CWI
Centrum Wiskunde & Informatica

Universiteit van Amsterdam

SOGYO

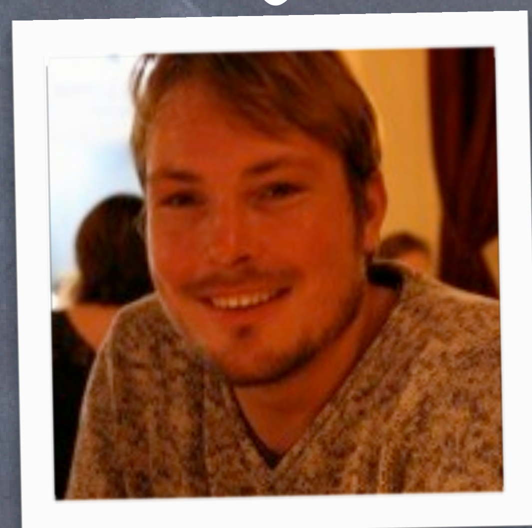
Constructing specialist software tools using Rascal

@jurgenvinju
April 24th 2012
@sogyo

Rascal Team

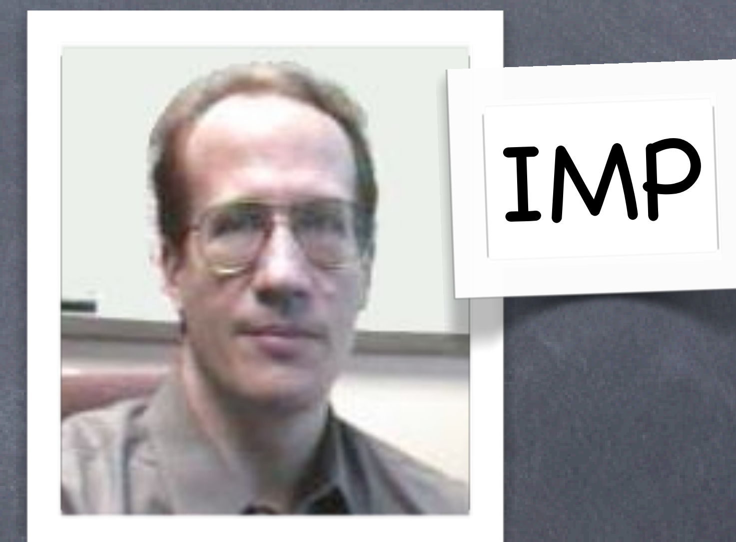
Paul
Klint

Jurgen
Vinju



Tijs
v/d Storm

Bob
Fuhrer



Centrum Wiskunde & Informatica

- Centrum Wiskunde & Informatica (www.cwi.nl)
  - where programming languages come from (1968)
  - where the internet started for Europe (1988)
- Fundamental research
  - algorithms, theories, languages, models, tools
- High societal relevance
  - Life Sciences, Energy, Logistics, Data, Software

CWI

Centrum Wiskunde & Informatica

- Centrum Wiskunde & Informatica (www.cwi.nl)
  - where programming languages come from (1968)
  - where the internet started for Europe (1988)
- Fundamental research
  - algorithms, theories, languages, models, tools
- High societal relevance
  - Life Sciences, Energy, Logistics, Data, Software

Python

Algol

W3C

CWI

Centrum Wiskunde & Informatica

- Centrum Wiskunde & Informatica (www.cwi.nl)
  - where programming languages come from (1968)
  - where the internet started for Europe (1988)
- Fundamental research
  - algorithms, theories, languages, models, tools
- High societal relevance
  - Life Sciences, Energy, Logistics, Data, Software

Python

Algol

W3C

CWI

Centrum Wiskunde & Informatica

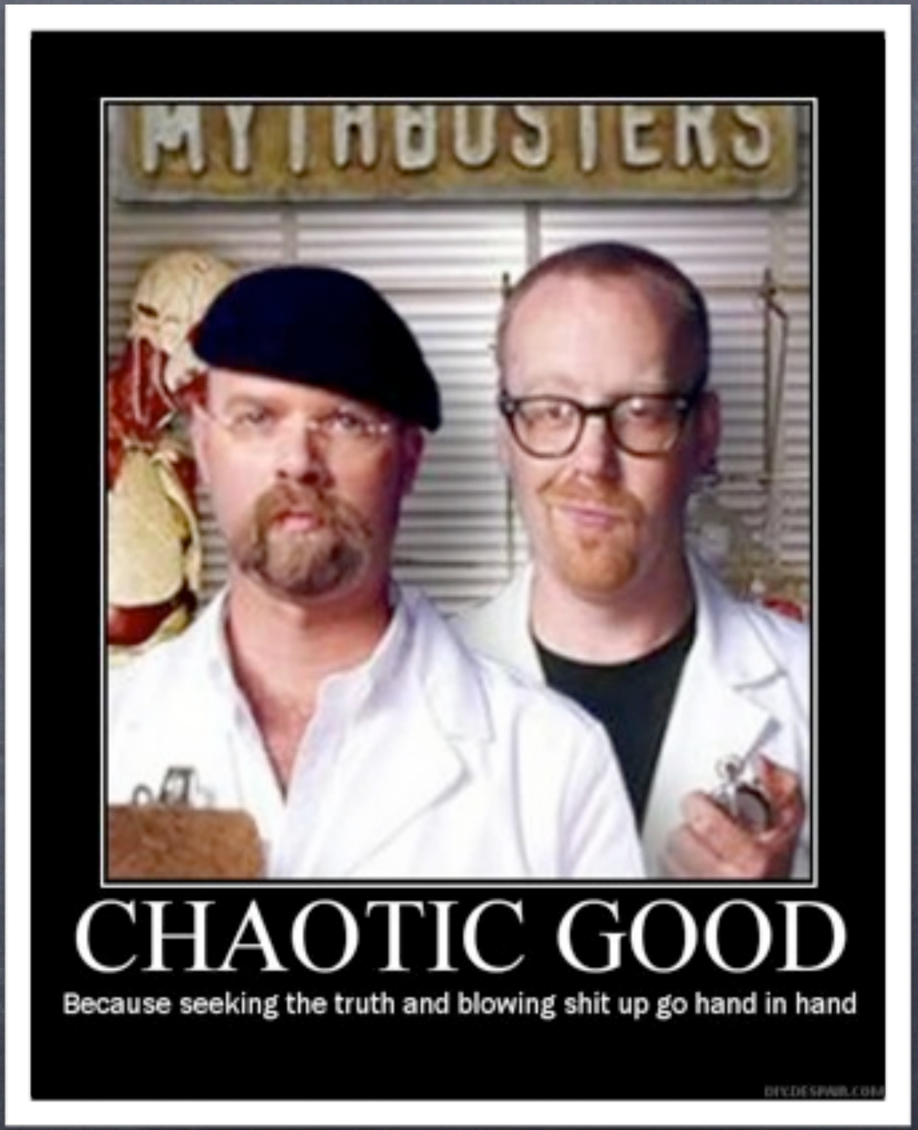
- Centrum Wiskunde & Informatica (www.cwi.nl)
  - where programming languages come from (1968)
  - where the internet started for Europe (1988)
- Fundamental research
  - algorithms, theories, languages, models, tools
- High societal relevance
  - Life Sciences, Energy, Logistics, Data, Software

Python

Algol

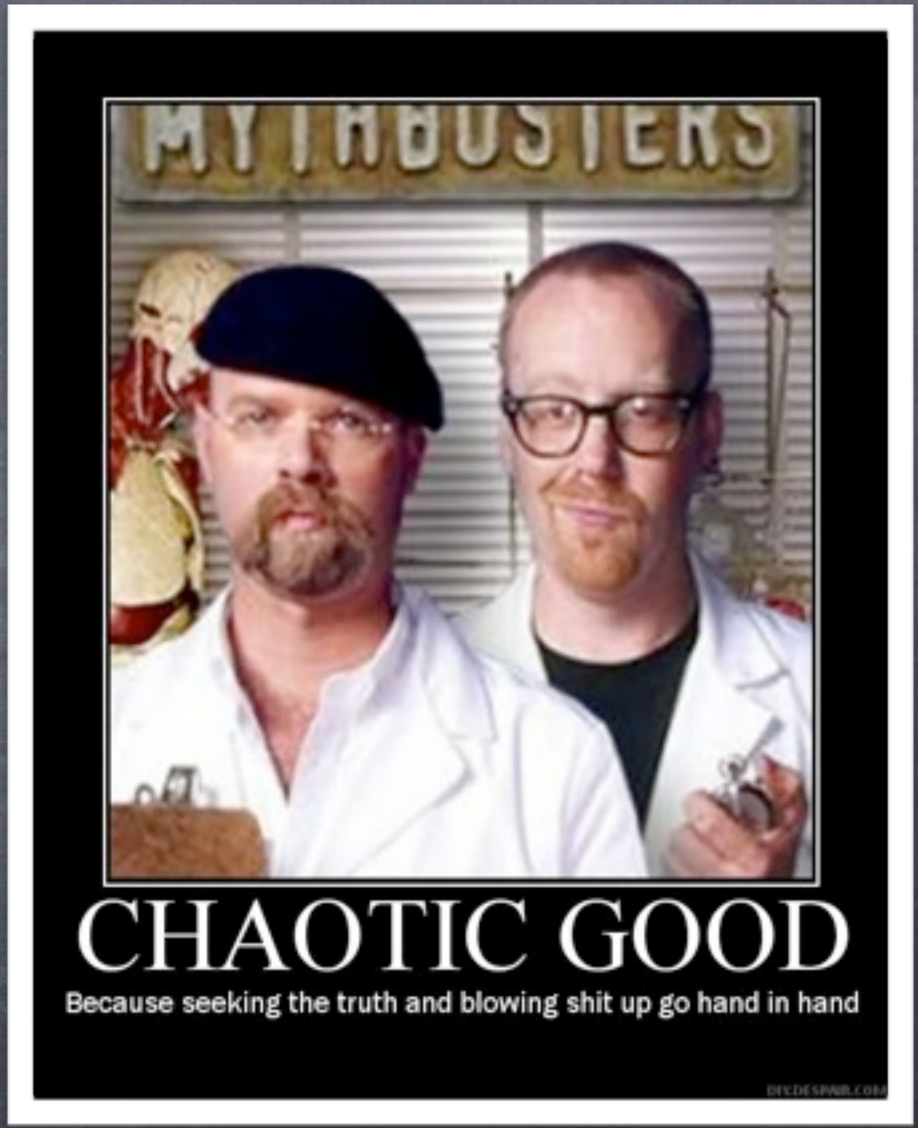
W3C

Disclaimer


MYTHBUSTERS
CHAOTIC GOOD
Because seeking the truth and blowing shit up go hand in hand

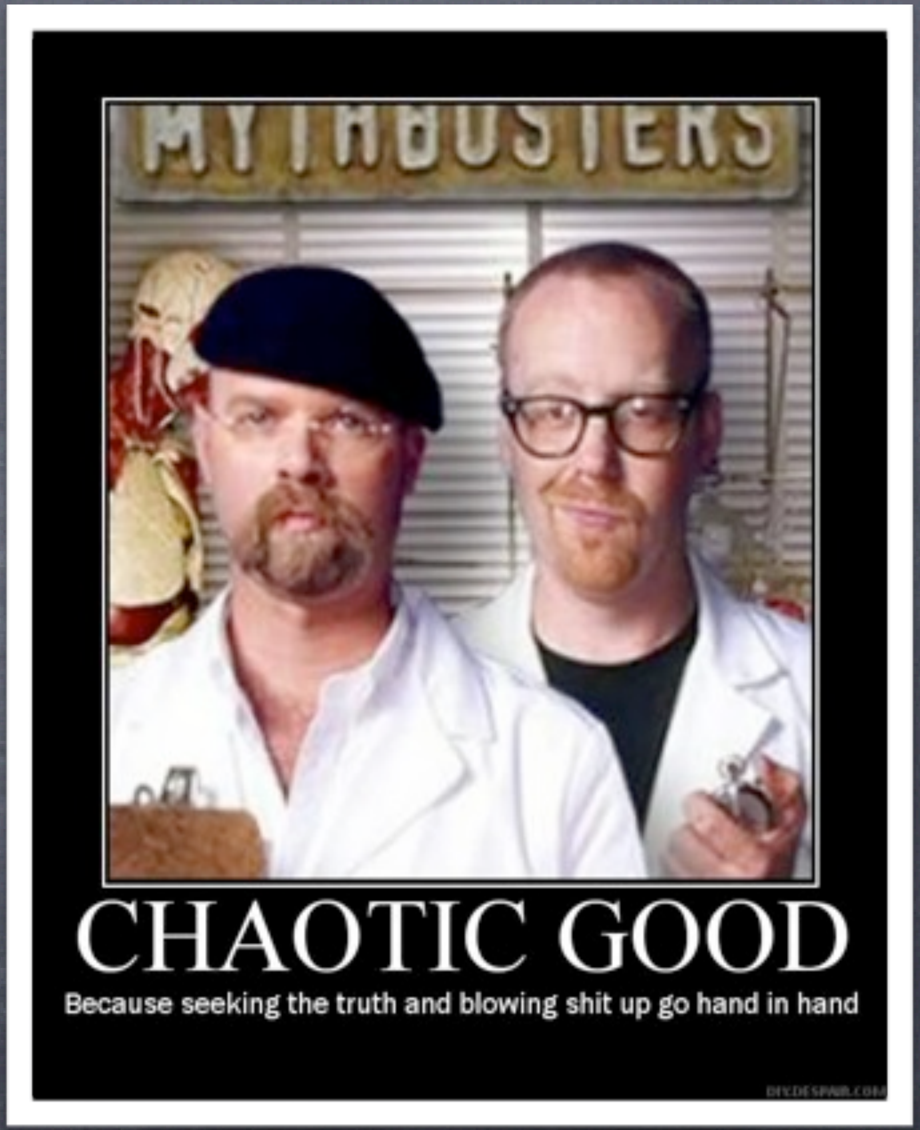
Disclaimer

- Do try this "at home"



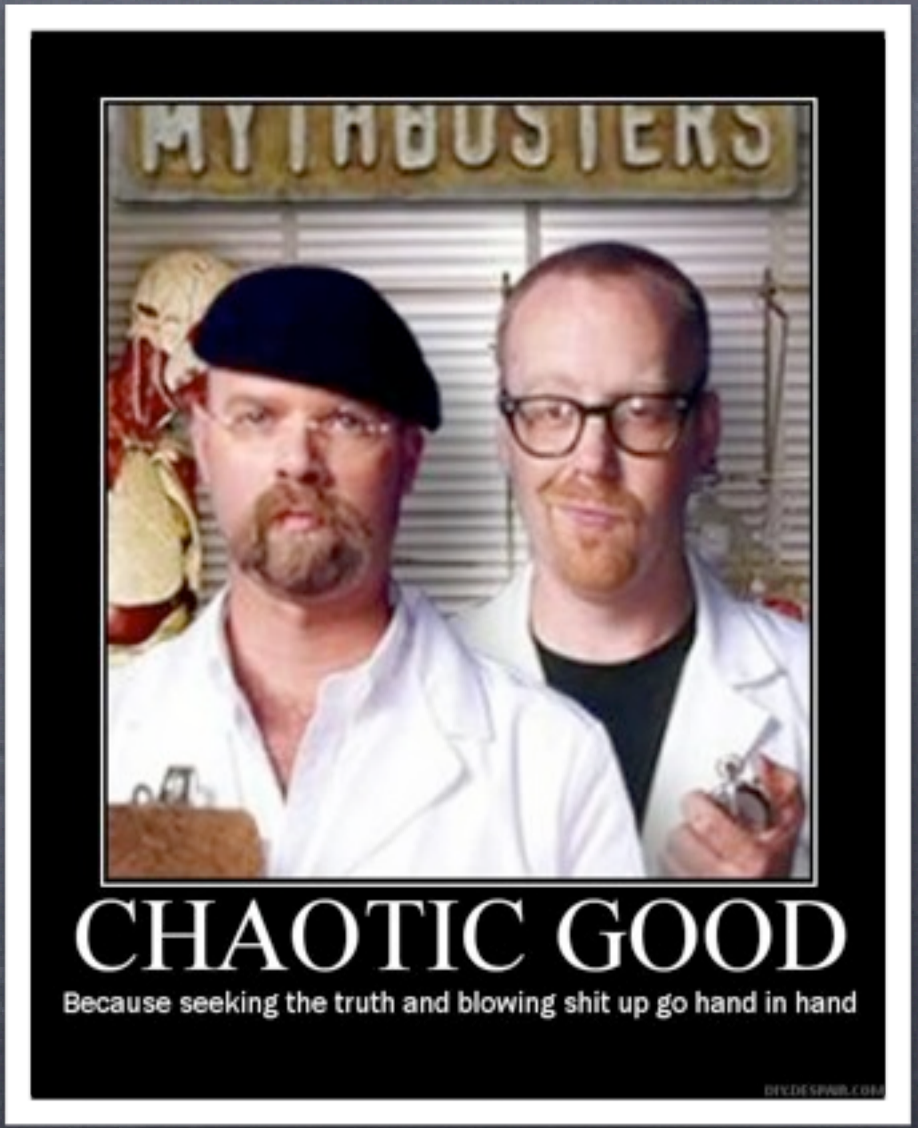
Disclaimer

- **Do try this "at home"**
- Rascal is under development & evaluation
  - beta quality
  - alpha guarantees
  - ready for proofs-of-concepts and one-offs



Disclaimer

- **Do try this "at home"**
- Rascal is under development & evaluation
  - beta quality
  - alpha guarantees
  - ready for proofs-of-concepts and one-offs
- We use it for our own research

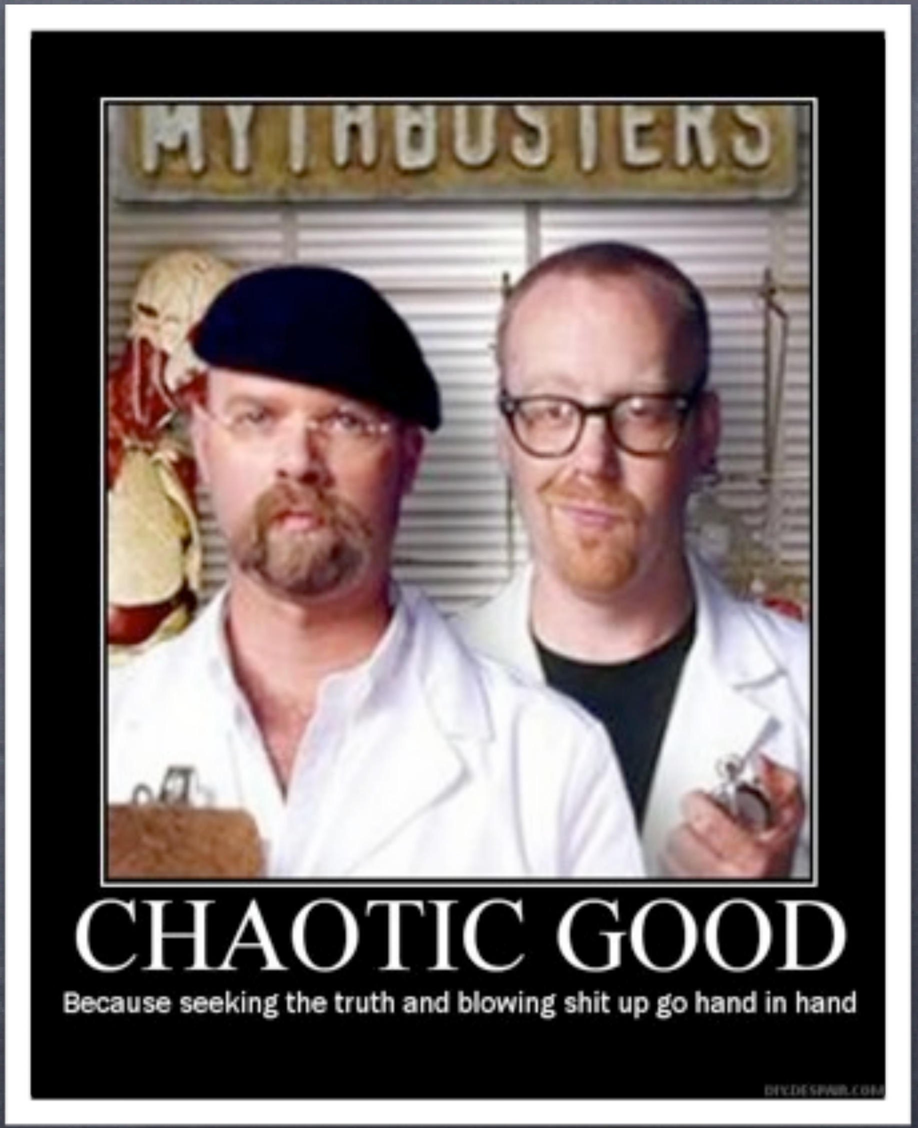


Disclaimer

- **Do try this "at home"**
- Rascal is under development & evaluation
  - beta quality
  - alpha guarantees
  - ready for proofs-of-concepts and one-offs
- We use it for our own research
- We use it in teaching at the Master Software Evolution (UvA) and Open Universiteit, etc.

Disclaimer

- **Do try this "at home"**
- Rascal is under development & evaluation
  - beta quality
  - alpha guarantees
  - ready for proofs-of-concepts and one-offs
- We use it for our own research
- We use it in teaching at the Master Software Evolution (UvA) and Open Universiteit, etc.
- Today is a not a crash course in Rascal programming
  - Check out http://tutor.rascal-mpl.org



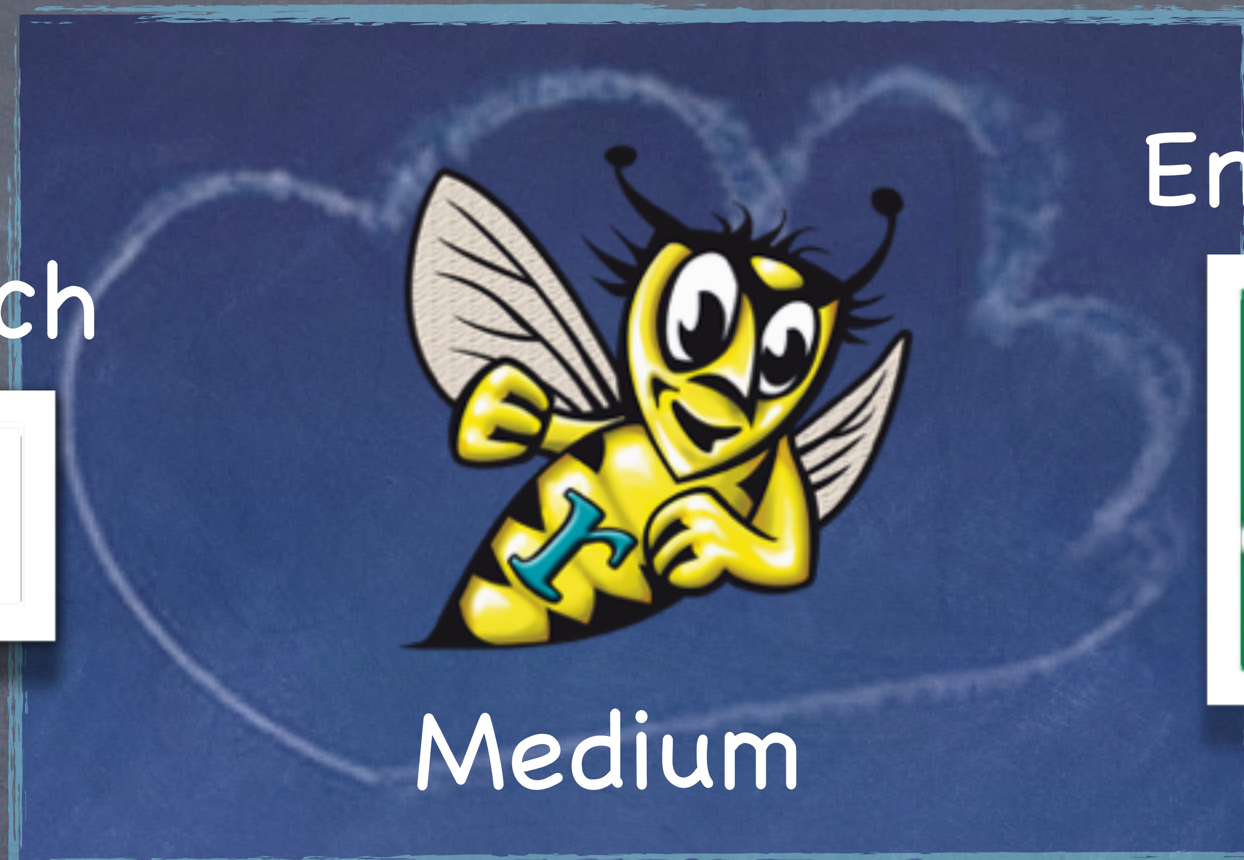
Why am I here?

Master Software Engineering

UNIVERSITEIT VAN AMSTERDAM

Engineering research

CWI Centrum Wiskunde & Informatica

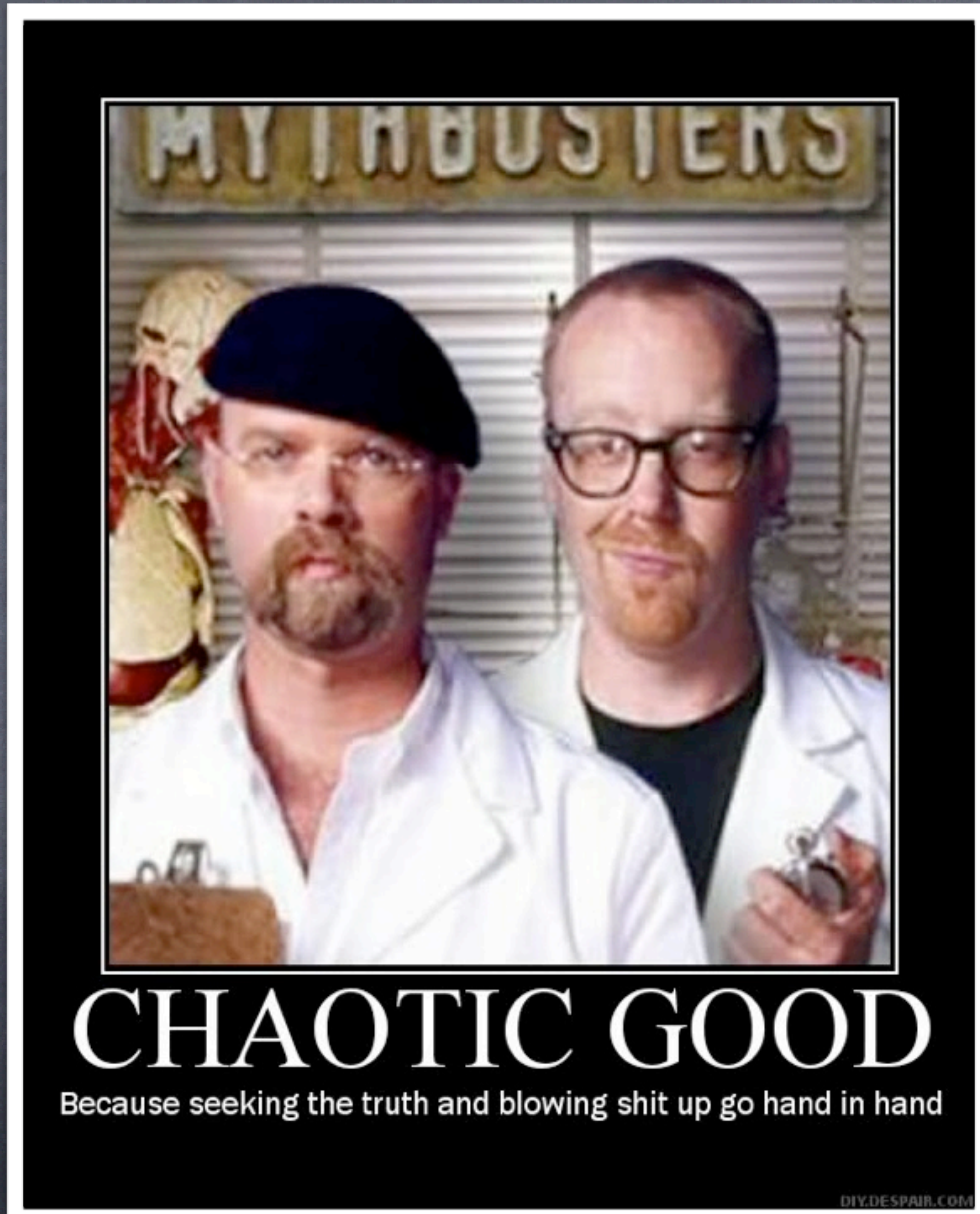


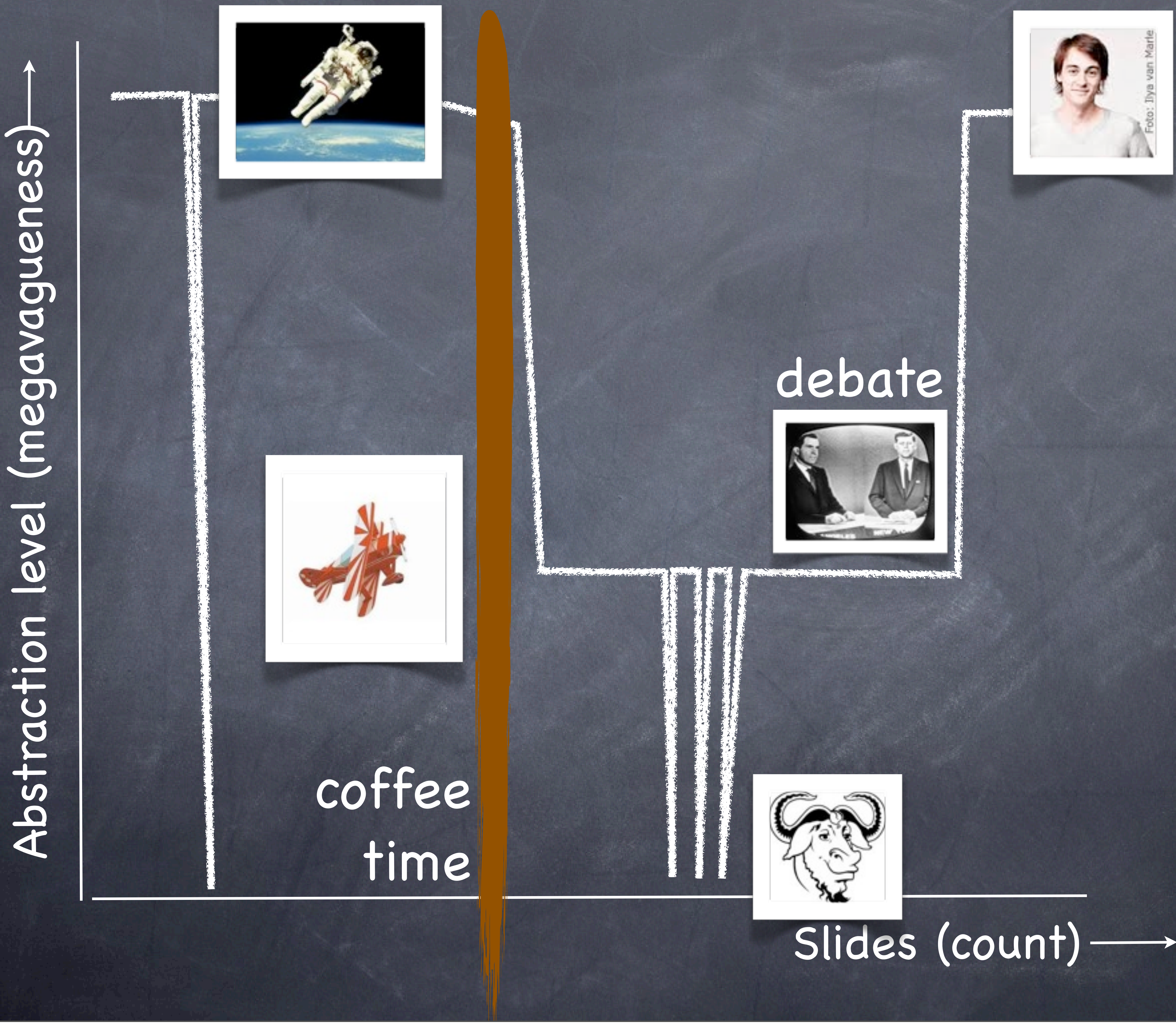
Medium

Engineering

SOGYO

Why am I here?

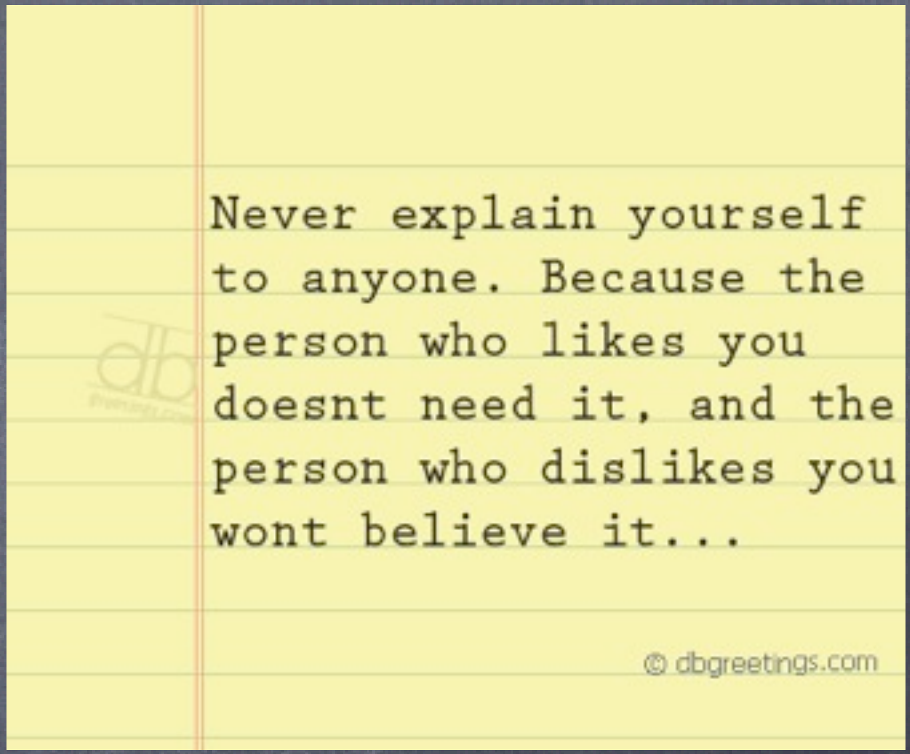

MYTHBUSTERS
CHAOTIC GOOD
Because seeking the truth and blowing shit up go hand in hand
DIY.DESPAIR.COM


Abstraction level (megavagueness)
Foto: Ilya van Marle
debate
coffee
time
Slides (count)

First 45 minutes

Why?

- Why does CWI:SEN1 invest in a meta-programming language?
- Why does UvA, OU, et al. teach it?



What?

- What is it from a bird's eye view
- What is it used for?

second 45 minutes

Metrics!

- Why build your own metrics?
- How to build your own metrics?
- S.W.O.T. discussion

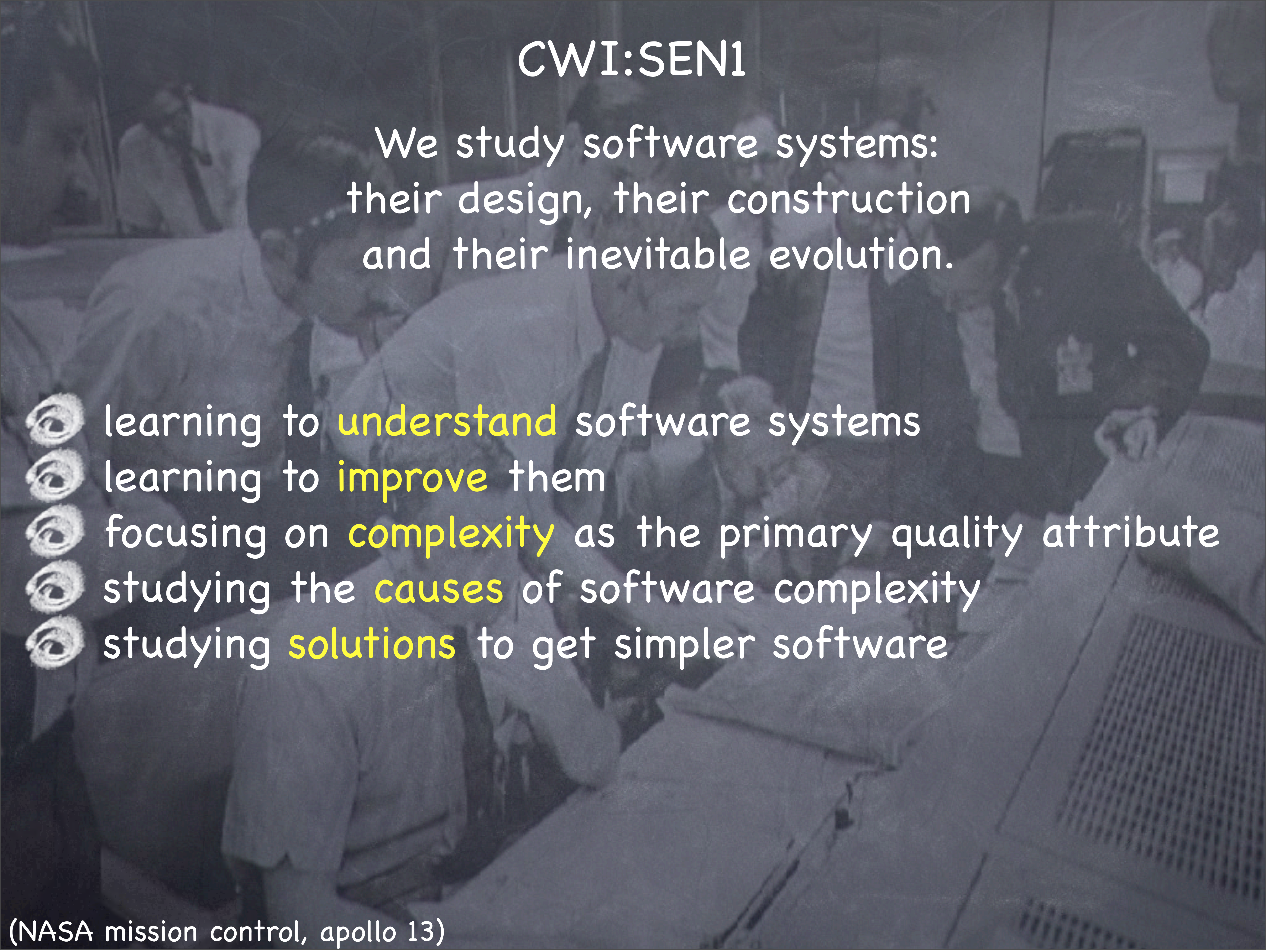
Refactorings!

- Why build your own refactorings?
- Example: change your design
- S.W.O.T. discussion

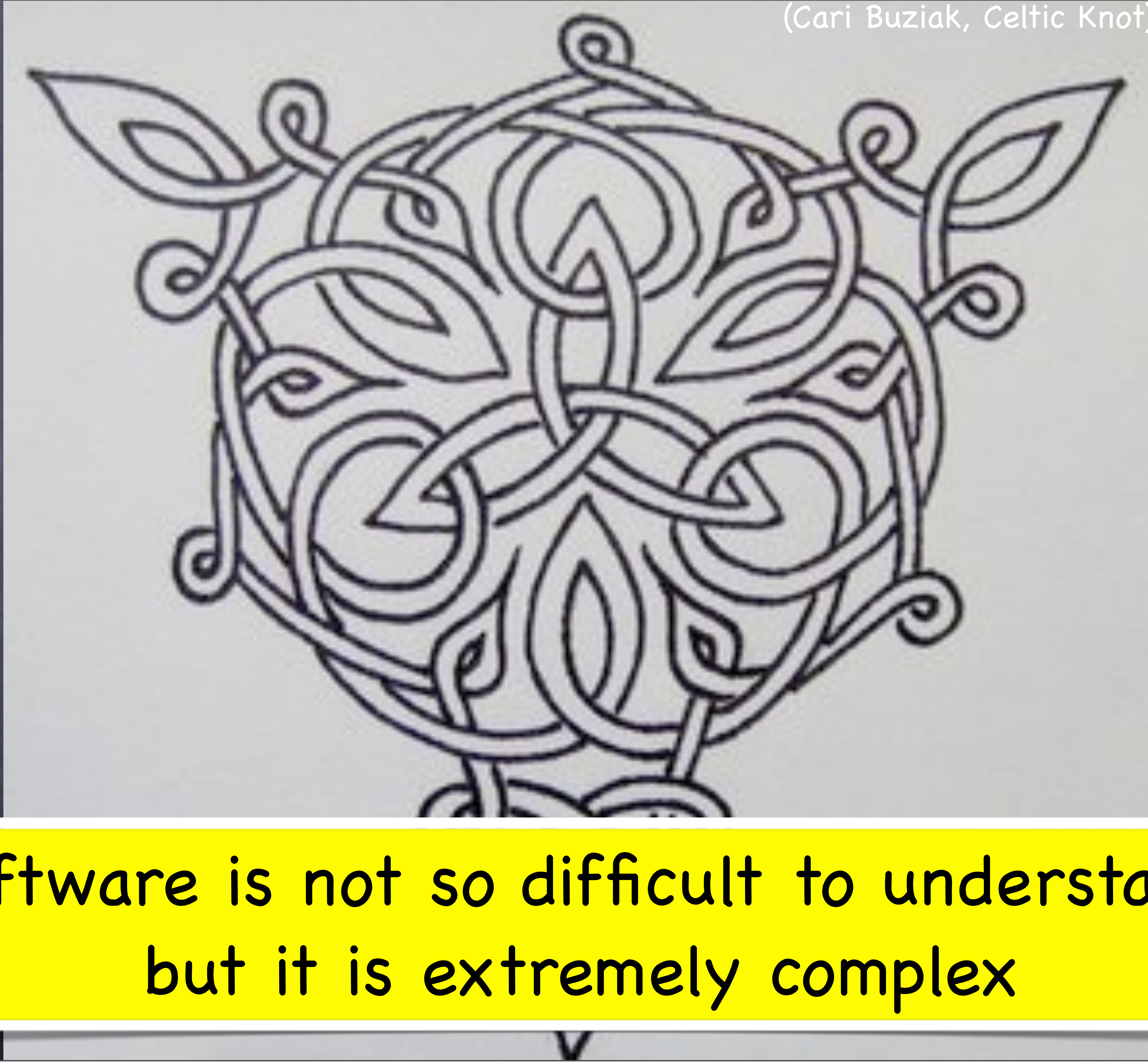
CWI:SEN1

We study software systems:
their design, their construction
and their inevitable evolution.

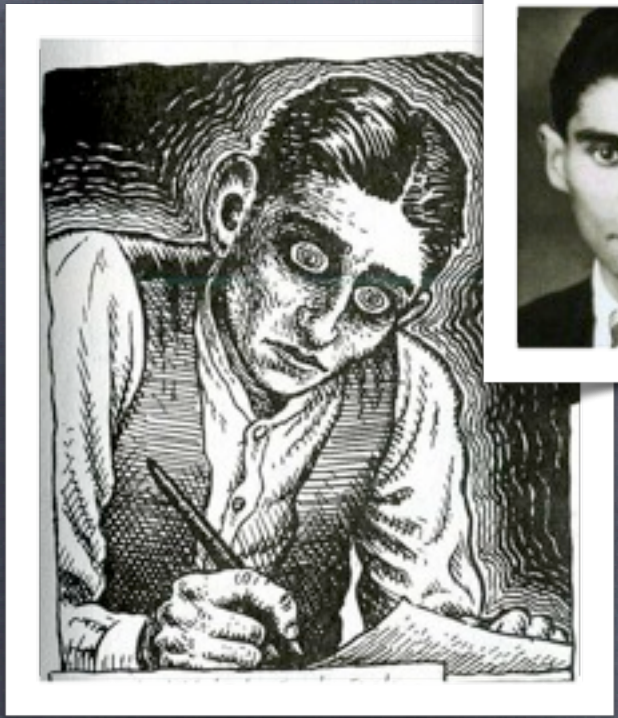
- learning to understand software systems
- learning to improve them
- focusing on complexity as the primary quality attribute
- studying the causes of software complexity
- studying solutions to get simpler software

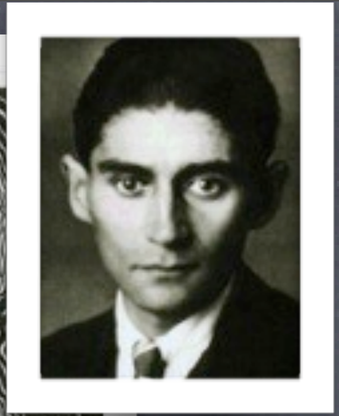


(NASA mission control, apollo 13)



Software is not so difficult to understand, but it is extremely complex





Kafkaesque

Software - large and complex structures of computer instructions, written and read by man, executed by computers

"marked by a senseless, disorienting, often menacing complexity..." (Infoplease.com)

The source code of "ls"

3894 lines

367 ifs

174 cases

Size does matter

- A normal Dutch company may own 3×10^{10} lines of code - 750,000,000 single column pages.
- It goes a few times around the globe, if printed.
- At 1 minute per page (?) that might take approximately 1427 years to read.
- Ergo, nobody has ever understood it, or will ever fully understand it.
- What a about 1M, 100k, 50k? Easy?

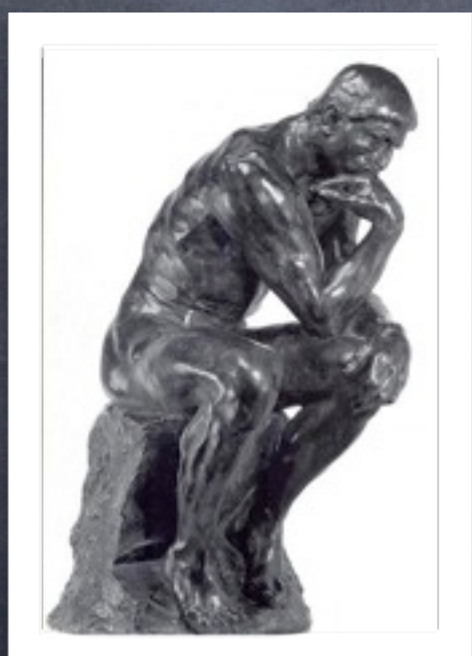

It's all about
understanding

It's all about **understanding**

- Maintenance represents the major part (50% - 80%) of the **cost of ownership** of Software
- Understanding takes much more time than editing
- So we should:
    - make simpler code
    - make understanding code easier

Size + Complexity -> Tools

Tool



Research

Application

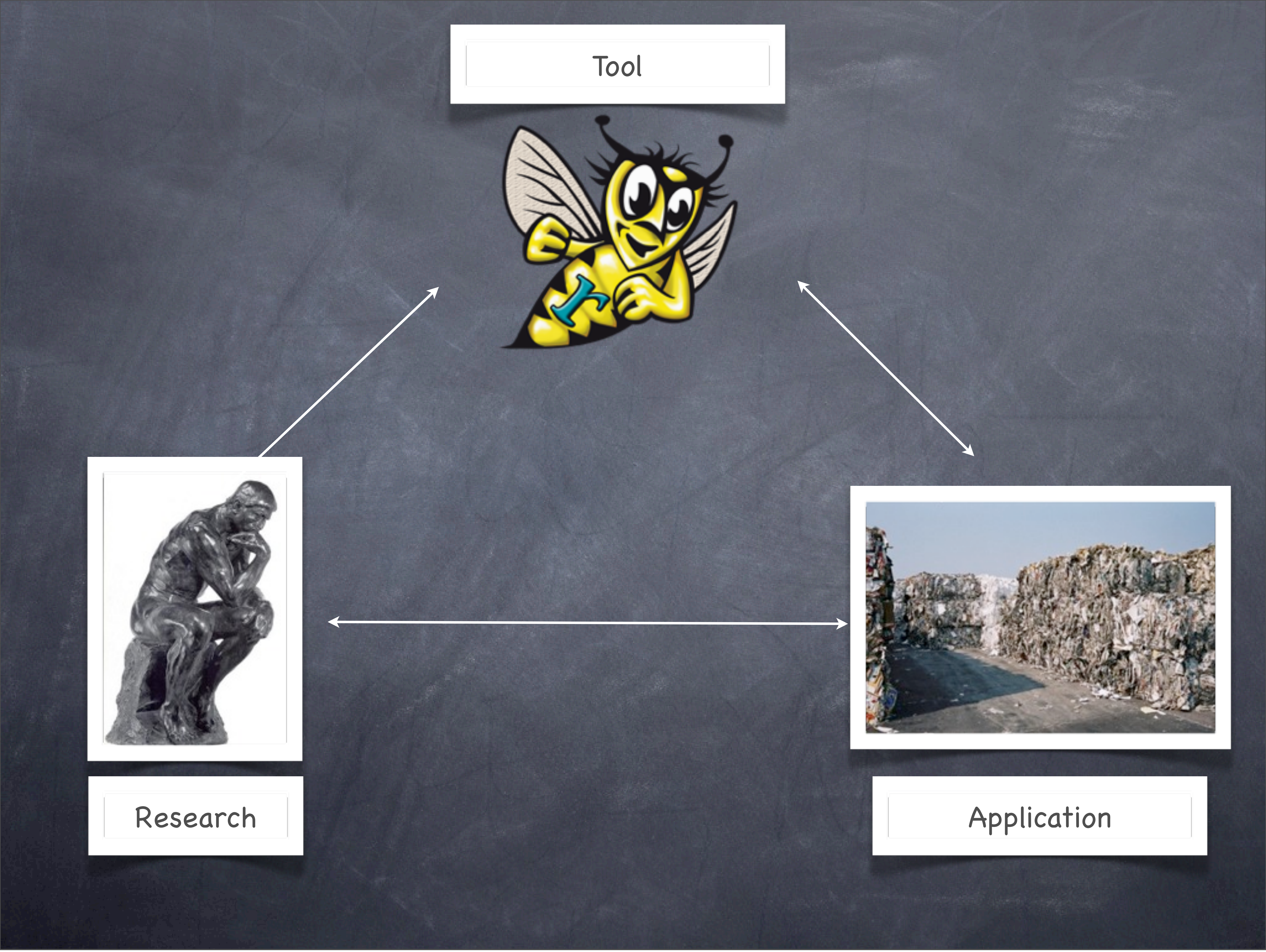

Sharpie
3m10FT
POWER TAPE


Which tools are used at Sogyo?
Sharpie

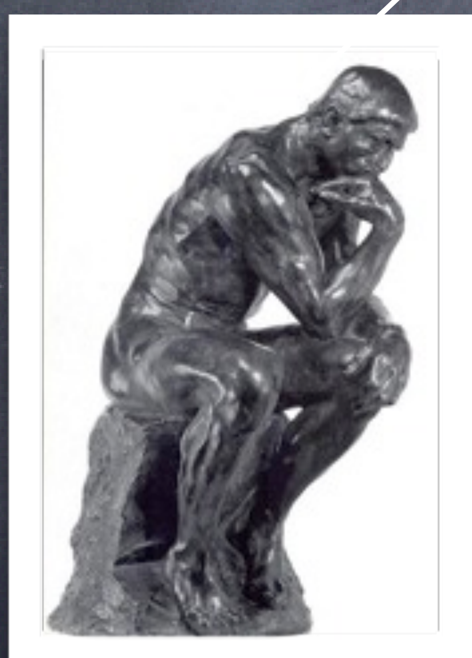
Tool

Research

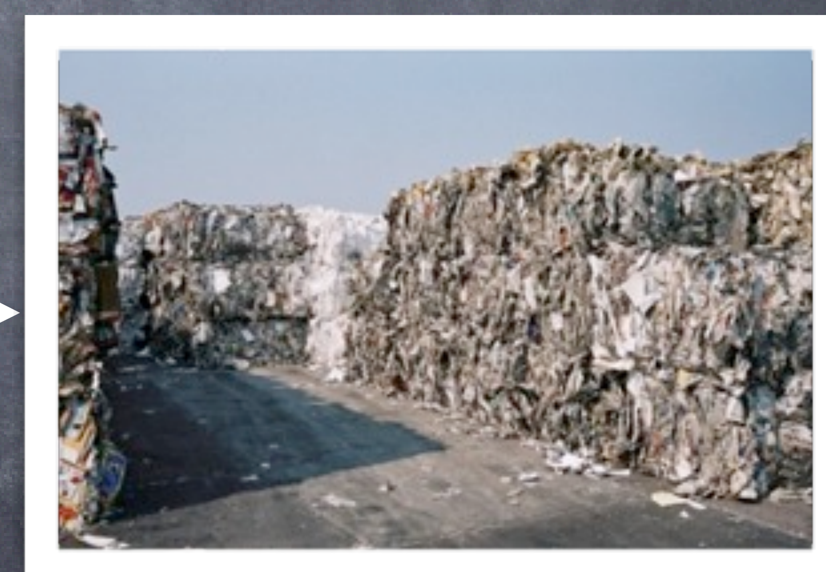
Application


Tool
Research
Application

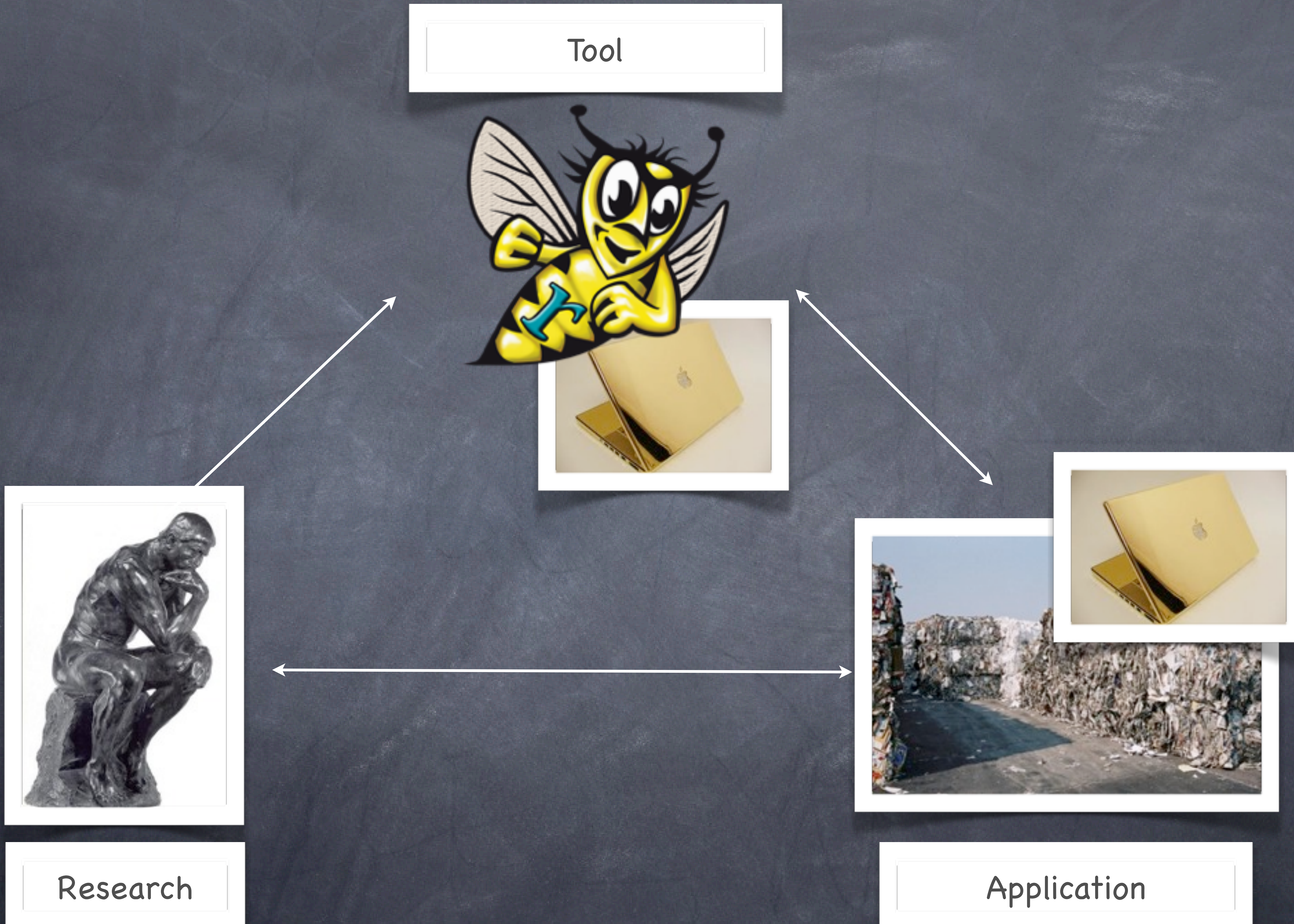
Tool



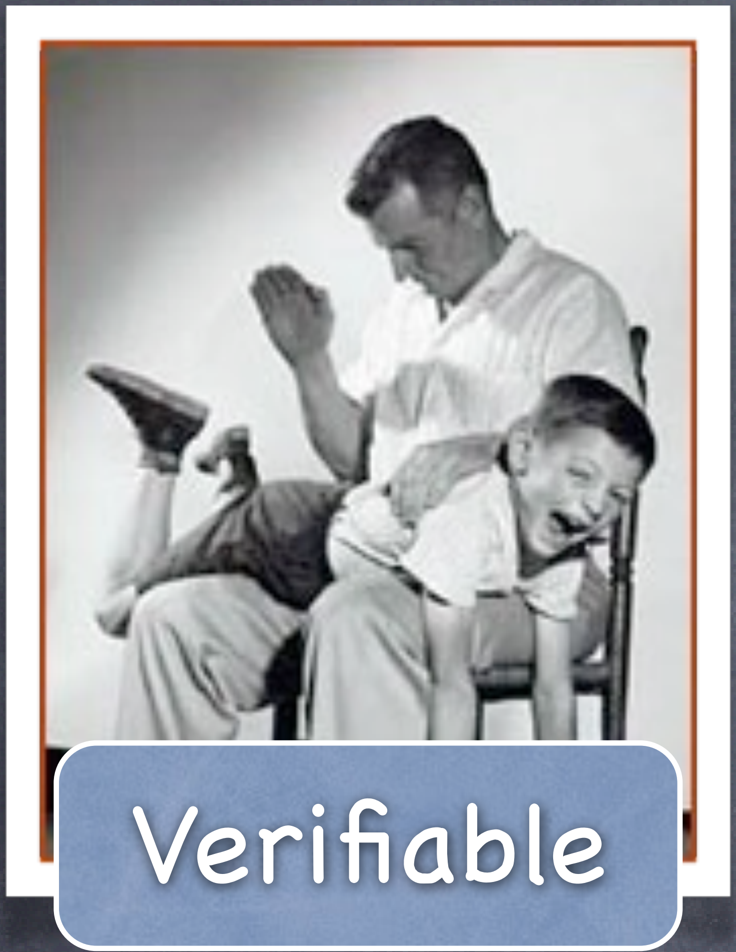
Research

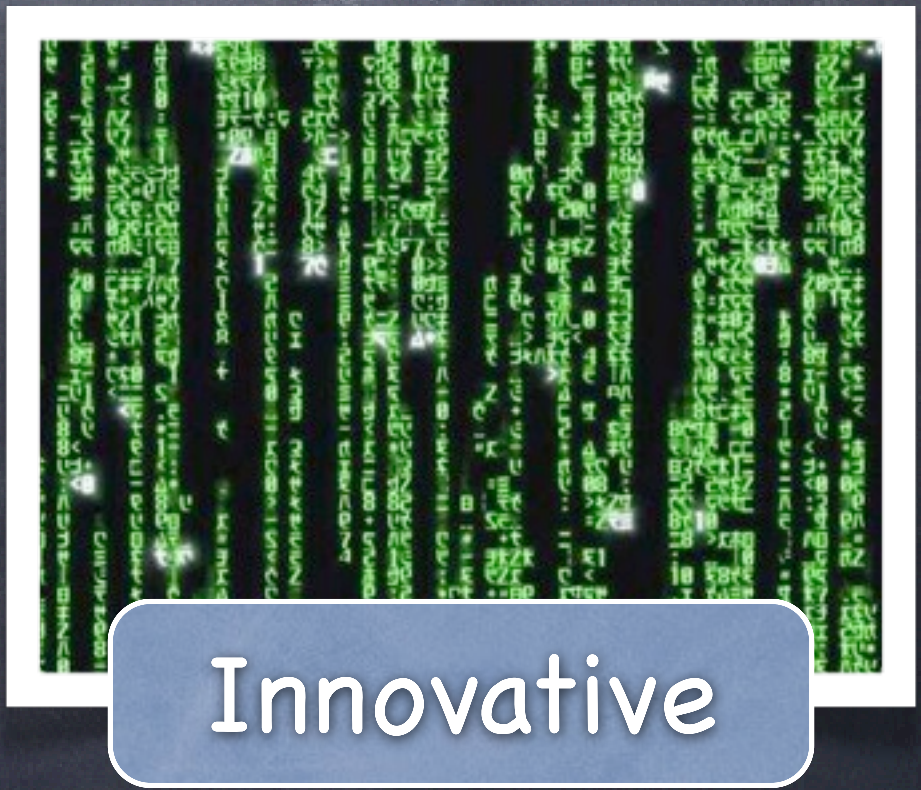


Application


Tool
Research
Application

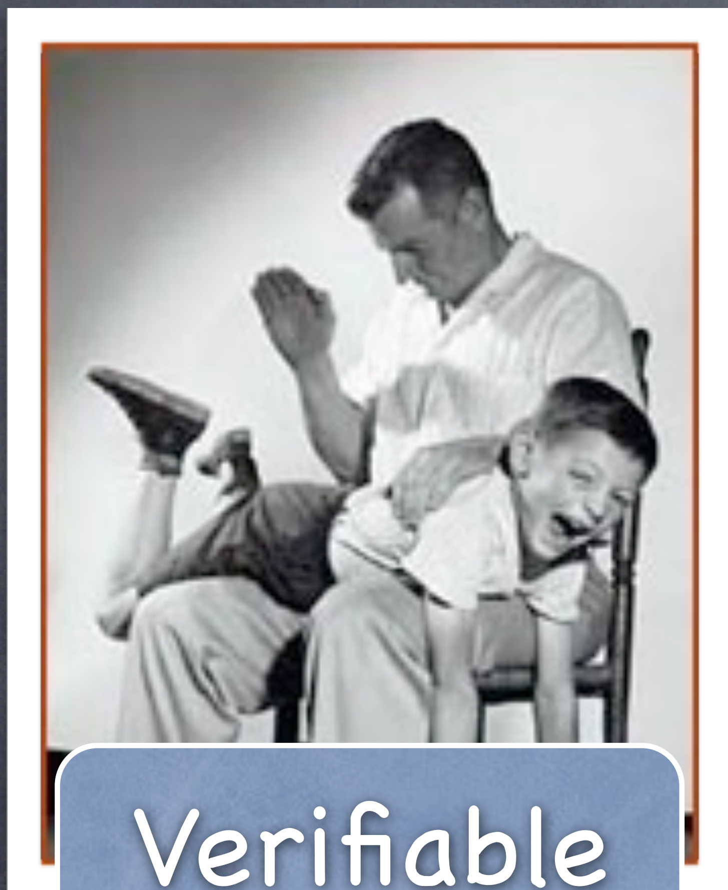
(open-source) tools are for improved research methods and improved transfer to society



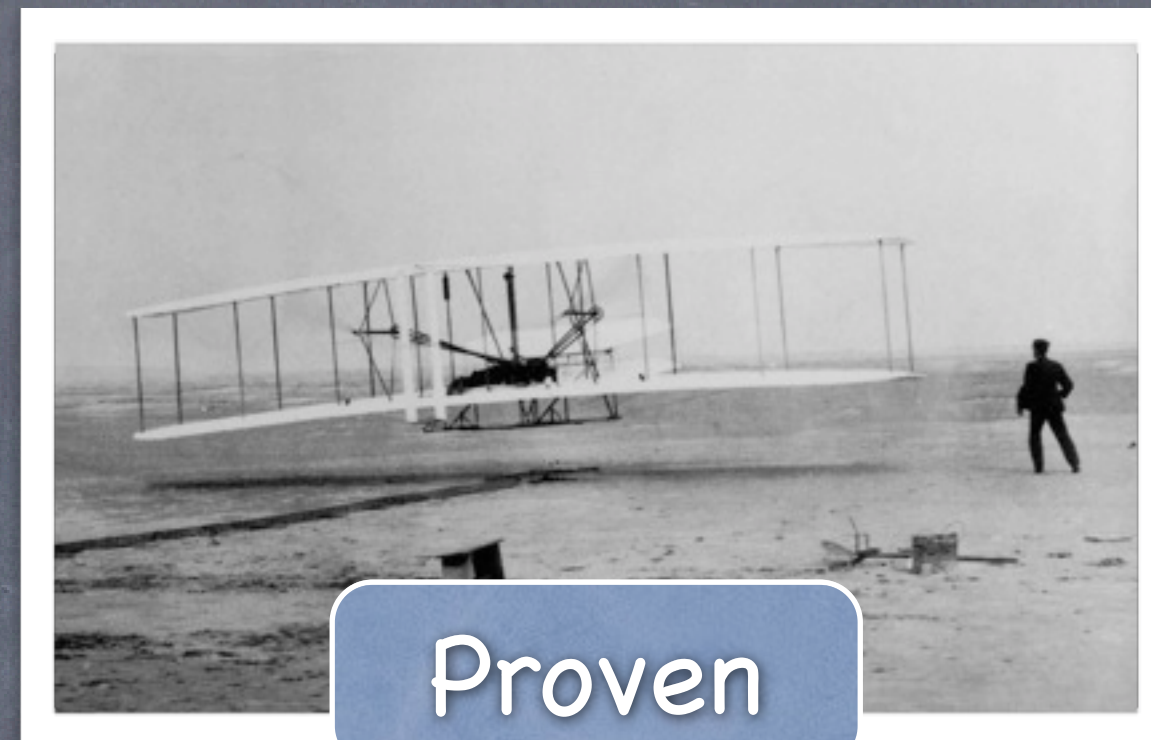


(open-source) tools are for improved research methods and improved

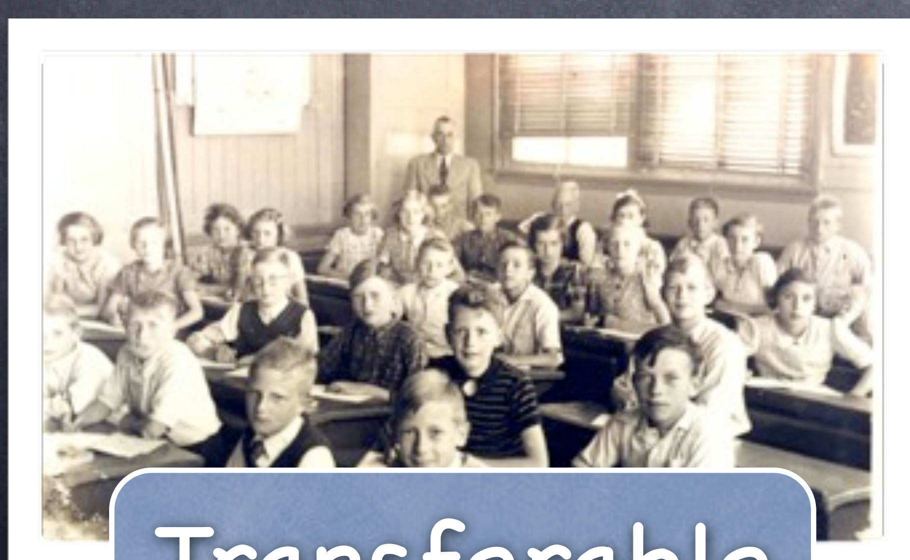
valorisation



Verifiable

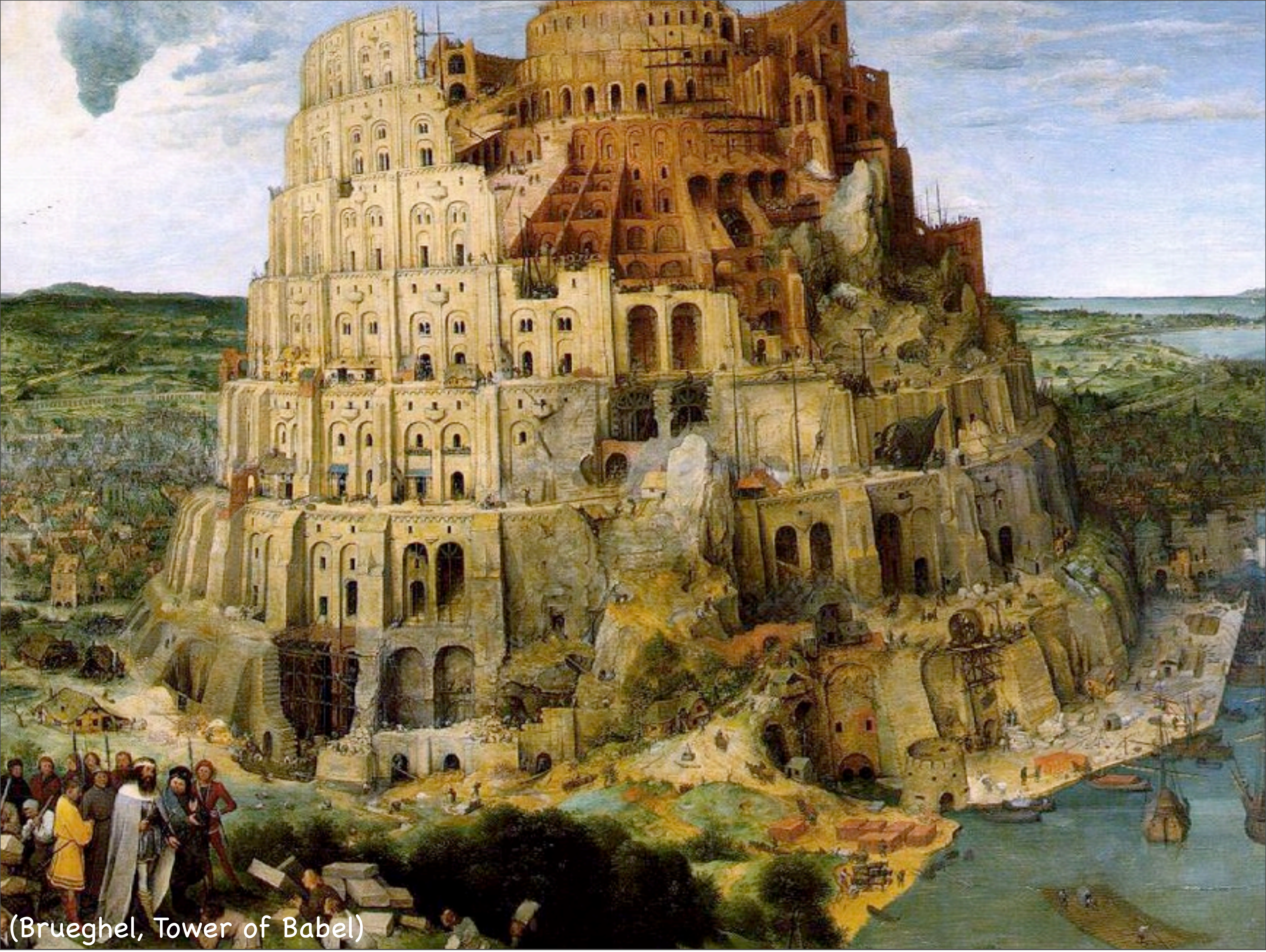


Proven



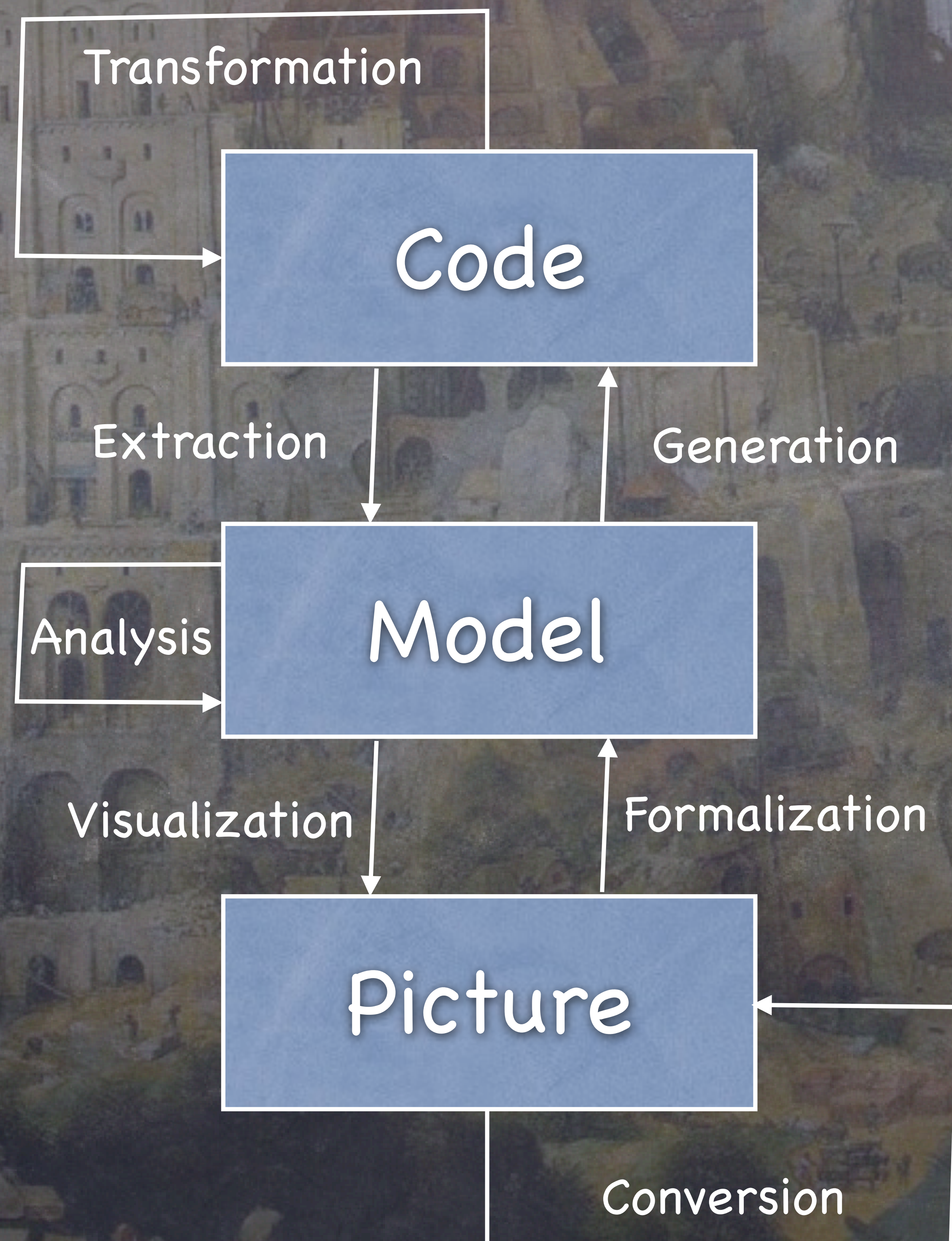
Transferable

Innovative



(Brueghel, Tower of Babel)

Meta Software



(Brueghel, Tower of Babel)

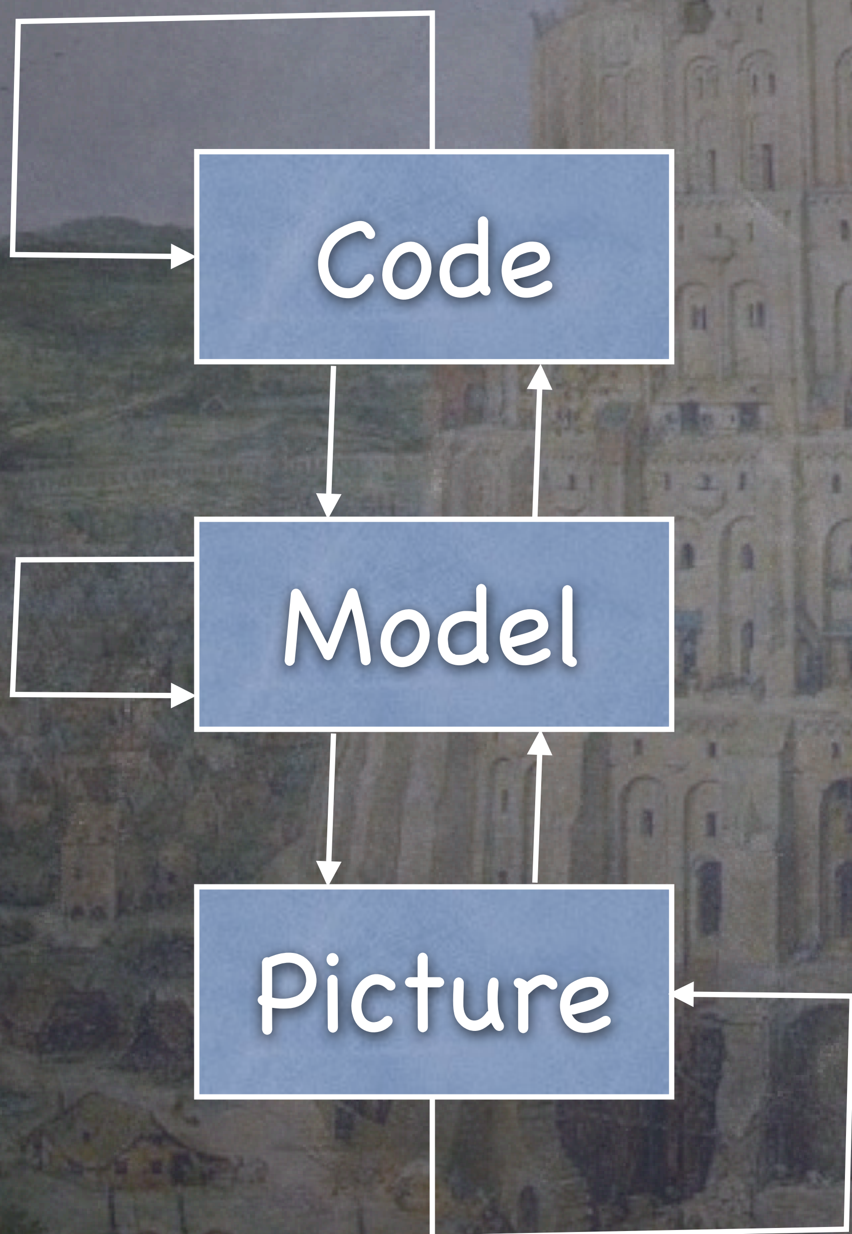
Analysis

- Dead code detection
- **Dependence analysis**
- Impact analysis
- Clustering
- Architecture recovery
- Code-to-model
- **Maintainability analysis**
- ...

Transformation

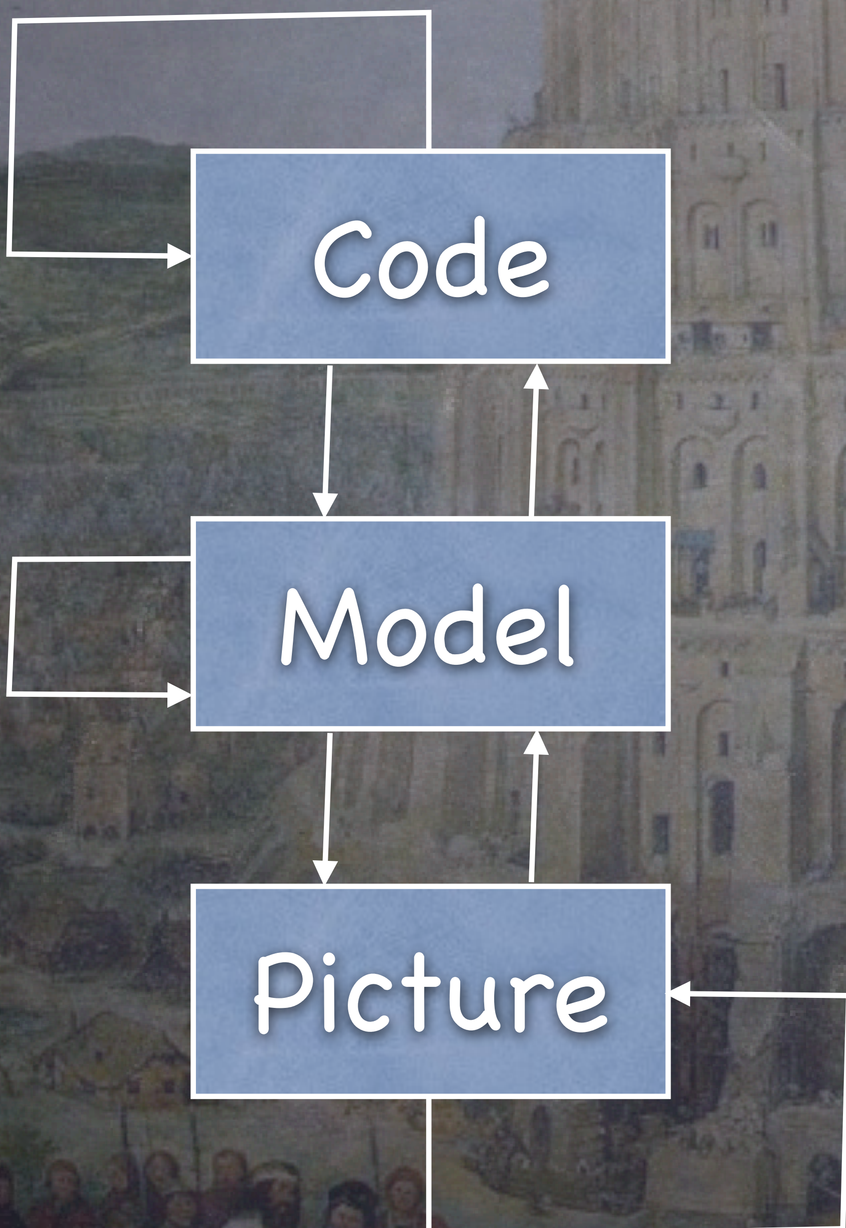
- Goto elimination
- Dialect transformation
- Aspect weaving
- **DSL** compilers
- **API migration**
- **Refactoring** design
- ...

3 Meta Software Challenges



1: Diversity

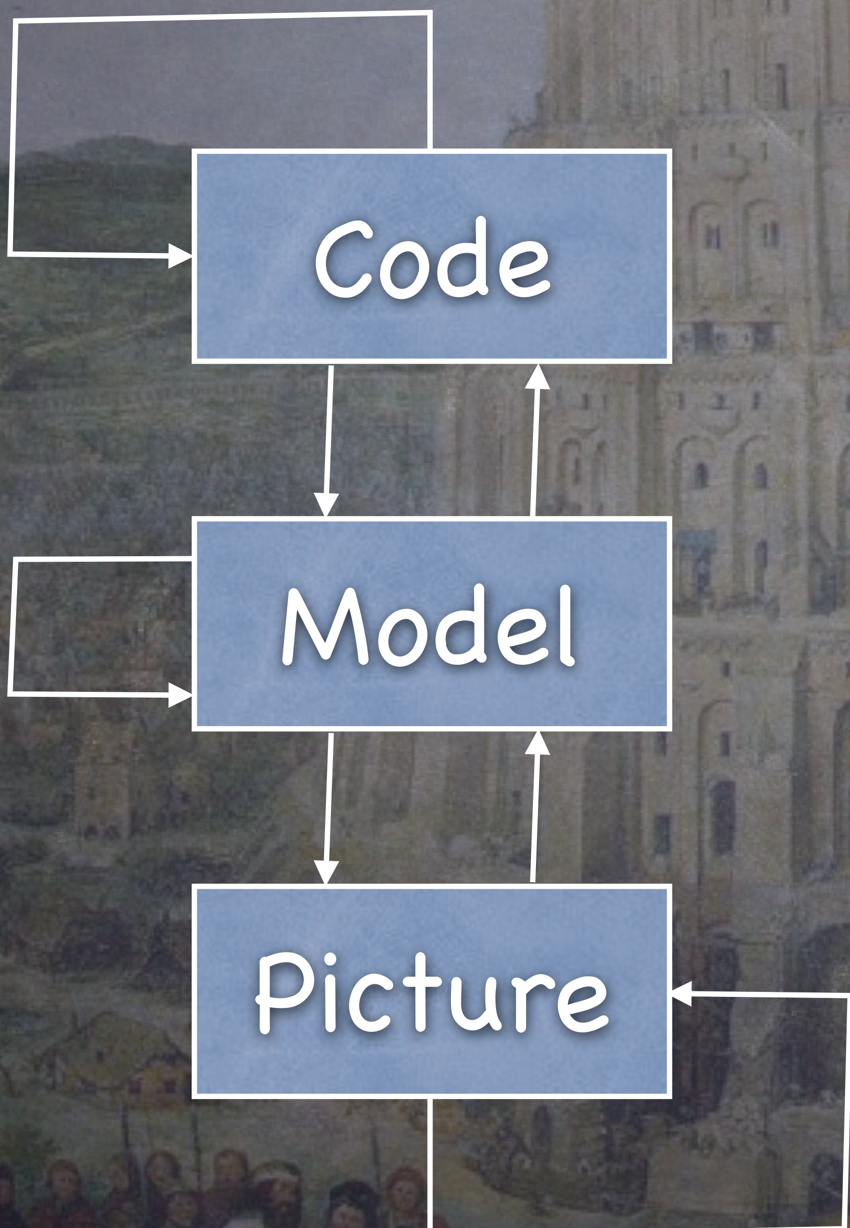
3 Meta Software Challenges

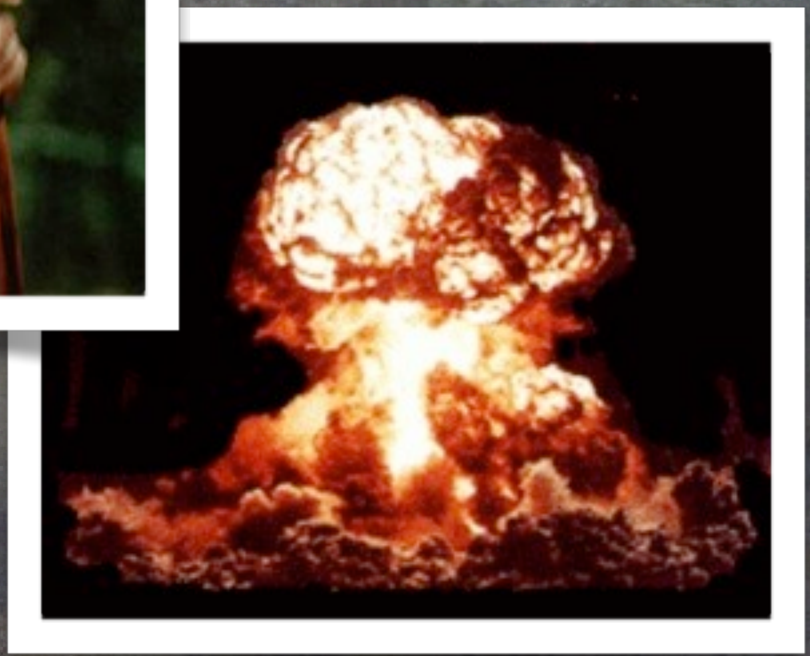


(Raphael, Parnassus)

2:Multi-disciplinary

3 Meta Software Challenges





3: Precision

vs.

Efficiency

Ingredients

Ingredients

Familiar notation

IDE integration

Interactive Documentation

Key enablers

Ingredients

Integration to tackle multi-disciplinary nature

Term Rewriting

Relational Calculus

Syntax definition

Familiar notation

IDE integration

Interactive Documentation

Key enablers

Ingredients

Language parametric

Generic programming

Modularity

Programming techniques for dealing with diversity and scale

Integration to tackle multi-disciplinary nature

Term Rewriting

Relational Calculus

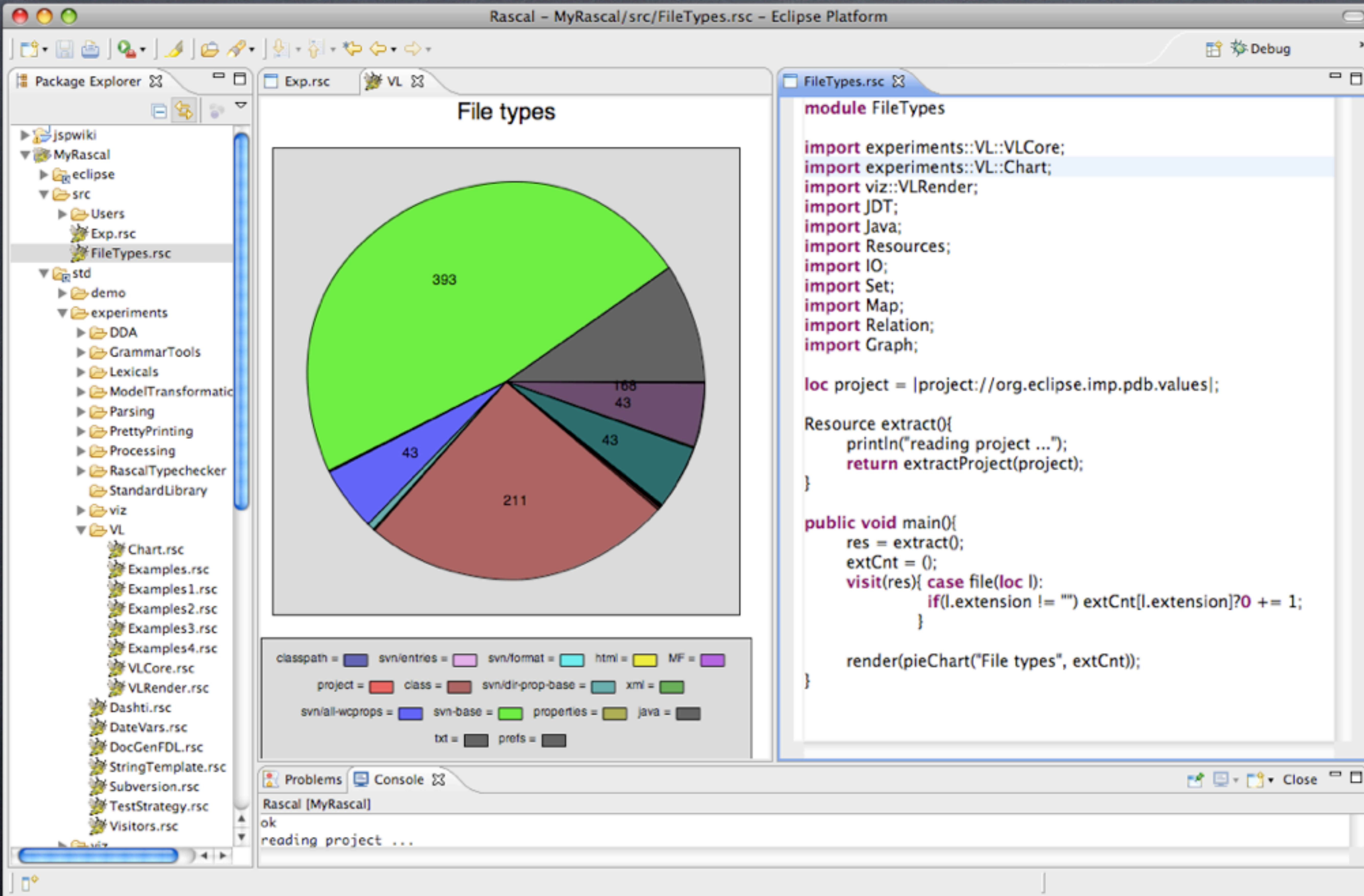
Syntax definition

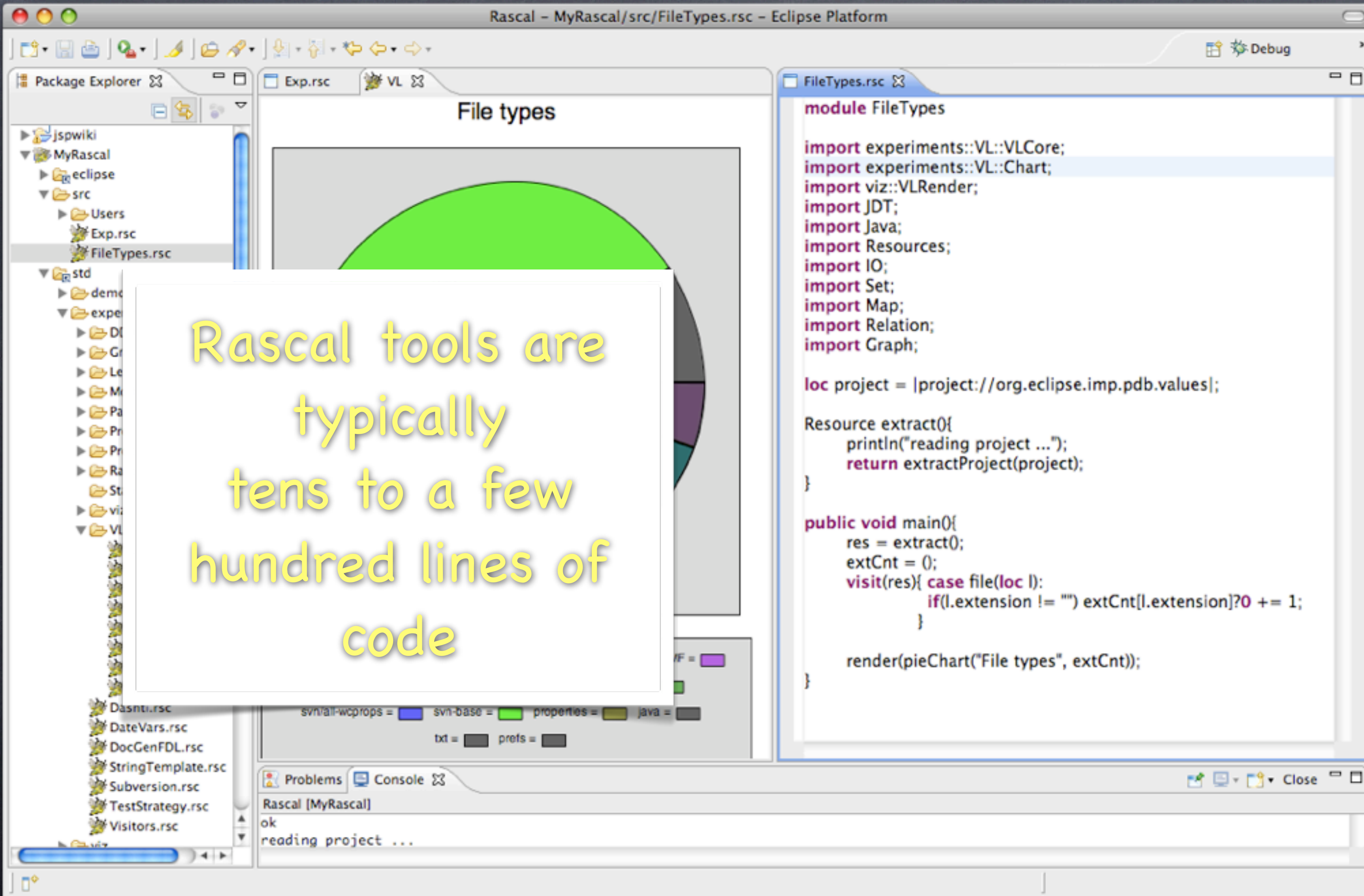
Familiar notation

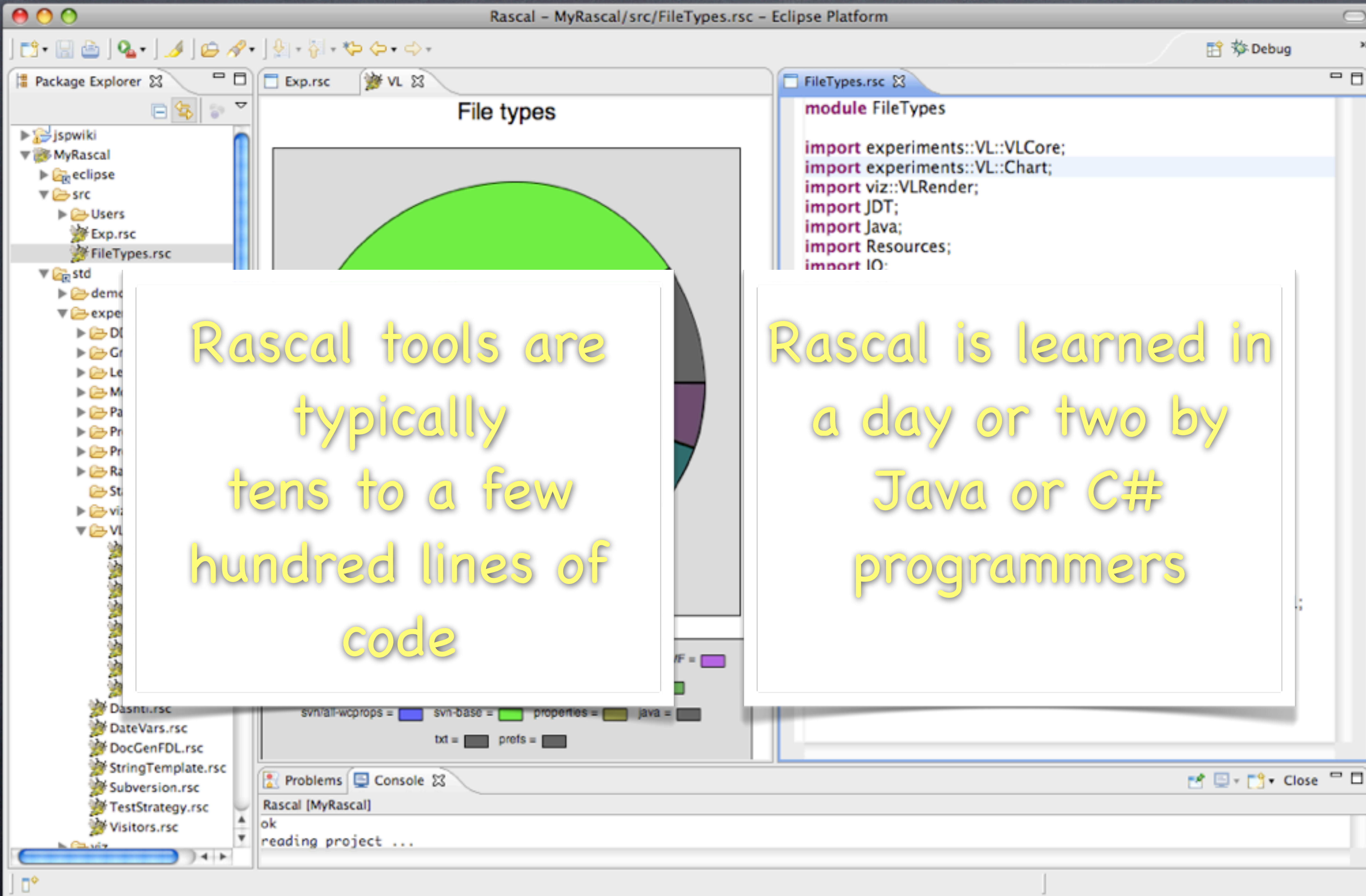
IDE integration

Interactive Documentation

Key enablers


Rascal - MyRascal/src/FileTypes.rsc - Eclipse Platform
Debug
Package Explorer
jspwiki
MyRascal
eclipse
src
Users
Exp.rsc
FileTypes.rsc
std
demo
experiments
DDA
GrammarTools
Lexicals
ModelTransformatic
Parsing
PrettyPrinting
Processing
RascalTypechecker
StandardLibrary
viz
VL
Chart.rsc
Examples.rsc
Examples1.rsc
Examples2.rsc
Examples3.rsc
Examples4.rsc
VLCore.rsc
VLRender.rsc
Dashti.rsc
DateVars.rsc
DocGenFDL.rsc
StringTemplate.rsc
Subversion.rsc
TestStrategy.rsc
Visitors.rsc
Exp.rsc
VL
File types
393
43
43
43
211
classpath = svn/entries = svn/format = html = MF =
project = class = svn/dir-prop-base = xml =
svn/all-wcprops = svn-base = properties = java =
txt = prefs =
FileTypes.rsc
module FileTypes
import experiments::VL::VLCore;
import experiments::VL::Chart;
import viz::VLRender;
import JDT;
import Java;
import Resources;
import IO;
import Set;
import Map;
import Relation;
import Graph;
loc project = |project://org.eclipse.imp.pdb.values|;
Resource extract(){
println("reading project ...");
return extractProject(project);
}
public void main(){
res = extract();
extCnt = ();
visit(res){ case file(loc l):
if(l.extension != "") extCnt[l.extension]?0 += 1;
}
render(pieChart("File types", extCnt));
}
Problems
Console
Close
Rascal [MyRascal]
ok
reading project ...


Rascal - MyRascal/src/FileTypes.rsc - Eclipse Platform
Debug
Package Explorer
jspwiki
MyRascal
eclipse
src
Users
Exp.rsc
FileTypes.rsc
std
DateVars.rsc
DocGenFDL.rsc
StringTemplate.rsc
Subversion.rsc
TestStrategy.rsc
Visitors.rsc
Exp.rsc
VL
File types
svn/all-wcprops =
svn-base =
properties =
java =
txt =
prefs =
FileTypes.rsc
module FileTypes
import experiments::VL::VLCore;
import experiments::VL::Chart;
import viz::VLRender;
import JDT;
import Java;
import Resources;
import IO;
import Set;
import Map;
import Relation;
import Graph;
loc project = |project://org.eclipse.imp.pdb.values|;
Resource extract(){
println("reading project ...");
return extractProject(project);
}
public void main(){
res = extract();
extCnt = ();
visit(res){ case file(loc l):
if(l.extension != "") extCnt[l.extension]?0 += 1;
}
render(pieChart("File types", extCnt));
}
Problems
Console
Close
Rascal [MyRascal]
ok
reading project ...
Rascal tools are typically tens to a few hundred lines of code



Rascal tools are typically tens to a few hundred lines of code

Rascal is learned in a day or two by Java or C# programmers

So why Rascal?

Because we want to understand software and make it simpler and we need tools for that, that deal with real software

So what is Rascal?

A programming language for manipulating source code and its derivates, for quickly building software analysis, transformation, generation, visualization tools

So what is Rascal?

A programming language for manipulating source code and its derivates, for quickly building software analysis, transformation, generation, visualization tools

Questions & Coffee!

second 45 minutes

Metrics!

- Why build your own metrics?
- How to build your own metrics?
- S.W.O.T. discussion

Refactorings!

- Why build your own refactorings?
- Example: change your design
- S.W.O.T. discussion

There is no substitute for thinking

- By design, Rascal does not think for you
- It is a tool itself that helps you “play” with software
- So, what is a good metric?
- So, when do you build a refactoring?

Software metrics

- Why?
- How?
- Pitfalls?

Why metrics?

"Numbers tell the tale"

Why metrics?

"Numbers tell the tale"

- Observing: progress, hot-spots, "quality"

Why metrics?

"Numbers tell the tale"

- Observing: progress, hot-spots, "quality"
- Relating: cost, deployment, process

Why metrics?

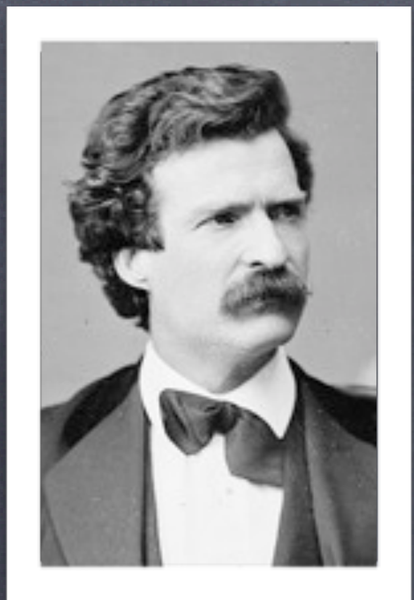
"Numbers tell the tale"

- Observing: progress, hot-spots, "quality"
- Relating: cost, deployment, process
- Predicting: bugs, tests, maintenance cost

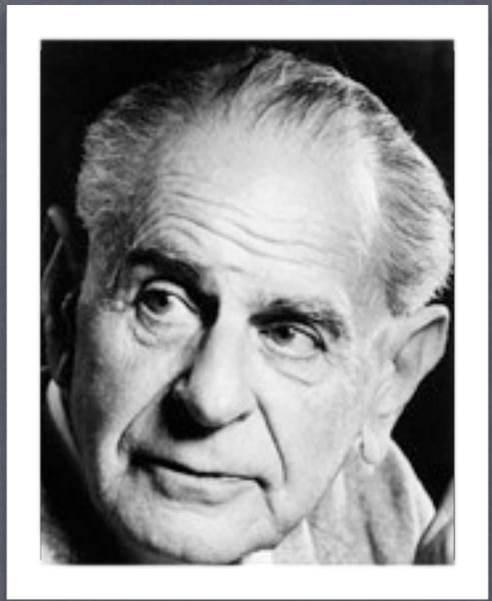
Why metrics?

"Numbers tell the tale"

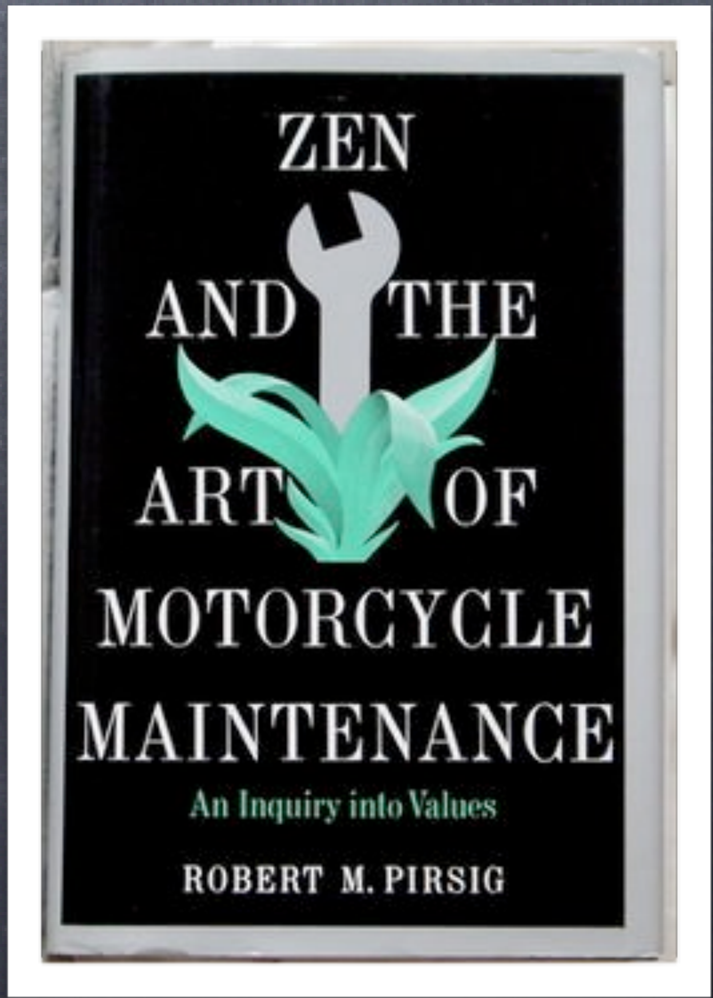
- Observing: progress, hot-spots, "quality"
- Relating: cost, deployment, process
- Predicting: bugs, tests, maintenance cost
- Strategic: business "intelligence"



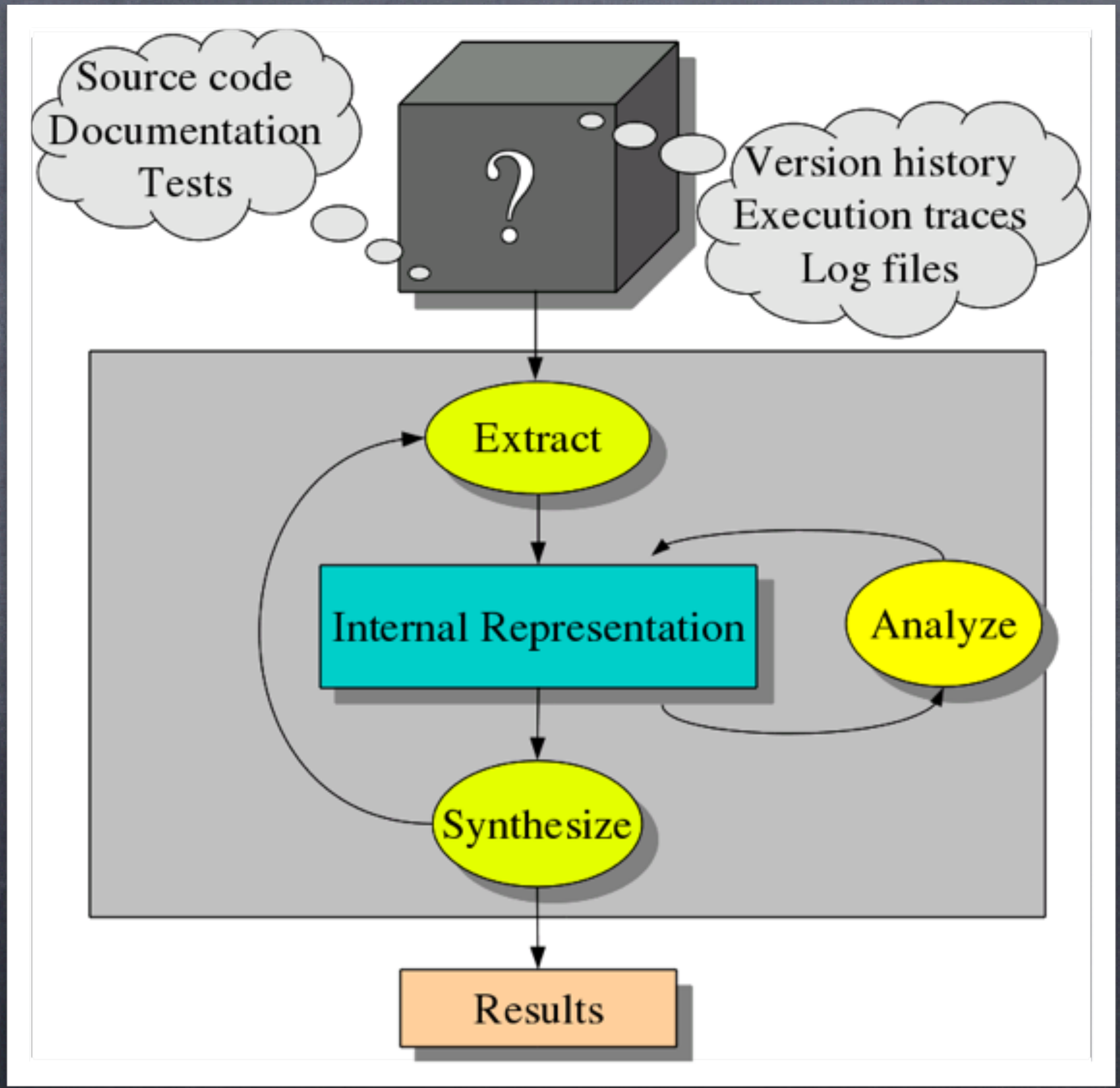
Why not metrics?



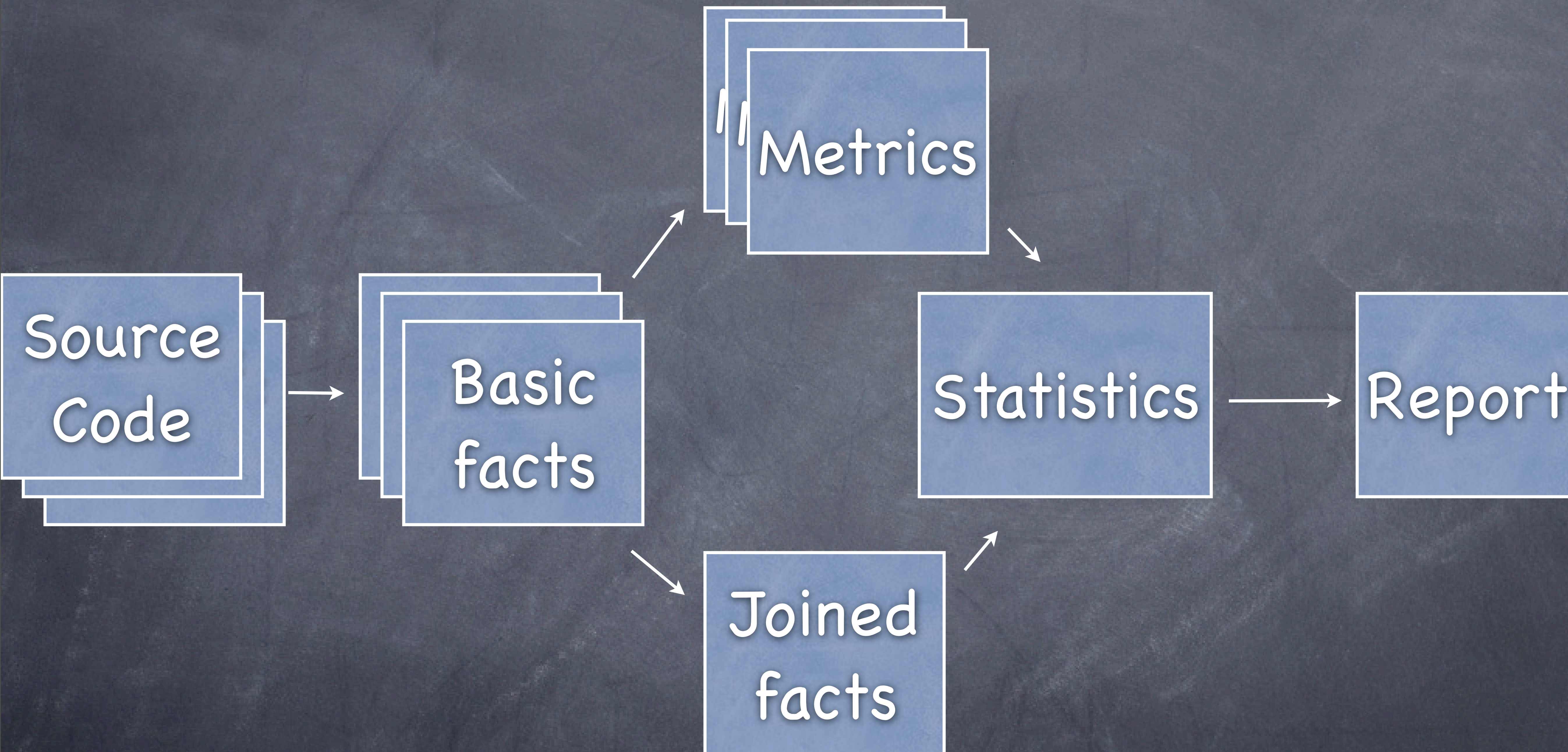
- What are we measuring anyway? The metric...
- Is quality measurable?
- "Lies, damn lies, and statistics"
- Is aggregation sound?
- Value judgments?
- Tunnelvision alert!



How to measure

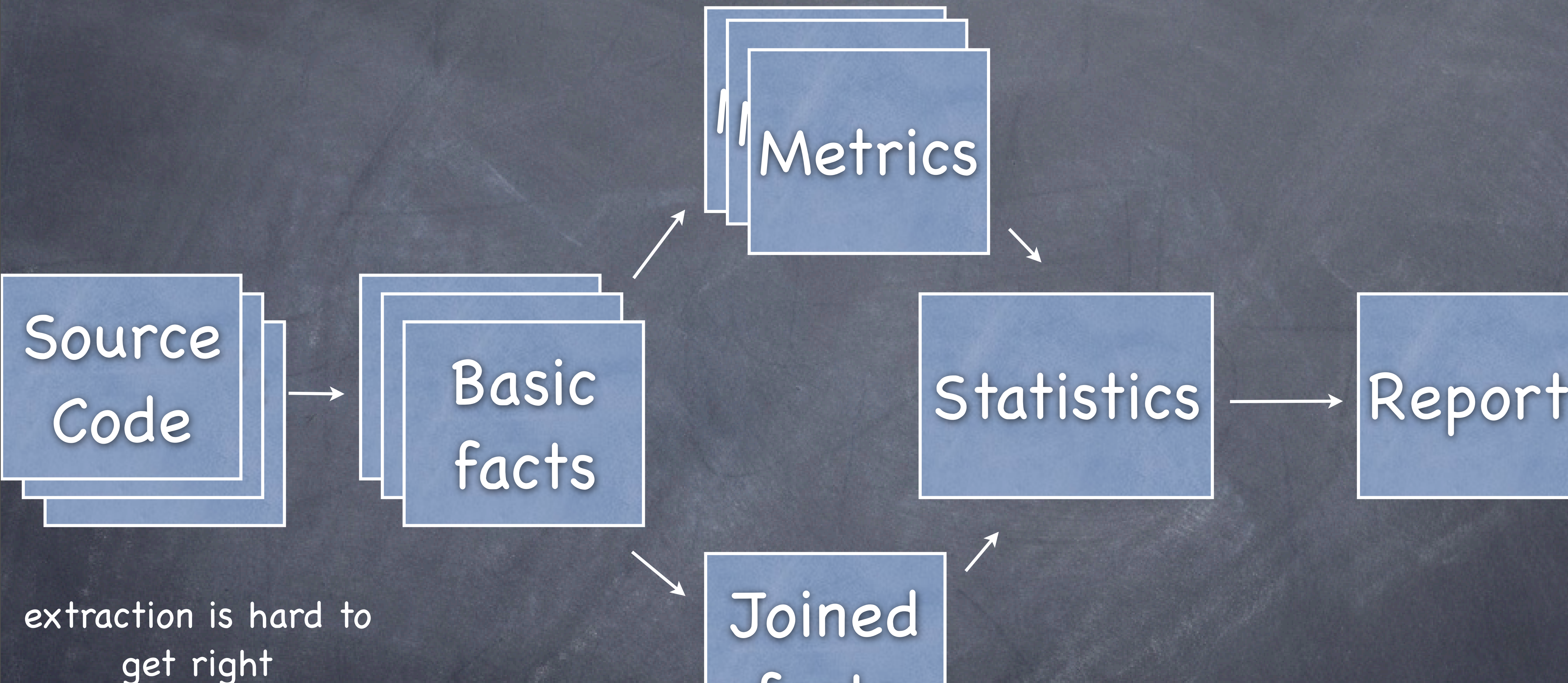


How to measure



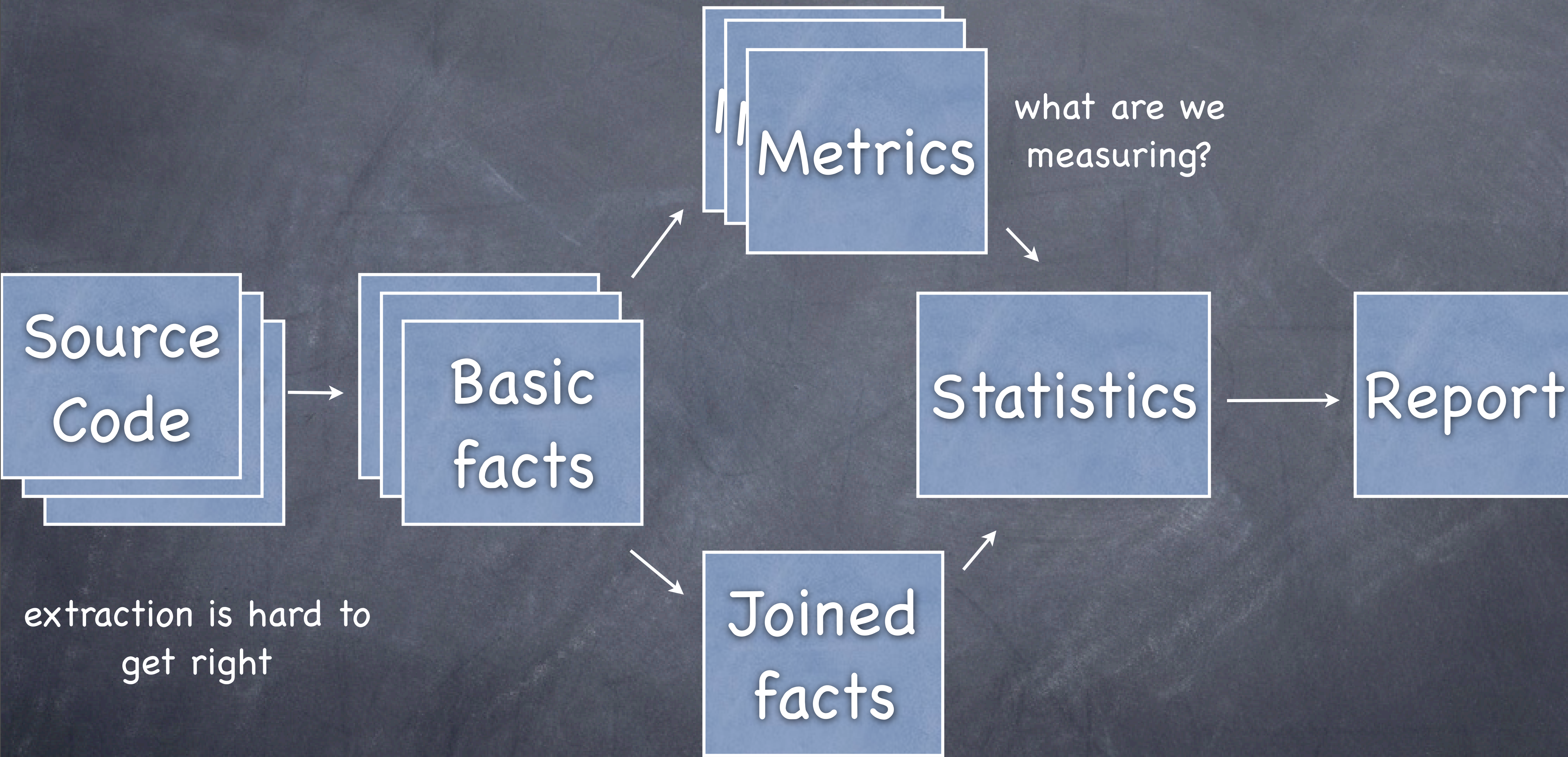
Rascal = "one-stop-shop": all in 20 lines

How to measure



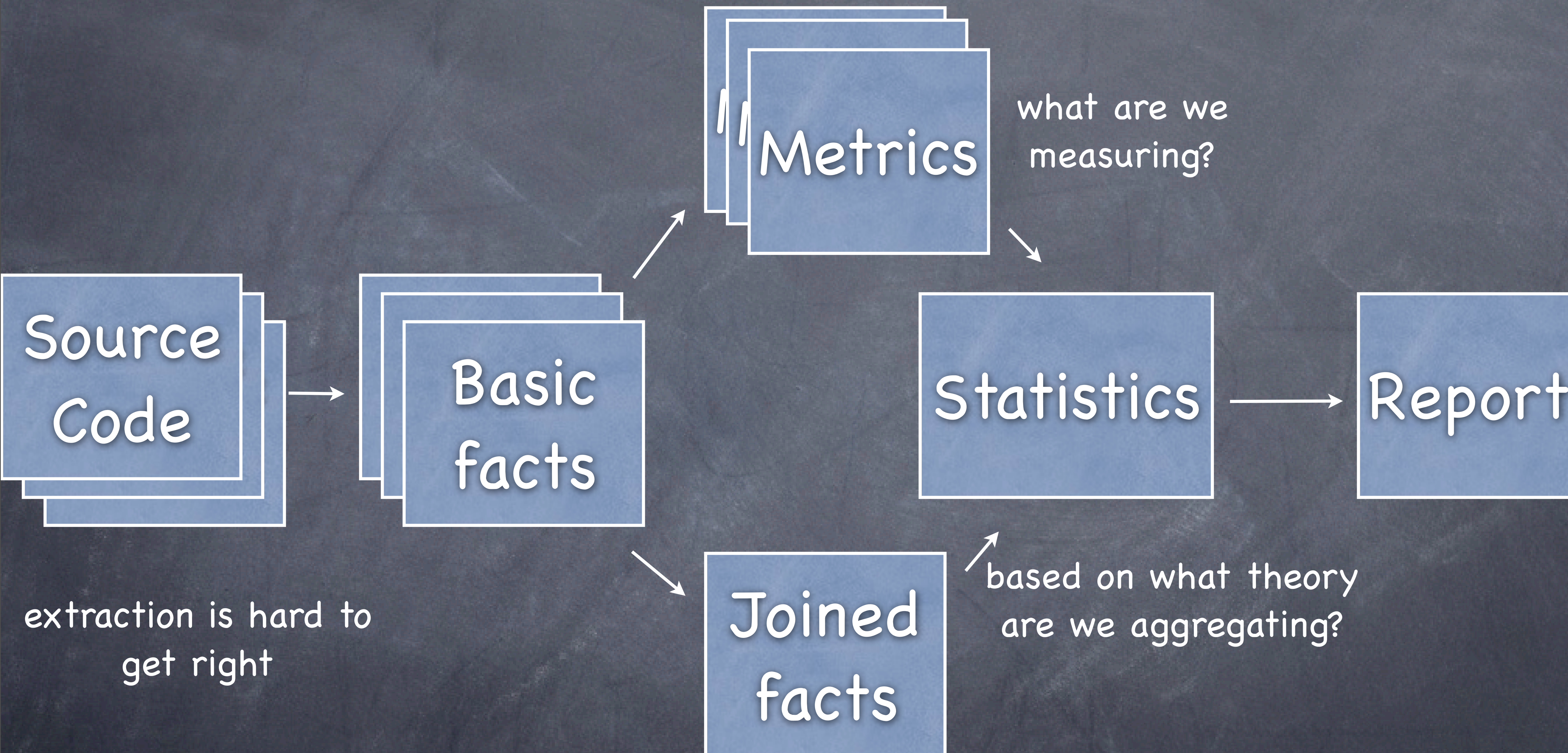
Rascal = “one-stop-shop”: all in 20 lines

How to measure



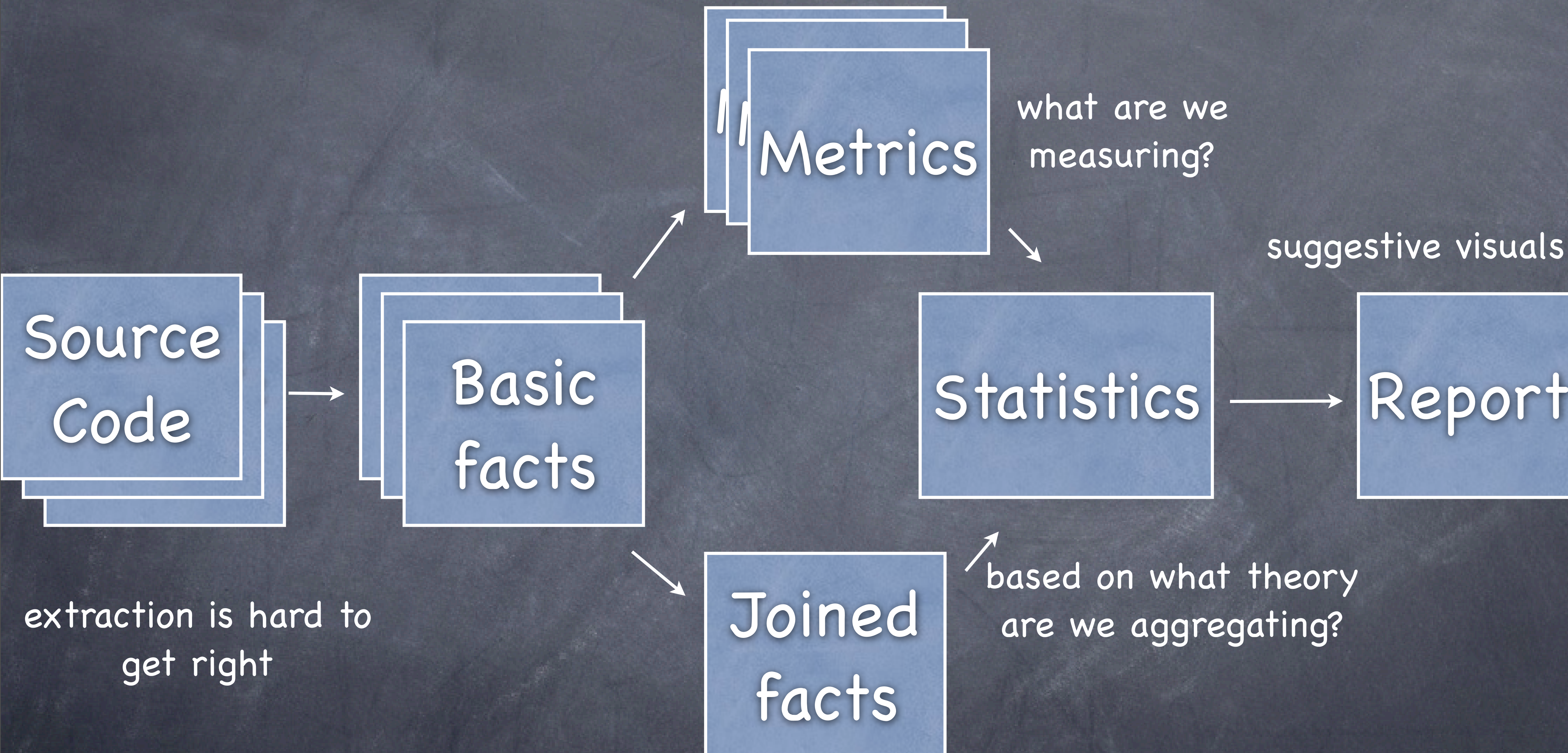
Rascal = "one-stop-shop": all in 20 lines

How to measure



Rascal = "one-stop-shop": all in 20 lines

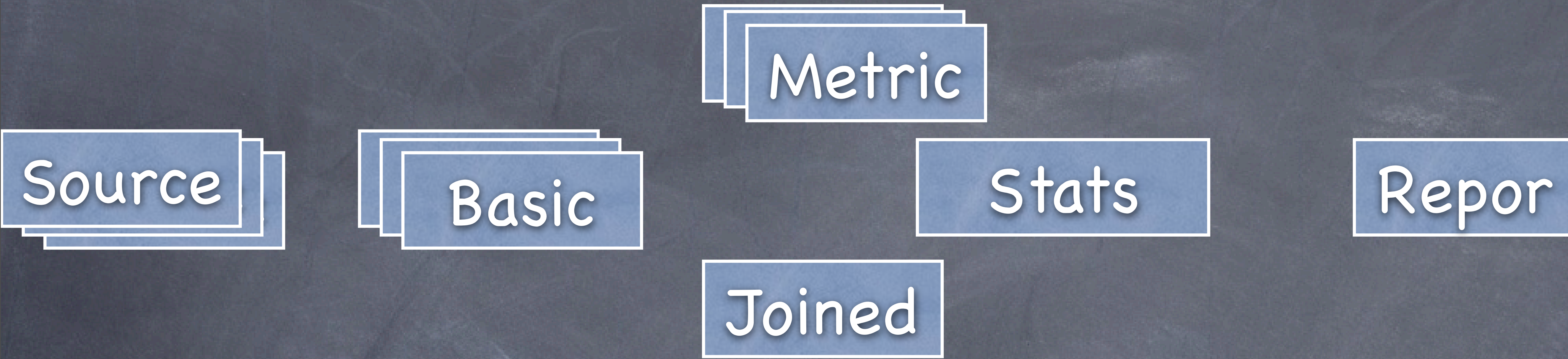
How to measure



Rascal = “one-stop-shop”: all in 20 lines

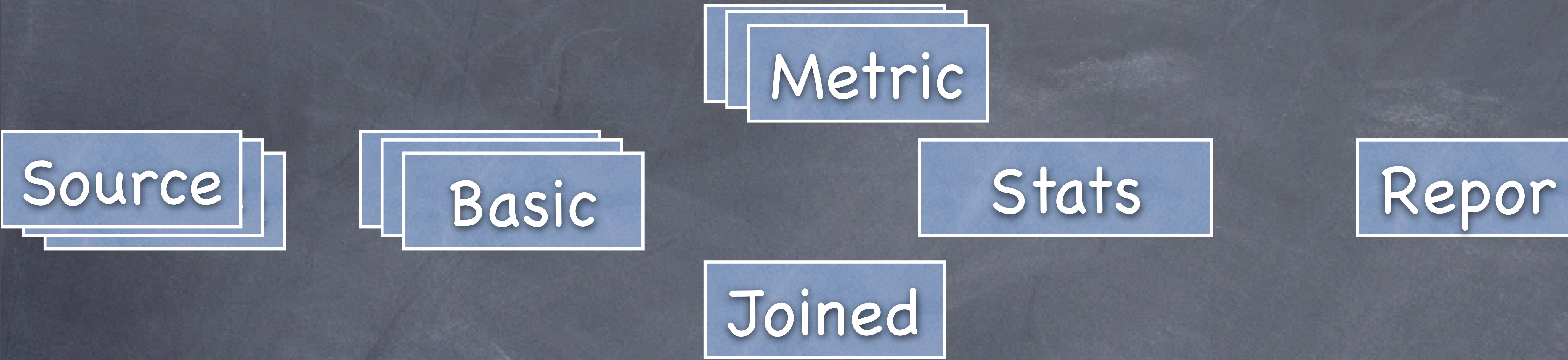
Example: Cyclomatic Complexity

- Code with lots of branches, loops and cases is hard to understand
- Every unique path should be tested and understood
- McCabe Cyclomatic Complexity is a measure for the number of unique control flow paths through a method/function/procedure
- High CC is bad, low CC is good



CC: basic facts

- `syntax CompilationUnit = “package” Id ...`
- `p = parse(#CompilationUnit, |file://myFile.java|)`
- a tree for every file in the system
- watch out for double files
- grammars are expensive animals
- regular expressions are tricky animals



CC: metric

- `(0 | it + 1 | /stat := p, isSplit(stat))`
- `isSplit(s) = s is if || s is while || s is case || s is …`
- Easy… but no two tools do the same!
- What is in Java, C#, PHP a control flow split?

CC: stats

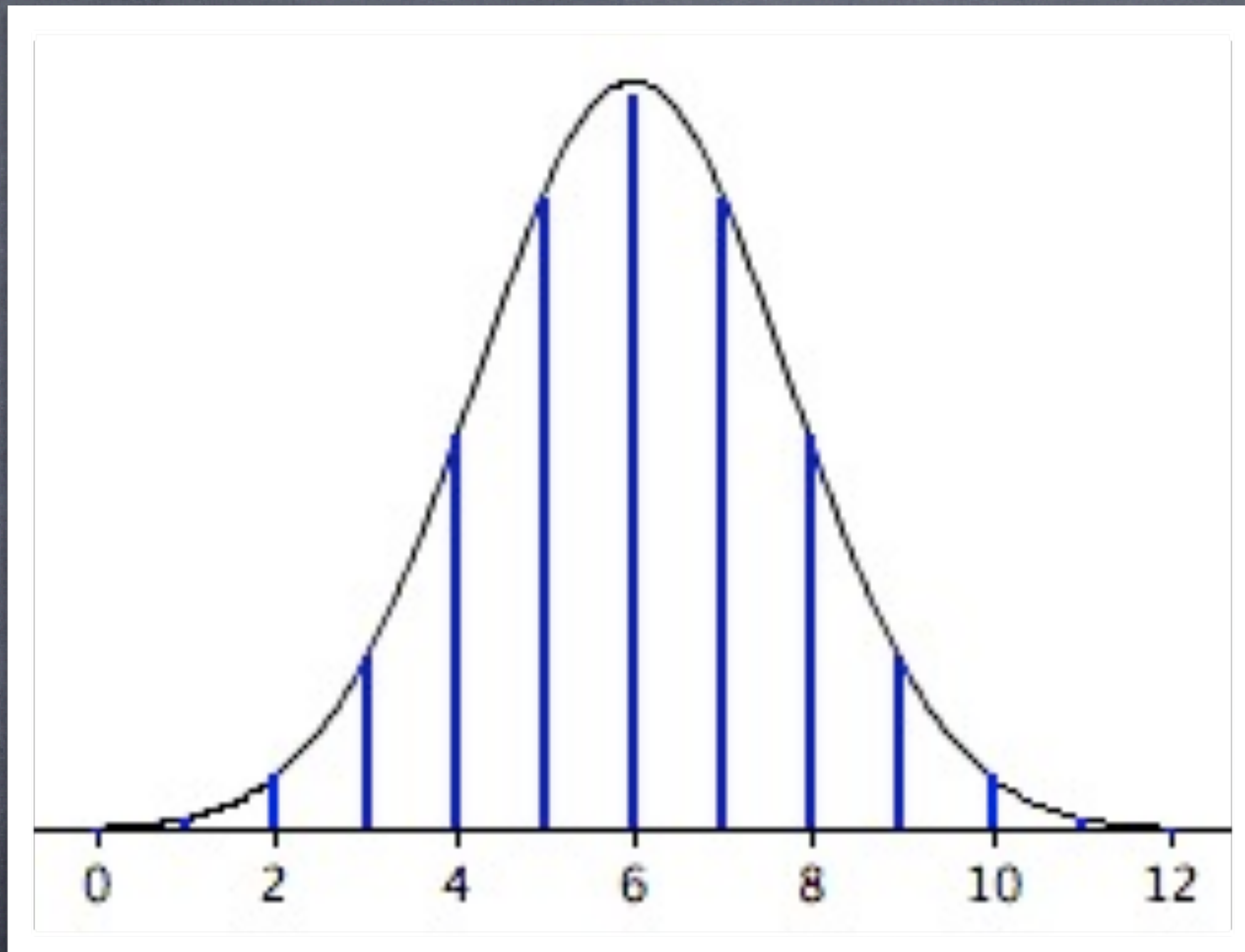
- `all = { <cl,cc(cl)> | cl <- classes }`
- `min(all), max(all), avg(all), mod(all)?`
- `threshold(i, rel[&T, int] r) = { x | <x,v> <- r, v >= i };?`
- What does an aggregate tell you?
- What does a value over a threshold tell you?

CC: stats

- The total CC metric of a class, a system, a package, does NOT MAKE SENSE AT ALL.
- **It measures something that does not exist** (control flow of a class?)
- A euro for every tool that computes this aggegrate...

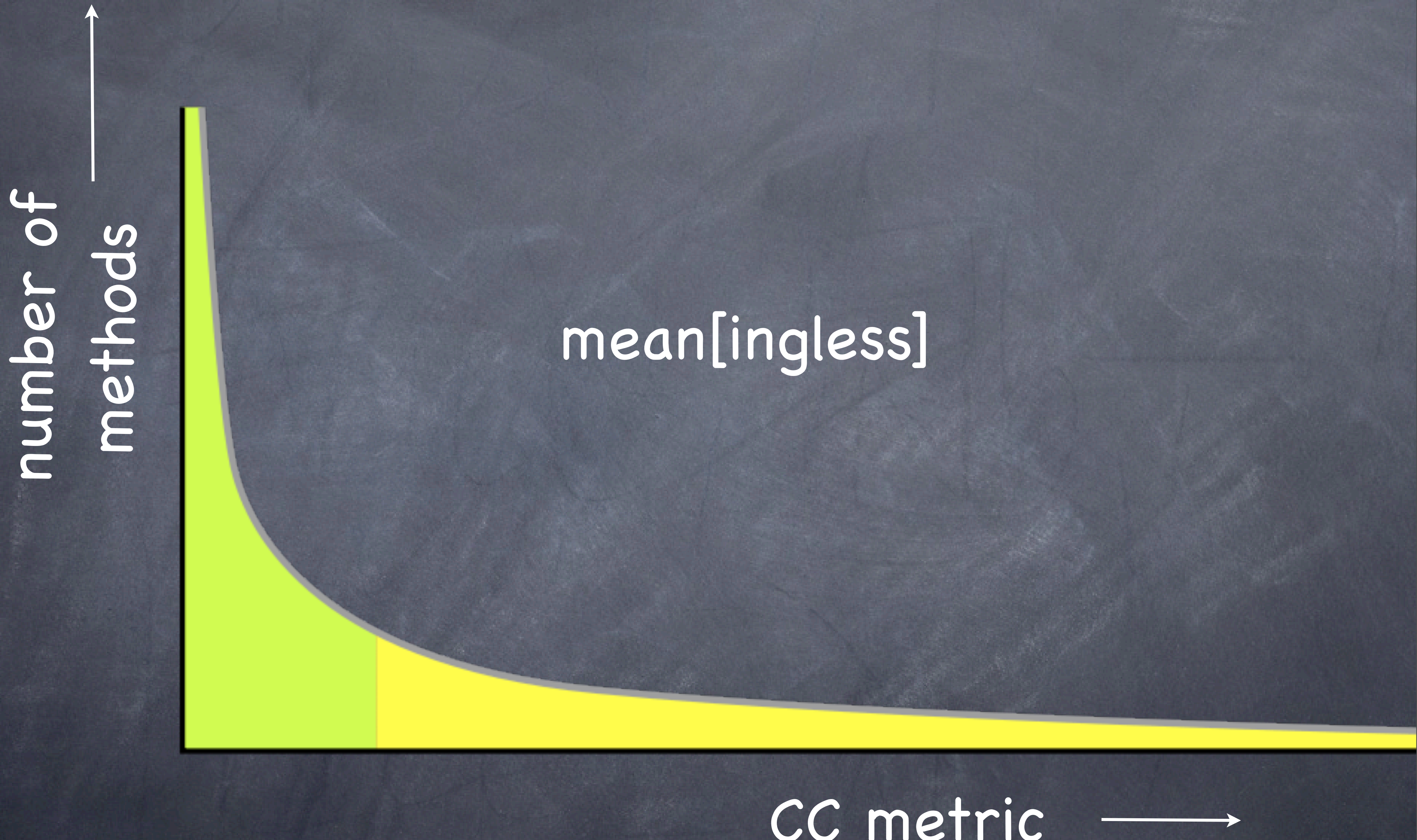
CC: distribution?

number of methods



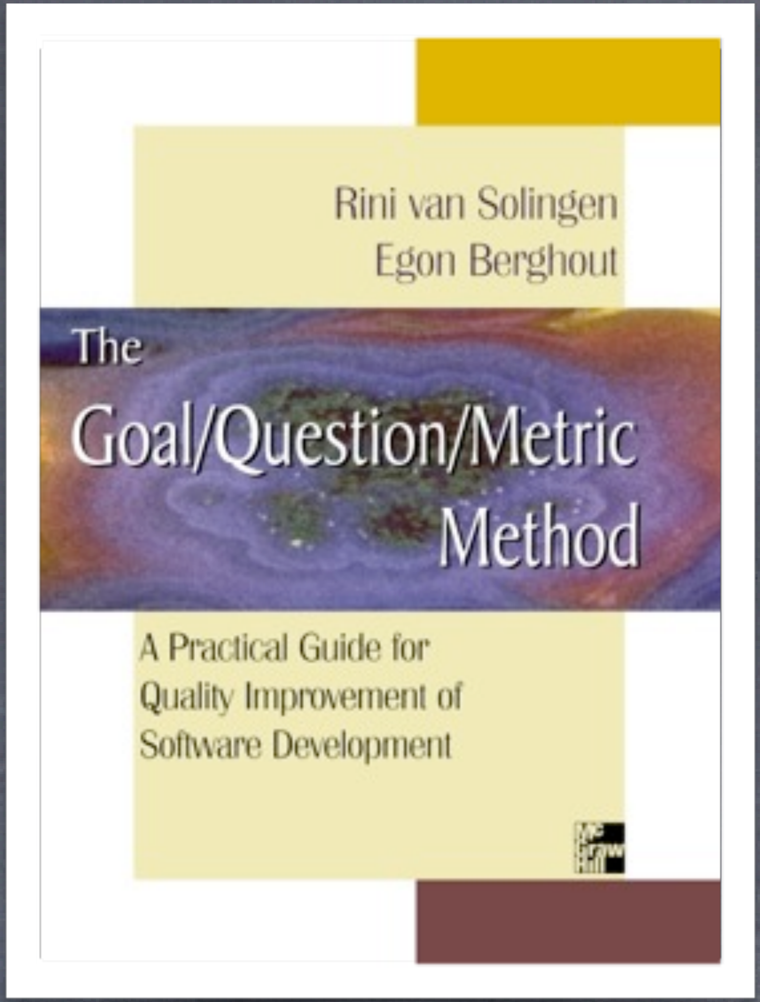
CC metric

CC: distribution!



How to do aggregate then?

- Software Improvement Group Maintainability model (www.sig.eu)
- "Count the number of lines of code that contribute to methods that have a high mccabe"
- Answers the question: what % of the system is really complex?
  - `"sum(NCLOC(methods) / sum(NCLOC(threshold(10, cc(methods)))"`
  - `threshold(10, cc(methods)` may be valueable info to programmers



- Works better to start from what you need to know than from what you can measure
- You'll find out when you can't measure it
- It will make sense if you can measure it

Learn from econometrics

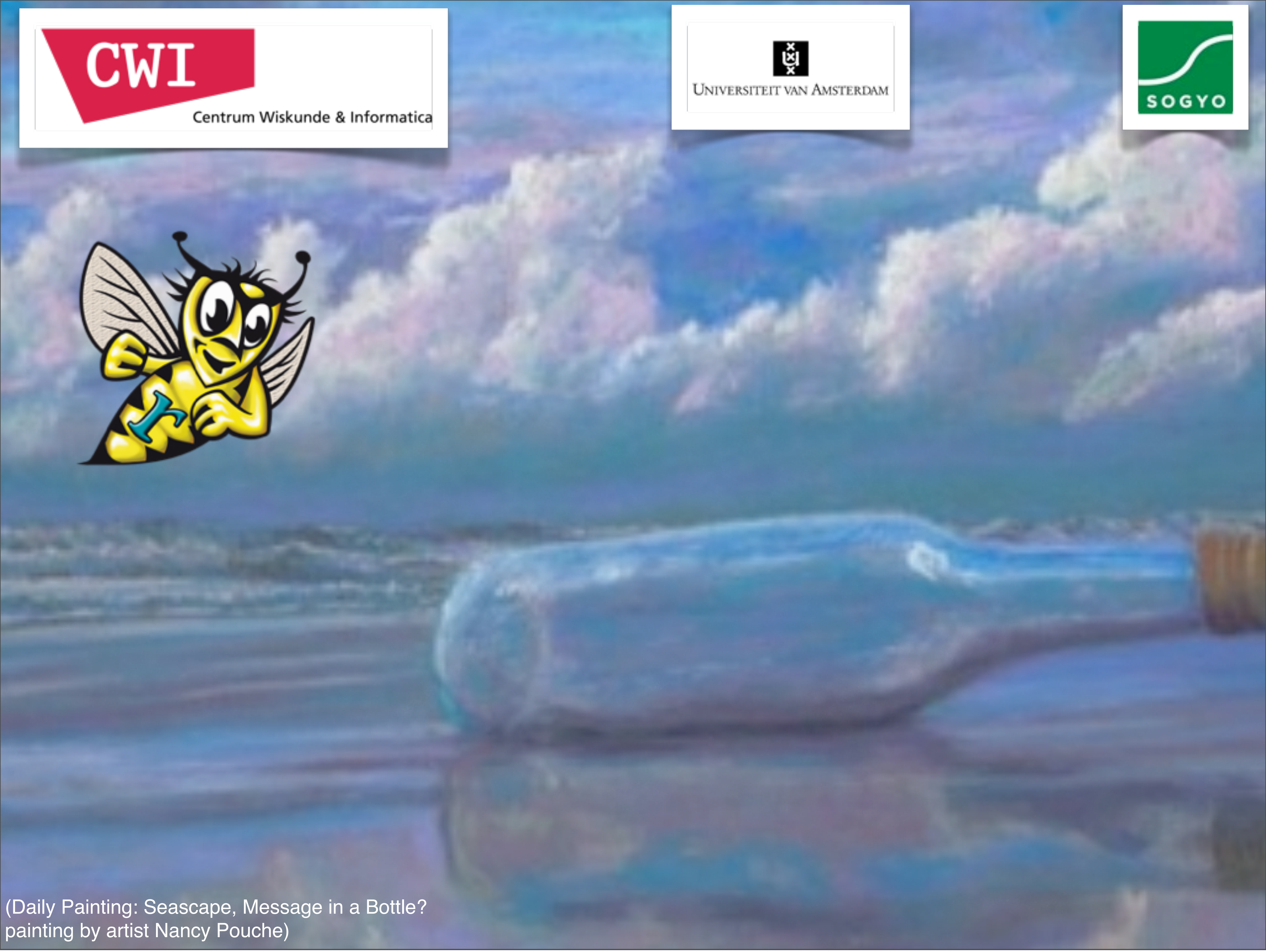
- Software metrics can learn from econometrics
- Study distributions, not aggegrations
- Study differences not general truths or thresholds
- "Gini, Theil, Hoover, Atkinson"
- Rascal (will) include libraries for all these tools
- There is no subtitute for thinking, but visualizing helps!

Rascal gives you

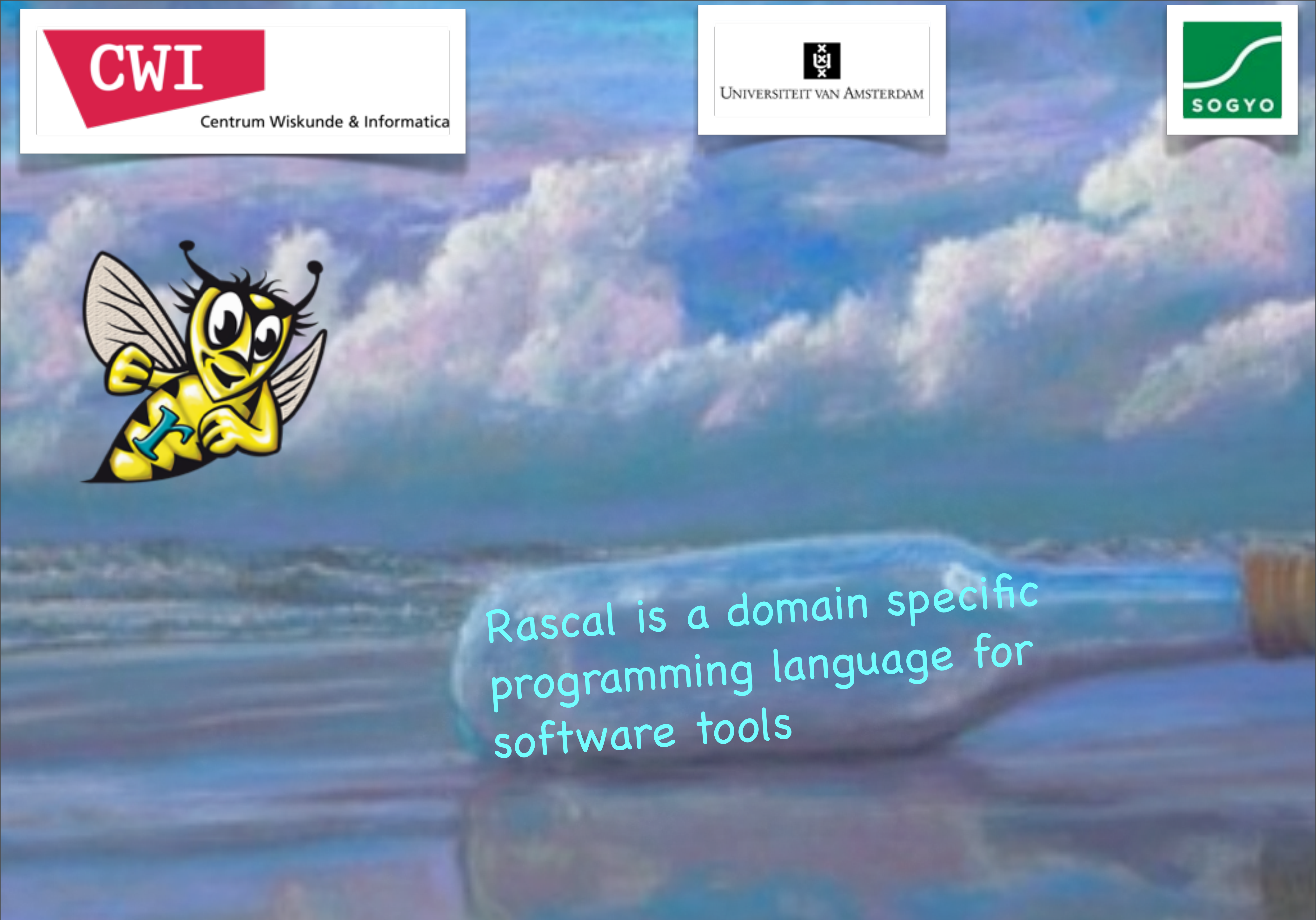
- Tools to make front-ends
- Libraries of front-ends
- Integration with Eclipse IDE
- Visualization
- Queries
- ...

What else?

- Modeling & Simulation
- Repository mining
- Domain specific languages
- Generating code
- Visualizating code
- Source-to-source transformation
- Refactoring



(Daily Painting: Seascape, Message in a Bottle?
painting by artist Nancy Pouche)



(Daily Painting: Seascape, Message in a Bottle? painting by artist Nancy Pouche)

CWI Centrum Wiskunde & Informatica

Universiteit van Amsterdam

SOGYO

http://www.cwi.nl/sen1

http://www.rascal-mpl.org

http://ask.rascal-mpl.org

http://tutor.rascal-mpl.org

Rascal is a domain specific programming language for software tools

(Daily Painting: Seascape, Message in a Bottle? painting by artist Nancy Pouche)

CWI Centrum Wiskunde & Informatica

Universiteit van Amsterdam

SOGYO

http://www.cwi.nl/sen1

http://www.rascal-mpl.org

http://ask.rascal-mpl.org

http://tutor.rascal-mpl.org

Rascal is a domain specific programming language for software tools

Master internships
Funding opportunities
Visit CWI

(Daily Painting: Seascape, Message in a Bottle? painting by artist Nancy Pouche)

CWI Centrum Wiskunde & Informatica

Universiteit van Amsterdam

SOGYO

http://www.cwi.nl/sen1

http://www.rascal-mpl.org

http://ask.rascal-mpl.org

http://tutor.rascal-mpl.org

Rascal is a domain specific programming language for software tools

Master internships
Funding opportunities
Visit CWI

mailto:Jurgen.Vinju@cwi.nl

twitter:@jurgenvinju

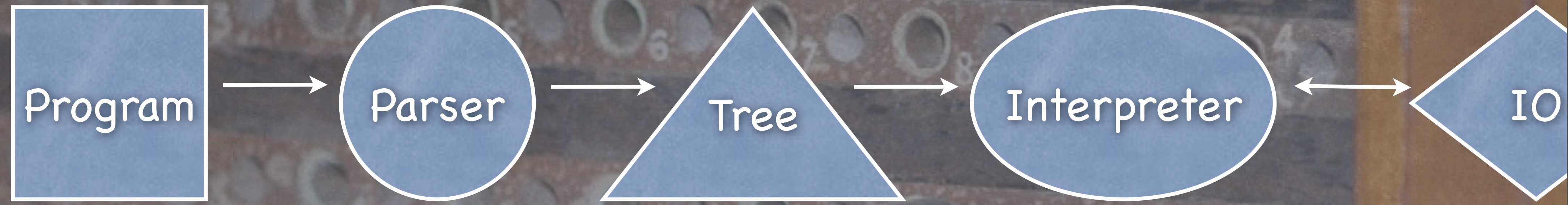
(Daily Painting: Seascape, Message in a Bottle? painting by artist Nancy Pouche)

A case of
Visitor versus Interpreter Pattern

Paul Klint, Mark Hills, Tijs van der Storm,
Jurgen Vinju

[TOOLS Europe 2011]

Case:



- Abstract syntax trees (ASTs)
- Different operations on ASTs
- 400 node types, 140 node type categories
- Dispatch, dispatch, dispatch (case distinction)
- Maintaining the ± 100 kLOC java code is the issue

We compare design (patterns) to learn which is best in which situations

We compare design (patterns) to learn which is simpler

We compare design (patterns) to learn which is faster

We compare design (patterns) to learn which is easier to change

AST instance

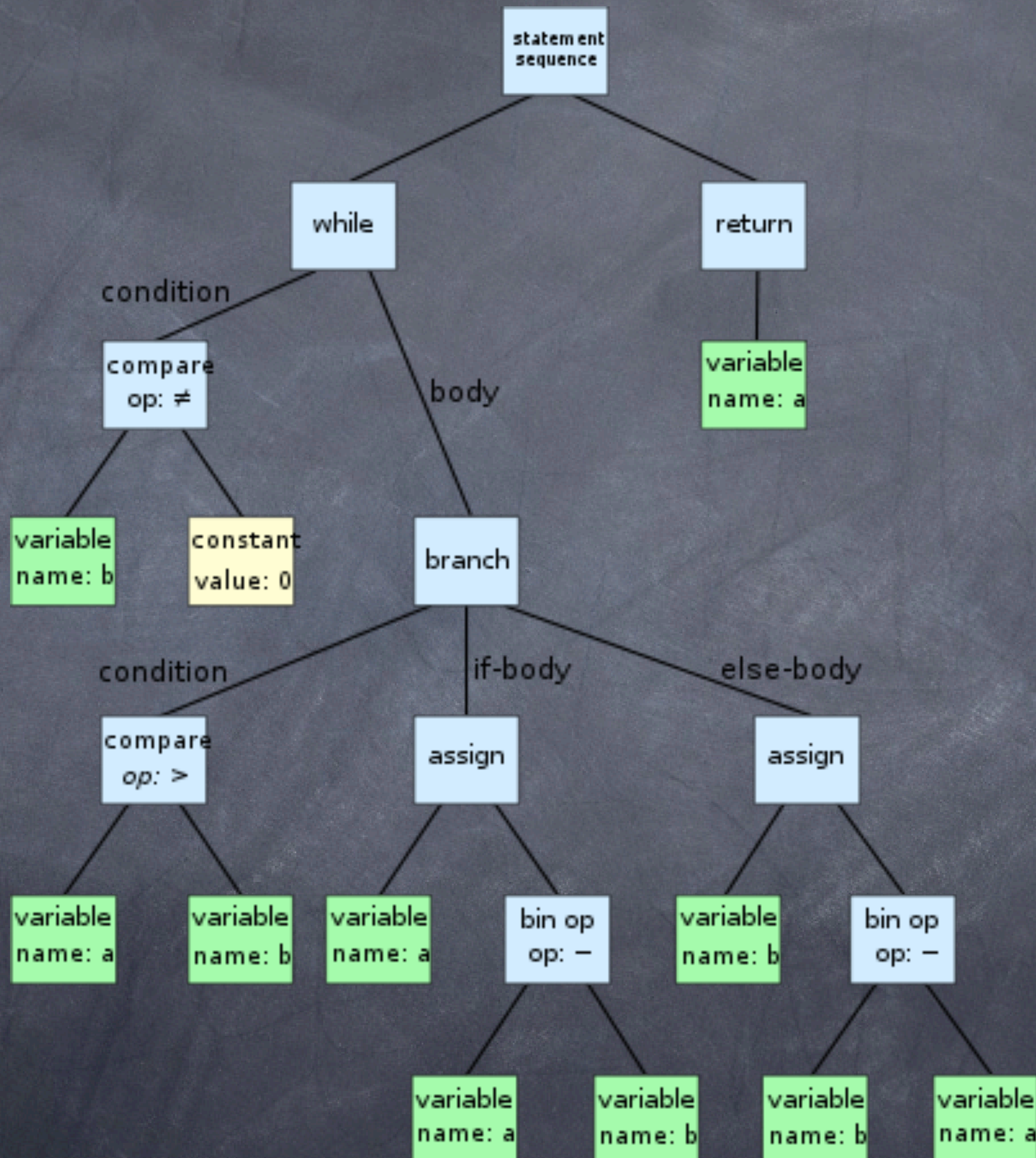


image from wikipedia.org

Composite Pattern

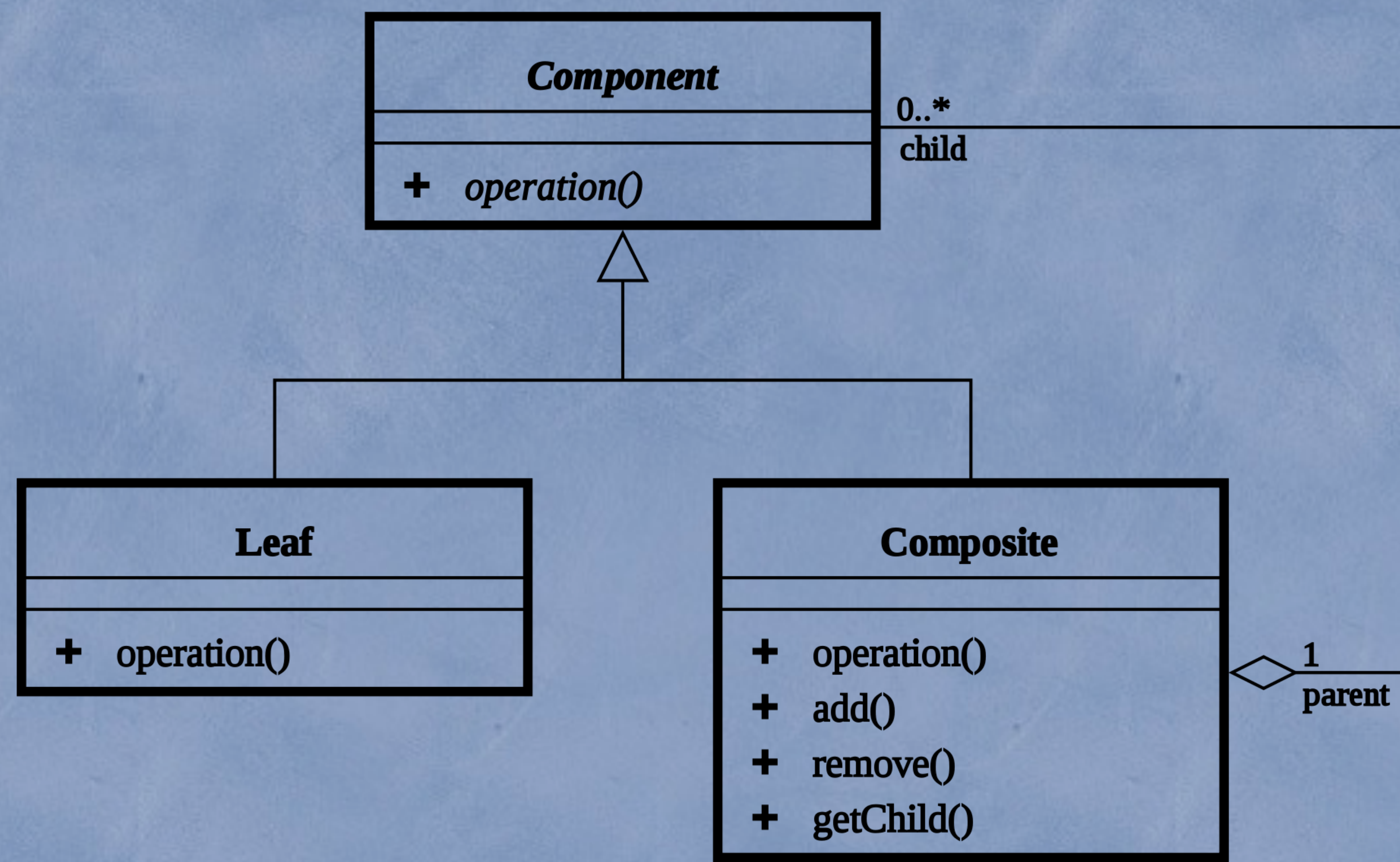


Fig. 2. The Composite Pattern[3]

image from wikipedia.org

Composite Pattern

Statement

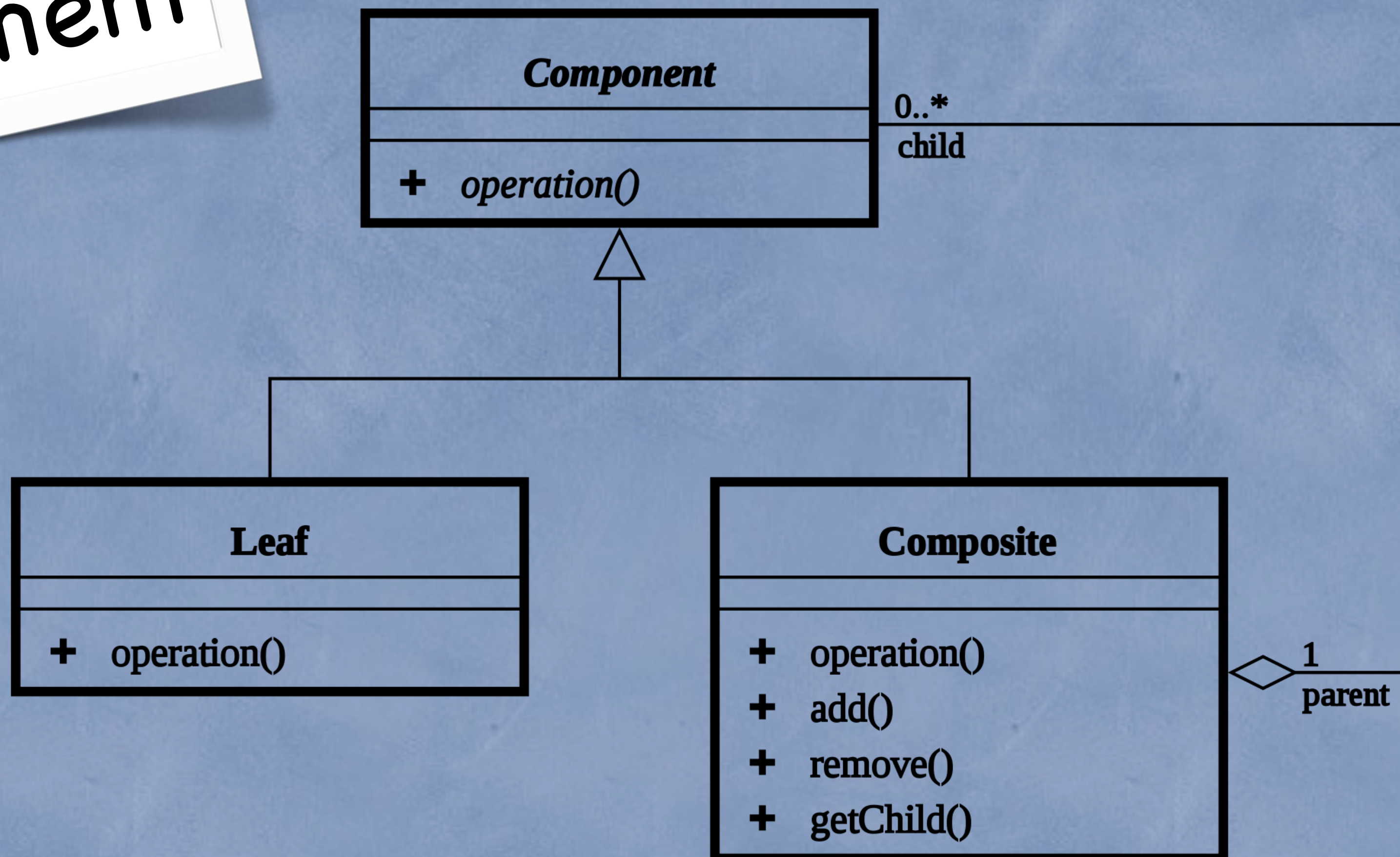


Fig. 2. The Composite Pattern[3]

image from wikipedia.org

Composite Pattern

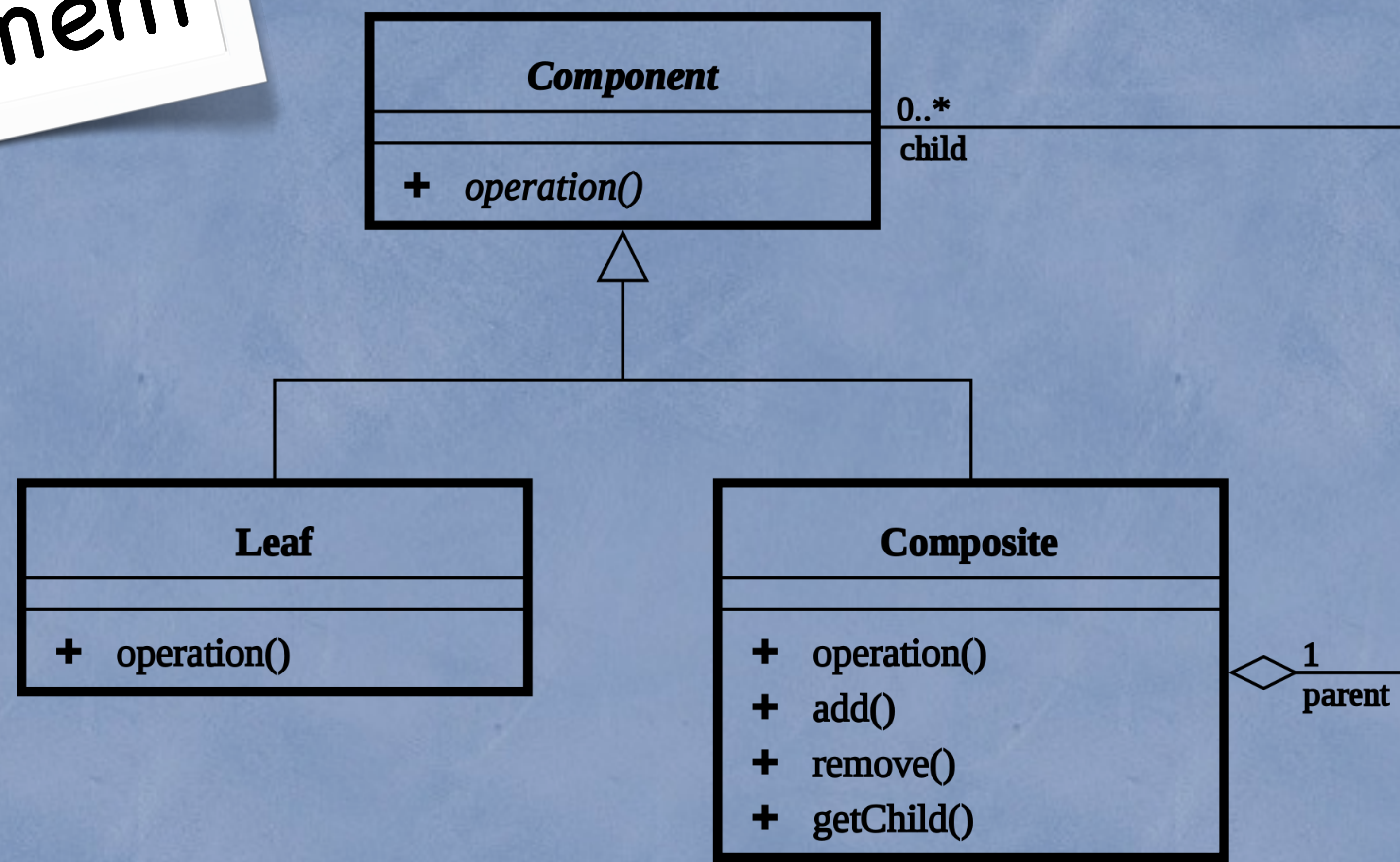


Fig. 2. The Composite Pattern[3]

Statement

NoOp

image from wikipedia.org

Composite Pattern

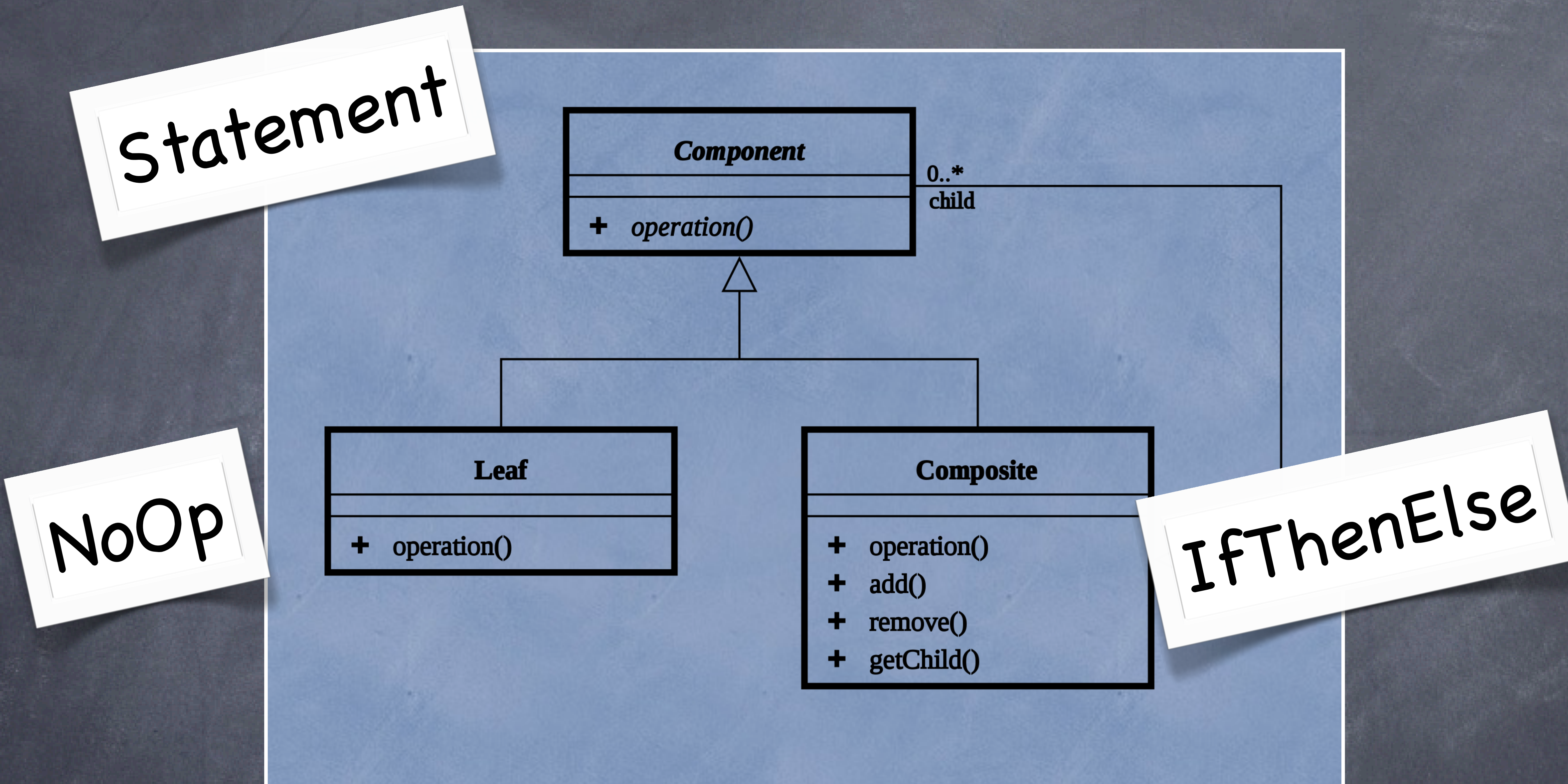


Fig. 2. The Composite Pattern[3]

image from wikipedia.org

Composite Pattern

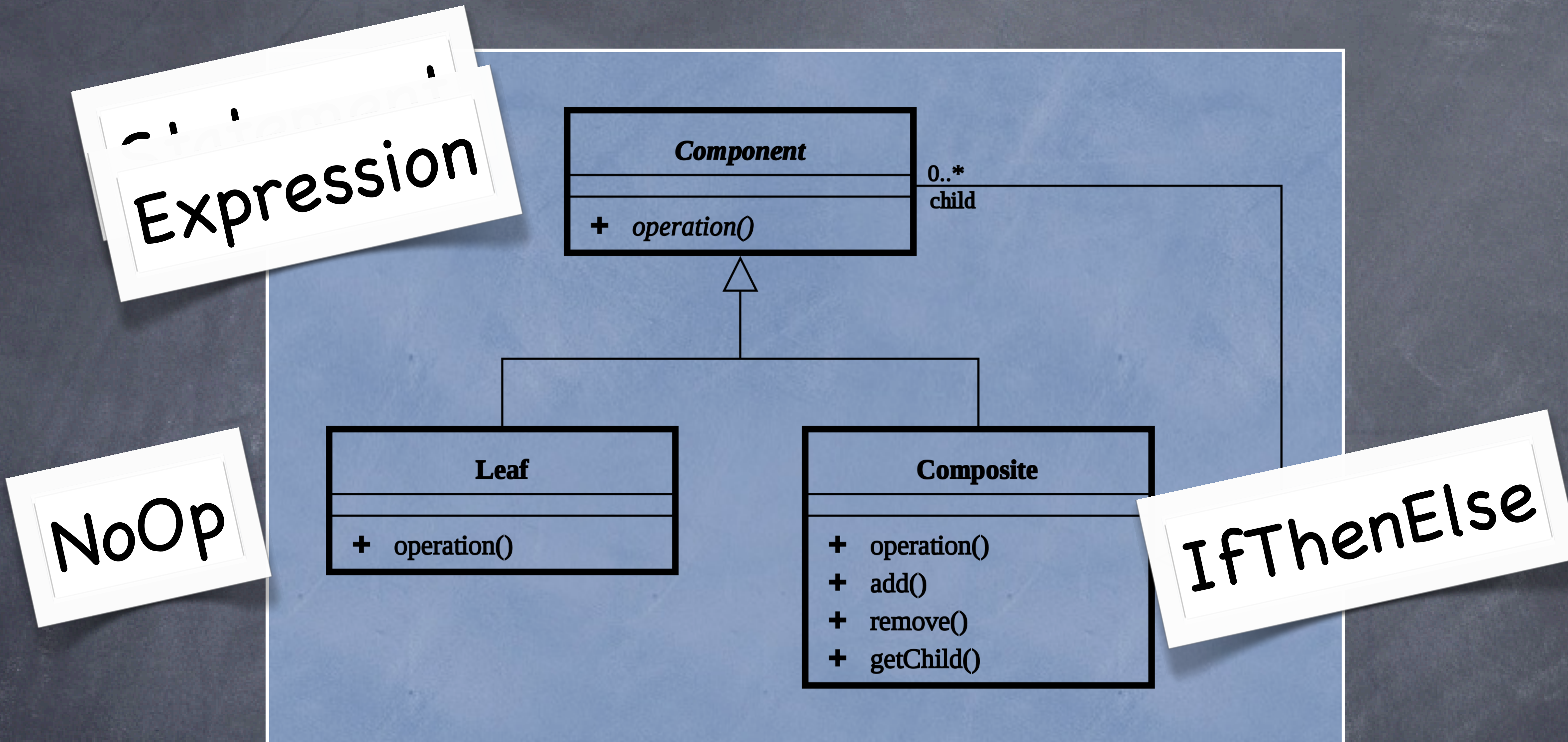


Fig. 2. The Composite Pattern[3]

image from wikipedia.org

Composite Pattern

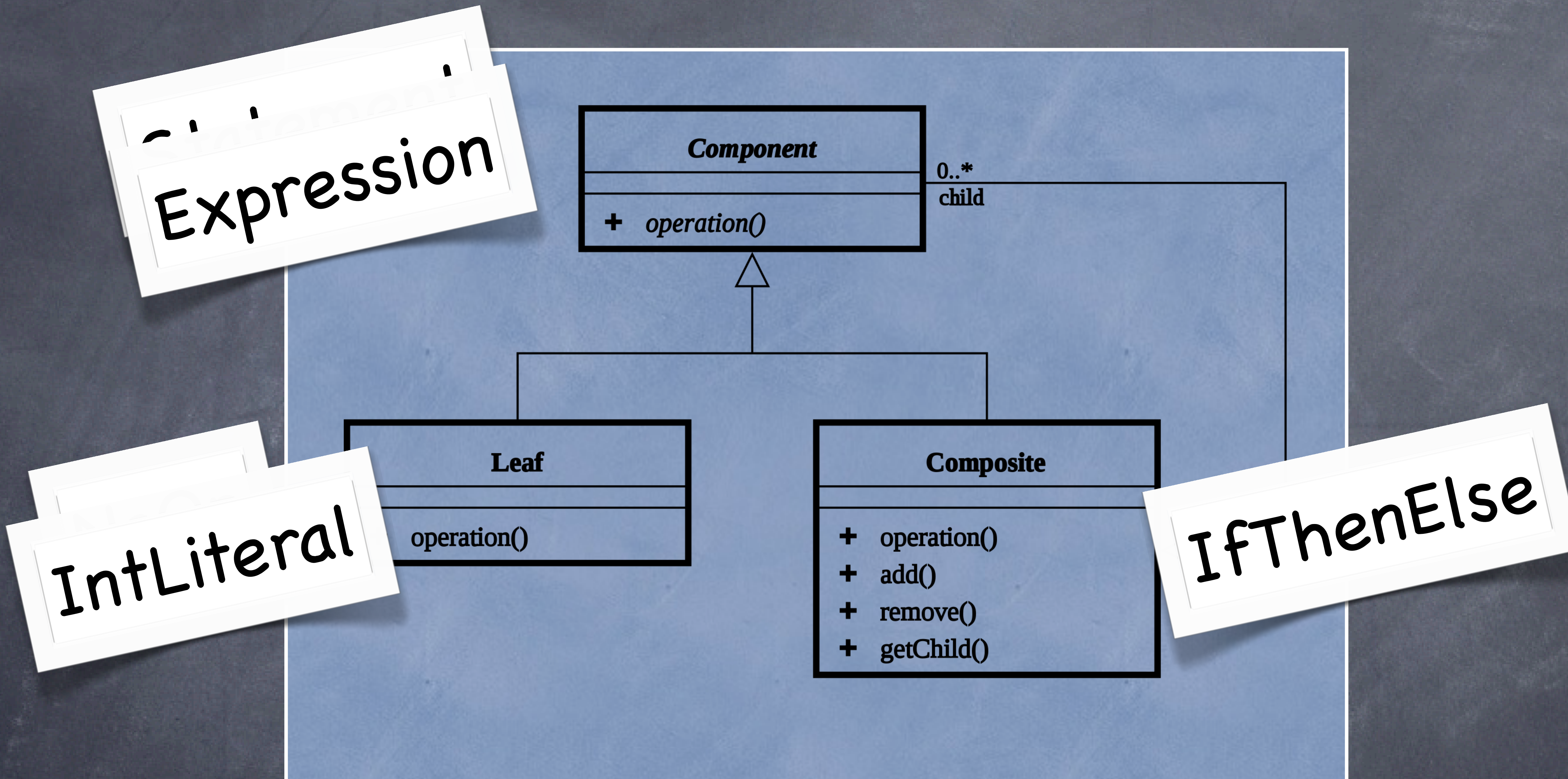


Fig. 2. The Composite Pattern[3]

image from wikipedia.org

Composite Pattern

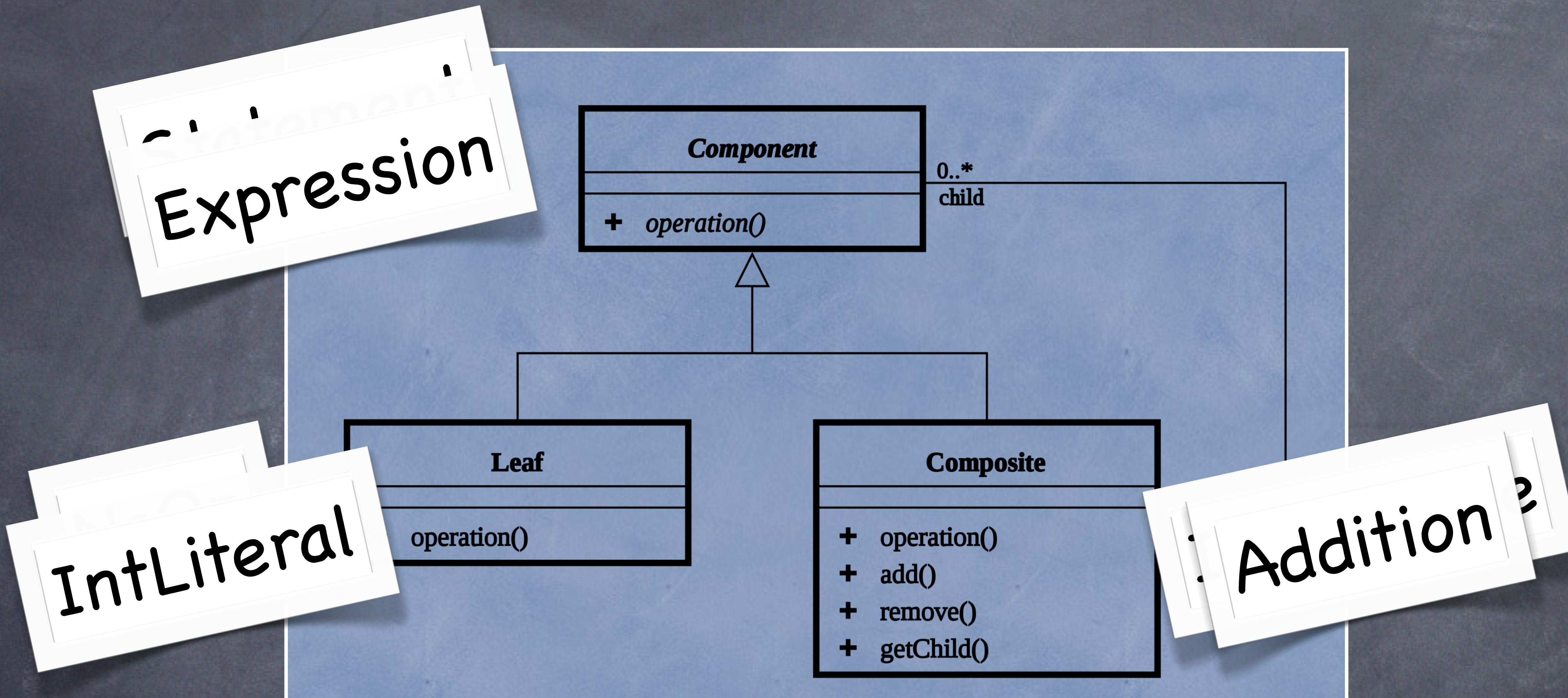


Fig. 2. The Composite Pattern[3]

image from wikipedia.org

Interpreter Pattern

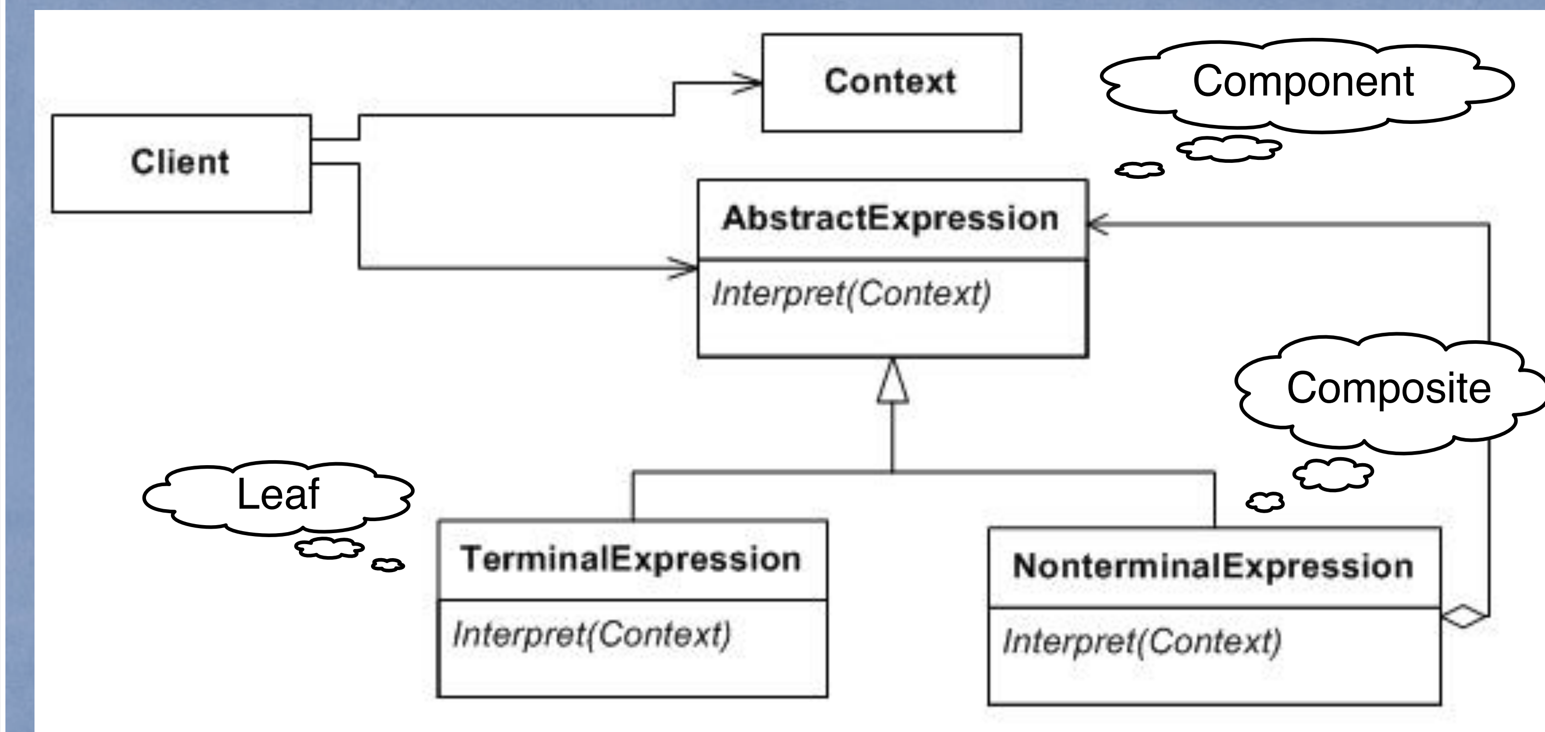


Fig. 4. The Interpreter Pattern with references to Composite (Figure 2).[7]

image from wikipedia.org

Interpreter Pattern

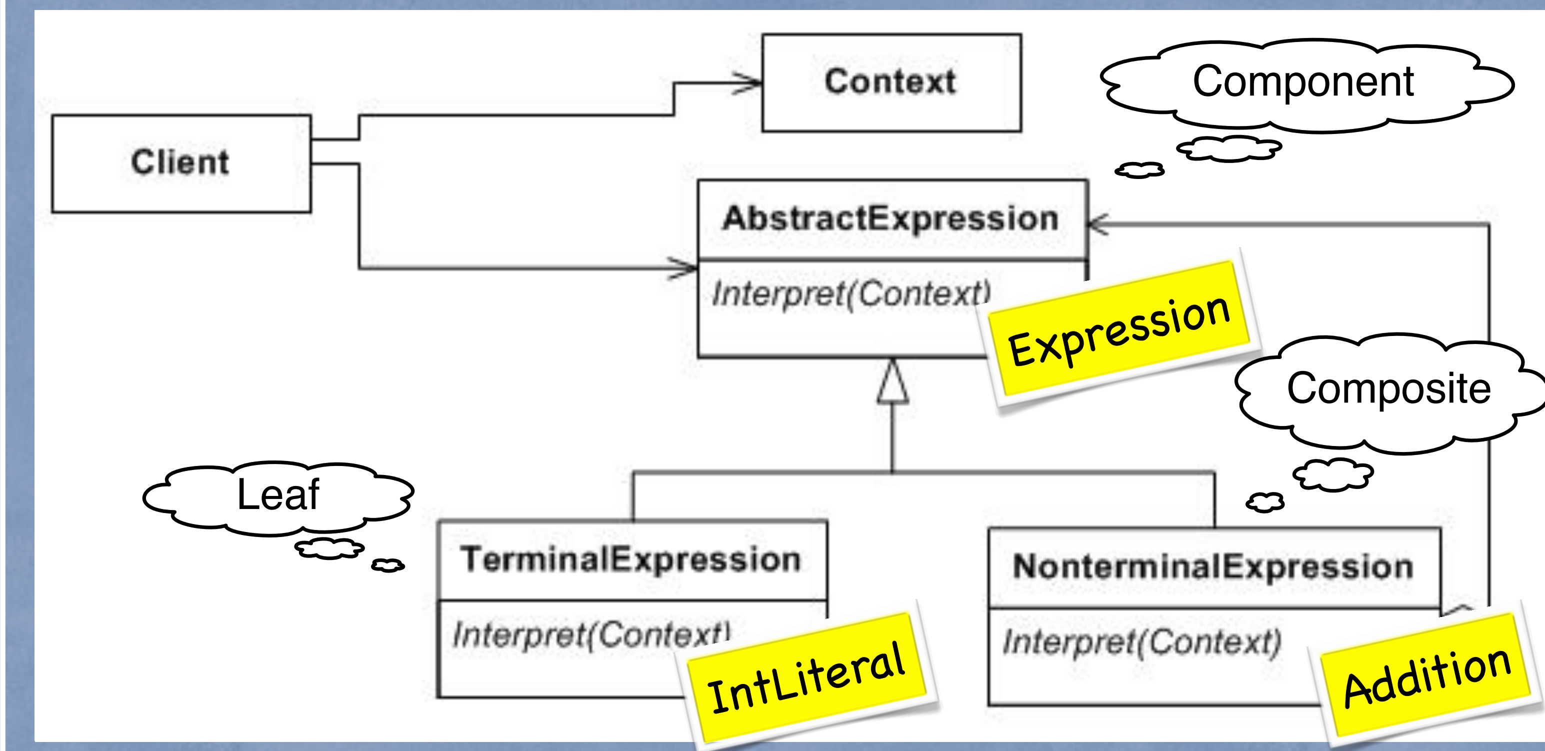


Fig. 4. The Interpreter Pattern with references to Composite (Figure 2).[7]

image from wikipedia.org

Visitor Pattern

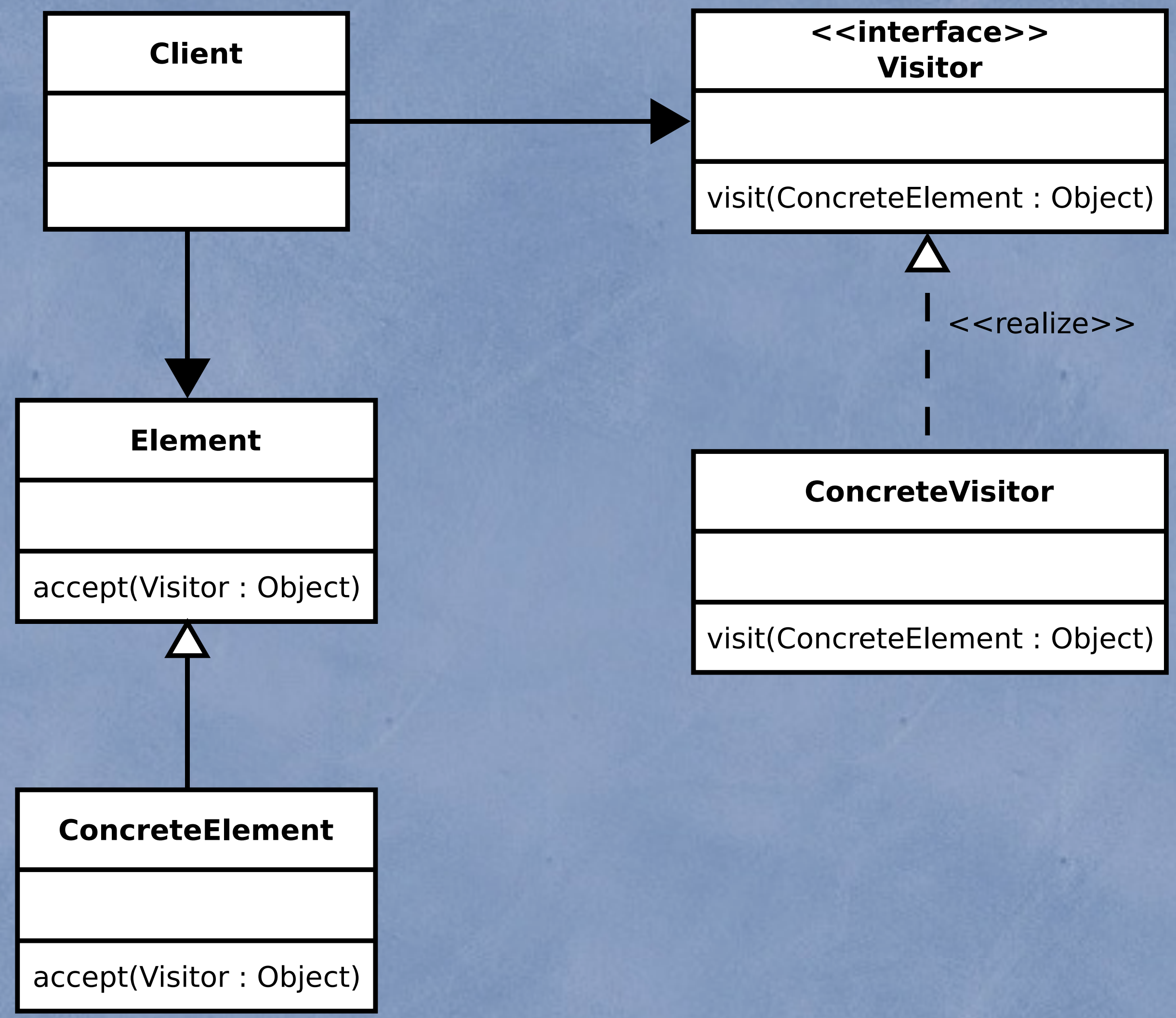


Fig. 3. The Visitor Pattern[4]

image from wikipedia.org

Visitor Pattern

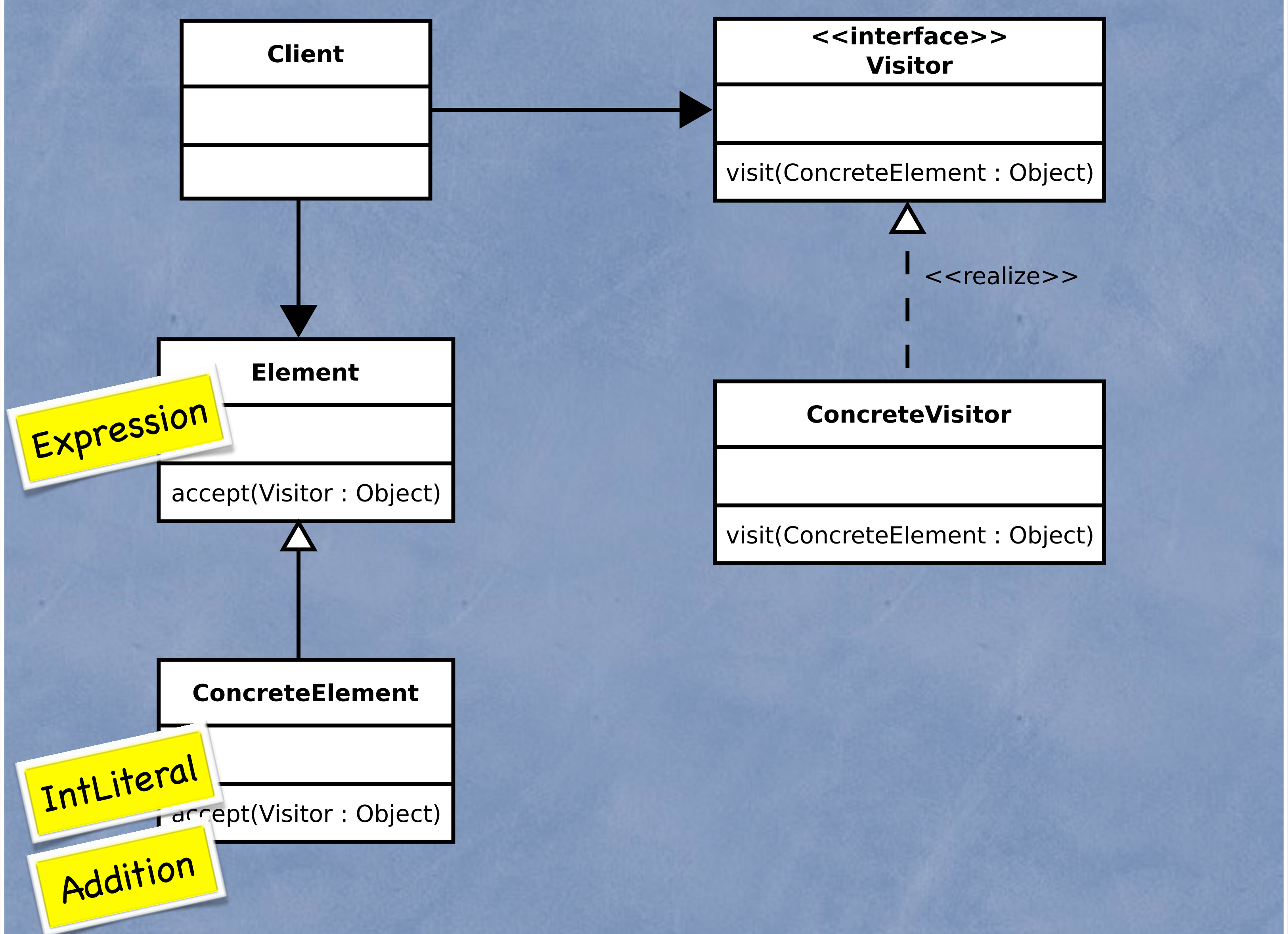


Fig. 3. The Visitor Pattern[4]

image from wikipedia.org

Visitor Pattern

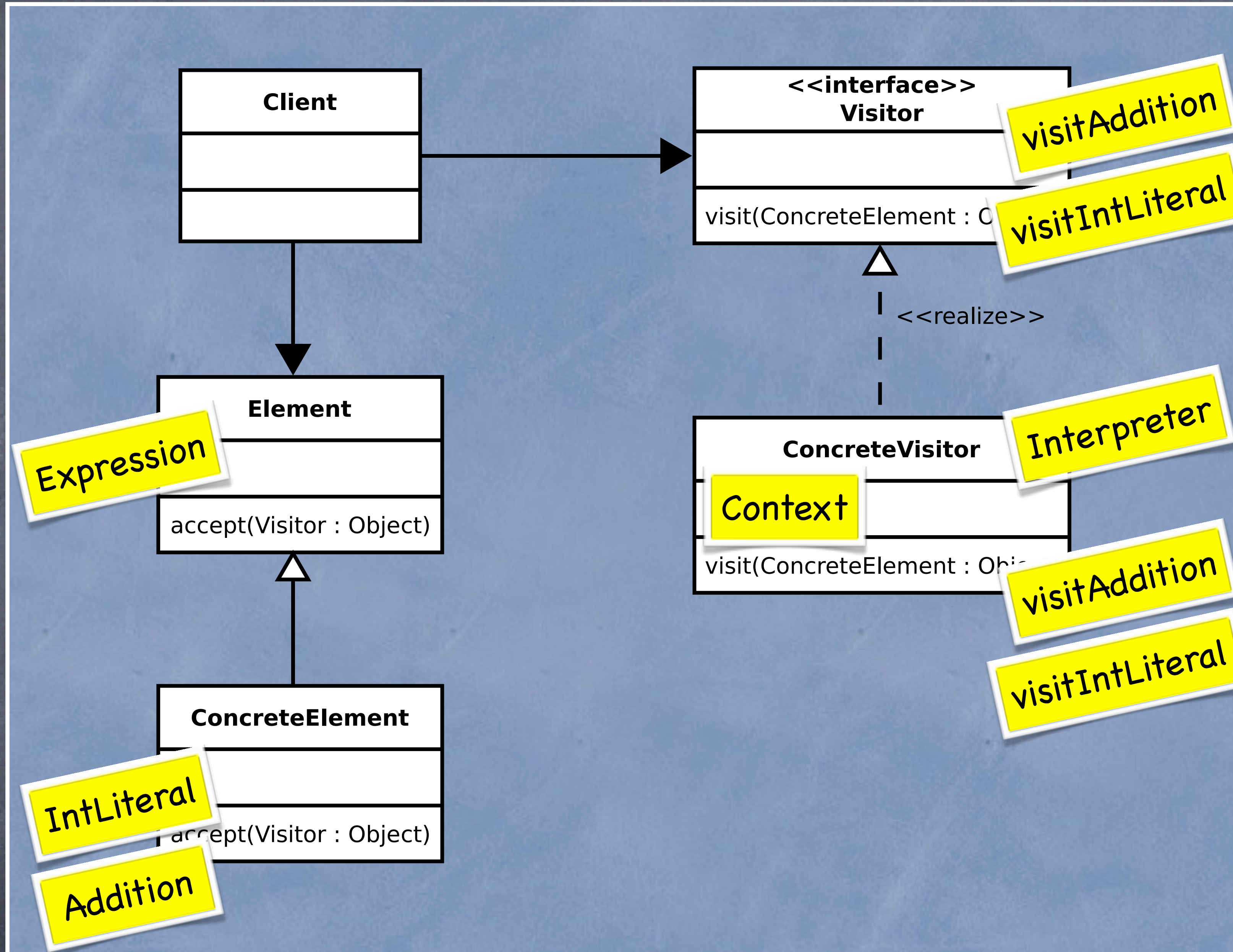


Fig. 3. The Visitor Pattern[4]

image from wikipedia.org

Visitor Pattern

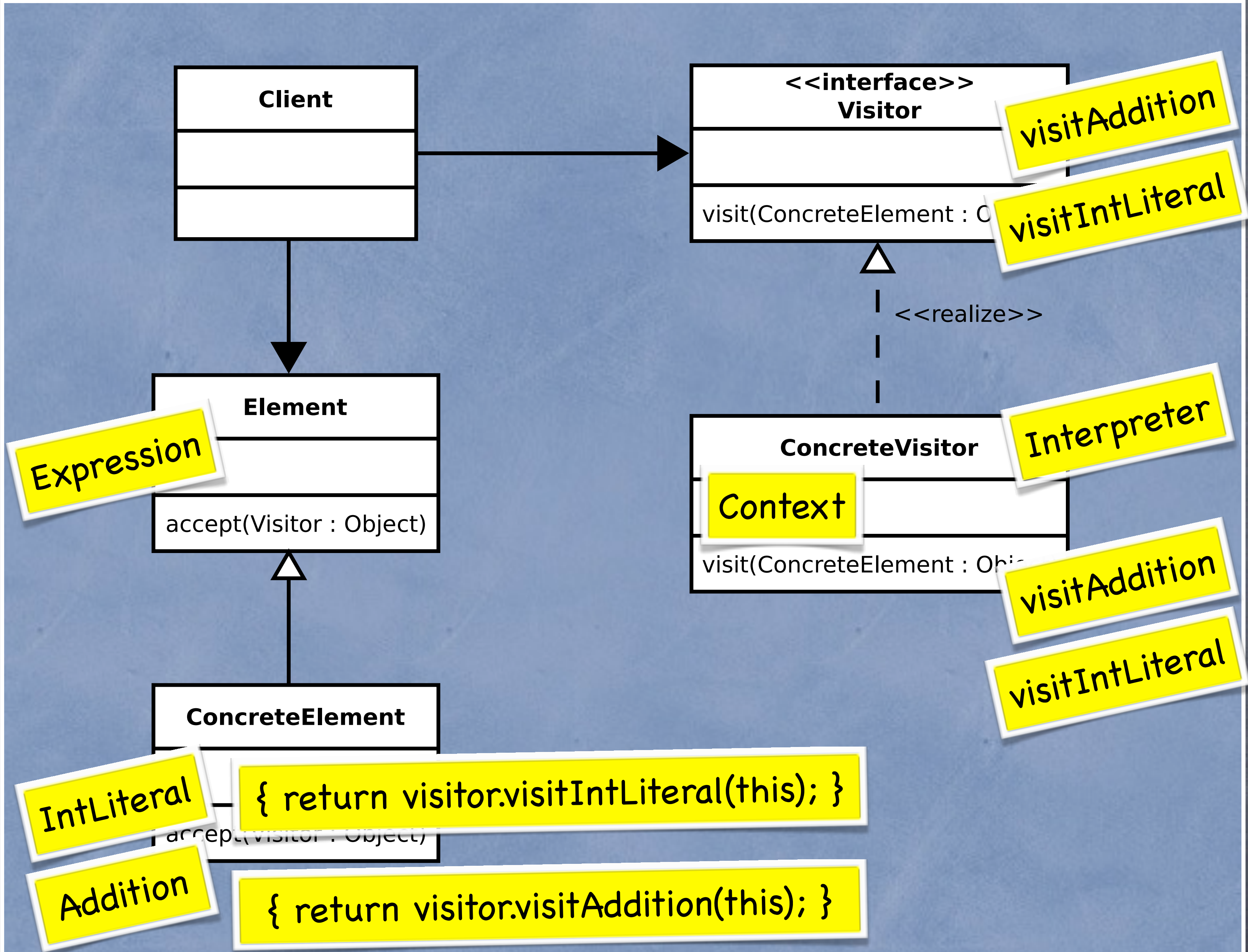


Fig. 3. The Visitor Pattern[4]

image from wikipedia.org

Visitor design pattern and the **Interpreter** design pattern are functionally inter-changeable

Visitor design pattern and the **Interpreter** design pattern are functionally inter-changeable

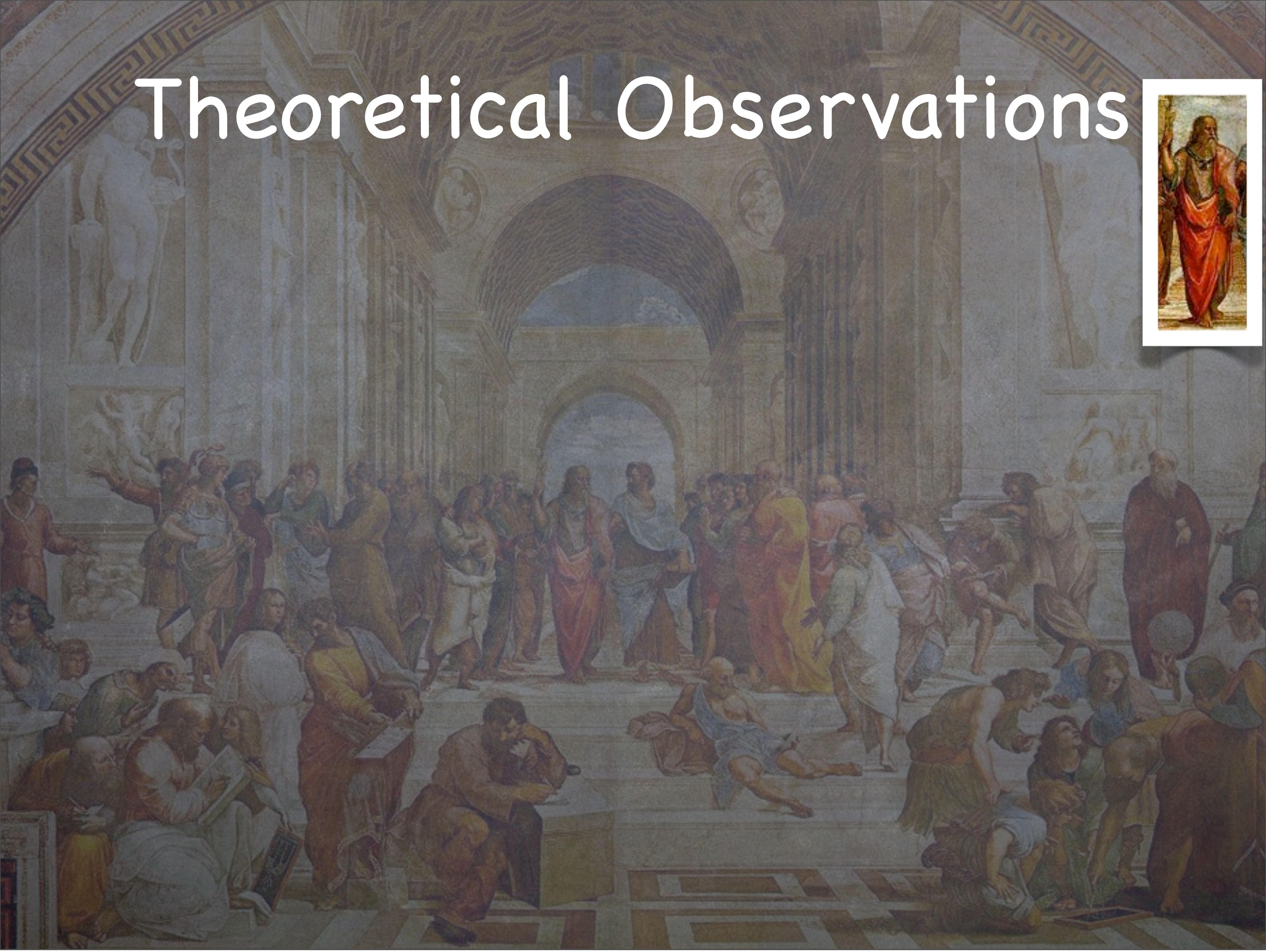
But, they are different in **non-functional** properties

Visitor design pattern and the **Interpreter** design pattern are functionally inter-changeable

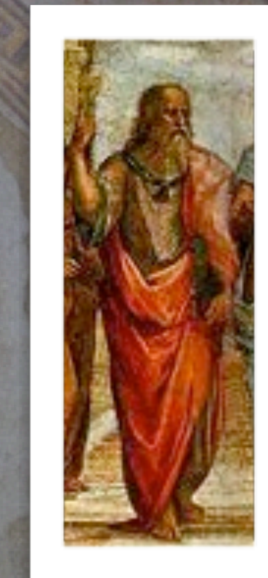
But, they are different in **non-functional** properties

And, these **emergent** properties tend to be difficult to predict

Theoretical Observations



Theoretical Observations



- Visitor is conceptually more complex
  - Interpreter is only a small extension of composite

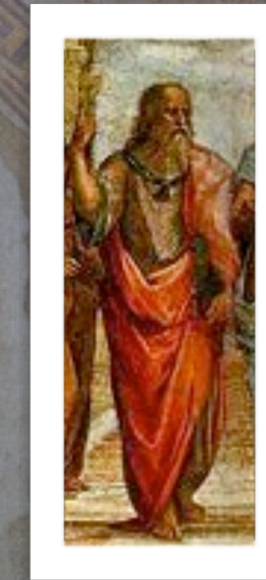
Theoretical Observations

- Visitor is conceptually more complex
  - Interpreter is only a small extension of composite
- Visitor encapsulates entire algorithms
  - Interpreter encapsulates language constructs

Theoretical Observations

- Visitor is conceptually more complex
  - Interpreter is only a small extension of composite
- Visitor encapsulates entire algorithms
  - Interpreter encapsulates language constructs
- Visitor's decoupling implies **dynamic** indirection
  - Interpreter has less dynamic dispatch

Theoretical Observations



- Visitor is conceptually more complex

Harder to maintain, right?

  - Interpreter is only a small extension of composite
- Visitor encapsulates entire algorithms
  - Interpreter encapsulates language constructs
- Visitor's decoupling implies **dynamic** indirection
  - Interpreter has less dynamic dispatch

Theoretical Observations

- Visitor is conceptually more complex
  - Interpreter is only a small extension of composite

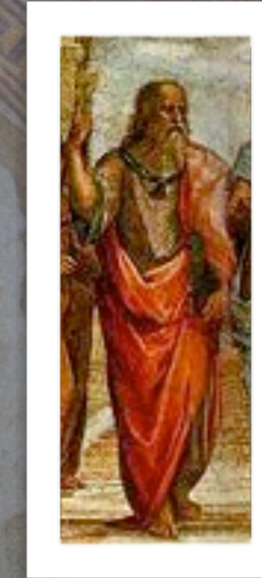
Harder to maintain, right?

- Visitor encapsulates
  - Interpreter encapsulates language constructs

Easy for adding algorithm, hard for adding new language construct, right?

- Visitor's decoupling implies **dynamic** indirection
  - Interpreter has less dynamic dispatch

Theoretical Observations



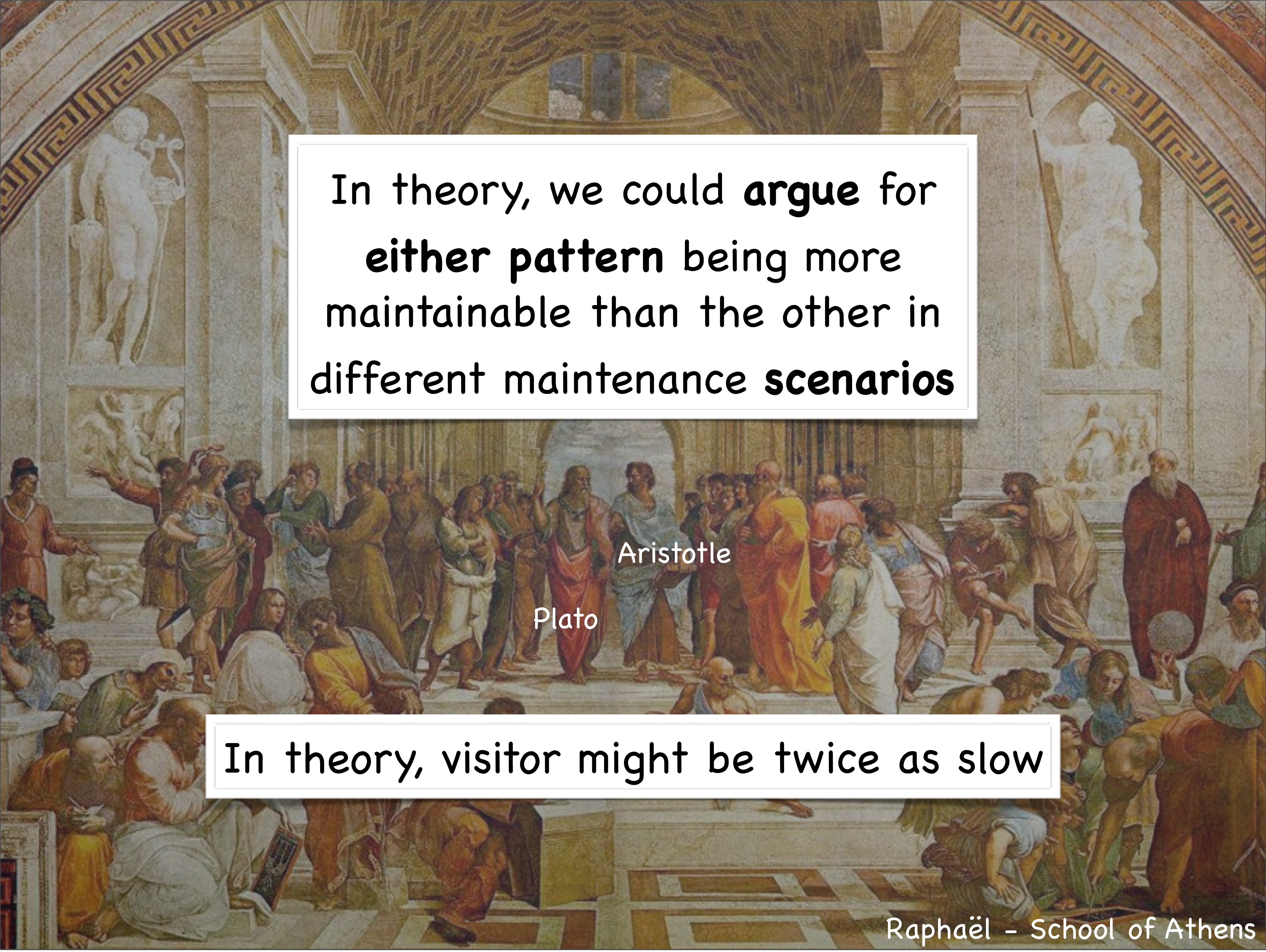
- Visitor is conceptually more complex
  - Interpreter is only a small extension of composite
- Visitor encapsulates ...
  - Interpreter encapsulates language constructs
- Visitor's dec... **dynamic** indirection
  - Interpreter has less dynamic dispatch

Harder to maintain, right?

Easy for adding algorithm, hard for adding new language construct, right?

Slower, right?

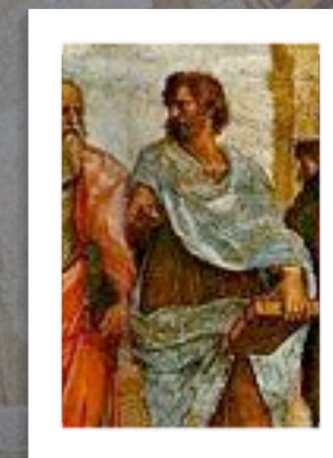
In theory, we could **argue** for **either pattern** being more maintainable than the other in different maintenance **scenarios**



In theory, visitor might be twice as slow

Raphaël - School of Athens

Empirical Observations



- Visitor-based interpreter is complex
  - Many visitors classes
  - Main interpreter is a "God class"
- Interpreter should run faster than this

Why this experiment?

Is the difference between Interpreter and Visitor **causing** a part of these two problems, or not at all?

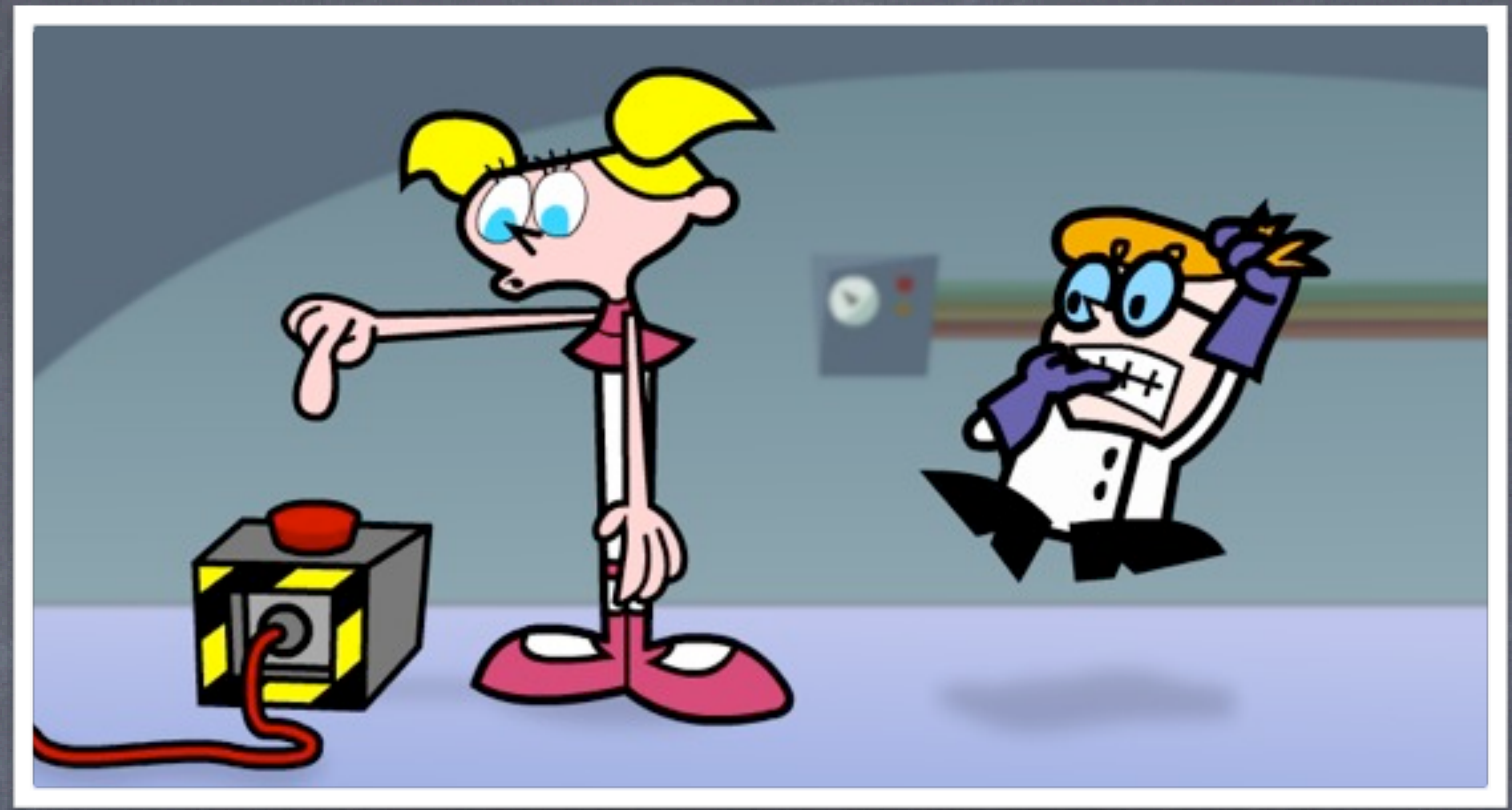
How does one answer such a question?

Why this lab setup?

Observing software "in the wild"

- In reality, there exist **no two different versions** of the same interpreter
- In reality, there are **many other factors** influencing maintenance and efficiency other than this design choice
- Reality is perhaps easy to see, but it is **very hard to understand**

Lab Experiment



cartoon network

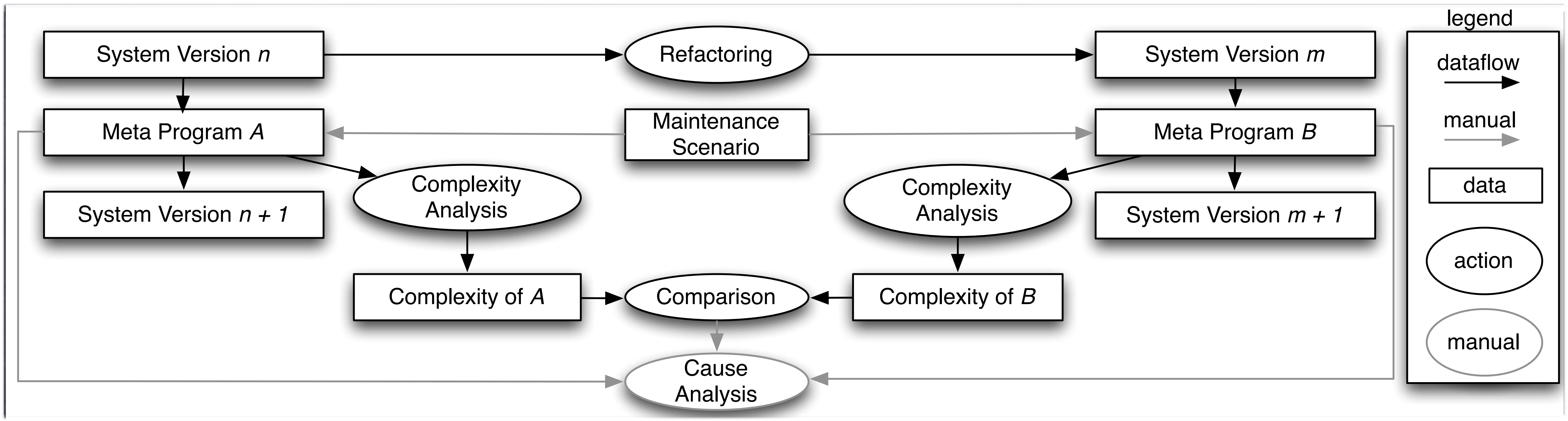
- In a lab we may **isolate** a factor
- In the lab we may **focus** on the effect
- In the lab we can observe **causality** more directly

Our Lab Setup

- Refactoring to get two versions
- Applying realistic maintenance scenarios
- Observing differences

A "refactoring" is an automated source-to-source program transformation that **guarantees** run-time **semantics** to be preserved.

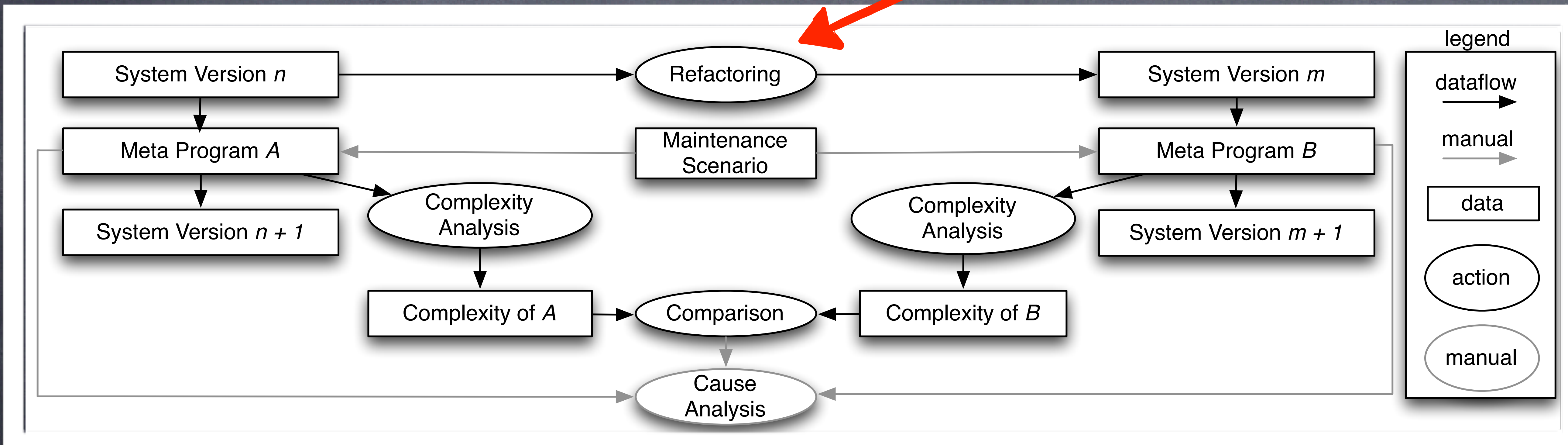
Isolating the variable



Rascal to implement Visitor to Interpreter **refactoring**

Isolating the variable

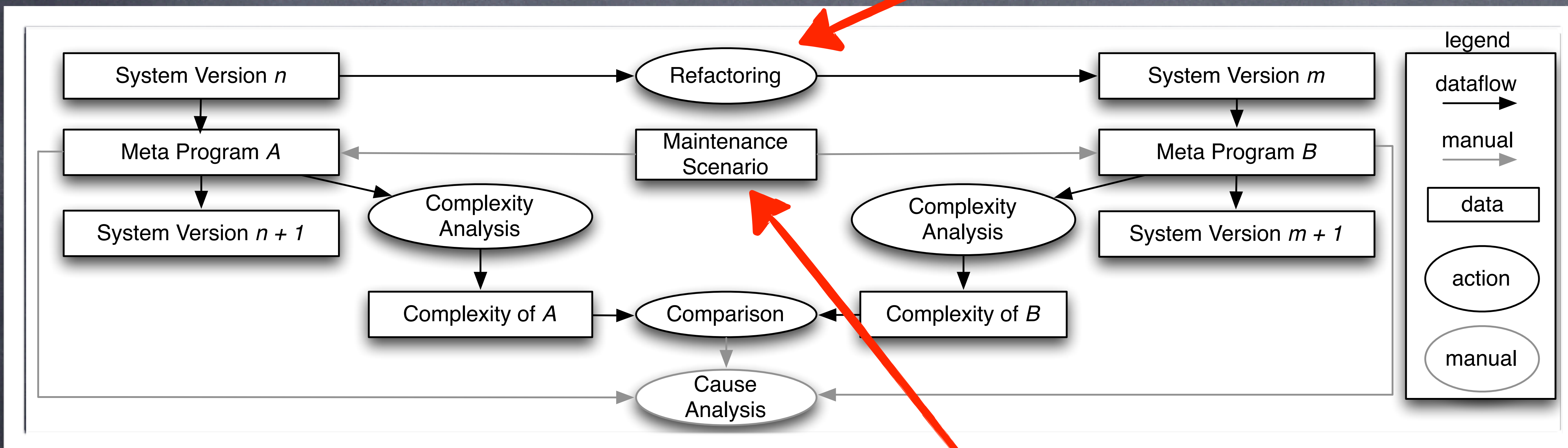
Key enabler



Rascal to implement Visitor to Interpreter **refactoring**

Isolating the variable

Key enabler

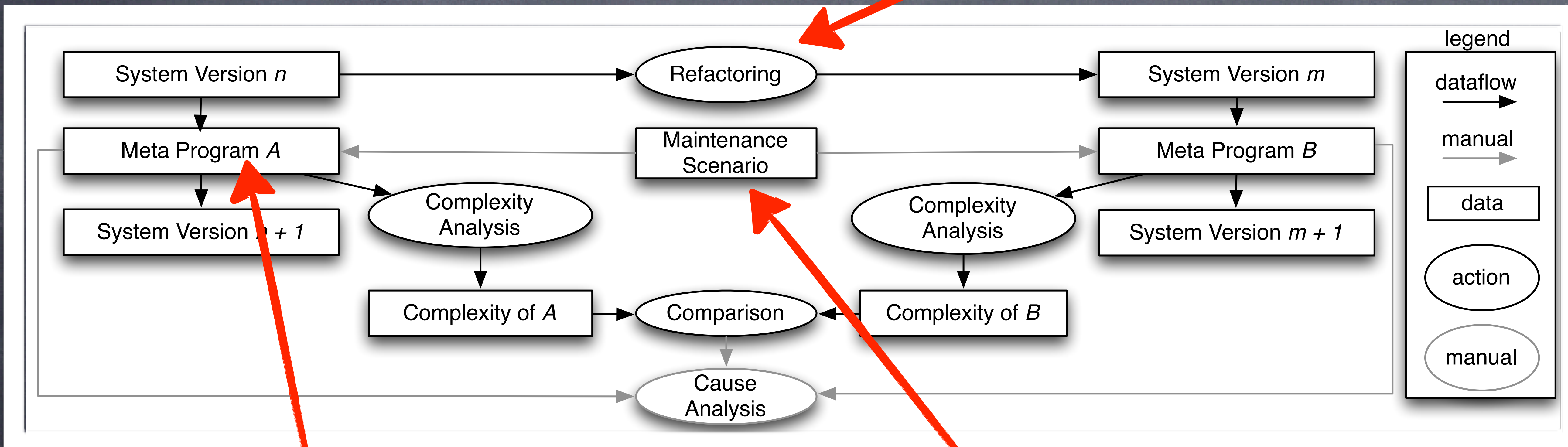


Manual labor

Rascal to implement Visitor to Interpreter **refactoring**

Isolating the variable

Key enabler

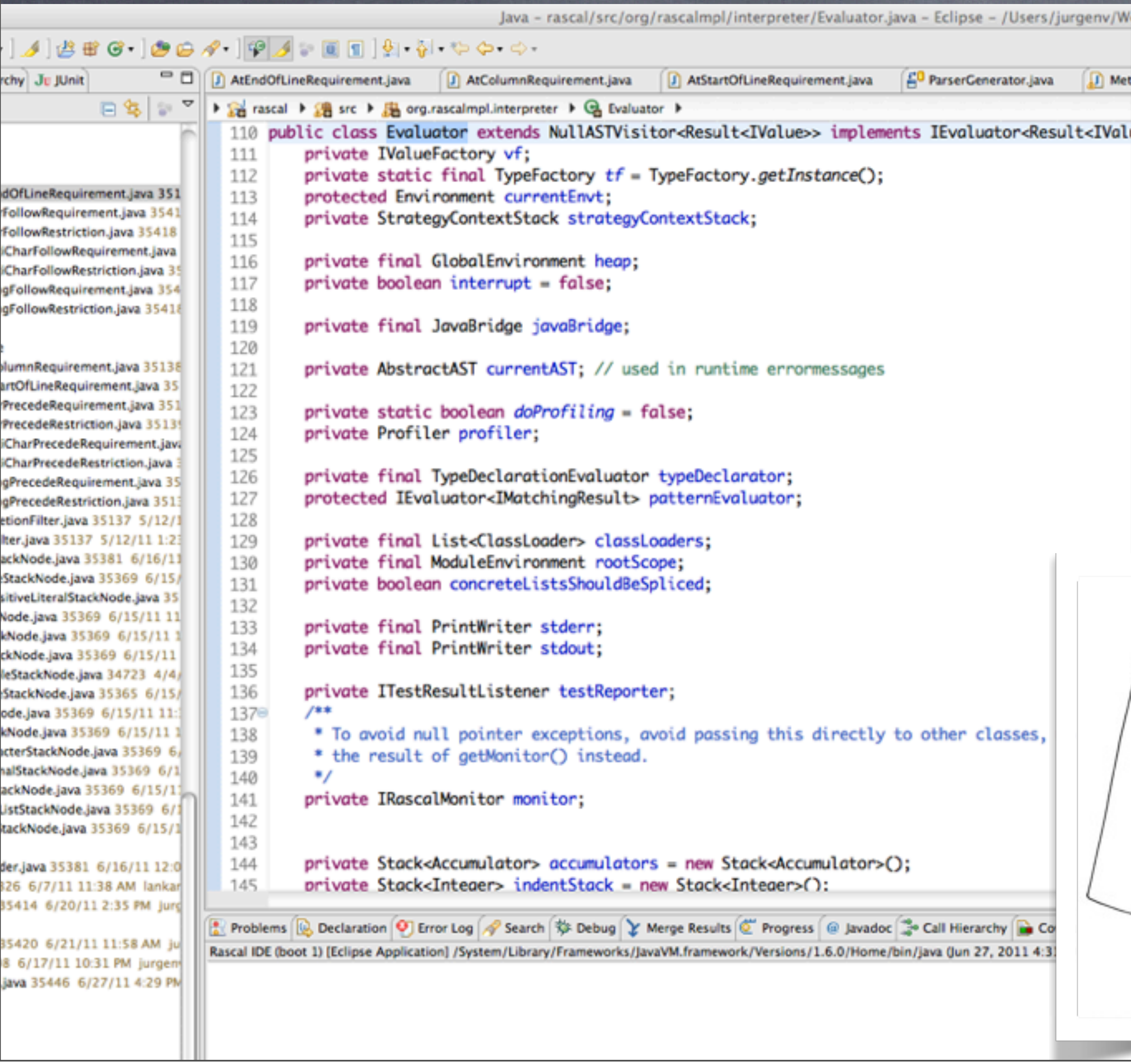


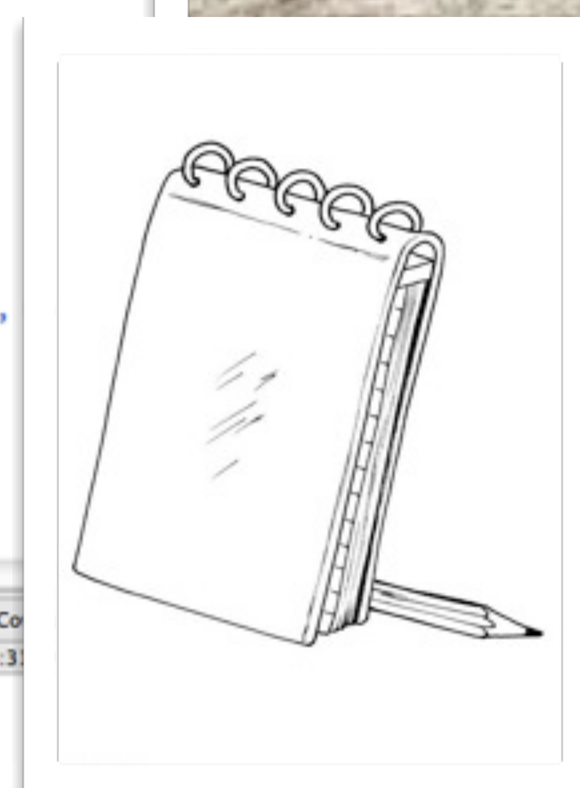
Traceability

Manual labor

Rascal to implement Visitor to Interpreter **refactoring**

Collecting data





Results

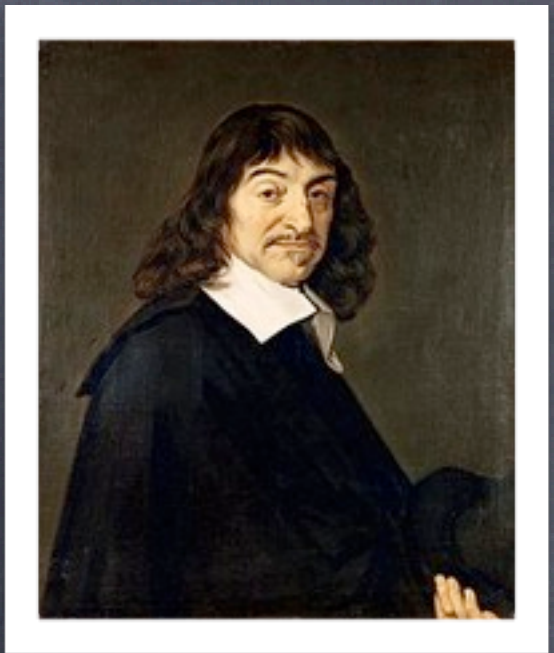
steps to add N constructs to **Visitor**
14 + 2N

steps to add N constructs to **Interpreter**
3N

S	**Visitor**	(COM)	**Interpreter**	(COM)	**Vis.>Int.**
S1	$ci^{11}(g^2a)^2)$	(18)	$m^2b(ef^2)^3(ga)^2$	(16)	yes
S1(N)	$ci^{11}(g^Na)^2)$	$(14+2N)$	$m^Nb(ef^N)^3(ga)^N$	$(4+6N)$	if $N \leq 2$
S1'(N,2)	$ci^{11}(g^Na)^2)$	$(14+2N)$	$m^N(ga)^N$	$(3N)$	if $N \leq 14$
S1'(N,M)	$ci^{9+M}(g^Na)^M$	$(10+NM+2M)$	$m^N(ga)^{MN}$	$(N+2MN)$	if $N \leq \frac{2M+10}{M+1}$
S2	i^2g^3iga	(8)	$i^2g^3gaig^3aiga$	(14)	no
S3	$dg^5egcg^{15}g^2a(eea)^4i^2h(ga)^3$	(43)	$d(ig)^2a(iga)^{15}(ig)^3gai$ $(ig^2)a(igg)^2anigaih(ga)^3$	(83)	no
S3'	$d(ga)^5egac(ga)^{15}(ga)^2$ $(eea)^4i^2h(ga)^3$	(70)	$d(ig)^2a(iga)^{15}(ig)^3gai$ $(ig^2)a(igg)^2anigaih(ga)^3$	(83)	no
S4	$mg^{11}a$	(13)	$bga(bga)^{11}$	(36)	[illegible]
S5	$biga$	(4)	bga	(3)	[illegible]

Table 2. A comparison of all maintenance programs (see Table 1).

break-even at N = 14



Why trust this?

Why trust this?

- **Construct** validity: are all aspects of maintainability observable in this experiment?

Why trust this?

- **Construct** validity: are all aspects of maintainability observable in this experiment?
- **Internal** validity: did you really do the best job possible in all scenarios?

Why trust this?

- **Construct** validity: are all aspects of maintainability observable in this experiment?
- **Internal** validity: did you really do the best job possible in all scenarios?
- **External** validity: does this say anything about the next interpreter I write in Java? The next maintenance? What if I don't use Eclipse? What if <blablabla>?

Why trust this?

other factors may still dominate, but that is why we compare two equivalent systems

- **Construct** validity: are all aspects of maintainability observable in this experiment?
- **Internal** validity: did you really do the best job possible in all scenarios?
- **External** validity: does this say anything about the next interpreter I write in Java? The next maintenance? What if I don't use Eclipse? What if <blablabla>?

Why trust this?

- **Construct** validity: are all aspects o maintainability observable in this experiment?

other factors may still dominate, but that is why we compare two equivalent systems

- **Internal** validity: did you really do t job possible in all scenarios?

there is no proof of that - we invite you to reproduce or invalidate the results

- **External** validity: does this say anything about the next interpreter I write in Java? The next maintenance? What if I don't use Eclipse? What if <blablabla>?

Why trust this?

- **Construct** validity: are all aspects o
maintainability observable in this experiment?

other factors may still dominate, but that is why we compare two equivalent systems

- **Internal** validity: did you really do t
job possible in all scenarios?

there is no proof of that - we invite you to reproduce or invalidate the results

- **External** validity: does this say anything
about the next interpreter I write i
The next maintenance? What if I do
Eclipse? What if <blablabla>?

we do **not** know

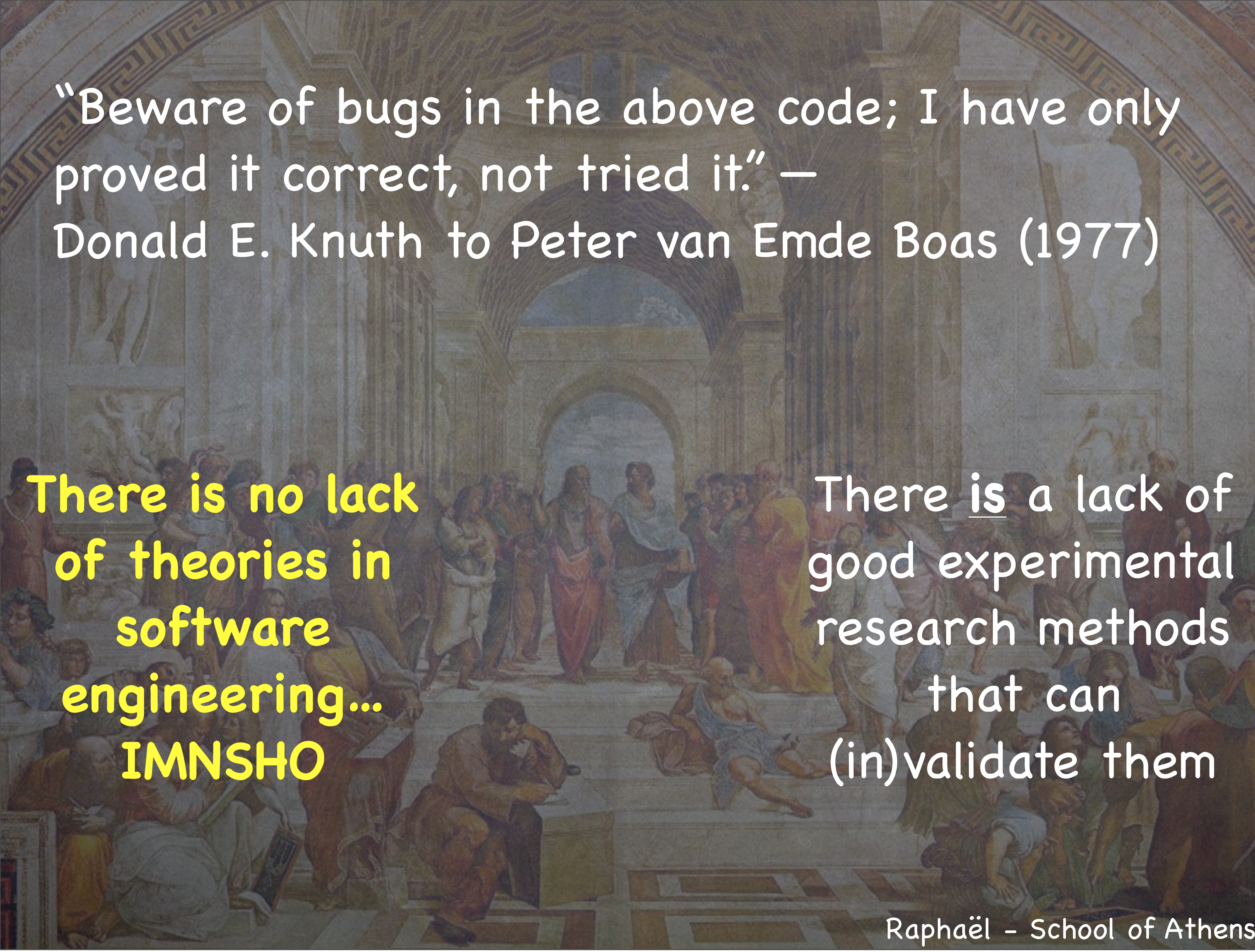
The role of Rascal

- Integration with Eclipse Java front-end
- Relations and trees to model abstract facts about Java
- Pattern matching and visit to analyze these models checking conditions
- Templates to generate new code
- < two man-weeks of work
- Opportunity to do it again, and again, and again!

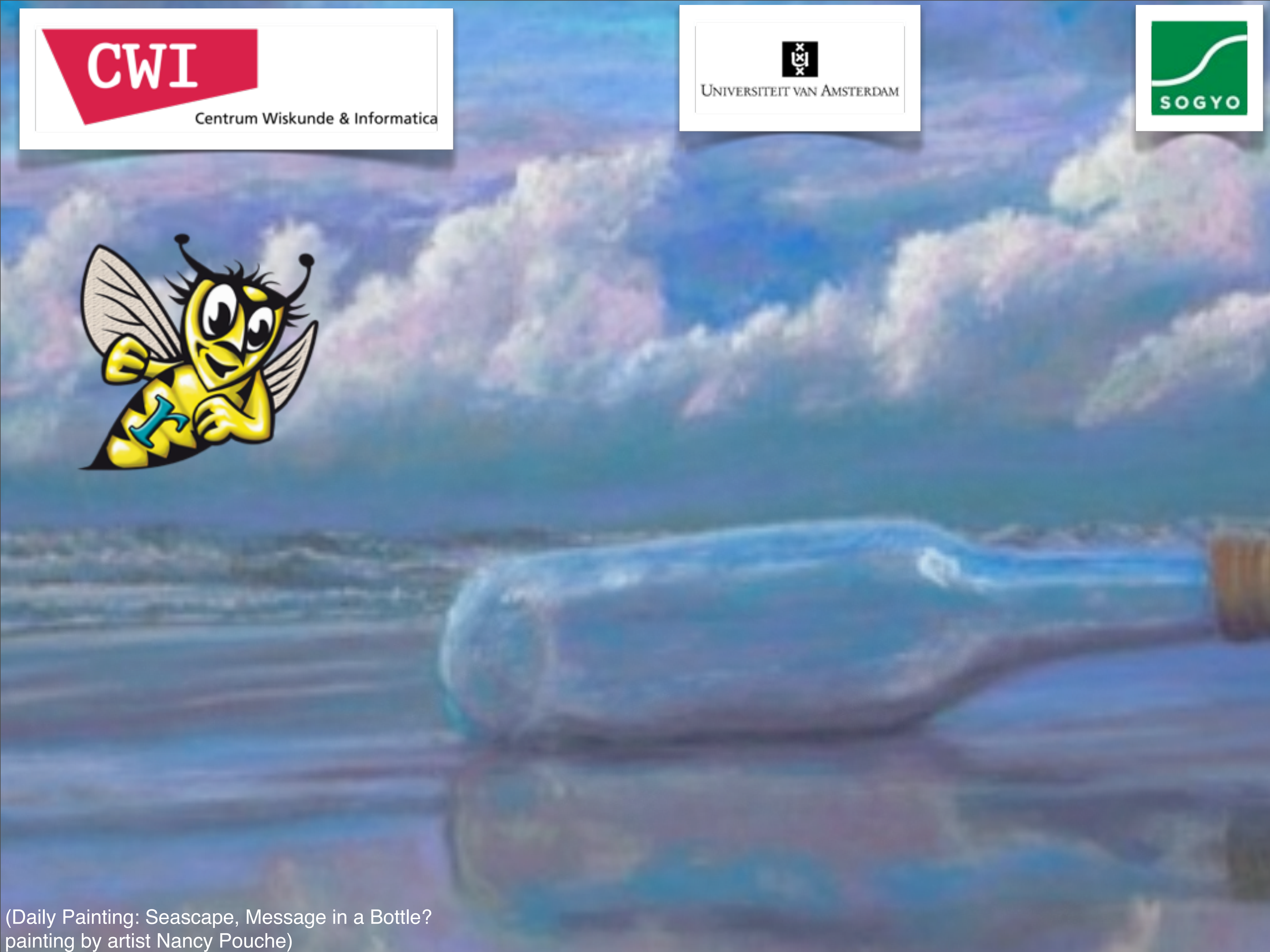
"Beware of bugs in the above code; I have only proved it correct, not tried it." —
Donald E. Knuth to Peter van Emde Boas (1977)

There is no lack of theories in software engineering... IMNSHO

There **is** a lack of good experimental research methods that can (in)validate them



Raphaël - School of Athens



(Daily Painting: Seascape, Message in a Bottle?
painting by artist Nancy Pouche)

CWI Centrum Wiskunde & Informatica

Universiteit van Amsterdam

SOGYO

(Daily Painting: Seascape, Message in a Bottle?
painting by artist Nancy Pouche)

CWI Centrum Wiskunde & Informatica

Universiteit van Amsterdam

SOGYO

http://www.cwi.nl/sen1

http://www.rascal-mpl.org

http://ask.rascal-mpl.org

http://tutor.rascal-mpl.org

Rascal is a domain specific programming language for software tools

(Daily Painting: Seascape, Message in a Bottle? painting by artist Nancy Pouche)

CWI
Centrum Wiskunde & Informatica

Universiteit van Amsterdam

SOGYO

http://www.cwi.nl/sen1

http://www.rascal-mpl.org

http://ask.rascal-mpl.org

http://tutor.rascal-mpl.org

Rascal is a domain specific programming language for software tools

Master internships
Funding opportunities
Visit CWI

(Daily Painting: Seascape, Message in a Bottle? painting by artist Nancy Pouche)

CWI
Centrum Wiskunde & Informatica

Universiteit van Amsterdam

SOGYO

http://www.cwi.nl/sen1

http://www.rascal-mpl.org

http://ask.rascal-mpl.org

http://tutor.rascal-mpl.org

Rascal is a domain specific programming language for software tools

Master internships
Funding opportunities
Visit CWI

mailto:Jurgen.Vinju@cwi.nl

twitter:@jurgenvinju

(Daily Painting: Seascape, Message in a Bottle? painting by artist Nancy Pouche)