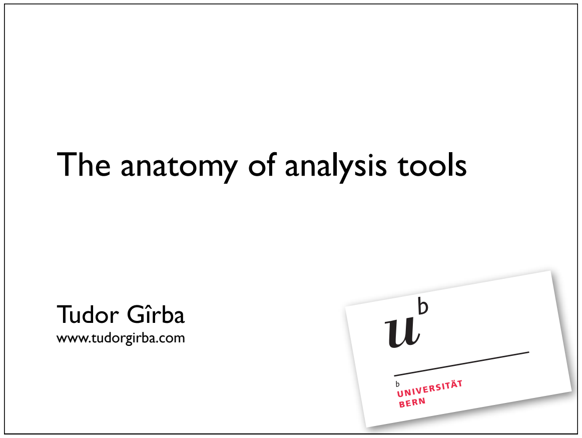
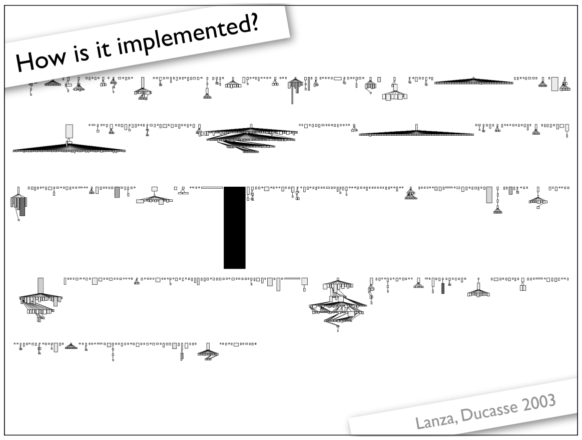


The anatomy of analysis tools

Tudor Gîrba
www.tudorgirba.com



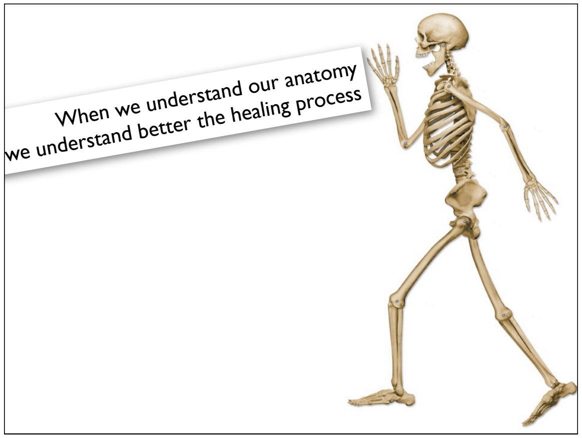
How is it implemented?

Lanza, Ducasse 2003

System complexity of ArgoUML.

How is it implemented?

But, why should you care?

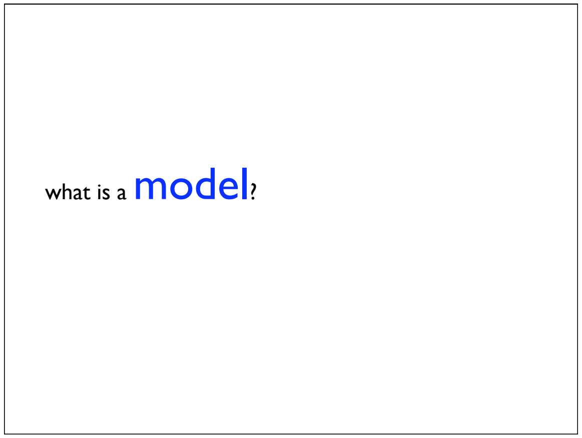


When we understand our anatomy
we understand better the healing process

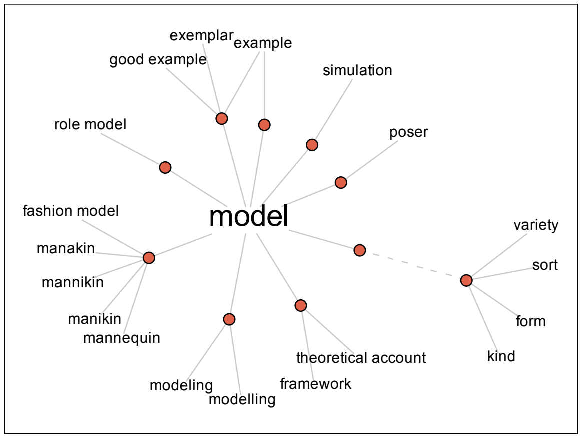
At the orthopedist, you often can see a skeleton in a corner, albeit not a walking one :).
In the same way, if you

Terminology

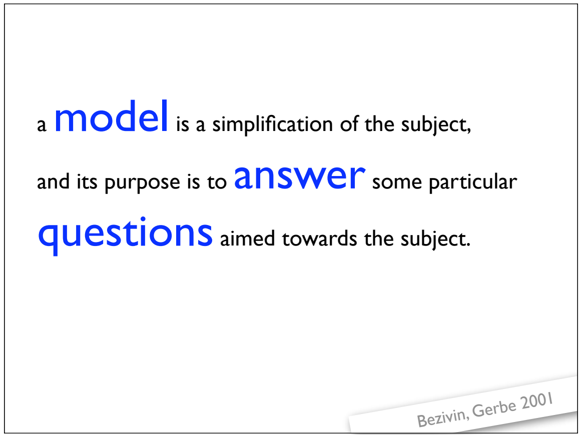
First, let’s agree on the terminology.



what is a model?



The picture is taken from www.visualthesaurus.com and it shows the nouns related to the model noun.

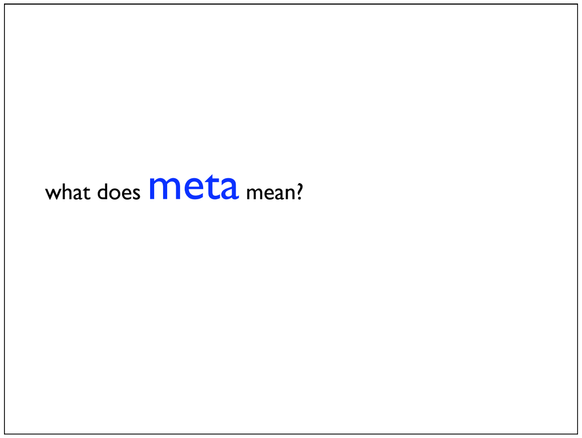


a **model** is a simplification of the subject,

and its purpose is to **answer** some particular **questions** aimed towards the subject.

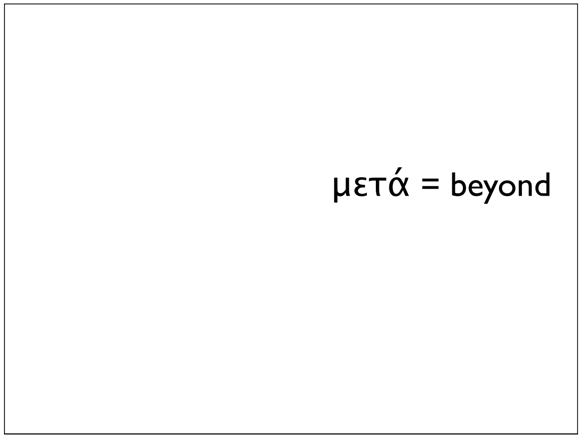
Bezivin, Gerbe 2001

Jean Bezivin and Olivier Gerbe. Towards a precise definition of the OMG/MDA framework. In Proceedings of Automated Software Engineering (ASE 2001), pages 273–282. IEEE Computer Society, 2001.



what does **meta** mean?

http://en.wikipedia.org/wiki/Meta



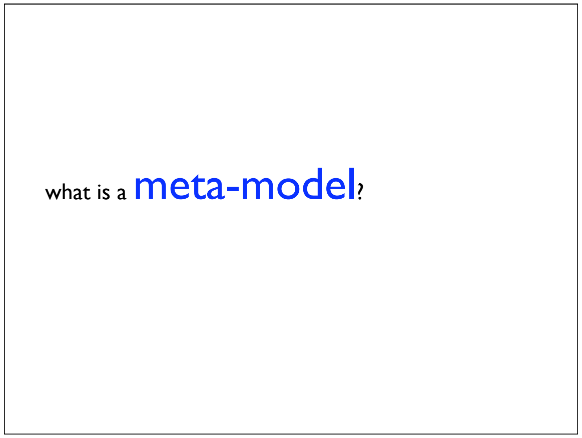
μετά = beyond

Meta comes from Greek and it means “beyond” or “after”.

The use of meta comes from the Metaphysics book of Aristotle.

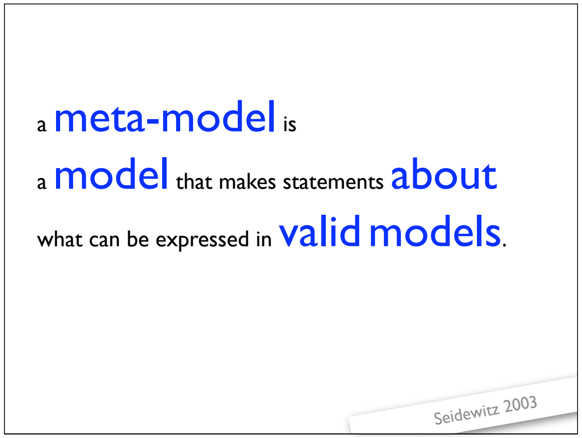
http://en.wikipedia.org/wiki/Metaphysics_(Aristotle)

The picture of Aristotle was painted by Francesco Hayez.
http://en.wikipedia.org/wiki/Francesco_Hayez



what is a meta-model?

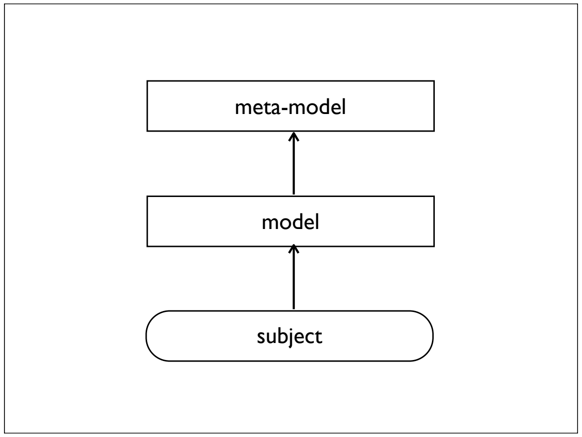
Is a meta-model a model of a model? No.



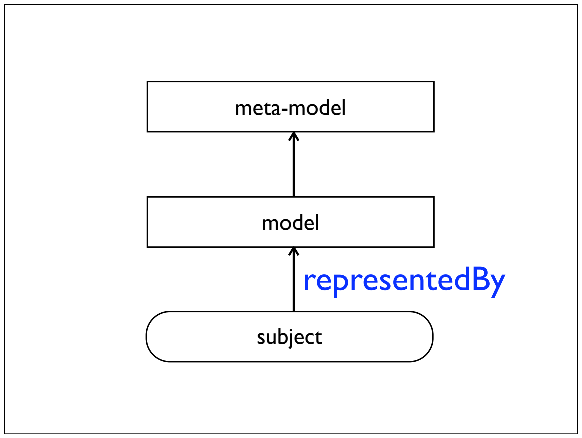
a **meta-model** is
a **model** that makes statements **about**
what can be expressed in **valid models**.

Seidewitz 2003

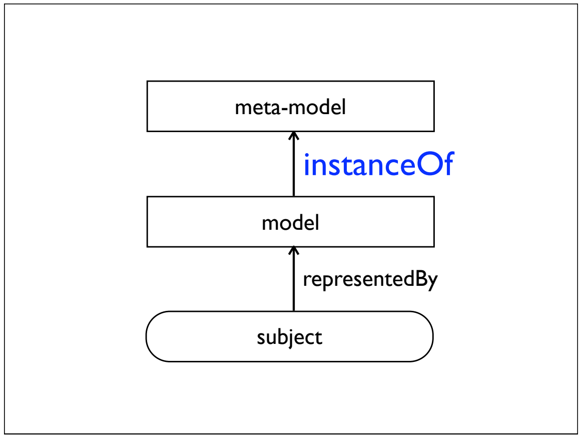
Ed Seidewitz. What models mean. IEEE Software, 20:26–32, September 2003.



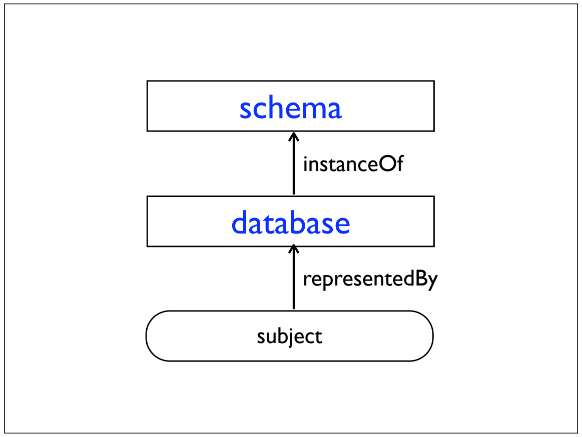
Is the relationship between model and subject the same as the one between meta-model and model? No.



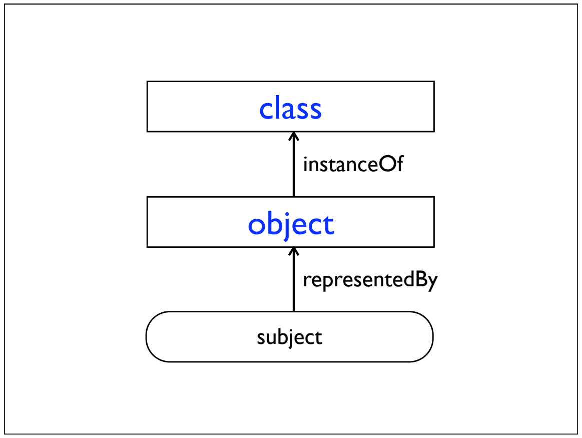
A model represents a subject, and its goal is to answer questions instead of the subject.



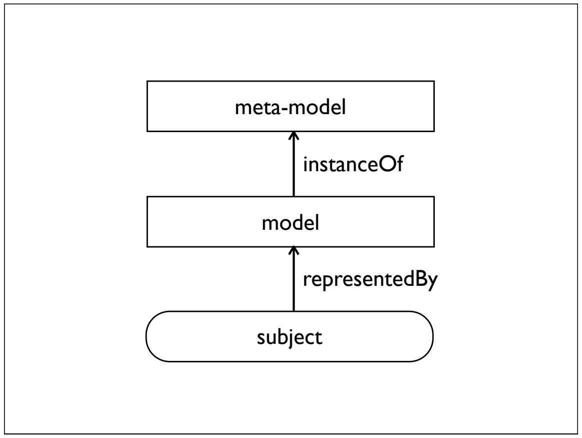
The meta-model describes the model.



When talking about database, the actual database is the model, while the schema is the meta-model.



Similarly, the object in an object-oriented system is a model, and the meta-model is the class.



meta-model

instanceOf

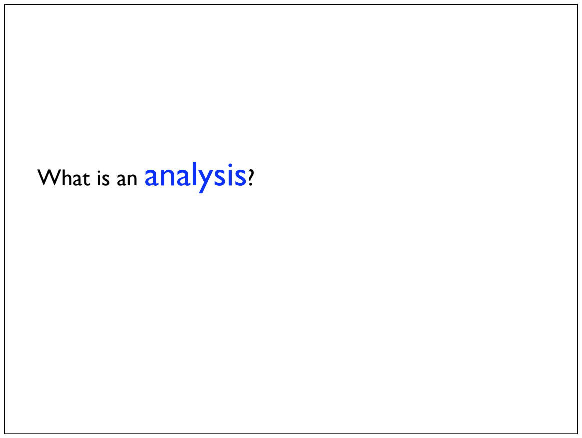
model

representedBy

subject

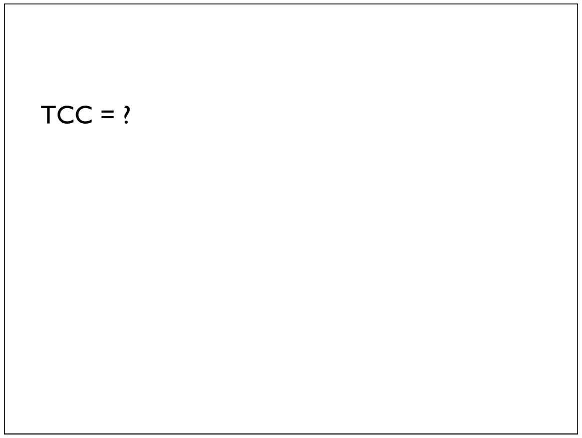
Terminology

Analysis and meta-models



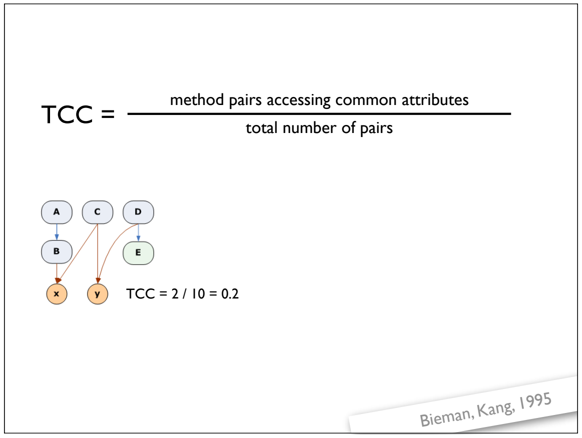
What is an analysis?

Before we go forward, what is an analysis?



TCC = ?

Let’s take an example. TCC. But first, what is TCC? :)

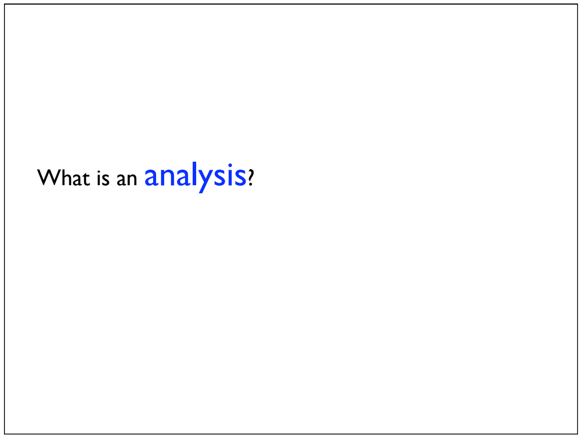


$$TCC = \frac{\text{method pairs accessing common attributes}}{\text{total number of pairs}}$$

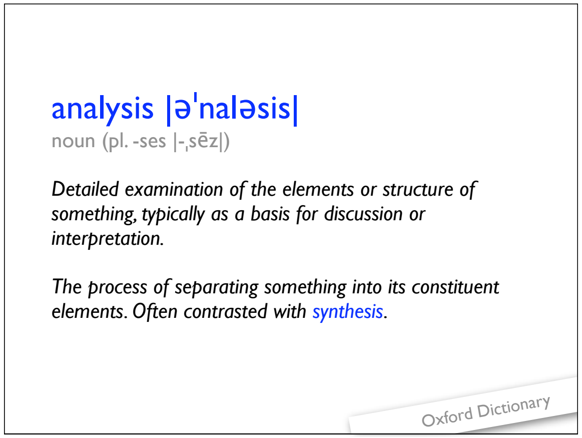
TCC = 2 / 10 = 0.2

Bieman, Kang, 1995

TCC stands for tight class cohesion and it is a metric of cohesion.



What is an analysis?



analysis |əˈnaləsis|

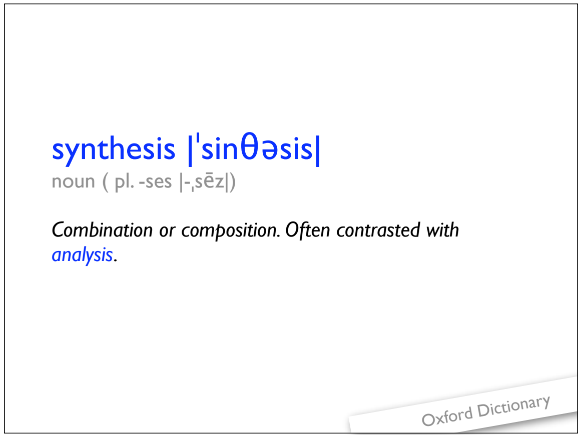
noun (pl. -ses |-ˌsēz|)

Detailed examination of the elements or structure of something, typically as a basis for discussion or interpretation.

The process of separating something into its constituent elements. Often contrasted with synthesis.

Oxford Dictionary

Let’s take a look at the definition

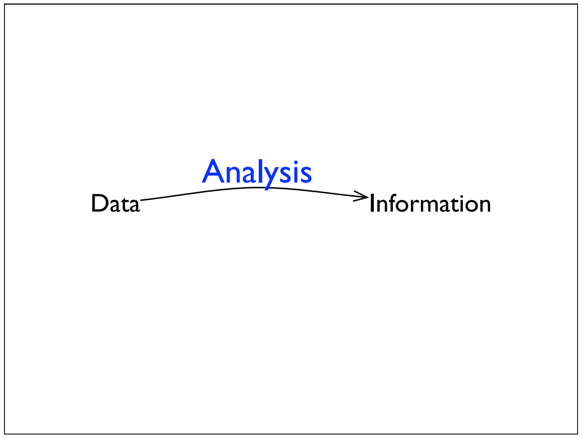


synthesis |ˈsinθəsis|

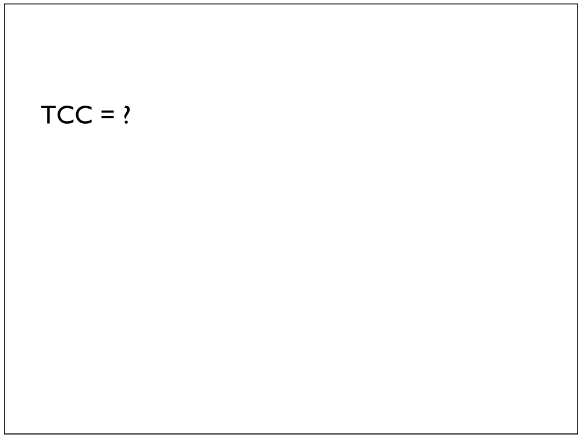
noun (pl. -ses |-ˌsēz|)

Combination or composition. Often contrasted with analysis.

Oxford Dictionary

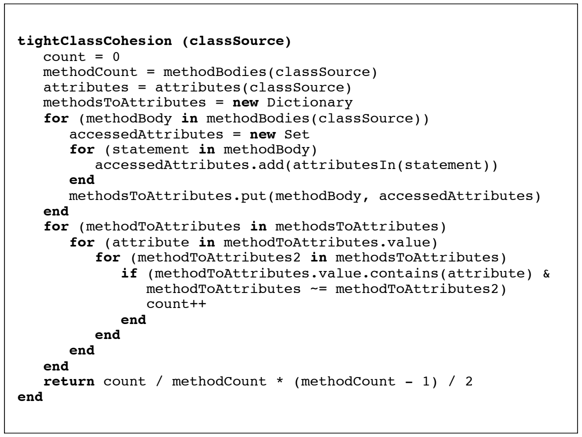


In the scope of this lecture analysis is to be seen as the transformation of data given by pure facts into information that conveys meaning.



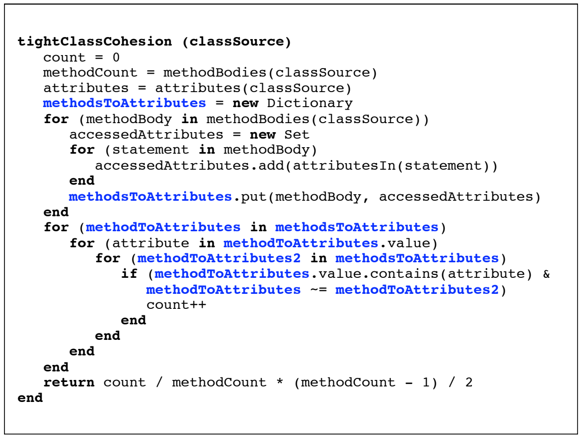
TCC = ?

How can we define TCC?



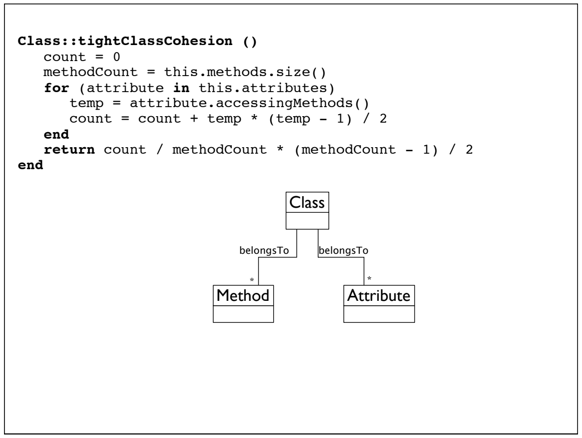
```
tightClassCohesion (classSource)
   count = 0
   methodCount = methodBodies(classSource)
   attributes = attributes(classSource)
   methodsToAttributes = new Dictionary
   for (methodBody in methodBodies(classSource))
      accessedAttributes = new Set
      for (statement in methodBody)
         accessedAttributes.add(attributesIn(statement))
      end
      methodsToAttributes.put(methodBody, accessedAttributes)
   end
   for (methodToAttributes in methodsToAttributes)
      for (attribute in methodToAttributes.value)
         for (methodToAttributes2 in methodsToAttributes)
            if (methodToAttributes.value.contains(attribute) &
               methodToAttributes ~= methodToAttributes2)
               count++
            end
         end
      end
   end
   return count / methodCount * (methodCount - 1) / 2
end
```

Let's take a look at a possible implementation of TCC that takes as input the source code of a class.



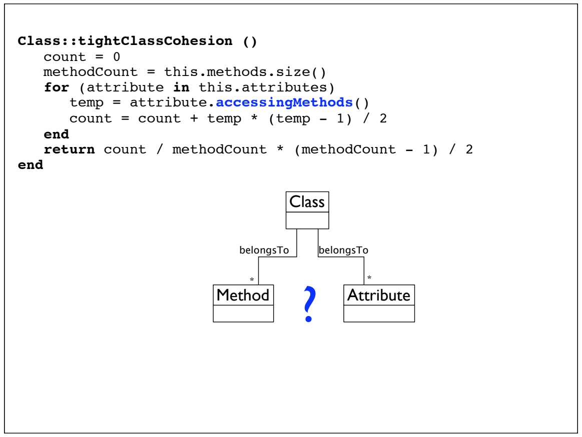
```
tightClassCohesion (classSource)
   count = 0
   methodCount = methodBodies(classSource)
   attributes = attributes(classSource)
   methodsToAttributes = new Dictionary
   for (methodBody in methodBodies(classSource))
      accessedAttributes = new Set
      for (statement in methodBody)
         accessedAttributes.add(attributesIn(statement))
      end
      methodsToAttributes.put(methodBody, accessedAttributes)
   end
   for (methodToAttributes in methodsToAttributes)
      for (attribute in methodToAttributes.value)
         for (methodToAttributes2 in methodsToAttributes)
            if (methodToAttributes.value.contains(attribute) &
               methodToAttributes ~= methodToAttributes2)
               count++
            end
         end
      end
   end
   return count / methodCount * (methodCount - 1) / 2
end
```

The code is difficult to follow because the computation of the metric is intertwined with the construction of some intermediate data structures. For example, the variables highlighted represent relationships that are needed for the computation.



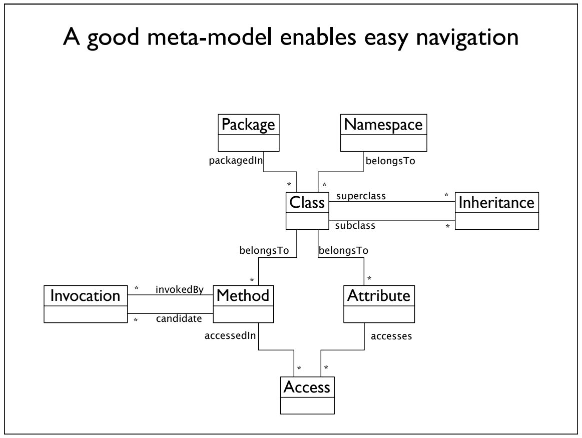
```
Class::tightClassCohesion ()
   count = 0
   methodCount = this.methods.size()
   for (attribute in this.attributes)
      temp = attribute.accessingMethods()
      count = count + temp * (temp - 1) / 2
   end
   return count / methodCount * (methodCount - 1) / 2
end
```

This is another way to implement the metric. The code is much simpler to read because it is based on a meta-model that is more suited for the computation.



```
Class::tightClassCohesion ()
   count = 0
   methodCount = this.methods.size()
   for (attribute in this.attributes)
      temp = attribute.accessingMethods()
      count = count + temp * (temp - 1) / 2
   end
   return count / methodCount * (methodCount - 1) / 2
end
```

Still, where does accessingMethods come from?



A good meta-model enables easy navigation

The picture shows FAMIX 2.1. Note how there are no arrows, which means that the relationships can be navigated in both directions. For example, an attribute can know about the accessing methods by going through all accesses that point to him and collecting the methods that initiate those accesses.

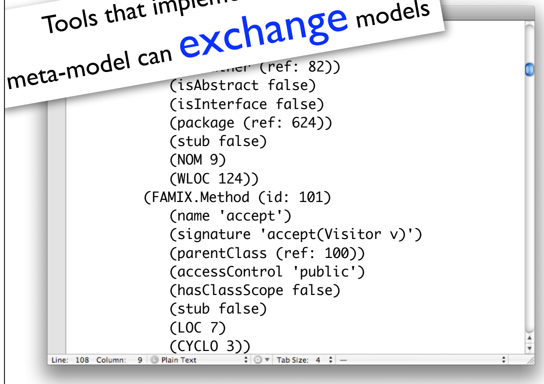
A **meta-model** offers

a **language**

A **common** meta-model offers

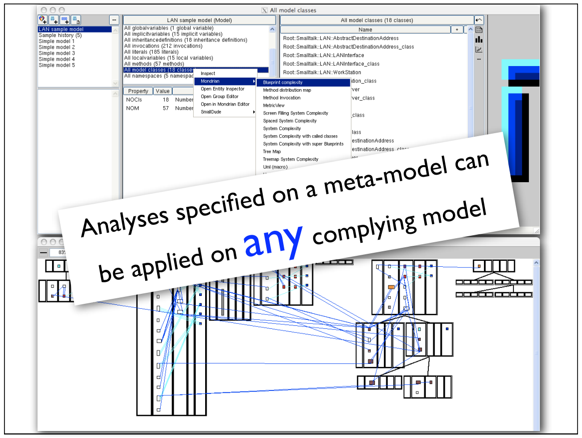
a **common** language

Tools that implement a common meta-model can **exchange** models



```
ner (ref: 82))
       (isAbstract false)
       (isInterface false)
       (package (ref: 624))
       (stub false)
       (NOM 9)
       (WLOC 124))
    (FAMIX.Method (id: 101)
       (name 'accept')
       (signature 'accept(Visitor v)')
       (parentClass (ref: 100))
       (accessControl 'public')
       (hasClassScope false)
       (stub false)
       (LOC 7)
       (CYCLO 3))
```

Two tools that share the same meta-model can exchange models that comply with these meta-models by using an exchange format. In this example, we see an MSE file format.



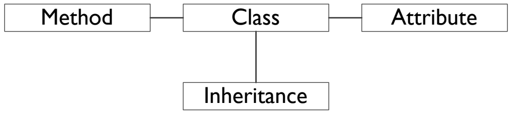
LAN sample model
Sample history (5)
Simple model 1
Simple model 2
Simple model 3
Simple model 4
Simple model 5

All globalvariables (1 global variable)
All implicitvariables (15 implicit variables)
All inheritancedefinitions (18 inheritance definitions)
All invocations (212 invocations)
All literals (185 literals)
All localvariables (15 local variables)
All methods (57 methods)
All model classes (18 classes)
All namespaces (5 namespac

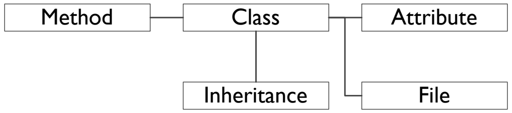
Property	Value	
NOCls	18	Number
NOM	57	Number

Root::Smalltalk::LAN::AbstractDestinationAddress
Root::Smalltalk::LAN::AbstractDestinationAddress_class
Root::Smalltalk::LAN::LANInterface
Root::Smalltalk::LAN::LANInterface_class
Root::Smalltalk::LAN::WorkStation

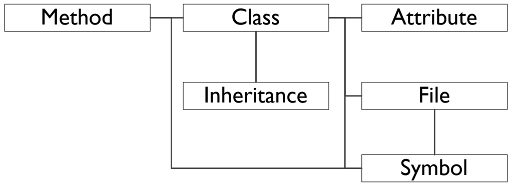
Analyses specified on a meta-model can be applied on any complying model



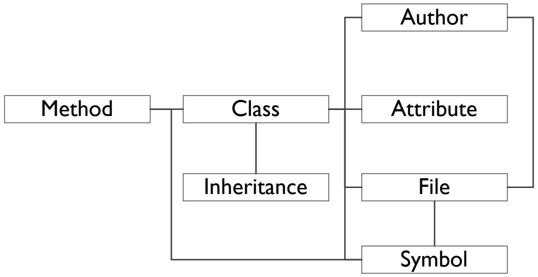
Method
Class
Attribute
Inheritance



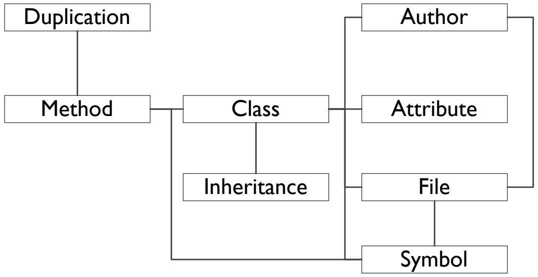
Method
Class
Attribute
Inheritance
File



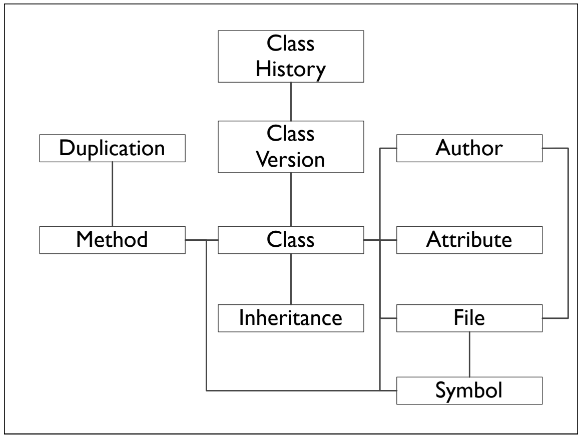
Method
Class
Attribute
Inheritance
File
Symbol

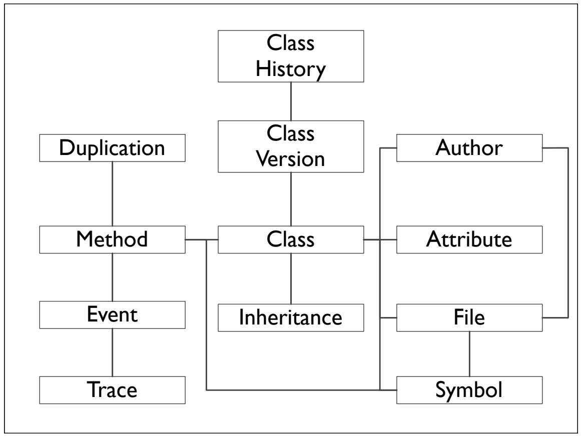


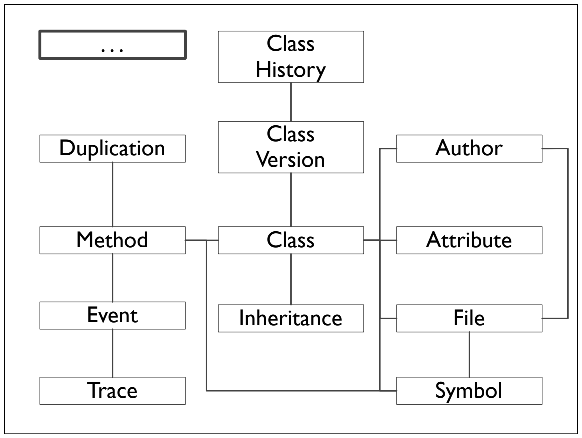
Author
Method
Class
Attribute
Inheritance
File
Symbol



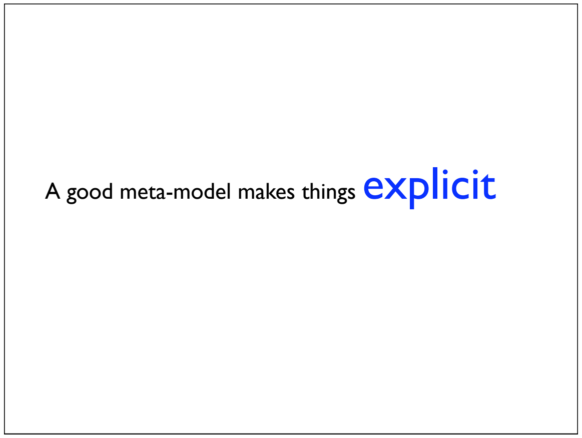
Duplication
Author
Method
Class
Attribute
Inheritance
File
Symbol





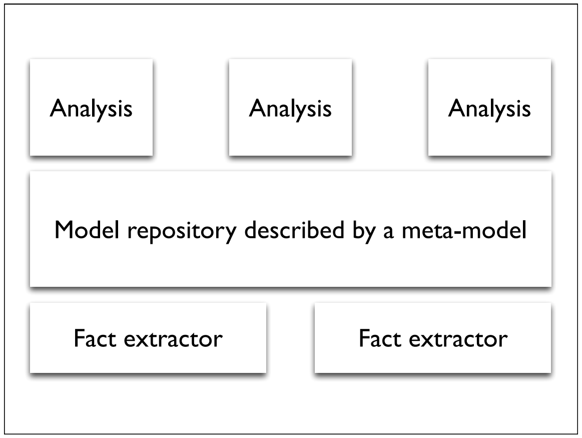


We can have many entities in a meta-model, depending on what we are interested in. Also, we can have many meta-models, depending on the point of view.



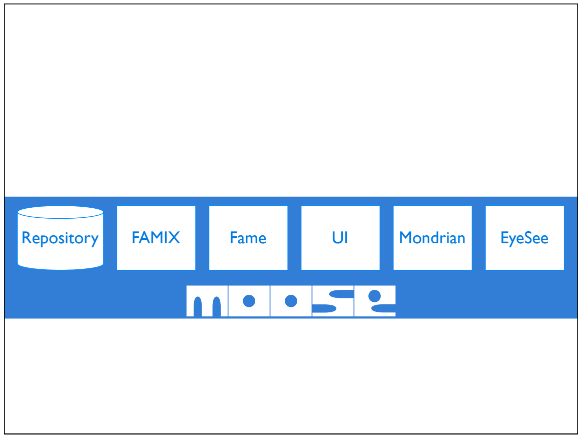
A good meta-model makes things explicit

Terminology
Analysis and meta-models
Analysis tools



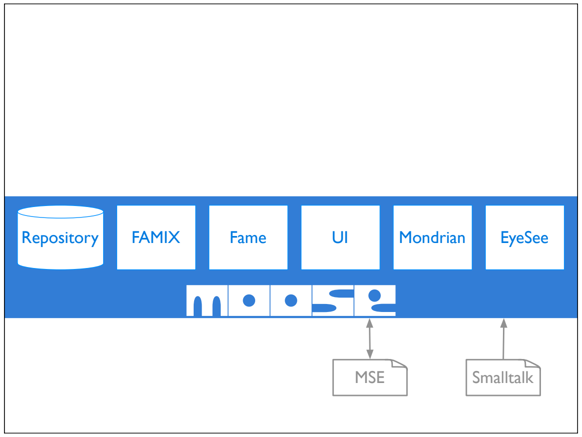
Analysis
Analysis
Analysis
Model repository described by a meta-model
Fact extractor
Fact extractor

The basic architecture of an analysis tool. Fact extractors extract data from the subject systems. This data is then stored in models that are described by meta-models. Analyses are specified based on the meta-model.



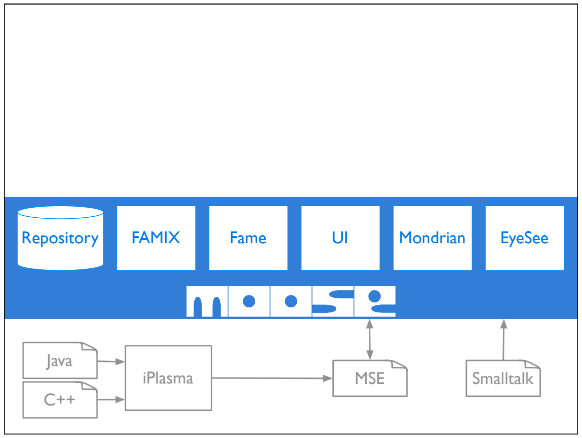
Repository
FAMIX
Fame
UI
Mondrian
EyeSee

Let's take a look at the architecture of Moose (http://moose.unibe.ch). At the core we have a Repository of models that are described by the FAMIX family of meta-models. Fame is an implementation of the meta-meta-model that describes FAMIX. UI, Mondrian and EyeSee are generic tools that work with any meta-models.



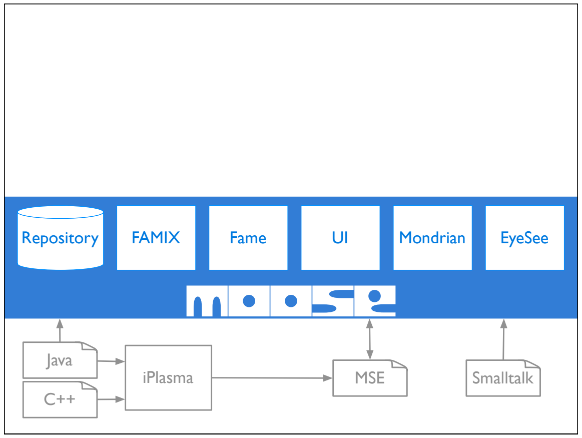
Repository
FAMIX
Fame
UI
Mondrian
EyeSee
MSE
Smalltalk

Data is imported either directly from Smalltalk, or through the MSE exchange format.



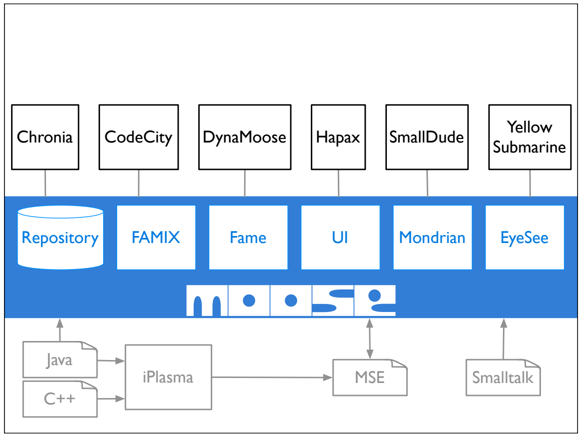
Repository
FAMIX
Fame
UI
Mondrian
EyeSee
Java
C++
iPlasma
MSE
Smalltalk

iPlasma is one external tool that can parse Java and C++ systems and exports models complying to FAMIX in an MSE format. These MSE files can then be imported into Moose.



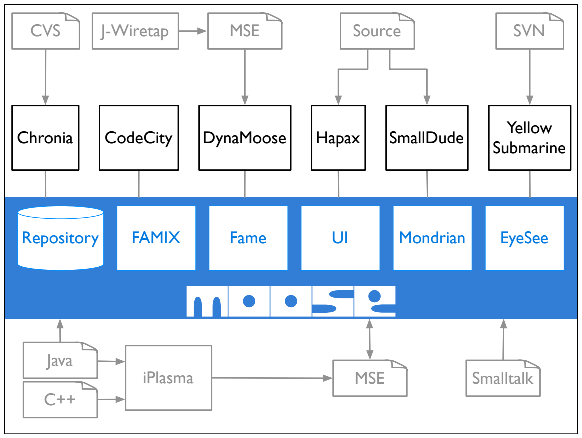
Repository
FAMIX
Fame
UI
Mondrian
EyeSee
Java
C++
iPlasma
MSE
Smalltalk

Recently, support was added for Java systems to be parsed directly.



Chronia
CodeCity
DynaMoose
Hapax
SmallDude
Yellow Submarine
Repository
FAMIX
Fame
UI
Mondrian
EyeSee
Java
C++
iPlasma
MSE
Smalltalk

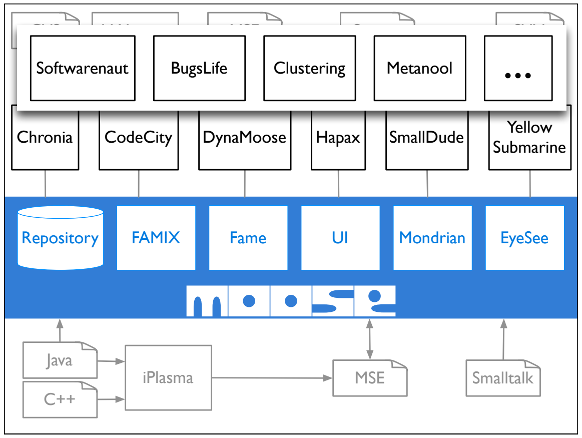
On top, several analyses tools are built.



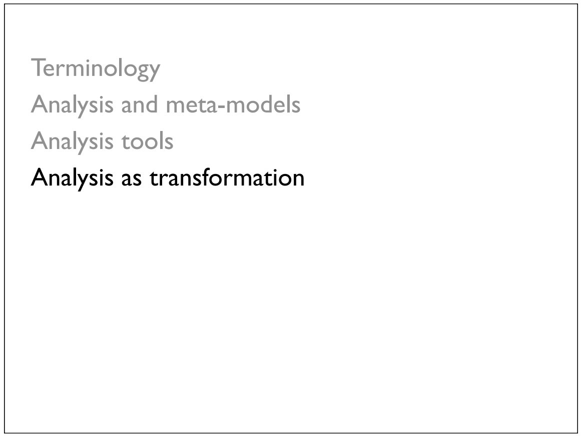
CVS
J-Wiretap
MSE
Source
SVN
Chronia
CodeCity
DynaMoose
Hapax
SmallDude
Yellow Submarine
Repository
FAMIX
Fame
UI
Mondrian
EyeSee
Java
C++
iPlasma
MSE
Smalltalk

These tools, at their turn, can also import data from other sources. Furthermore, in the case of Moose these tools can also extend FAMIX with new kinds of entities due to the Fame engine.

Adrian Kuhn and Toon Verwaest, “FAME, A Polyglot Library for Metamodeling at Runtime,” Workshop on Models at Runtime, 2008, pp. n10. http://www.iam.unibe.ch/~scg/cgi-bin/scgbib.cgi/abstract=yes?Kuhn08c



And there are even more such tools.

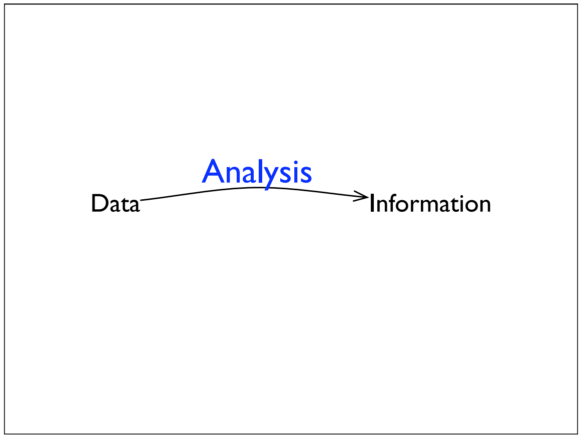


Terminology

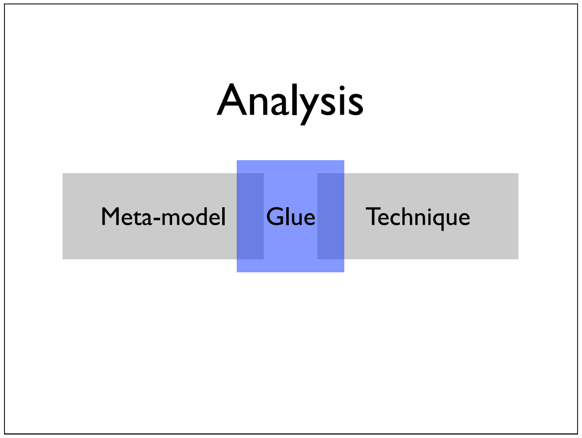
Analysis and meta-models

Analysis tools

Analysis as transformation

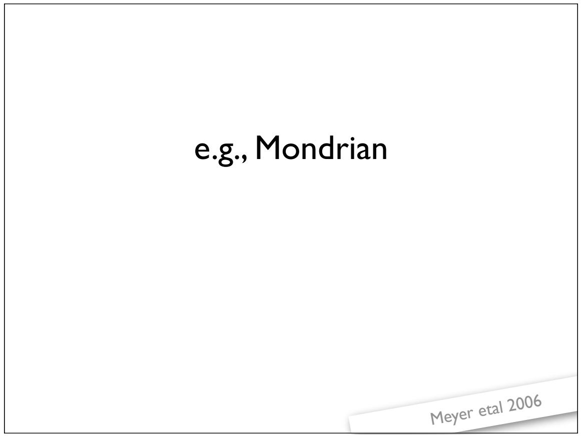


In the scope of this lecture analysis is to be seen as the transformation of data given by pure facts into information that conveys meaning.



Analysis

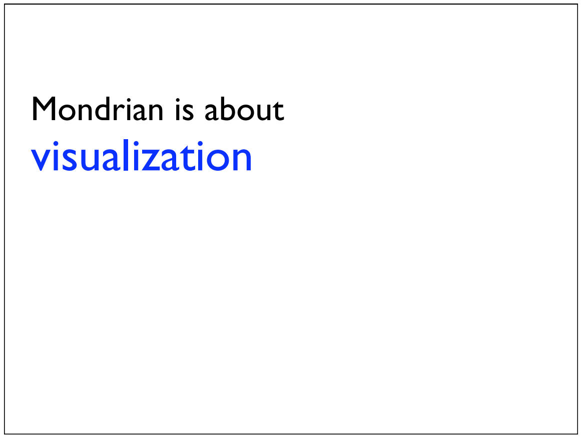
Analysis = meta-model + technique + glue



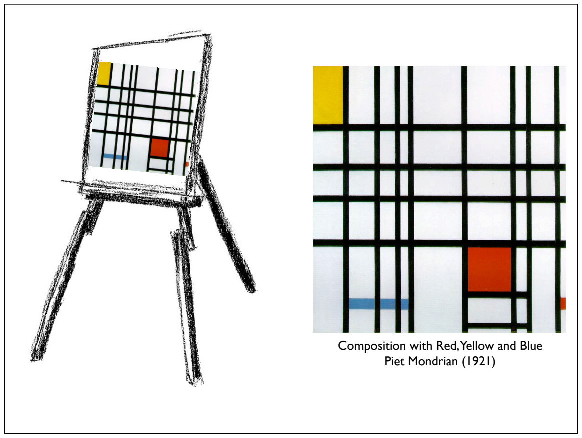
e.g., Mondrian

Meyer etal 2006

Michael Meyer, Tudor Gîrba and Mircea Lungu, "Mondrian: An Agile Visualization Framework," ACM Symposium on Software Visualization (SoftVis 2006), ACM Press, New York, NY, USA, 2006, pp. 135—144.
Michael Meyer and Tudor Gîrba, "Mondrian: Scripting Visualizations," European Smalltalk User Group 2006 Technology Innovation Awards, August 2006, It received the 2nd prize.
http://moose.unibe.ch/mondrian

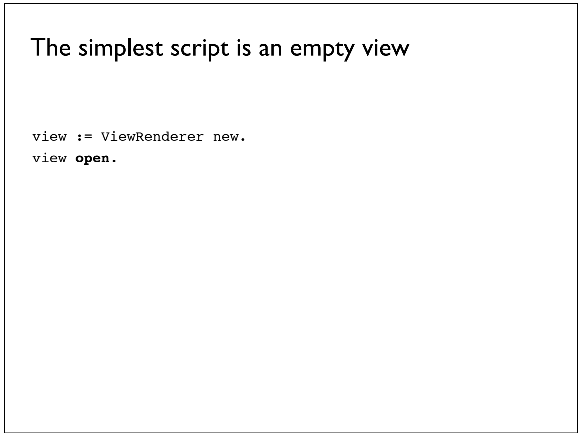


Mondrian is about
visualization



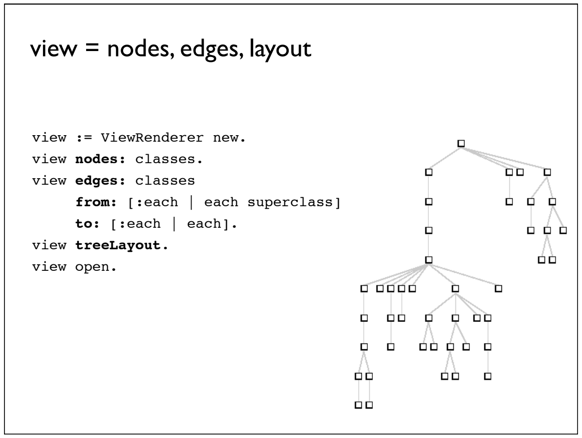
Composition with Red, Yellow and Blue
Piet Mondrian (1921)

Mondrian was a painter that saw the world as boxes and lines. Similarly, the visualization engine takes the point of view.



The simplest script is an empty view

```
view := ViewRenderer new.
view open.
```

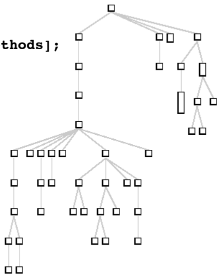


view = nodes, edges, layout

```
view := ViewRenderer new.
view nodes: classes.
view edges: classes
     from: [:each | each superclass]
     to: [:each | each].
view treeLayout.
view open.
```

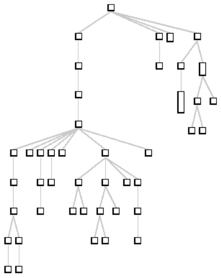
Shapes are responsible for drawing

```
view := ViewRenderer new.
view newShape rectangle;
     height: [:each | each numberOfMethods];
     withBorder.
view nodes: classes.
view edges: classes
     from: [:each | each superclass]
     to: [:each | each].
view treeLayout.
view open.
```



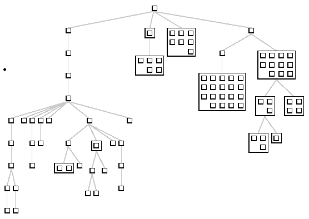
Blocks can be replaced by symbols

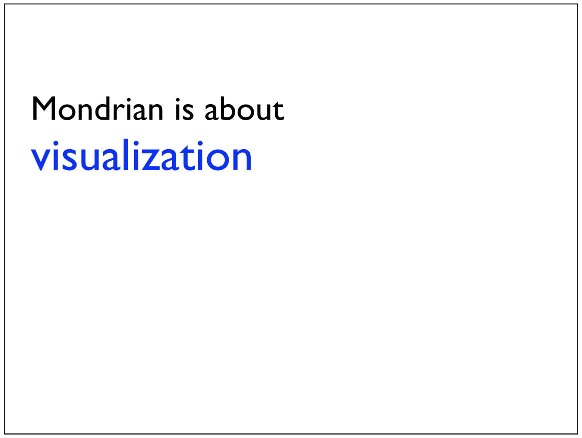
```
view := ViewRenderer new.
view newShape rectangle;
     height: #numberOfMethods;
     withBorder.
view nodes: classes.
view edgesFrom: #superclass.
view treeLayout.
view open.
```



Nesting is done through blocks

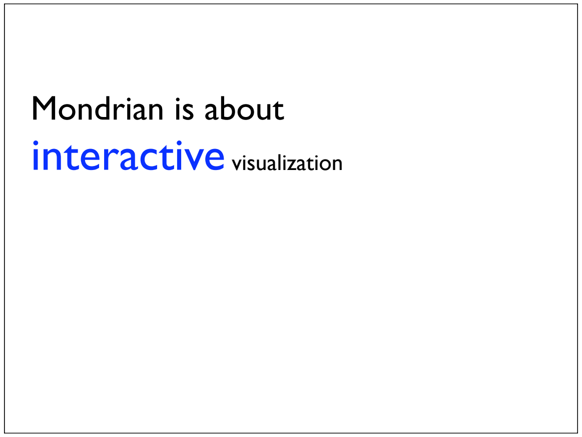
```
view := ViewRenderer new.
view newShape rectangle; withBorder.
view nodes: classes forEach: [:each |
  view nodes: each methods.
  view gridLayout
].
view edgesFrom: #superclass.
view treeLayout.
view open.
```





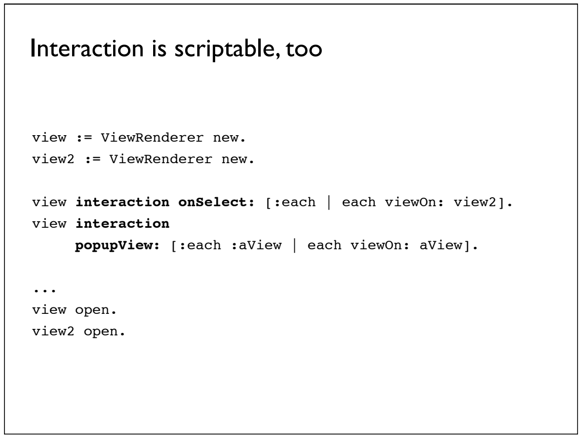
Mondrian is about

visualization



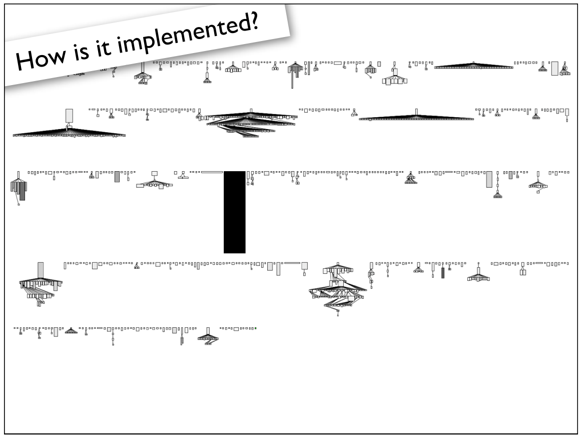
Mondrian is about

interactive visualization



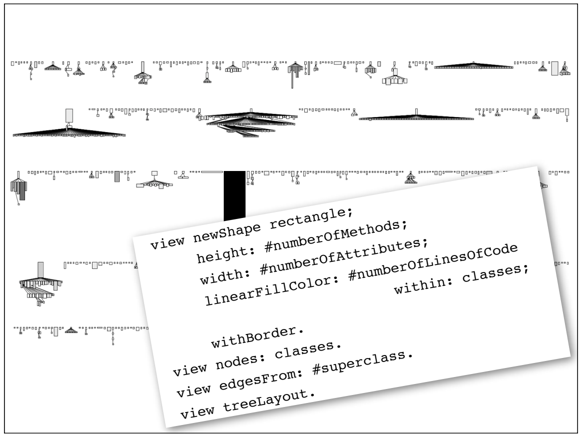
Interaction is scriptable, too

```
view := ViewRenderer new.
view2 := ViewRenderer new.

view interaction onSelect: [:each | each viewOn: view2].
view interaction
     popupView: [:each :aView | each viewOn: aView].

...
view open.
view2 open.
```



How is it implemented?

So, how is this implemented?

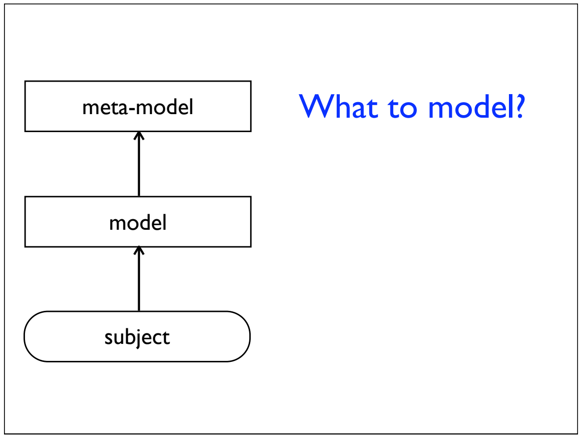


```
view newShape rectangle;
      height: #numberOfMethods;
      width: #numberOfAttributes;
      linearFillColor: #numberOfLinesOfCode
                       within: classes;

      withBorder.
view nodes: classes.
view edgesFrom: #superclass.
view treeLayout.
```

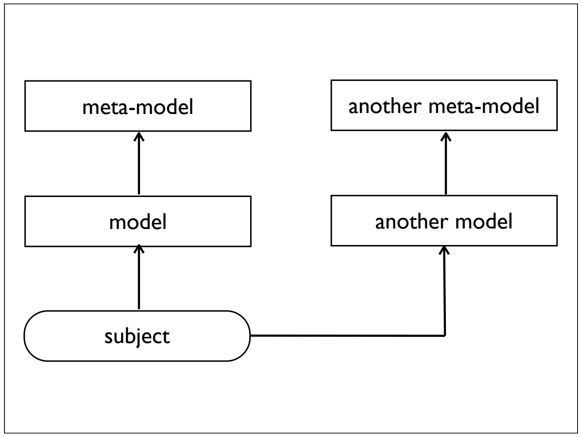
Analysis = meta-model + technique + glue.
In this case, the visualization was specified using a generic graph technique. The nodes are drawn according to the metrics that are defined on top of the basic meta-model of the code structure and that are directly accessible as properties. Furthermore, edges are obtained by navigating from each class to the superclass, again according to the meta-model.

Although the visualization is not trivial, the glue code is small.

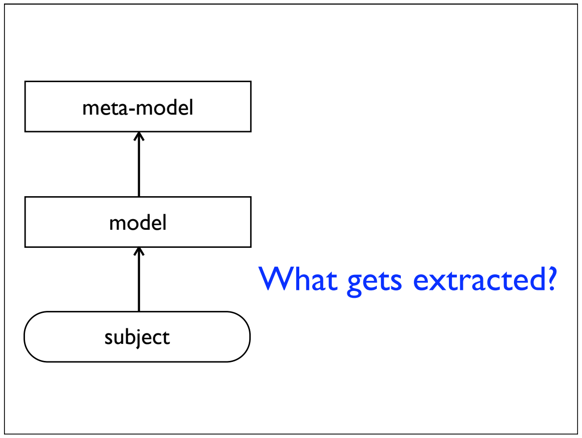
Terminology
Analysis and meta-models
Analysis tools
Analysis as transformation
Issues



What to model?

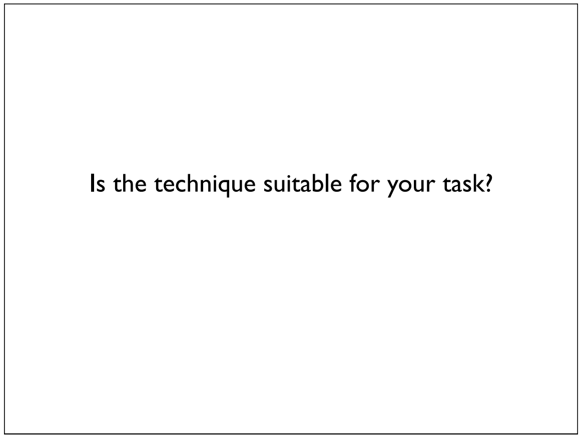


Any given subject can be modeled in several ways according to the point of interest. For example, some meta-models will make explicit as many things as possible, while others could favor memory space and keep the explicitness to a minimum.



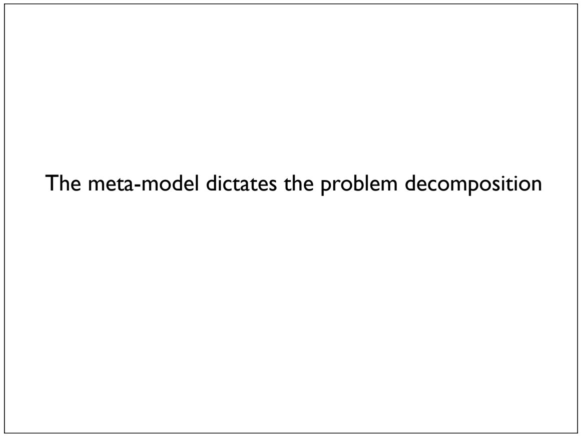
What gets extracted?

While the meta-model specifies what kind of information can get in the model, there is still the question of how much information from the actual system did get in the model. For example, when parsing a system, did the parser resolve all invocations, or did it leave out all the invocations to the library methods?

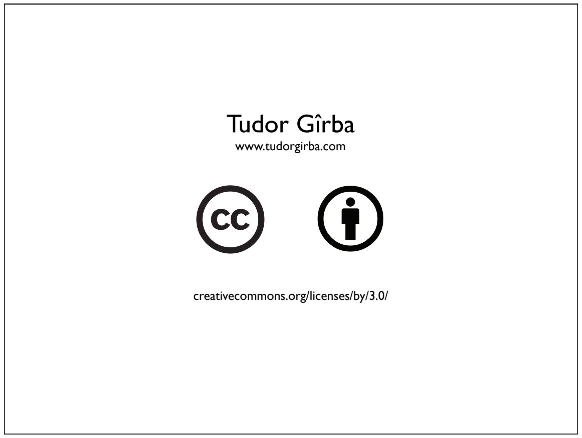


Is the technique suitable for your task?

When you have a hammer, everything looks like a nail. When the glue code can get long and ugly also because the technique is not appropriate for what you want to achieve.



The meta-model dictates the problem decomposition



Tudor Gîrba
www.tudorgirba.com