New Frontiers of Reverse Engineering

Gerardo Canfora
Massimiliano Di Penta
FOSE / ICSE 2007

Reverse engineering is analyzing a subject system to:

identify components and their relationships, and

create more abstract representations.

Chikofky & Cross, 90

Why reverse engineer?

In 1944, 3 B-29 had to land in Russia

Requirement: Copy everything fast!

Approach: disassemble, run, test

TU-4 Result: 105,000 pieces reassembled in 2 years

Reading code...

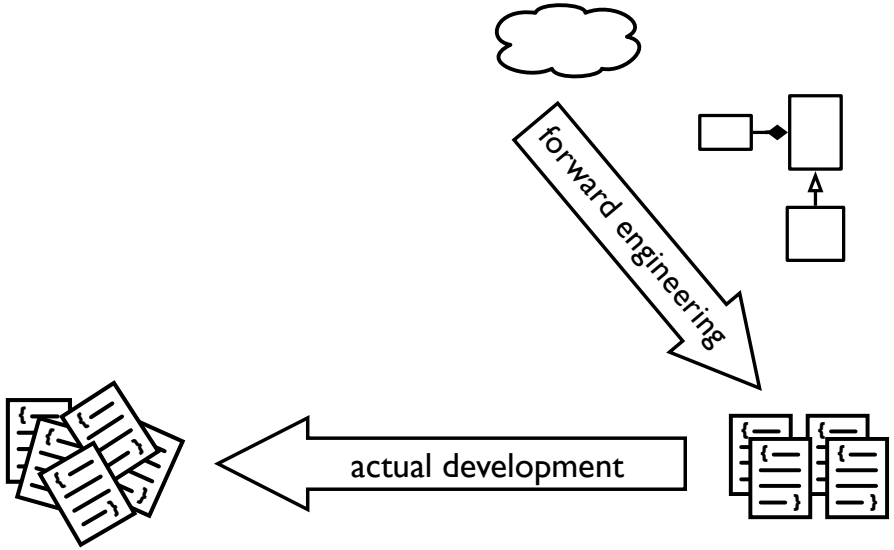
100'000 lines of code

* 2 = 200'000 seconds

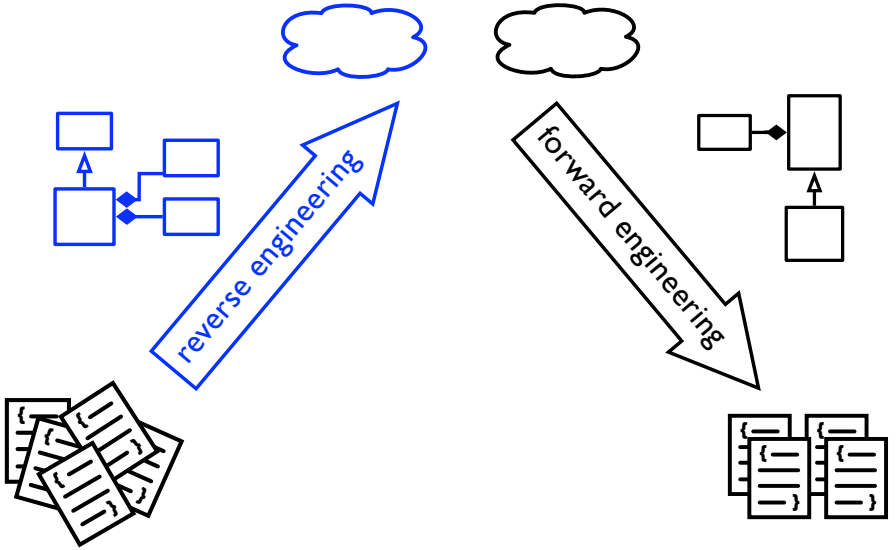
/ 3600 = 56 hours

/ 8 = 7 days

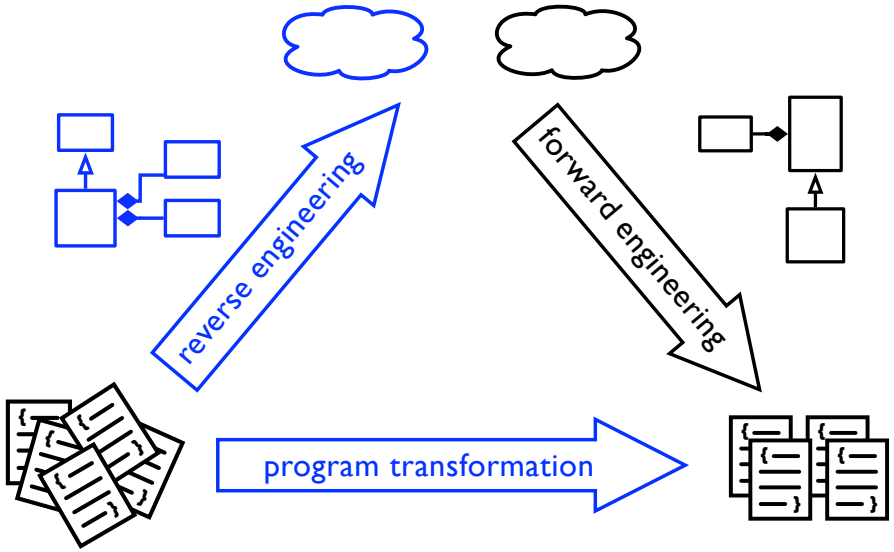
How development happens



Reengineering life cycle



Reengineering life cycle



Where is the "real action" happening?

International Conference on Software Maintenance (26)

Working Conference on Reverse Engineering (16)

Intl. Workshop on Source Code Analysis and Manipulation (10)

International Conference on Program Comprehension (17)

Directions in Reverse Engineering

You got to be careful if you don't know where you're going, because you might not get there.

Yogi Berra

1 Software Understanding

2 Design Recovery

1 Software Understanding

Achievements

Understanding and Migrating Procedural Code

Create Models and Model Extractors/Parsers

Clone Detection and Analysis

Aspect Mining

Visualizing Software Artifacts

1 Software Understanding

Achievement 1.1: Understanding and Migrating Procedural Code

Cope with Y2K problem

Created many intermediary representations

Identify objects in legacy code

Migration from procedural to object-oriented system

- issue of changing the language without changing the paradigm

Achievements in Mid '90s

1 Software Understanding

Achievement 1.2: Create Models and Model Extractors/Parsers

Parsing source-code and performing rule-base transformations

Parsing problems due

a. diversity of language (500 problem)

b. macros and preprocessor directives (C/C++)

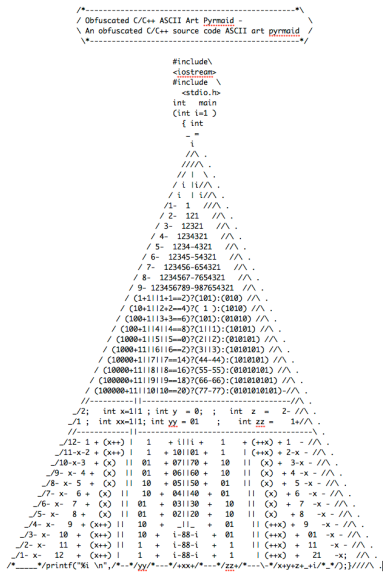
Island/Lake Parsing

parse only program constructs that you care about [Moonen]

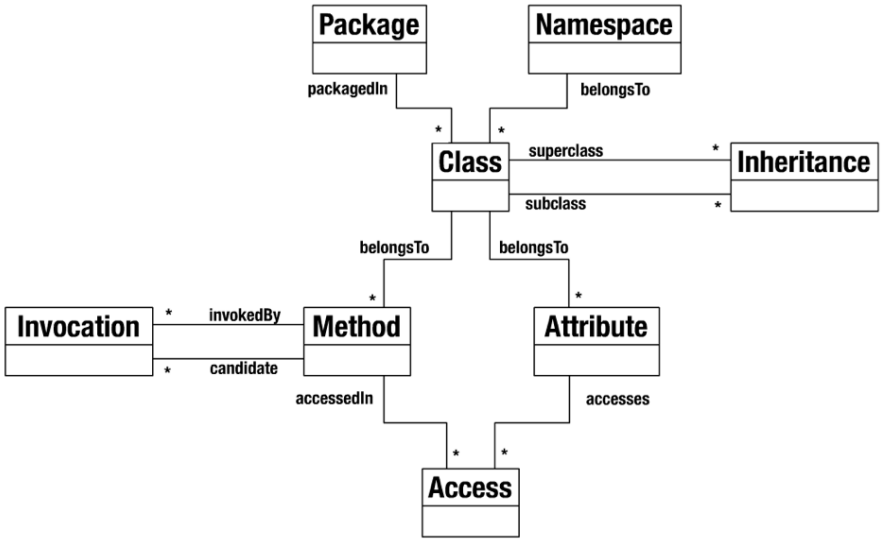
NEW!

Use parsing facilities in IDEs

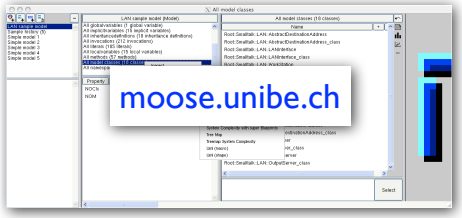
Parsing can be a problem...

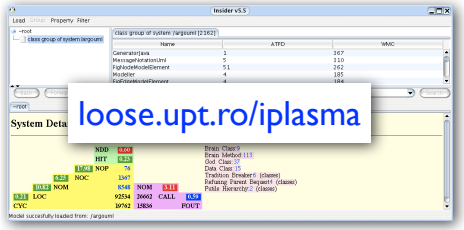


Meta-Model Example: FAMIX

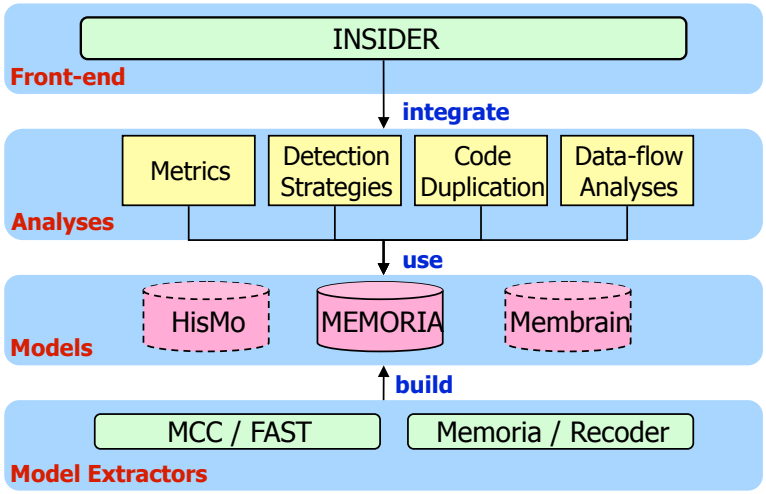


Model Extractors and Analysis Toolkits





Example: iPlasma Toolkit



1 Software Understanding

2 Design Recovery

1 Software Understanding

Achievement 1.3: Clone Detection and Analysis

Two Issues

a. precision (issue: false positives)

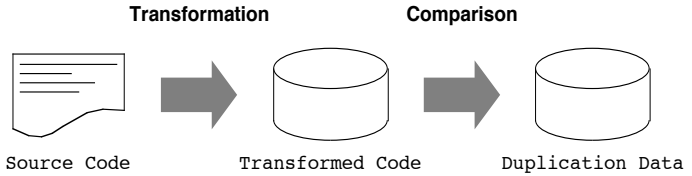
b. recall (issue: false negatives)

Three Approaches

a. token-based (high recall > yet, much noise)

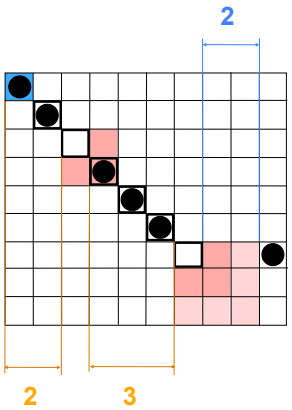
b. AST-based (high precision > yet, misses true cases)

c. metrics-based (language-independent)



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

Chains of Duplication



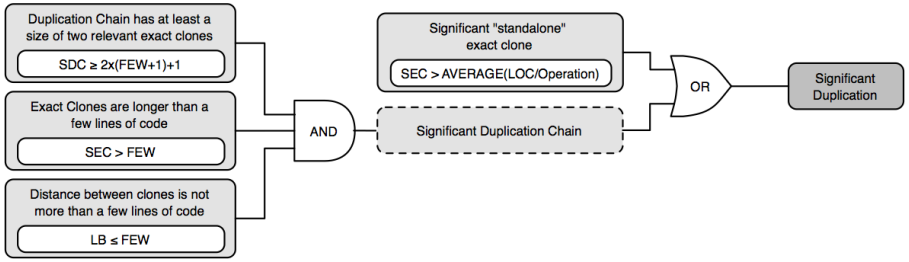
Line bias (LB)

Minimum length of a fragment
(SEC - Standalone Exact Clone)

Significant Duplication Chain (SDC)

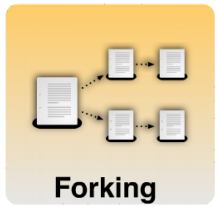
Archeology of Code Duplication

Wettel, Marinescu 2005
Lanza, Marinescu, 2006

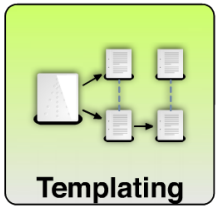


"Cloning Considered Harmful" considered harmful

Kapser, Godfrey 2006



Example:
Hardware Variation



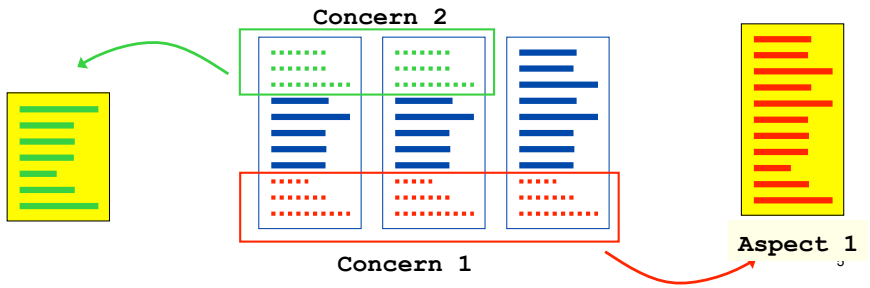
Example:
API/Library
Protocols

Example:
Bug Workaround

Need to understand the reason of cloning before deciding if harmful

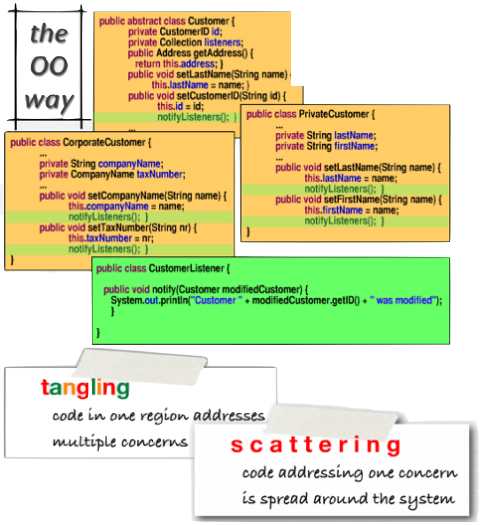
1 Software Understanding

Achievement 1.4: Aspect Mining



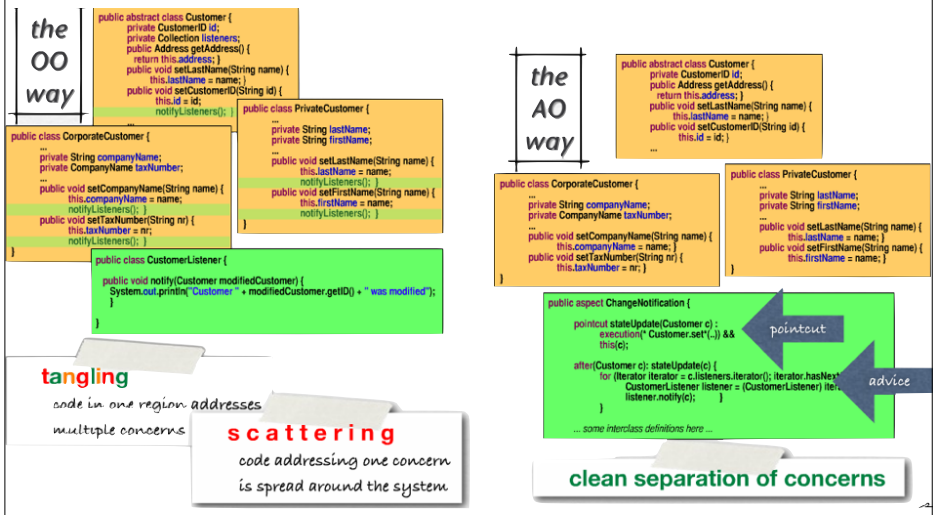
Separate the principal decomposition from crosscutting concerns

Aspects in a Nutshell



from K.Mens, A.Kellens, J.Krinke "Pitfalls in Aspect Mining"

Aspects in a Nutshell



from K.Mens, A.Kellens, J.Krinke "Pitfalls in Aspect Mining"

Aspect Mining Techniques

FANIN (methods called from many different places)

Clones

Correlation of line co-changes [Canfora]

1 Software Understanding

Achievement 1.5: Visualizing Software Artifacts

A picture is worth

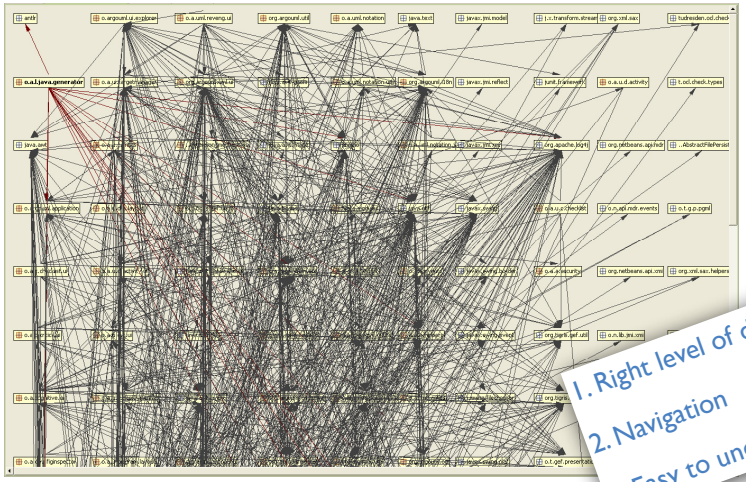
a **thousand words**.

[unknown]

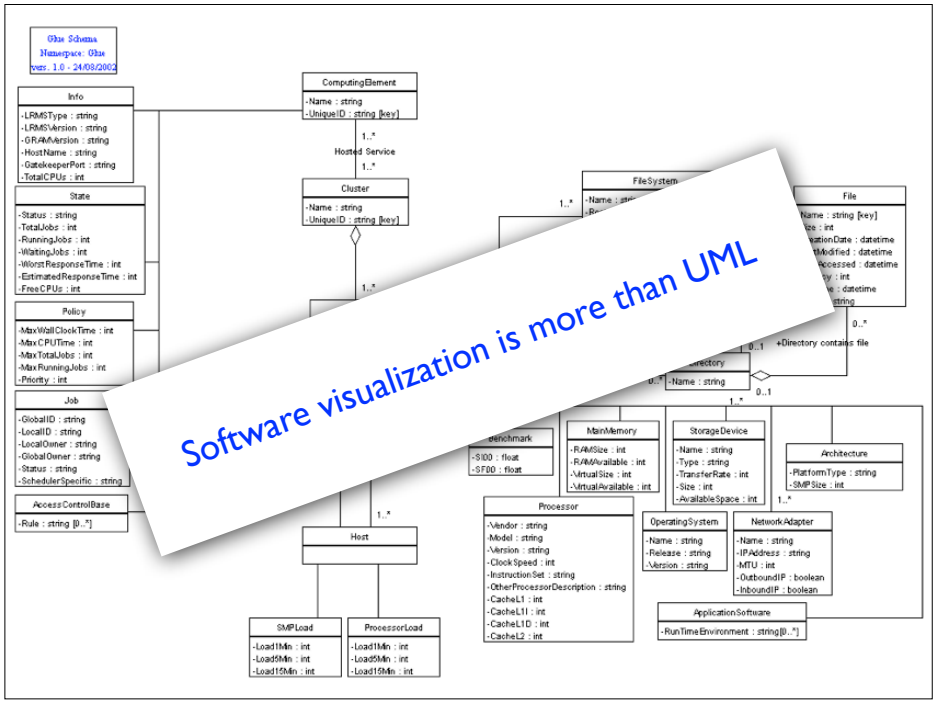
..depends on the picture

[Lanza]

Visualization does not guarantee understanding

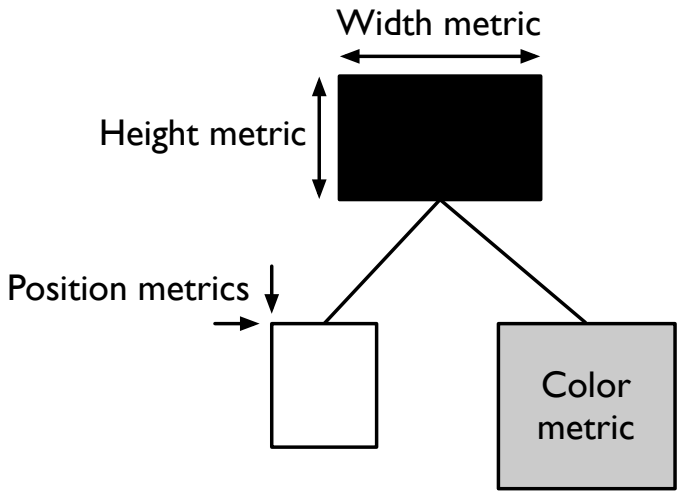


1. Right level of detail
2. Navigation
3. Easy to understand



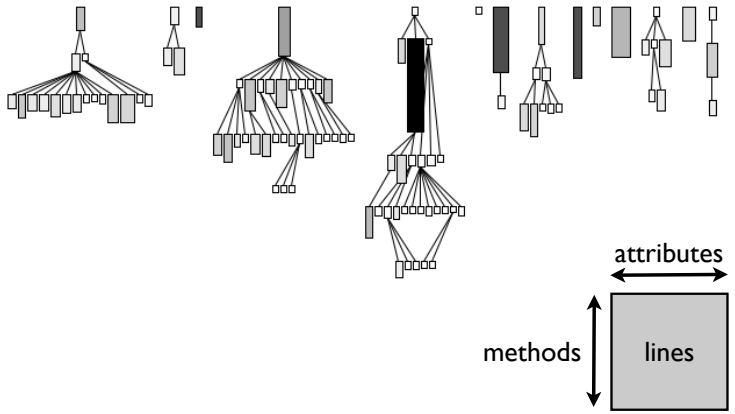
Polymetric Views show up to 5 metrics.

Lanza, 2003

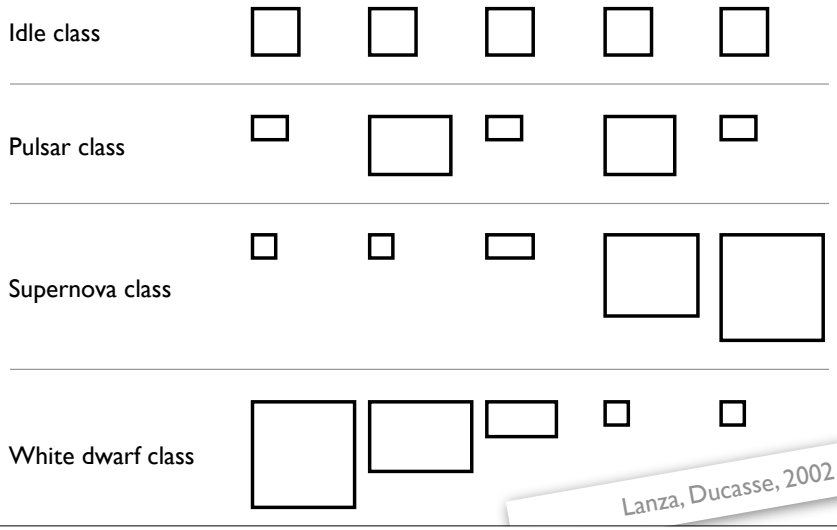


System Complexity shows class hierarchies.

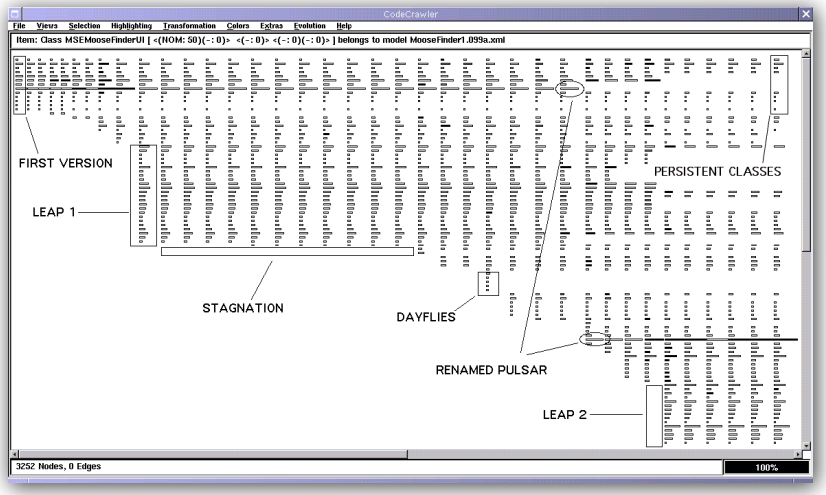
Lanza, Ducasse, 2003



Evolution Matrix shows changes in classes

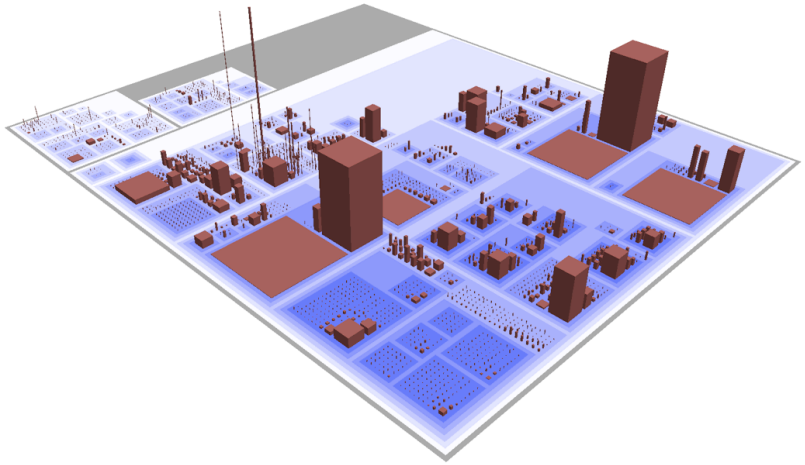


Lanza, Ducasse, 2002



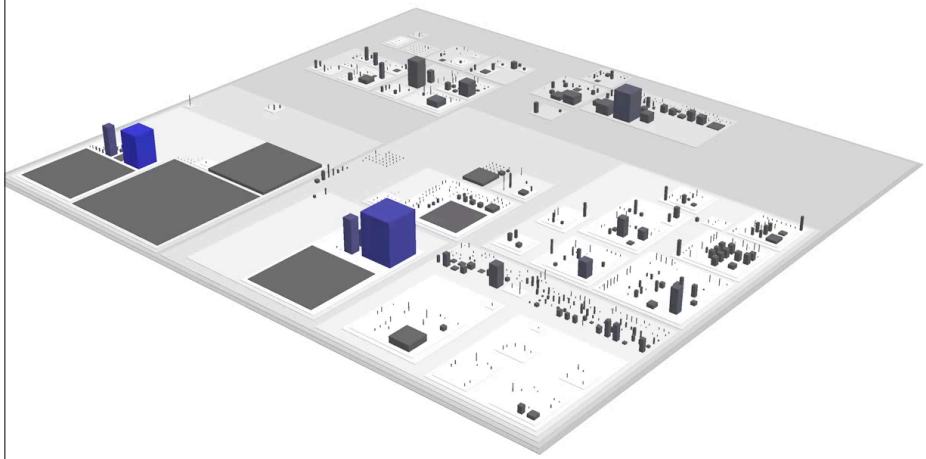
Code City shows where your code lives.

Wettel, Lanza, 2007



classes are buildings grouped in quarters of packages

Look carefully how the system evolves in time!



1 Software Understanding

Trends

Understanding Systems with High Dynamicity

Understanding Cross-Language Systems

Mining Software Repositories

1 Software Understanding

Trend 1.1: Understanding Systems with High Dynamicity

Reflection and loading classes at run-time

BAD

- affects points-to analysis
- Solution: DA must complement SA

GOOD

- access to members of classes ;
- support for analysis JVMTI instead of instrumenting code

1 Software Understanding

Trend 1.2: Understanding Cross-Language Systems

Especially in the case of Web Applications (and .NET)

How to analyze in an **integrated manner** different types of code that coexist

- HTML,
- Object-oriented,
- SQL,
- Scripting code

1 Software Understanding

Trend 1.3: Mining Software Repositories

Analyze versioning systems (study the evolution of software)

- Changes correlated with other faultiness [Zeller]
- Co-changes correlation with code duplication [Geiger etal]
- Co-changes to refine detection of design flaws [Ratiu etal]

Analyze developers behavior

Moved from 1D (SA)
to 2D (SA + DA)
to 3D (SA + DA + Time)

2 Design Recovery

Achievements

Recovery of UML Models (class and object diagrams)

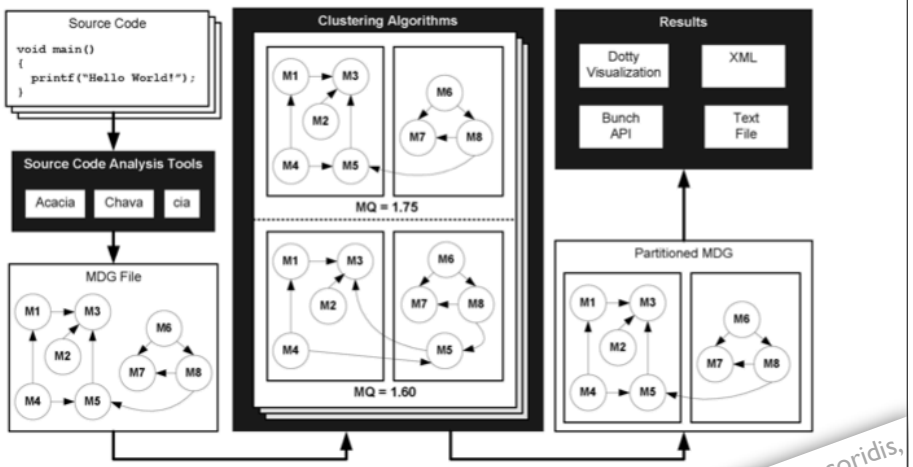
Identifying Design Patterns (motifs) in code (Guéhéneuc)

Clustering-Based Architecture Recovery

Feature/Concept Location

2 Design Recovery

Achievement 2.2: Clustering-Based Architecture Recovery



Mitchell, Mancoridis, TSE 2006

Formal Concept Analysis (FCA) in a nutshell.

Incidence Table

elements \ properties	P1	P2	P3	P4	P5	P6
A		X			X	X
B		X		X		
C			X		X	X
D		X		X		X
E					X	X

Legend:

X — a binary relation between the element on the row and the property on the column

from T.Girba "Modeling History to Understand Software Evolution". PhD 2003

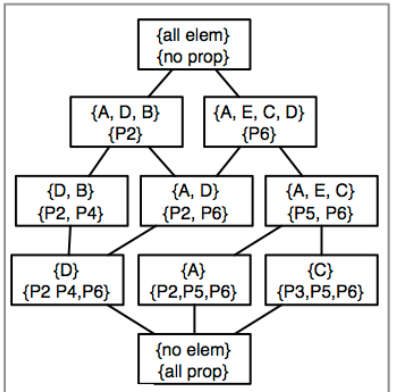
Formal Concept Analysis (FCA) in a nutshell.

Incidence Table

elements \ properties	P1	P2	P3	P4	P5	P6
A		X			X	X
B		X		X		
C			X		X	X
D		X		X		X
E					X	X

FCA

Concept Latice



Legend:

X — a binary relation between the element on the row and the property on the column

{X, Y} {P1, P2} — a concept representing X and Y having in common property P1 and P2

from T.Girba "Modeling History to Understand Software Evolution". PhD 2003

2 Design Recovery

Achievement 2.2: Clustering-Based Architecture Recovery

Element = Function

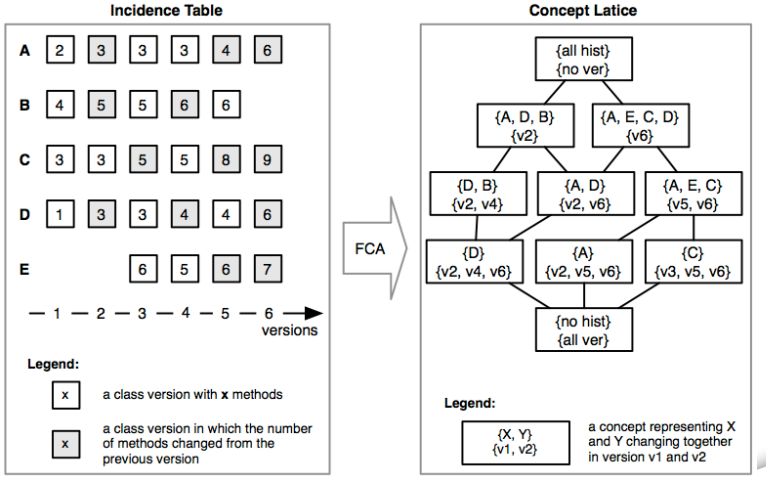
Property = Type used by Function

	returns `stack`	*returns* `queue`	*has* `stack` *arg.*	*has* `queue` *arg.*	*uses* `stack` *fields*	*uses* `queue` *fields*	*not* `queue` *fields*
`initStack`	√				√		√
`initQ`		√				√	
`isEmptyS`			√		√		√
`isEmptyQ`				√	√	√	
`push`			√		√		√
`enq`				√	√	√	
`pop`			√		√		√
`deq`				√		√	

Siff, Reps, TSE 1999

2 Design Recovery

Achievement 2.2: Clustering-Based Architecture Recovery



Girba, PhD, 2003

2 Design Recovery

Trends

Extraction of Object Diagrams and Constraints (pre/post conditions)

Migration to Web 2.0 applications

Interactive RE environments

2 Design Recovery

Trend 2.3: Interactive RE environments

RE: not only source code info; but also developers rationales

2 Problems: (i) incompleteness ; (ii) semi-automatic

Solution: interactively improve artifact presentation

- give feedback to RE system (efficiently!)
- RE systems should learn (machine learning, GA)
- best in IDEs

Issues and Challenges

A Continuous RE

B RE for SOA and Autonomic Computing

A Continuous RE

Exploit RE in the Fwd. Eng. process (especially in IDEs)

Benefits

1. Clearer picture (understanding) of the developed system
2. Permanent consistency checks (design vs. code vs. tests)
3. Better QA hints
   - continuous monitoring
   - learning tools for better traceability links

B RE for SOA and Autonomic Computing

SOA - separation of software ownership (product) from software use (service)

Autonomic Computing - self-adaptation and self-evolution

B RE for SOA and Autonomic Computing

SOA

Each service offers a limited view

Danger: system as an orchestration of various services

discrepancy between terminology = harder to understand

no access to source-code

B RE for SOA and Autonomic Computing

Systems with Autonomic Capabilities

Autonomic Discovery

find and bind new services when default is not ok

Self-Healing

change composition, reconfiguring/repairing system, interrupting exec

Danger: high dynamism

pieces composing the execution known only at run-time

What about RE Tools as Composition of Services? :)