Software Engineering: the war against complexity

Jurgen J. Vinju
Centrum Wiskunde & Informatica (CWI)

CHAQ Change-centric Quality Assurance
open tool demonstrations event
*at **Antwerp University***
on February 24th, 2015

CWI TU/e Inria INVENTORS FOR THE DIGITAL WORLD

CWI SWAT

Douglas DC-2 “KLM Uiver”

Great Design

We want great design for software too

- trustworthy
- cheap
- versatile
- simply beautiful

Great Design

We need *great design for software too*

- trustworthy
- cheap
- versatile
- simply beautiful

Great Design

- The DC-2 is obviously a high-quality design
  - it does not crash and handles very well
  - it does not wear quickly
  - yet, it is easy to maintain
  - it's a small investment compared to what you can earn with it
  - it can take on any cargo, including passengers, comfortably
  - it's both good in general and good in detail; every detail matters
  - it's very, very shiny
- We know pretty well how to describe, judge and improve airplan quality

Software Design

Most software does not have to actually *fly*, so it's not as hard to design as the DC-2…

Common belief that "software" is indeed "soft"

- Ugly software also works…
- If software breaks, we just fix it…

We know this is not true

Software Design

So, what exactly is
good software design ?
and
why does it matter ?

Software quality is hard to observe

Software quality is hard to observe

- if you can't see it, it does not mean it does not exist

Software quality is hard to observe

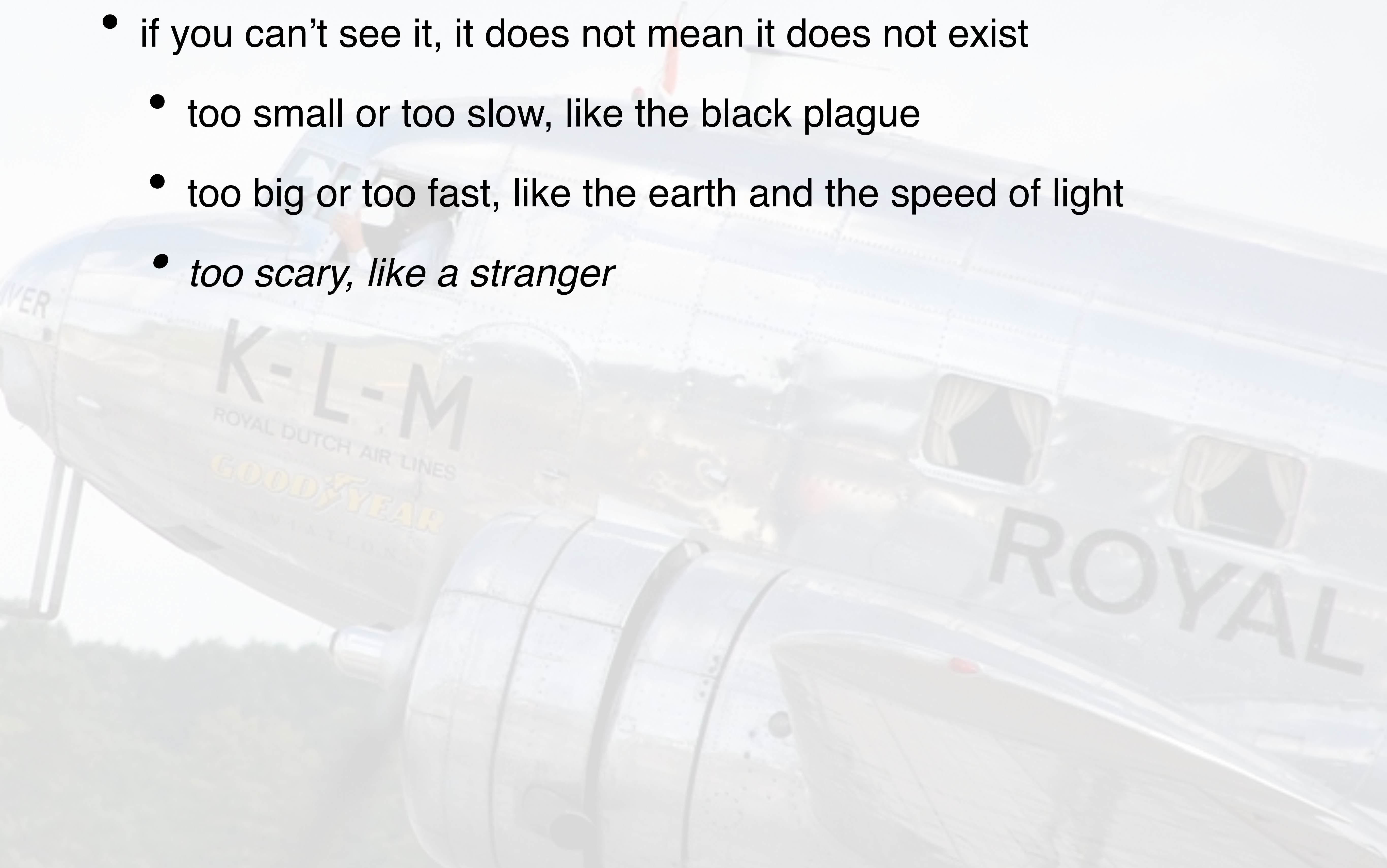
- if you can’t see it, it does not mean it does not exist
  - too small or too slow, like the black plague

Software quality is hard to observe

- if you can't see it, it does not mean it does not exist
  - too small or too slow, like the black plague
  - too big or too fast, like the earth and the speed of light

Software quality is hard to observe

- if you can’t see it, it does not mean it does not exist
  - too small or too slow, like the black plague
  - too big or too fast, like the earth and the speed of light
  - *too scary, like a stranger*



Software quality is hard to observe

- if you can’t see it, it does not mean it does not exist
  - too small or too slow, like the black plague
  - too big or too fast, like the earth and the speed of light
  - *too scary, like a stranger*
  - too subjective, like art

Software quality is hard to observe

- if you can't see it, it does not mean it does not exist
  - too small or too slow, like the black plague
  - too big or too fast, like the earth and the speed of light
  - *too scary, like a stranger*
  - too subjective, like art
- Software

Software quality is hard to observe

- if you can’t see it, it does not mean it does not exist
  - too small or too slow, like the black plague
  - too big or too fast, like the earth and the speed of light
  - *too scary, like a stranger*
  - too subjective, like art
- Software
  - *consists of many small details*

Software quality is hard to observe

- if you can't see it, it does not mean it does not exist
  - too small or too slow, like the black plague
  - too big or too fast, like the earth and the speed of light
  - *too scary, like a stranger*
  - too subjective, like art
- Software
  - *consists of many small details*
  - *evolves slowly but surely*

Software quality is hard to observe

- if you can’t see it, it does not mean it does not exist
  - too small or too slow, like the black plague
  - too big or too fast, like the earth and the speed of light
  - *too scary, like a stranger*
  - too subjective, like art
- Software
  - *consists of many small details*
  - *evolves slowly but surely*
  - *too big to read and too fast to trace*

Software quality is hard to observe

- if you can’t see it, it does not mean it does not exist
  - too small or too slow, like the black plague
  - too big or too fast, like the earth and the speed of light
  - *too scary, like a stranger*
  - too subjective, like art
- Software
  - *consists of many small details*
  - *evolves slowly but surely*
  - *too big to read and too fast to trace*
  - *new concept to most people*

Software quality is hard to observe

- if you can't see it, it does not mean it does not exist
  - too small or too slow, like the black plague
  - too big or too fast, like the earth and the speed of light
  - *too scary, like a stranger*
  - too subjective, like art
- Software
  - *consists of many small details*
  - *evolves slowly but surely*
  - *too big to read and too fast to trace*
  - *new concept to most people*
  - *quality is contextual*

Software quality is hard to observe

- if you can't see it, it does not mean it does not exist
  - too small or too slow, like the black plague
  - too big or too fast, like the earth and the speed of light

Agenda make software quality known to and observable by non-software-specialists, creating more traction for investing in software quality

  - *evolves slowly but surely*
  - *too big to read and too fast to trace*
  - *new concept to most people*
  - *quality is contextual*

Complexity Dominates Software Quality

Software quality is about subjective requirements

correct, testable, efficient, secure, flexible,

but all of these depend on

COMPLEXITY

(¬simplicity)

Wikipedia

Complexity

Complexity is generally used to characterize something with many parts where those parts interact with each other in multiple ways. The study of these complex linkages is the main goal of complex systems theory.

Thesaurus

complicated adjective

the complicated election rules: COMPLEX, intricate, involved, convoluted, tangled, elaborate, impenetrable, knotty, tricky, ...

Wikipedia

Simplicity

Simplicity is the state or quality of being simple. Something which is easy to understand or explain is simple, in contrast to something complicated. Alternatively, as Herbert A. Simon suggested, something is simple or complex depending on the way we choose to describe it. In some uses, simplicity can be used to imply beauty, purity, or clarity. Simplicity may also be used in a negative connotation to denote a deficit or insufficiency of nuance or complexity of a thing, relative to what is supposed to be required.

Complexity Trumps

- Correctness & security:
  - can’t verify what you can’t define
  - debilitating high cost
- Testable:
  - can’t test what’s not independend
- Efficiency:
  - can’t pin-point causes of bottlenecks
- Flexible:
  - can’t predict impact of change

Software Complexity Agenda

Software Complexity Agenda

- *Philosophy (what is software complexity?)*

Software Complexity Agenda

- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*

Software Complexity Agenda

- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*
- *Engineering*
  - *Maintenance (what can we do about it?)*
  - *Construction (how can we prevent it?)*:

Software Complexity Agenda

- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*
- *Engineering*
    - *Maintenance (what can we do about it?)*
    - *Construction (how can we prevent it?)*:
- Conclusion (holistic perspective)
    - Meta-tools
    - Public/private collaboration

Software Complexity Agenda

- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*
- *Engineering*
  - *Maintenance (what can we do about it?)*
  - *Construction (how can we prevent it?)*:
- Conclusion (holistic perspective)
  - Meta-tools
  - Public/private collaboration

3 examples

Software Complexity Agenda

- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*
- *Engineering*
  - *Maintenance (what can we do about it?)*
  - *Construction (how can we prevent it?)*:
- Conclusion (holistic perspective)
  - Meta-tools
  - Public/private collaboration

3 examples

one-stop-shop advertisement

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

The source code of “ls”

3894 lines

367 ifs

174 cases

The source code of "ls"

3894 lines

367 ifs

174 cases

If Kafka would write a book today…

This kind of software exists everywhere:

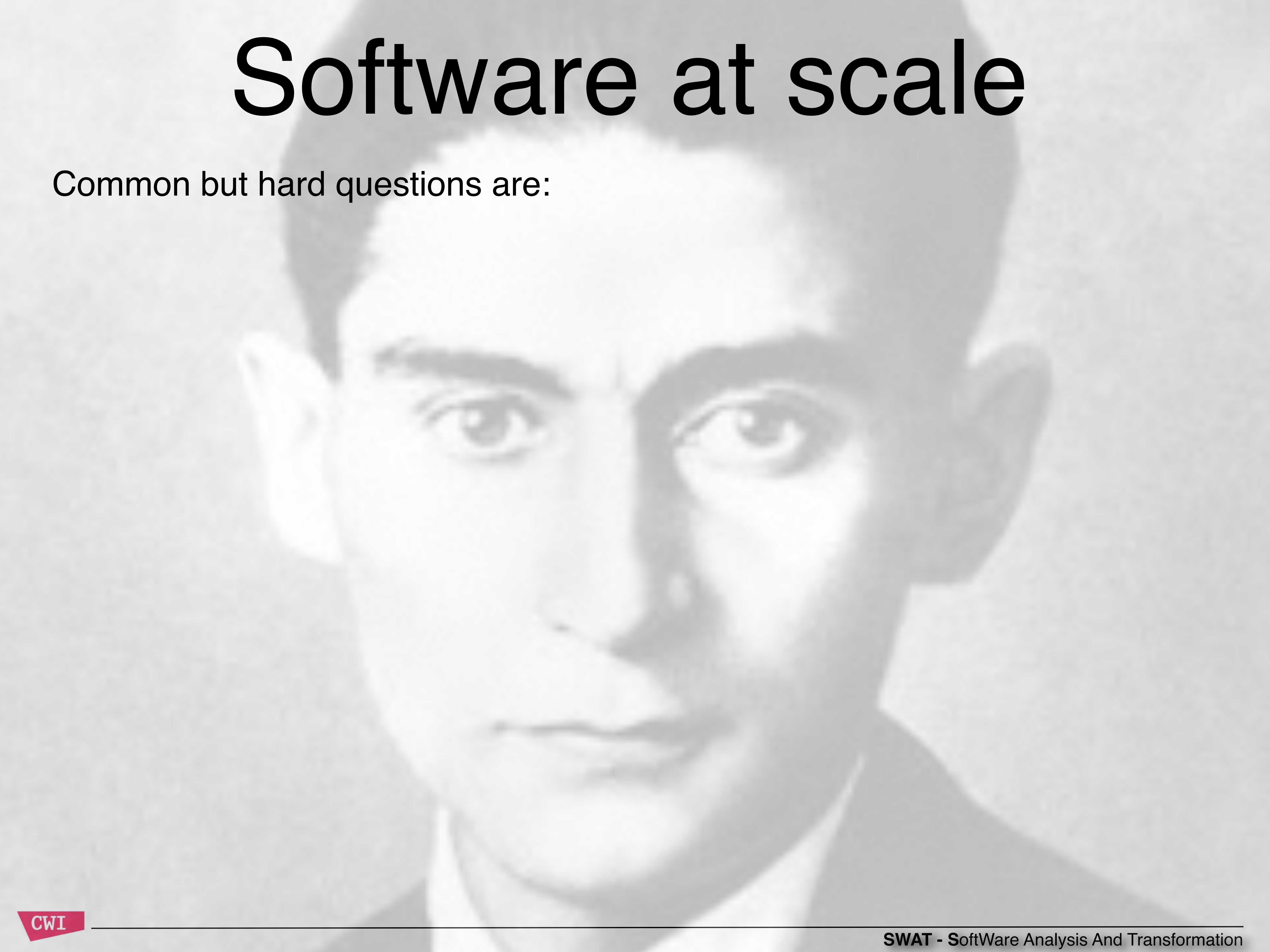
- 10K to 25M lines of code
- 2 to 10 programming languages and dialects
- 20 to 200 dependencies on library components and frameworks
- 10 to 1000 programmers
- 1 to 1M users
- 10 to 40 years lifetime
- “IT happens”

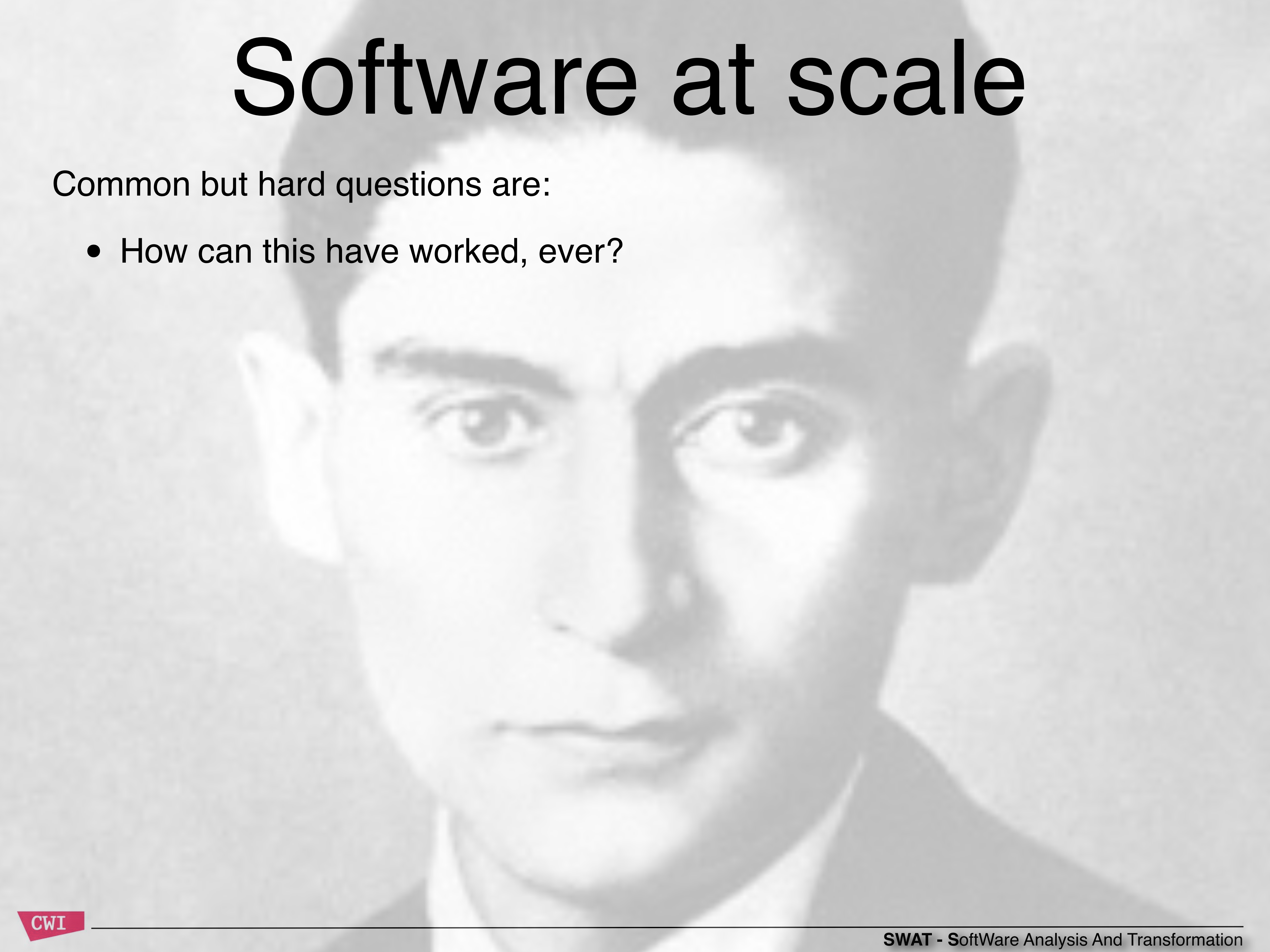
having a nightmarishly complex, bizarre, or illogical quality

Software at scale

Software at scale

Common but hard questions are:





Software at scale

Common but hard questions are:

- How can this have worked, ever?

Software at scale

Common but hard questions are:

- How can this have worked, ever?
- What is it? What does it do? When? How? Why?

Software at scale

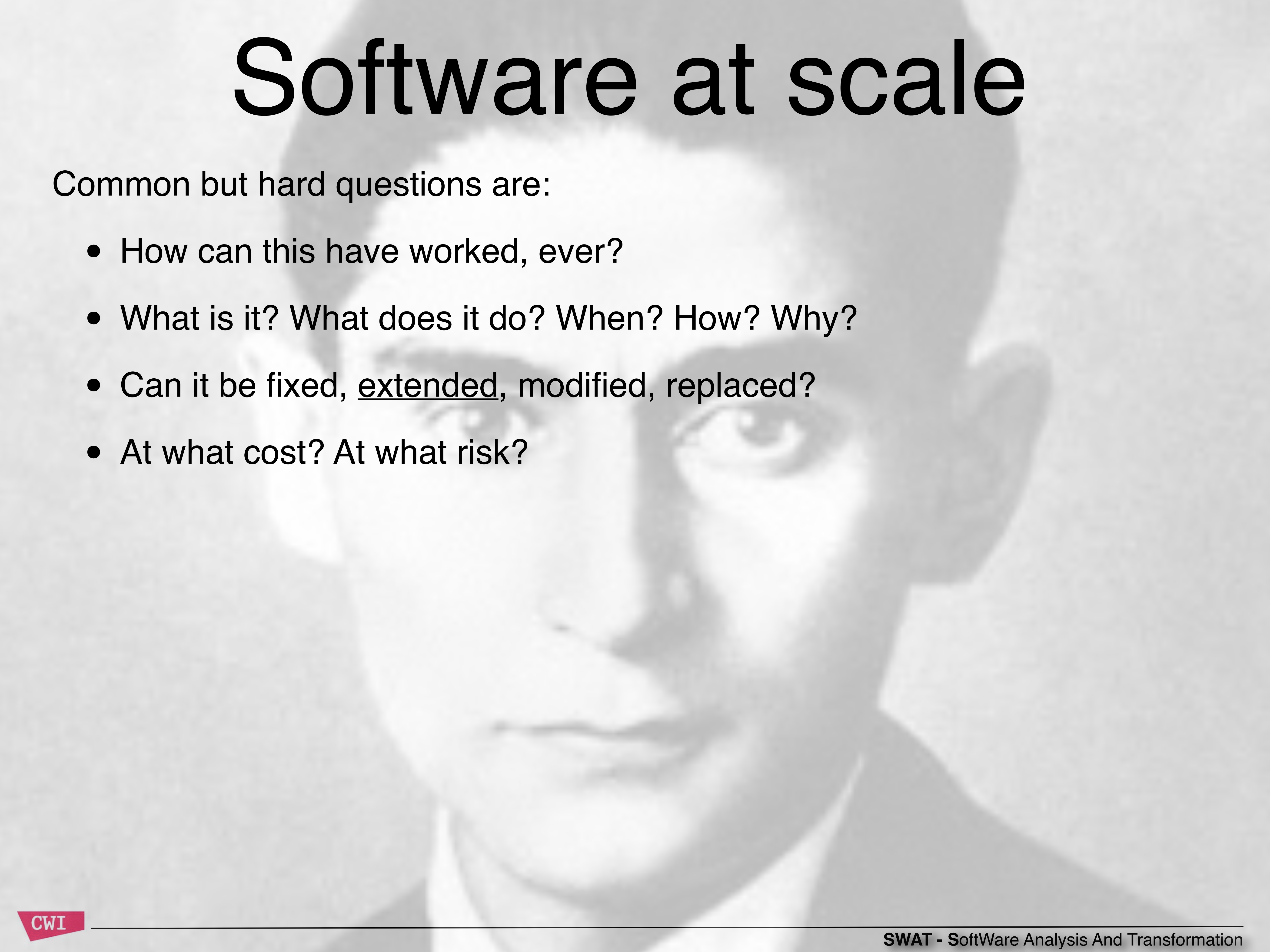
Common but hard questions are:

- How can this have worked, ever?
- What is it? What does it do? When? How? Why?
- Can it be fixed, extended, modified, replaced?

Software at scale

Common but hard questions are:

- How can this have worked, ever?
- What is it? What does it do? When? How? Why?
- Can it be fixed, extended, modified, replaced?
- At what cost? At what risk?



Software at scale

Common but hard questions are:

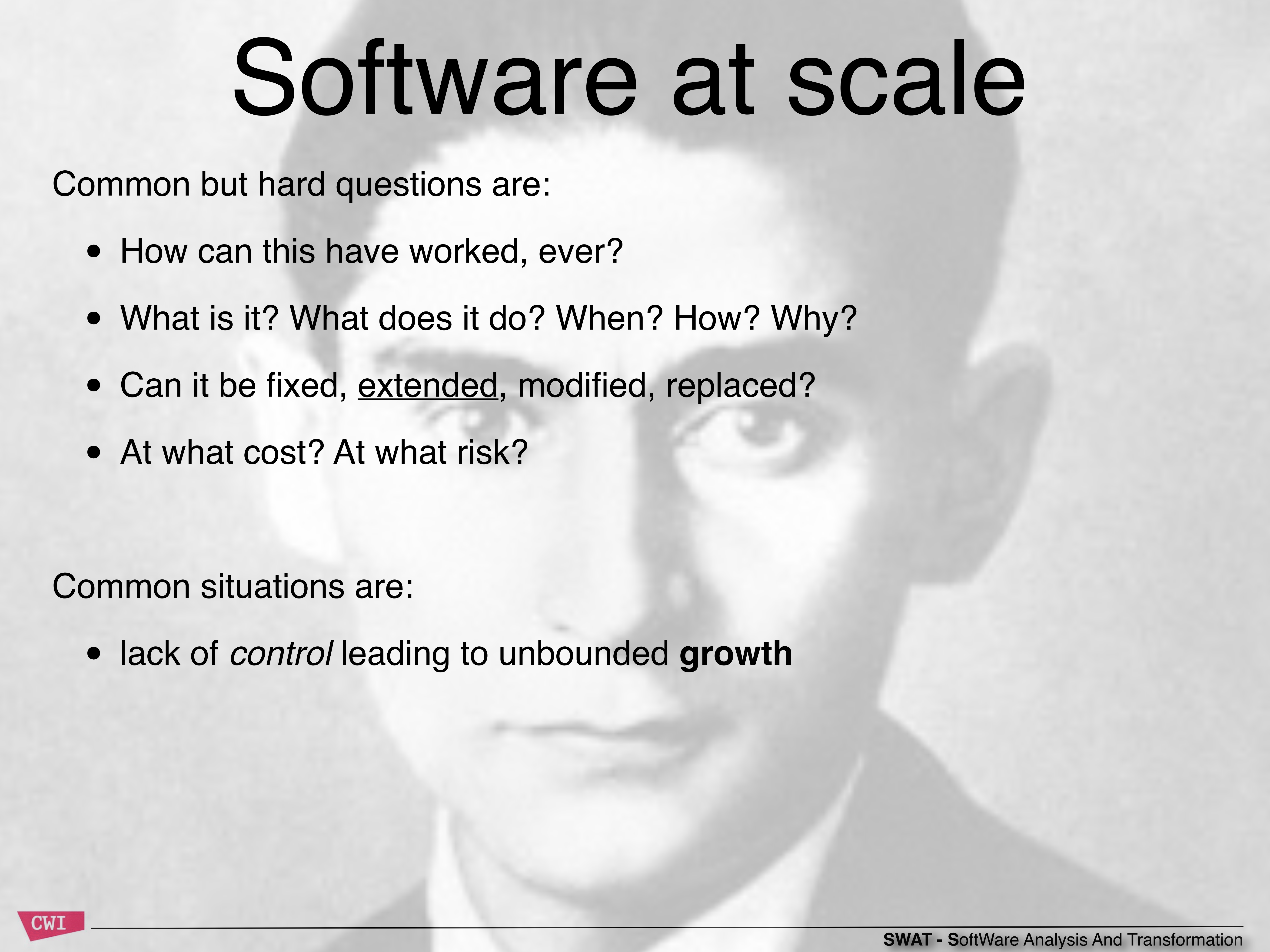
- How can this have worked, ever?
- What is it? What does it do? When? How? Why?
- Can it be fixed, <u>extended</u>, modified, replaced?
- At what cost? At what risk?

Software at scale

Common but hard questions are:

- How can this have worked, ever?
- What is it? What does it do? When? How? Why?
- Can it be fixed, extended, modified, replaced?
- At what cost? At what risk?

Common situations are:



Software at scale

Common but hard questions are:

- How can this have worked, ever?
- What is it? What does it do? When? How? Why?
- Can it be fixed, extended, modified, replaced?
- At what cost? At what risk?

Common situations are:

- lack of *control* leading to unbounded **growth**

Software at scale

Common but hard questions are:

- How can this have worked, ever?
- What is it? What does it do? When? How? Why?
- Can it be fixed, extended, modified, replaced?
- At what cost? At what risk?

Common situations are:

- lack of *control* leading to unbounded **growth**
- lack of *predictability*, leading to unbounded **cost**

Software at scale

Common but hard questions are:

- How can this have worked, ever?
- What is it? What does it do? When? How? Why?
- Can it be fixed, extended, modified, replaced?
- At what cost? At what risk?

Common situations are:

- lack of *control* leading to unbounded **growth**
- lack of *predictability*, leading to unbounded **cost**
- lack of long term *perspective*, leading to **ill-informed decisions**

Software at scale

Common but hard questions are:

- How can this have worked, ever?
- What is it? What does it do? When? How? Why?
- Can it be fixed, extended, modified, replaced?
- At what cost? At what risk?

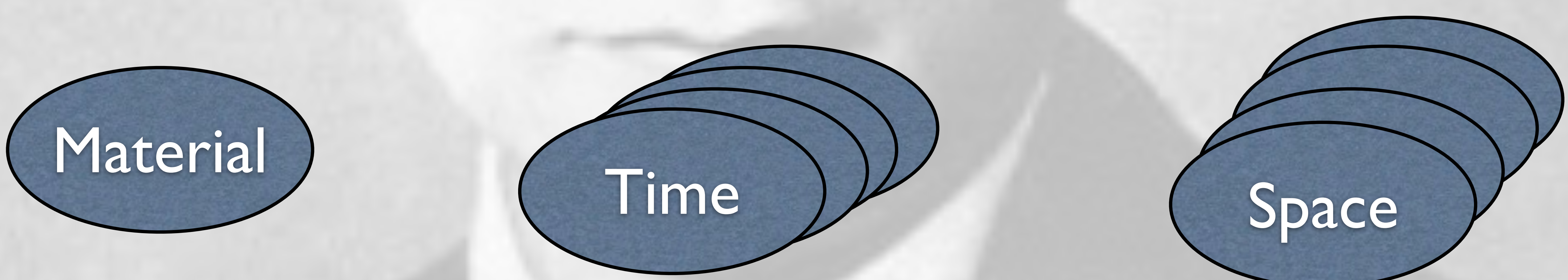
Common situations are:

- lack of *control* leading to unbounded **growth**
- lack of *predictability*, leading to unbounded **cost**
- lack of long term *perspective*, leading to **ill-informed decisions**
- complex software is the enemy of quality

Software at scale

Software Complexity is exhibited by:

- heterogeneity (different kinds of parts)
- code volume (textually)
- dependence (semantics)
- encapsulation (nesting)
- distribution (deployment)
- evolution (versions)



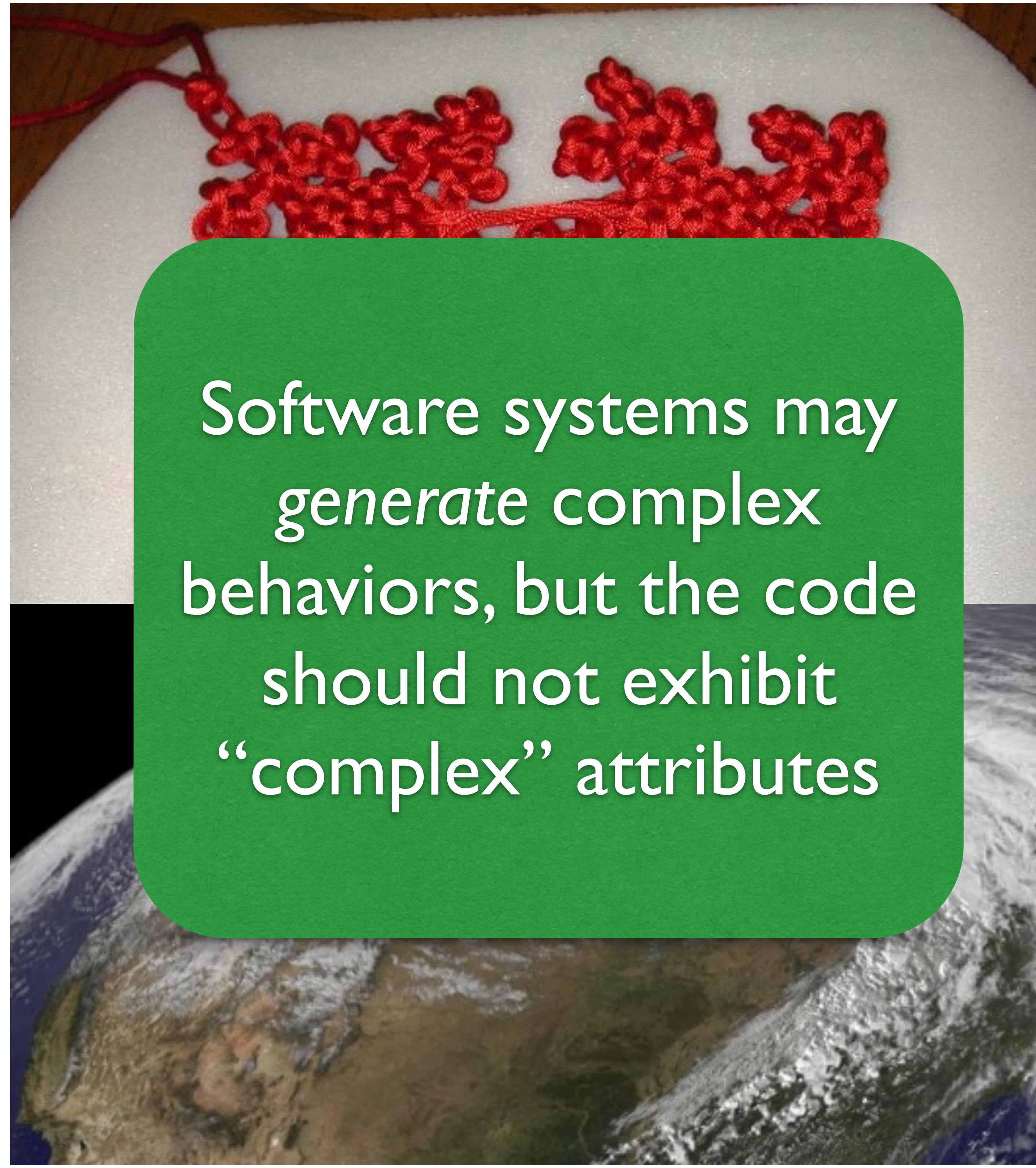
Complex *or* Complicated?

- *Complicated =* many interrelated parts
  - linear: small change = small impact
  - predictable: straight flow, local failure
  - decomposable: manageable
- *Complex* = unpredictable & hard to manage
  - emergent: whole is more than sum
  - non-linear: small change = big impact?
  - cascading failure
  - hysteresis: you must understand its history
  - indivisible

[CSIS paper: "Organizing for a Complex World: The Way Ahead]

Complex *or* Complicated?

- *Complicated* = many interrelated parts
  - linear: small change = small impact
  - predictable: straight flow, local failure
  - decomposable: manageable
- *Complex* = unpredictable & hard to manage
  - emergent: whole is more than sum
  - non-linear: small change = big impact?
  - cascading failure
  - hysteresis: you must understand its history
  - indivisible



[CSIS paper: "Organizing for a Complex World: The Way Ahead]

Software Complexity Agenda

Software Complexity Agenda

- *Philosophy (what is software complexity?)*

Software Complexity Agenda

- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*

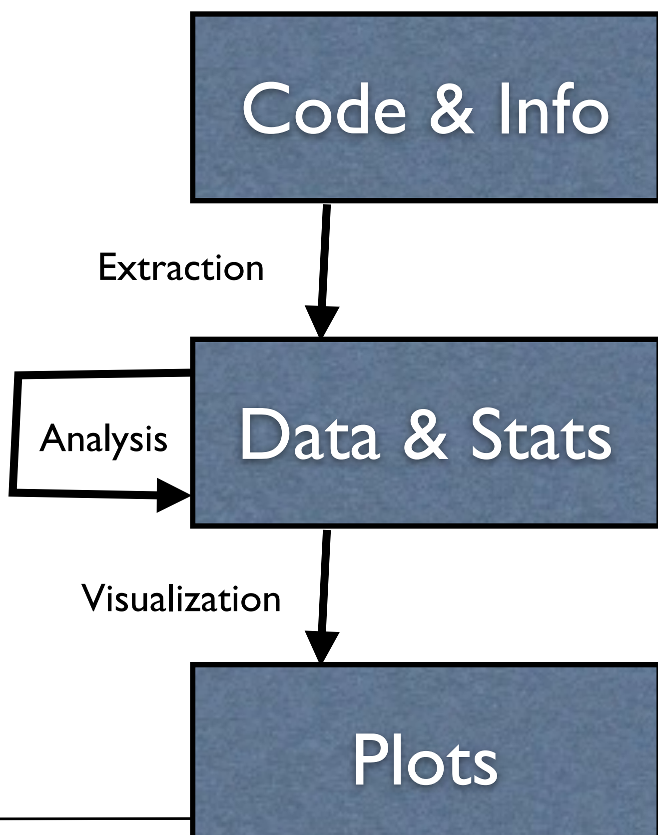
Software Complexity Agenda

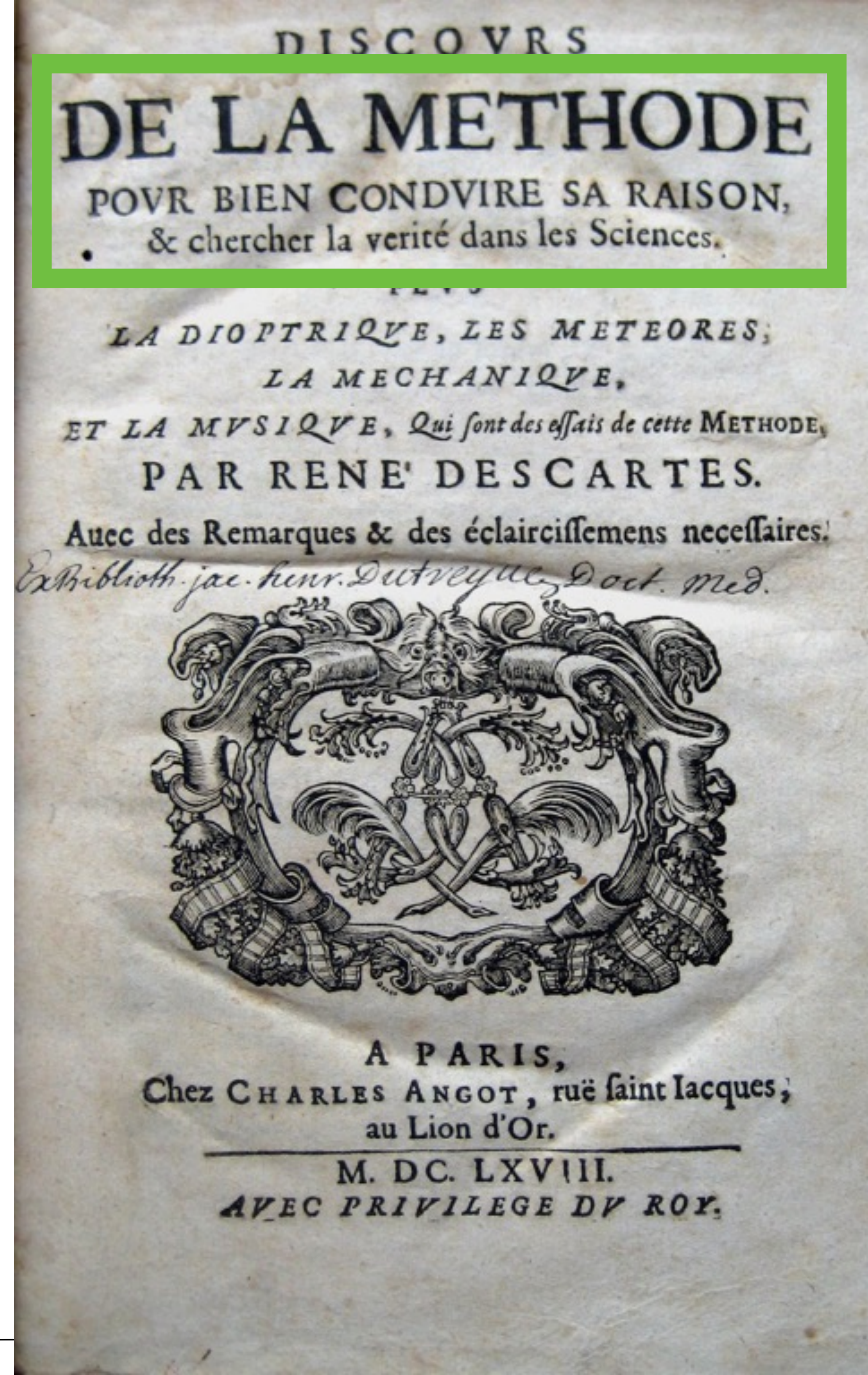
- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*
- *Engineering*
  - *Maintenance (what can we do about it?)*
  - *Construction (how can we prevent it?)*:

Software Complexity Agenda

- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*
- *Engineering*
  - *Maintenance (what can we do about it?)*
  - *Construction (how can we prevent it?)*:
- Conclusion (holistic perspective)
  - Meta-tools
  - Public/private collaboration

Science





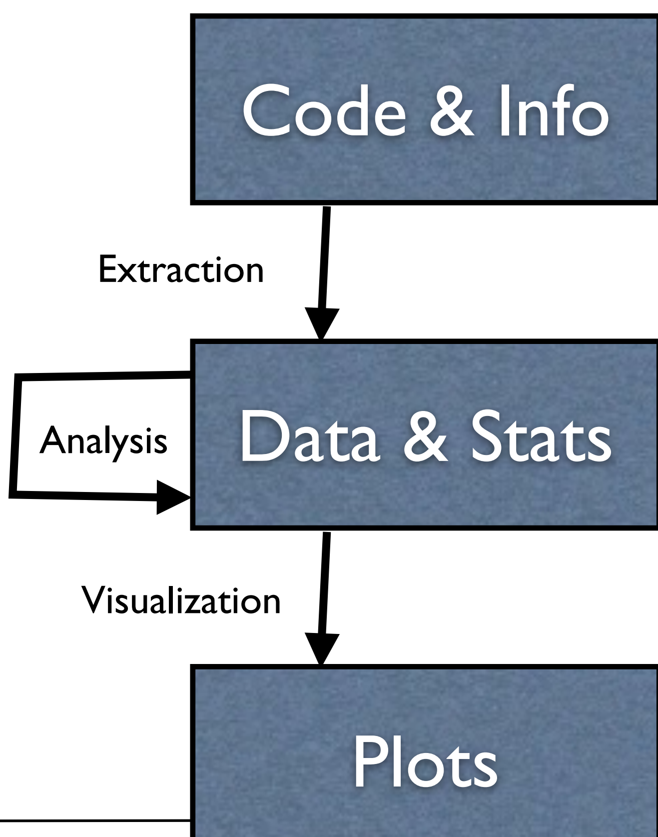
DISCOVRS
DE LA METHODE
POVR BIEN CONDVIRE SA RAISON,
& chercher la verité dans les Sciences.

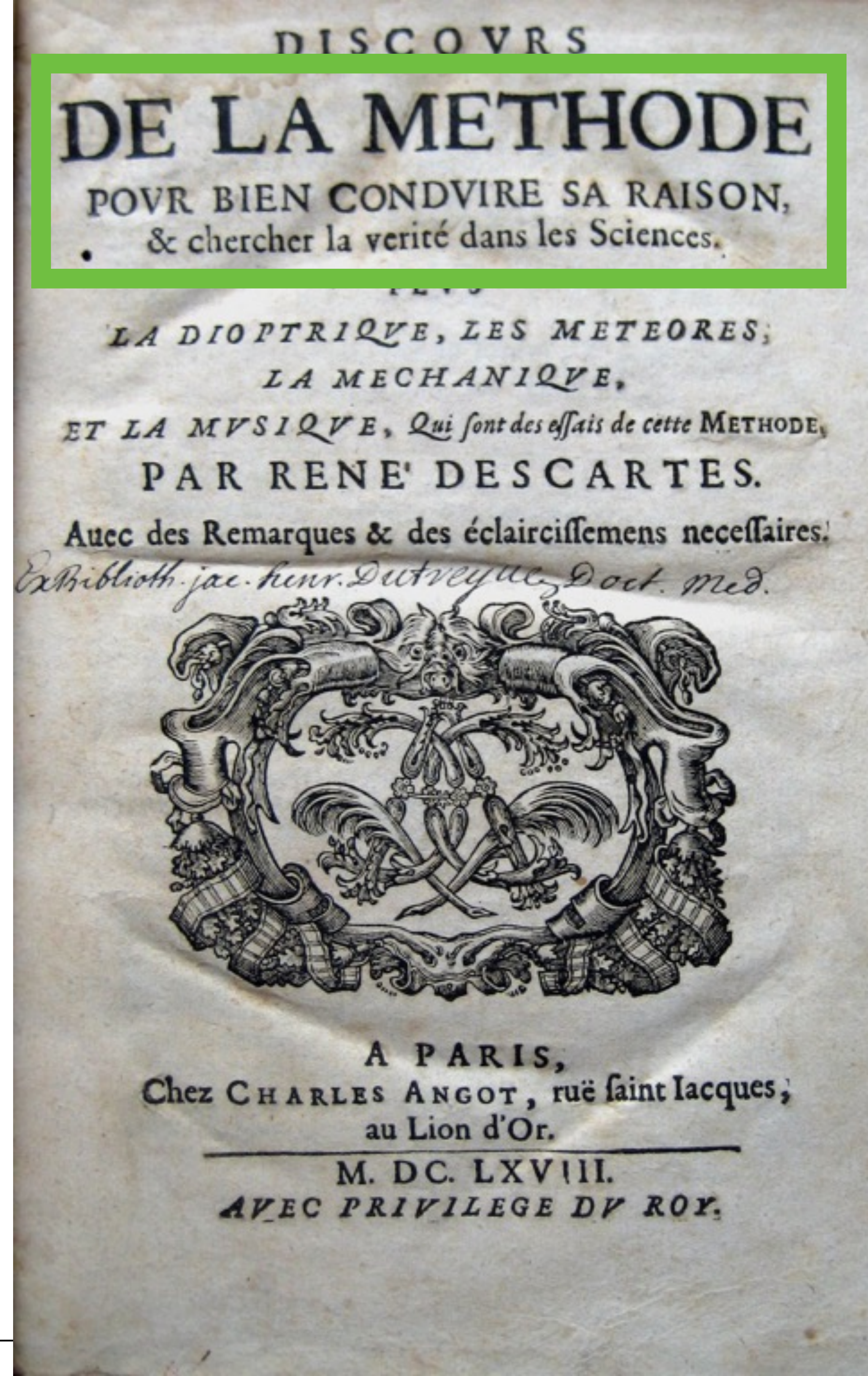
LA DIOPTRIQVE, LES METEORES,
LA MECHANIQVE,
ET LA MVSIQVE, Qui ſont des eſſais de cette METHODE,
PAR RENE' DESCARTES.
Auec des Remarques & des éclairciſſemens neceſſaires.

A PARIS,
Chez CHARLES ANGOT, ruë ſaint Iacques,
au Lion d'Or.
M. DC. LXVIII.
AVEC PRIVILEGE DV ROY.

Science

- Software Analytics





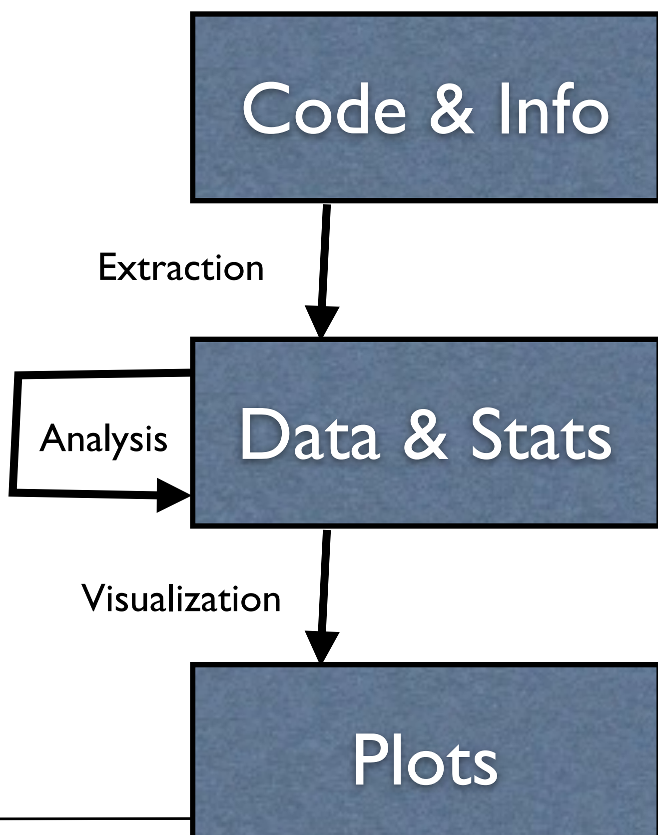
DISCOVRS
DE LA METHODE
POVR BIEN CONDVIRE SA RAISON,
& chercher la verité dans les Sciences.

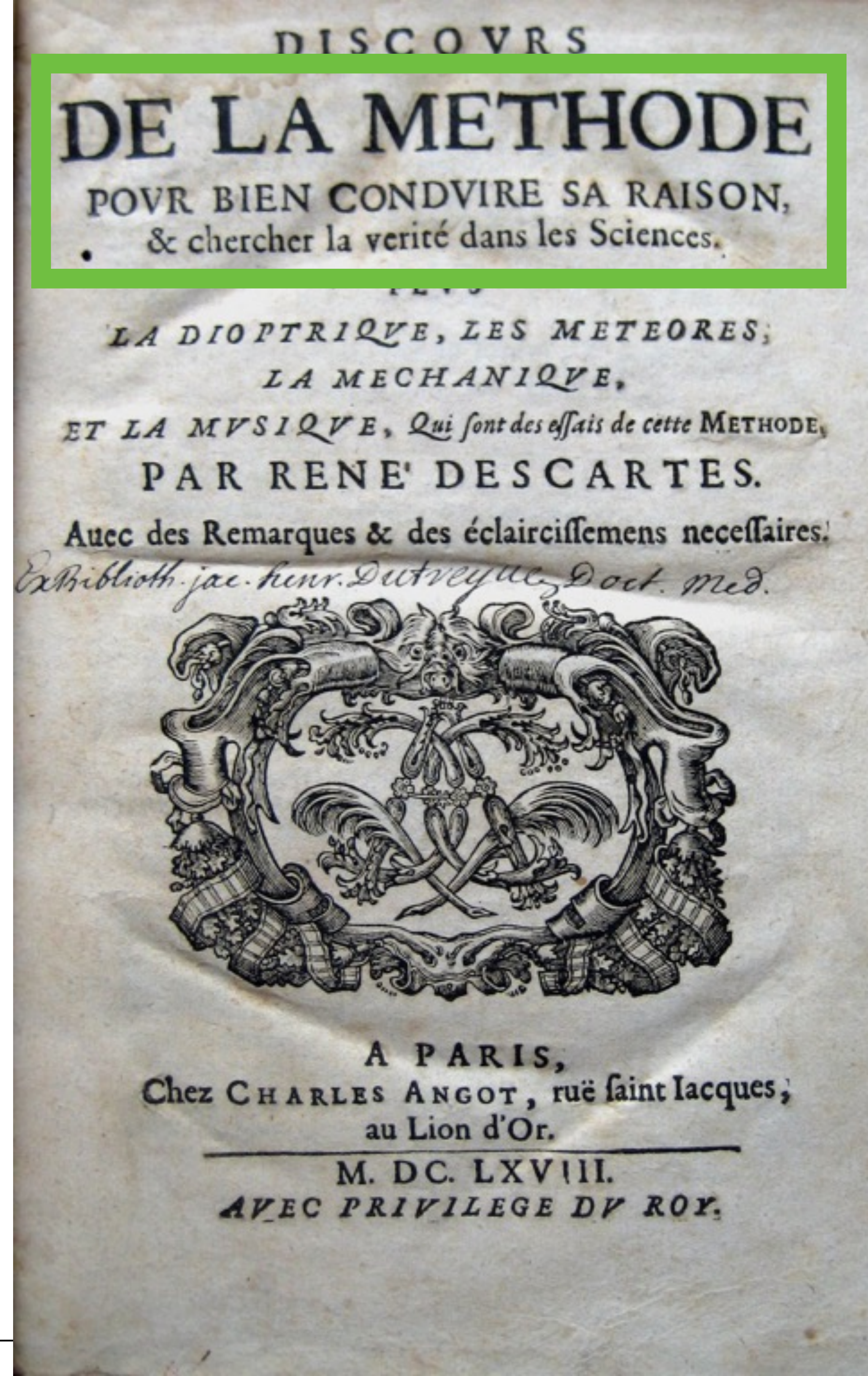
LA DIOPTRIQVE, LES METEORES,
LA MECHANIQVE,
ET LA MVSIQVE, Qui ſont des eſſais de cette METHODE,
PAR RENE' DESCARTES.
Auec des Remarques & des éclairciſſemens neceſſaires.

A PARIS,
Chez CHARLES ANGOT, ruë ſaint Iacques,
au Lion d'Or.
M. DC. LXVIII.
AVEC PRIVILEGE DV ROY.

Science

- Software Analytics
- “debunking” common beliefs





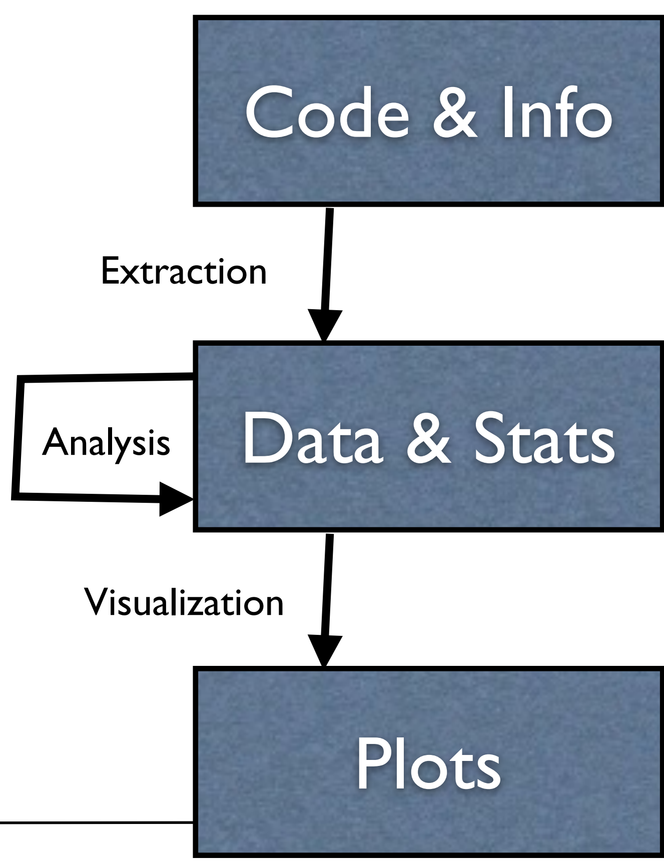
DISCOVRS
DE LA METHODE
POVR BIEN CONDVIRE SA RAISON,
& chercher la verité dans les Sciences.

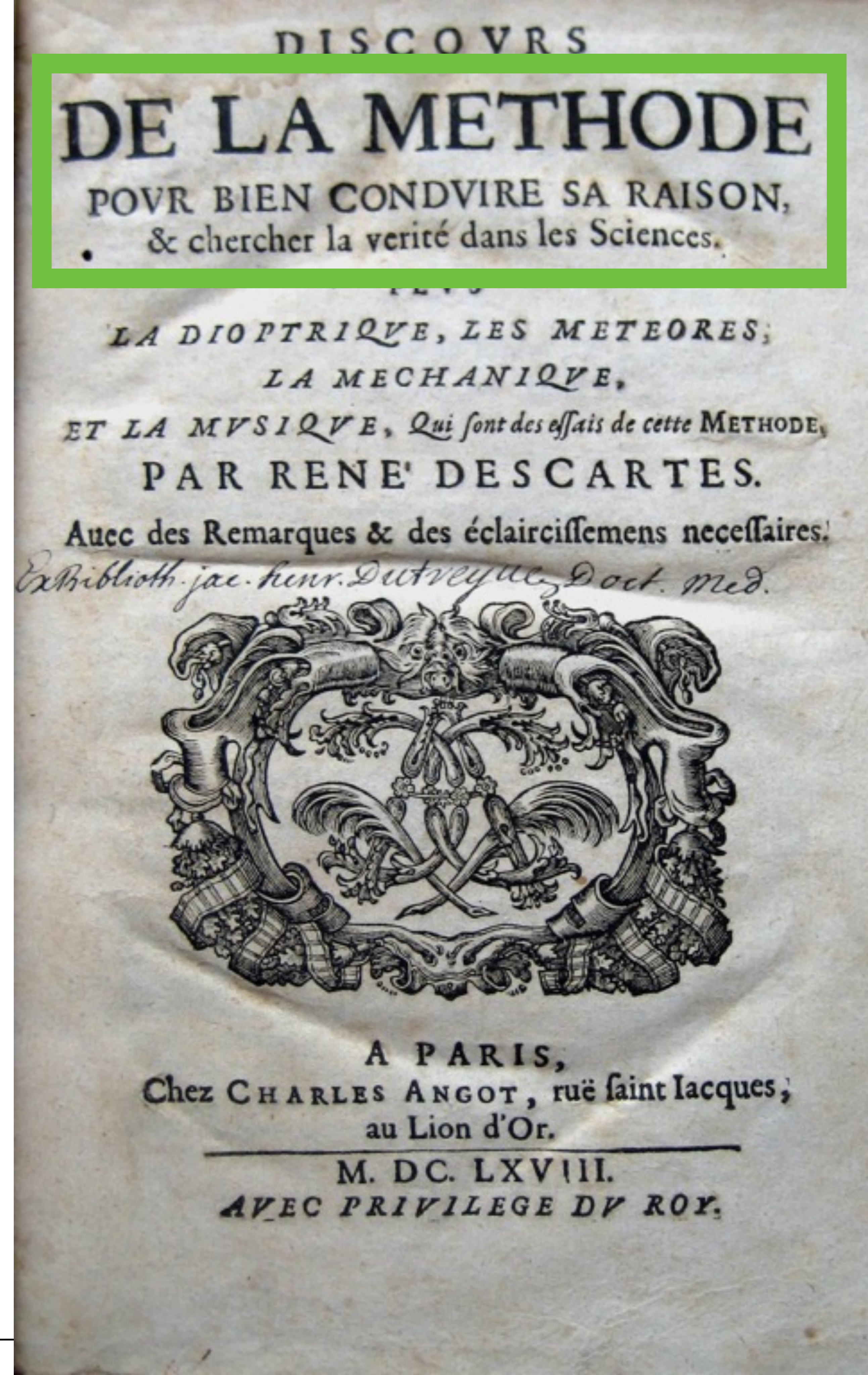
LA DIOPTRIQVE, LES METEORES,
LA MECHANIQVE,
ET LA MVSIQVE, Qui ſont des eſſais de cette METHODE,
PAR RENE' DESCARTES.
Auec des Remarques & des éclairciſſemens neceſſaires.
Ex Biblioth. jac. henr. Dutveyue Doct. med.

A PARIS,
Chez CHARLES ANGOT, ruë ſaint Iacques,
au Lion d'Or.
M. DC. LXVIII.
AVEC PRIVILEGE DV ROY.

Science

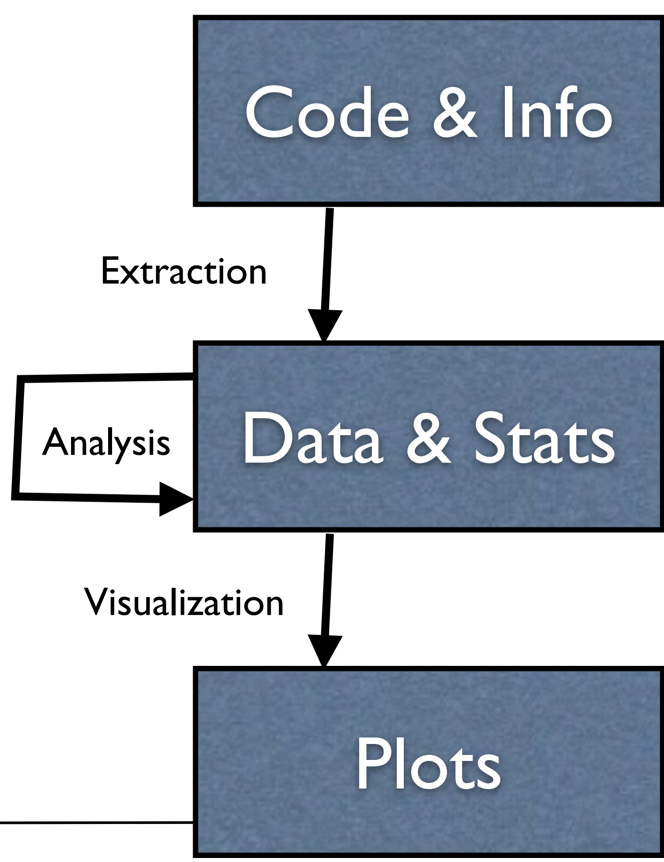
- Software Analytics
- “debunking” common beliefs
- “discovering” new truths by observation/experimentation

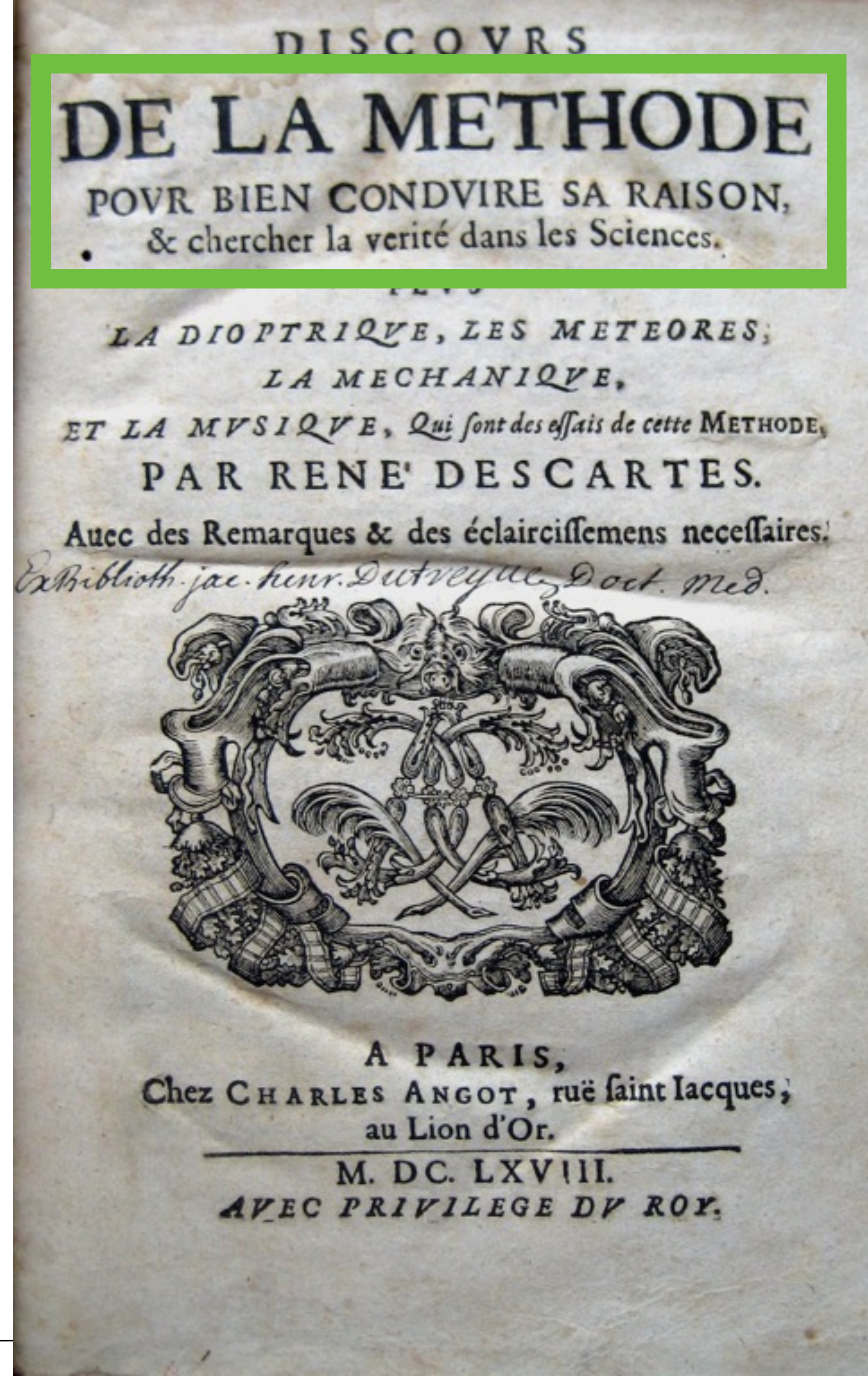




Science

- Software Analytics
- “debunking” common beliefs
- “discovering” new truths by observation/experimentation
- mining software repositories!





Science of SLOC & CC

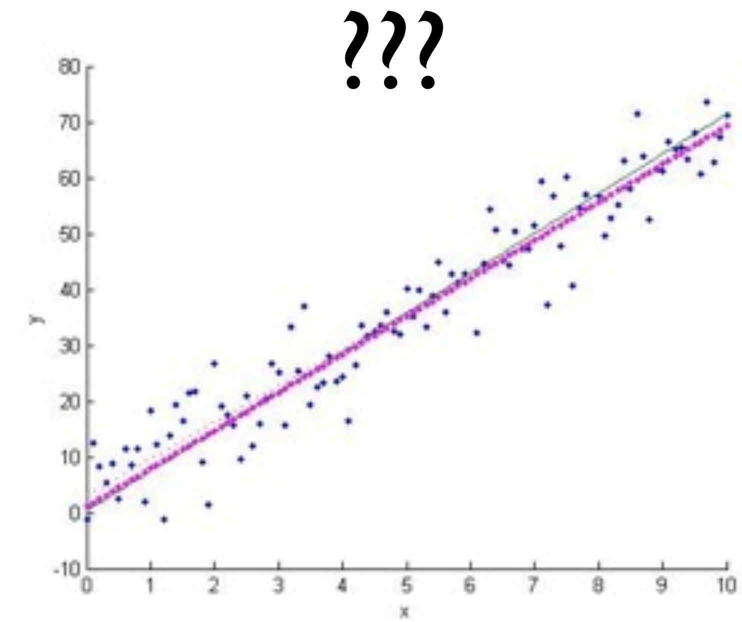
Davy Landman, ICSM2014

Submitted to JSEP

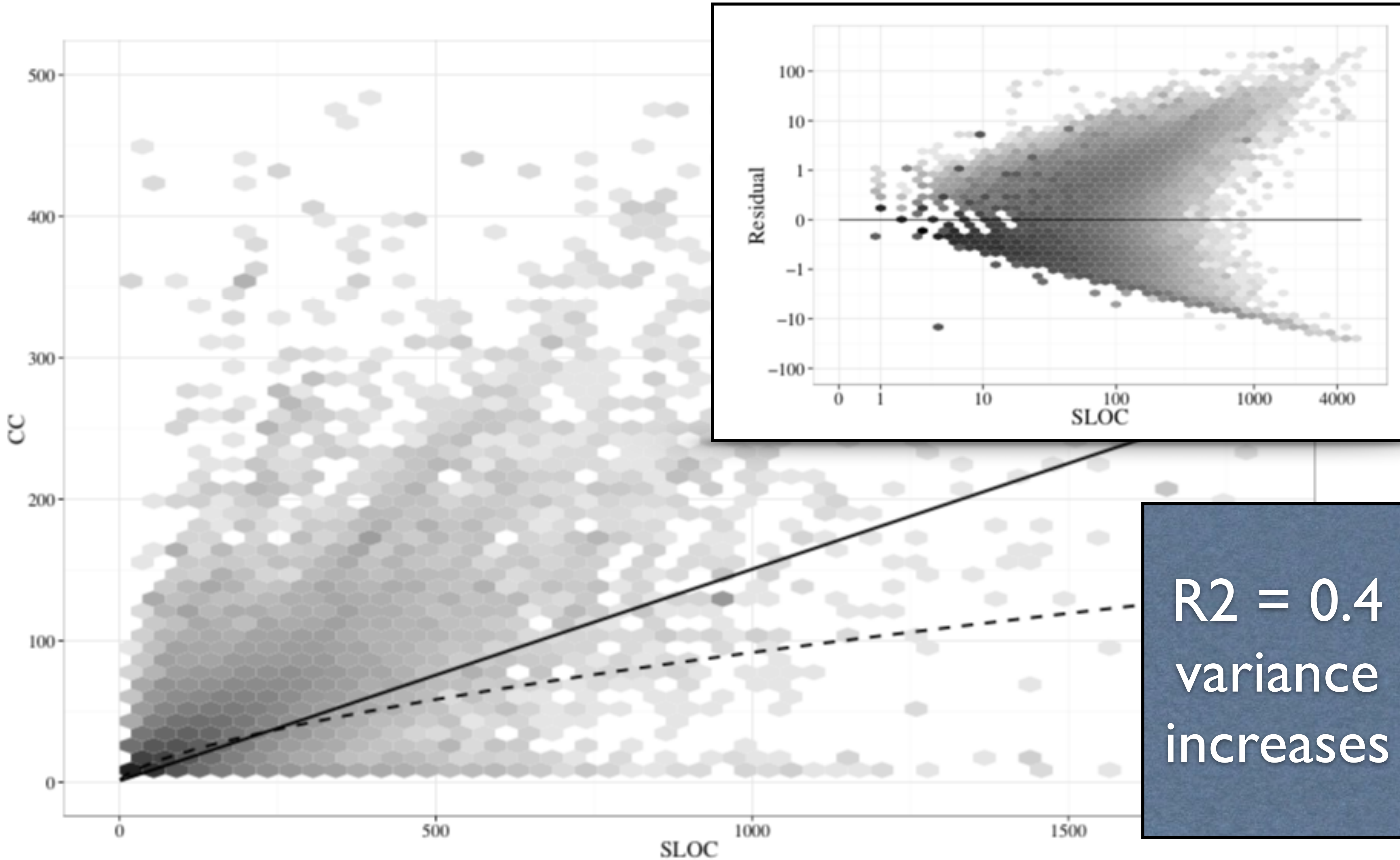
- Source Lines of Code (SLOC)
  - a measure of "volume"
  - indicating effort of reading and writing, complexity
- Cyclomatic Complexity (CC)
  - linearly independent control flow paths (how many splitting points)
  - a measure of testing effort (test cases needed to cover all blocks)
  - *indicating* effort of understanding, complexity, maybe…

Science of SLOC & CC

- Hypothesis: **SLOC** = *a* * **CC** + *b* ?
  - both a measure of volume? which other dimension?
  - should we even measure both?
- Literature on this on smaller corpora
  - answer yes
  - answer yes, when summed up to the file level
  - answer yes, if we apply logarithmic transformations
- Let’s check this.
  - because in theory a lot more code is possible
  - because repeated sum (multiplication) is the essence of “linearity”



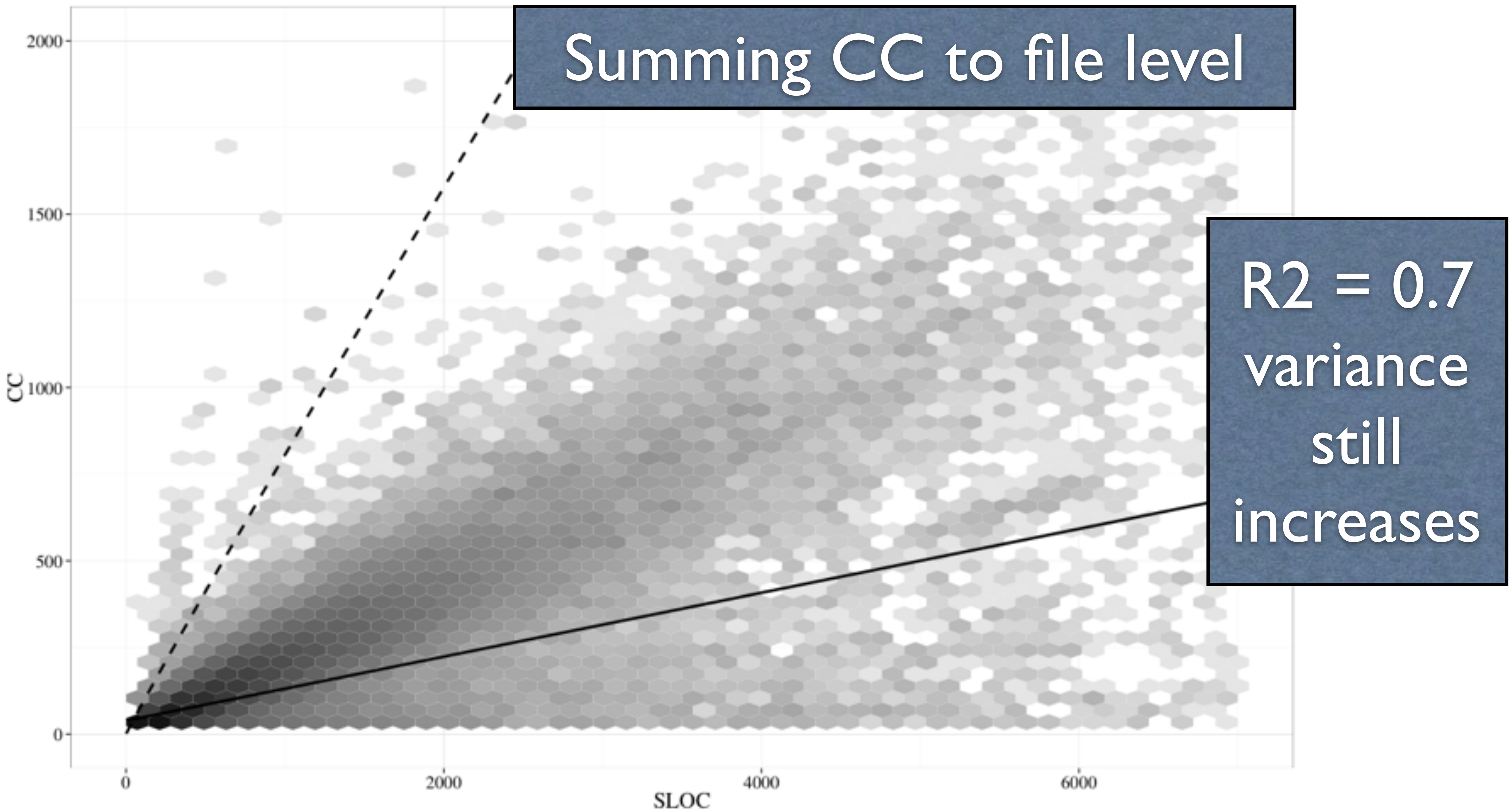
Scatter plots



R2 = 0.4
variance increases

17.6 million methods

Transformations and Aggregation

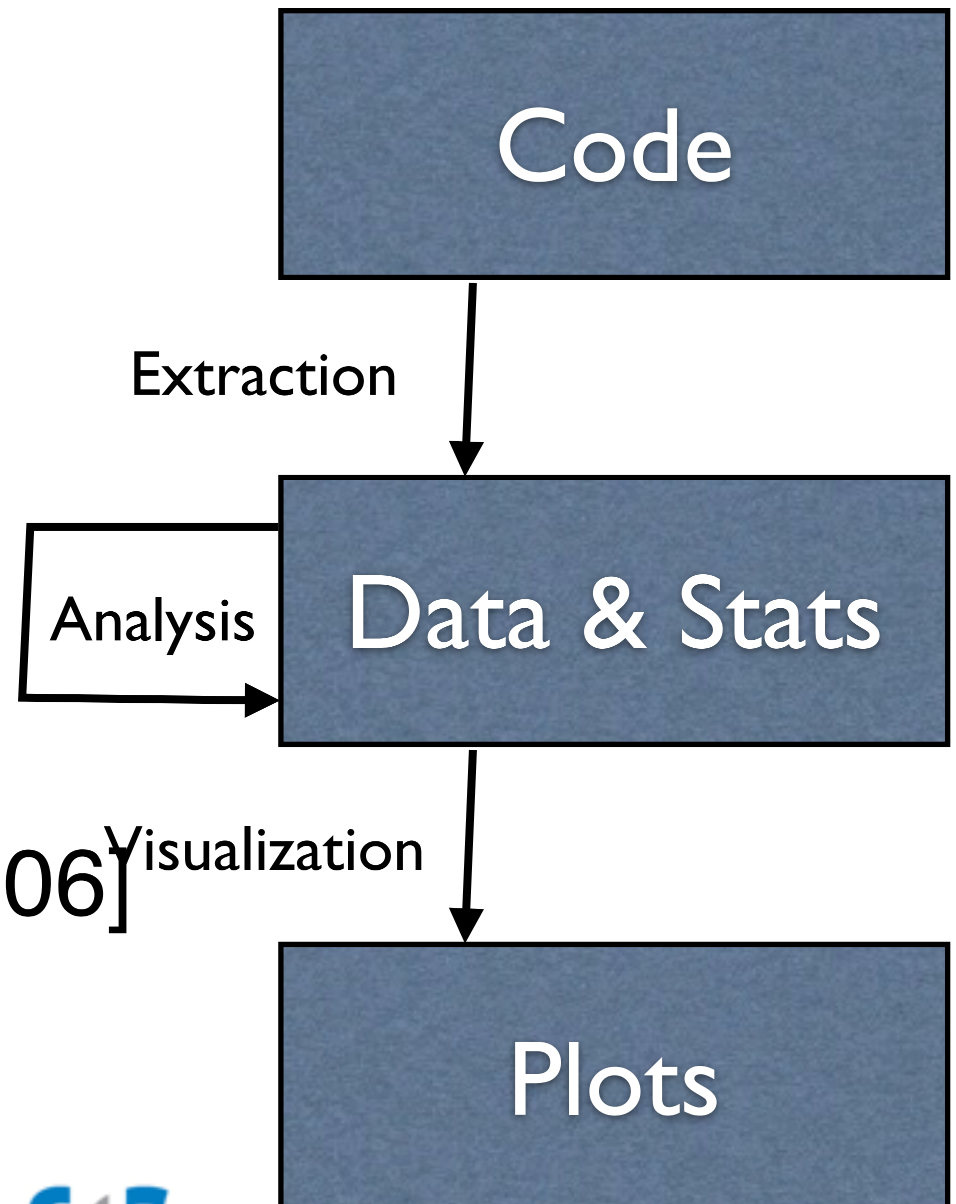


Sum makes correlation better…

A/B test shows that aggregation is indeed a cause of strong correlation

The truth about CC/SLOC

- No linear correlation
- “Dissappointing" truth
- “Actionable”
  - keep on measuring CC!
- Avoided the interpretation of CC
  - see [SCAM2012] and [Abran ’06]
- Application
  - Software Improvement Group



Software Complexity Agenda

Software Complexity Agenda




- *Philosophy (what is software complexity?)*

Software Complexity Agenda

- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*

Software Complexity Agenda

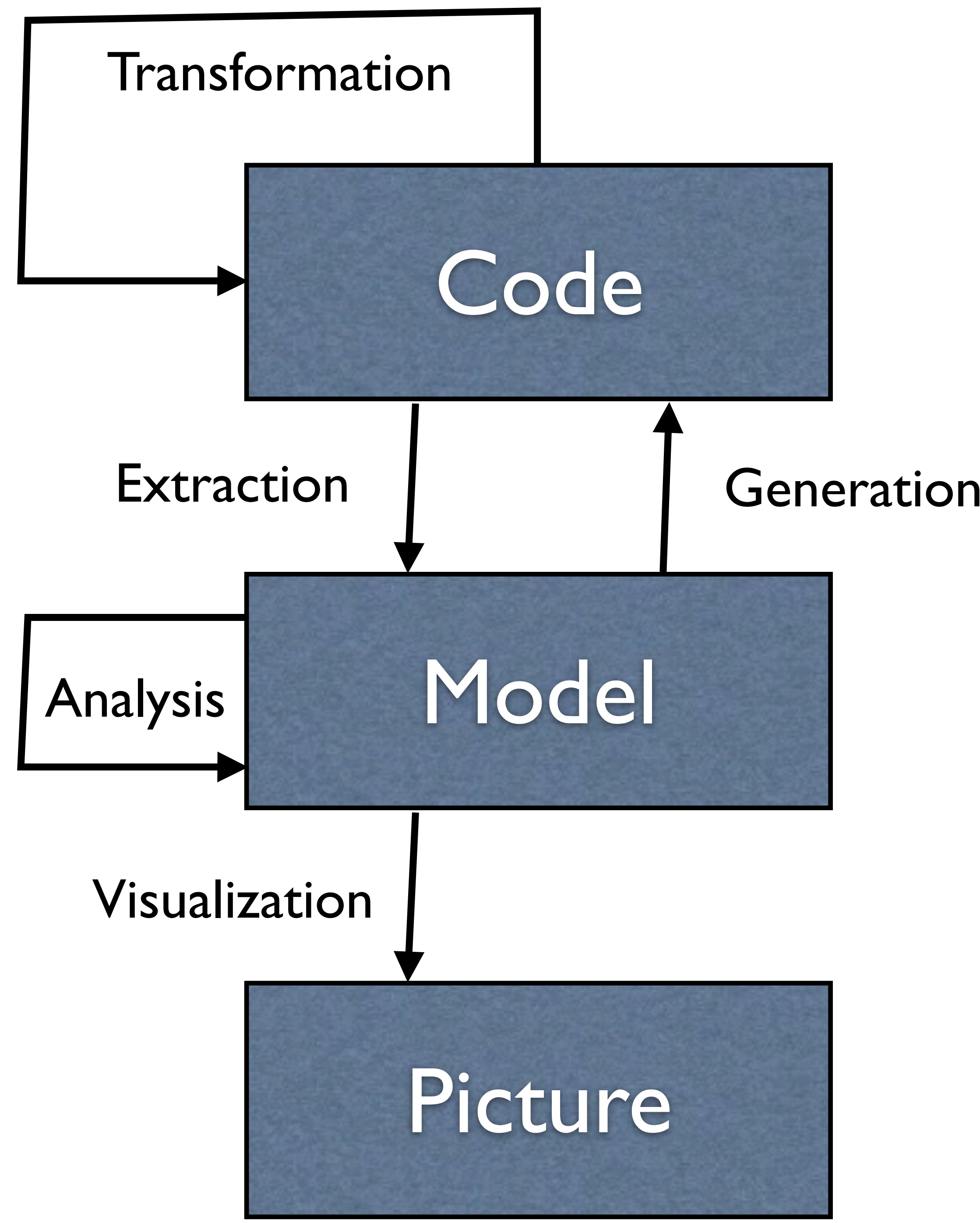
- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*
- *Engineering*
    - *Maintenance (what can we do about it?)*
    - *Construction (how can we prevent it?)*:

Software Complexity Agenda

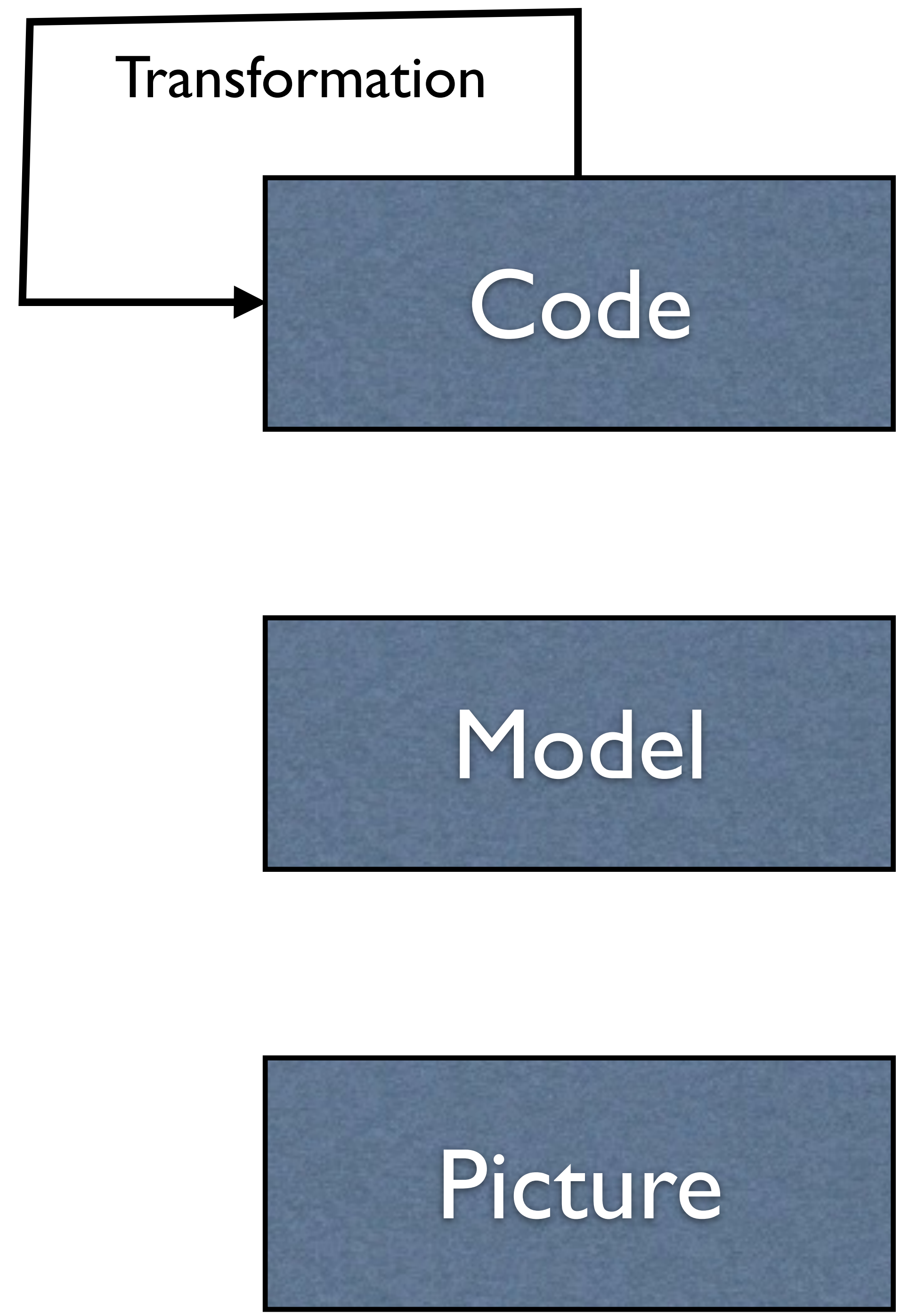
- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*
- *Engineering*
    - *Maintenance (what can we do about it?)*
    - *Construction (how can we prevent it?)*:
- Conclusion (holistic perspective)
    - Meta-tools
    - Public/private collaboration

Maintenance

- Activities:
  - Reverse engineering
  - Re-engineering
  - Visualization
  - Refactoring
- "understanding" specimens
- about efficiency and effectivity
  - tools for getting it right, faster
  - tools for mitigating complexity



- Refactoring is improving internal quality
  - reducing complexity
  - without changing functionality.



```
public abstract class AbstractCollection implements Collection {
  public void addAll(AbstractCollection c) {
    if (c instanceof Set) {
      Set s = (Set)c;
      for (int i=0; i < s.size(); i++) {
        if (!contains(s.getElementAt(i))) {
          add(s.getElementAt(i));
        }
      }
    } else if (c instanceof List) {
      List l = (List)c;
      for (int i=0; i < l.size(); i++) {
        if (!contains(l.get(i))) {
          add(l.get(i));
        }
      }
    } else if (c instanceof Map) {
      Map m = (Map)c;
      for (int i=0; i<m.size(); i++)
        add(m.keys[i], m.values[i]);
    }
  }
```

Duplicated Code

Duplicated Code

Alternative Classes with Different Interfaces

Switch Statement

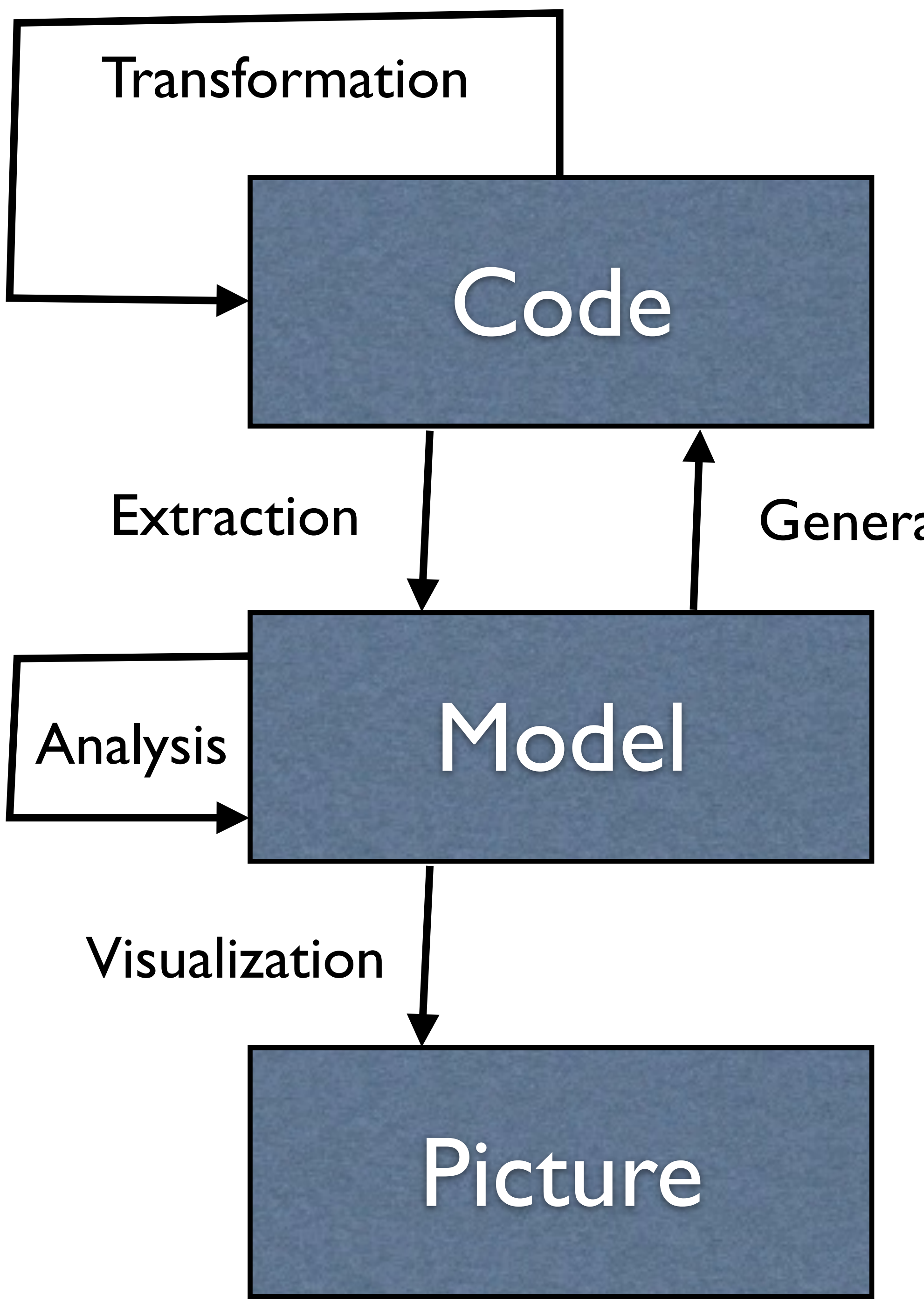
Inappropriate Intimacy

Long Method

[Joshua Kerievsky, industriallogic.com]

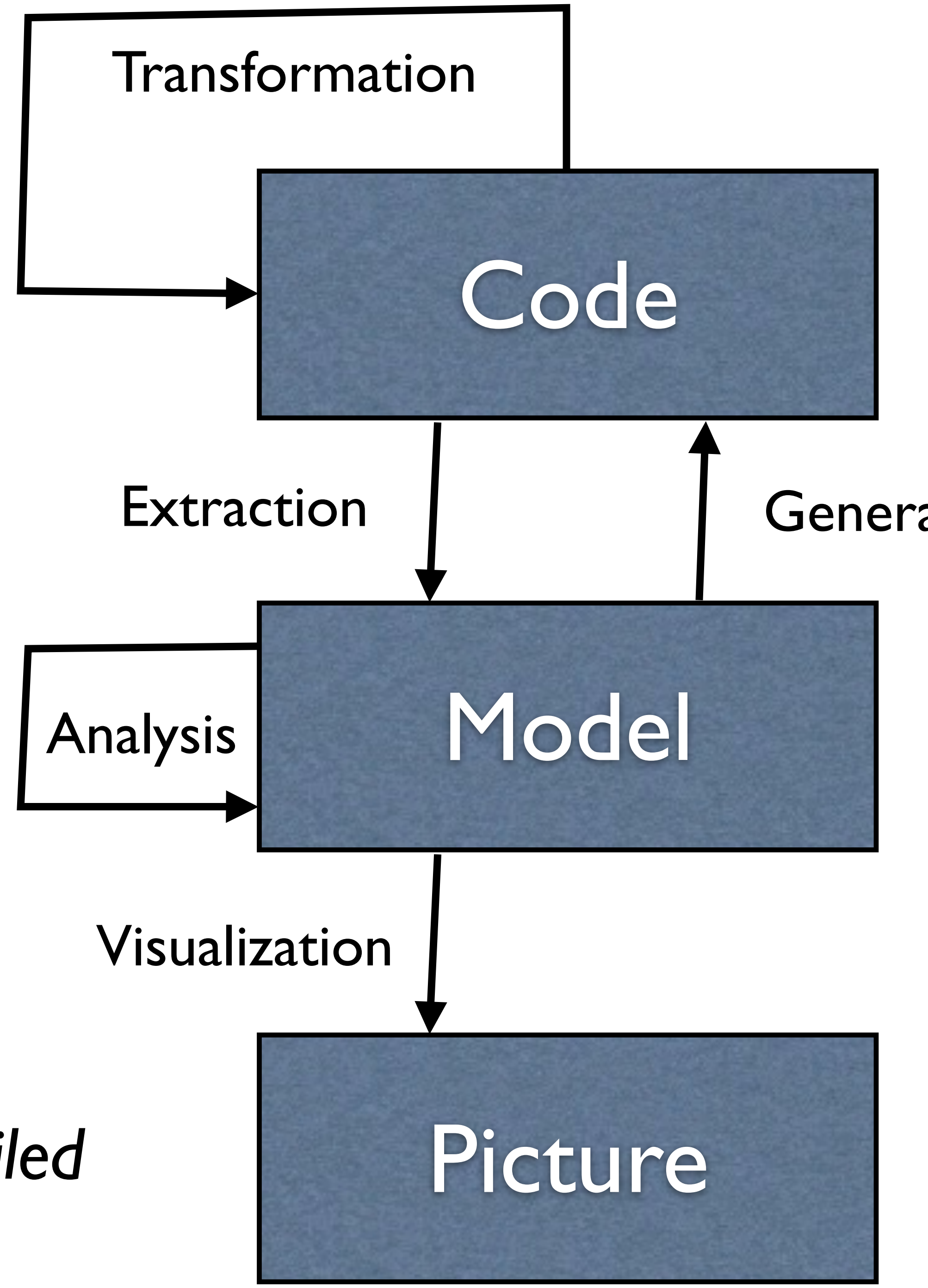
Refactoring Tools

- help by:
  - analyzing conditions
  - transforming everywhere
  - user interactions
    - preview
    - undo

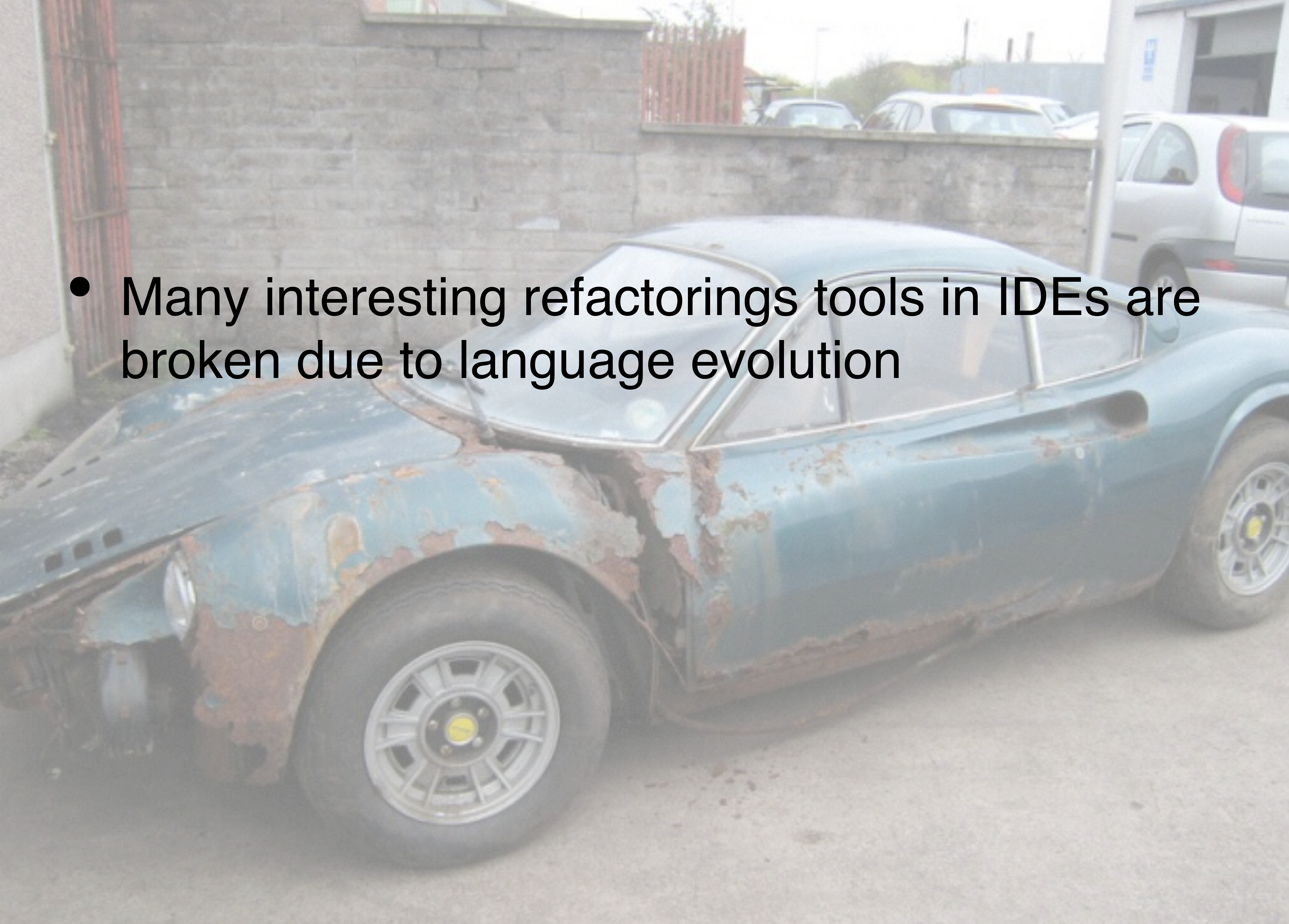


Refactoring Tools

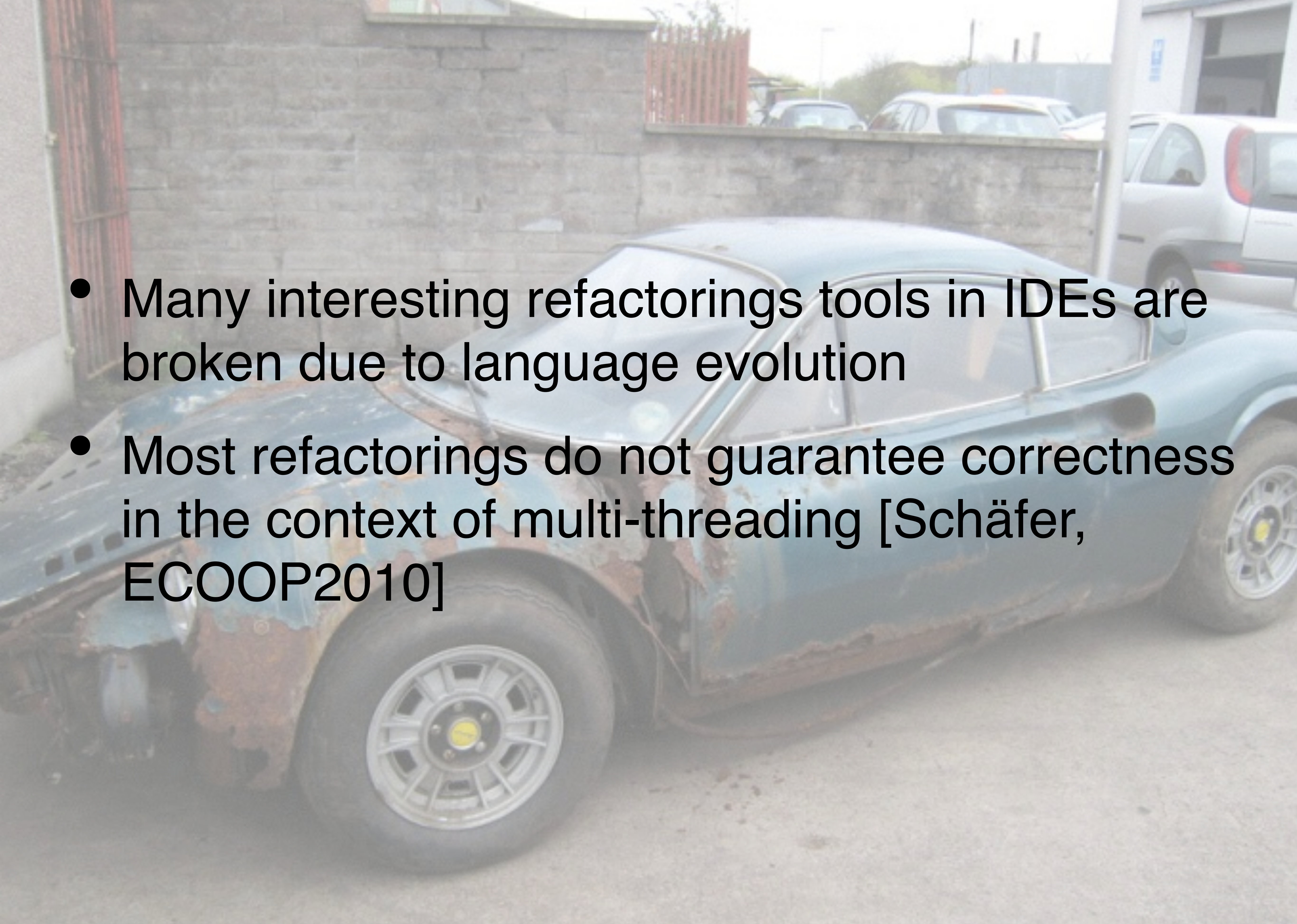
- help by:
  - analyzing conditions
  - transforming everywhere
  - user interactions
    - preview
    - undo



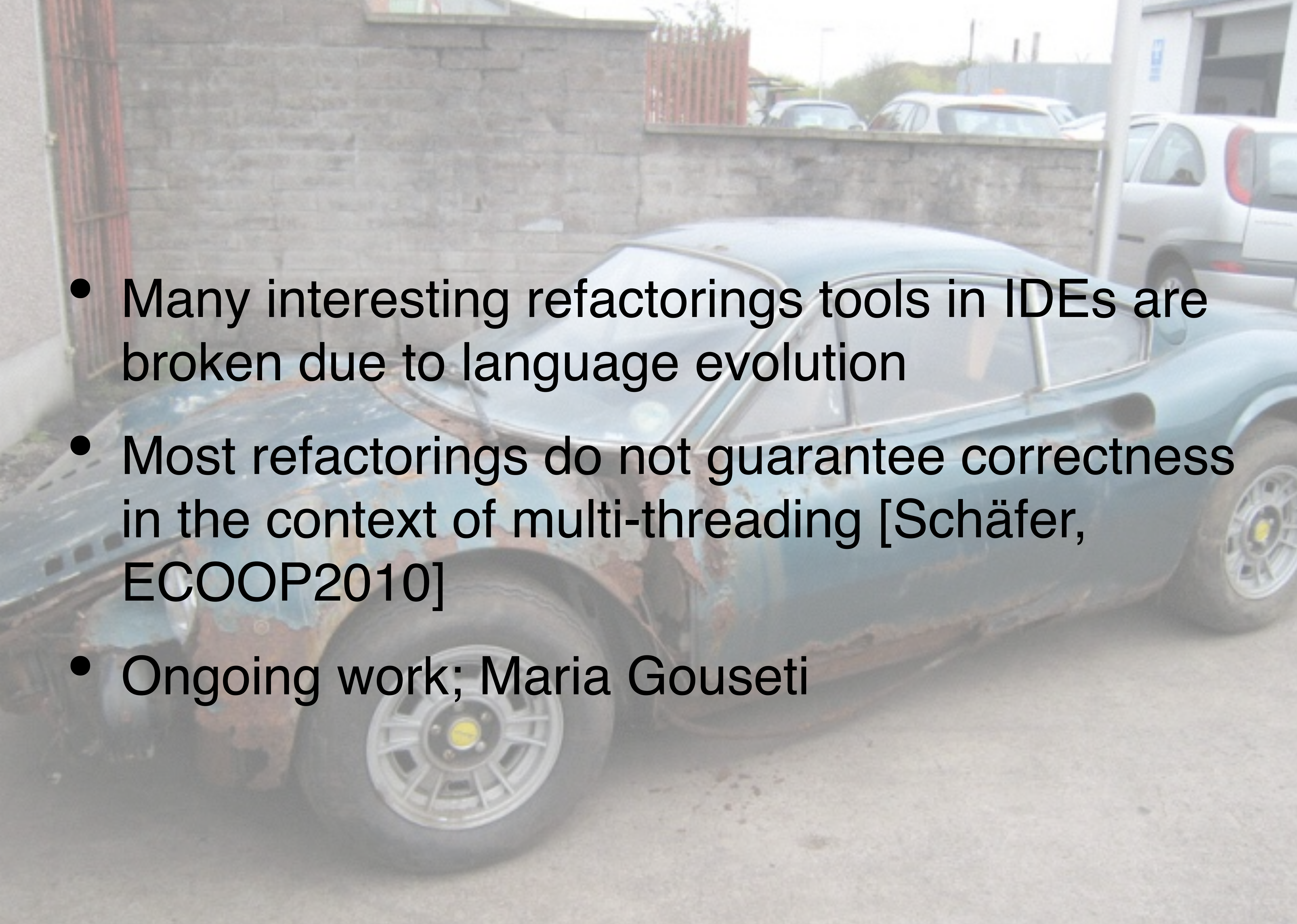
The value and heavy lifting is in the highly detailed model of programming language syntax, static and dynamic semantics



- Many interesting refactorings tools in IDEs are broken due to language evolution



- Many interesting refactorings tools in IDEs are broken due to language evolution
- Most refactorings do not guarantee correctness in the context of multi-threading [Schäfer, ECOOP2010]



- Many interesting refactorings tools in IDEs are broken due to language evolution
- Most refactorings do not guarantee correctness in the context of multi-threading [Schäfer, ECOOP2010]
- Ongoing work; Maria Gouseti

```
class C2 implements TM {
    static class A {
        synchronized static void m() {}
        synchronized static void n() {}
    }
    static class B {
    }
    @Override
    public void m1() {
        synchronized (B.class) { A.m(); }
    }
    @Override
    public void m2() {
        synchronized (A.class) { A.n(); }
    }
}
```

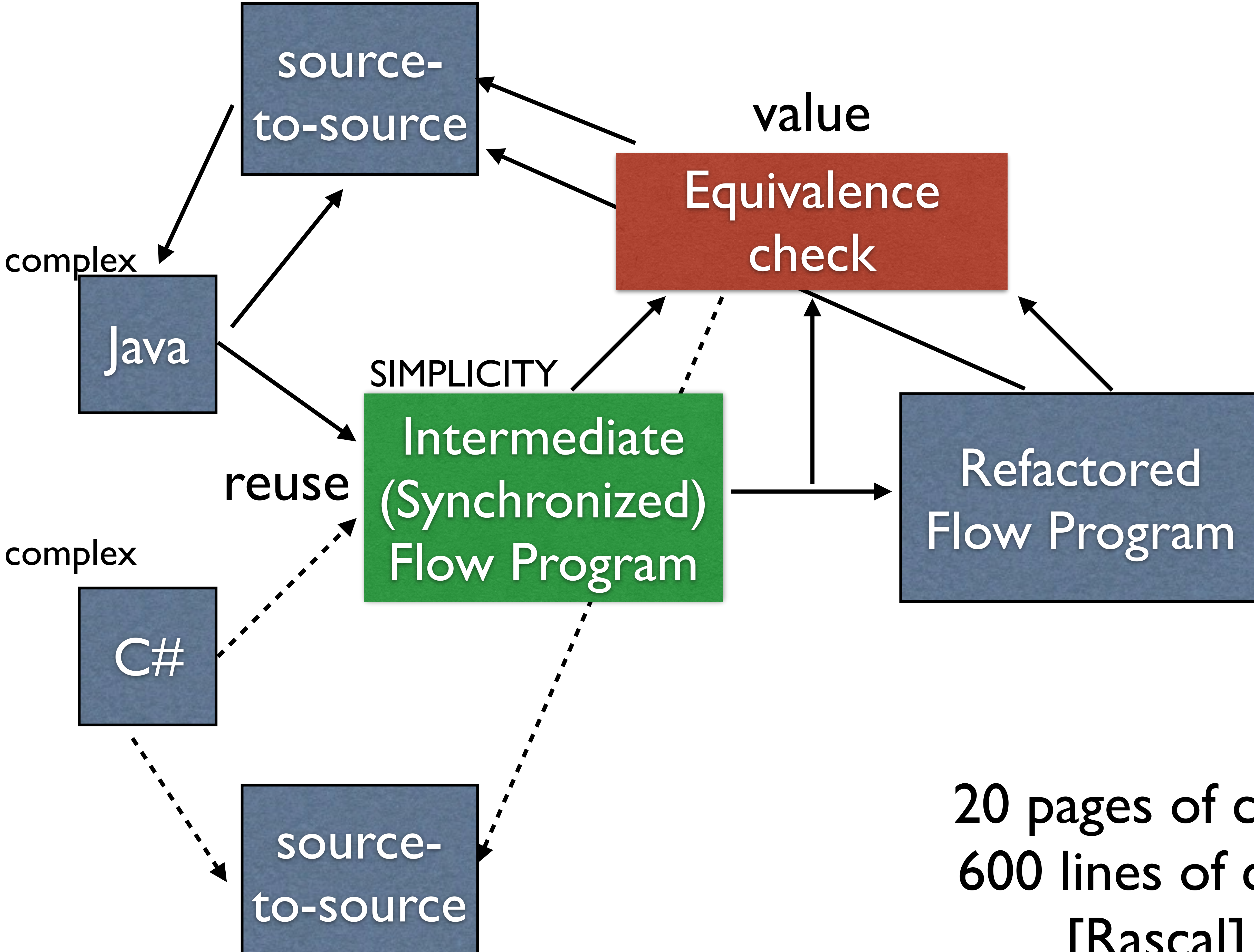
Original

```
class C2 implements TM {
    static class A {
        synchronized static void m() {}
    }
    static class B {
        synchronized static void n() {}
    }
    @Override
    public void m1() {
        synchronized (B.class) { A.m(); }
    }
    @Override
    public void m2() {
        synchronized (A.class) { B.n(); }
    }
}
```

Refactored

MOVE METHOD introduces a deadlock, when *m1()* locks on *B.class* and *m2()* locks on *A.class* and both threads are blocked on the lock held by the other one

[Schäffer 2010]



20 pages of code,
600 lines of code
[Rascal]

vs

- Refactoring can tools help improving quality
- They are complicated
- First simplify the tools
- Then simplify the code

vs

- Refactoring can tools help improving quality
- They are complicated
- First simplify the tools
- Then simplify the code

What if programmers spend less time on debugging accidental problems and spend it on hard features for business value instead?

Software Complexity Agenda

Software Complexity Agenda

- *Philosophy (what is software complexity?)*

Software Complexity Agenda

- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*

Software Complexity Agenda

- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*
- *Engineering*
  - *Maintenance (what can we do about it?)*
  - *Construction (how can we prevent it?)*:

Software Complexity Agenda

- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*
- *Engineering*
  - *Maintenance (what can we do about it?)*
  - *Construction (how can we prevent it?)*:
- Conclusion (holistic perspective)
  - Meta-tools
  - Public/private collaboration

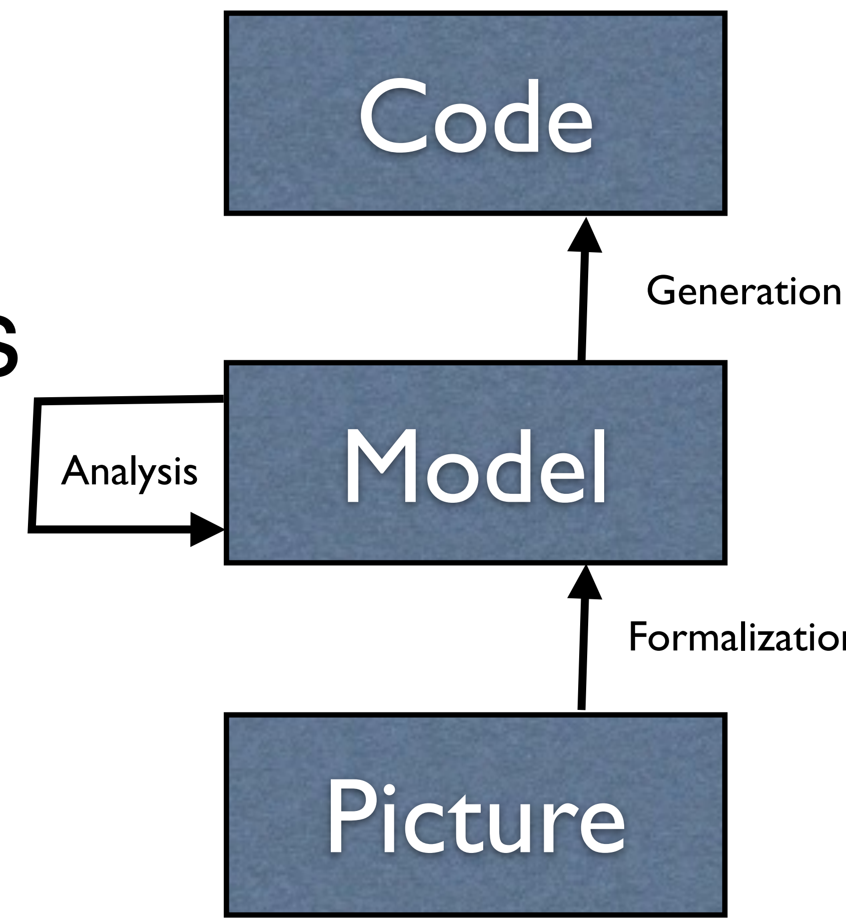
Construction

- Correct-by-construction
- Variability by prediction
- Model Driven Engineering
- Software Architecture
- Formal Methods
- Programming languages
- “make better software”

n

Domain Specific Languages

- Requirements=domain analysis
- Separate what is fixed from what is variable (predict)
- Language for domain experts
- No accidental complexity
- Multiple back-ends
  - Technology evolution
  - Different Audiences



Digital Forensics

[Jeroen van den Bos, Tijs van der Storm]

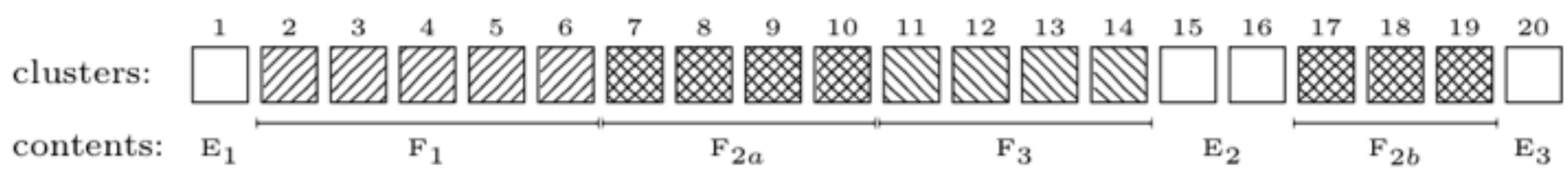


Fig. 1. An example set of contiguous clusters on a storage device

- Digital evidence is messy
- Technology is highly variable (cameras, formats)
- Evidence needs to be collected from terabytes within days

Derric Language

```
format PNG
  strings ascii
  sign false
  unit byte
  size 1
  type integer

sequence
  Signature IHDR
  Chunk* IDAT IDAT* Chunk*
  IEND

structures
Signature {
  marker: 137,80,78,71,13,10,26,10;
}

Chunk {
  length: lengthOf(chunkdata) size 4;
  chunktype: !"IDAT" size 4;
  chunkdata: size length;
  crc: checksum(algorithm="crc32-ieee",
         init="allone",start="lsb",
         end="invert",store="msbfirst",
         fields=chunktype+chunkdata)
       size 4;
}

IHDR = Chunk {
  chunktype: "IHDR";
  chunkdata: {
    width: !0 size 4;
    height: !0 size 4;
    bitdepth: 1|2|4|8|16;
    colourtype: 0|2|3|4|6;
    compression: 0;
    filter: 0;
    interlace: 0|1;
  }
}

IDAT = Chunk {
  chunktype: "IDAT";
  chunkdata: compressed(
               algorithm="deflate",
               layout="zlib",
               fields=chunkdata)
             size length;
}

IEND {
  length: 0 size 4;
  chunktype: "IEND";
  crc: 0xAE, 0x42, 0x60, 0x82;
}
```

https://github.com/jvdb/derric

Derric Results

Component	Implementation	Size (SLOC)
Grammar	RASCAL	52
JPEG description	DERRIC	92
PNG description	DERRIC	58
Structure-based matching (code generator)	RASCAL	510
Bifragment gap (runtime)	Java	72
Brute force (runtime)	Java	44
Utilities (runtime)	Java	256
	Total:	1084

- Just as fast or faster than hand-optimized C++ code
- Derric definitions retargeted to other algorithms
- Derric definitions transformed for speed trade-offs

[ICSE'11, ICMT'12, ECFMA'13]

Software Complexity Agenda

Software Complexity Agenda




- *Philosophy (what is software complexity?)*

Software Complexity Agenda

- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*

Software Complexity Agenda

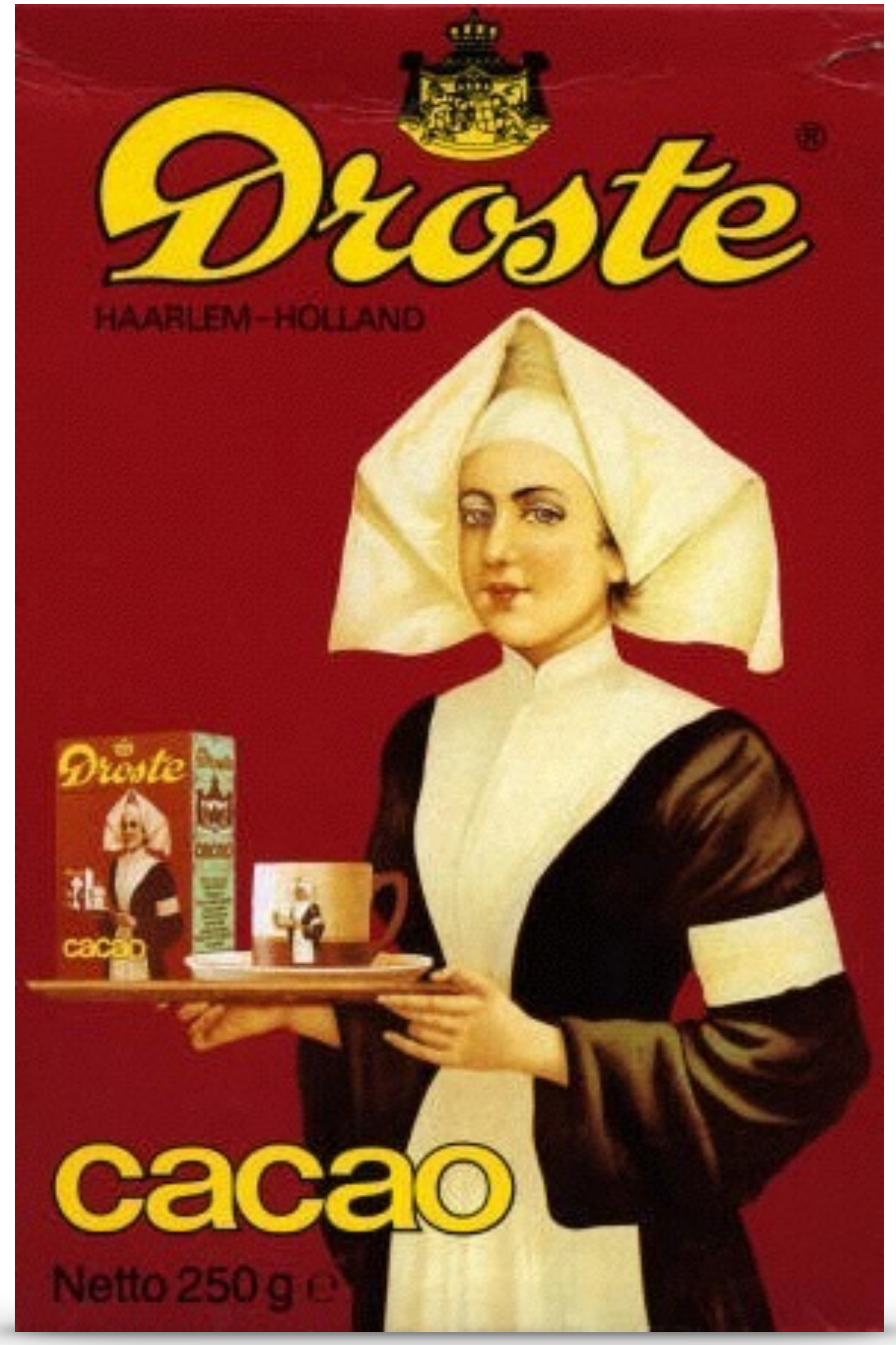
- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*
- *Engineering*
  - *Maintenance (what can we do about it?)*
  - *Construction (how can we prevent it?)*:

Software Complexity Agenda

- *Philosophy (what is software complexity?)*
- *Science (what is the truth about software complexity?)*
- *Engineering*
  - *Maintenance (what can we do about it?)*
  - *Construction (how can we prevent it?)*:
- Conclusion (holistic perspective)
  - Meta-tools
  - Public/private collaboration

Holistic & Reflective

- **Key**: software which reads and writes software
  - Science
  - Maintenance
  - Construction
- **Meta Domain**
  - tools share similar character
  - transfer *theory* to industry
  - transfer *knowledge* to research



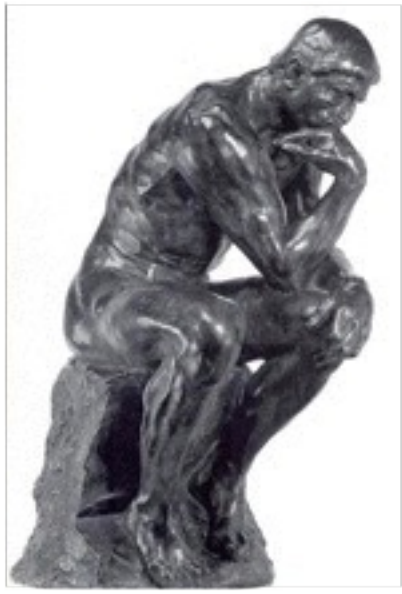
Symbiosis

Symbiosis

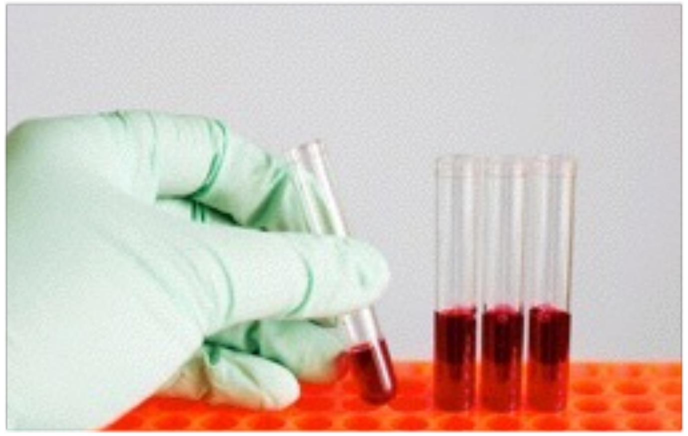
- Maintenance and Construction need scientific and industrial validation
- Maintenance and Construction need input from **Mining**
- Science needs “what if” scenarios; hypotheses
- Maintenance and Construction need programming language models, analysis, visualization, generation, …
- Industry needs predictions, tools, expert engineers
- Academia needs data, domain expertise and researchers

Public/Private collaboration

Tools enable exchange



Research



Engineering

Collaboration Portfolio

- Science
  - Software Improvement Group
  - OSSMETER EU Project (www.ossmeter.org) (holistic quality assessment)
  - Code (metrics), Meta-data (versions, bugs, questions), Natural language (sentiments)
- Maintenance
  - Dutch Banking/Insurance companies (re-engineering, reverse engineering)
  - High-tech industries (embedded systems, networks, television)
- Construction
  - Games (EQUA project)
  - NFI (“CSI Netherlands”, evidence collection)
  - Tax office, financial auditing companies (fraud detection)
  - Banks (configuration, verification, modeling & simulation)
  - High-tech industries (protocols, state machines, configuration)

[logo’s omitted]

OSSMETER Automated Measurement and Analysis of Open Source Software

Software
Industry & Research
thrive in the current climate of
public/private collaboration

=

opportunity + <u>responsibility</u>

Software
Industry & Research
thrive in the current climate of
public/private collaboration

=

opportunity + <u>responsibility</u>

Software
Industry & Research
thrive in the current climate of
public/private collaboration

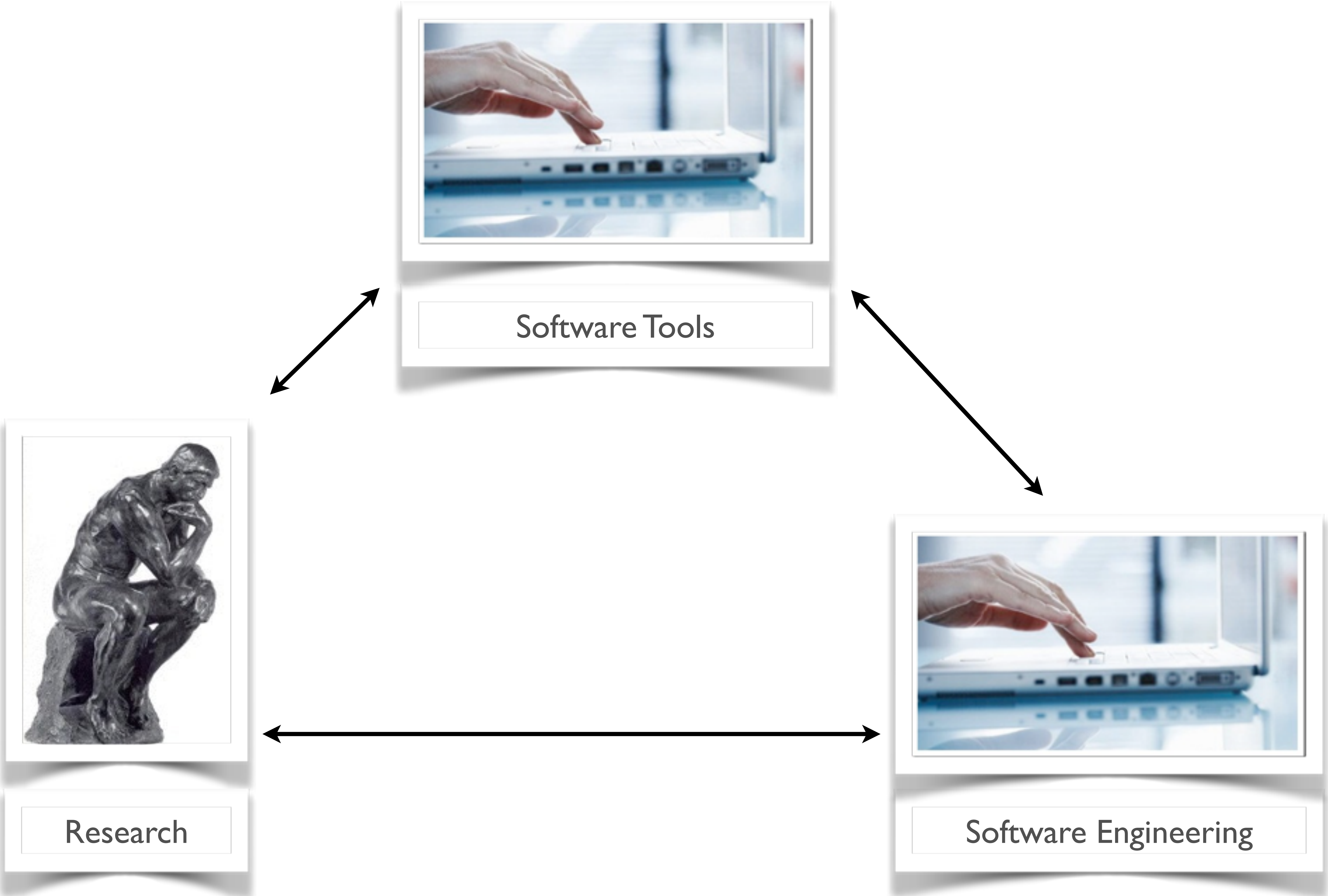
=

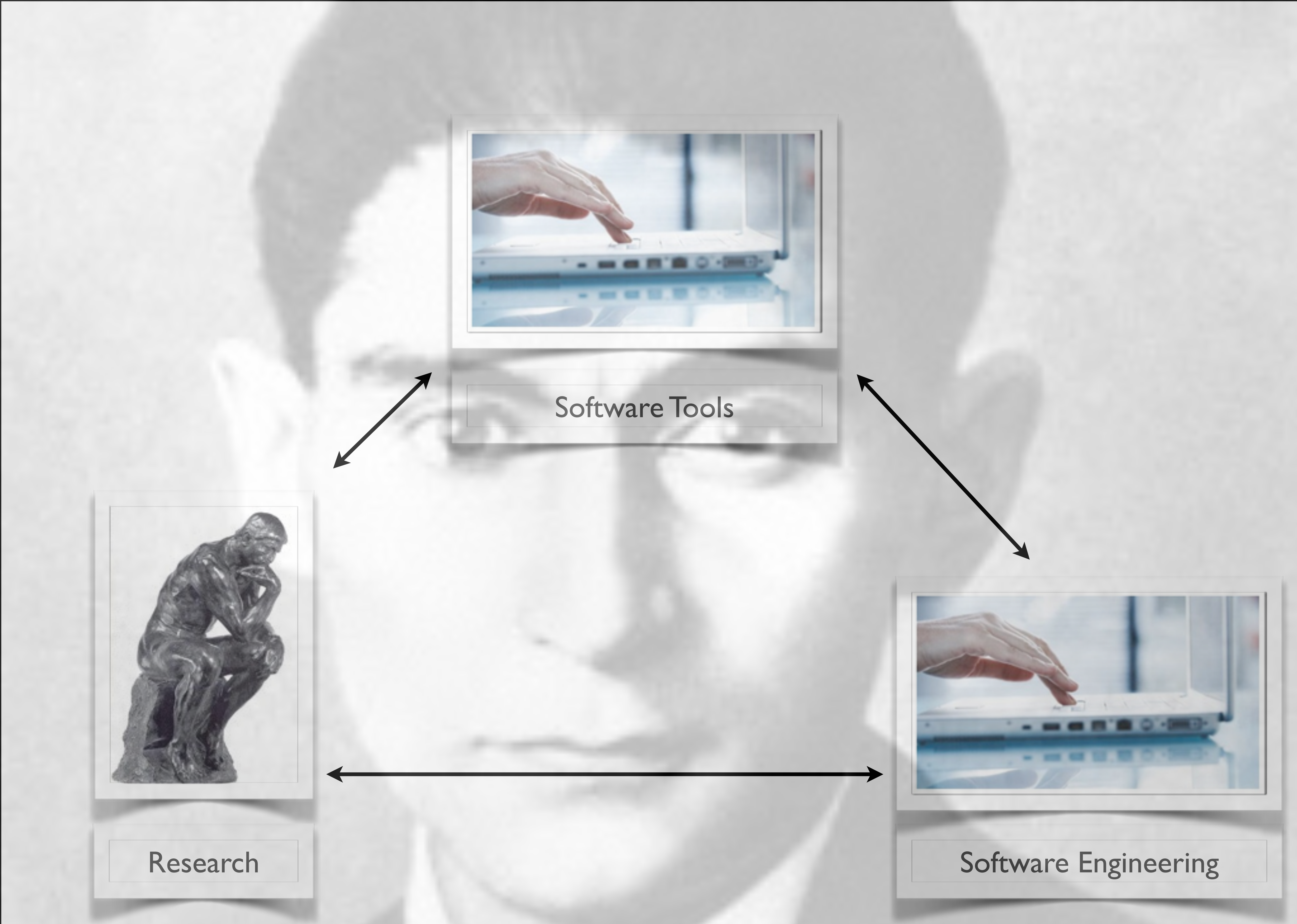
opportunity + <u>responsibility</u>

Software
Industry & Research
thrive in the current climate of
public/private collaboration

=

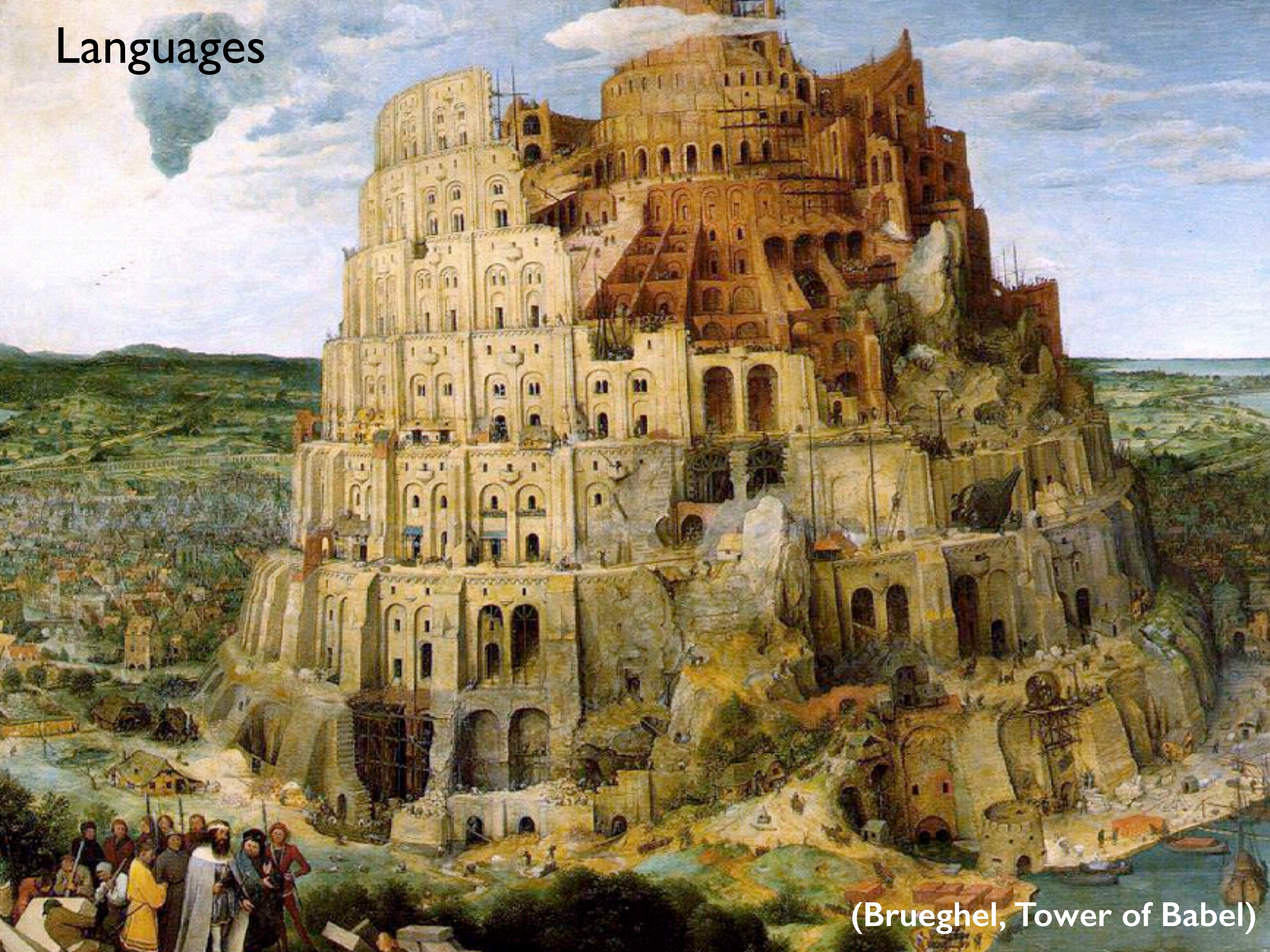
opportunity + responsibility


Software Tools
Research
Software Engineering


Software Tools
Research
Software Engineering

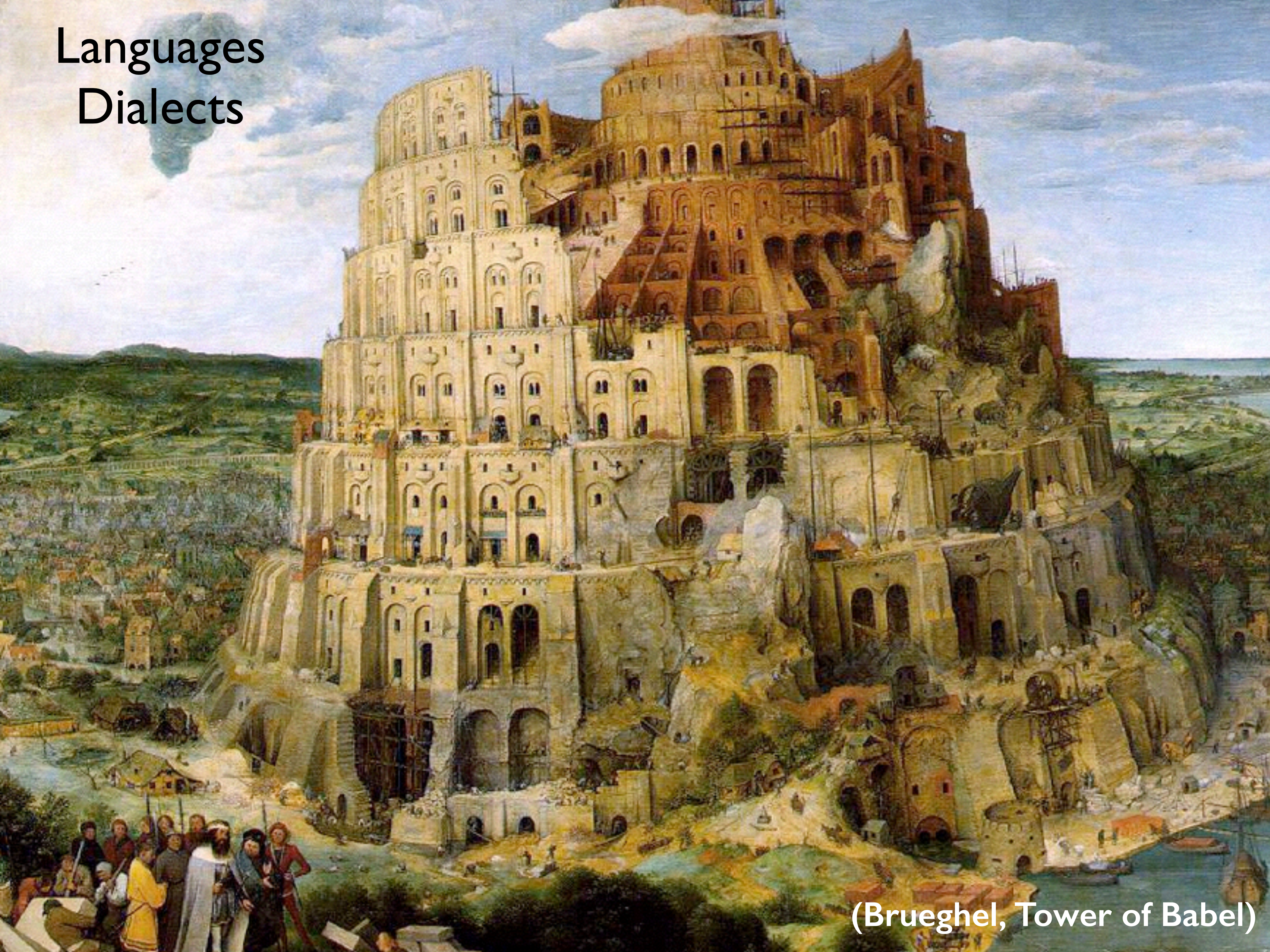
(Brueghel, Tower of Babel)

Languages



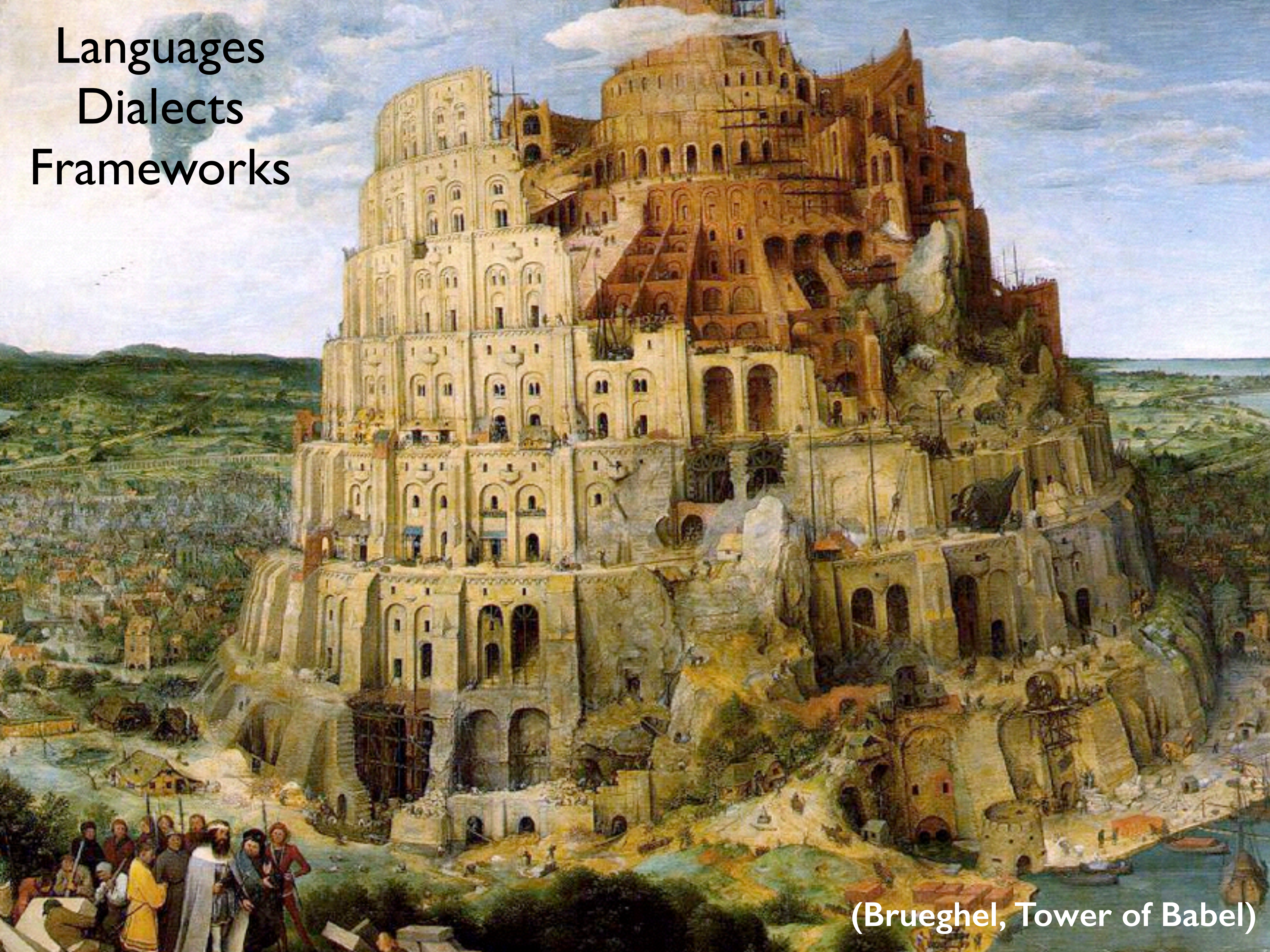
(Brueghel, Tower of Babel)

Languages
Dialects

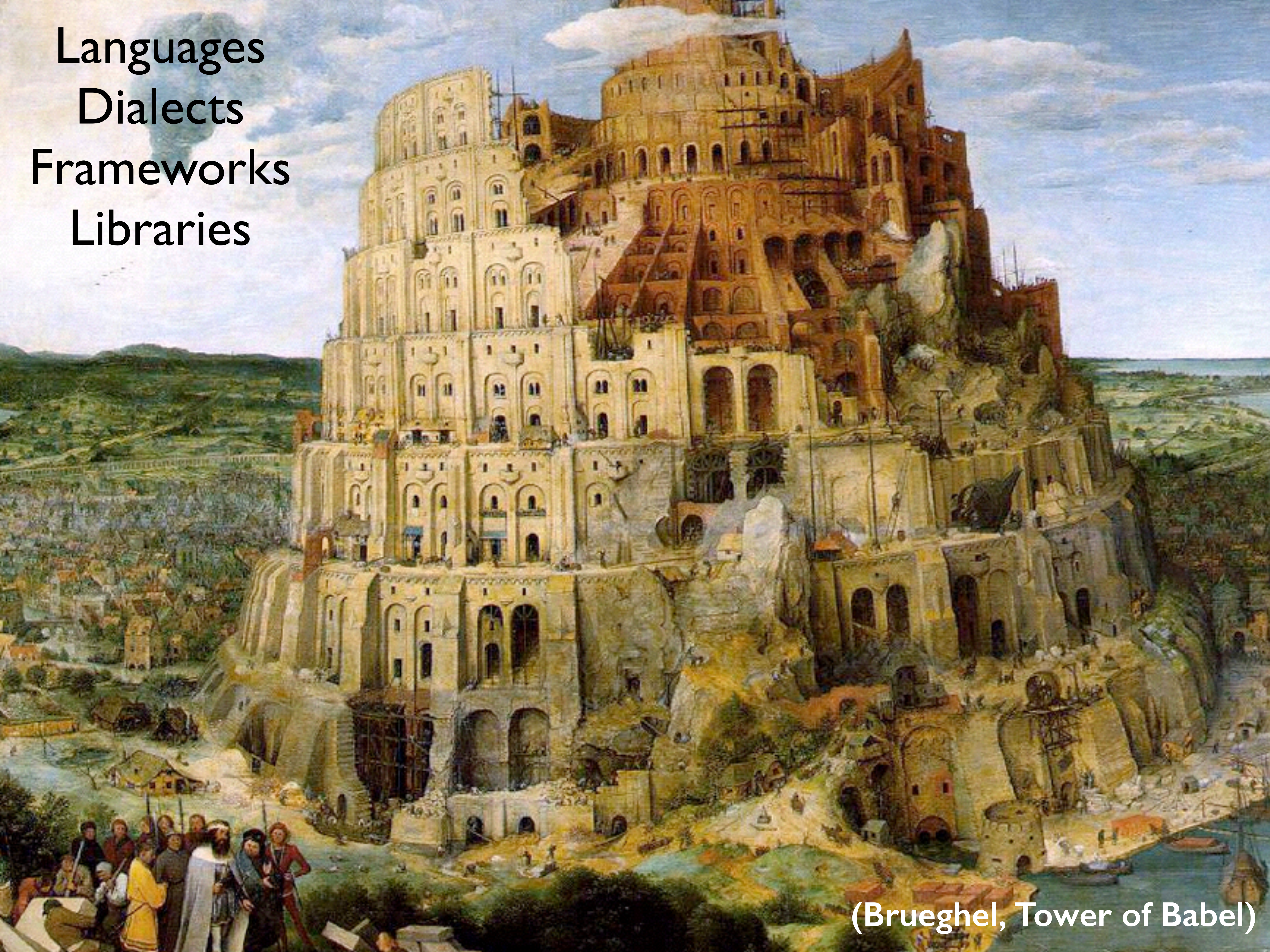


(Brueghel, Tower of Babel)

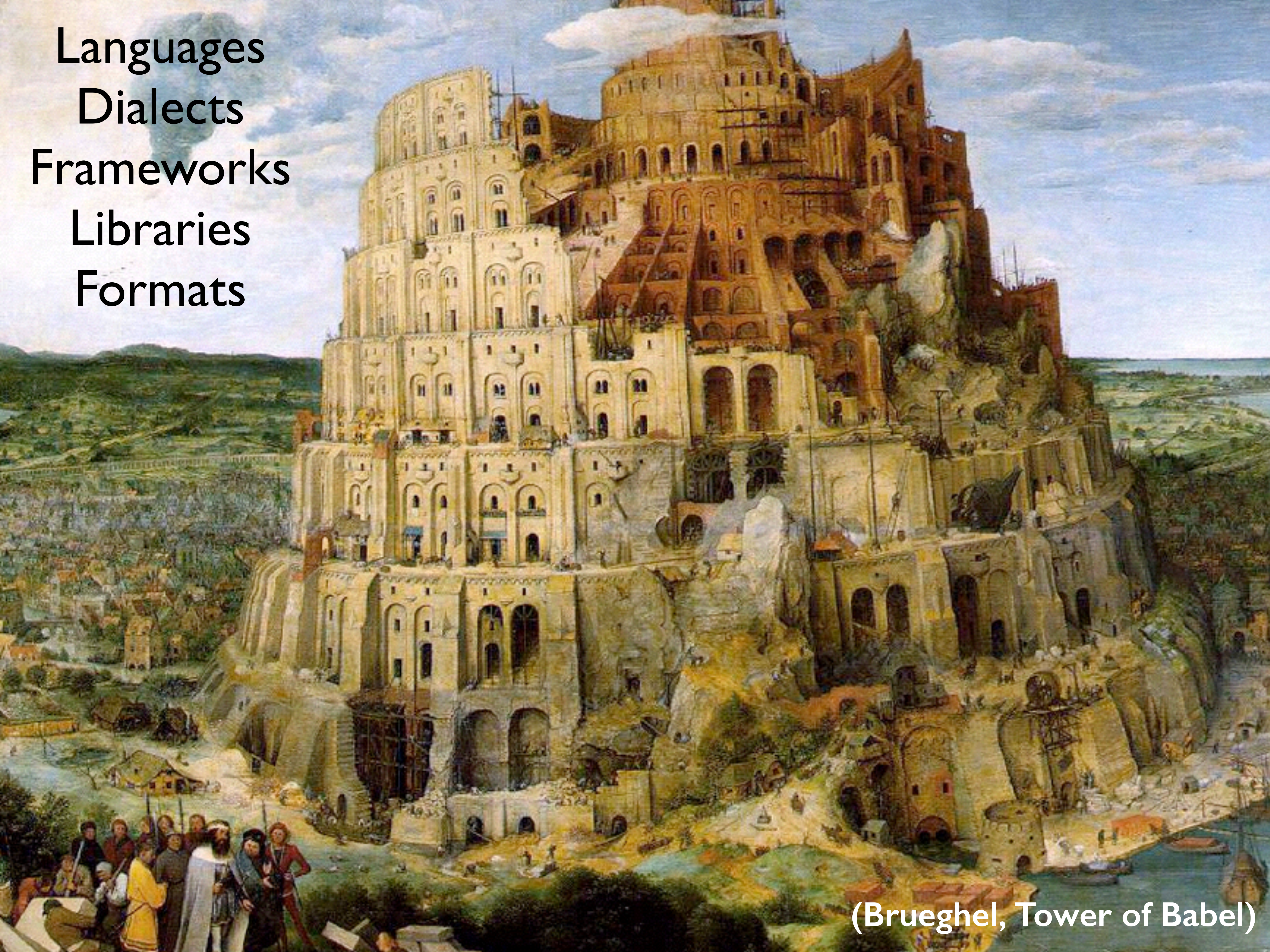
Languages
Dialects
Frameworks



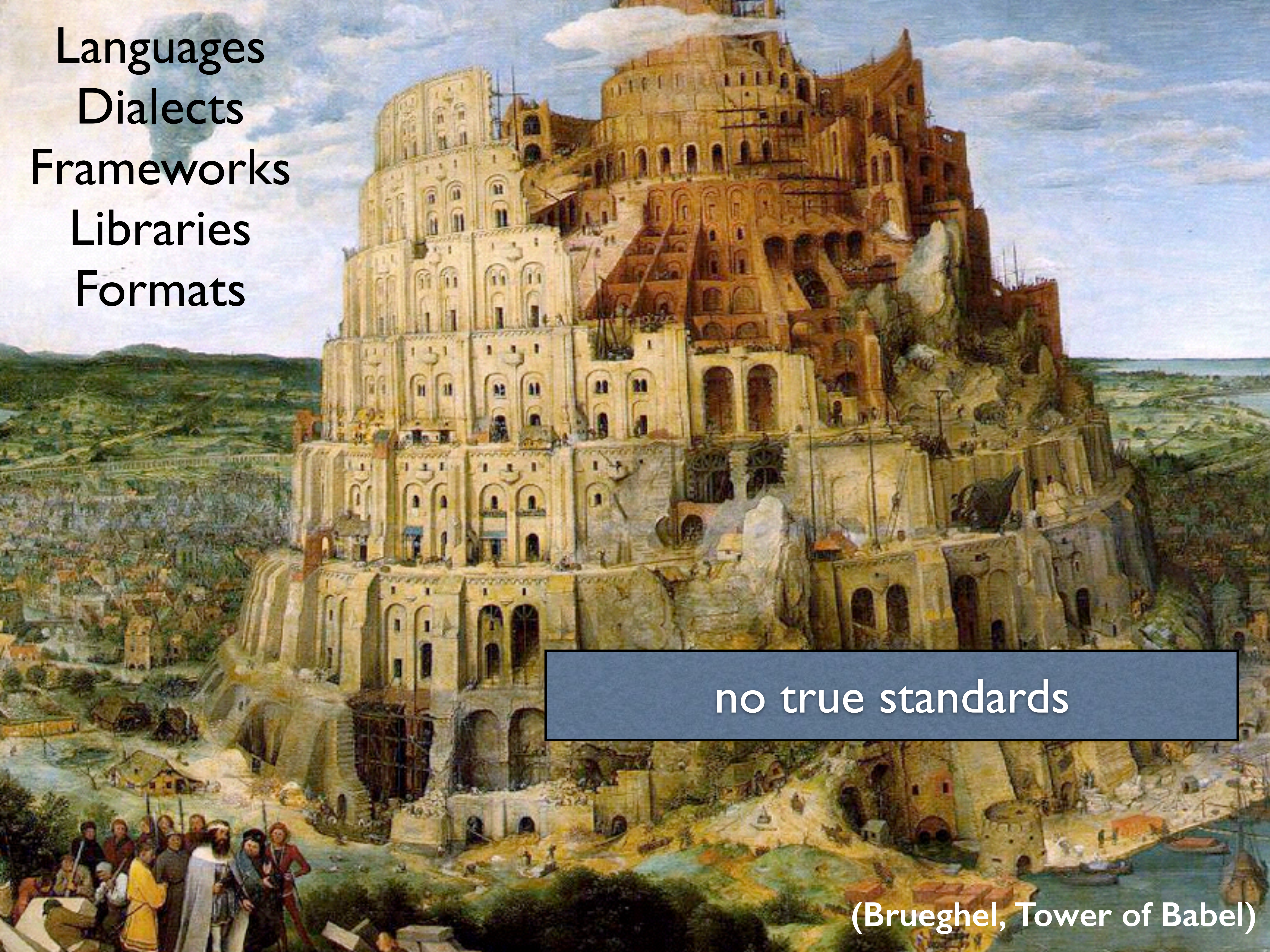
(Brueghel, Tower of Babel)



(Brueghel, Tower of Babel)



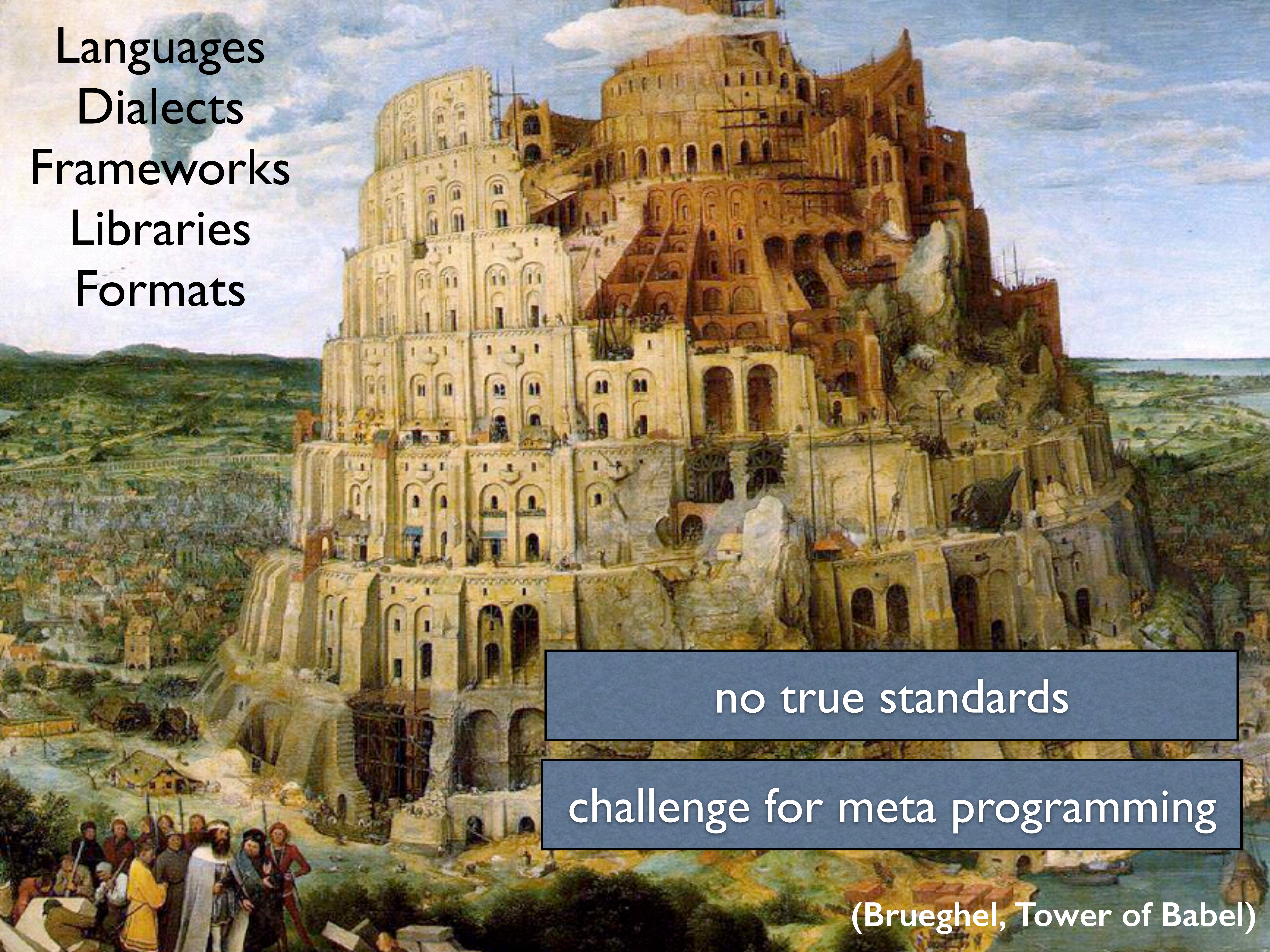
(Brueghel, Tower of Babel)


Languages
Dialects
Frameworks
Libraries
Formats
no true standards
(Brueghel, Tower of Babel)

Languages
Dialects
Frameworks
Libraries
Formats

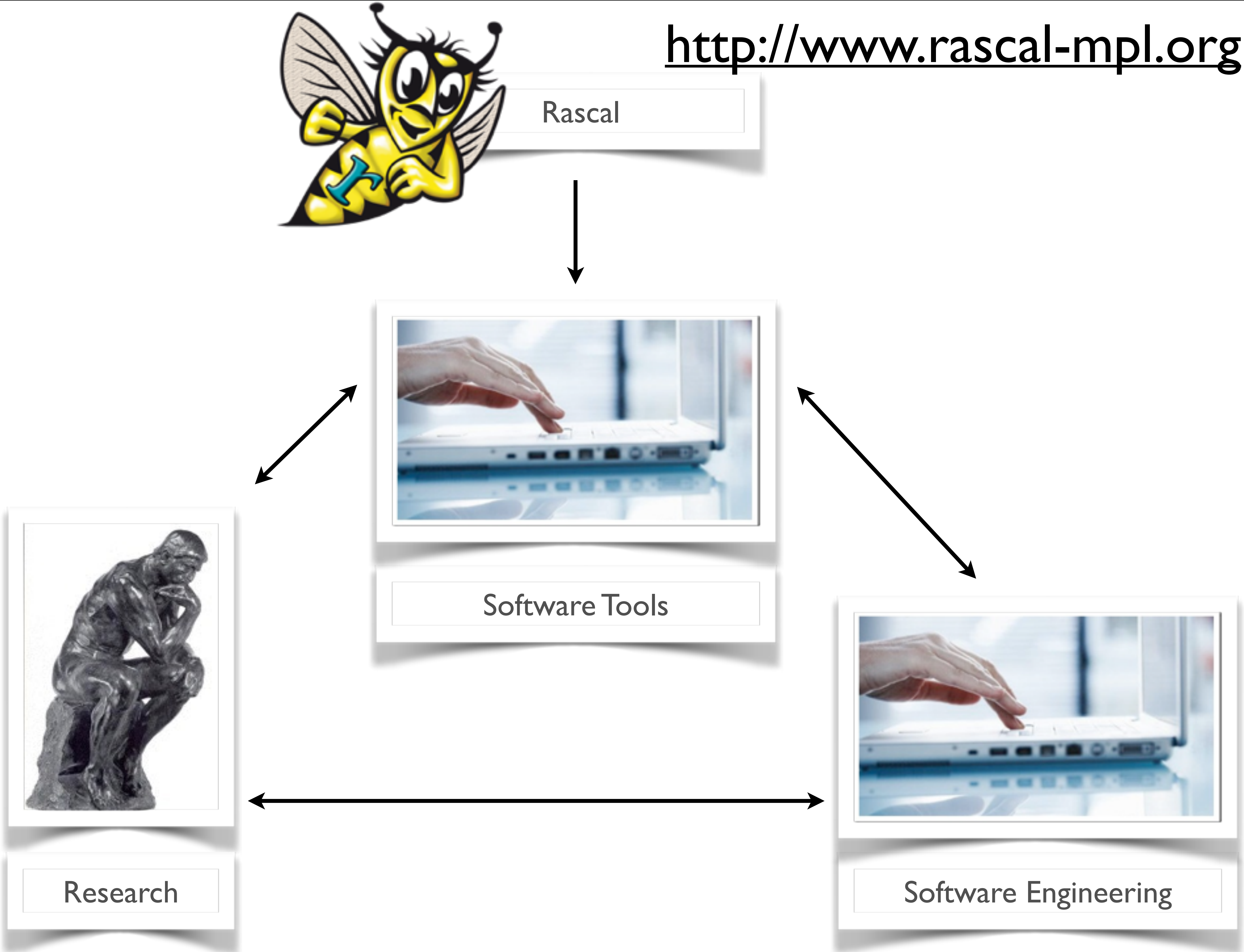
no true standards

challenge for meta programming



(Brueghel, Tower of Babel)

http://www.rascal-mpl.org

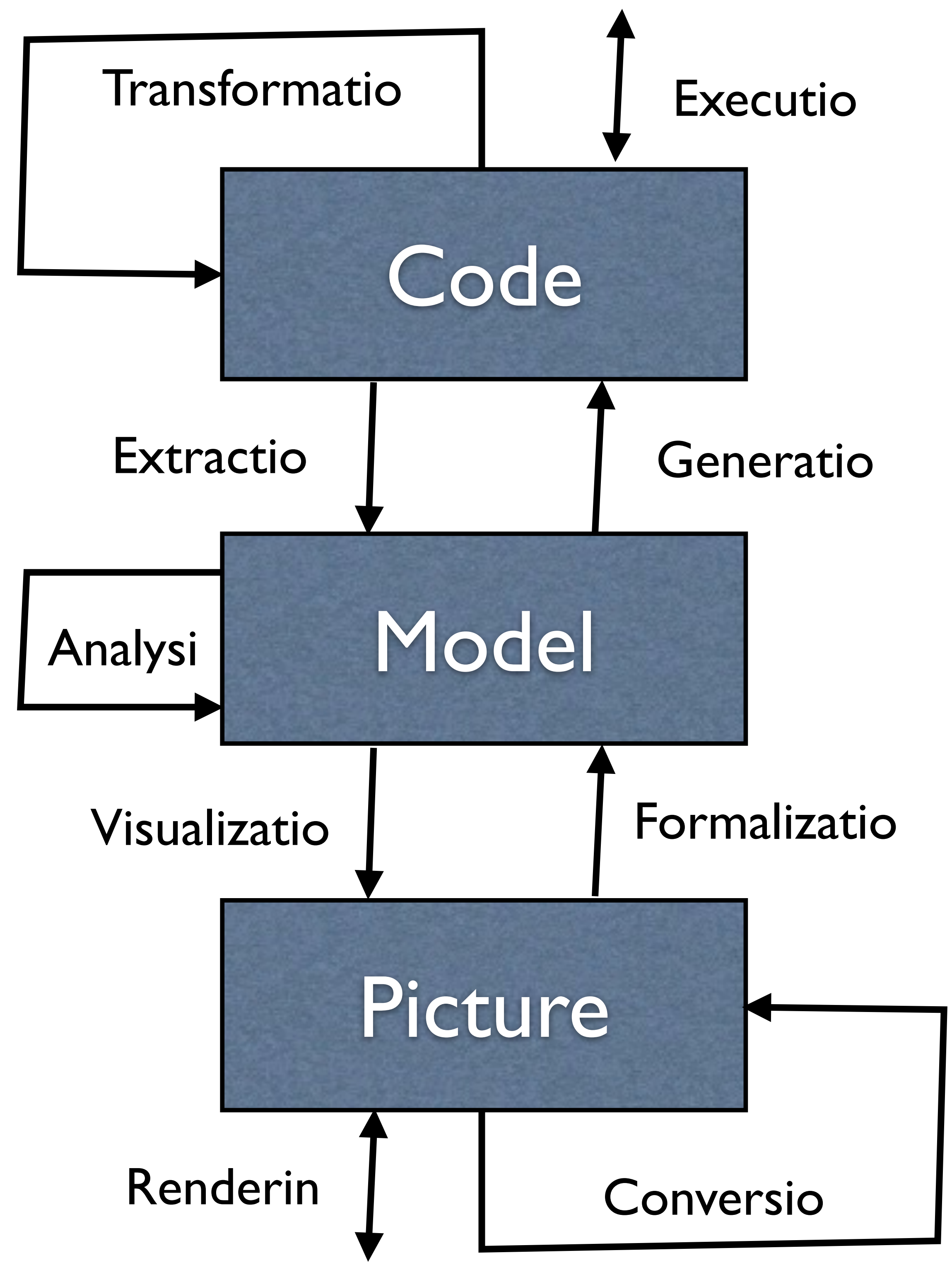


http://www.rascal-mpl.org

Rascal
is
a
language
for
meta
programming

(which we apply
for science,
maintenance and
construction in research
and industry)

“risky” investment
10 year perspective



Conclusion

Conclusion

- Software Complexity Agenda
  - Philosophy
  - Science
  - Maintenance
  - Construction

Conclusion

- Software Complexity Agenda
  - Philosophy
  - Science
  - Maintenance
  - Construction
- Going meta is the key
  - **Tools** enable collaboration
  - Tools manage accidental **complexity**
  - Community is necessary to mitigate cost
  - Education needs to go meta

Conclusion

- Software Complexity Agenda
  - Philosophy
  - Science
  - Maintenance
  - Construction
- Going meta is the key
  - **Tools** enable collaboration
  - Tools manage accidental **complexity**
  - Community is necessary to mitigate cost
  - Education needs to go meta
- Let engineers focus on **value**