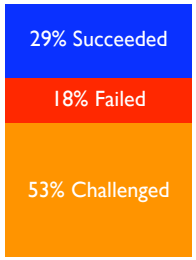
Metrics and Problem Detection

Jorge Ressia

Software is complex.



The Standish Group, 2004

After 50 years, software is not “soft” anymore. It is heavy and dificult to manage.

http://www.standishgroup.com/sample_research/PDFpages/q3-spotlightpdf

How large is your project?

1’000’000 lines of code

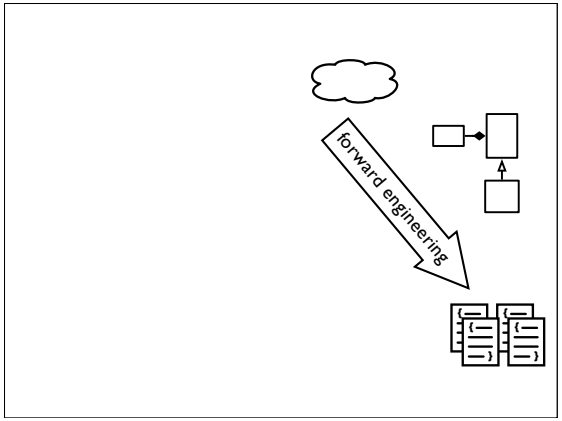
* 2 = 2’000’000 seconds

/ 3600 = 560 hours

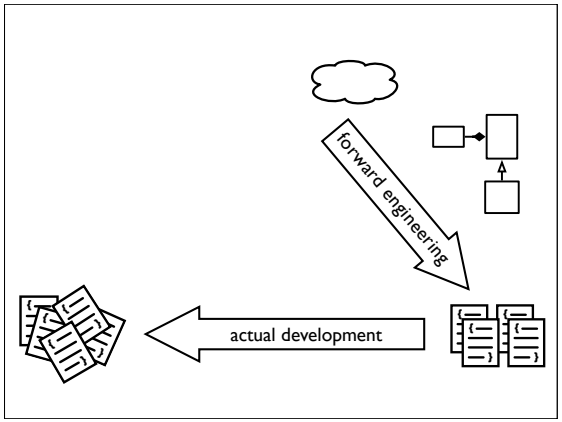
/ 8 = 70 days

/ 20 = 3 months

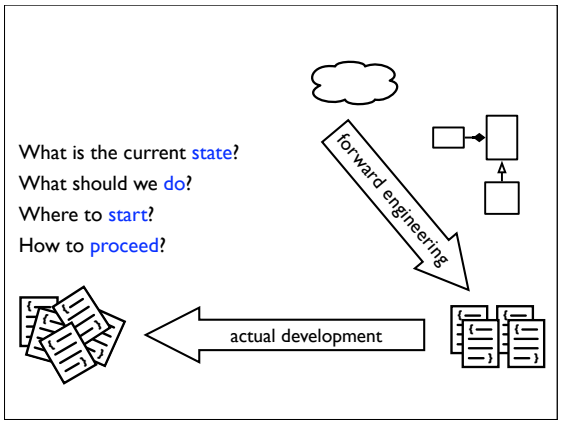
For example, if you get a piece of software of 1’000’000 lines of code it would take you 3 months to read it if your reading speed is 2 seconds per line of code.



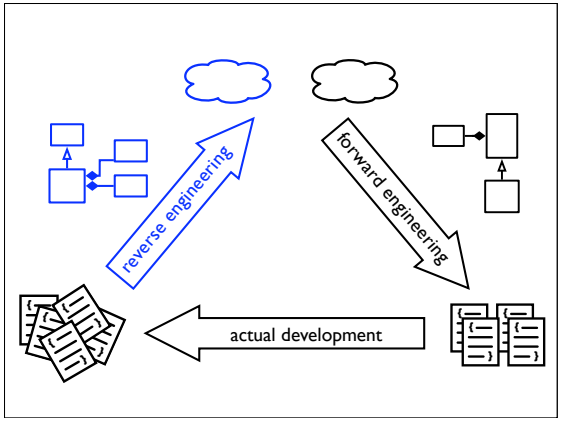
In most projects, the actual development happens only at the code level, with only little documentation maintenance.



In most projects, the actual development happens only at the code level, with only little documentation maintenance.



In most projects, the actual development happens only at the code level, with only little documentation maintenance.



In most projects, the actual development happens only at the code level, with only little documentation maintenance.

Reverse engineering is analyzing a subject system to:

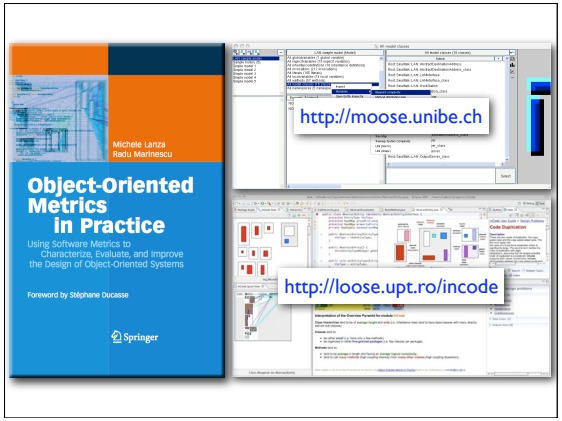
identify components and their relationships, and

create more abstract representations.

Chikofky & Cross, 90

Elliot Chikofsky and James Cross II, “Reverse Engineering and Design Recovery: A Taxonomy,” IEEE Software, vol. 7, no. 1, January 1990, pp. 13-17.
http://dx.doi.org/10.1109/52.43044

A large system contains lots of details.



This is the background of the talk.

1 Metrics

2 Design Problems

3 Code Duplication

Metrics

1

You cannot control

what you cannot measure.

Tom de Marco

Metrics are functions that assign numbers to products, processes and resources.

Software metrics are measurements which relate to software systems, processes or related documents.

When you can measure what you are speaking about and express it in numbers, you know something about it;
but when you cannot measure, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science.

Metrics compress system traits into numbers.

Let's see some examples...

Examples of size metrics

NOM - number of methods
NOA - number of attributes
LOC - number of lines of code
NOS - number of statements
NOC - number of children

Lorenz, Kidd, 1994
Chidamber, Kemerer, 1994

McCabe cyclomatic complexity (CYCLO) counts the number of independent paths through the code of a function.

McCabe, 1977

✓ it reveals the minimum number of tests to write

✗ interpretation can't directly lead to improvement action

Weighted Method Count (WMC) sums up the complexity of class' methods (measured by the metric of your choice; usually CYCLO).

Chidamber, Kemerer, 1994

✓ it is configurable, thus adaptable to our precise needs

✗ interpretation can't directly lead to improvement action

Depth of Inheritance Tree (DIT) is the (maximum) depth level of a class in a class hierarchy.

Chidamber, Kemerer, 1994

✓ inheritance is measured

✗ only the potential and not the real impact is quantified

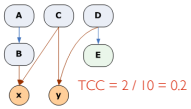
Coupling between objects (CBO) shows the number of classes from which methods or attributes are used.

Chidamber, Kemerer, 1994

✓ it takes into account real dependencies not just declared ones

✗ no differentiation of types and/or intensity of coupling

Tight Class Cohesion (TCC) counts the relative number of method-pairs that access attributes of the class in common.



TCC = 2 / 10 = 0.2

Bieman, Kang, 1995

✓ interpretation can lead to improvement action

✓ ratio values allow comparison between systems

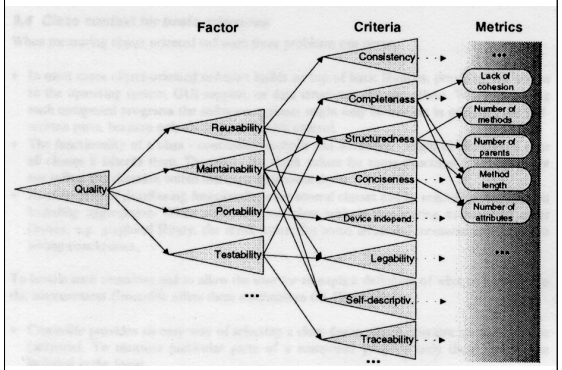
Access To Foreign Data (ATFD) counts how many attributes from other classes are accessed directly from a measured class.

Marinescu 2006

• • •

Design Problems

2

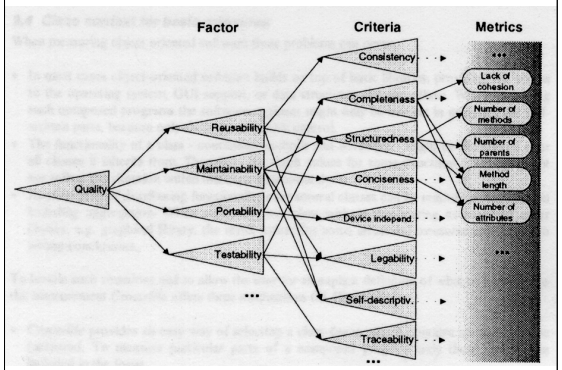


McCall, 1977

Metrics alone do not say anything about the quality of the system

Metrics Assess and Improve Quality!

Really?

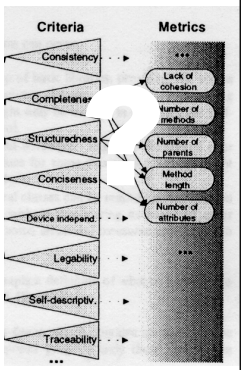


McCall, 1977

Problem 1: metrics granularity

capture symptoms, not causes of problems

in isolation,
they don't lead to improvement solutions



Problem 2: implicit mapping

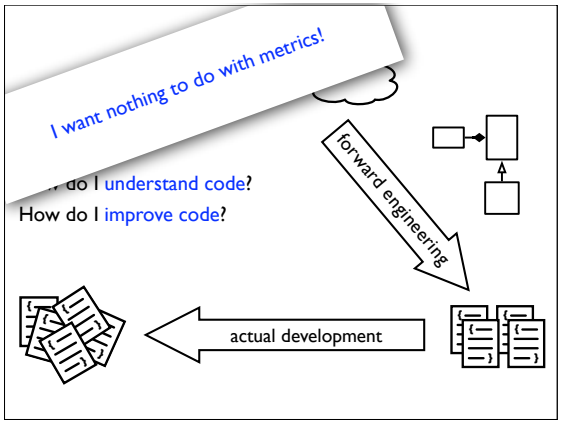
we don't reason in terms of metrics,
but in terms of design principles

2 big obstacles in using metrics:

Thresholds make metrics hard to interpret

Granularity make metrics hard to use in isolation

Can metrics help me
in what I really care for? :)



Understand the Code

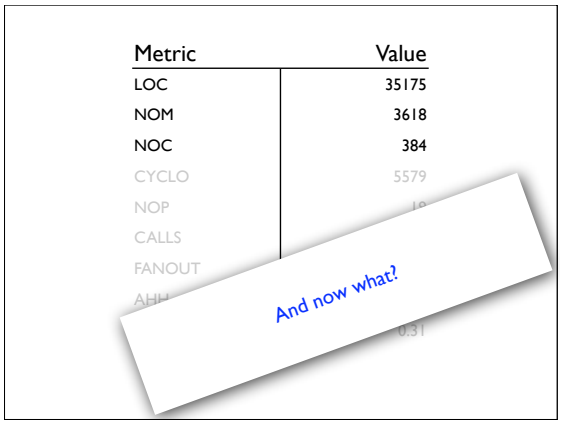
e.g. „insourced“ code
you are relocated to a new team

Improve the Code

e.g. refactor the design to make it portable
e.g. make my subsystem more flexible to a change of requirements

How to get an initial understanding of a system?

Metric	Value
LOC	35175
NOM	3618
NOC	384
CYCLO	5579
NOP	19
CALLS	15128
FANOUT	8590
AHH	0.12
ANDC	0.31



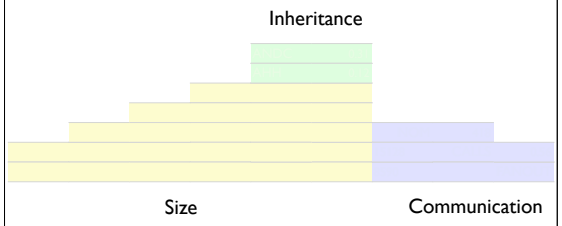
We need means to compare.

hierarchies?

coupling?

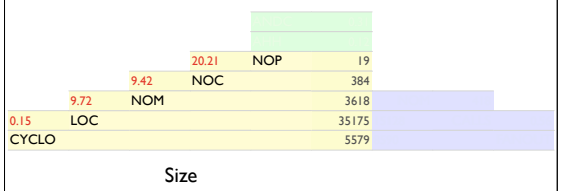
The Overview Pyramid provides a metrics overview.

Lanza, Marinescu 2006



The Overview Pyramid provides a metrics overview.

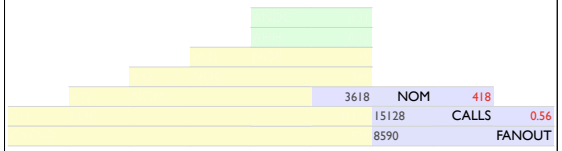
Lanza, Marinescu 2006



Size

The Overview Pyramid provides a metrics overview.

Lanza, Marinescu 2006



Communication

CALLS: Number of operation calls
FANOUT: Number of Called Classes

The Overview Pyramid provides a metrics overview.

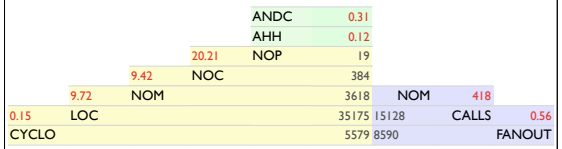
Lanza, Marinescu 2006

Inheritance

ANDC: Average Number of Derived Classes
AHH: Average Hierarchy Height

The Overview Pyramid provides a metrics overview.

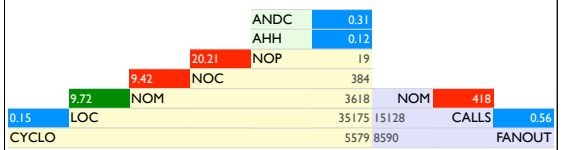
Lanza, Marinescu 2006



	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
...						

The Overview Pyramid provides a metrics overview.

Lanza, Marinescu 2006

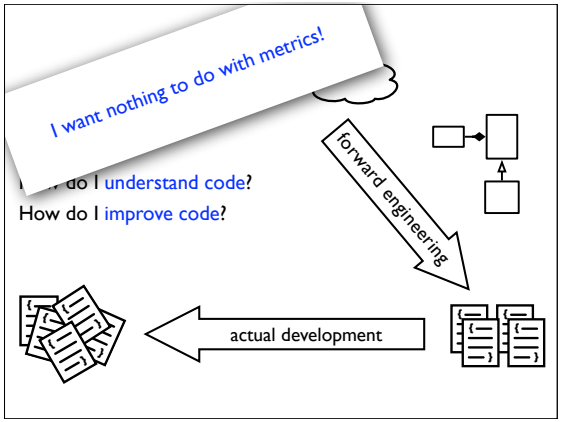


close to high | close to average | close to low

The Overview Pyramid provides a metrics overview.

Lanza, Marinescu 2006

close to high | close to average | close to low



How do I improve code?

Understand the Code

- e.g. „insourced“ code
- you are relocated to a new team

Improve the Code

- e.g. refactor the design to make it portable
- e.g. make my subsystem more flexible to a change of requirement
- I want to have NOTHING TO DO with metrics! ;-)

Quality is more than 0 bugs.

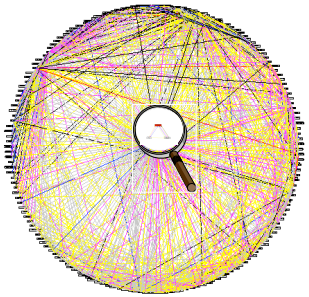
Breaking design principles, rules and best practices

deteriorates the code;

it leads to design problems.

Imagine changing just a small design fragment

and 33%
of all classes
would require changes



Design problems are

expensive
frequent
unavoidable

How to detect and eliminate them?

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

Riel, 1996

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

God Classes
centralize the intelligence of the system,
do everything and
use data from small data-classes.

God Classes

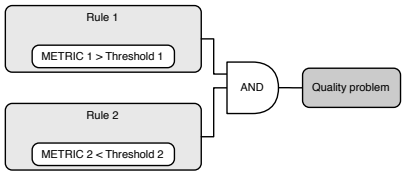
are complex,
are not cohesive,
access external data.

God Classes

are complex,	WMC is high
are not cohesive,	TCC is low
access external data.	ATFD more than few

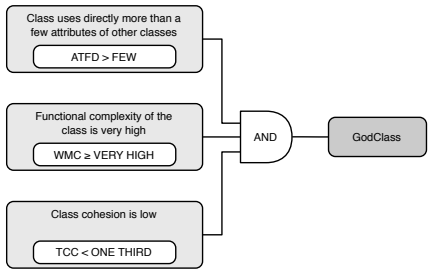
Compose metrics into queries using logical operators

Detection Strategies are metric-based queries to detect design flaws. Lanza, Marinescu 2006



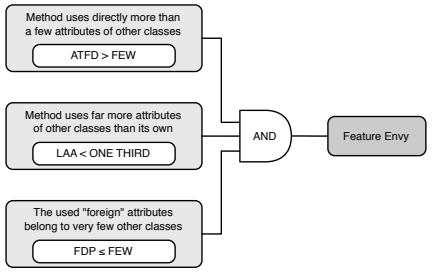
A God Class centralizes too much intelligence in the system.

Lanza, Marinescu 2006



An Envious Method is more interested in data from a handful of classes.

Lanza, Marinescu 2006

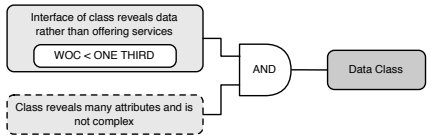


Feature Envy - Martin Fowler, Kent Beck, John Brant, William Opdyke and Don Roberts, Refactoring: Improving the Design of Existing Code, Addison Wesley, 1999.

ATFD: access to foreign data, counts distinct attributes accessed from other classes
LAA: Locality of attribute accesses
FDP: foreign data providers

Data Classes are dumb data holders.

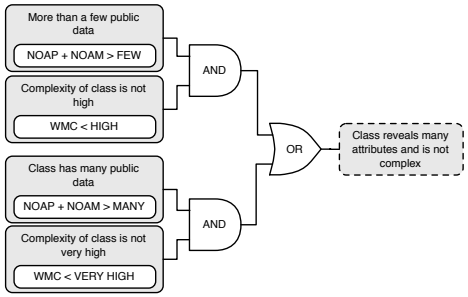
Lanza, Marinescu 2006



WOC: weight of class

Data Classes are dumb data holders.

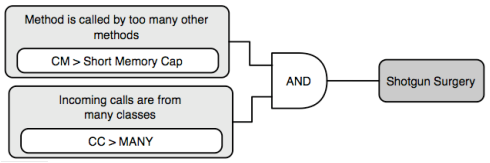
Lanza, Marinescu 2006



NOAP: number of public attributes
NOAM: number of accessor methods

Shotgun Surgery depicts that a change in an operation triggers many (small) in a lot of different operation and classes.

Lanza, Marinescu 2006



CM = Changing Methods (Number of calls)
CC = Changing Classes

Code Duplication

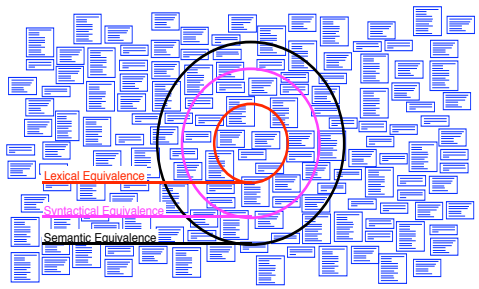
3

What is Code Duplication?

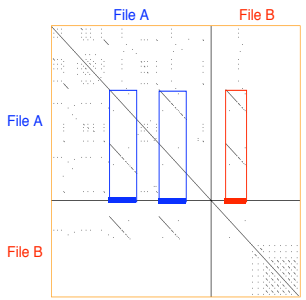
What are the problems of it?

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

Code Duplication Detection



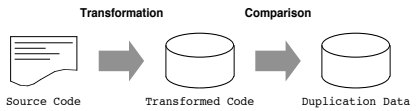
Visualization of Copied Code Sequences



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)



Author	Level	Transformed Code	Comparison Technique
Johnson 94	Lexical	Substrings	String-Matching
Ducasse 99	Lexical	Normalized Strings	String-Matching
Baker 95	Syntactical	Parameterized Strings	String-Matching
Mayrand 96	Syntactical	Metrics Tuples	Discrete comparison
Kontogiannis 97	Syntactical	Metrics Tuples	Euclidean distance
Baxter 98	Syntactical	AST	Tree-Matching

Noise Elimination

```
…
//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+]
```

Assumption:

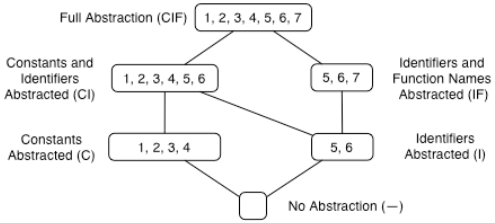
- Code segments are just copied and changed at a few places

Noise elimination transformation

- remove white space, comments
- remove lines that contain uninteresting code elements

(e.g., just ‘else’ or ‘}’)

Enhanced Simple Detection Approach



Code Comparison Step

As before, but now

Collect consecutive matching lines into match sequences

Allow holes in the match sequence

Evaluation of the Approach

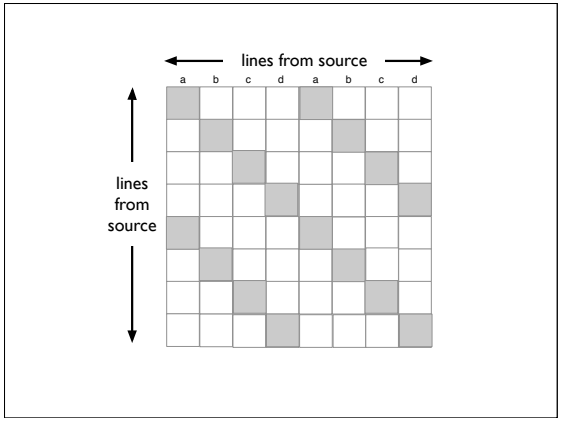
Advantages

Identifies more real duplication, language independent

Disadvantages

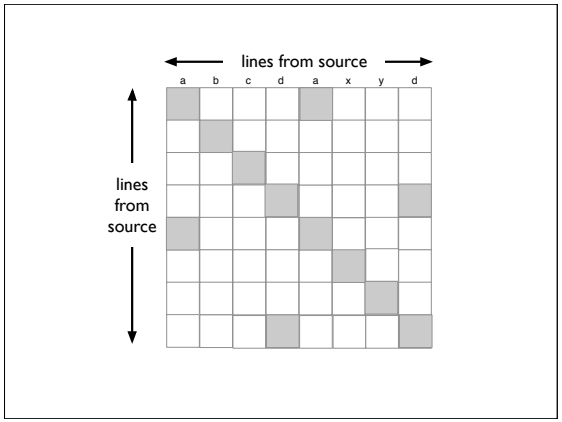
Less simple

Misses copies with (small) changes on every line

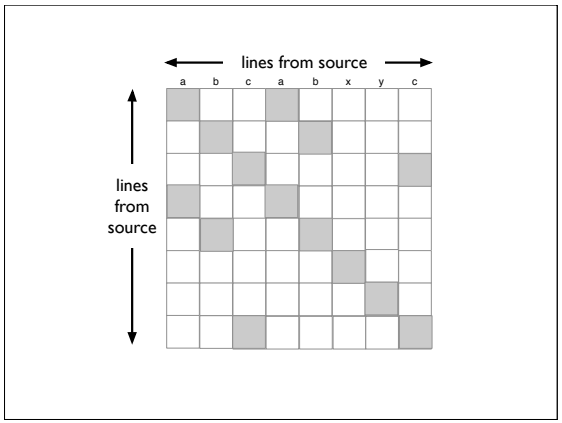


Visualization provides insights into the duplication situation
A simple version can be implemented in three days
Scalability issue

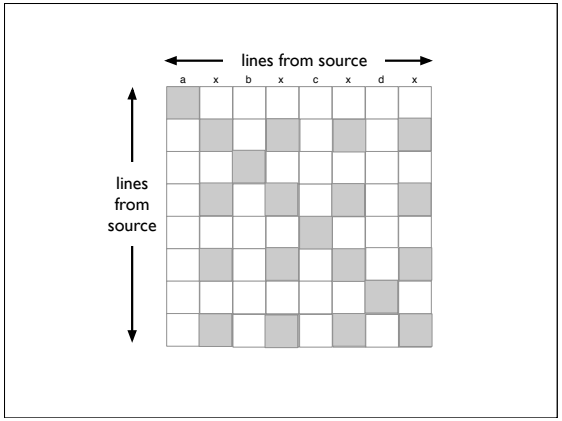
Dotplots — Technique from DNA Analysis
Code is put on vertical as well as horizontal axis
A match between two elements is a dot in the matrix
Exact Copies



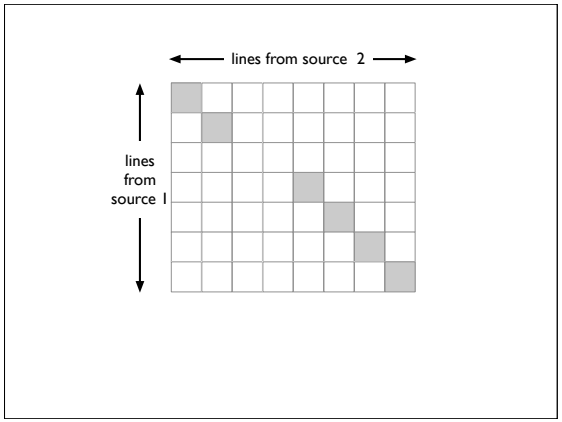
Copies with variations



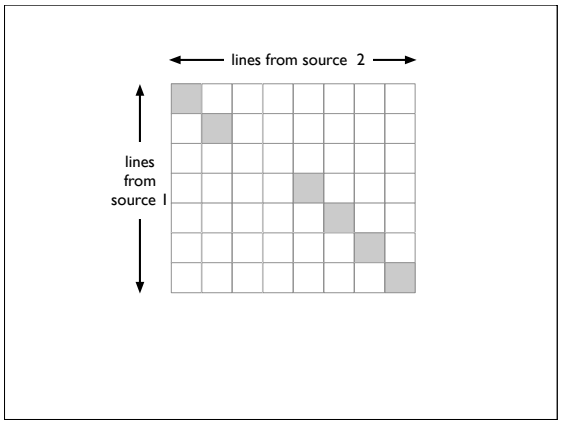
Insert / Delete



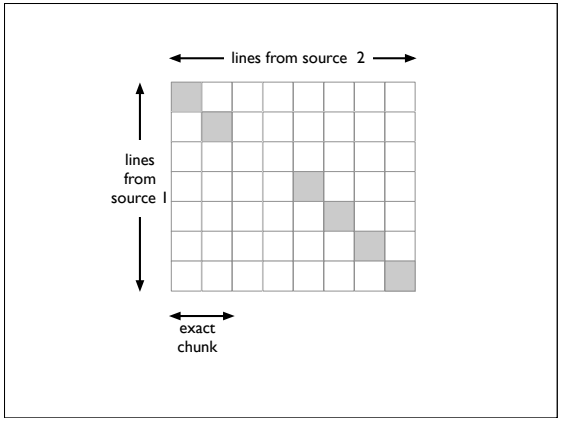
Repetitive Code Elements

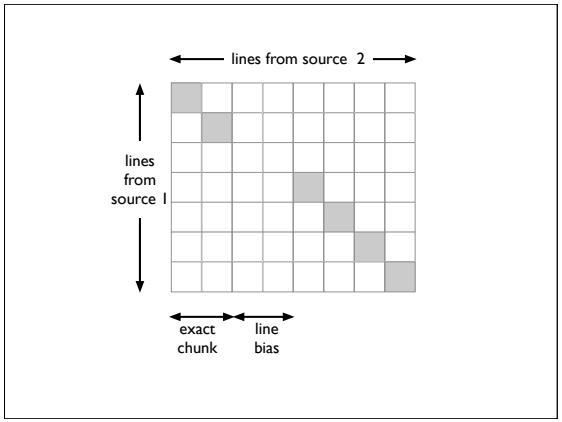


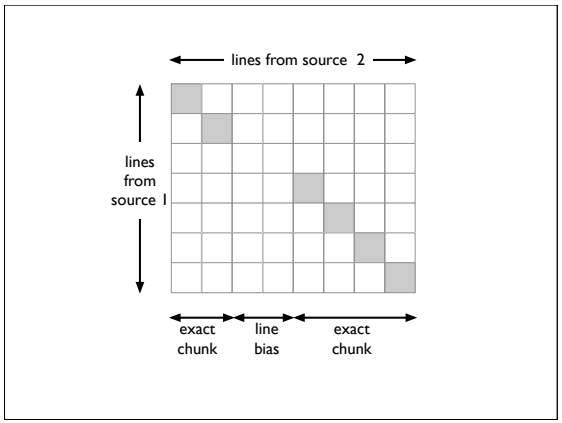
Mihai Balint, Tudor Gîrba and Radu Marinescu, “How Developers Copy,” Proceedings of International Conference on Program Comprehension (ICPC 2006), 2006, pp. 56—65



Mihai Balint, Tudor Gîrba and Radu Marinescu, **“How Developers Copy,”** *Proceedings of International Conference on Program Comprehension (ICPC 2006)*, 2006, pp. 56—65


lines from source 2
lines from source 1
exact chunk

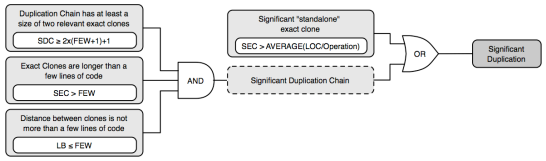

lines from source 2
lines from source 1
exact chunk
line bias


lines from source 2
lines from source 1
exact chunk
line bias
exact chunk

Significant Duplication:

- It is the largest possible duplication chain uniting all exact clones that are close enough to each other.
- The duplication is large enough.

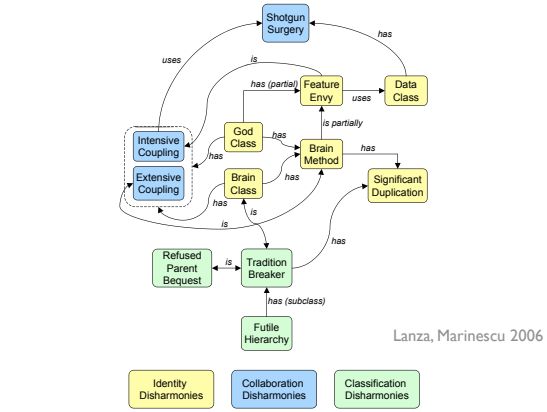
Lanza, Marinescu 2006



1 Metrics

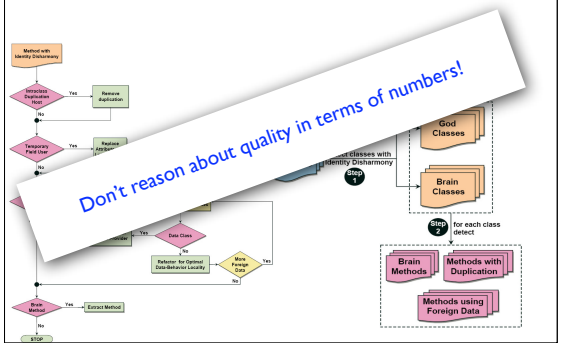
2 Design Problems

3 Code Duplication

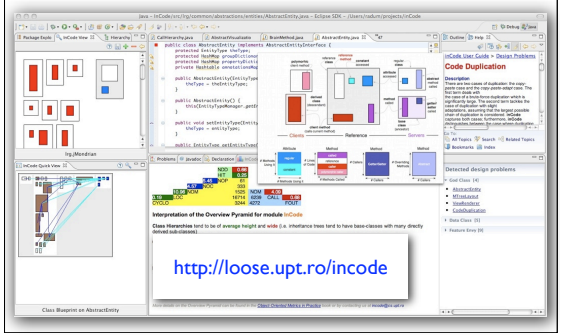


SEC: Size of Exact Clone measures the size of a clone in terms of lines of code.
SDC: Size of Duplication chain, a duplication chain is a block of duplication composed of exact clones that are close enough to be considered as belonging together.
LB: Line Bias is the distance between two consecutive exact clones

Follow a clear and repeatable process



QA is part of the the Development Process



Tudor Gîrba
www.tudorgirba.com

Jorge Ressia