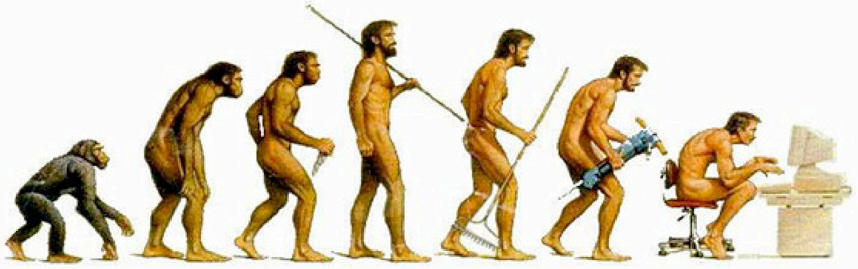
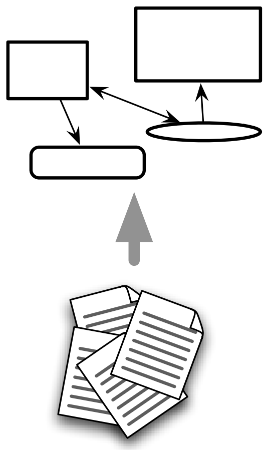


https://lore.cmi.ua.ac.be/serious

Fact Extraction



Tool CHain

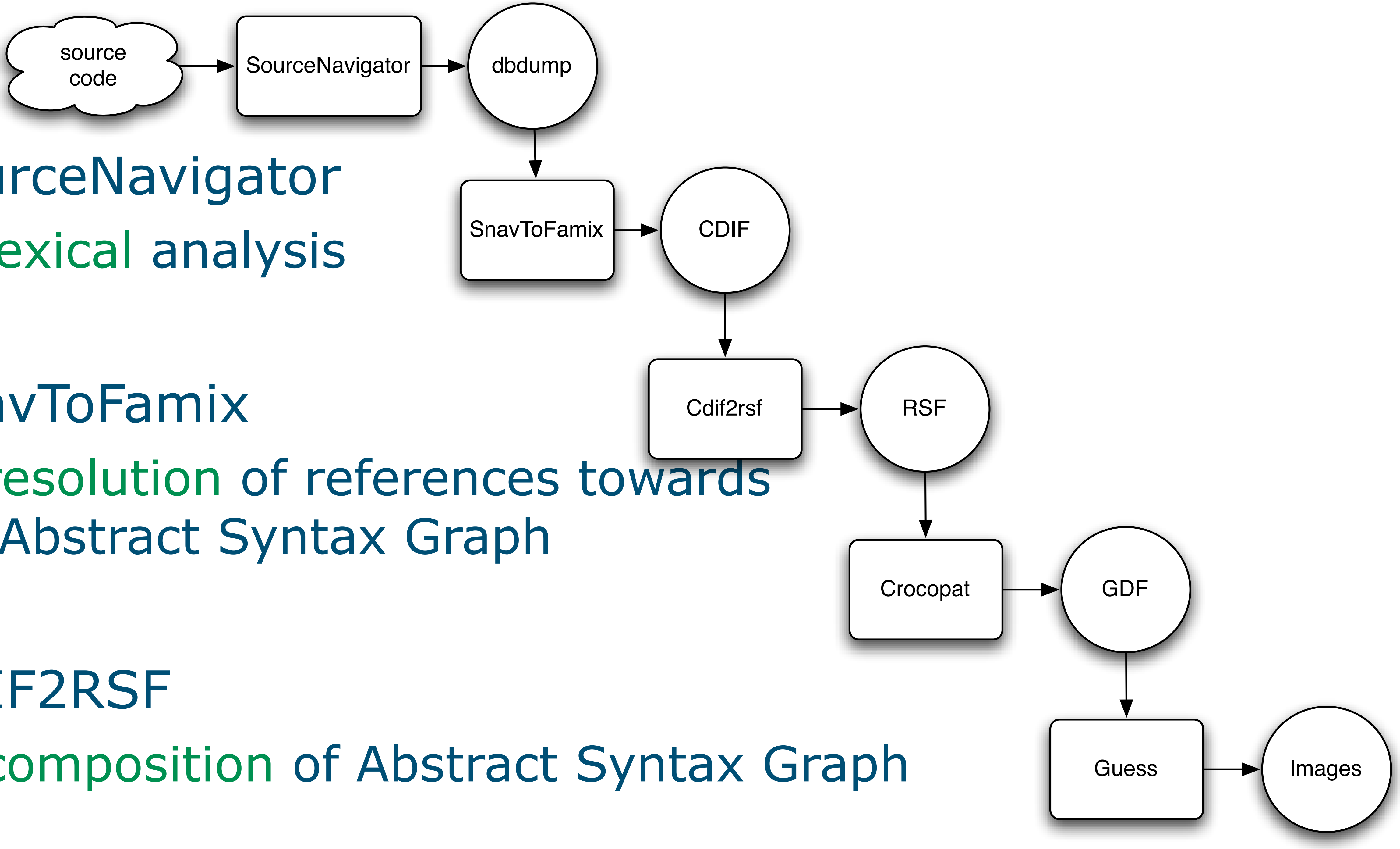
FETCH
a Fact Extraction Tool CHain

Bart Du Bois and Bart Van Rompaey
Lab On REengineering (LORE)
Dept. of Mathematics and Computer Science

Goals

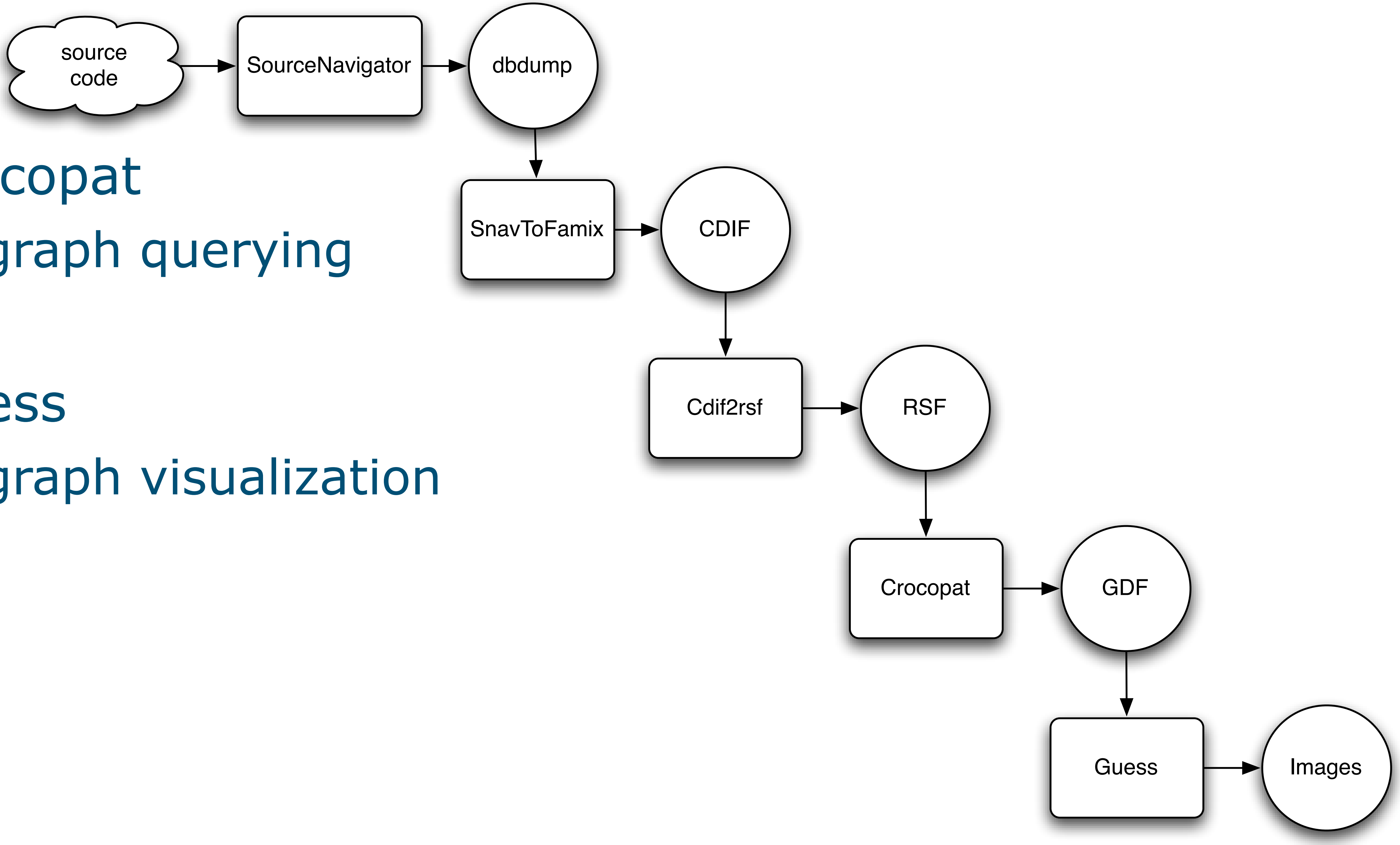
- Enable static analysis of large C++ systems
  - Robustness
    - should not require compilable code
  - Performance
    - should enable fast check-adjust-check cycles
  - Integration
    - should enable post-processing with other reverse engineering/reengineering environments, e.g. MOOSE
  - Licensing
    - no commercial licenses

Chosen solution: tool chain



- SourceNavigator
  - lexical analysis
- SnavToFamix
  - resolution of references towards Abstract Syntax Graph
- CDIF2RSF
  - composition of Abstract Syntax Graph

Chosen solution: tool chain

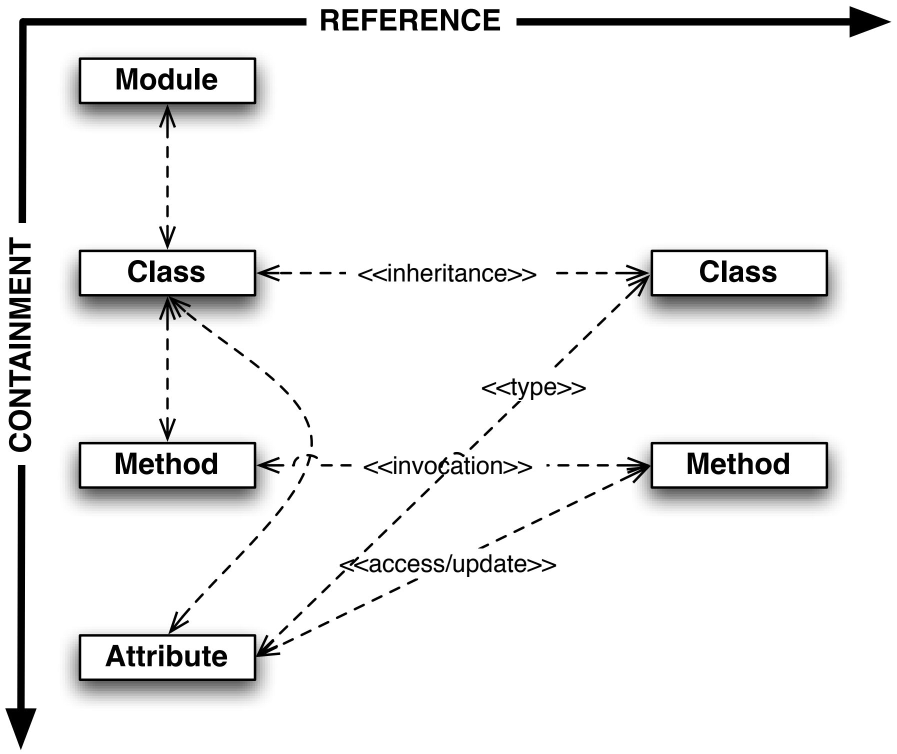


- Crocopat
  - graph querying
- Guess
  - graph visualization

Output

- Abstract Syntax Graph as FAMIX model
  - "standard" in reverse engineering/reengineering

- Analyses as queries
  - Simple queries
    - localization of entities/relations
    - navigation through graph
  - Advanced queries
    - (anti-)pattern detection



Typical usage scenario’s

- Design recovery
- Study the exceptional entities
- Dependency analysis
  - Clustering analysis
  - Manual navigation
- Pattern detection
- Metric calculation

Throughout:
visual representation for exploration, textual representation for reports

(Test) design recovery

- Identify entities and relations between them
  - e.g., focus on test code

Legend

○ production class

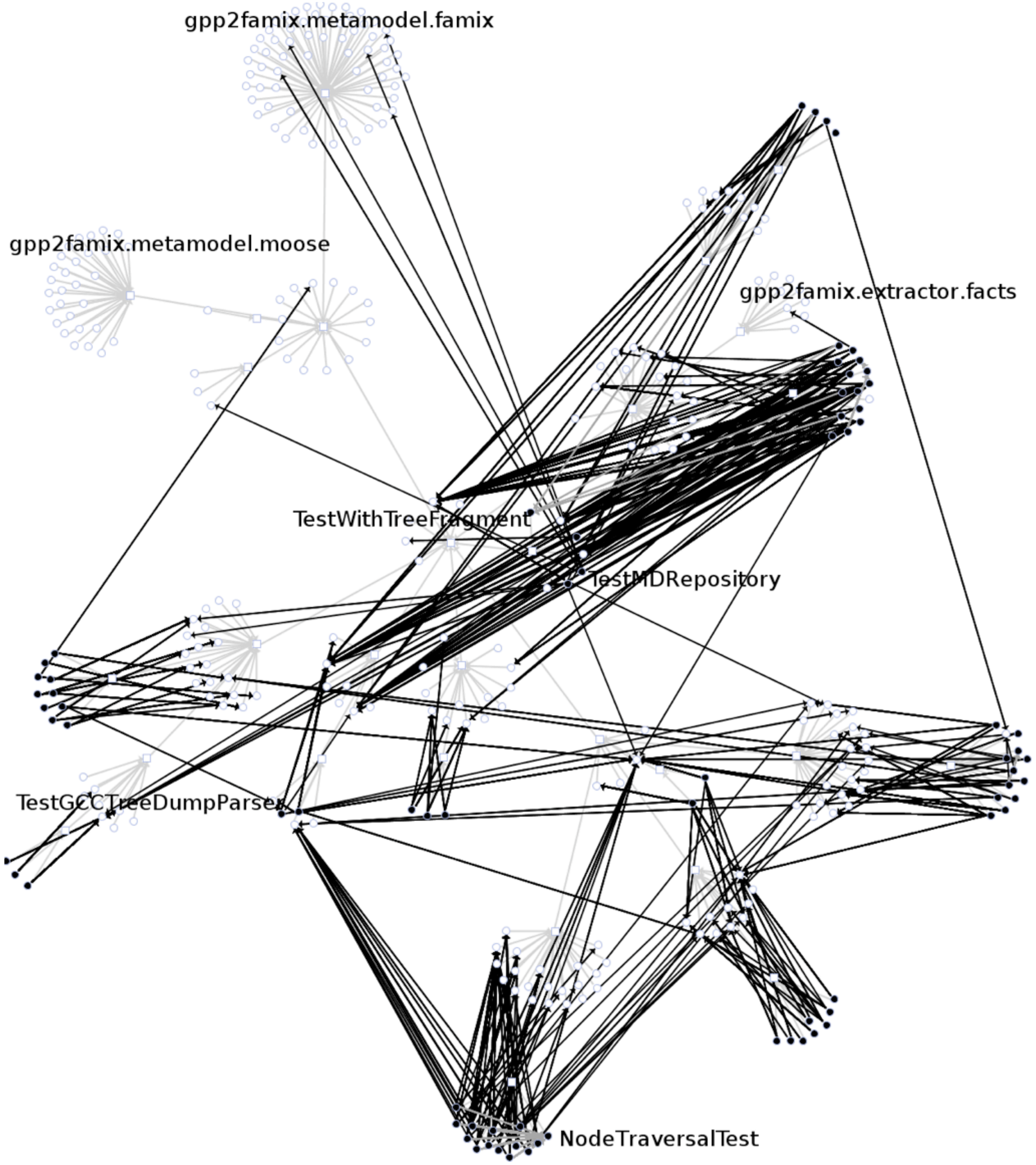
● test case

□ production module

■ test module

— containment

— test coverage

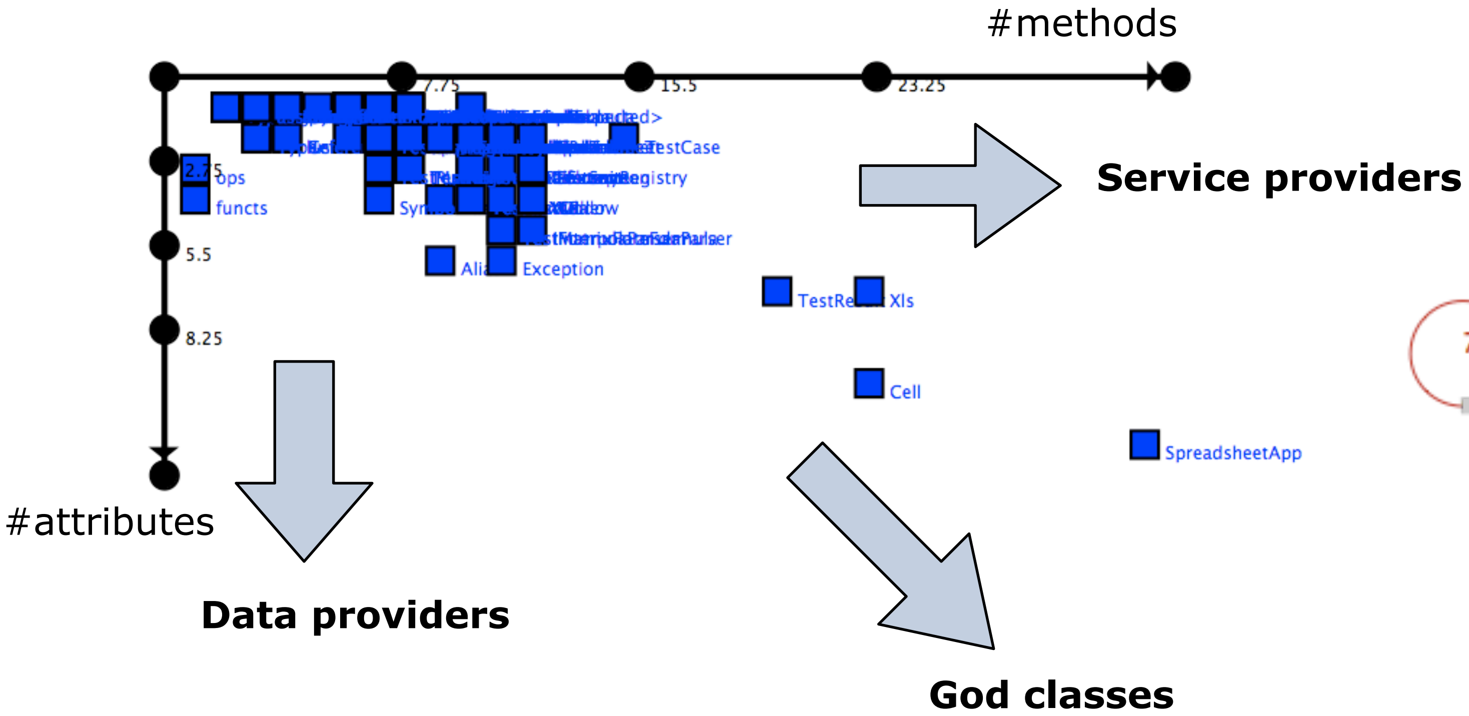

gpp2famix.metamodel.famix
gpp2famix.metamodel.moose
gpp2famix.extractor.facts
TestWithTreeFragment
TestMDRepository
TestGCCTreeDumpParser
NodeTraversalTest

Study the Exceptional Entities

- Identify potential design problems

@design-level

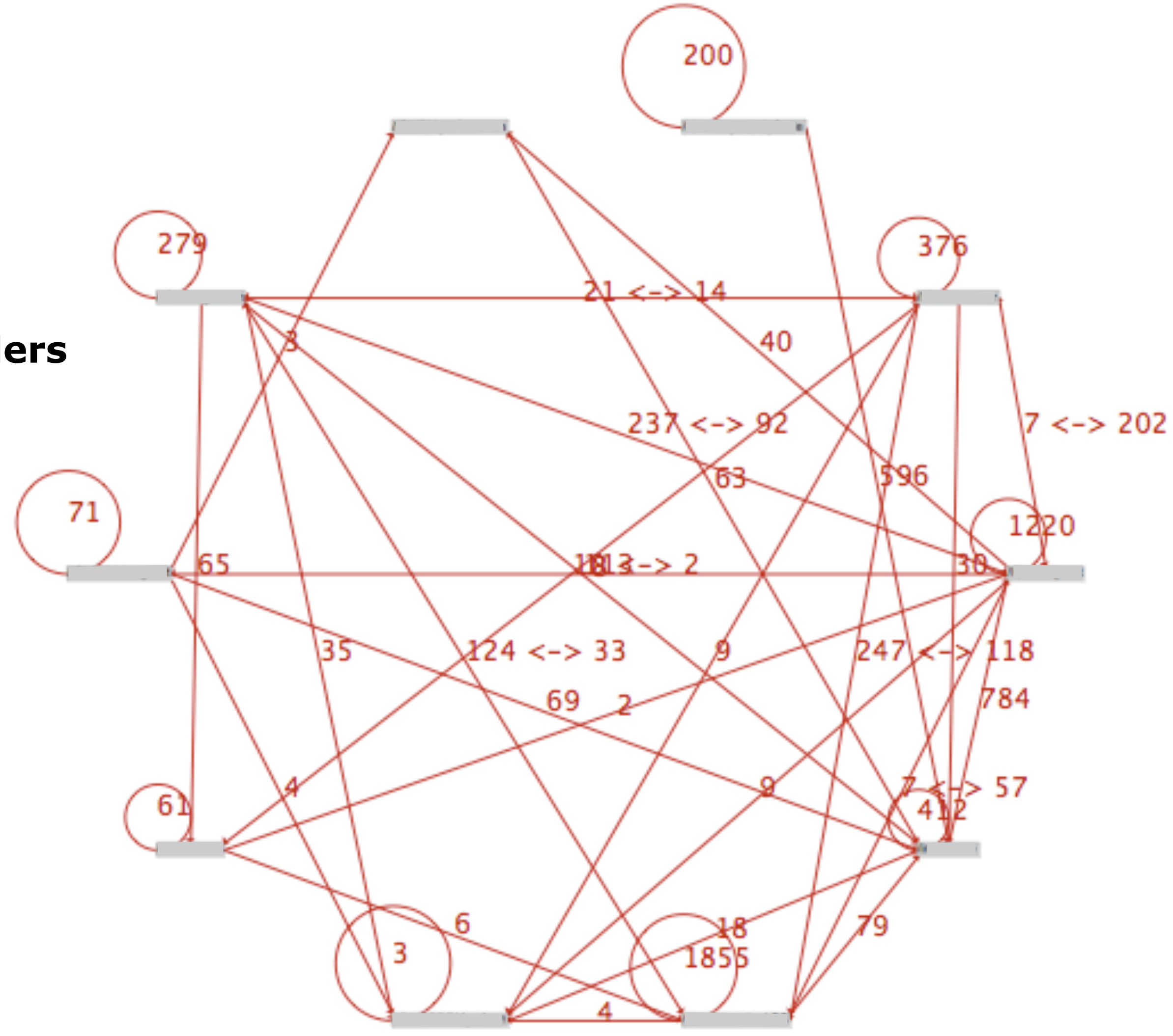
@arch-level



Service providers

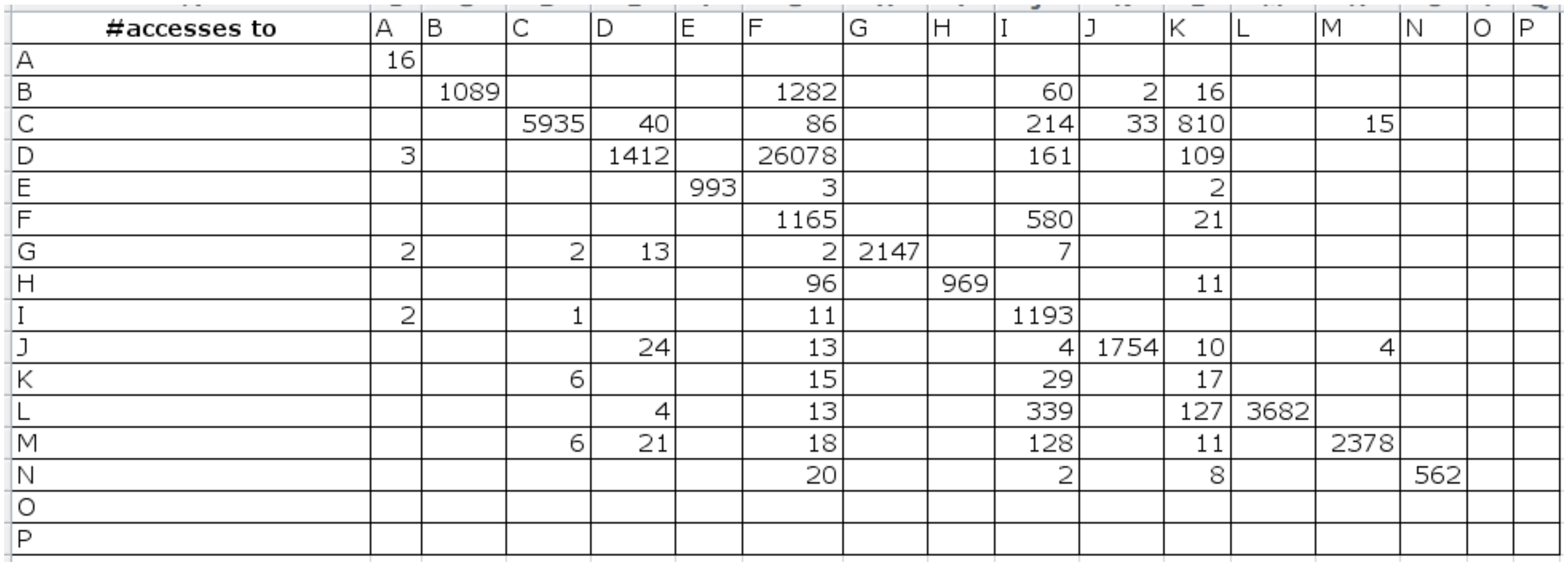
Data providers

God classes



Dependency analysis

- Identify dependencies between model parts
  - e.g., between modules



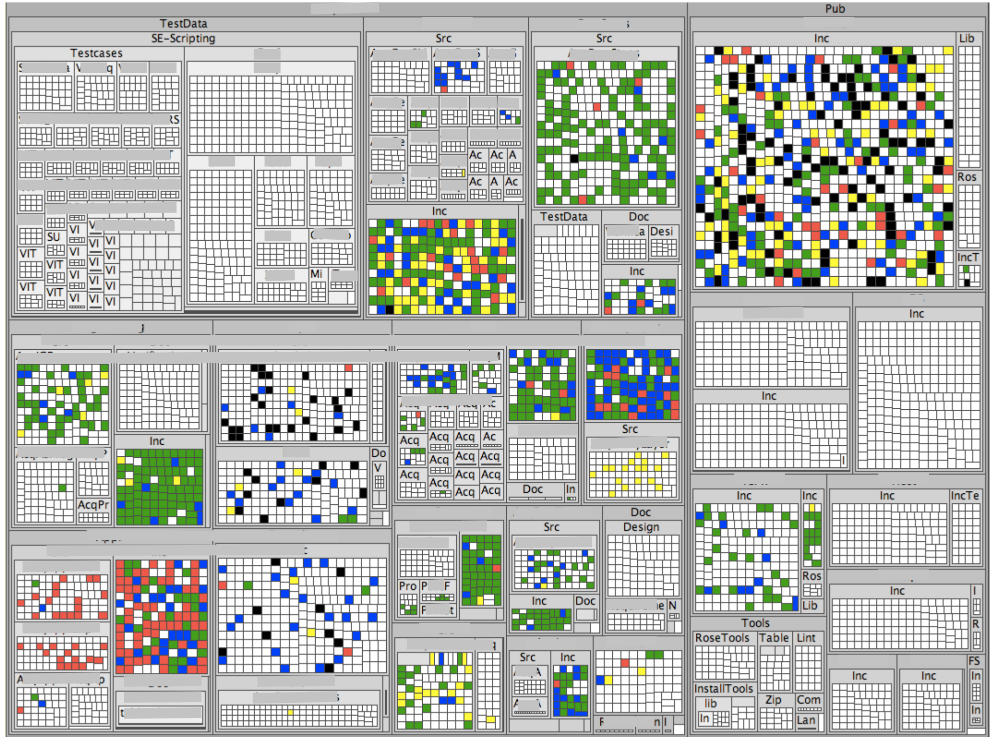
#accesses to	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
A	16															
B		1089				1282			60	2	16					
C			5935	40		86			214	33	810		15			
D	3			1412		26078			161		109					
E					993	3					2					
F						1165			580		21					
G	2		2	13		2	2147		7							
H						96		969			11					
I	2		1			11			1193							
J				24		13			4	1754	10		4			
K			6			15			29		17					
L				4		13			339		127	3682				
M			6	21		18			128		11		2378			
N						20			2		8			562		
O																
P																

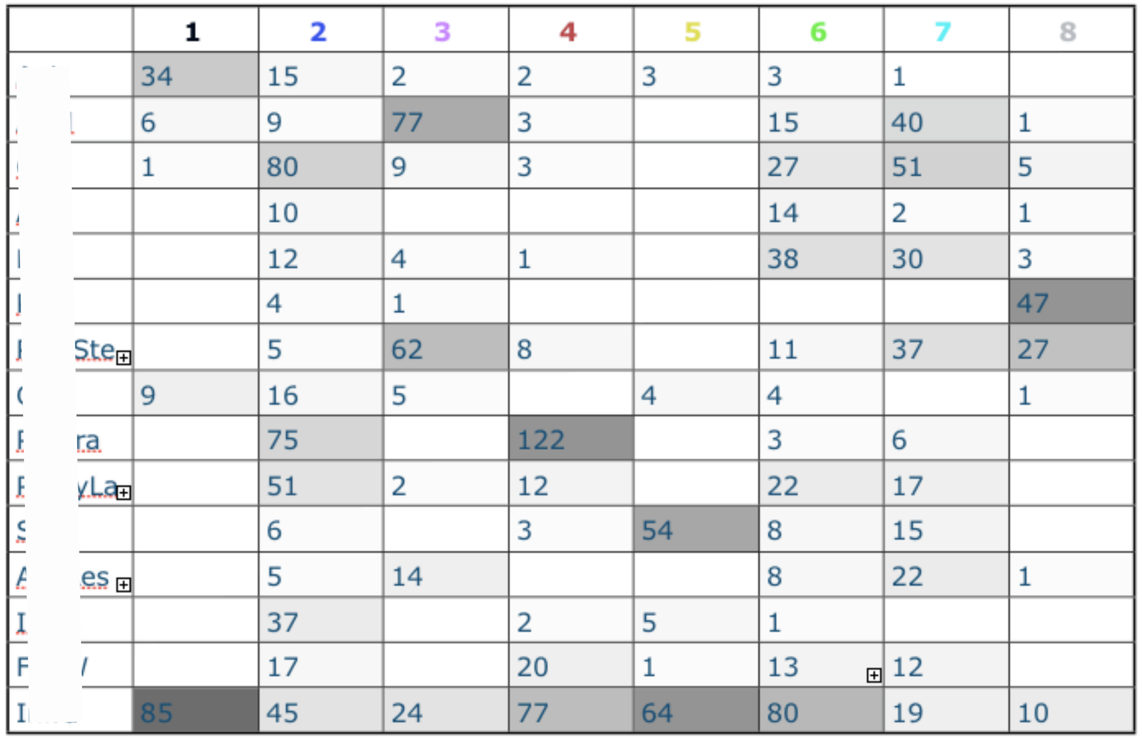
Cluster analysis

- Compare proposed decomposition with actual

Are classes in the same module interacting strongly? (same color)

Are strongly interacting classes grouped in a single module?

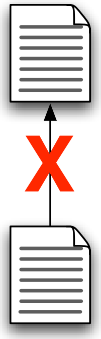




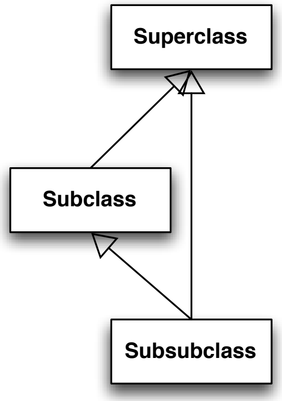
	1	2	3	4	5	6	7	8
	34	15	2	2	3	3	1	
	6	9	77	3		15	40	1
	1	80	9	3		27	51	5
		10				14	2	1
		12	4	1		38	30	3
		4	1					47
Ste⊞		5	62	8		11	37	27
	9	16	5		4	4		1
ra		75		122		3	6	
yLa⊞		51	2	12		22	17	
S		6		3	54	8	15	
es ⊞		5	14			8	22	1
I		37		2	5	1		
F /		17		20	1	13 ⊞	12	
I	85	45	24	77	64	80	19	10

Pattern detection

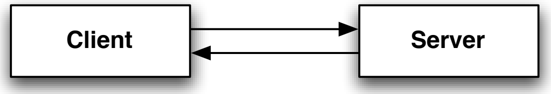
- Identify occurrences of specified relationships
  - E.g., redundant includes



  - Redundant inheritance relations



  - Cyclic dependencies



Metric calculation

- Typically in function of previous analyses
  - coupling / cohesion
    - incoming dependencies
    - outgoing dependencies
  - complexity
    - cyclomatic complexity
    - large functions/methods/classes
    - complex navigation code
      - e.g., invocation/access chaining
    - complex inheritance hierarchies

Current status

- Ongoing incremental and iterative development

- Undergoing manual verification of results
  - completeness
  - validity

- Recording interesting usage scenario's
  - relevant queries
  - improve user-friendliness