SEN1:SWAT

ATEAMS

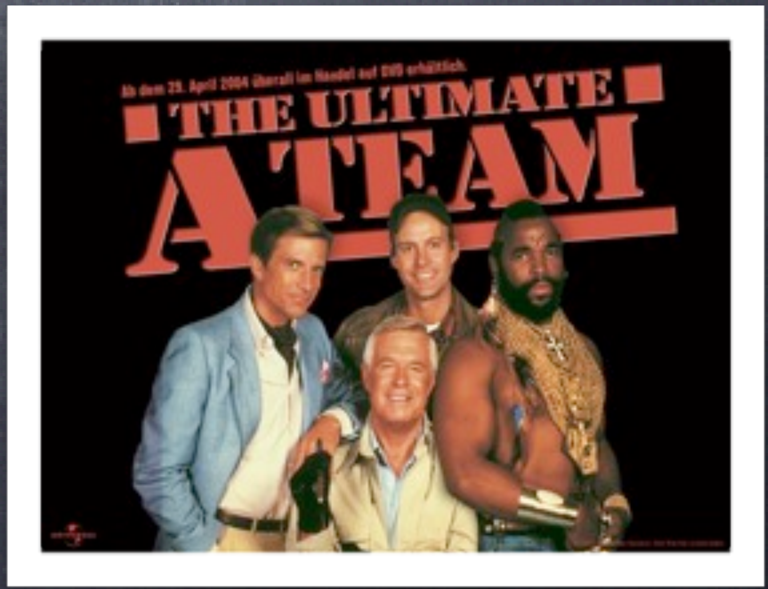
# Introducing Rascal for meta programming
# and
# Eyeballing the Cyclomatic Complexity Metric

Jurgen Vinju
@RMOD, INRIA Lille
May 11th 2012

# CWI SWAT ≡ INRIA ATEAMS

- **S**oft**W**are **A**nalysis and **T**ransformation
  - Meta programming & DSLs
  - Parsing, Term Rewriting
  - ASF+SDF, Rascal, ToolBus, ATerm

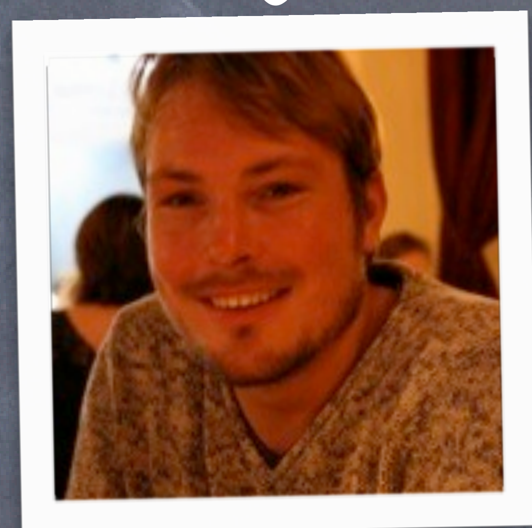
- **A**nalysis and **T**ransformation based on r**E**li**A**ble tool co**M**position**S**
  - RScript, Rascal
  - Eclipse IMP



# Rascals

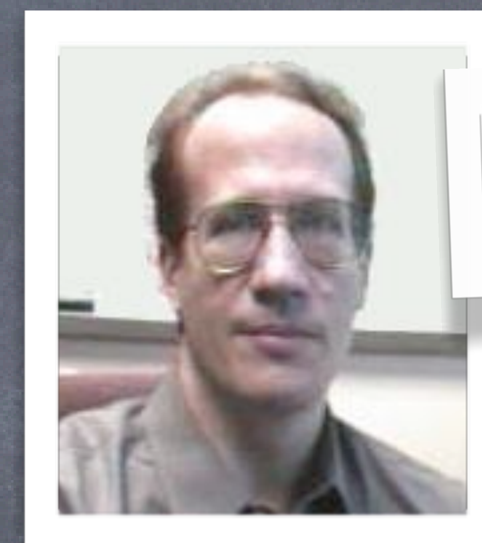
Paul Klint

Jurgen Vinju



Tijs v/d Storm

Bob Fuhrer



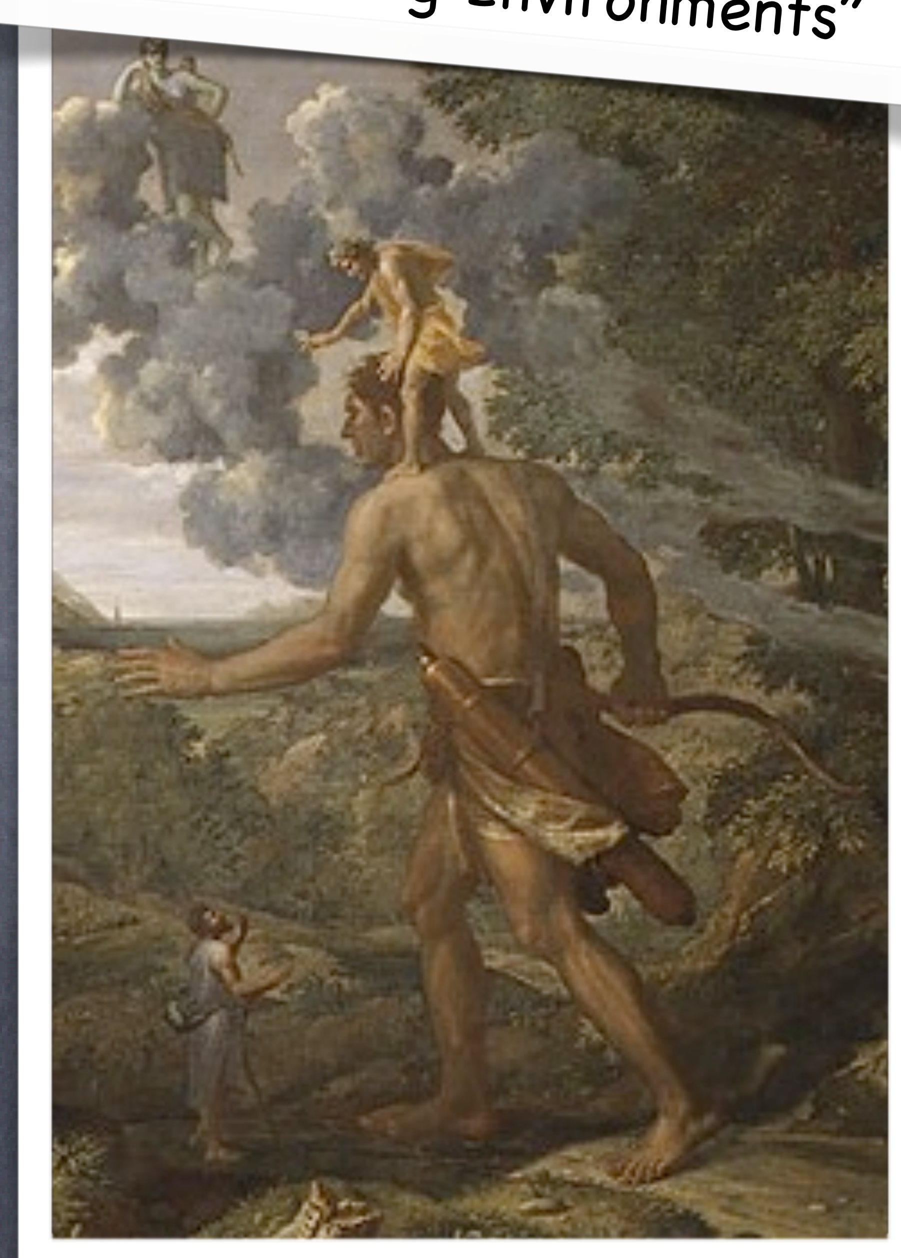
IMP

# Credits

How Tijs & I were drafted...

- Esprit: GIPE I & GIPE II (90's)
- ASF+SDF Meta-Environment (00's)
- Eclipse
- **I**DE **M**eta Tooling **P**latform (IMP)
- Rascal is a part of IMP now
- Rascal draws inspiration from countless other projects (see SCAM 2009 paper for references)

"Generation of Interactive Programming Environments"



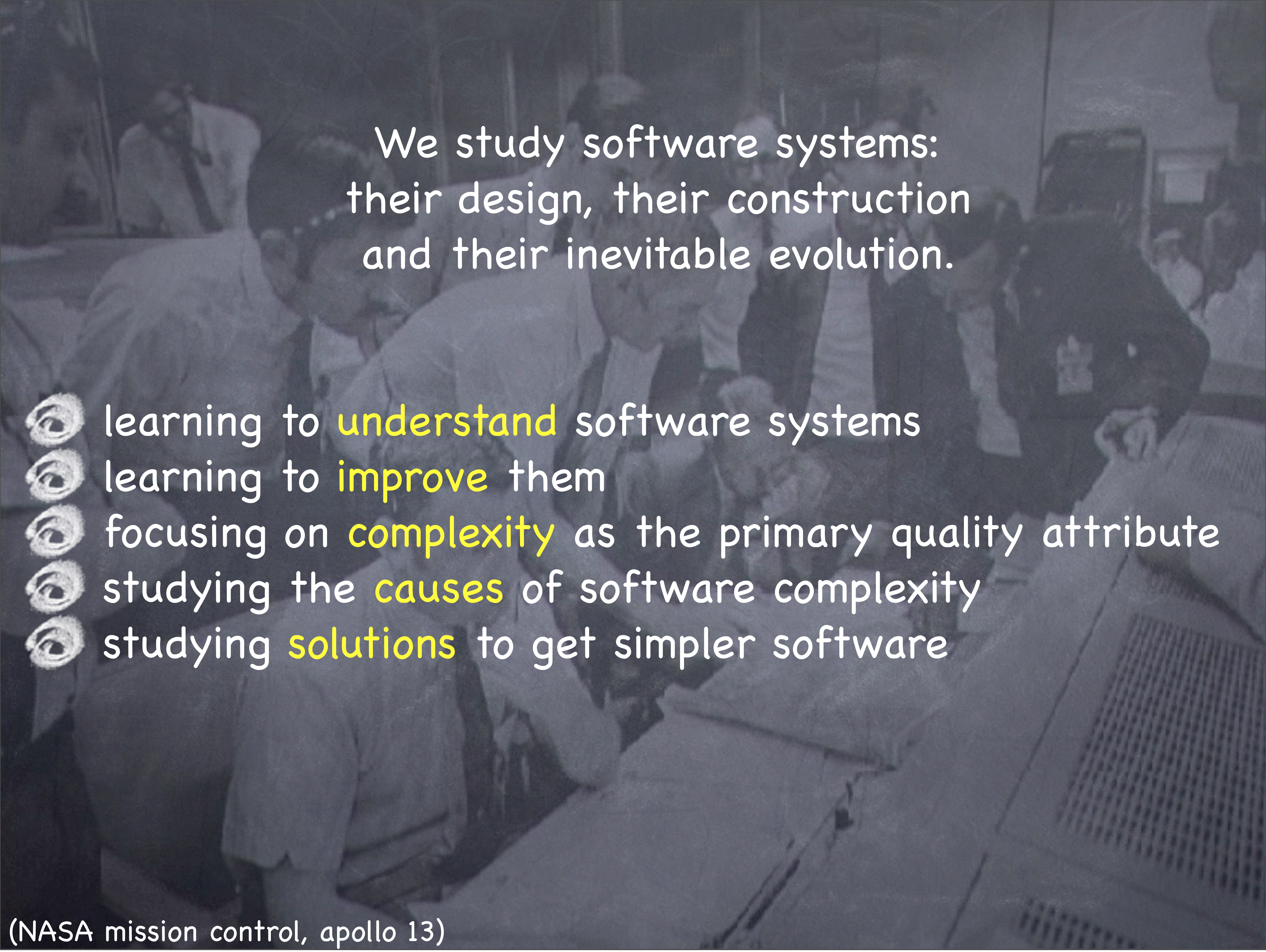
# Why?

- 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? (one example)

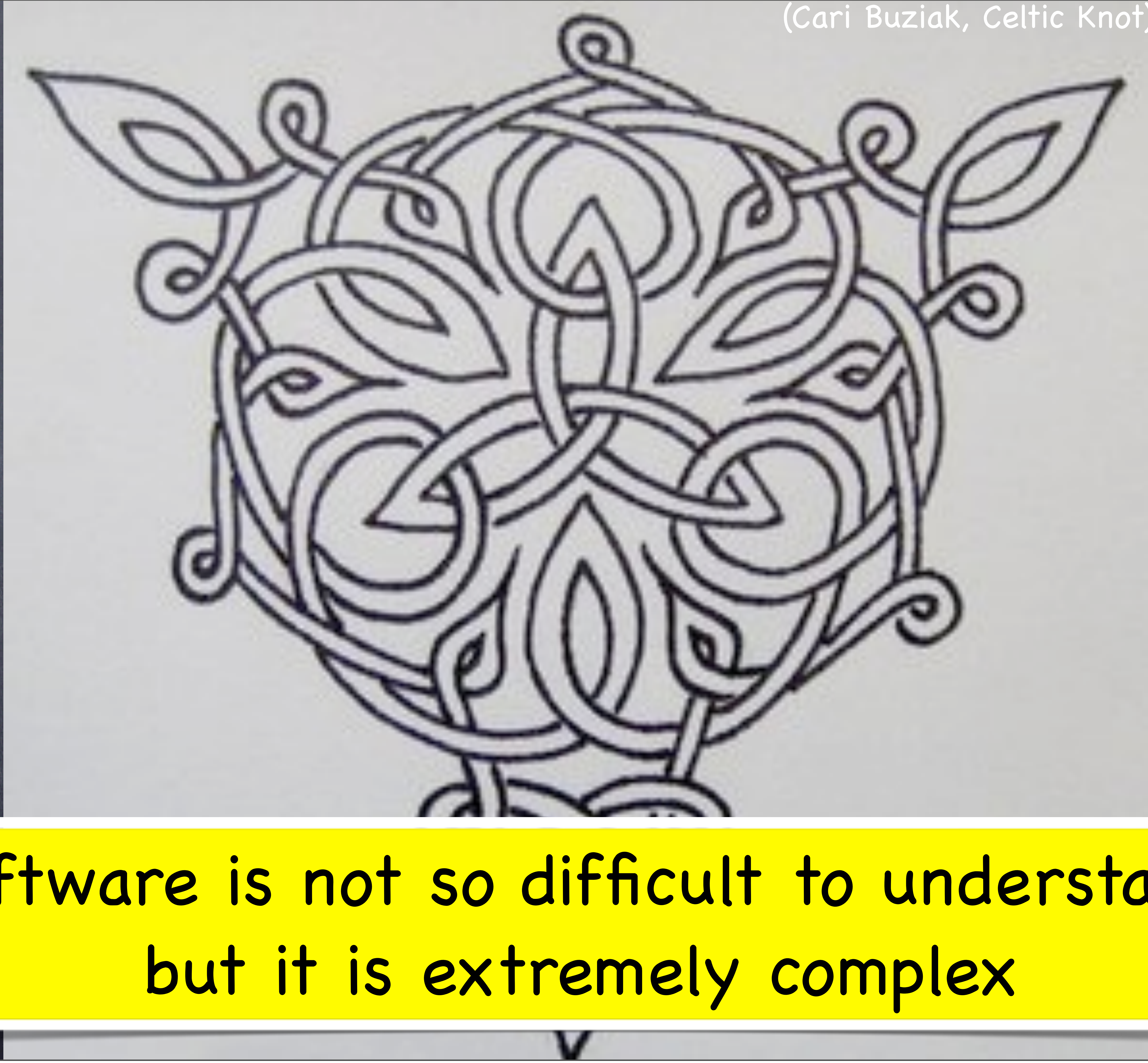
(¬How)



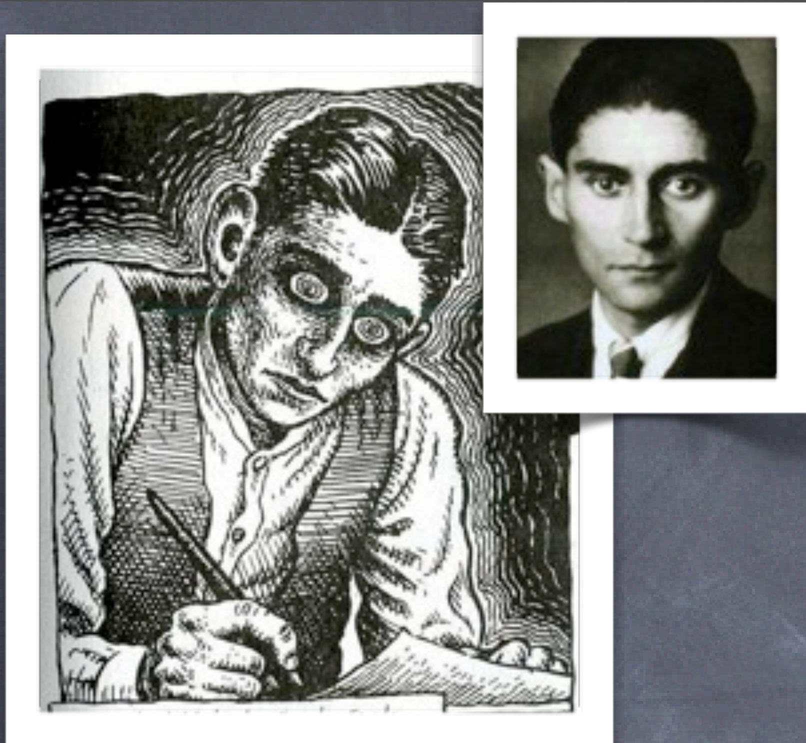
We study software systems:
their design, their construction
and their inevitable evolution.

- 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

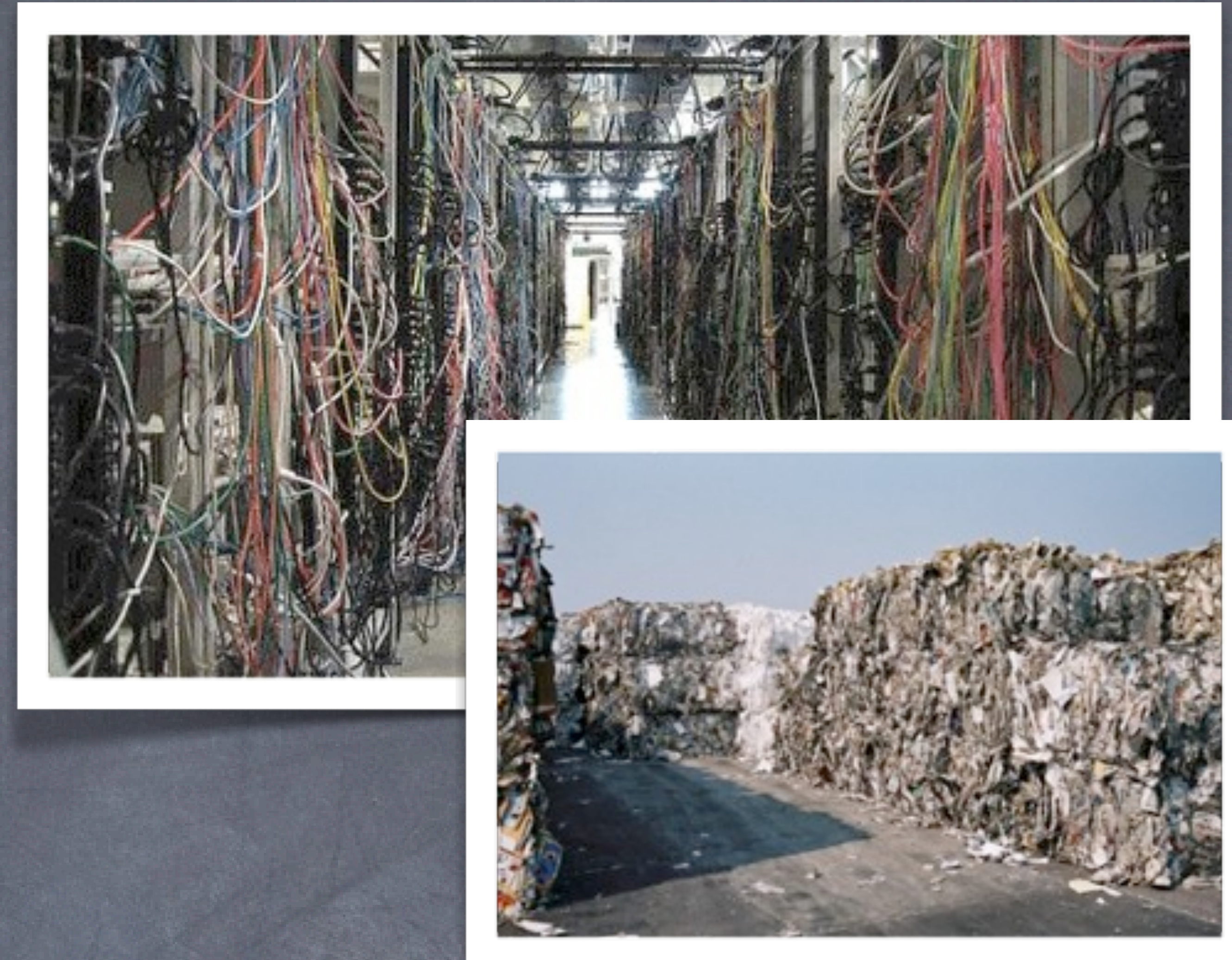
(NASA mission control, apollo 13)




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”

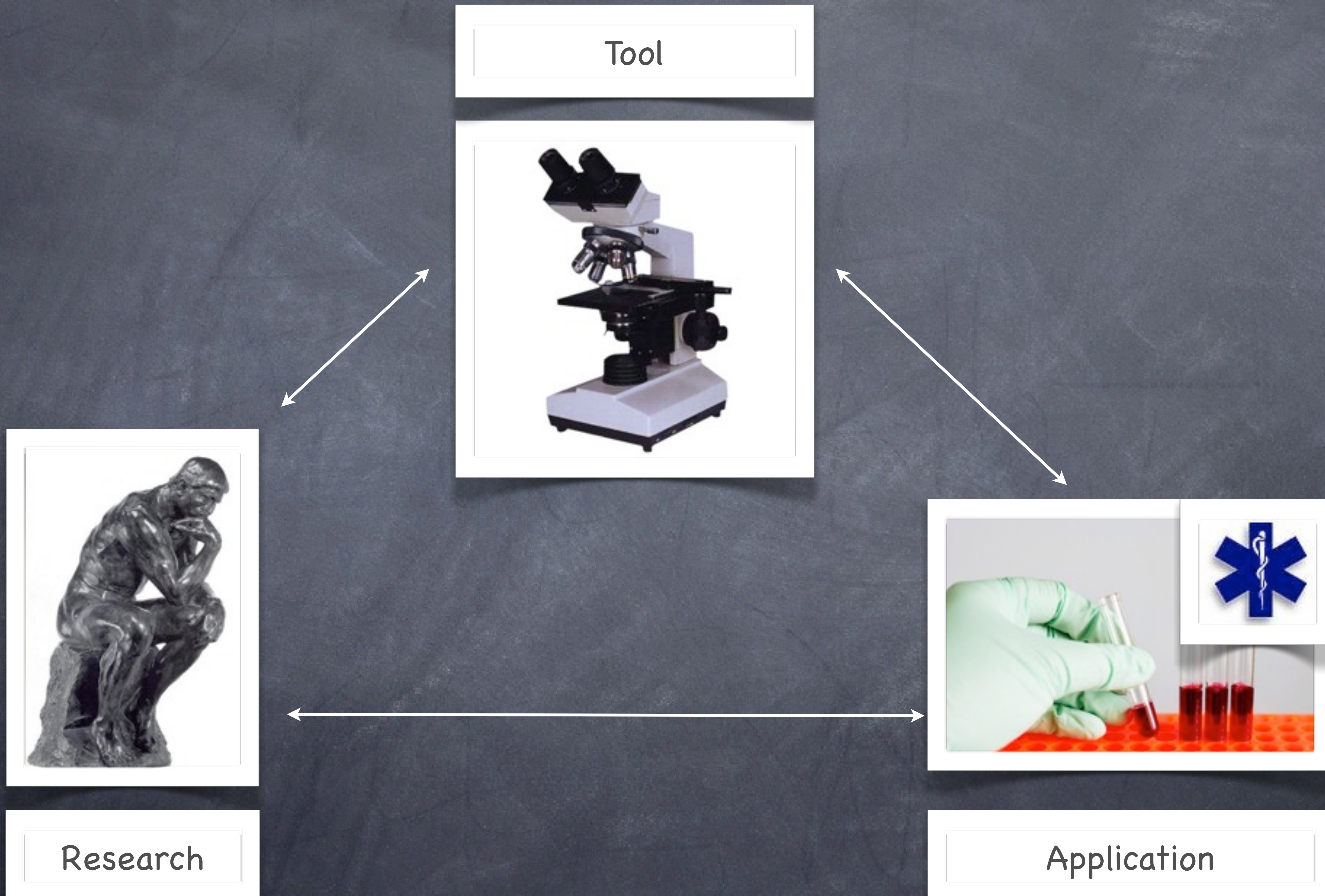
3894 lines

367 ifs

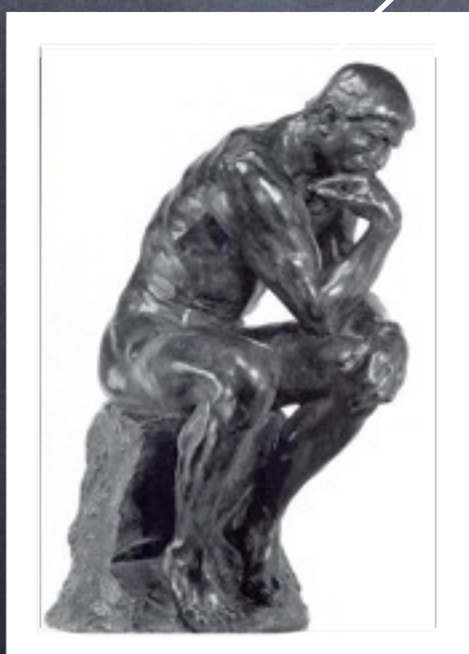
174 cases

# Size does matter

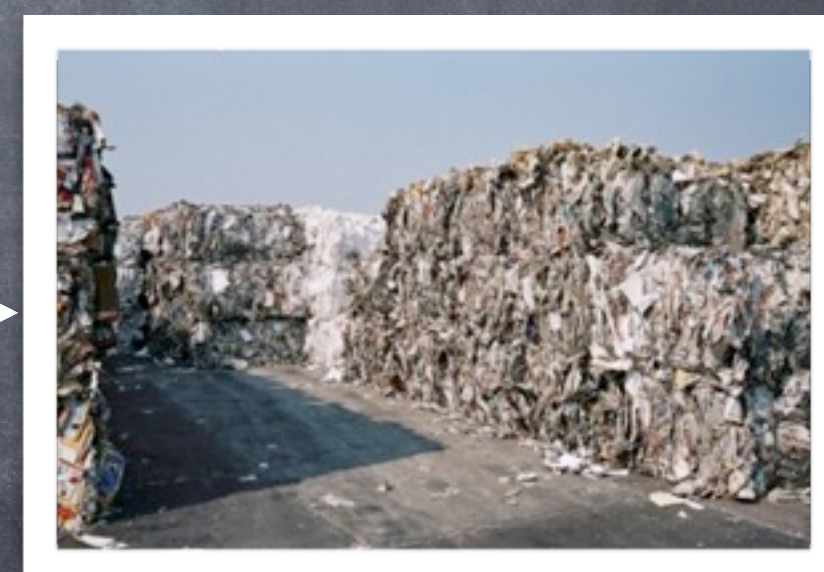
- A normal company may own $3 \times 10^{10}$ lines of code - 750,000,000 single column pages.
- It goes a few times around the globe, if printed.
- At 1 minute per page (?) that might take approximately 1427 years to read.
- Ergo, nobody has ever understood it, or will ever fully understand it.


Tool
Research
Application

Tool

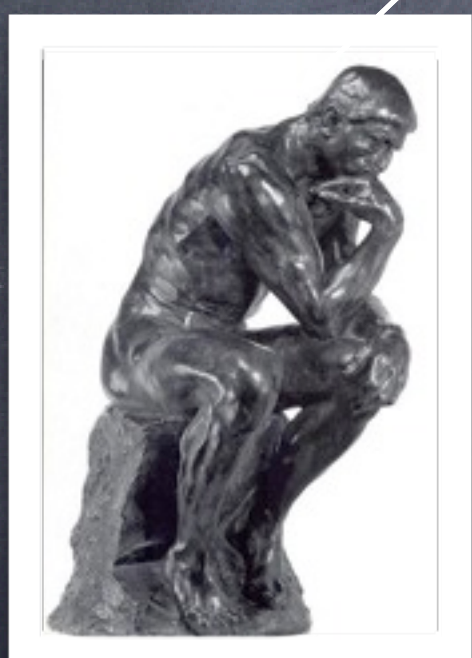


Research

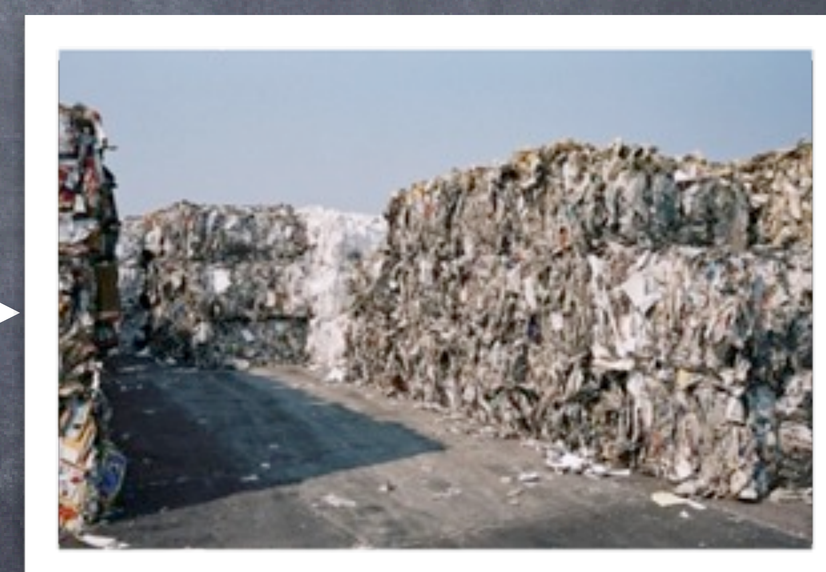


Application

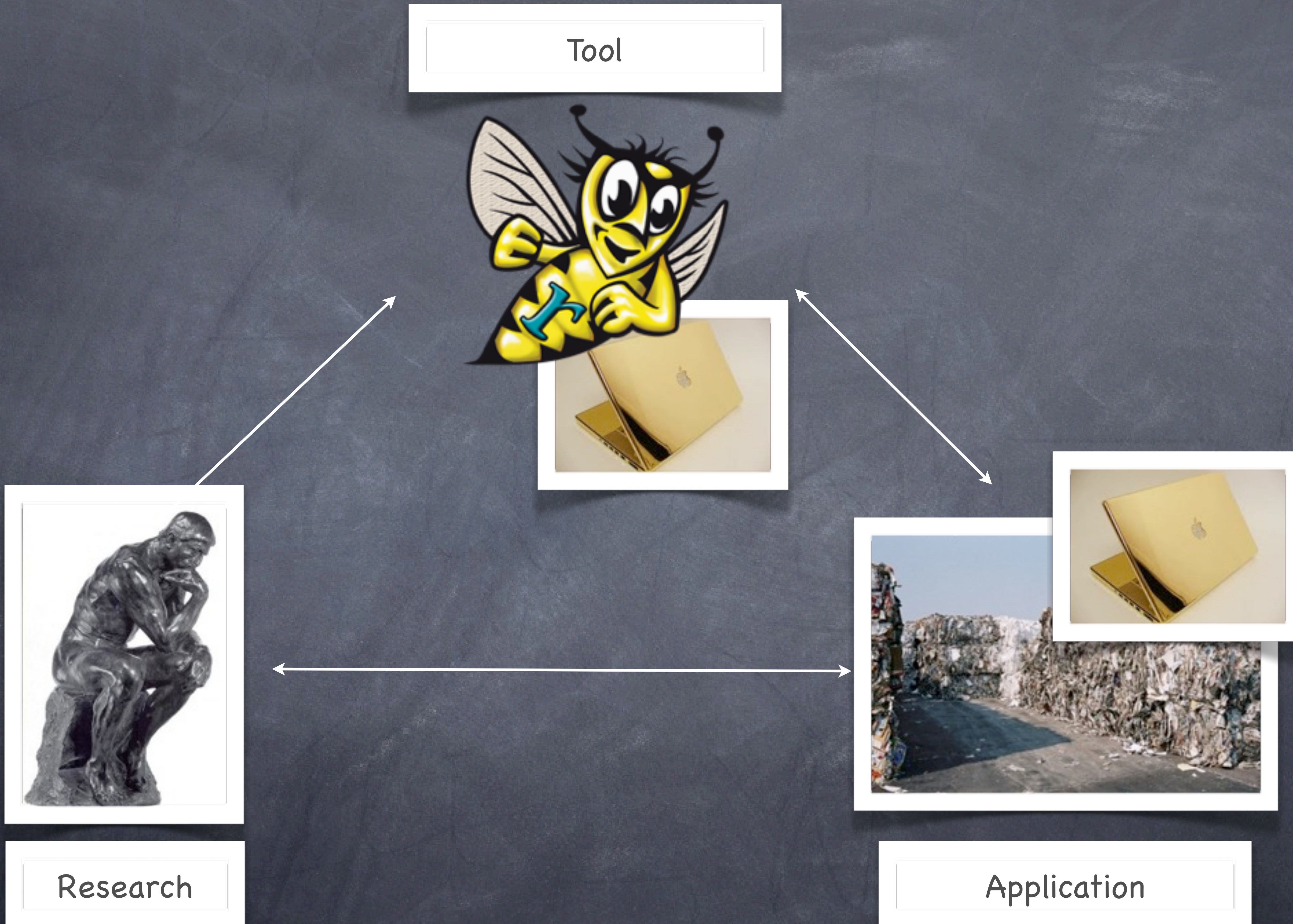
Tool



Research



Application

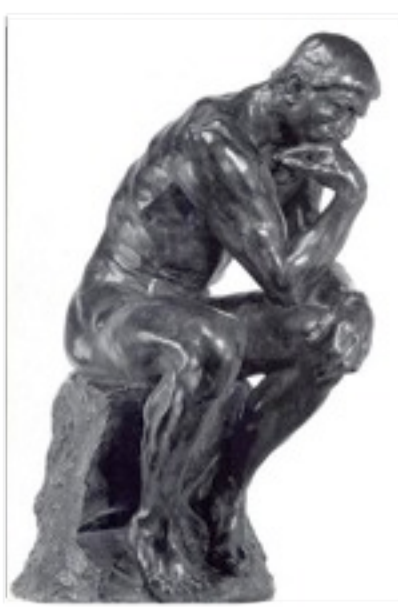

Tool
Research
Application

# Why we need Rascal @CWInl

Meta Tool

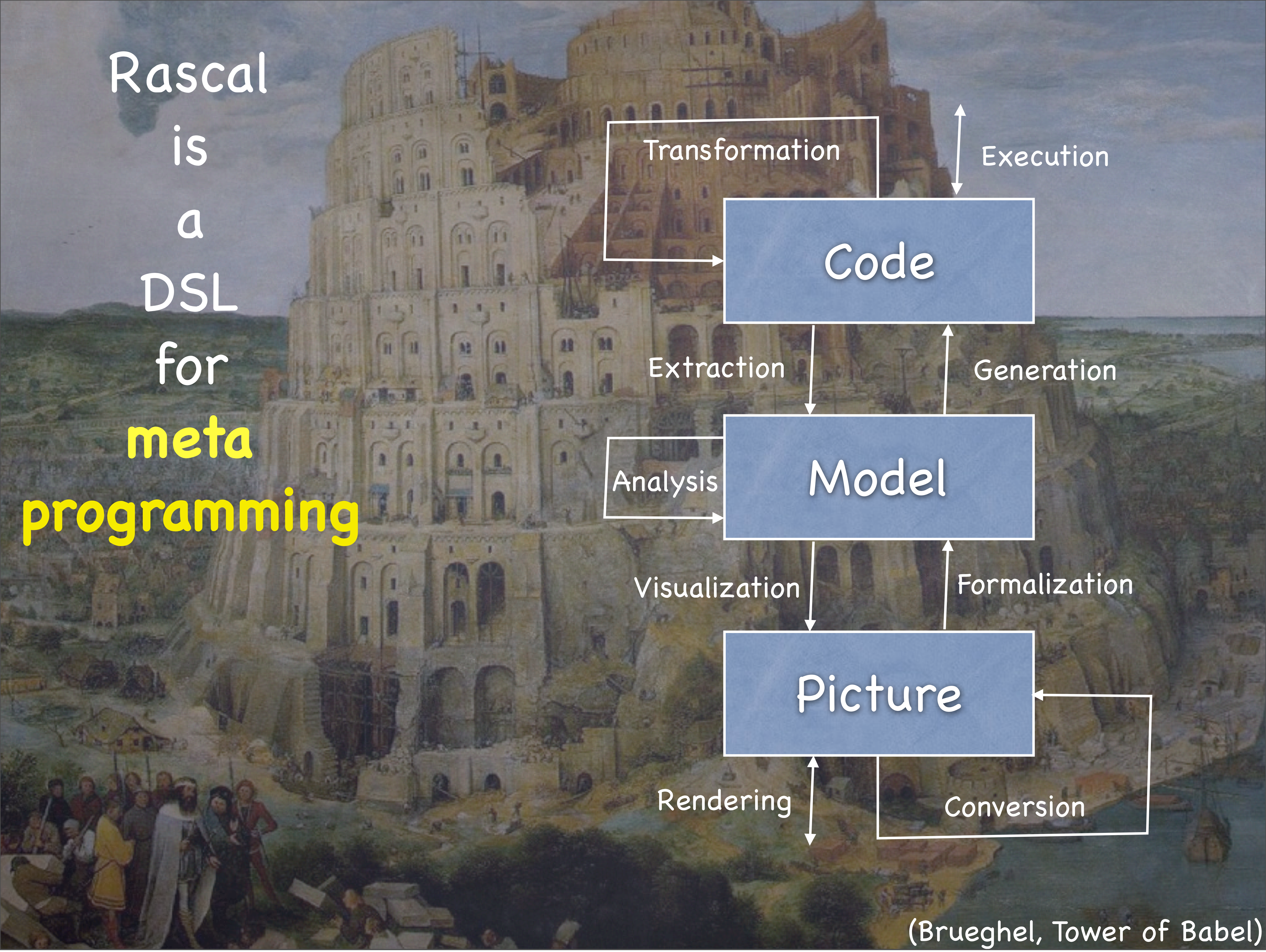
Tools

Every week
a new tool
a new DSL

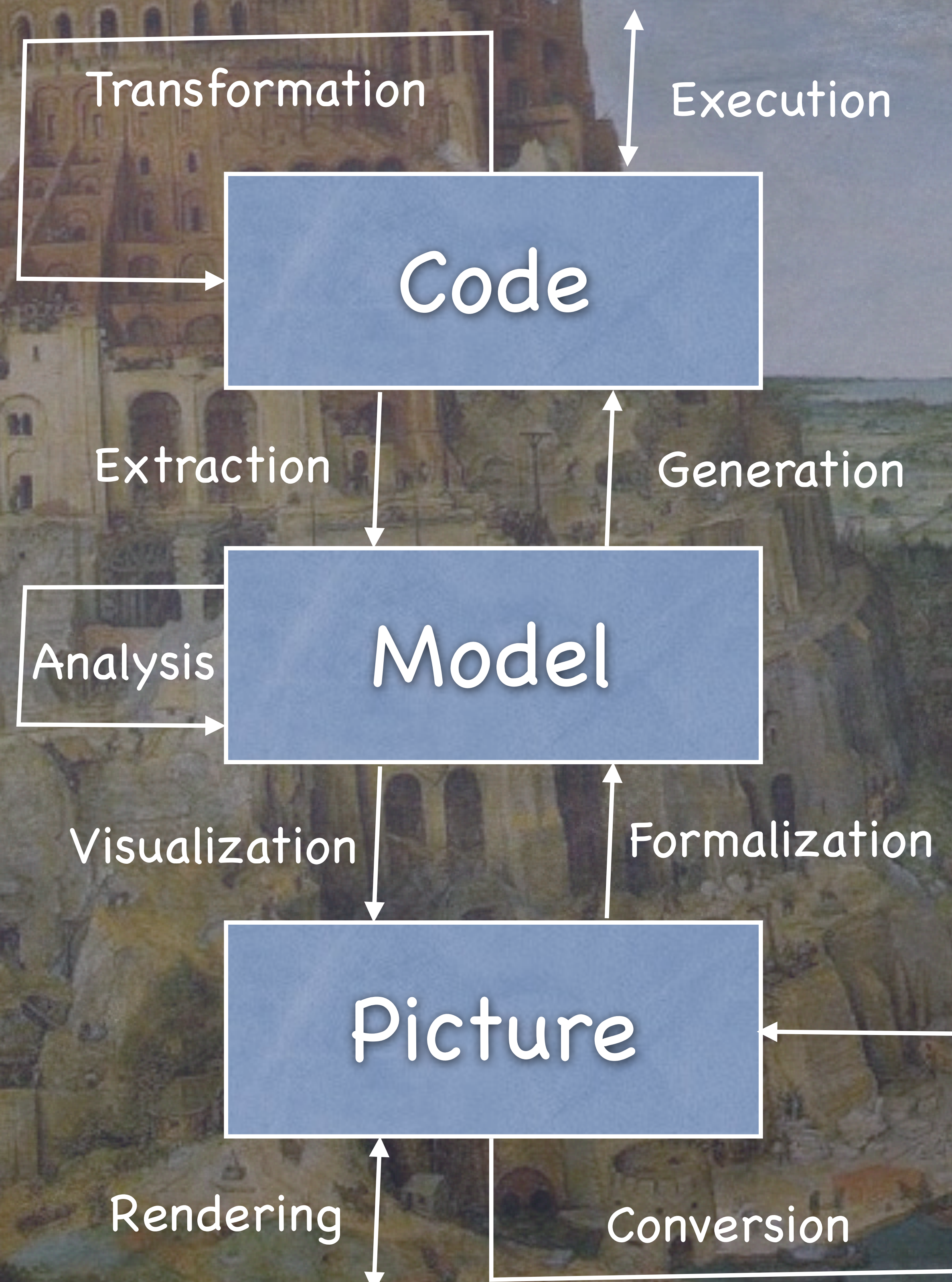


Research

Application


Rascal is a DSL for meta programming
Transformation
Execution
Code
Extraction
Generation
Analysis
Model
Visualization
Formalization
Picture
Rendering
Conversion
(Brueghel, Tower of Babel)

Rascal
is
a
DSL
for
**meta**
**programming**
=
moving
between
representations
of
source code



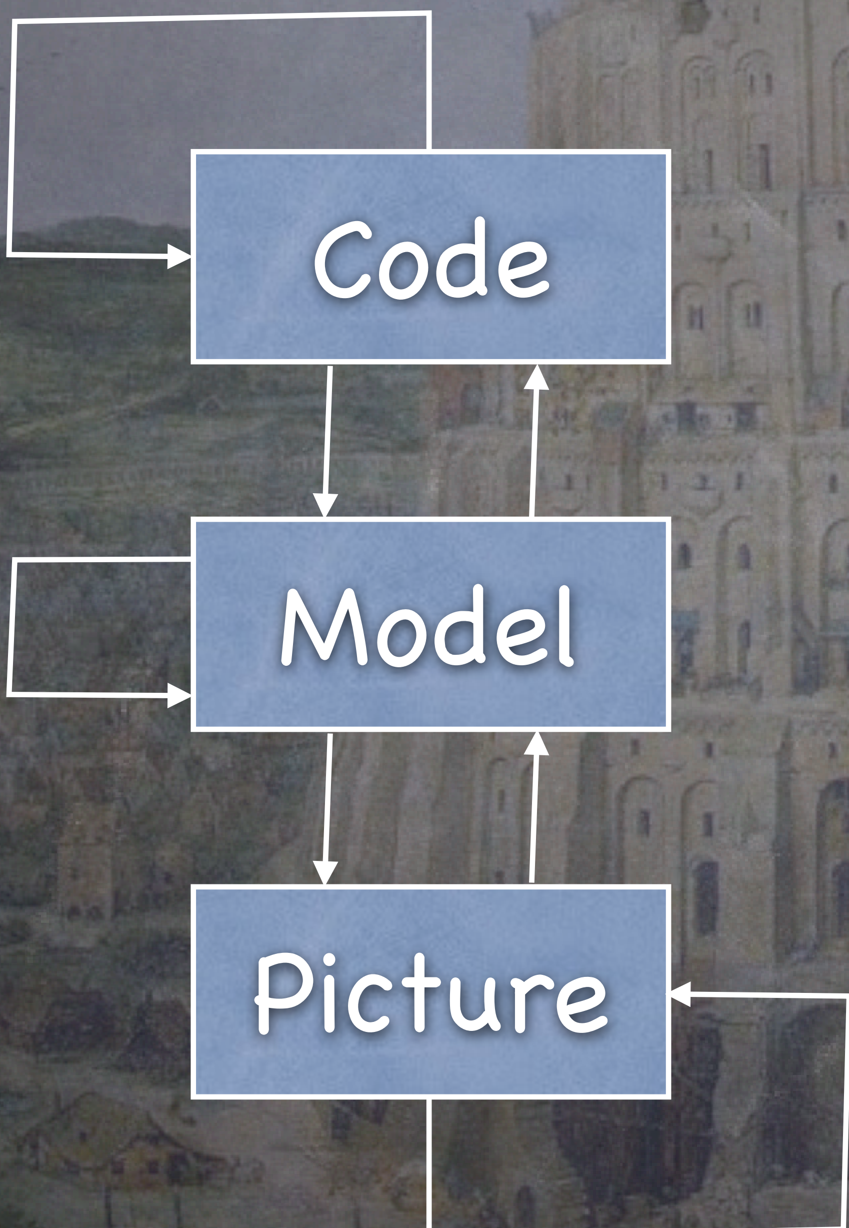

(Brueghel, Tower of Babel)

Rascal is/will be
a “ONE-STOP-SHOP”

for

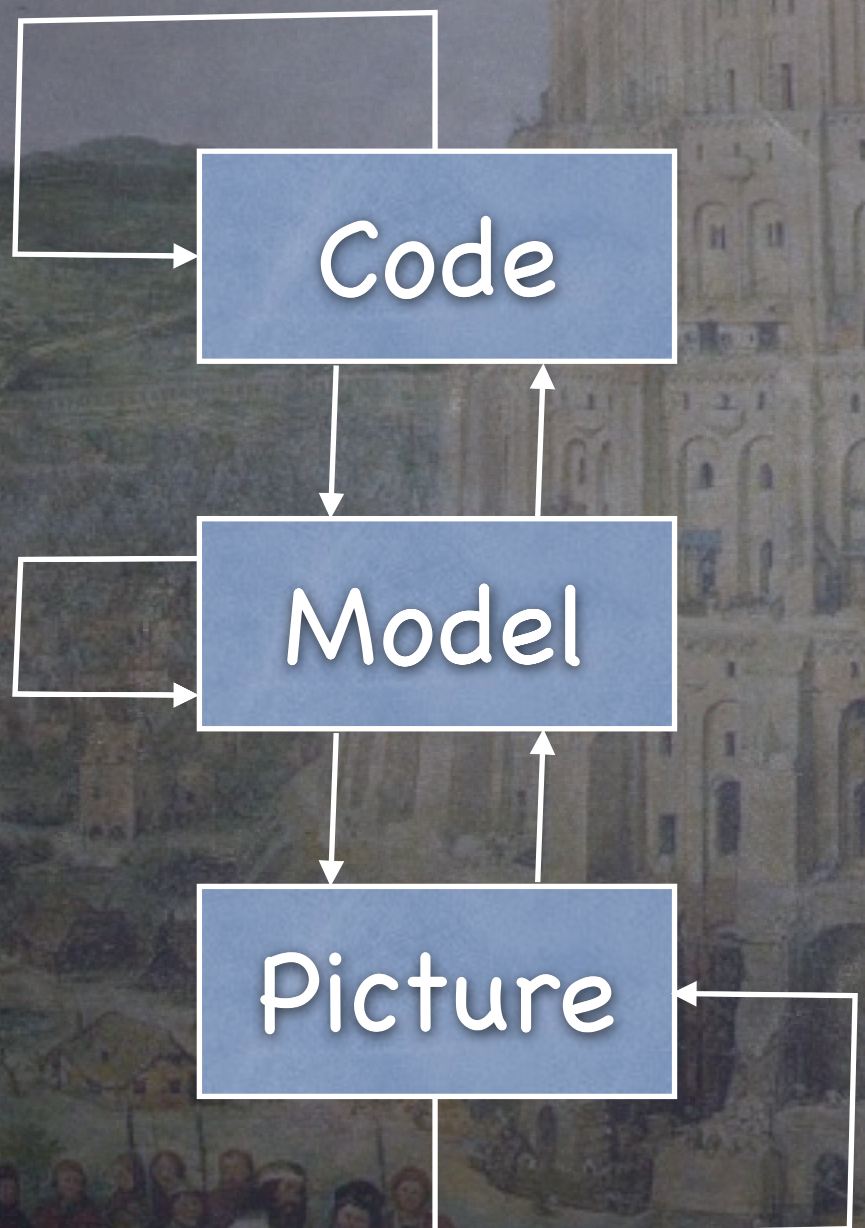
analysis
transformation
generation
visualization
IDE construction
etc.
(meta programming)

# 3 Meta Software Challenges




1: Diversity

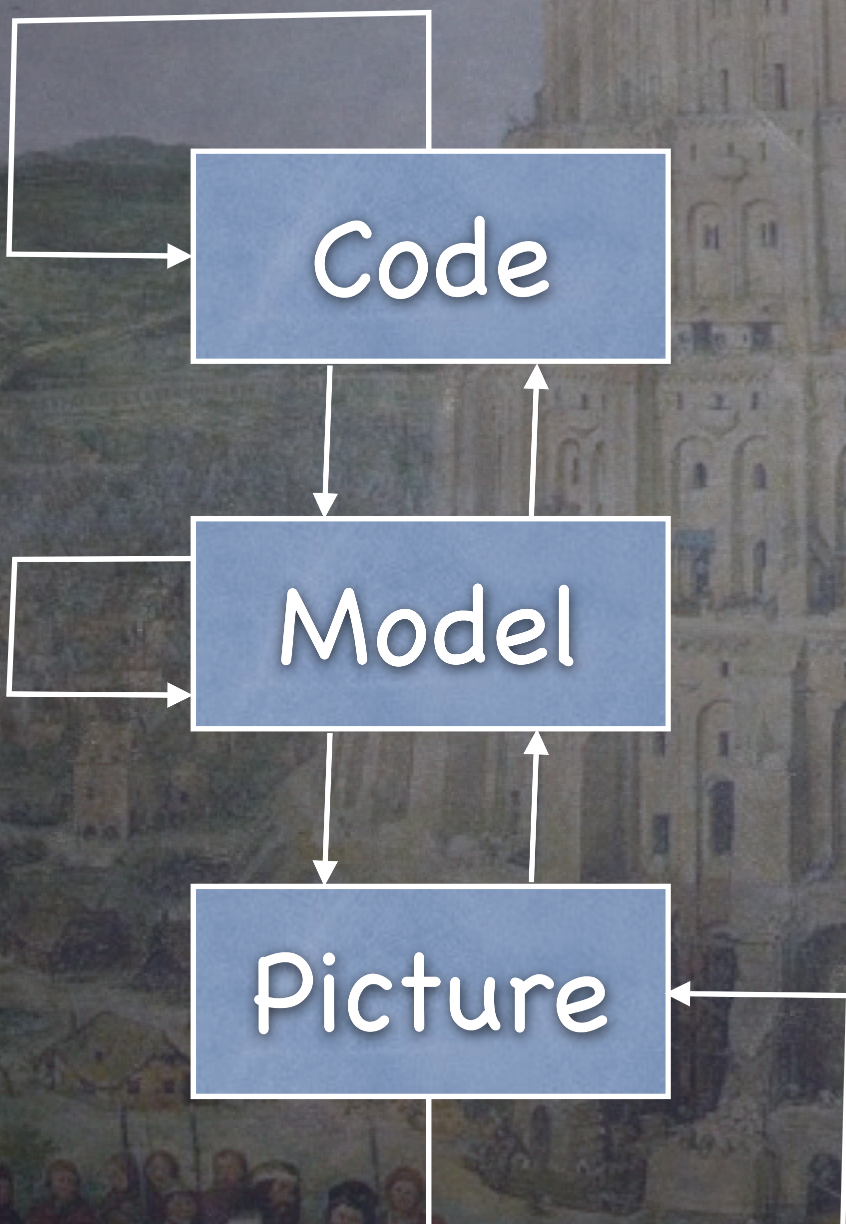
# 3 Meta Software Challenges




(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

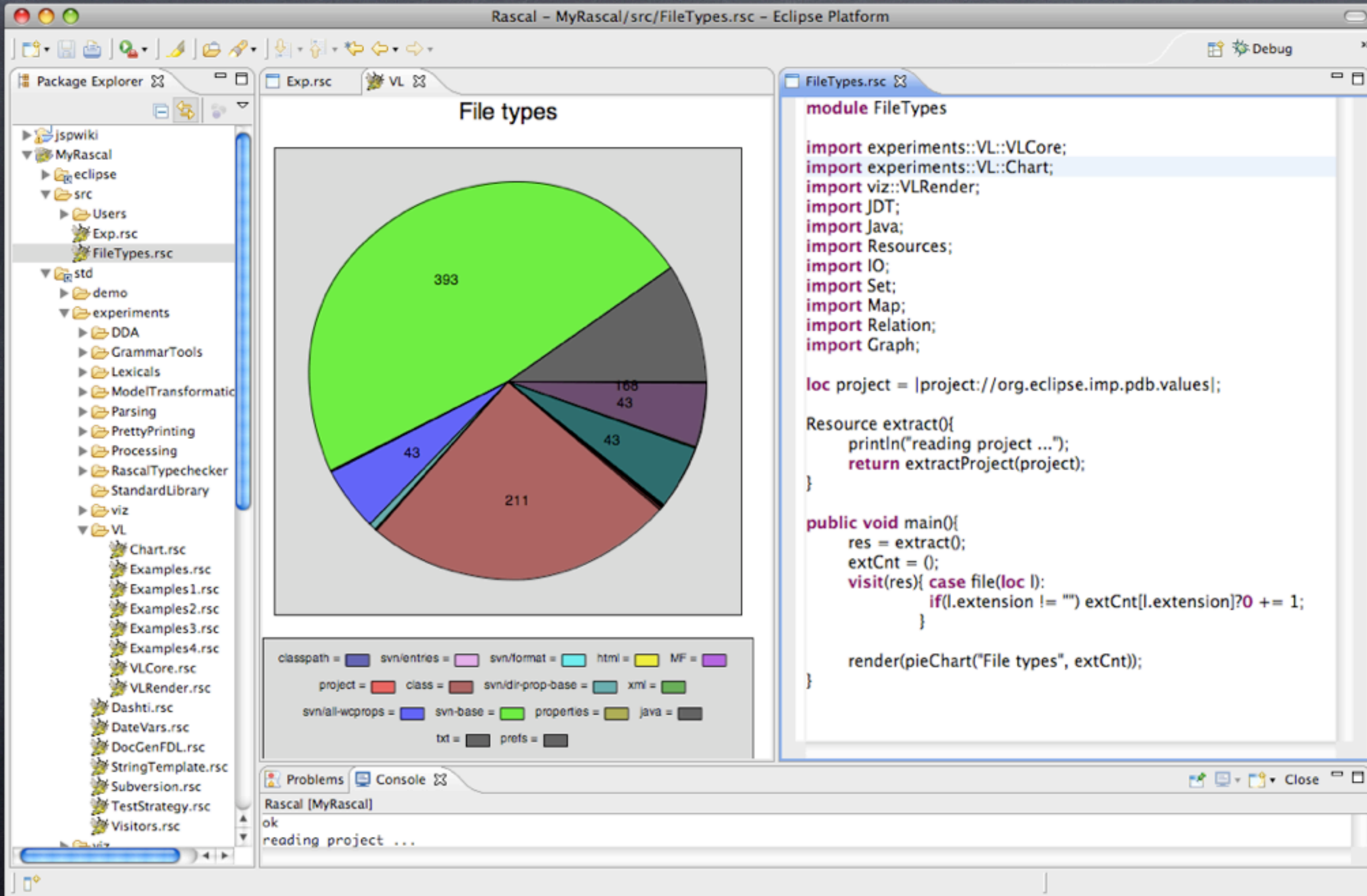
Syntax definition

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 ...

# Get more detail

- http://www.rascal-mpl.org
- http://tutor.rascal-mpl.org
- http://ask.rascal-mpl.org
- GTTSE 2009; SCAM 2009; FTFJP 2010; TOOLS 2011; SLE 2011; ICSE 2011;

Rascal is a domain specific programming language for software research

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

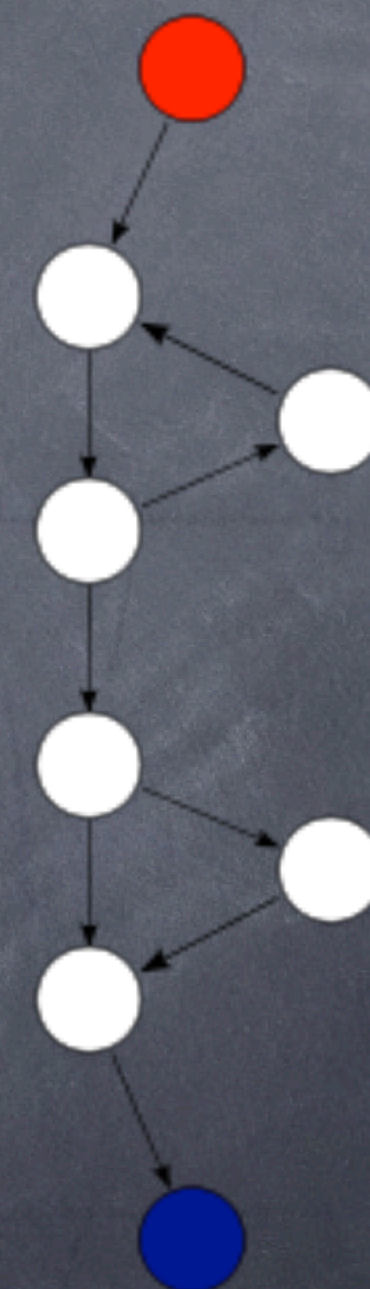
# Eyeballing Cyclomatic Complexity Metric

- Ongoing work with Mike Godfrey
- Typical application of Rascal
- Submitted to SCAM (Very cool Working Conference on Source Code Analysis and Manipulation, http://www.ieee-scam.org

# Cyclomatic Complexity

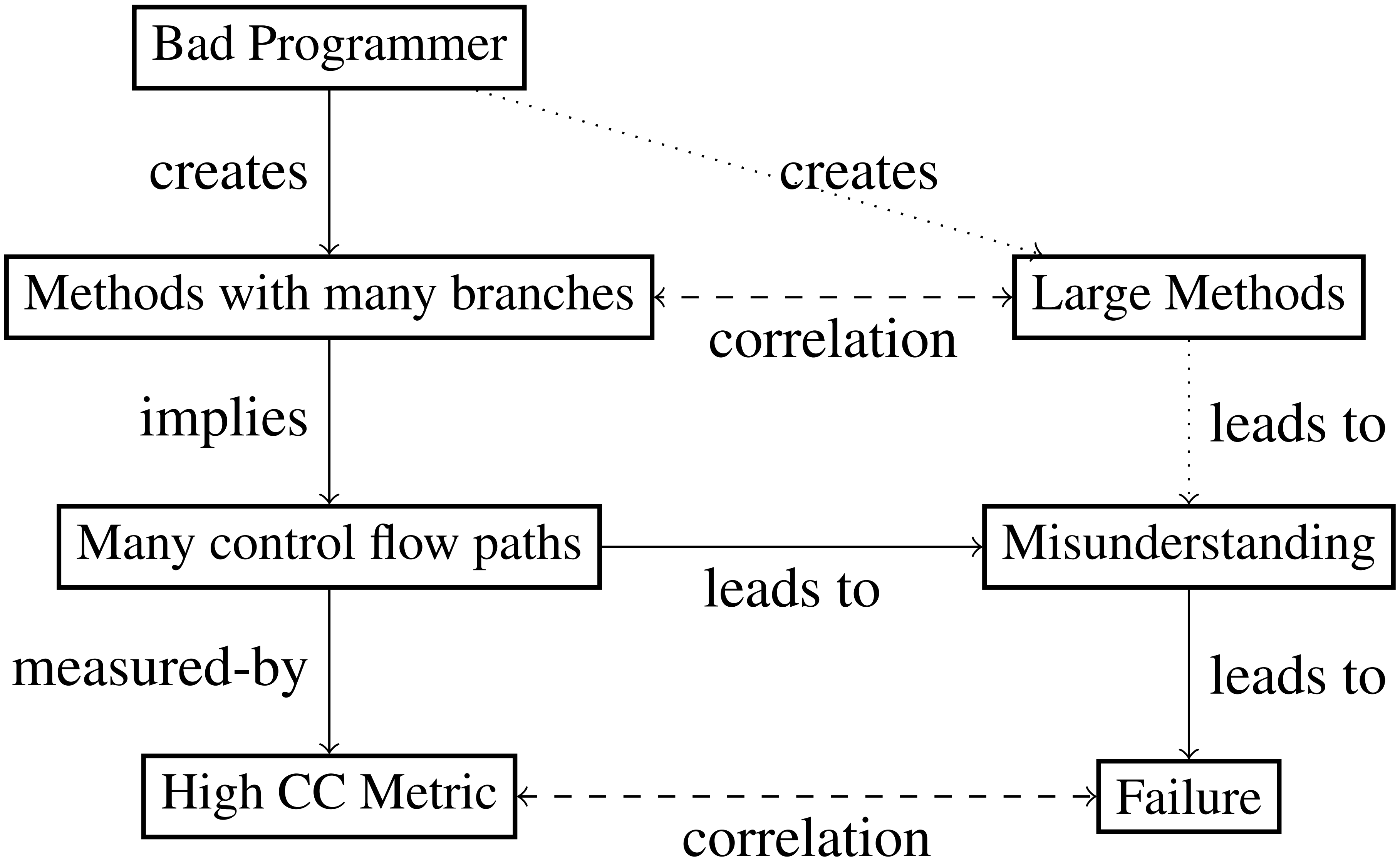
- Simple metric
- More and more popular
- Finding “complex” code
- It is a metric!
- But what does it measure?

# McCabe Cyclomatic Complexity



- Is defined on the control flow graph of a procedure/method/unit of code
- Measures the number of linear independent paths through the code
- Upperbound for the number of tests at least needed
- Indicate understandability because ...

[wikipedia]


Bad Programmer
creates
creates
Methods with many branches
correlation
Large Methods
implies
leads to
Many control flow paths
leads to
Misunderstanding
measured-by
leads to
High CC Metric
correlation
Failure

# But...

```
  i = 0;
  goto body;
loop:
  if (i == 10)
    goto done;
  i++;
body:
  print(i);
  goto loop;
done:
```

```
i = 0;
do
  print(i);
while (i++ != 10);
```

# And…

```
switch(⊥) {
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
}
```

# Questions

- What is the basis of CC measuring understandability?
- What does control flow look like in the first place?
- Initial questions
  - how many times does CC make methods look more complex than they really are? (false positives)
  - how many times does CC make methods look simpler than they really are? (false negatives)

# Problem

- There are too many methods in the world to study
- How can we generalize over something that is so utterly diverse?

# Idea: control flow patterns

- Use a meta programming solution! (of course!)

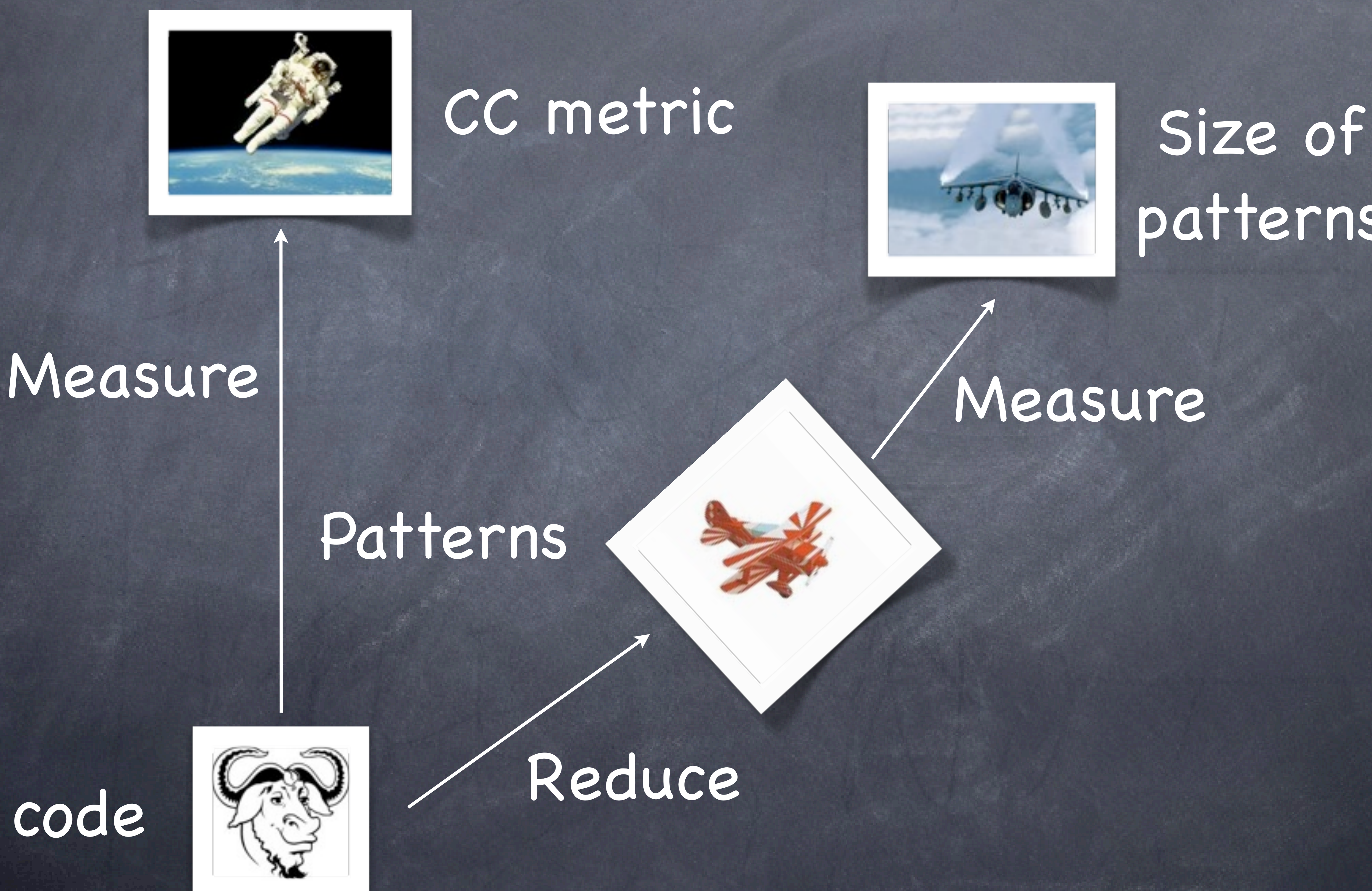
```
while (x >= 0) {
  if (x % 2 == 0)
    print("even");
  x--;
}
return 1;
```

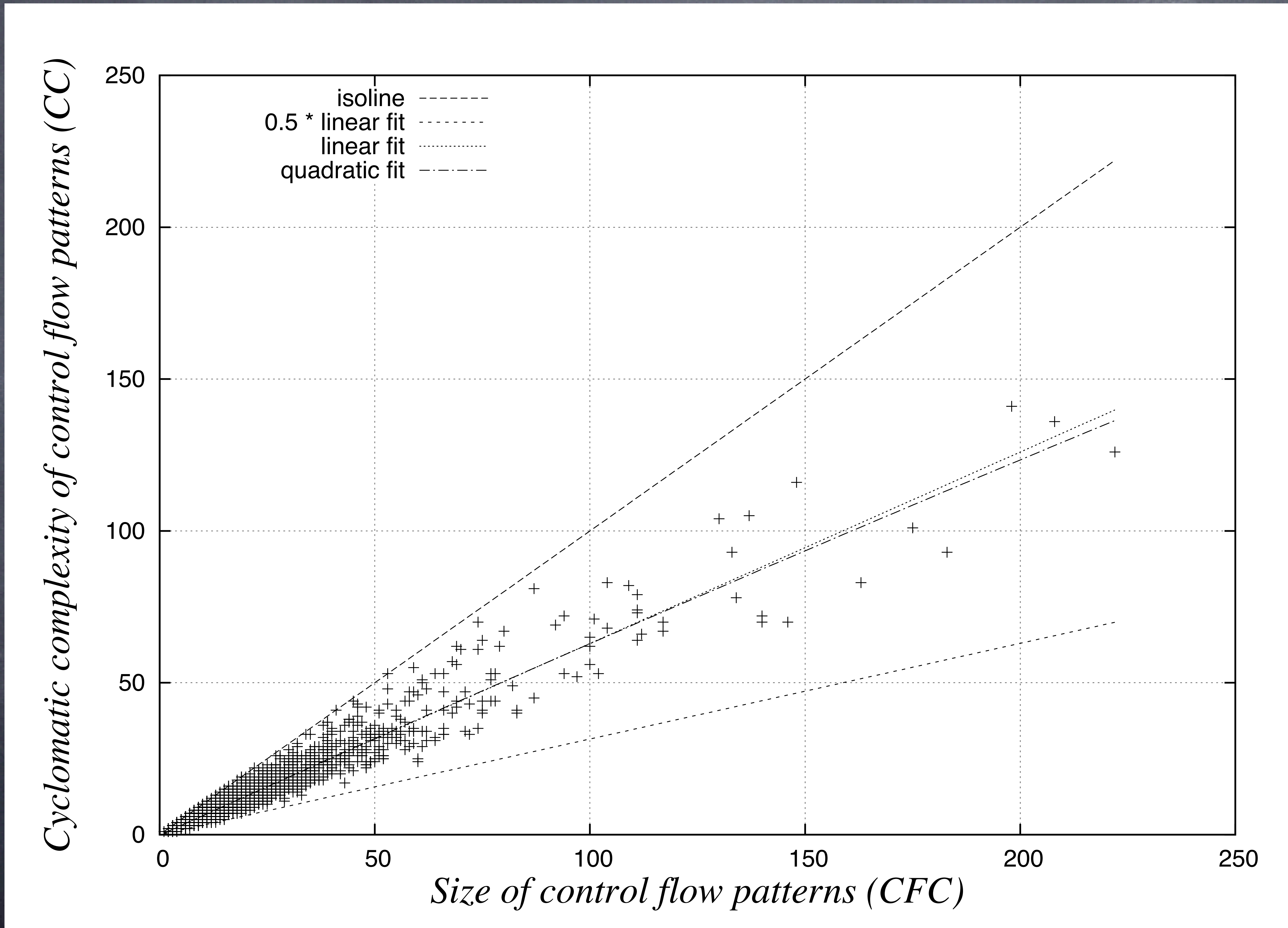
```
while (⊥) {
  if (⊥)
    ⊥

}
return ⊥;
```

```
public bool isFork(AstNode p) =
        p is doStatement
        || p is enhancedForStatement
        || p is forStatement
        || p is ifStatement
        || p is switchCase
        || p is catchClause
        || p is whileStatement
        ;

public AstNode reduce(AstNode body) = visit(body) {
    case methodDeclaration(_,_,_,_,_,_,bl) => methodBody(bl)
    case AstNode p => noop() when ! isControl(p)
    case str x => "_"
    case block([]) => noop()
    case block([AstNode n]) => n
    case some(noop()) => y when Option[AstNode] y := none()
    case x:[list[AstNode] a, noop(), list[AstNode] b] => y when list[AstNode] y := [e | AstN
};
```

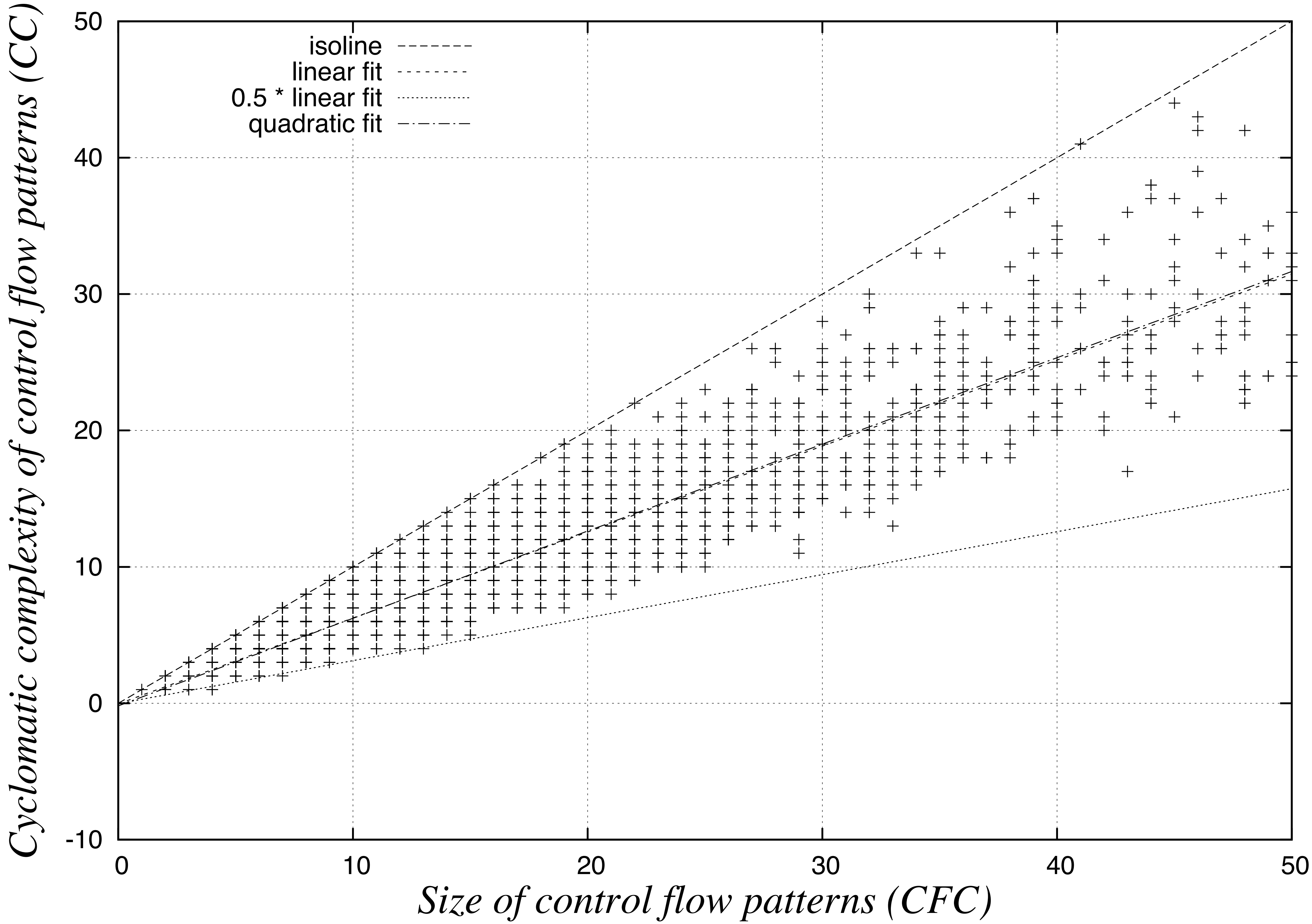
| Project | #Meth | #Pat |
|---|---|---|
| compendium | 7736 | 1271 (16%) |
| Tomcat70 | 16018 | 2211 (13%) |
| dsbudget | 306 | 64 (20%) |
| xml-commons-external | 3346 | 91 (2%) |
| apache-ant | 10278 | 1391 (13%) |
| bcel | 3076 | 286 (9%) |
| hsqldb | 5326 | 1013 (19%) |
| smallsql | 2556 | 353 (13%) |
| **Merged** | 48642 | 5633 (11%) |

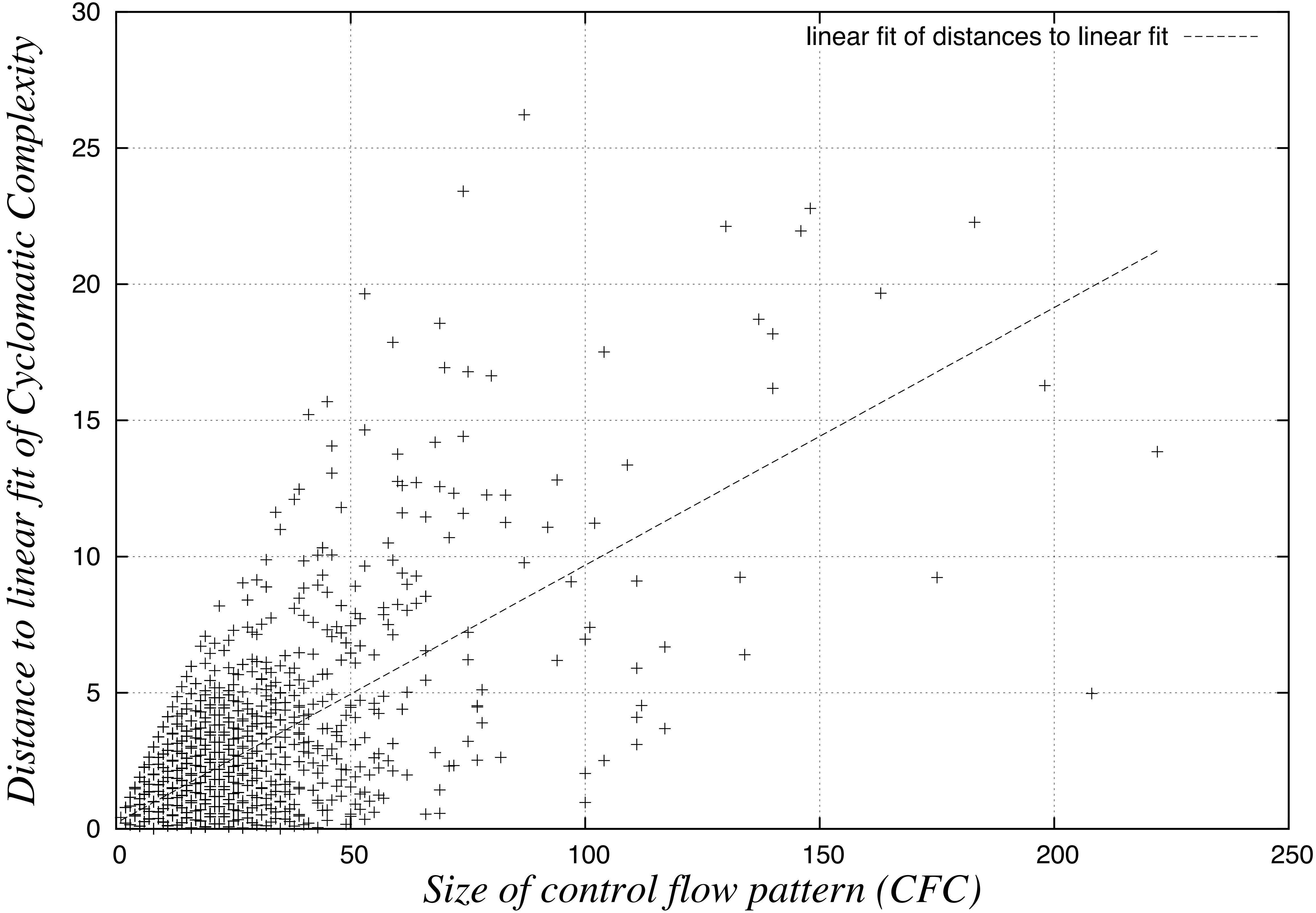
# What does CC miss?





isoline
0.5 * linear fit
linear fit
quadratic fit
Cyclomatic complexity of control flow patterns (CC)
Size of control flow patterns (CFC)
0
50
100
150
200
250
0
50
100
150
200
250


isoline
linear fit
0.5 * linear fit
quadratic fit
Cyclomatic complexity of control flow patterns (CC)
Size of control flow patterns (CFC)
50
40
30
20
10
0
-10
0
10
20
30
40
50



linear fit of distances to linear fit
Distance to linear fit of Cyclomatic Complexity
Size of control flow pattern (CFC)
0
5
10
15
20
25
30
0
50
100
150
200
250

# So

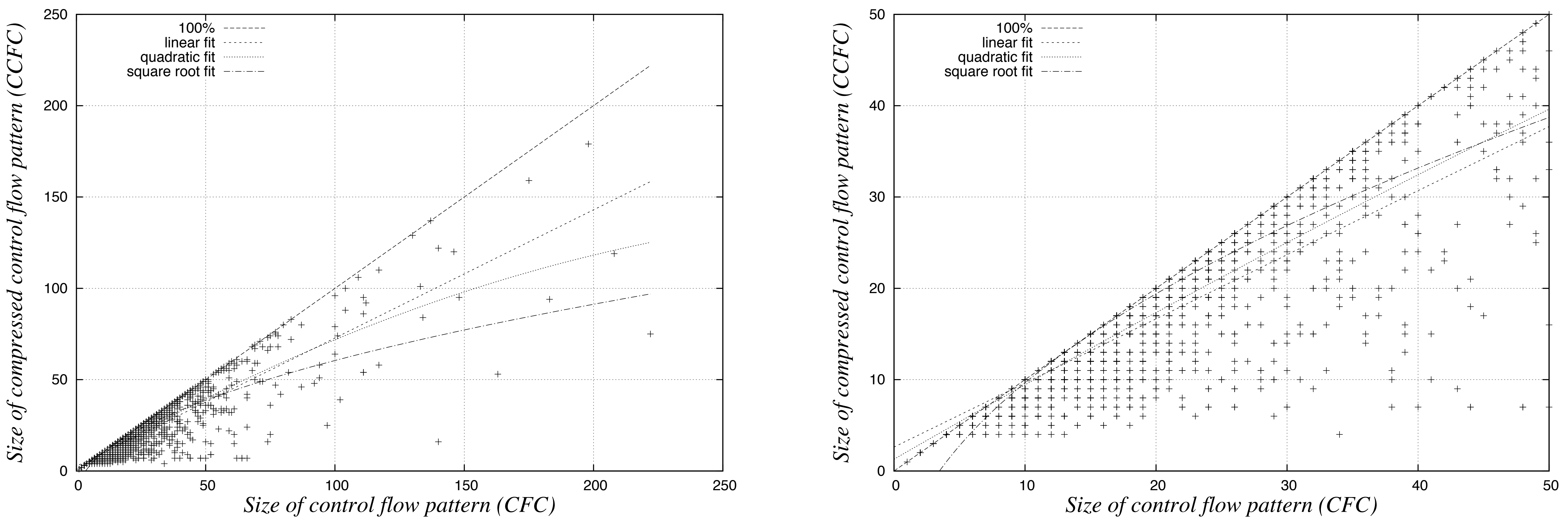
- If we assume that to understand a method you have to understand all of its control flow
- Then CC in theory does not measure accurately enough
- And in practise CC indeed does not predict the sizes of methods at all
- So we have to do more work

# Next idea: compress!

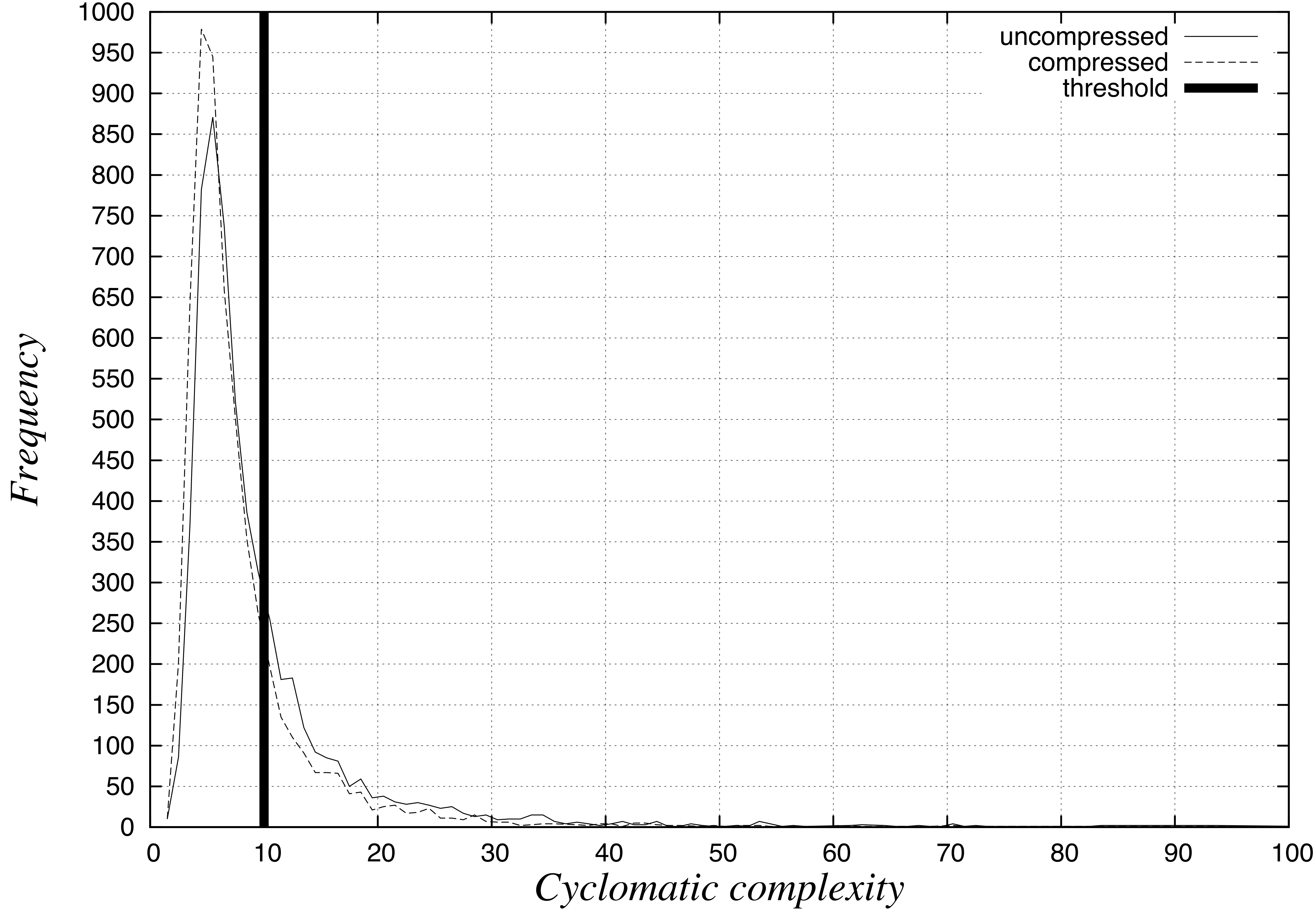
```
switch(⊥) {
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
  case ⊥ : return ⊥;
}
```

```
switch (⊥) {
  ℛ (case ⊥ : return ⊥;)
}
```

```
public AstNode compress(AstNode body) = innermost visit(body) {
  case [*a, repeated([*n]), n, *b] => [*a, repeated([*n]), *b]
  case [*a, x, *c, x, c, *d] => [*a, repeated([x,*c]), *d]
  case block(repeated(n)) => repeated(n)
};
```



100%
linear fit
quadratic fit
square root fit
Size of compressed control flow pattern (CCFC)
Size of control flow pattern (CFC)
0
50
100
150
200
250
0
10
20
30
40
50


uncompressed
compressed
threshold
Frequency
Cyclomatic complexity
1000
950
900
850
800
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0
0
10
20
30
40
50
60
70
80
90
100

# So

- Compression affects all method sizes
- Compression makes methods drop under 10
- Compression affects larger methods most
- Compression separates generated/simple code from the really hideous parts

- We have found indications that:
  - CC is not good because it misses complexity
  - CC is not good because it sees complexity where there is none
- Now what? We'll see...

Rascal is
a “ONE-STOP-SHOP”
for meta programming

http://www.rascal-mpl.org

Control flow patterns are a way
of studying control flow

http://homepages.cwi.nl/~jurgenv