INRIA

RMod

Quality and Software Visualization

Dr. Stéphane Ducasse
stephane.ducasse@inria.fr
http://stephane.ducasse.free.fr/

RMOD expertise

RMod

Supporting software evolution and software composition

Axis 1: Reengineering

Maintaining large software systems

Moose: a powerful platform for reengineering

Nokia, Daimler, Harman-Becker, Siemens, Cincom

Axis 2: Dynamic languages to support evolution

Revisiting fundamental aspects of OO languages

Reuse Traits: Fortress (SUN Microsystems), Perl-6, Scala (EPFL), Squeak, Dr-Scheme,

Security and Dynamic Languages

A word of presentation

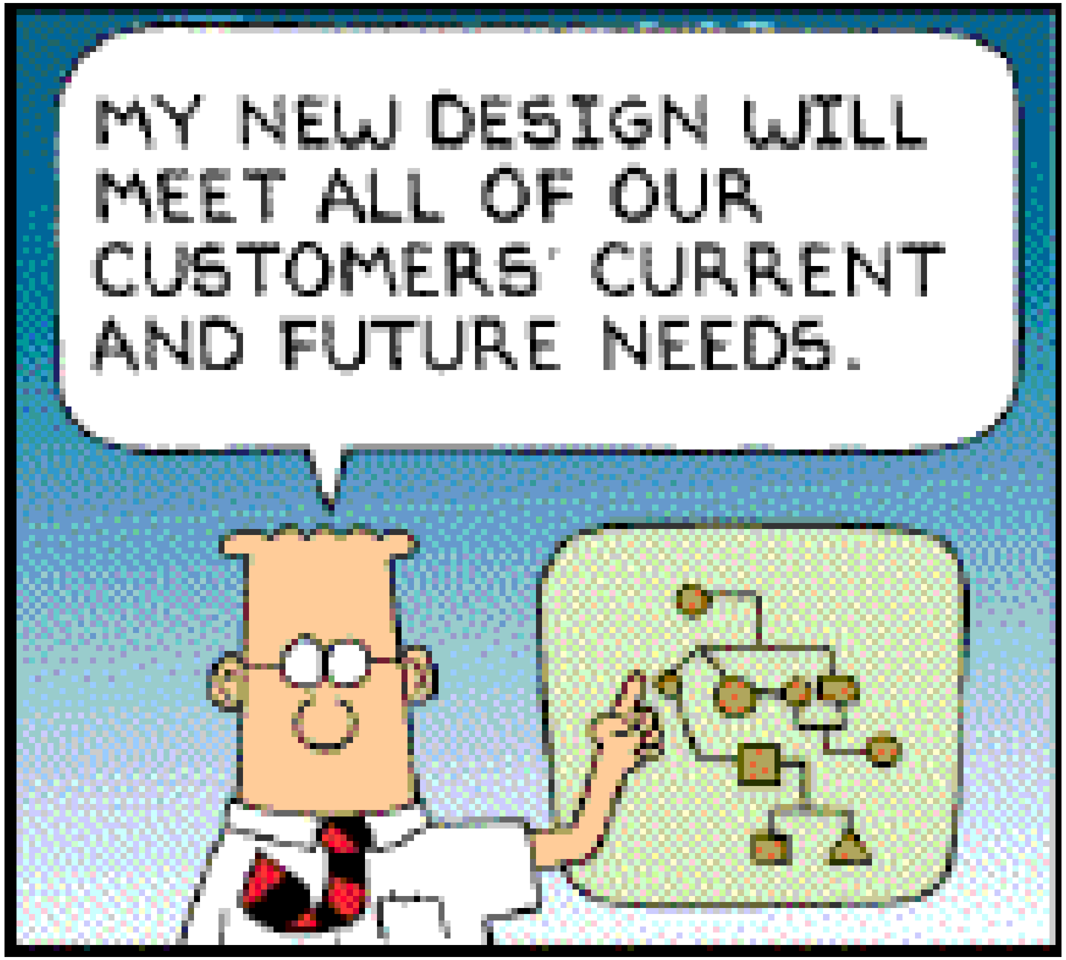
Since 1996 Moose (reengineering platform)

Object-Oriented Reengineering Patterns

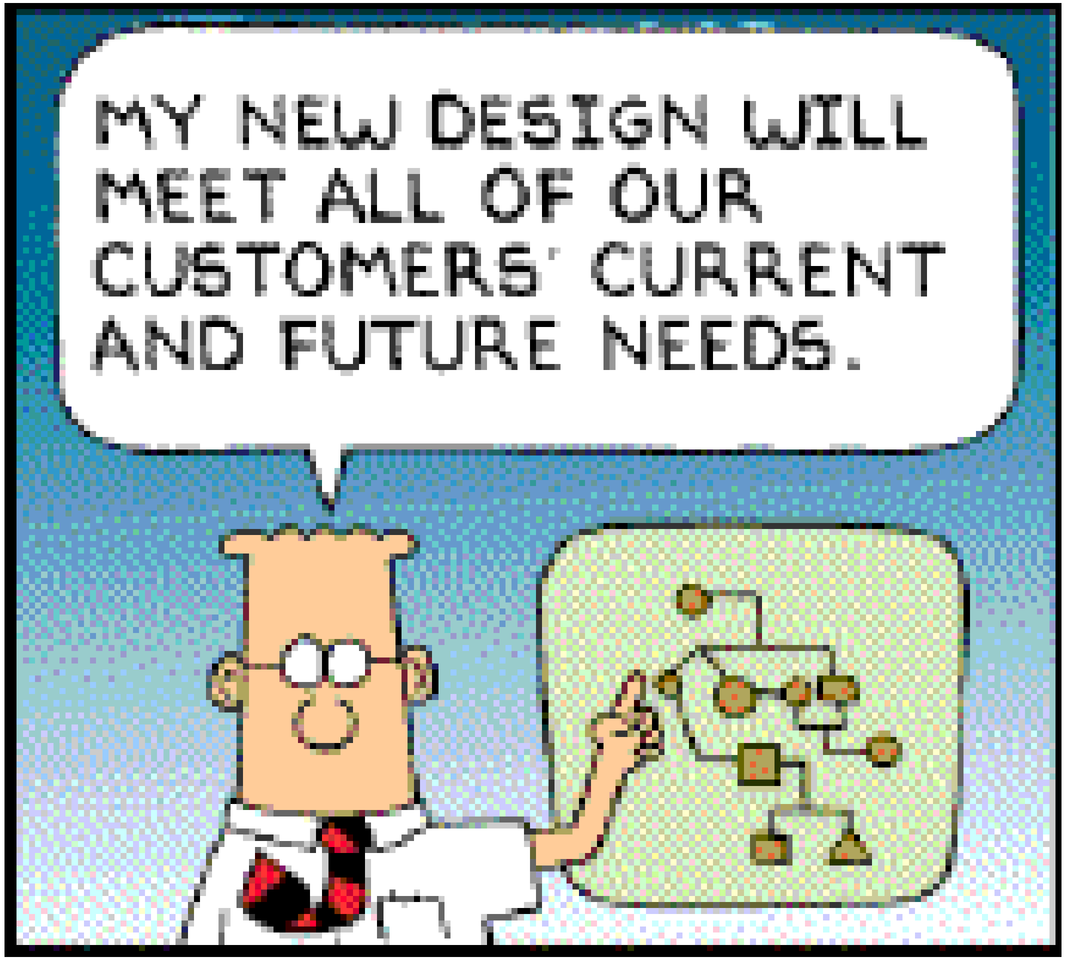
Grounded in reality
Maintainer of open-source projects

Worked with:

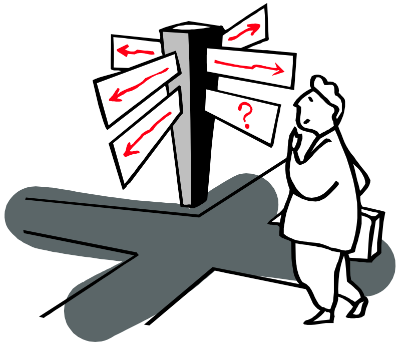
- Harman-Becker AG
- Bedag AG,
- Nokia, Daimler


MY NEW DESIGN WILL MEET ALL OF OUR CUSTOMERS' CURRENT AND FUTURE NEEDS.

Let’s face it, this is the Graal

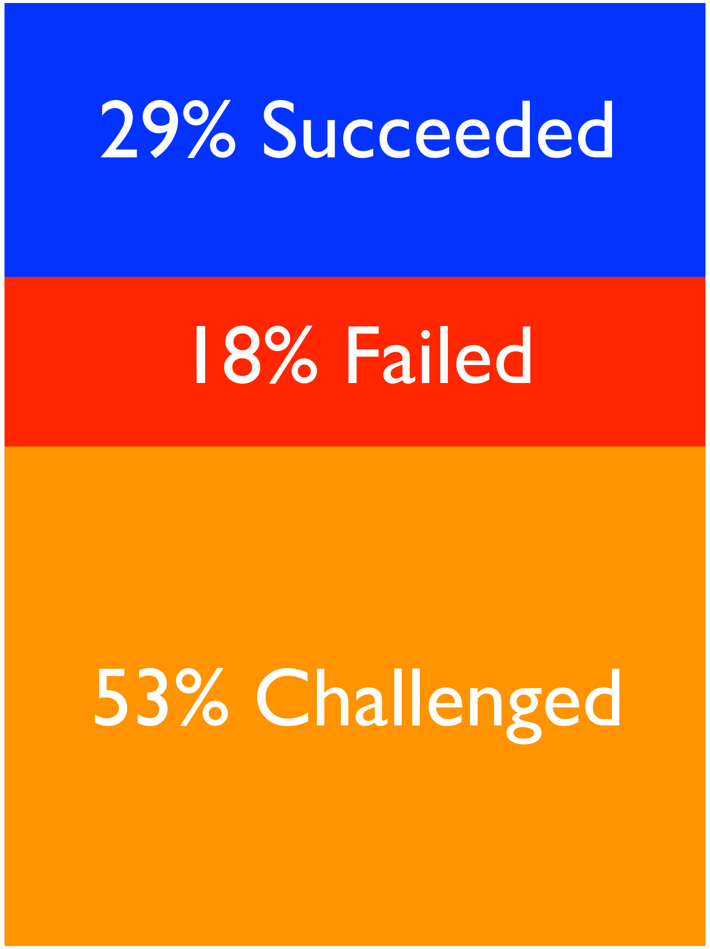

MY NEW DESIGN WILL MEET ALL OF OUR CUSTOMERS' CURRENT AND FUTURE NEEDS.

Roadmap



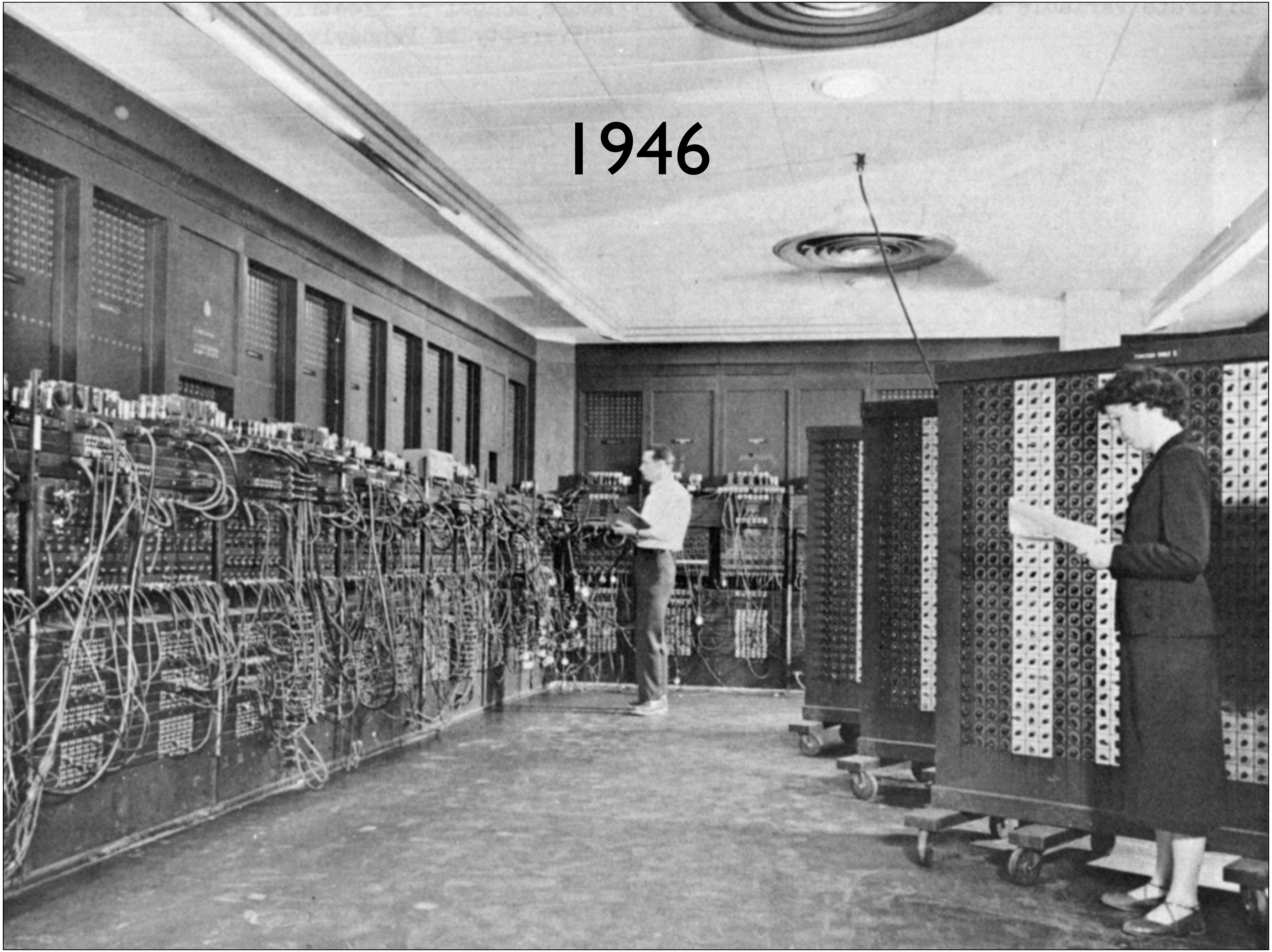
- ***Some software development facts***
- Our approach
  - Supporting maintenance
  - Moose an open-platform
- Visual principles in 3 min
- *Some* visual examples
- Conclusion

Software...



The Standish Group, 2004

Software is complex.

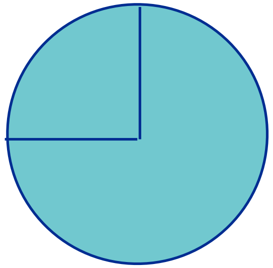


1946

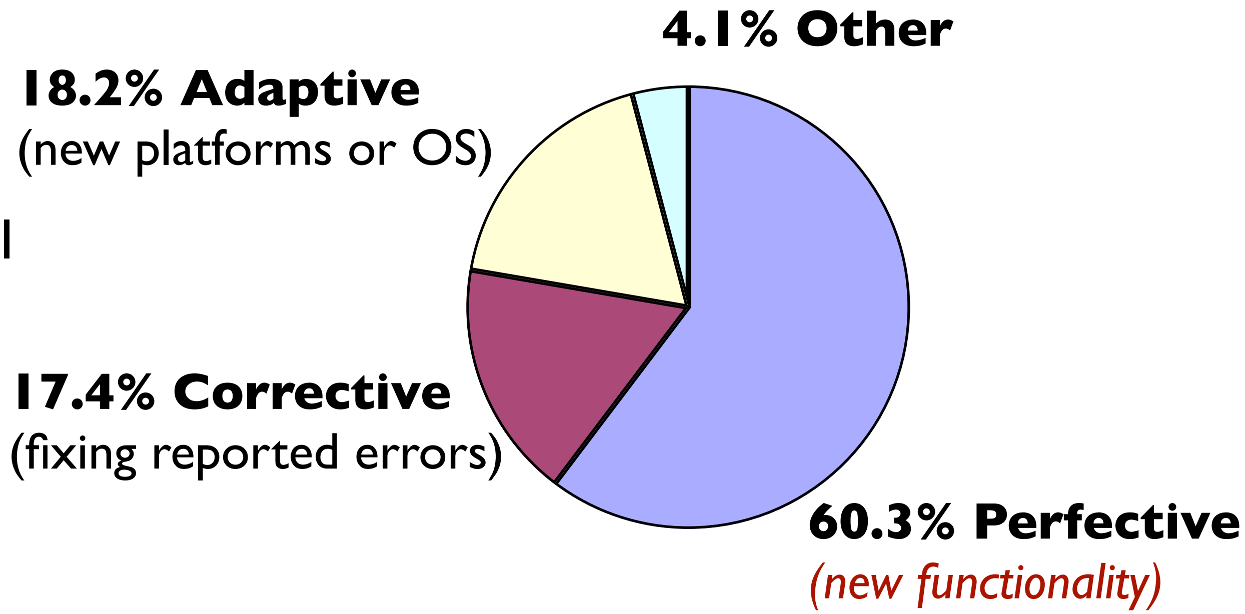
How large is your project?

1’000’000 lines of code
* 2 = 2’000’000 seconds
/ 3600 = 560 hours
/ 8 = 70 days
/ 20 = 3 months

Maintenance: ***Continuous*** Development



Between 50% and 75% of global effort is spent on “maintenance” !



The bulk of the maintenance cost is due to new functionality

even with better requirements, it is ***hard*** to predict new functions

Lehman’s Software Evolution Laws

Continuous Change: “A program that is used in a real-world environment must change, or become progressively less useful in that environment.”

Software Entropy: “As a program evolves, it becomes more complex, and extra resources are needed to preserve and simplify its structure.”

System evolution is like... SimCity

Software are living…

Early decisions were certainly good at that time

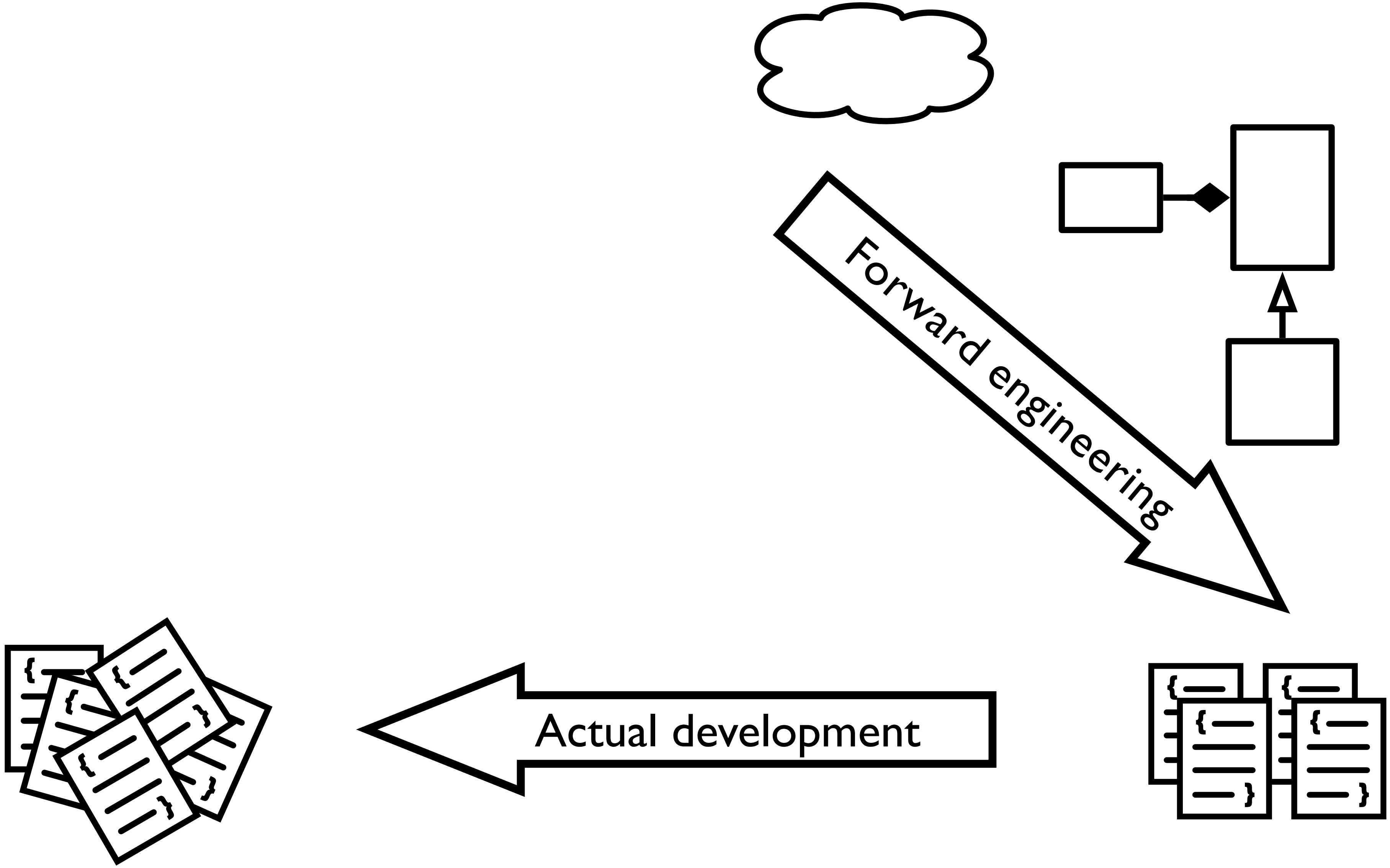
But the context ***changes***

Customers ***change***

Technology ***changes***

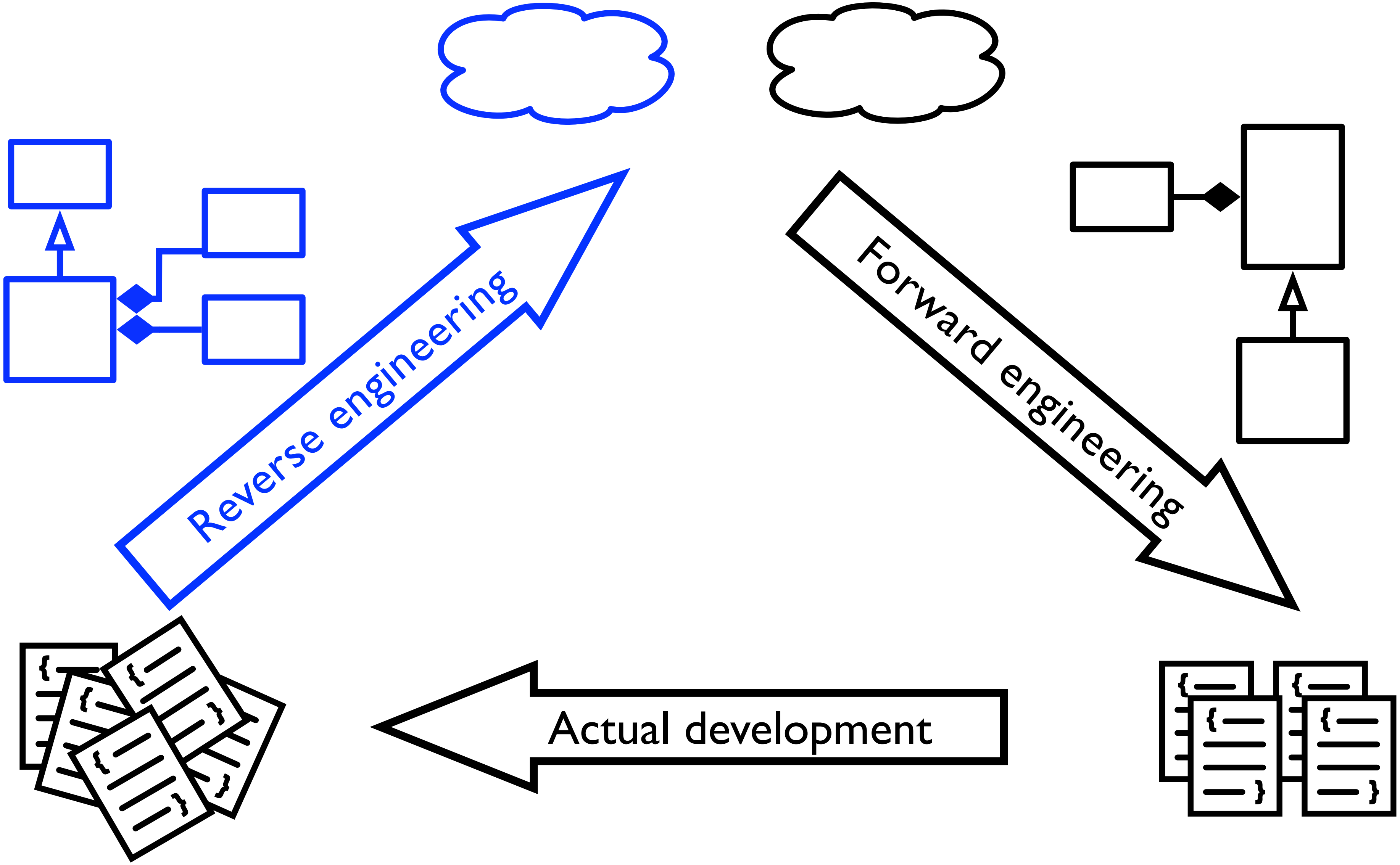
People ***change***

Software development is ***more*** than forward engineering.

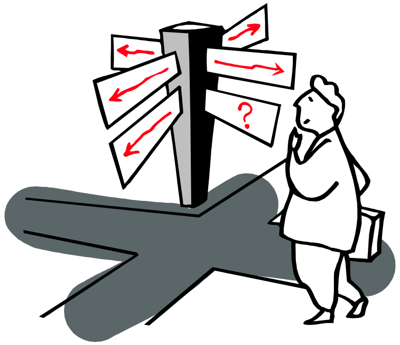


Maintenance is

is needed to evolve the code.

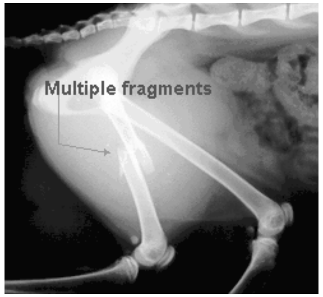


Roadmap



- Some software development facts
- ***Our approach***
  - Supporting maintenance
  - Moose an open-platform
- Visual principles in 3 min
- *Some* visual examples
- Conclusion

Help teams maintaining large software



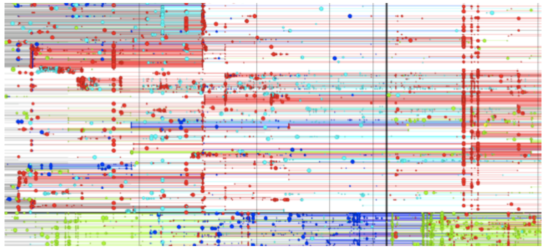
What is the xray for software?

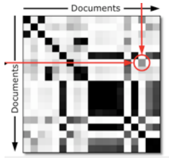
- code, people, practices

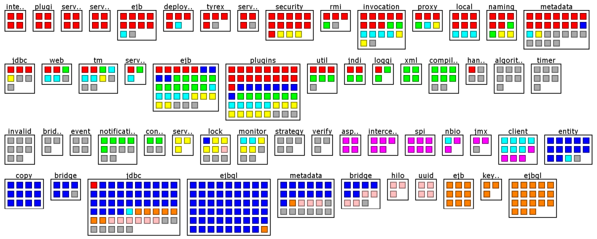
Which analyses?

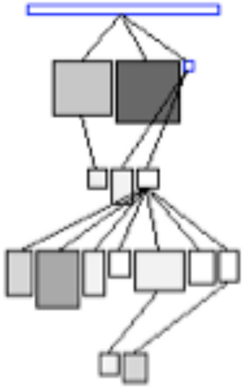
How can you monitor your system (dashboards....)

How to present extracted information?

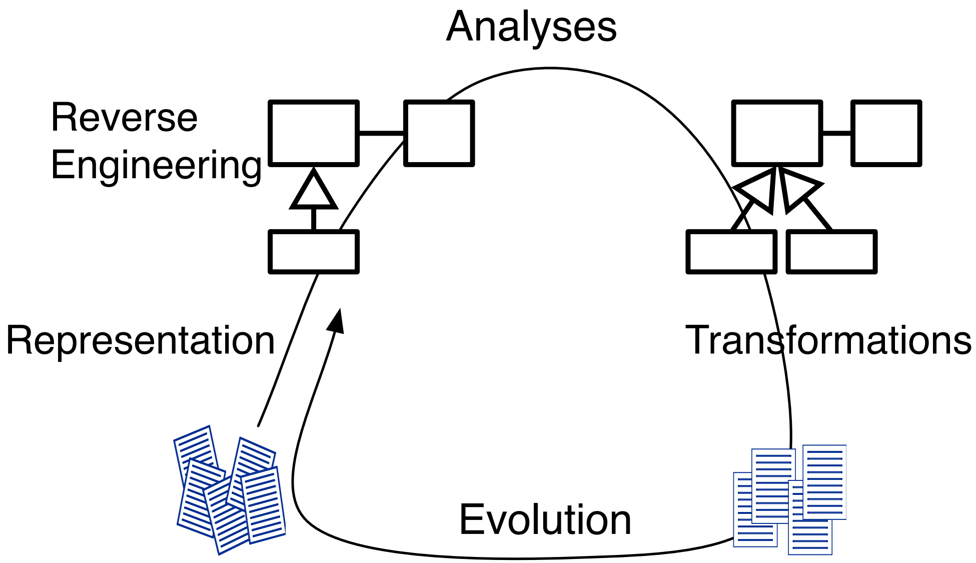








Since 1996...



Topics

Metamodeling, metrics, program understanding, visualization, evolution analysis, duplicated code detection, code Analysis, refactorings, test generation...

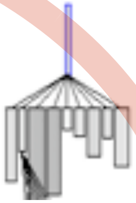
Contributions

Moose: an open-source extensible reengineering environment: (Lugano, Bern, Annecy, Anvers, Louvain la neuve, ULB, UTSL)

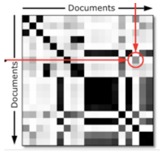
Contacts

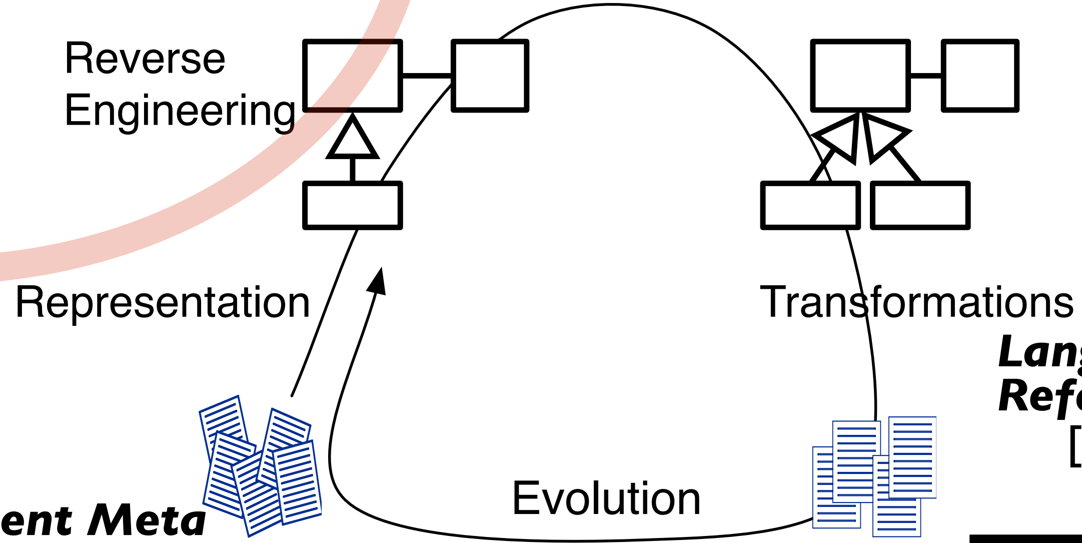
Harman-Becker (3 Millions C++), Bedag (Cobol), Nokia, ABB, IMEC

Understanding Large Systems
[WCRE99,TSI00,TSE03]
Static/Dynamic Information
[ICSM99]
Feature Analysis
[JSME 06]
Class Understanding
[OOPSLA01,TSE04]
Package Blueprints
[ICSM 07]
Distribution Maps
[ICSM 06]



Software Metrics
[LMO99, OOPSLA00]
Duplicated Code Identification
[ICSM99, ICSM02]
Group Identification
[ASE03]
Test Generation
[CSMR 06]
Concept Identification
[WCRE 06]





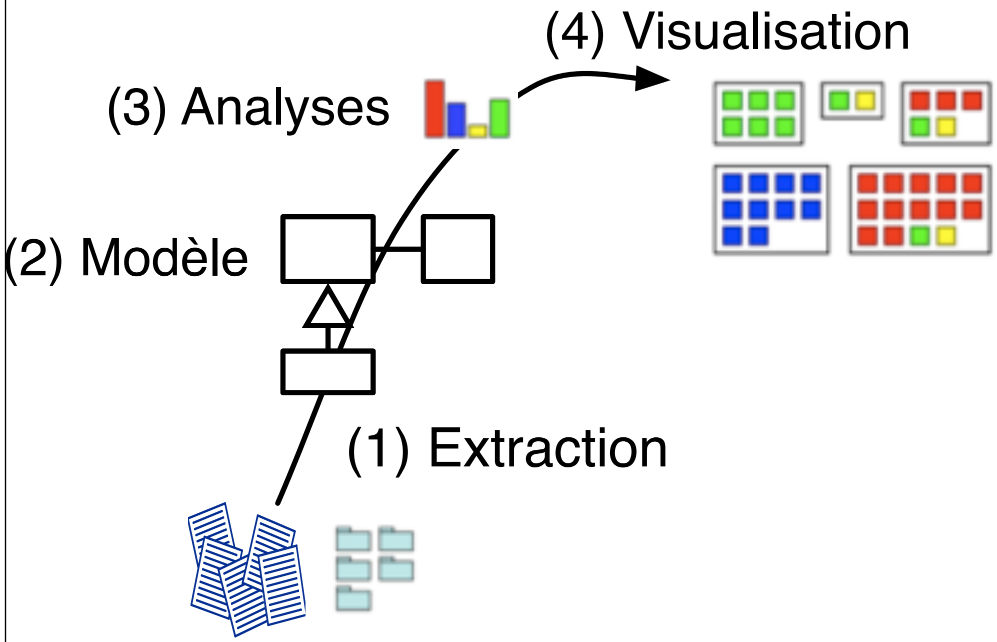
Language Independent Refactorings
[IWPSE 00]

Language Independent Meta Model (FAMIX)
[UML99]
An Extensible Reengineering Environment (Moose)
[Models 06]

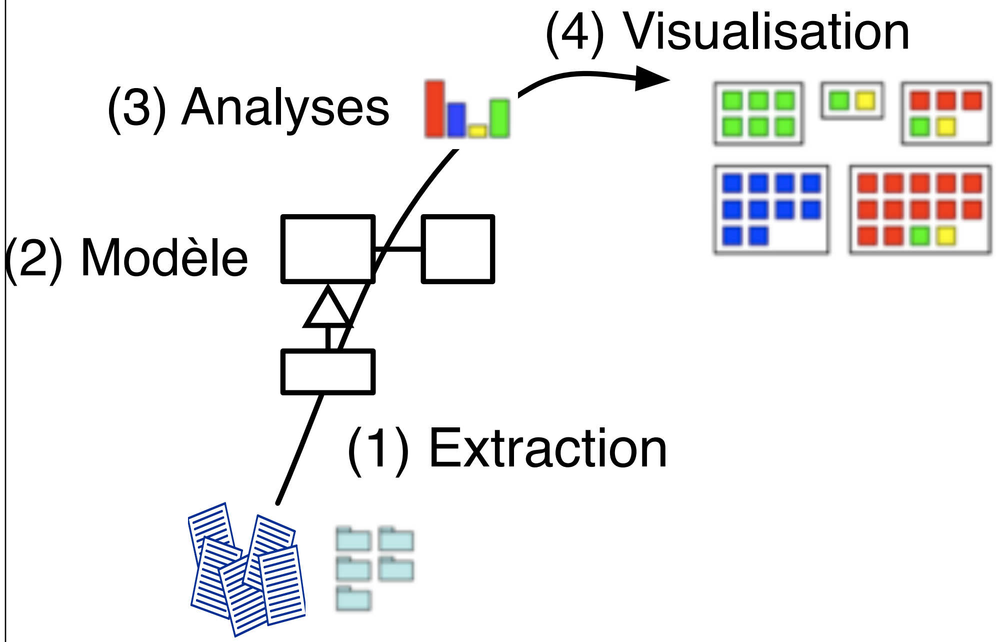
Reengineering Patterns
Version Analyses
[ICSM 05]
HISMO metamodel
[JSME 05]

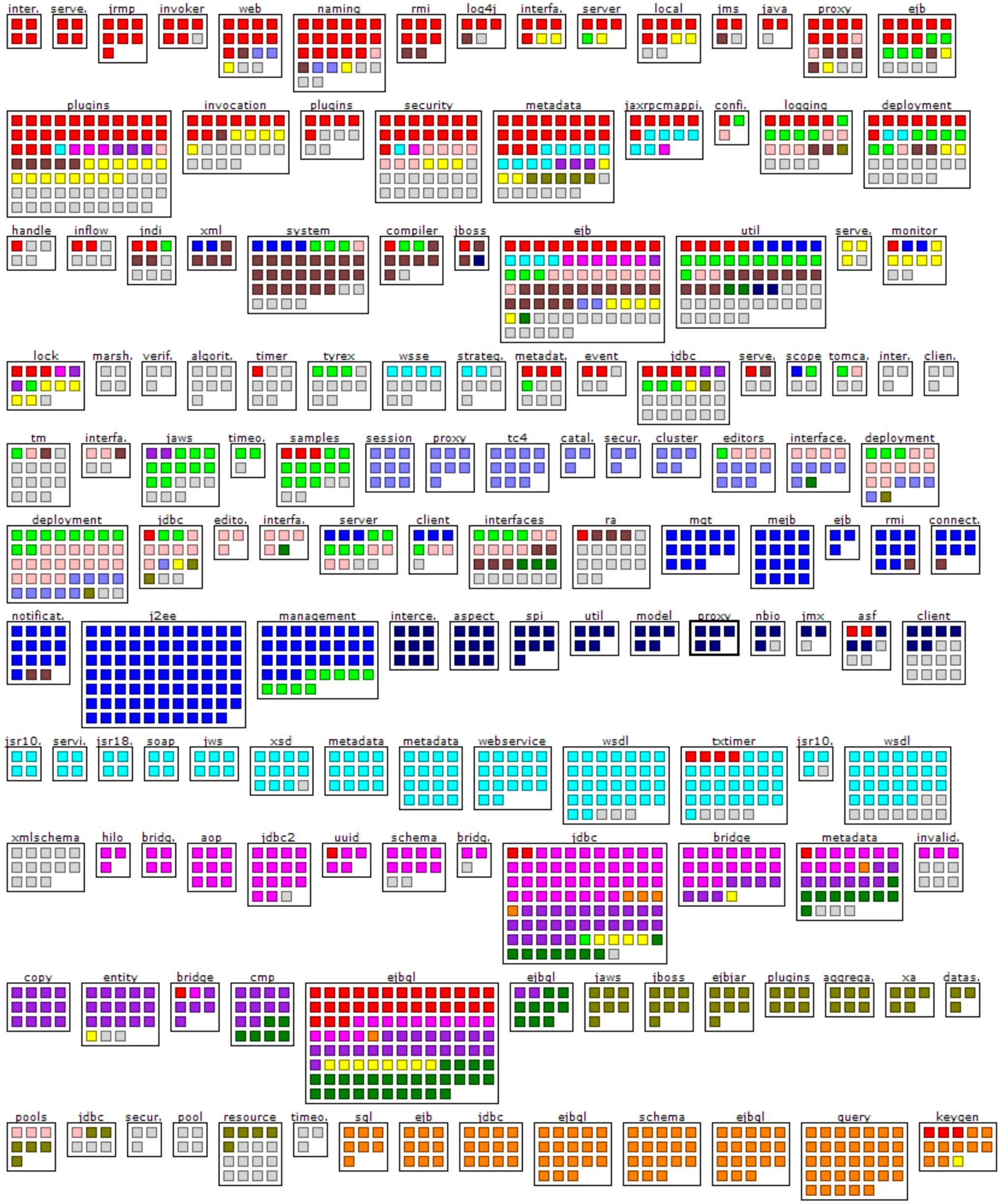
OBJECT-ORIENTED REENGINEERING PATTERNS
Serge Demeyer Stéphane Ducasse Oscar Nierstrasz

An example: who is responsible of what?

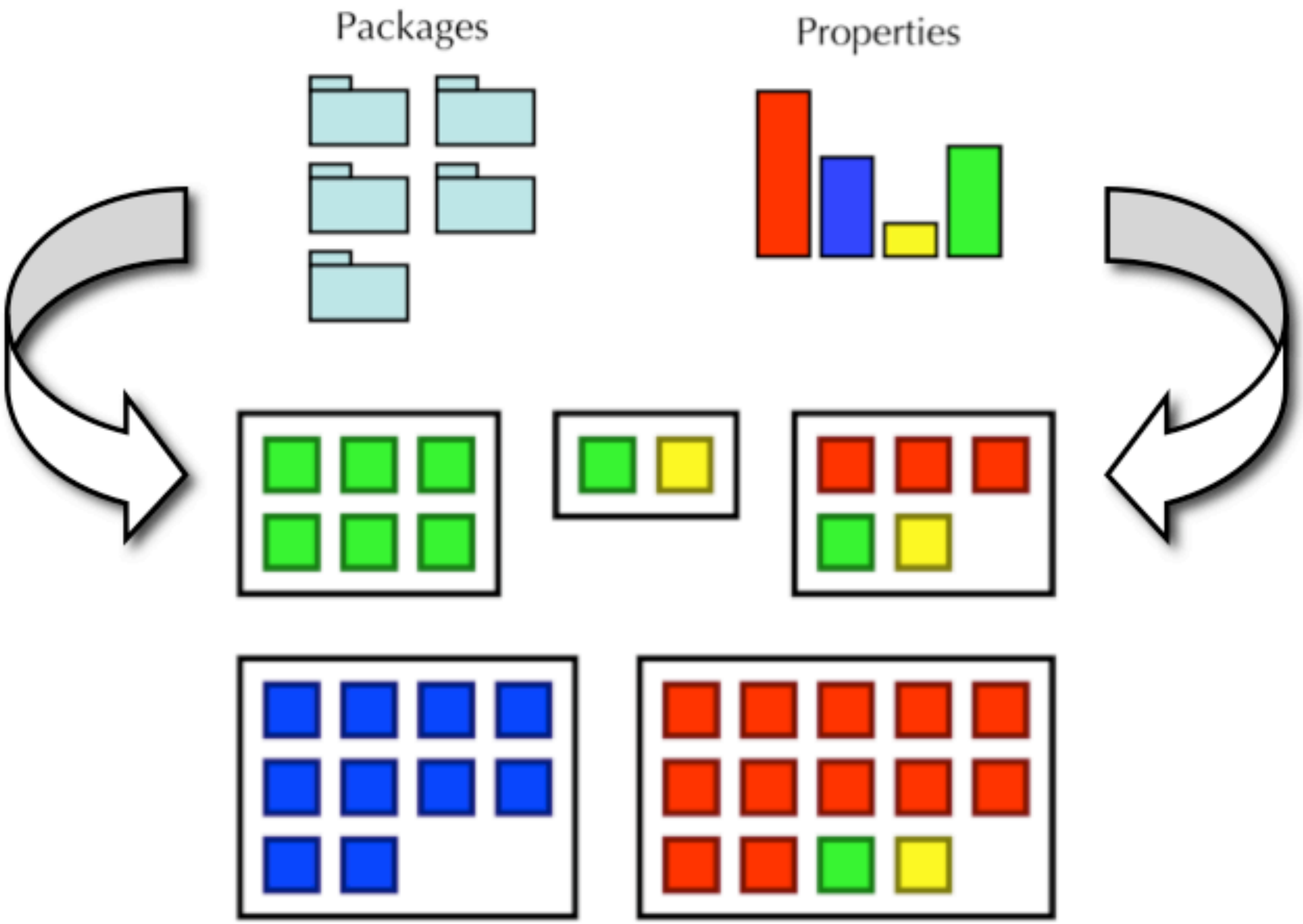


An example: who is responsible of what?

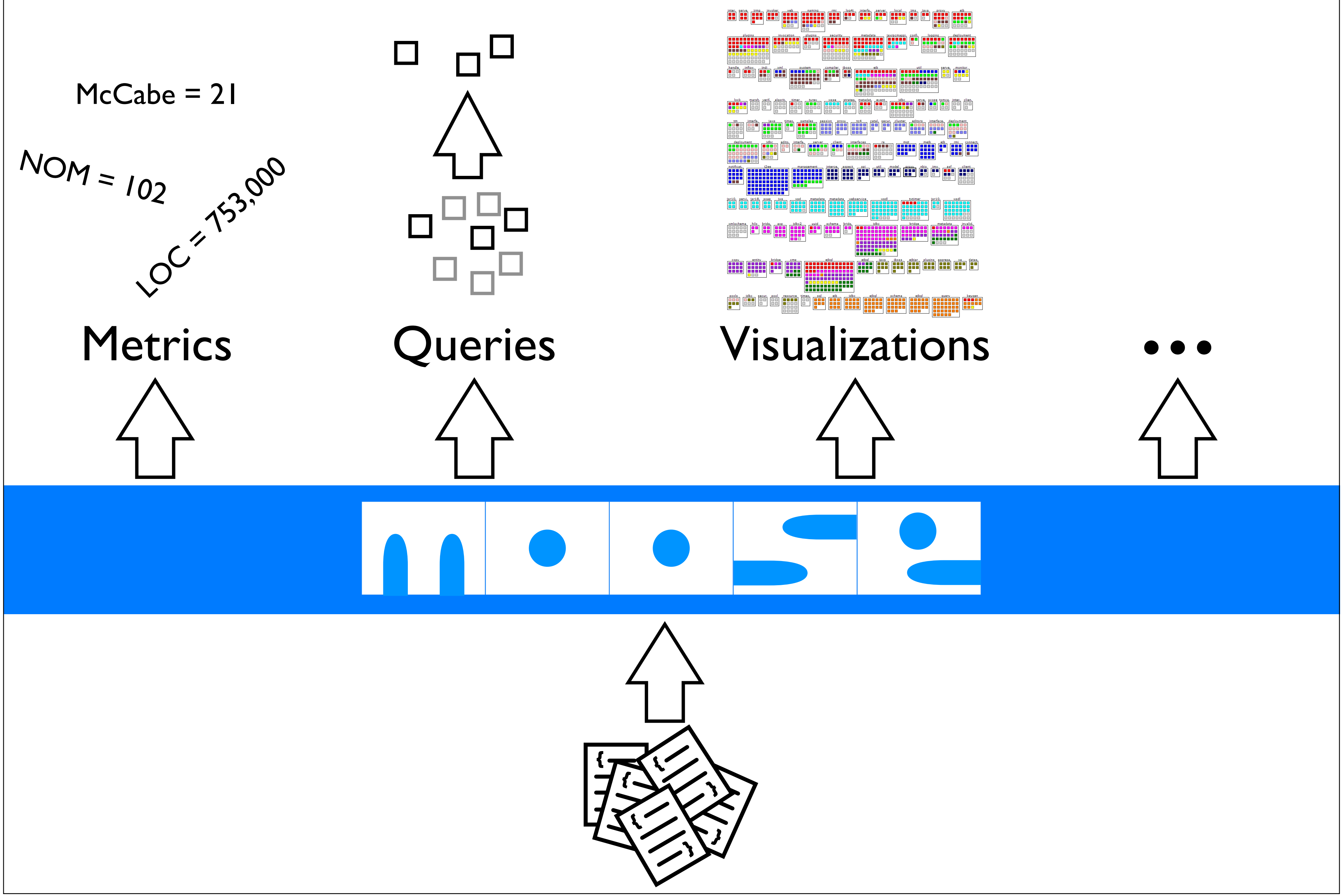




Distribution Map



Moose is a powerful environment



McCabe = 21

NOM = 102

LOC = 753,000

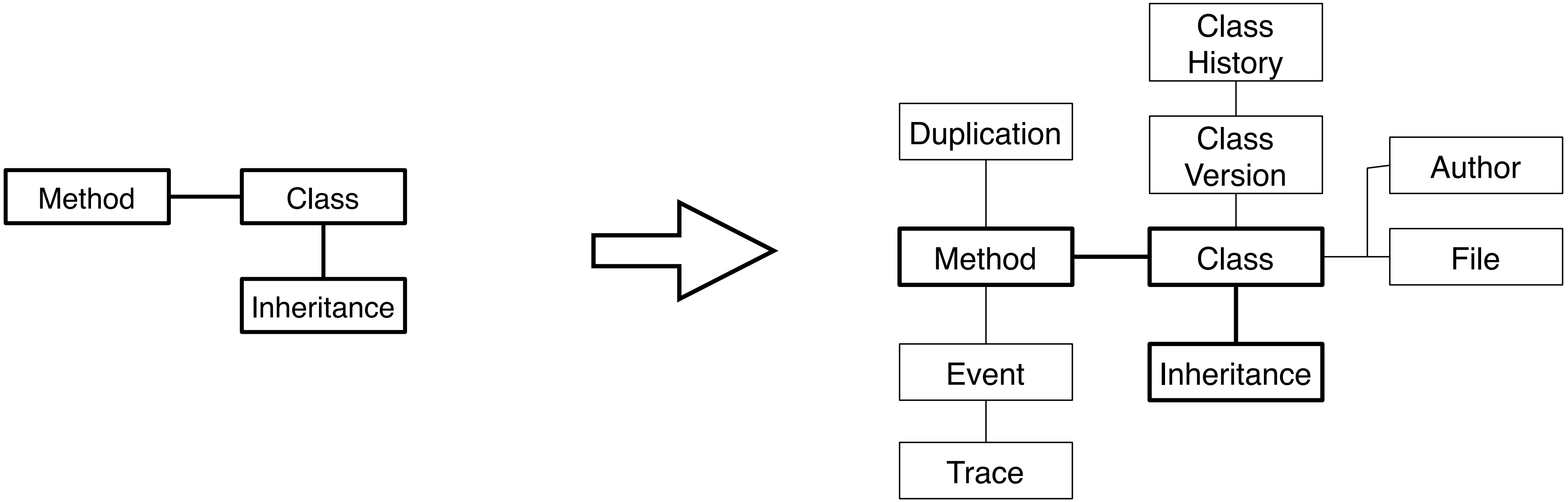
Metrics

Queries

Visualizations

...

Moose is designed to be extensible



open

meta-described

Moose has been validated on real life systems

Several large, industrial case studies (NDA)

- Harman-Becker
- Nokia
- Daimler
- Siemens

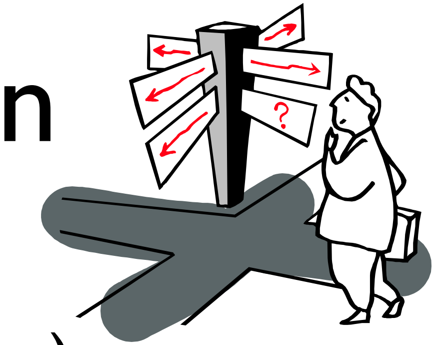
Different implementation languages (C++, Java, Smalltalk, Cobol)

- We use external C++ parsers

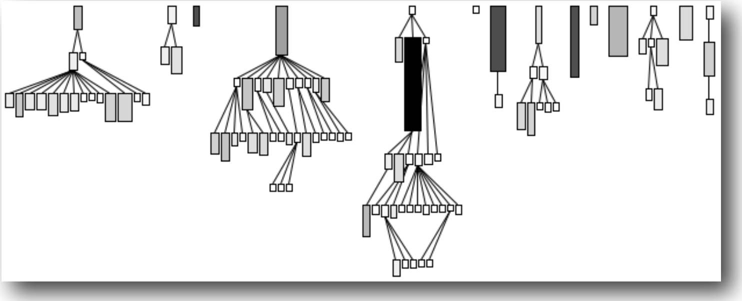
Different sizes

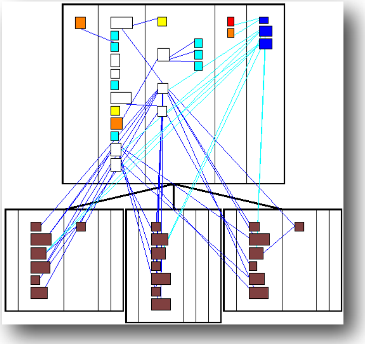
Moose is used in several research groups

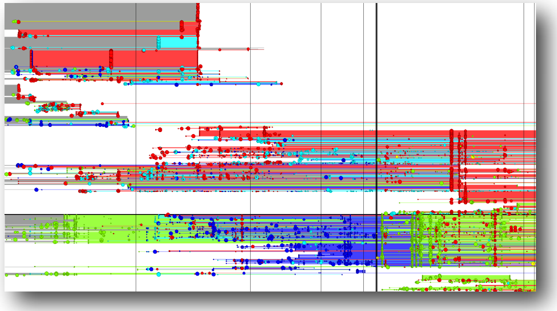
Visualization principles in 3 min

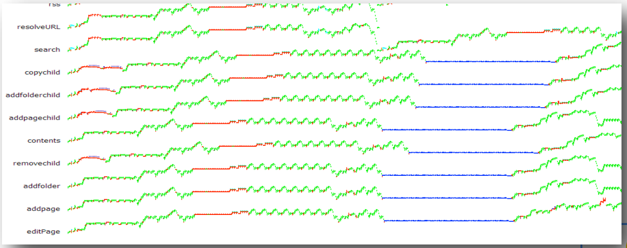


- Preattentive visualization (unconscious < 200ms)
- Gestalt principles (from 1912)
- 70% of our sensors are dedicated to vision









How many 5?

33321234665090000967666898778353367
78667609109198189717464330398 21768
3446786586088022116768768778 9762

How many 5?

33321234665090000096766689877835367
7866760910919818971746433039821768
34467865860880221167687687789762

Preattentive attributes

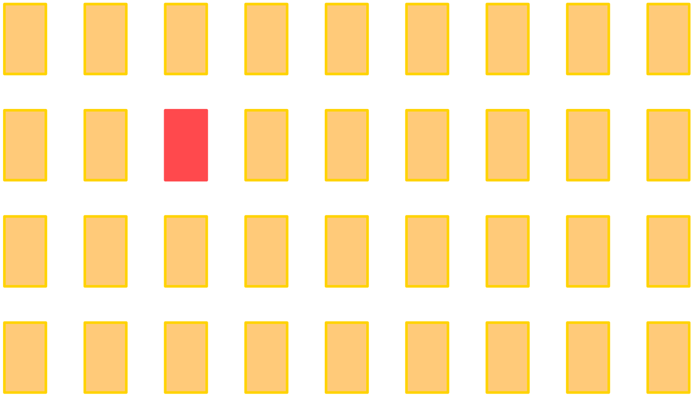
Color intensity

Form: orientation, line length, line width, size, shape, added marks, enclosure

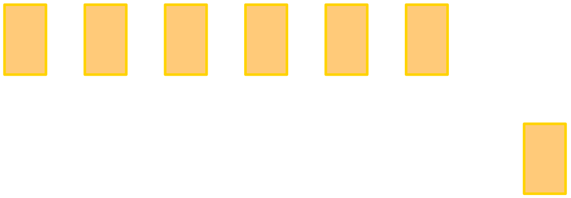
Spatial position (2D location)

Motion (flicker)

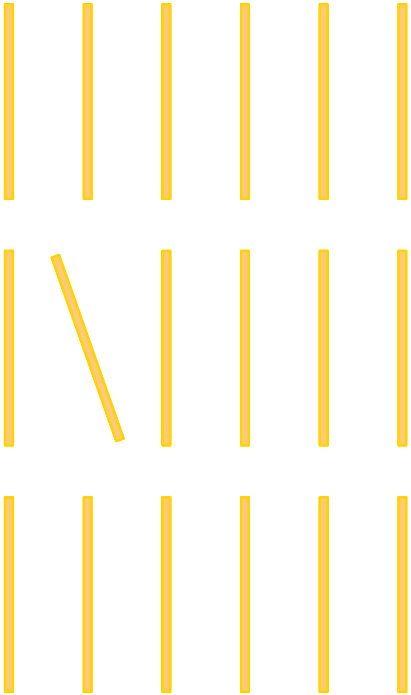
Color / intensity



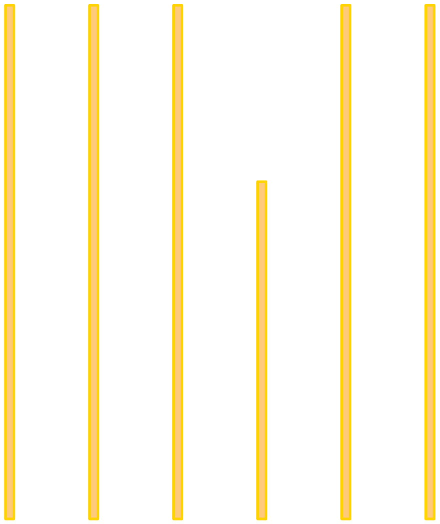
Position



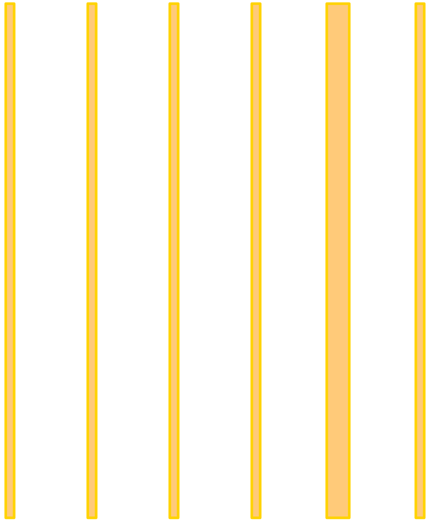
Form / Orientation



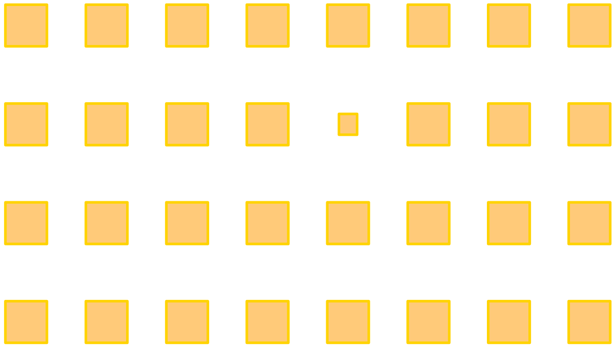
Form / Line length



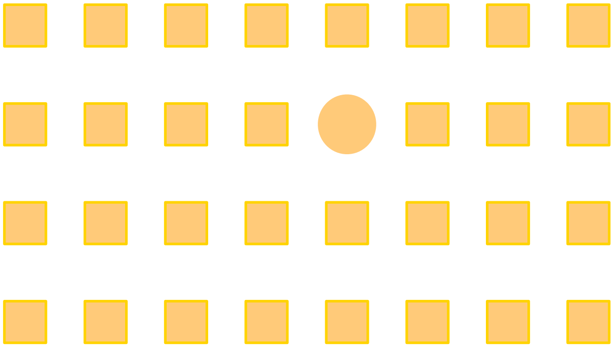
Form / Line width



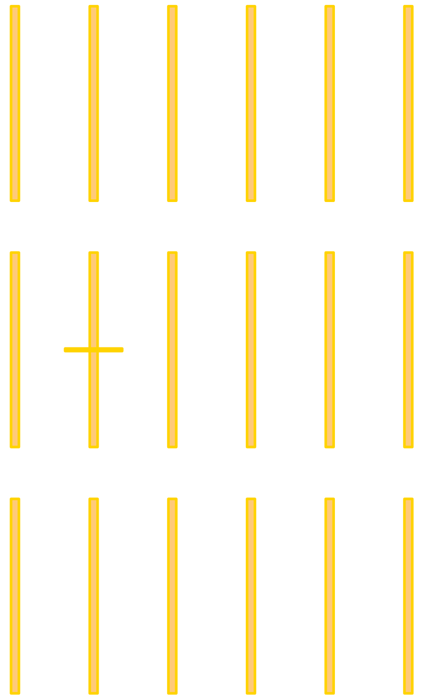
Form / Size



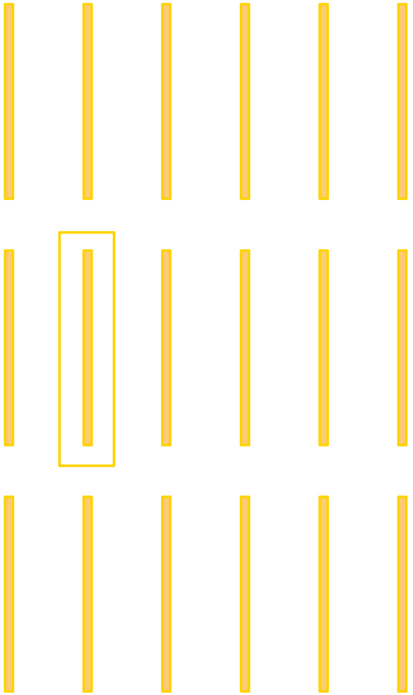
Form / Shapes



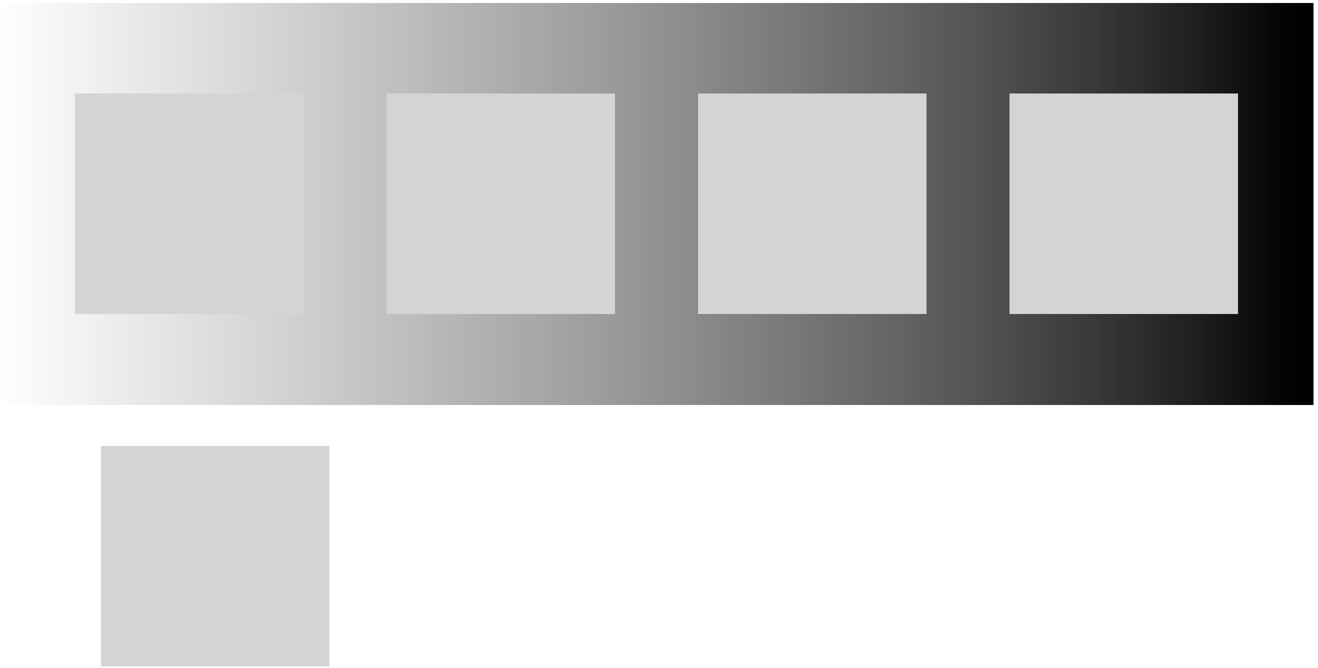
Form / Added marks



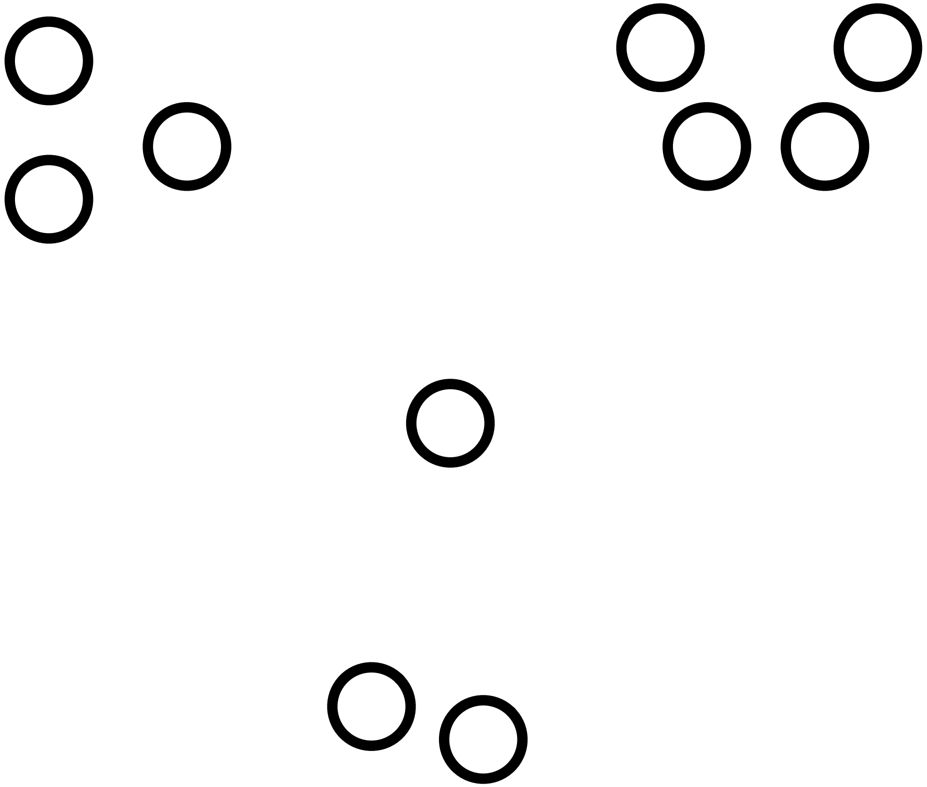
Form / Enclosure



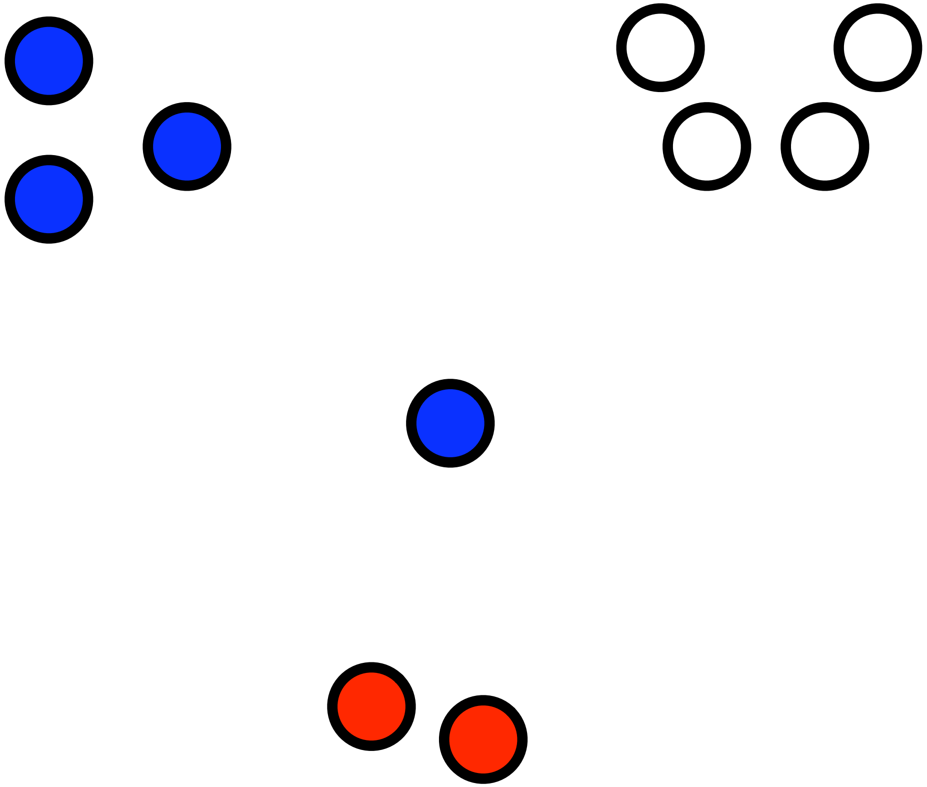
Context



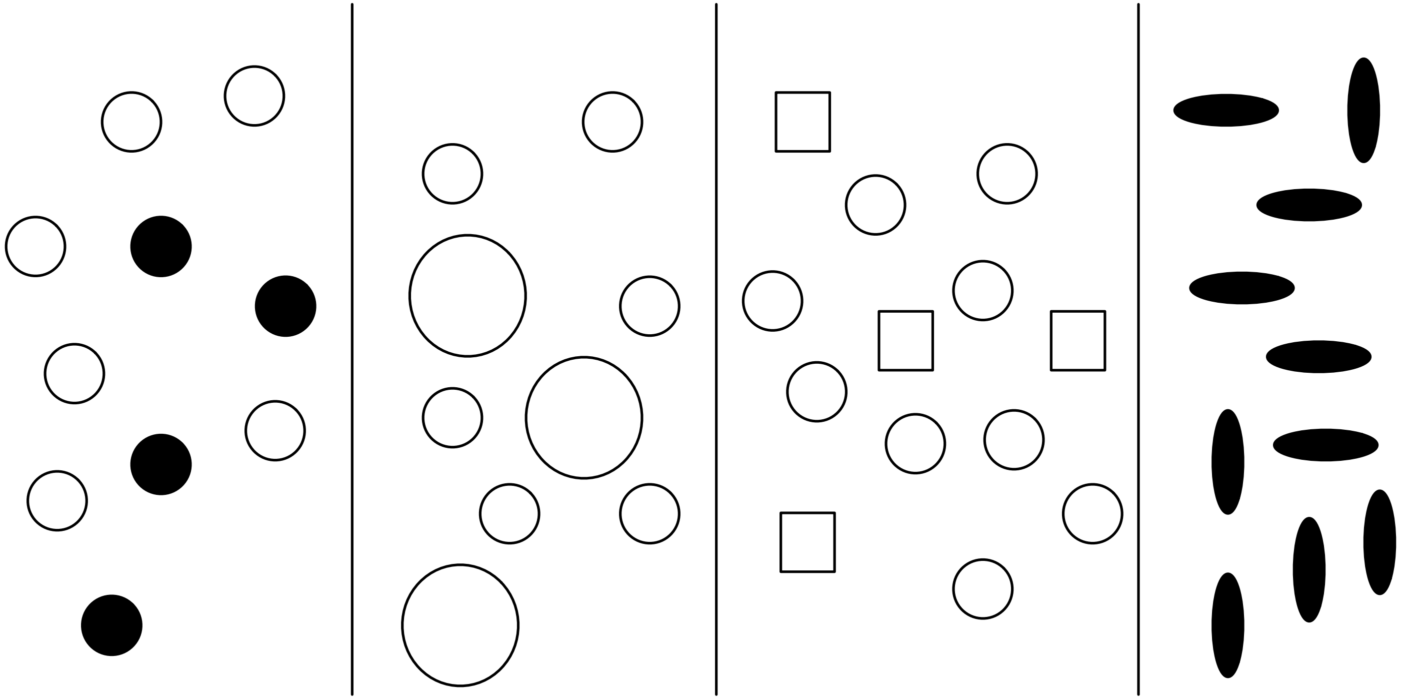
Principle of Proximity



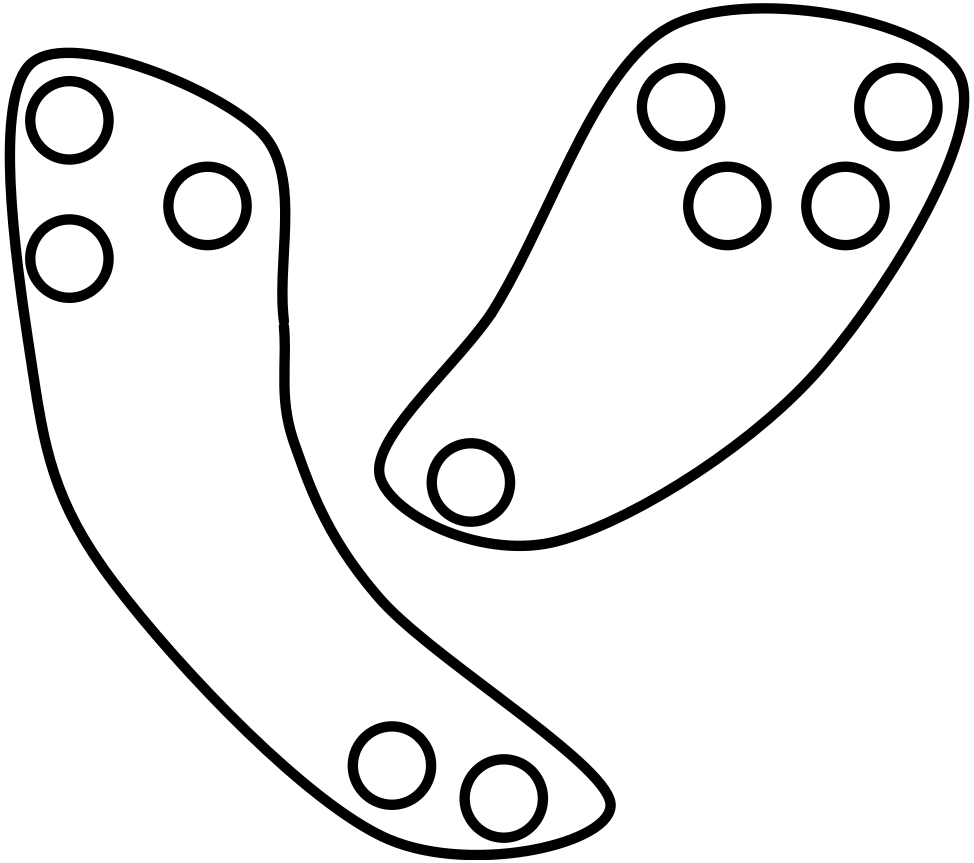
Principle of Similarity



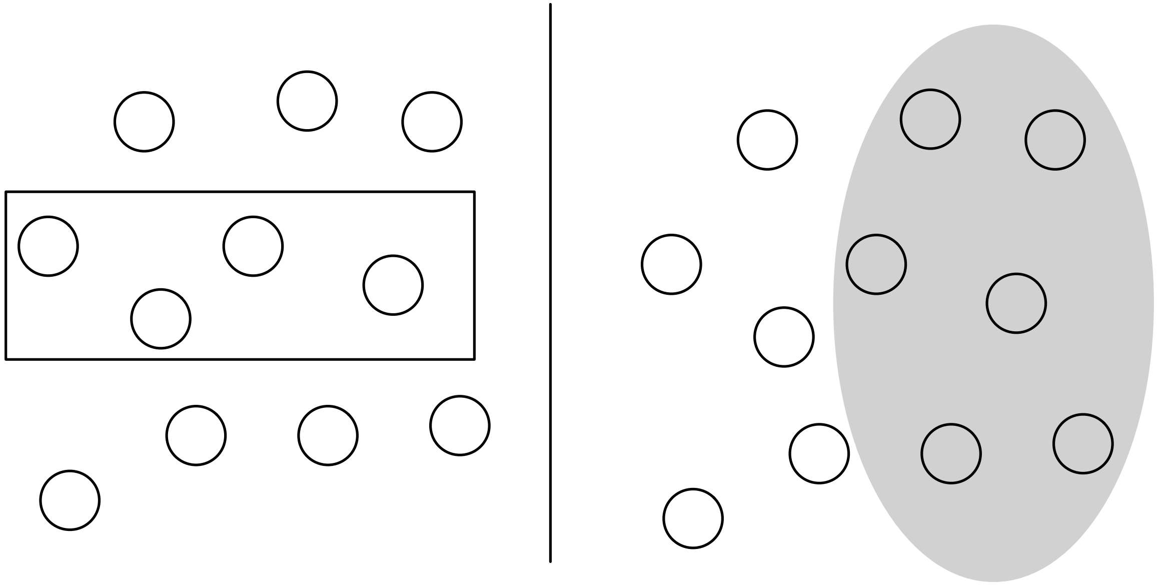
Principle of Similarity



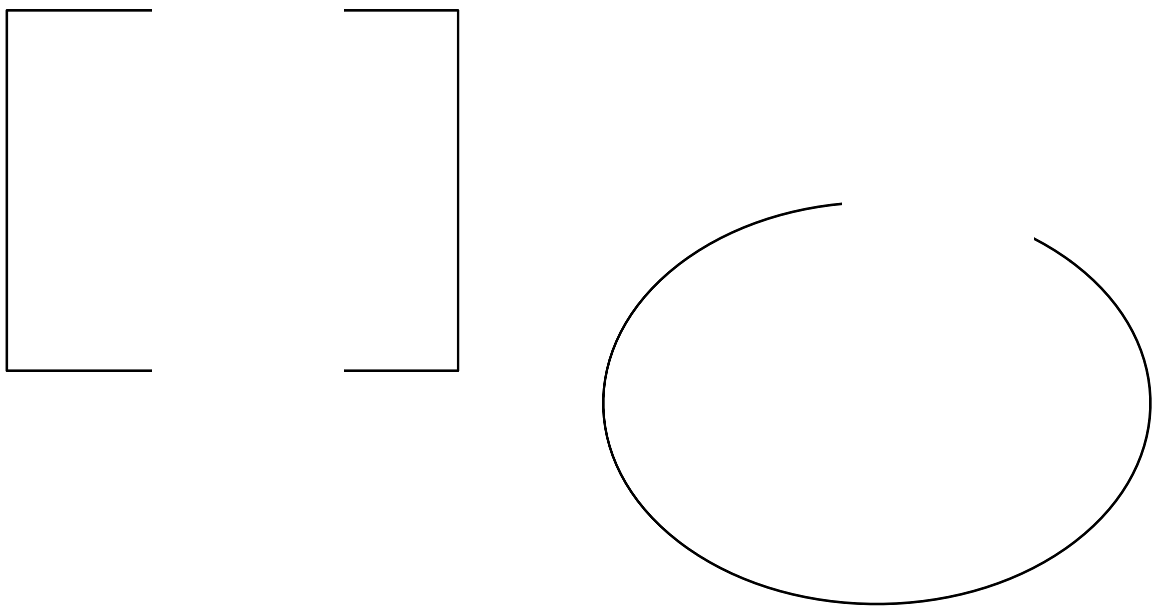
Principle of Enclosure



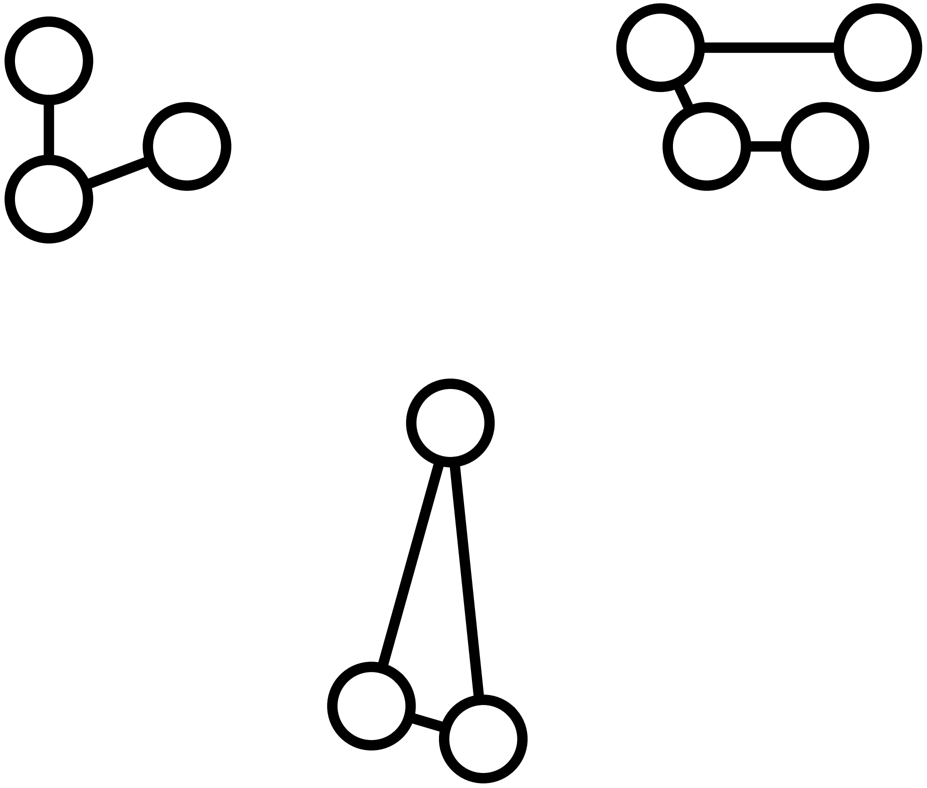
Principle of Enclosure



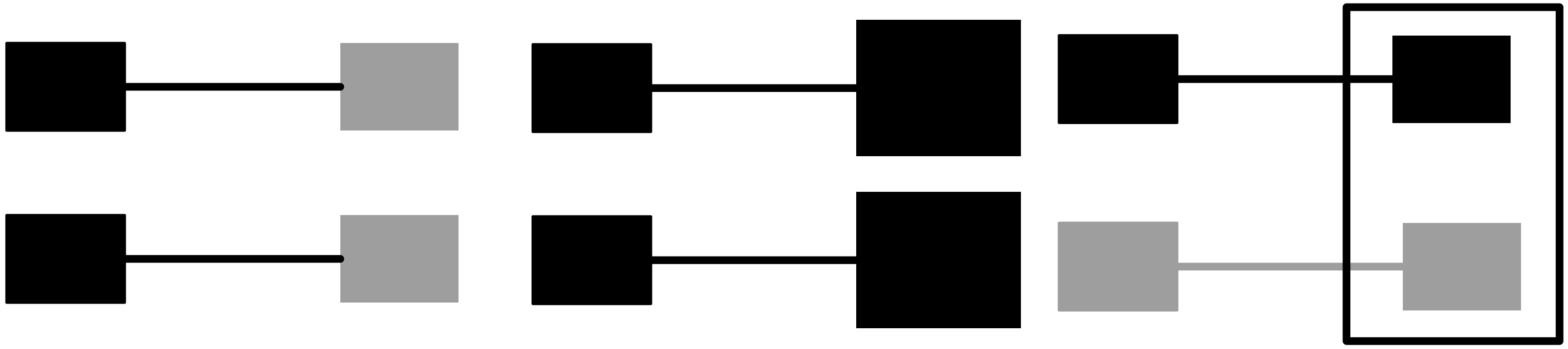
Principle of Closure



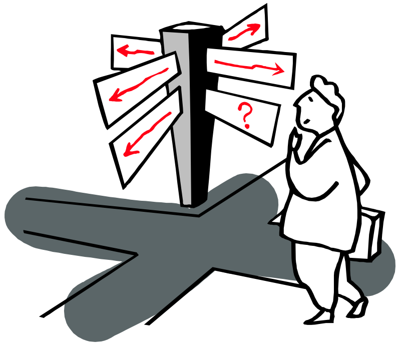
Principle of connectivity



Principle of connectivity



Roadmap



- Some software development facts
- Our approach
  - Supporting maintenance
  - Moose an open-platform
- Visual principles in 3 min
- ***Some visual examples***
- Conclusion

Challenges in Visualization

Screen size

Max 12 colors

Edge-crossing

Limited short-term memory (three to nine)

Extracting semantics out

Beauty cannot be a goal

Get some help from

- Gestalt principles
- pre-attentive visualization

Understanding large systems

Understanding code is difficult!

Systems are large

Code is abstract

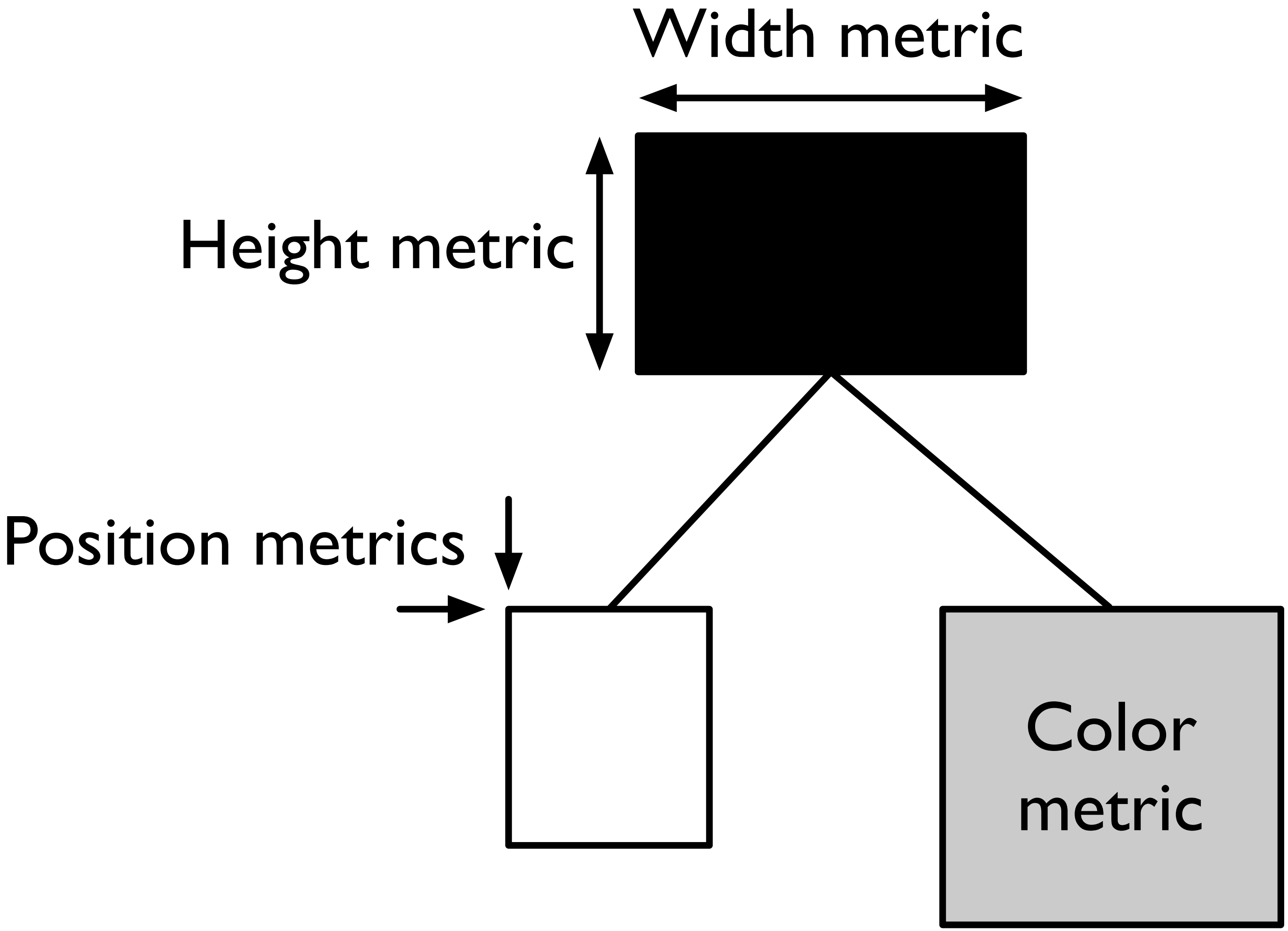
Should I really convinced you?

Some existing approaches

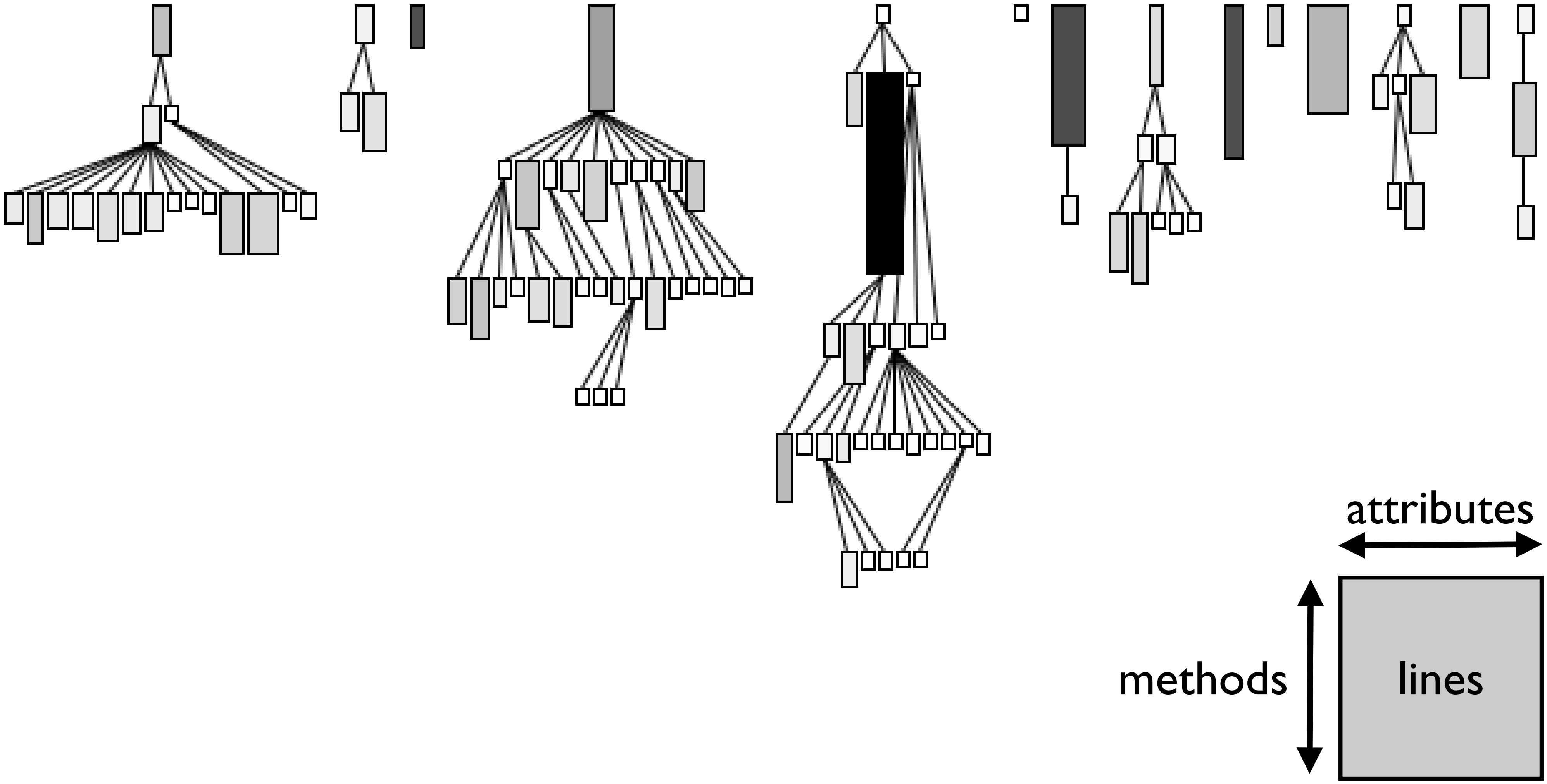
- Metrics: you often get ***meaningless*** results once ***combined***
- Visualization: often beautiful but ***with little*** meaning

Polymetric views show up to 5 metrics.

Lanza etal, 03

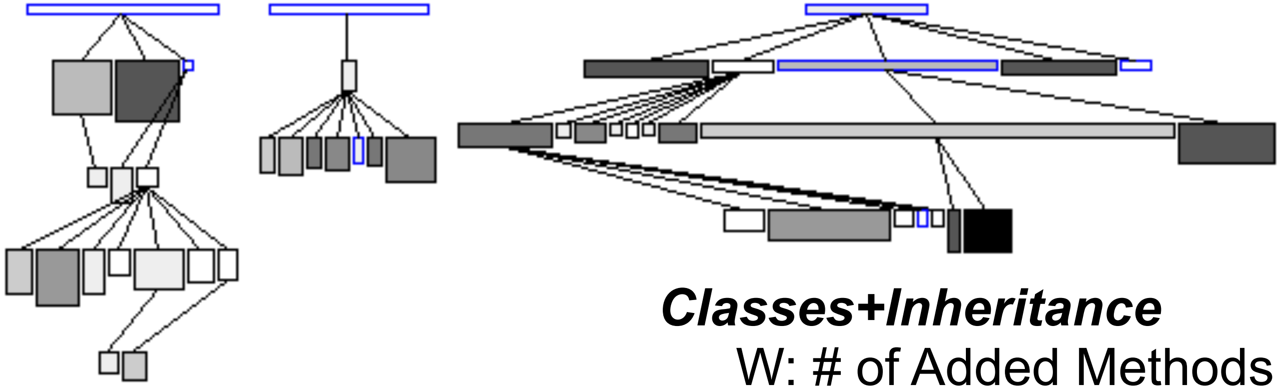


System Complexity shows class hierarchies.



Polymetric views condense information

To get a feel of the inheritance semantics: adding vs. reusing

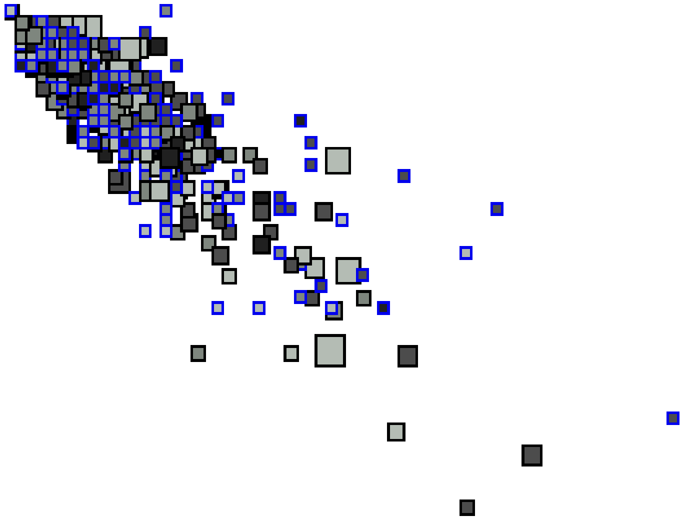


Classes+Inheritance

W: # of Added Methods

H: # of Overridden Metho

C: # of Method Extended



methods

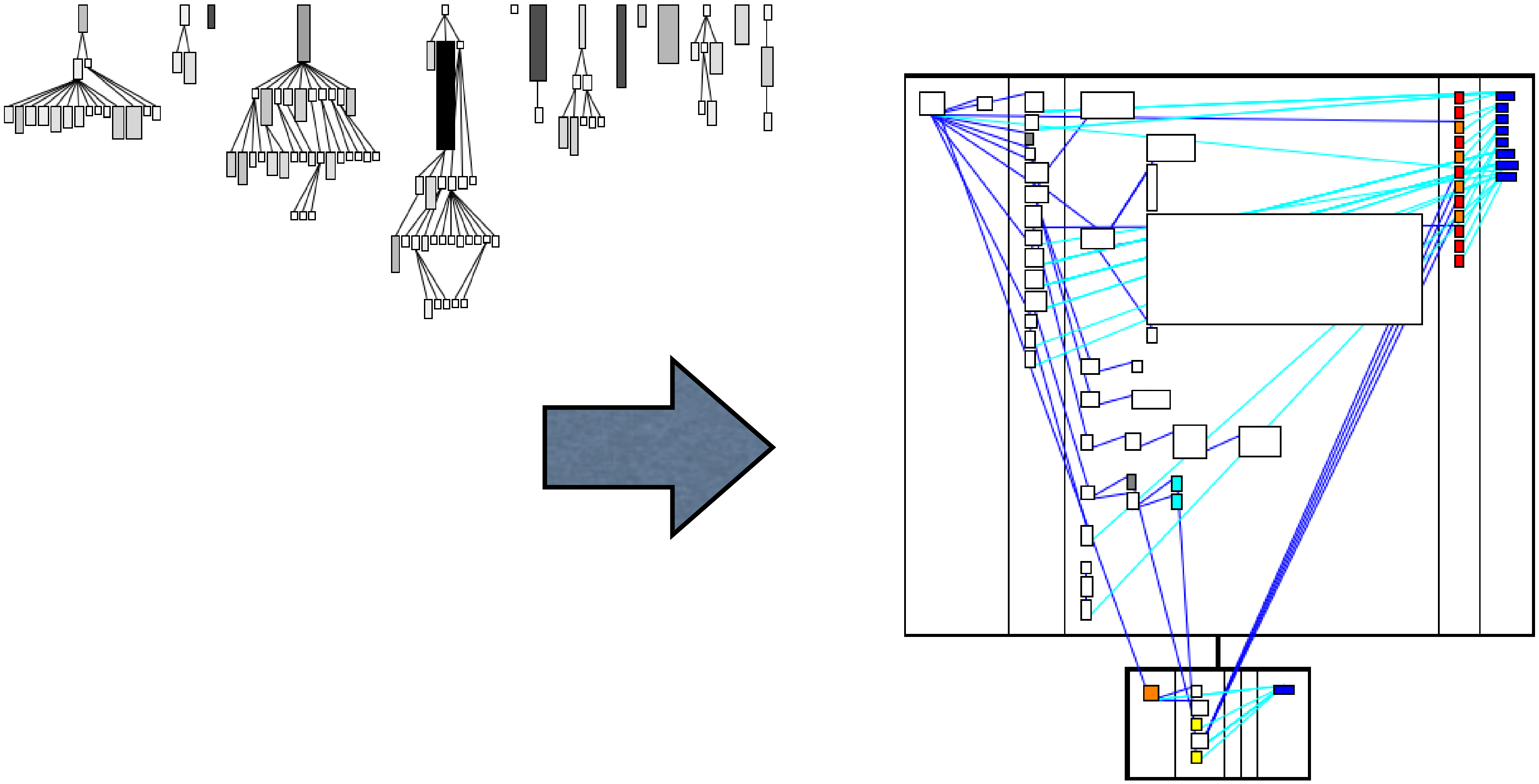
LOC

# statements

# parameters

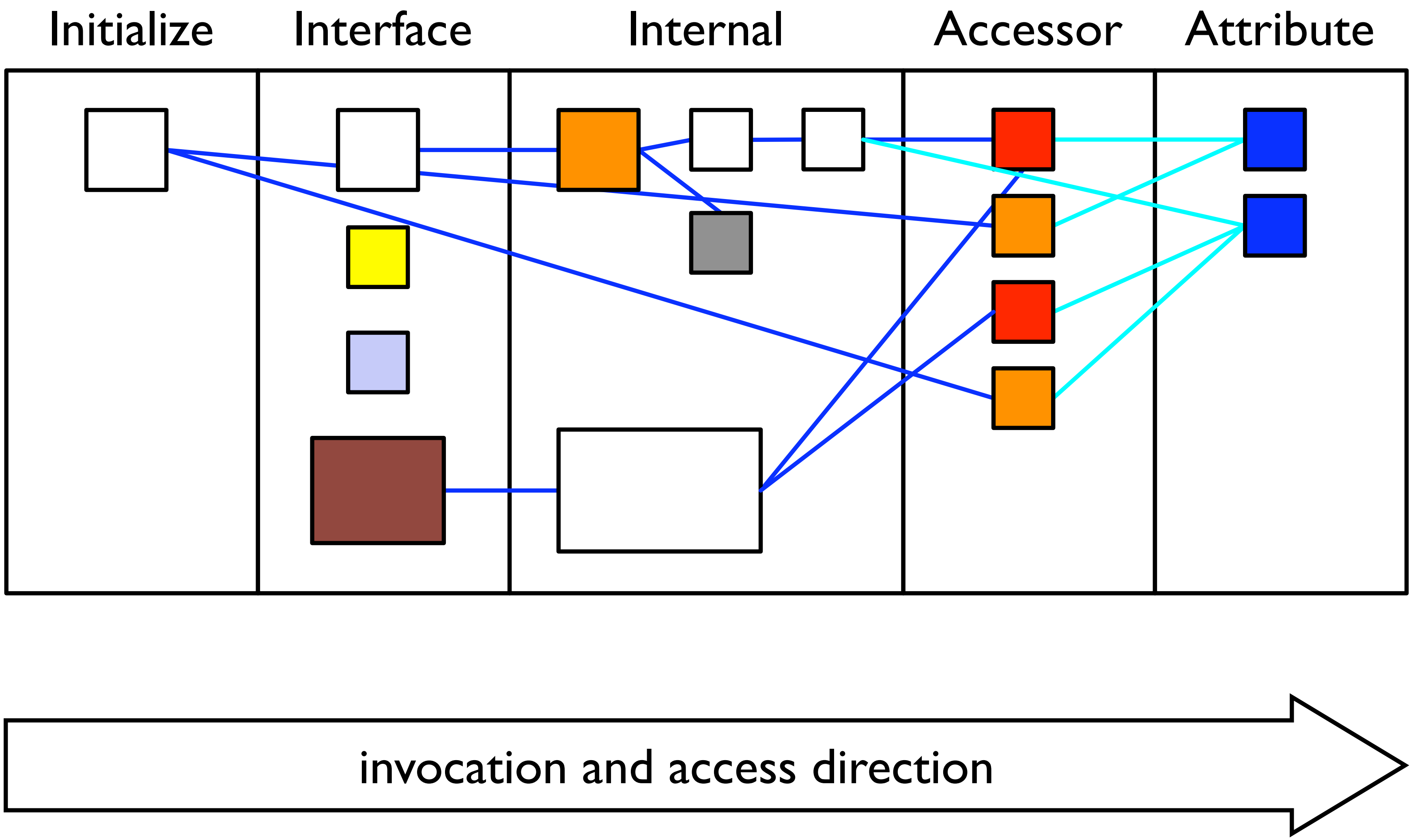
Understanding classes

Understanding even a class is difficult!

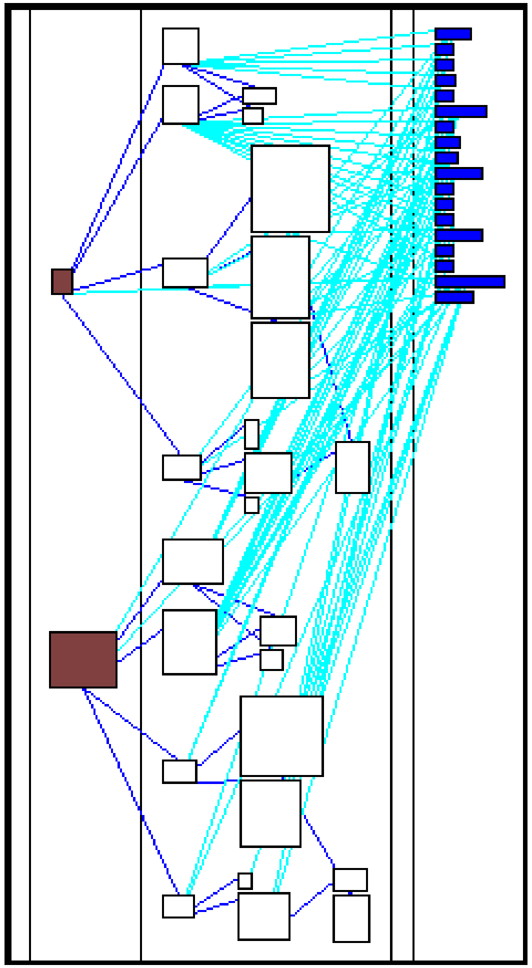


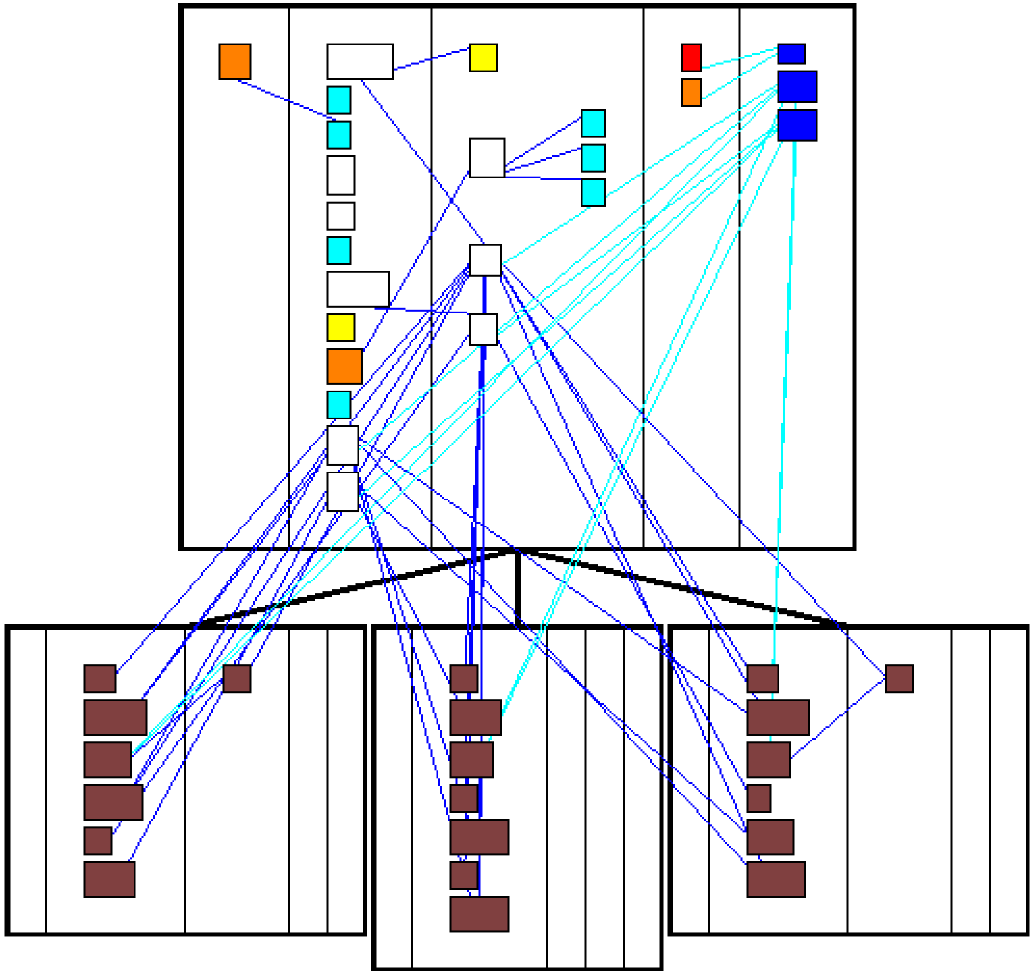
Class Blueprint shows class internals.

Ducasse, Lanza, 05



Class Blueprint shows class internals.



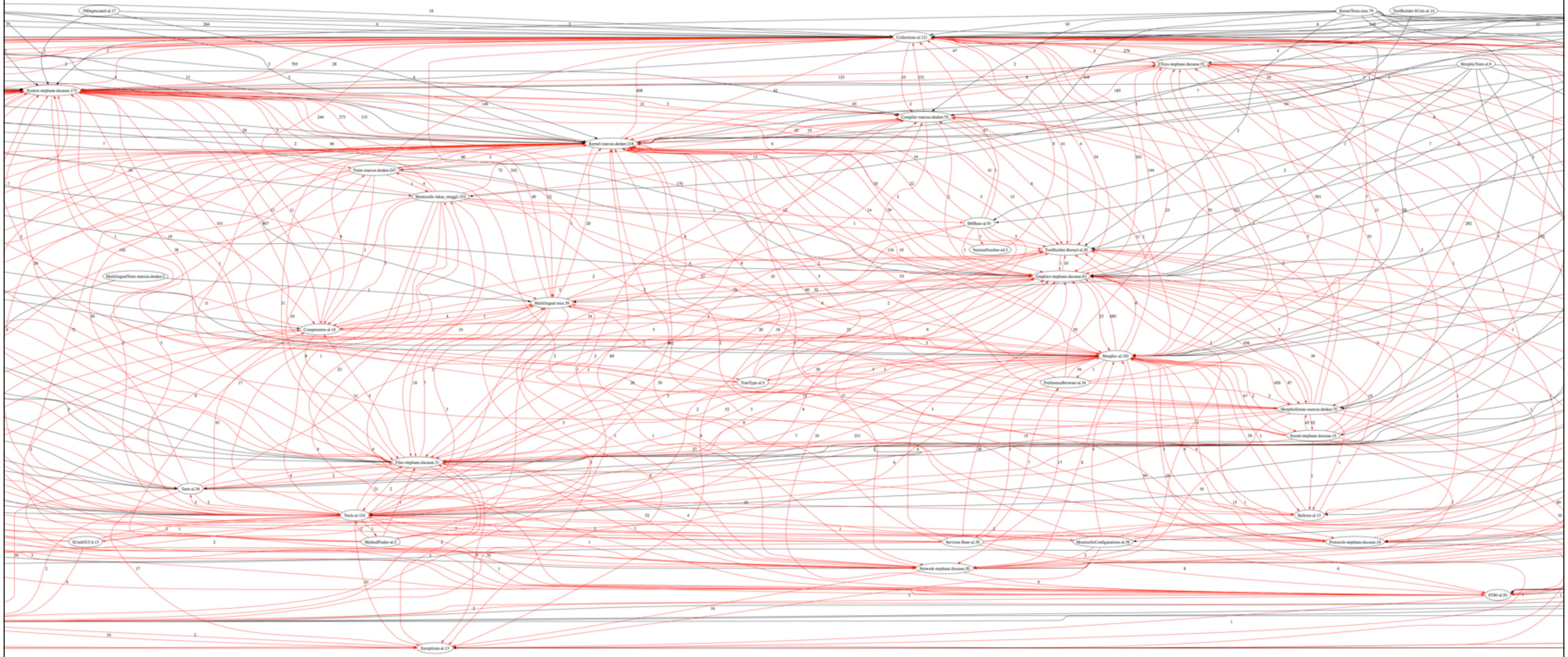


Cycles?

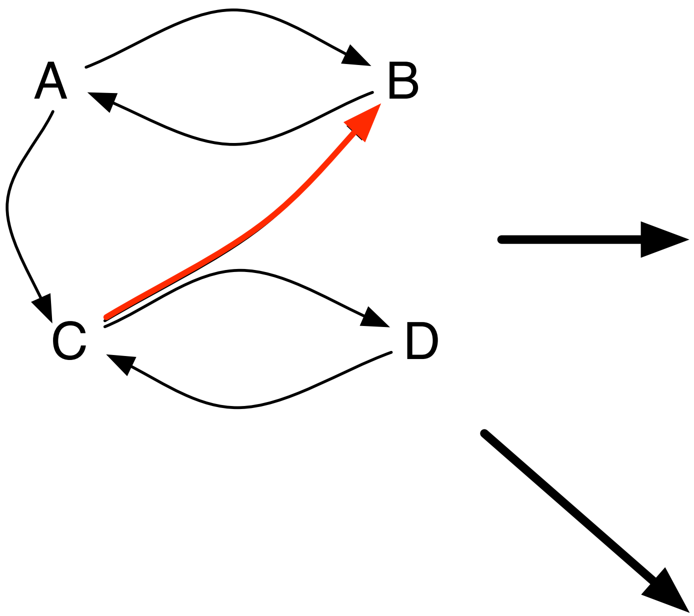
Identify
Understand
Fix

Enhancing Dependency Structural Matrix

Graph you said?



Building a DSM

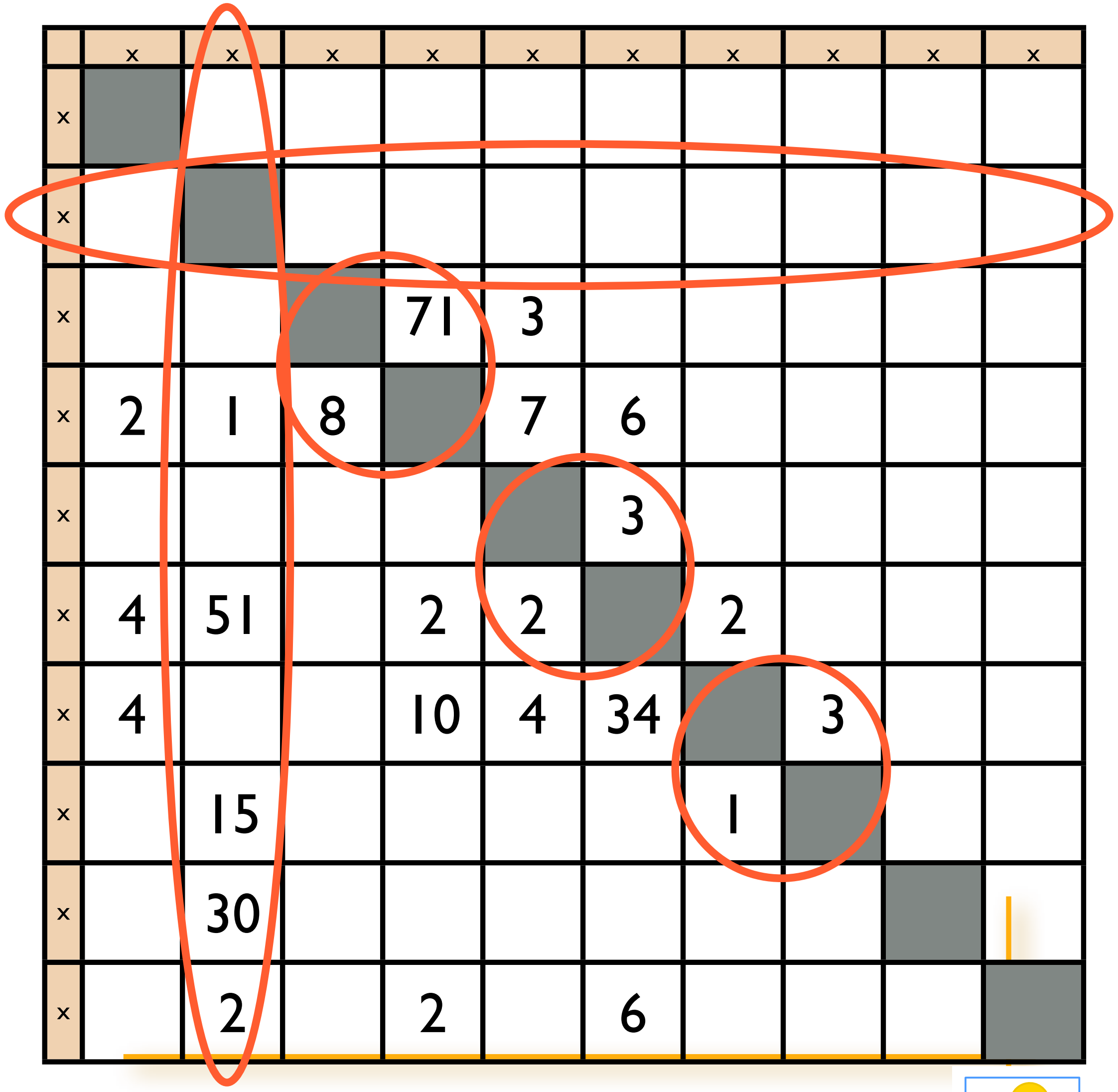


	A	B	C	D
A	■	X		
B	X	■	X	
C	X		■	X
D			X	■

	A	B	C	D
A	0	1	0	0
B	0	0	1	0
C	1	0	0	1
D	0	0	1	0

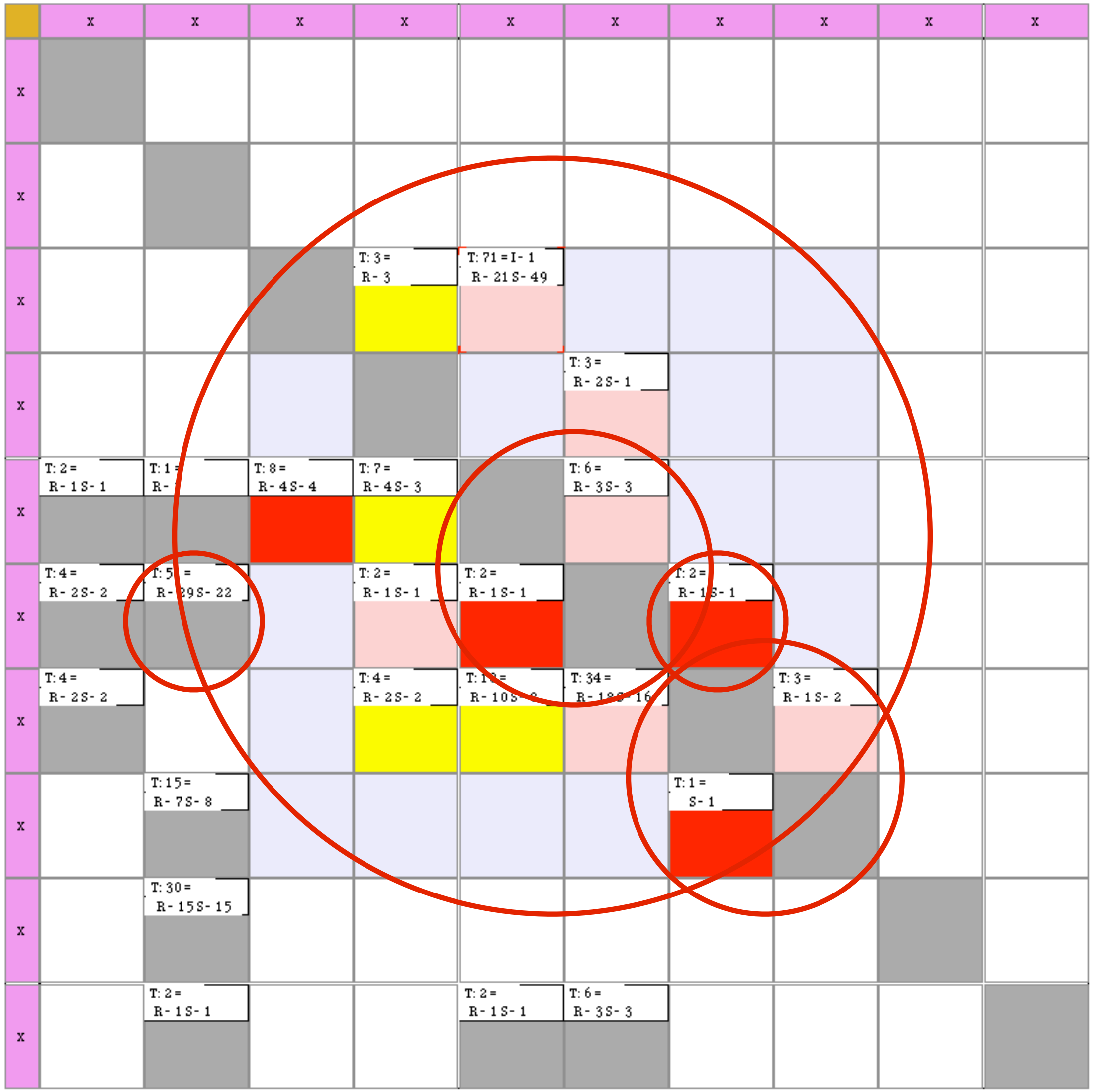
7 Packages visualization

1 cell = 1 dependency
1 column = used packages
1 line = using packages



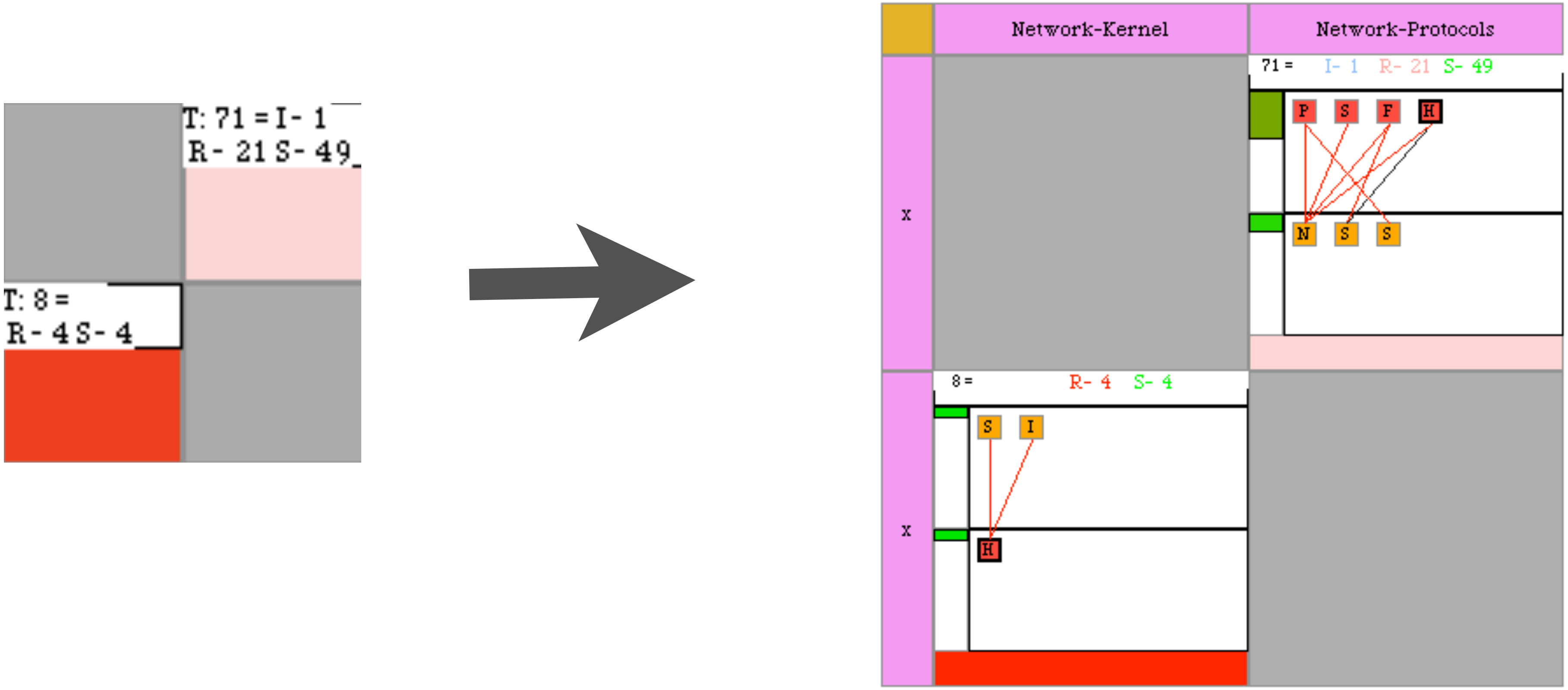
	x	x	x	x	x	x	x	x	x	x
x	▨									
x		▨								
x			▨	71	3					
x	2	1	8	▨	7	6				
x					▨	3				
x	4	51		2	2	▨	2			
x	4			10	4	34	▨	3		
x		15					1	▨		
x		30							▨	
x		2		2		6				▨

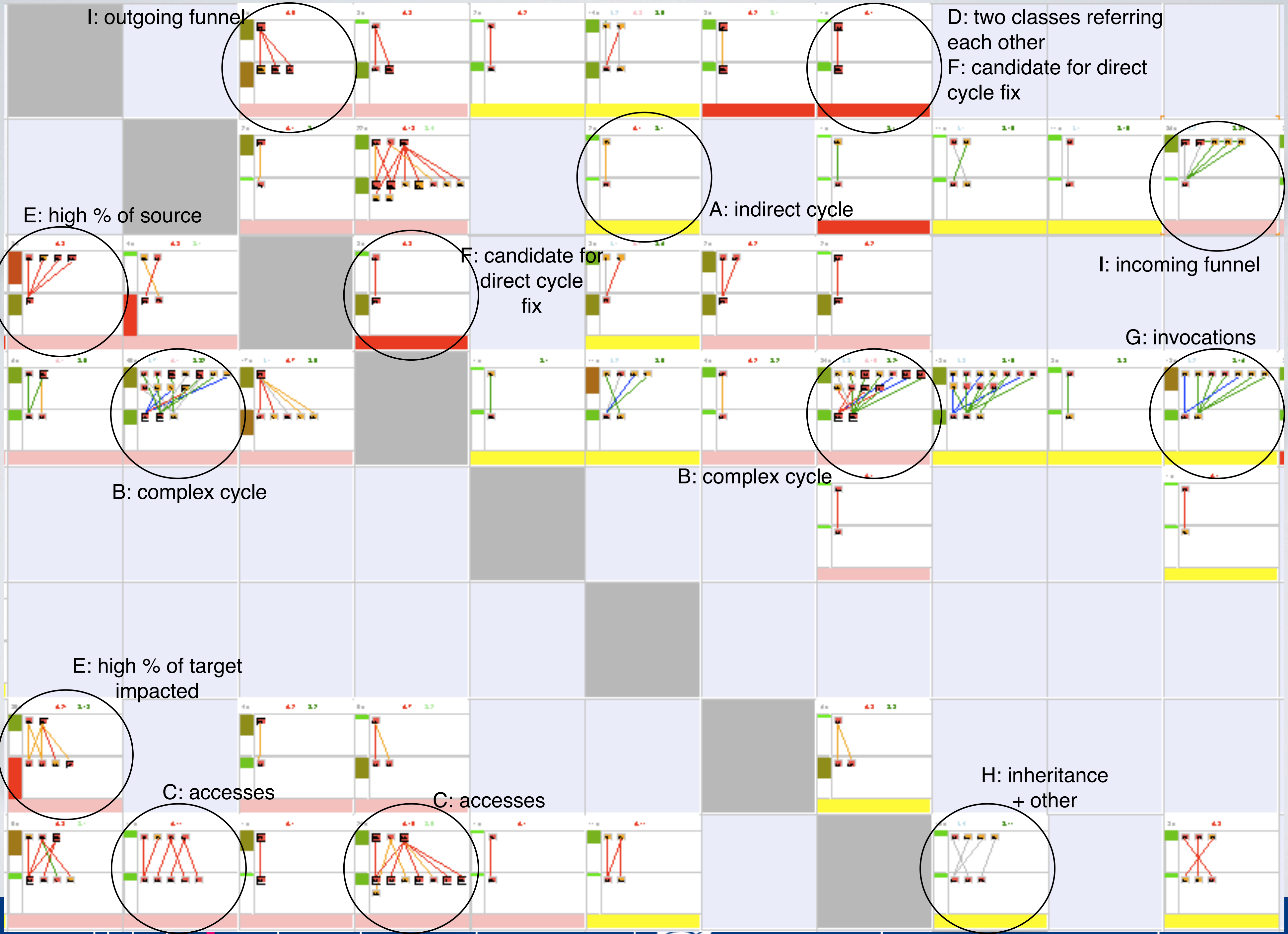
Identify cycles



	x	x	x	x	x	x	x	x	x	x
x										
x										
x				T: 3= R- 3	T: 71 =I- 1 R- 21 S- 49					
x						T: 3= R- 2S- 1				
x	T: 2= R- 1S- 1	T: 1= R- 1	T: 8= R- 4S- 4	T: 7= R- 4S- 3		T: 6= R- 3S- 3				
x	T: 4= R- 2S- 2	T: 51 = R- 29S- 22		T: 2= R- 1S- 1	T: 2= R- 1S- 1		T: 2= R- 1S- 1			
x	T: 4= R- 2S- 2			T: 4= R- 2S- 2	T: 18= R- 10S- 8	T: 34= R- 18S- 16		T: 3= R- 1S- 2		
x		T: 15= R- 7S- 8					T: 1= S- 1			
x		T: 30= R- 15S- 15								
x		T: 2= R- 1S- 1			T: 2= R- 1S- 1	T: 6= R- 3S- 3				

Causes and distribution





I: outgoing funnel

D: two classes referring each other

F: candidate for direct cycle fix

A: indirect cycle

E: high % of source

F: candidate for direct cycle fix

I: incoming funnel

G: invocations

B: complex cycle

B: complex cycle

E: high % of target impacted

C: accesses

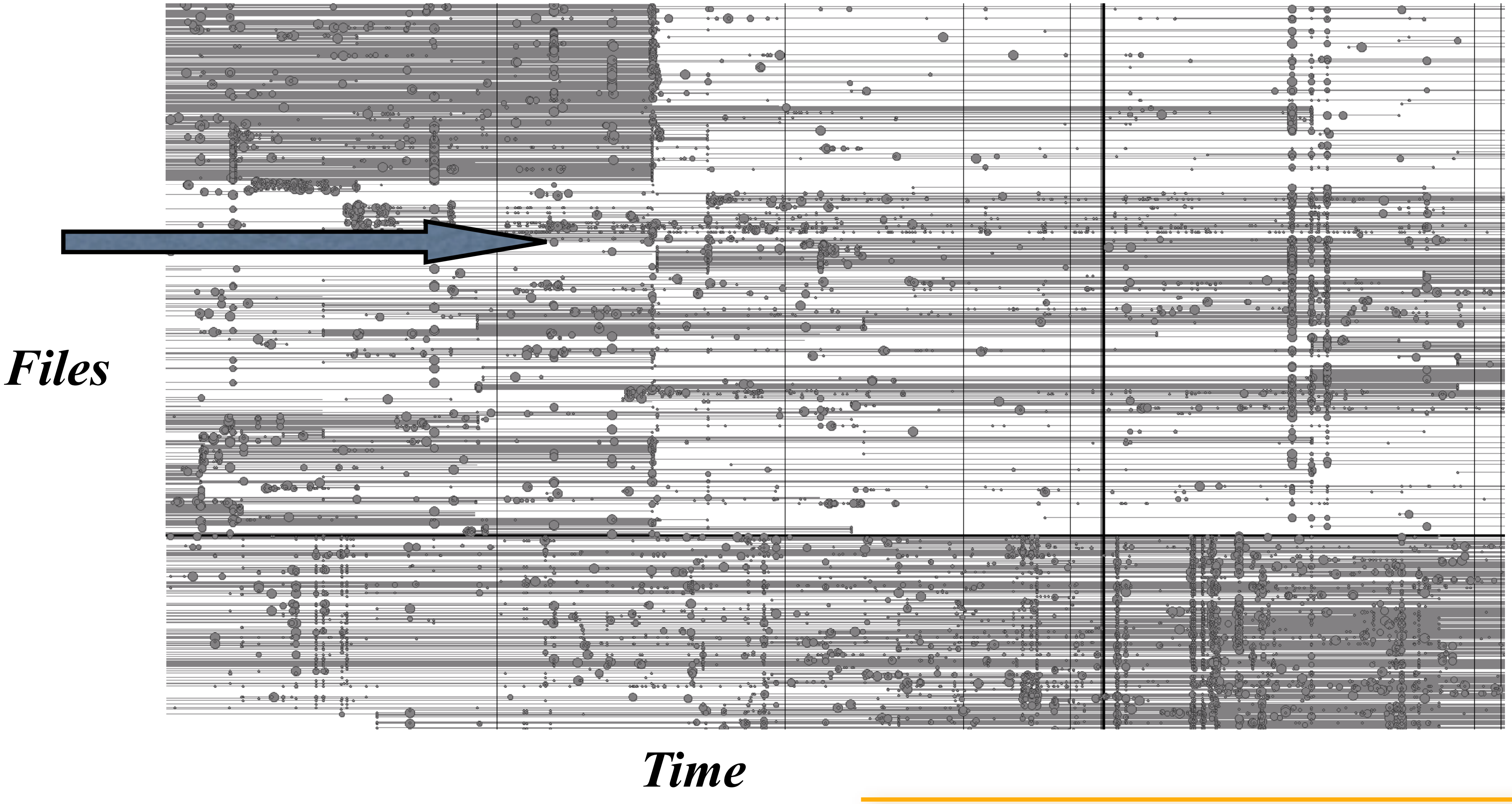
C: accesses

H: inheritance + other

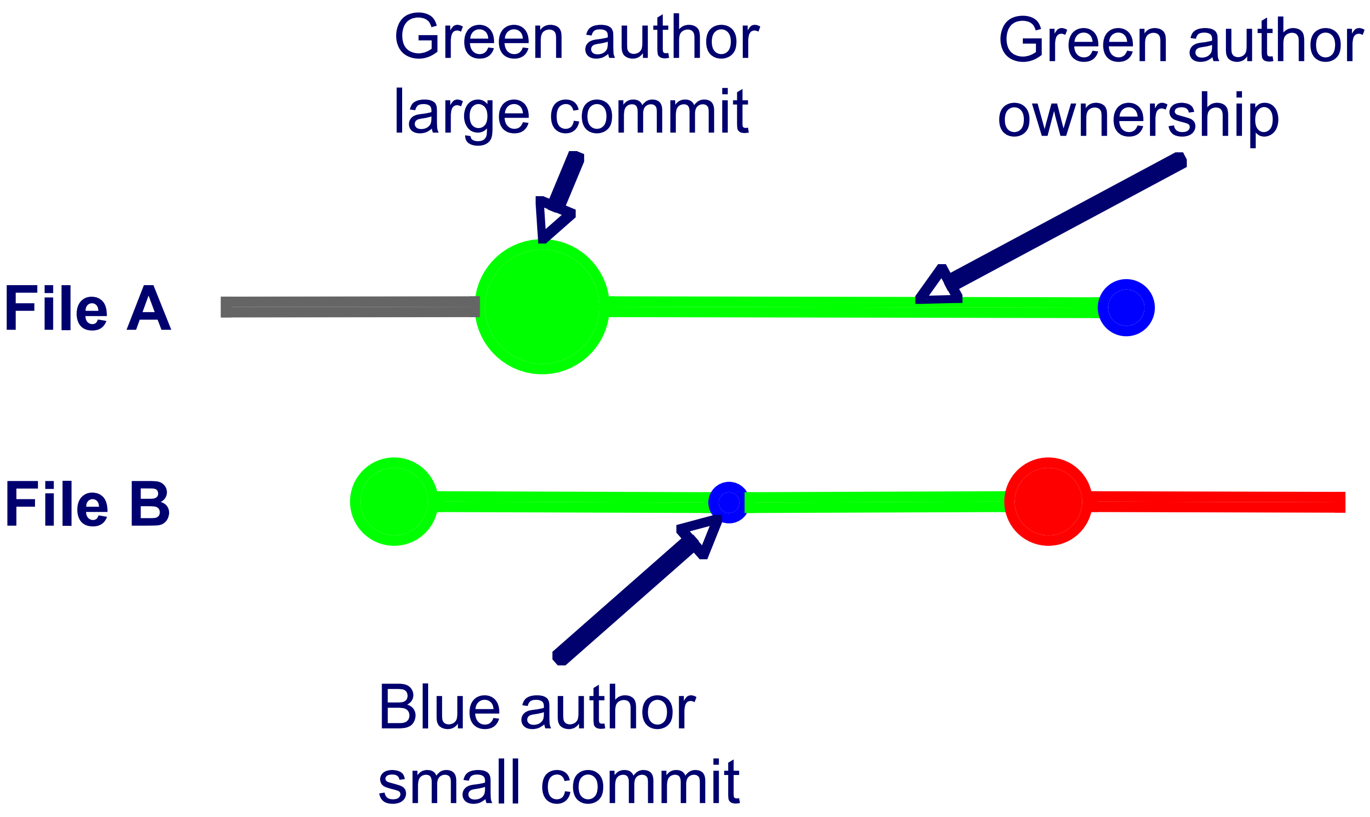
How developers develop?

- More efficient to put people working together in the same office?
- How can we optimize software development?

Who did that?



Line colors show which author owned which files in which period



Green author large commit

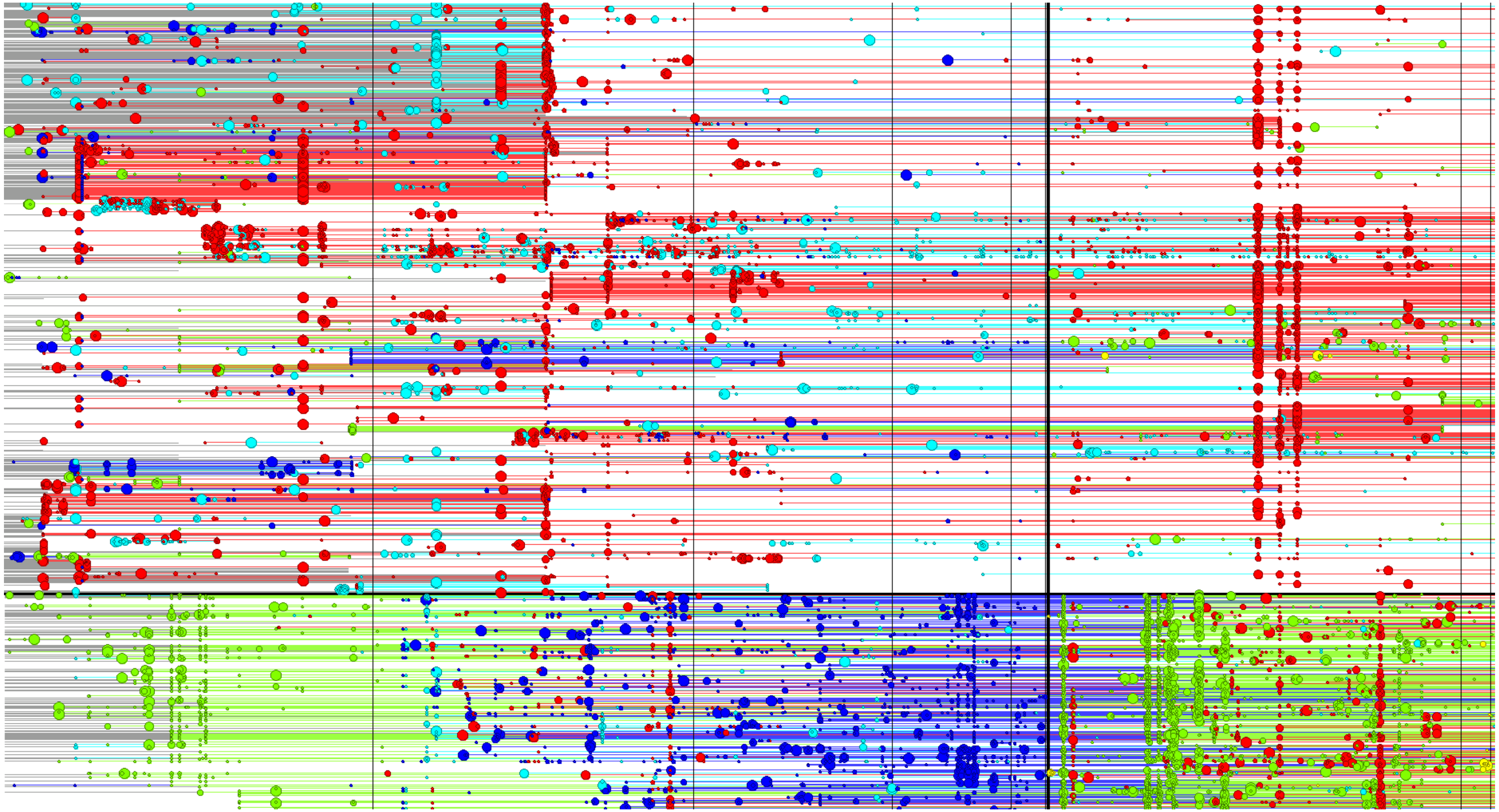
Green author ownership

File A

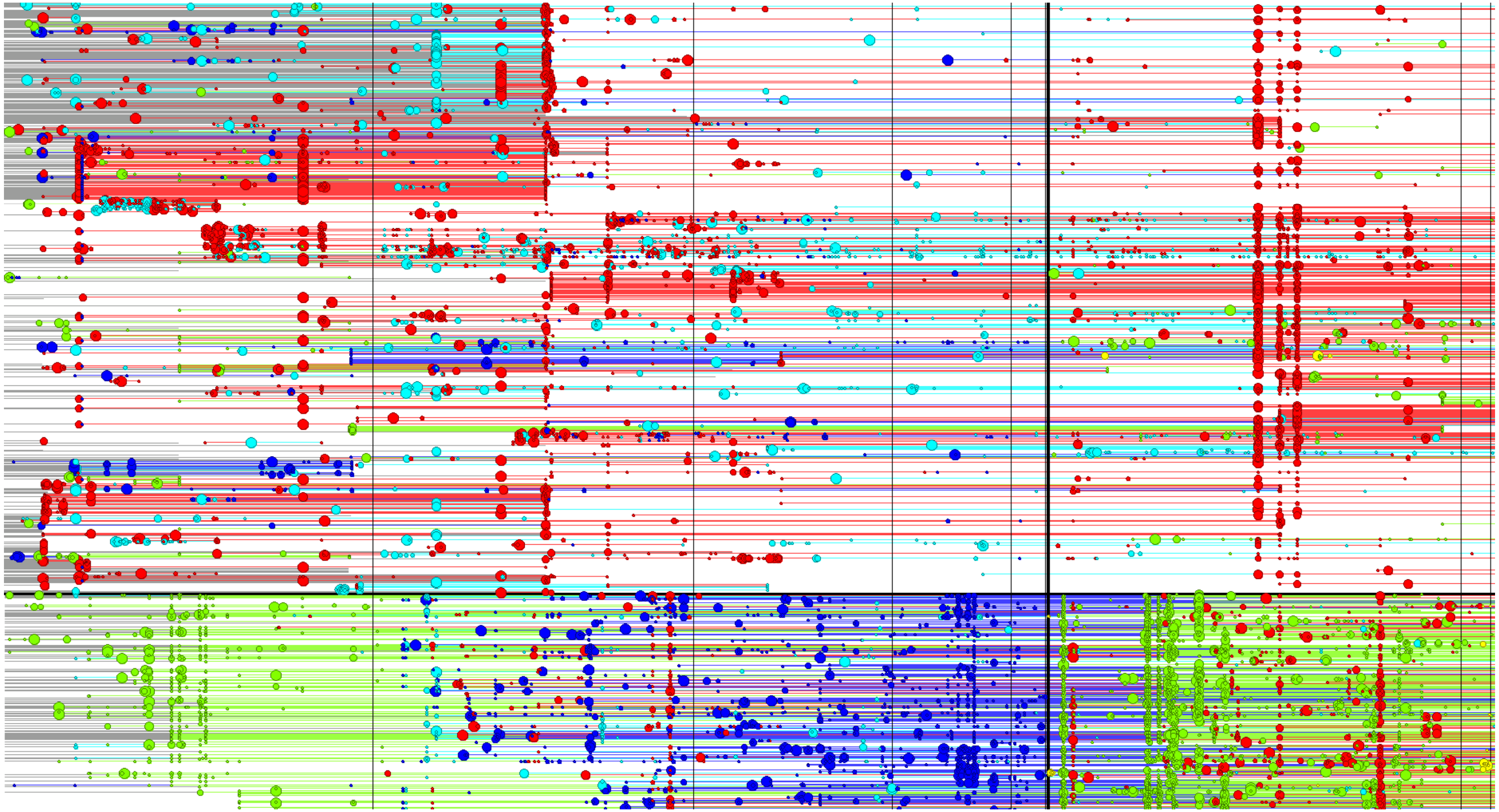
File B

Blue author small commit

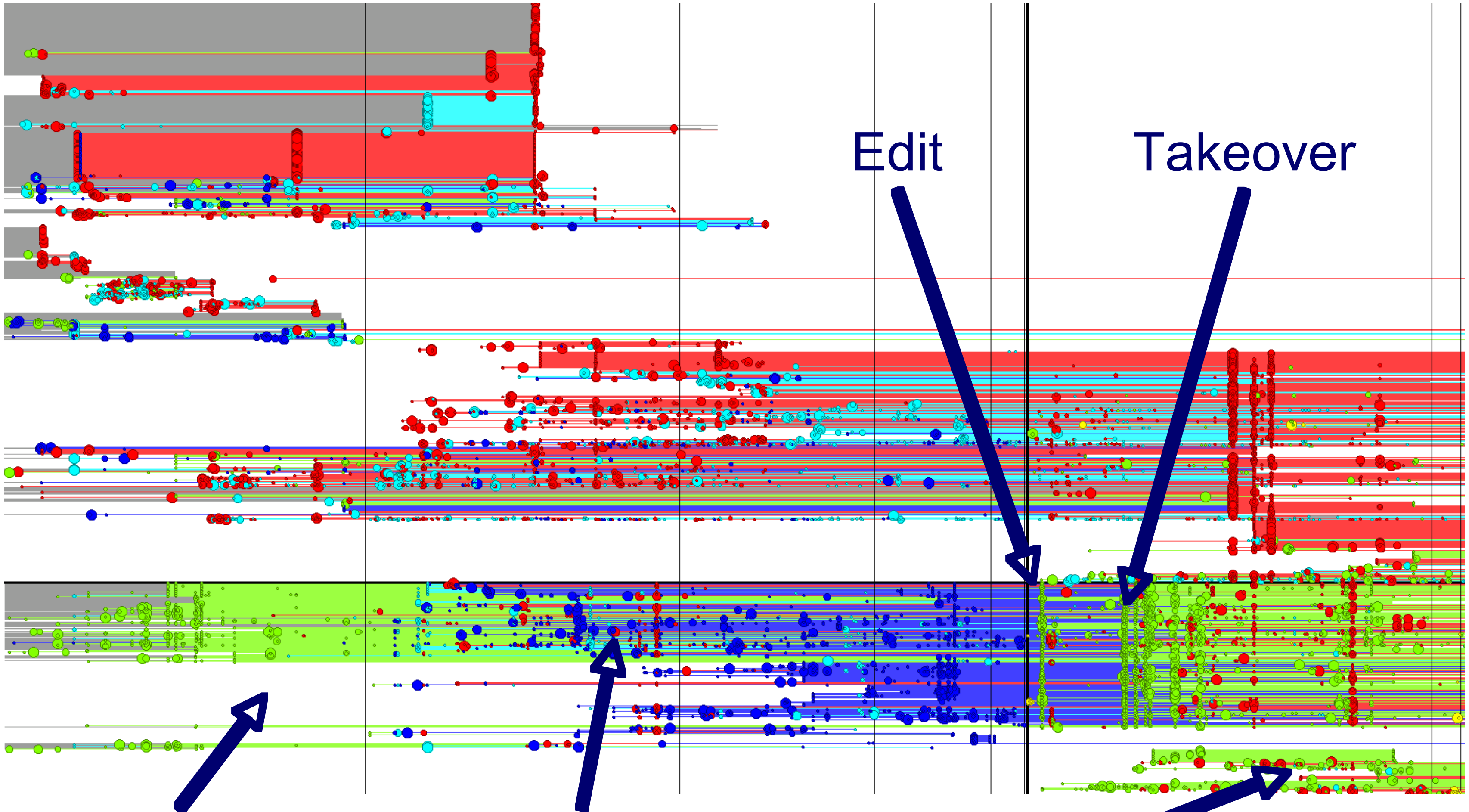
Which author “possesses” which files?



Alphabetical order is no order!



Based on similar commit signature



Language Independent

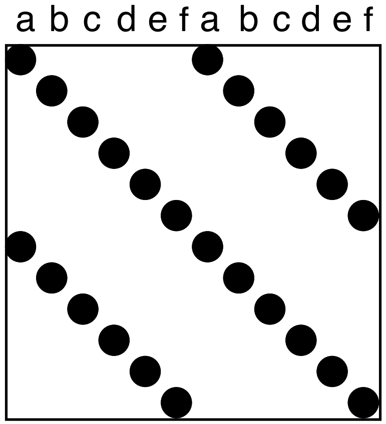
Language independent, Textual,
[ICSM'99], M. Rieger's PhD. Thesis

Duploc handled
Pascal, Java, Smalltalk, Python,
Cobol, C++, PDP-11, C

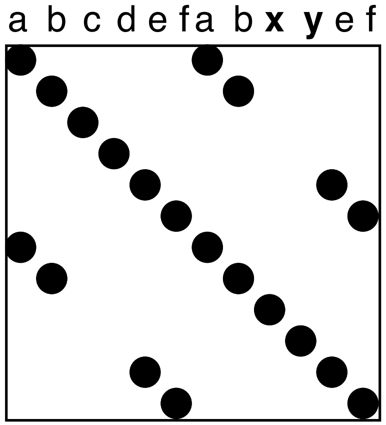
Slower than other approaches but...

Max 45 min to adapt our approach to a new language

Between 3% and 10%
less identification than parametrized match

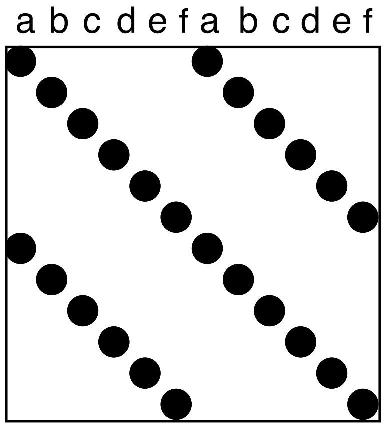


Exact Copies

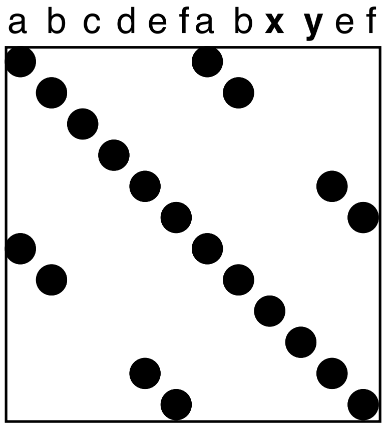


Copies with

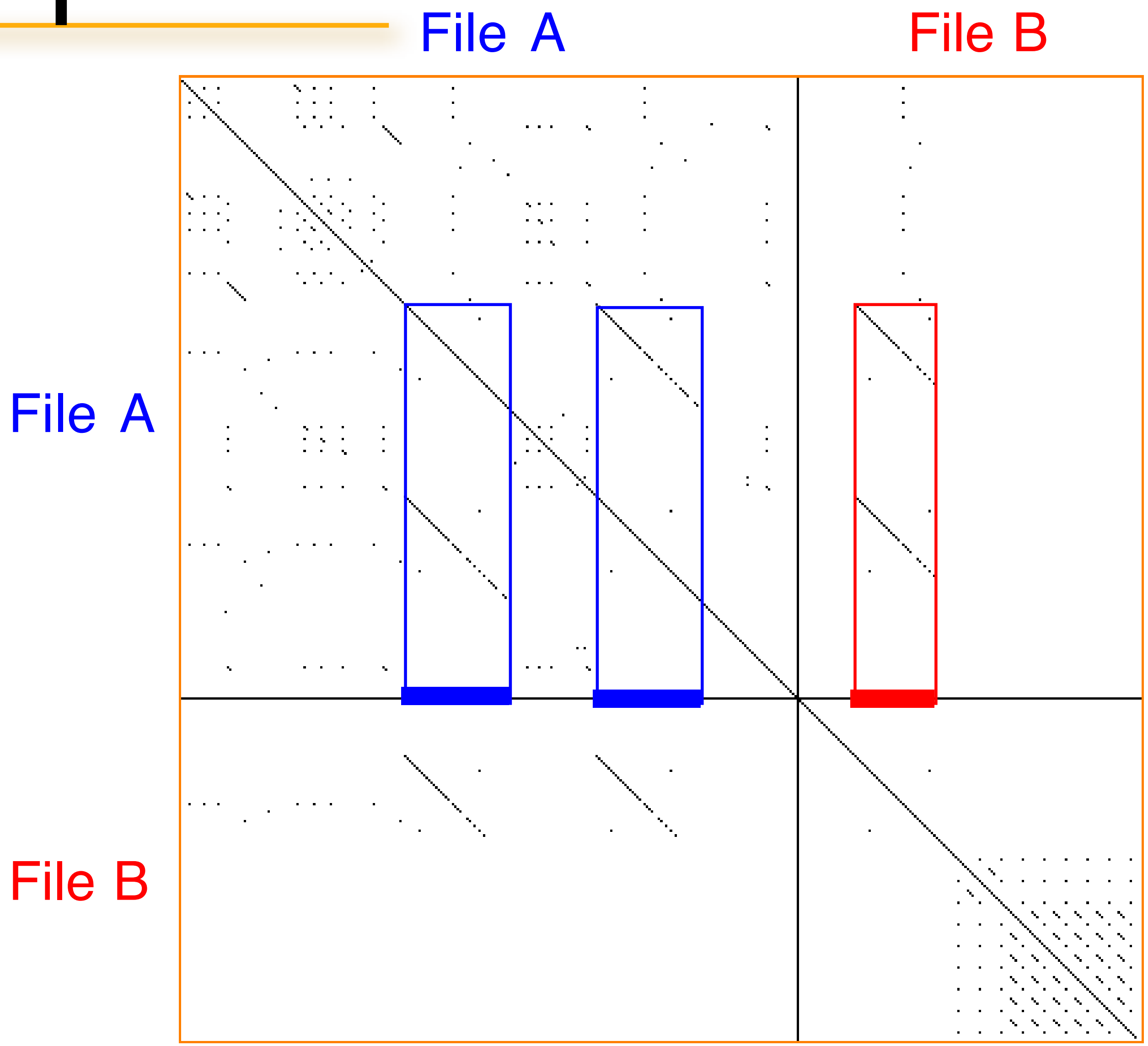
A Conceptual Matrix



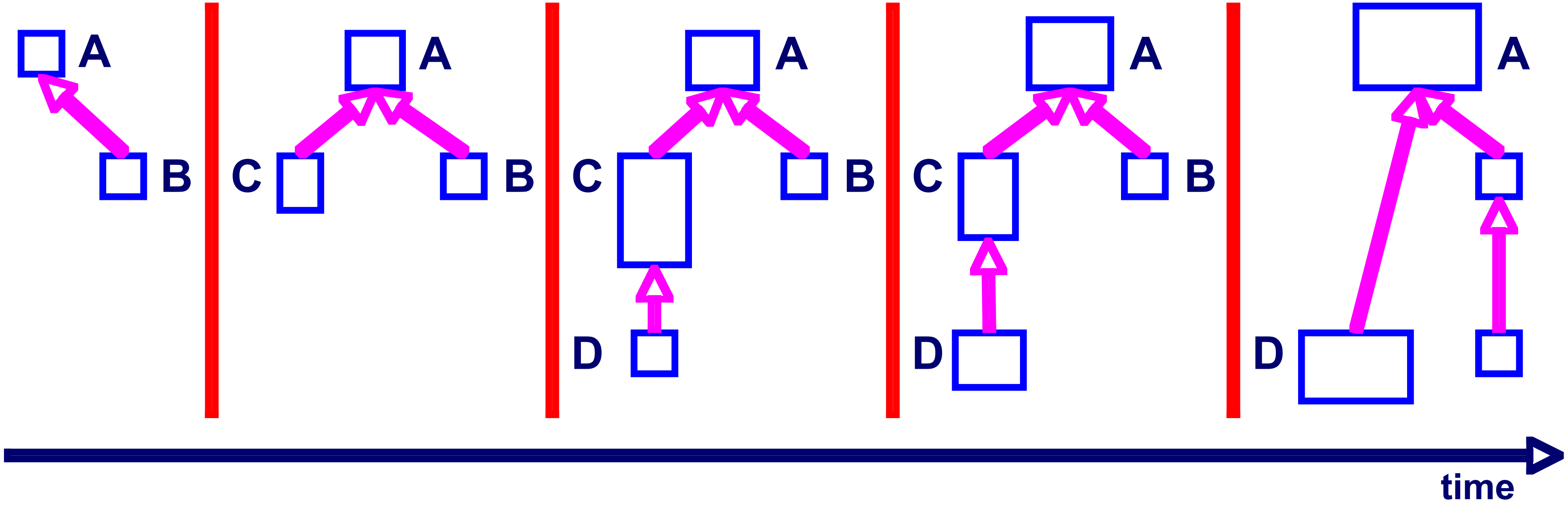
Exact Copies



Copies with Variations



Evolution holds useful information

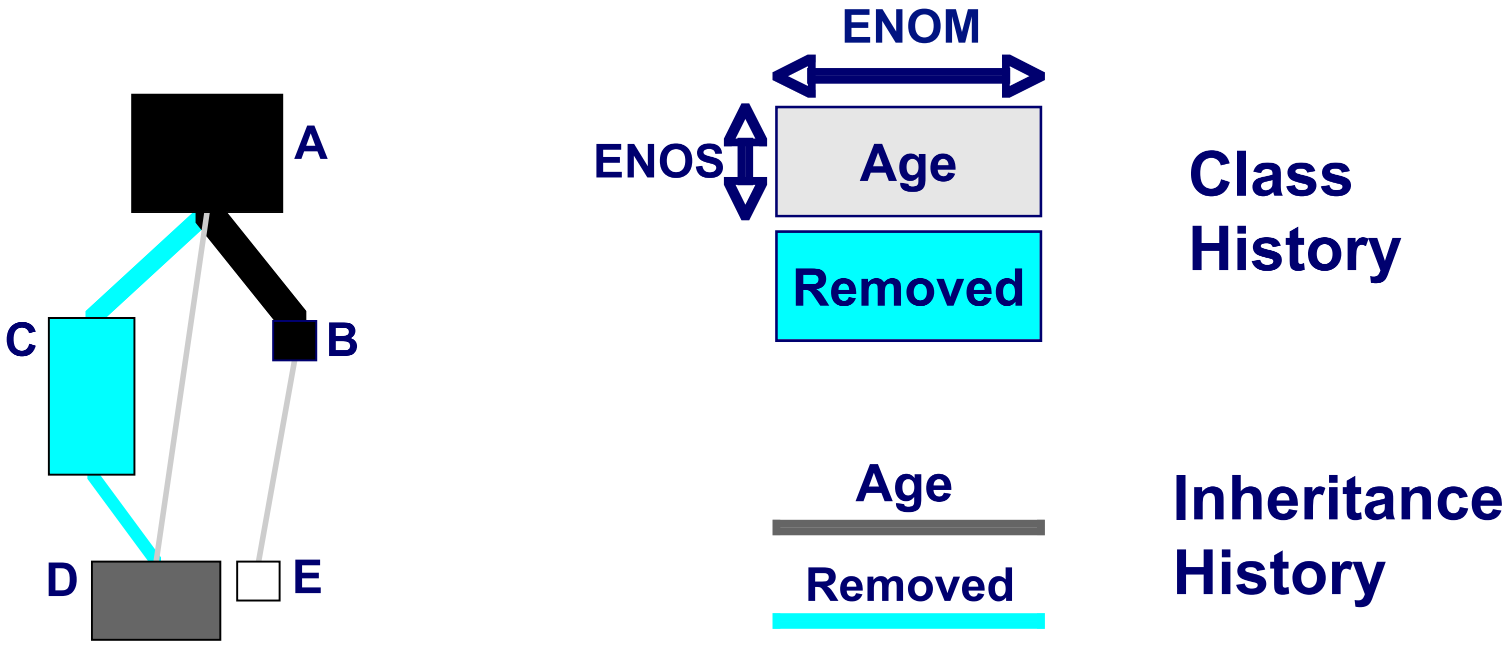


A is persistent   C was removed

B is stable   E is newborn

D inherited from C and then from A   …

Hierarchy Evolution Complexity View characterizes class hierarchy histories

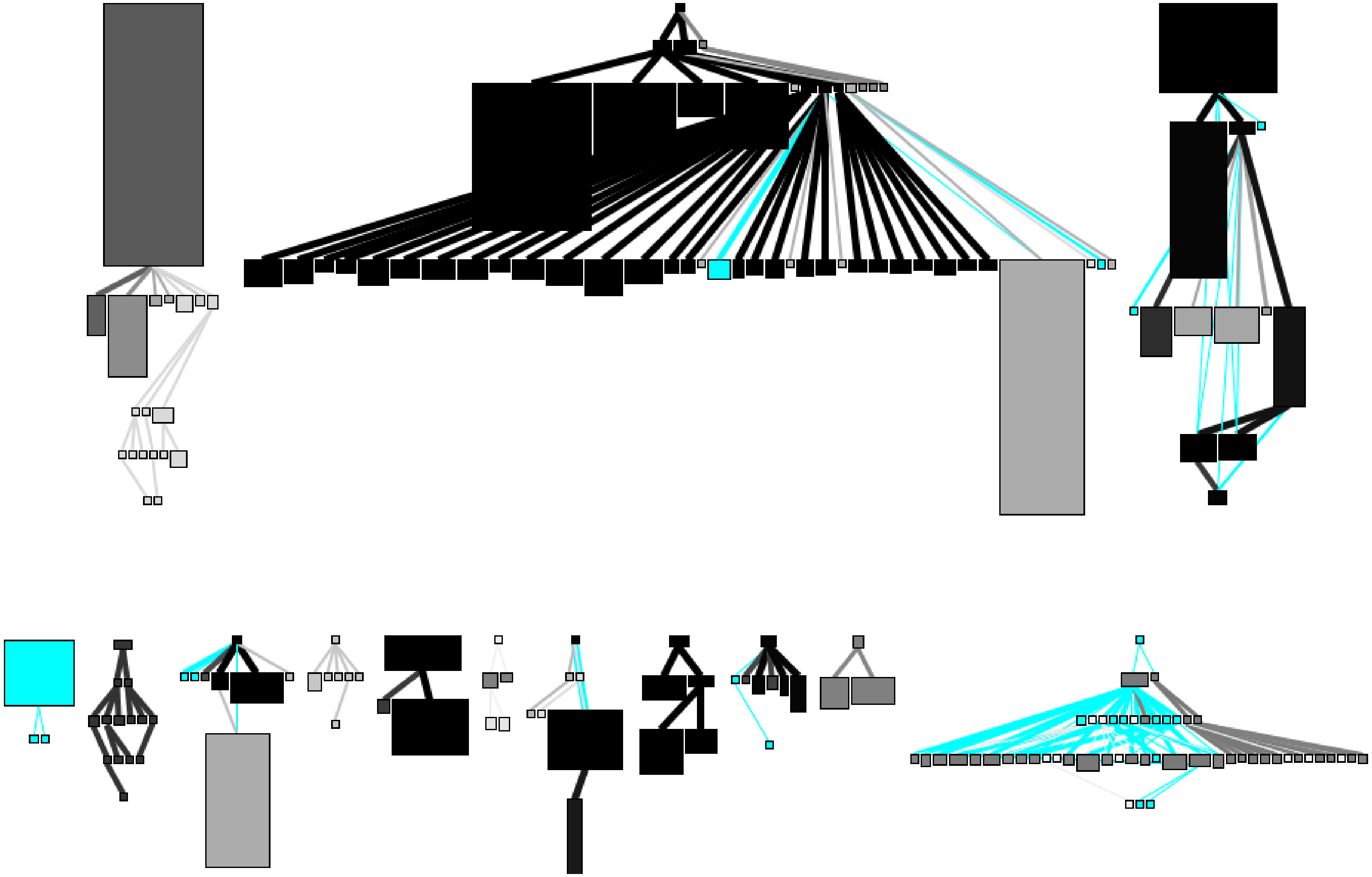


A is persistent C was removed

B is stable E is newborn

D inherited from C and then from A …

Class hierarchies over 40 versions of Jun - a 740 classes, 3D framework



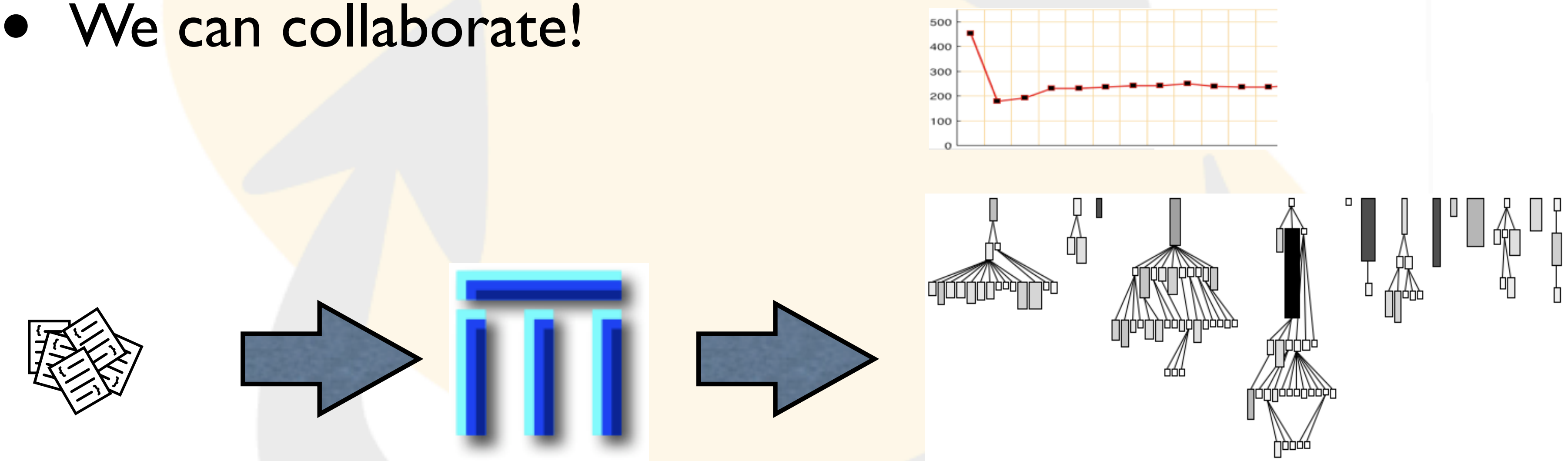
We are interested in your

Remodularization/Repackaging
SOA - Service Identification
Architecture Extraction/Validation
Software Quality
Cost prediction
EJB Analysis
Business rules extraction
Model transformation

and also language challenges...

Evolution is difficult

- We are interested in ***your*** problems!
- Moose is open-source, you can use it, extend it, change it
- We can collaborate!



NOM > 10 &
LOC > 100