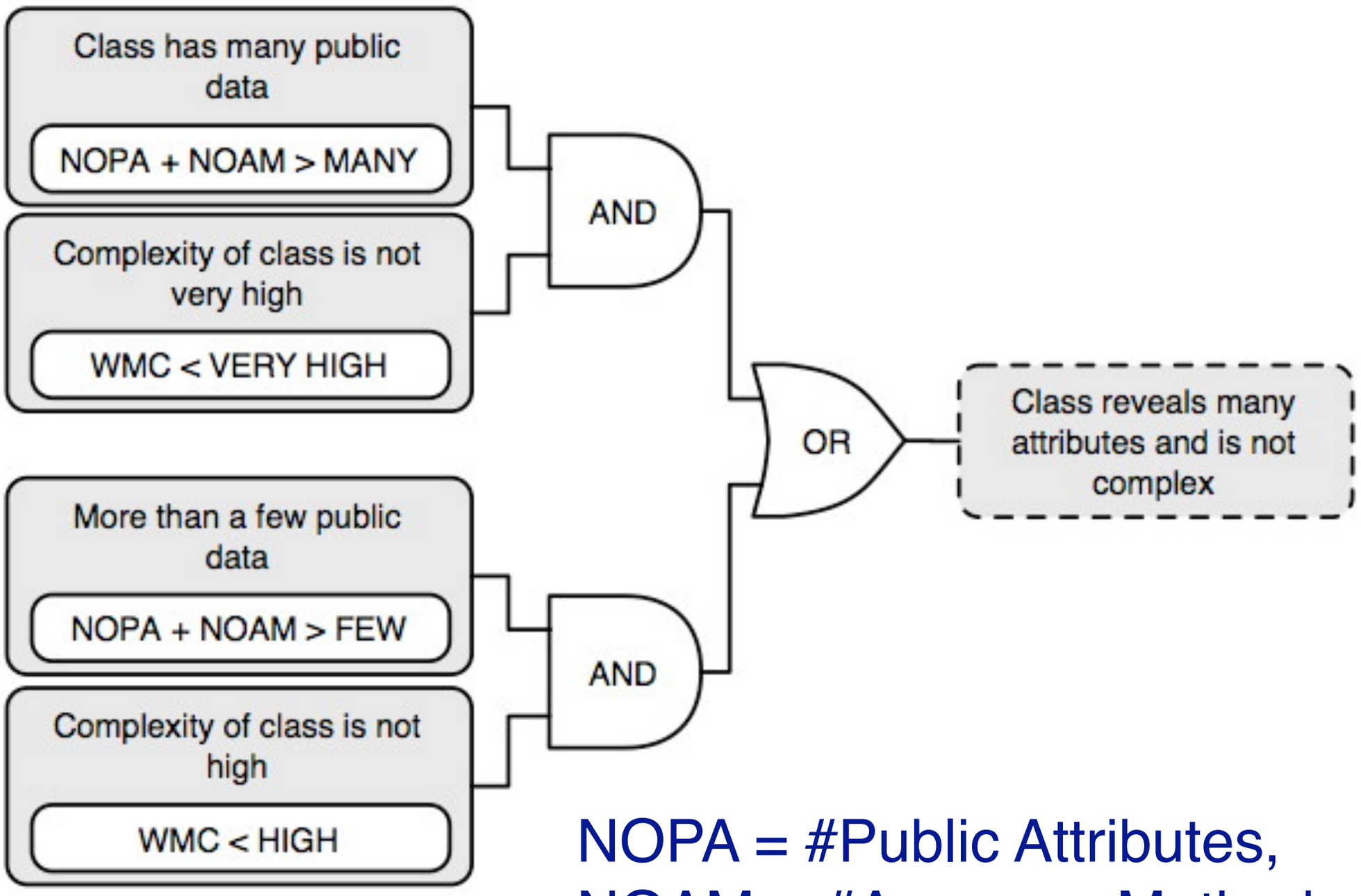
- FEW
- MANY
- TOP
- HIGH
- ONE THIRD ...

Data Classes are dumb data holders



WOC - Weight Of a Class	
Definition	The number of "functional" public methods divided by the total number of public members (Mar02a)

Data Classes are dumb data holders



NOPA = #Public Attributes,
NOAM = #Accessor Methods

Roadmap

- Software Metrics
  - Size / Complexity Metrics
  - Quality Metrics
- Metric-based Problem Detection
  - Detecting Outliers
  - Encoding Design Problems
- **Moose** *(separate presentation)*

What you should know!

- What is a metric?
- Why goals are important for choosing metrics?
- Why is the context of the metrics important?
- What is the difference between coupling and cohesion?
- What metrics are present on the overview pyramid?
- What do detection strategies detect?
- Why reading code is bad?
- How does Moose solve the data analysis problems?

Can you answer these questions?

- What metrics are important for measuring some of your projects (bachelor's thesis project, etc…)?
- What are acceptable ranges of CYCLO for different languages?
- Why is it hard to calculate CYCLO for Smalltalk methods?
- What insights can a class blueprint provide besides ATFD?
- What are the strategies of solving code smells detected by detection strategies?