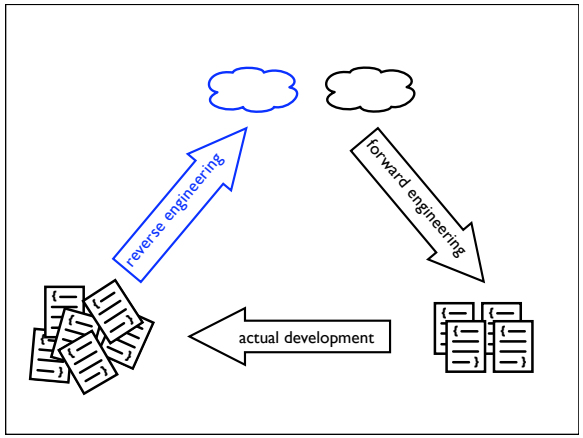
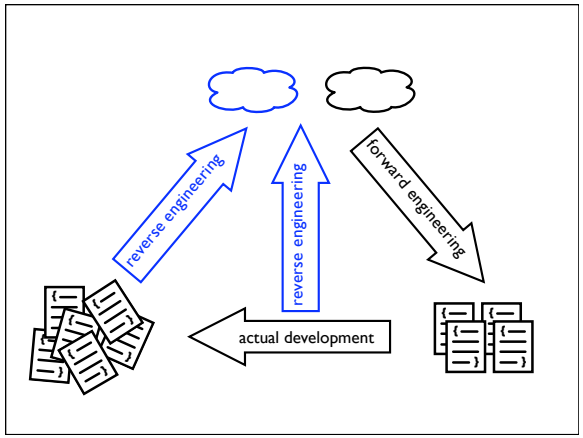
Software understanding
in the large

Tudor Gîrba
www.tudorgirba.com

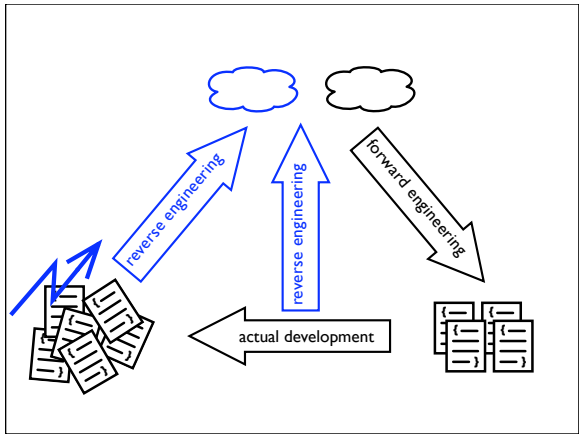
Dr. House provides good examples of reasoning about unknown systems (or patients :)). The system is measured statically, it is monitored permanently, there are interviews with people involved, facts are always questioned and everything is correlated.



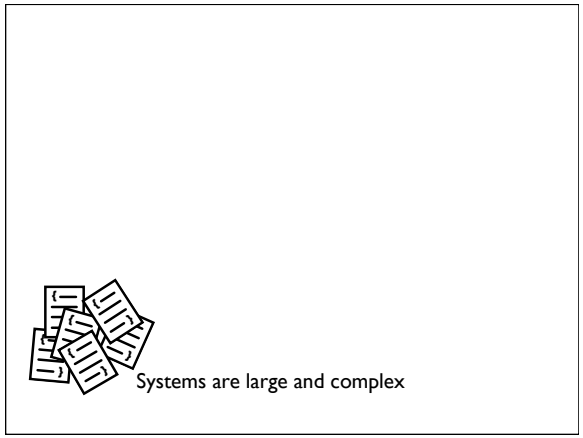
Reverse engineering is needed to re-synchronize the original idea with the reality of the implementation.



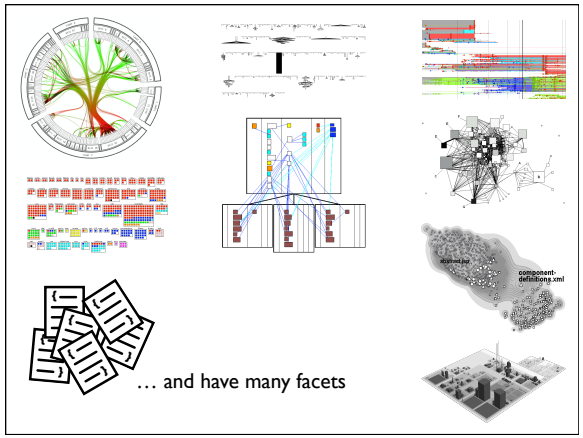
Not only the static structure is useful for reverse engineering, but also the history of the system.



Furthermore, running the system and analyzing its runtime behavior can also reveal useful information.



… and they simply contain too many details



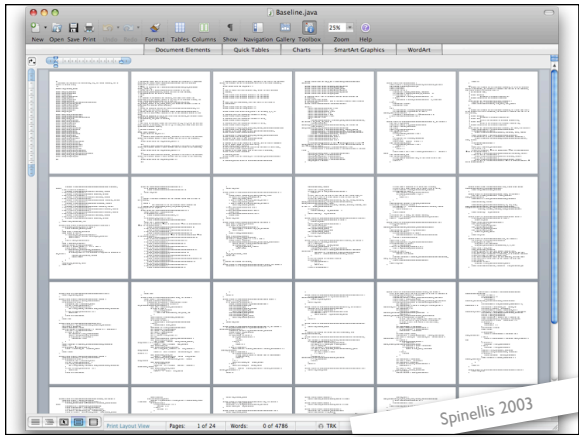
Software systems are complex and they have many facets. A plethora of tools exist to show them.

Simple tools can get you far

Still, tools need to be fancy to be useful.

```
for i in $( ls ); do
   echo `wc -l $i` >> temp
done
sort -nr temp | head -10
```

A simple script for computing the number of the lines of code.



Diomidis Spinellis, Code Reading The Open Source Perspective, Addison-Wesley, 2003.
One simple tool is to use an editor (in this case Word) to display an overview of the source code. One particular issue revealed by this simple approach is the indentation level.

Jorge Ressia created the following script to put all java files into one file that can be open with Word:

```
for eachFile in `find -f $1 | grep -E *java$`
do
   cat $eachFile >> concatFile.java
done
```

Indentation correlates with complexity.

Hindle etal 2008

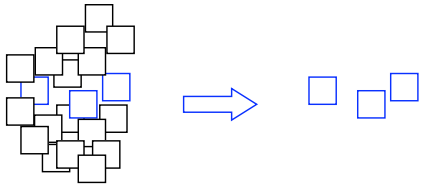
Hindle, A. Godfrey, M.W. Holt, R.C. Reading Beside the Lines: Indentation as a Proxy for Complexity Metric, IEEE International Conference on Program Comprehension, 2008 (ICPC 2008), 133-142.

A recent study has shown that there exist a significant correlation between the indentation level and the complexity of a piece of code. As such, we can identify complex parts without needing to compute the complexity metric.

Simple tools can get you far

Queries help reduce the analysis space

Queries reduce the analysis space



Queries take an input and produce an output that is typically significantly reduced in size.

Queries can be expressed using various languages or paradigms. For example, SQL is an imperative language, while XSLT is a logical language.

Intensional Views ensure rules

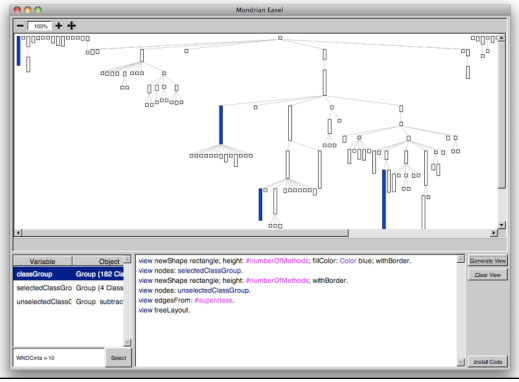
```
acceptsClassOfType(?VisitorClass,?VisitedClass) if
  methodWithNameInClass(?Method,?Selector,?VisitorClass),
  ['accept*' match:?Selector asString],
  argumentOfMethod(?Argument,?Method),
  ['*',(?VisitedClass name asString),
   '*' match:?Argument asString]
```

Mens etal 2006

Kim Mens, Andy Kellens, Frédéric Pluquet and Roel Wuyts, “Co-evolving Code and Design with Intensional Views — A Case Study,” Journal of Computer Languages, Systems and Structures, vol. 32, no. 2, 2006, pp. 140—156.

Intensional Views are logic rules that ensure rules. For example, this rule ensures that the argument of an accept method is named after the class it should be accepted.

Visual queries put results in perspective

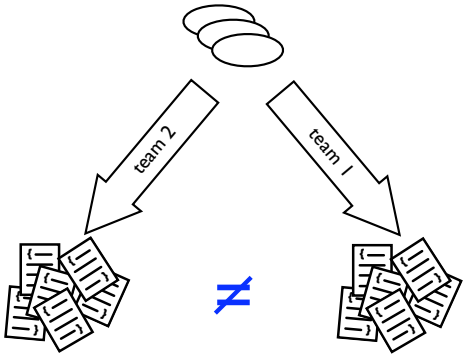


Queries indeed reduce the analysis space, but it is exactly this reduction that makes interpreting the results difficult to interpret. Visualizing the results superimposed on an overview of the system provides a context and eases the interpretation.

Simple tools can get you far

Queries help reduce the analysis space

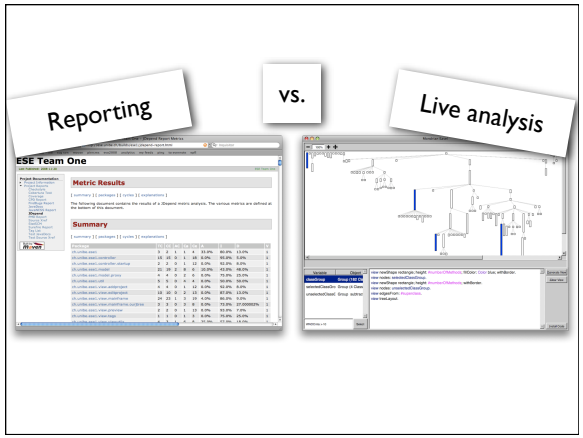
Every system is special



Give the same set of use cases to two different teams and you will get different sets of mess.

Every system is special.

Every system is different. Thus, treating them the same is bound to produce too generic results.



Reporting tools can be made part of a continuous integration process. However, they are (or at least are perceived as being) rigid, and they tend to treat each system in the same way, by employing the same reporting rules. On the other side of the spectrum, a tool that allows live analyses enables the reverse engineer to customize the analysis to match the system at hand.

A reverse engineer must consciously make the choice for which tool to use based on the situation. Even if a reporting tool is used, the queries must match the system at hand.

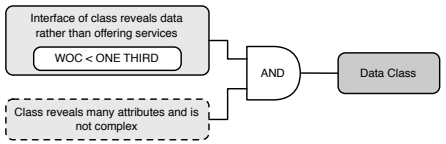
Simple tools can get you far

Queries help reduce the analysis space

Every system is special

Every technology is special

Data Classes are dumb data holders



Lanza, Marinescu 2006

Data classes are signs of poor distribution of responsibilities in object-oriented systems.

Use a Transfer Object to encapsulate the business data. A single method call is used to send and retrieve the Transfer Object.

When the client requests the enterprise bean for the business data, the enterprise bean can construct the Transfer Object, populate it with its attribute values, and pass it by value to the client.

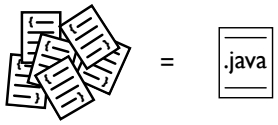
java.sun.com

However, one J2EE pattern promotes creating pure data classes to package the data that is to be transfered over the network. The detection of data classes needs to filter these classes out, as they are obviously not design flaws.

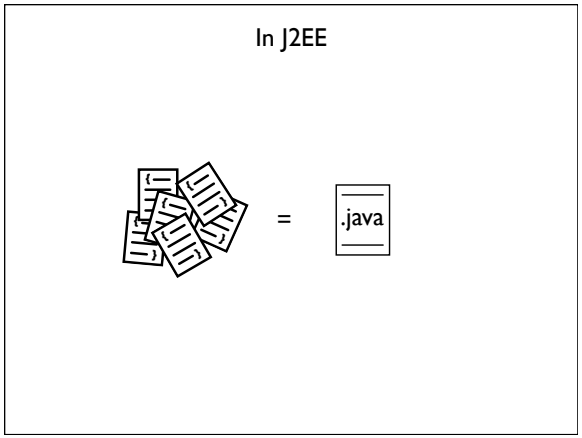
Every technology is special.

Let's see what other variation points are there in a technology.

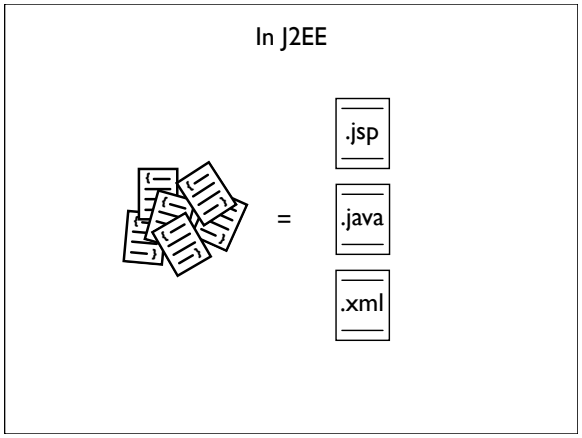
In Java



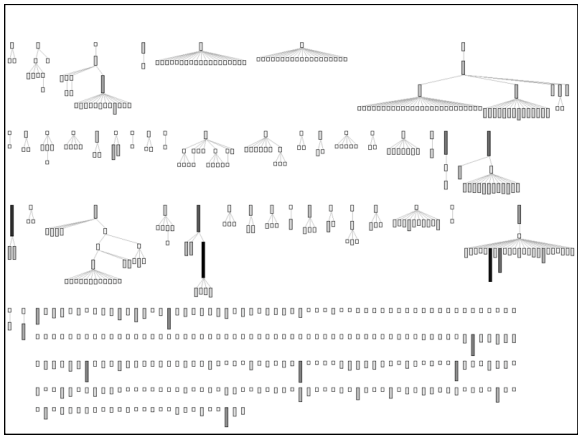
In Java, sources are to be found in the .java files.



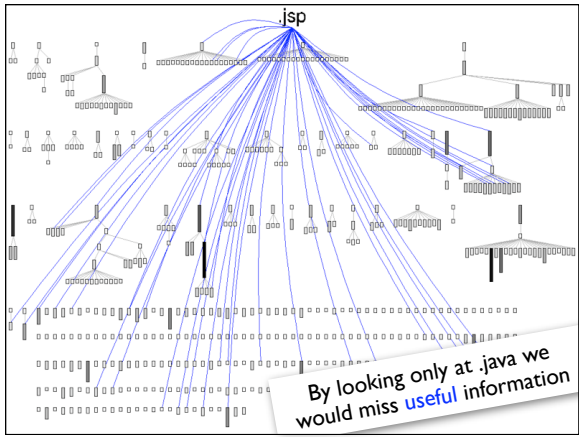
The Java 2 Enterprise Edition is based on Java. Based on Java, but more than just Java.



A significant amount of source code can be found in form of Java Server Pages (JSP) and XML descriptors.

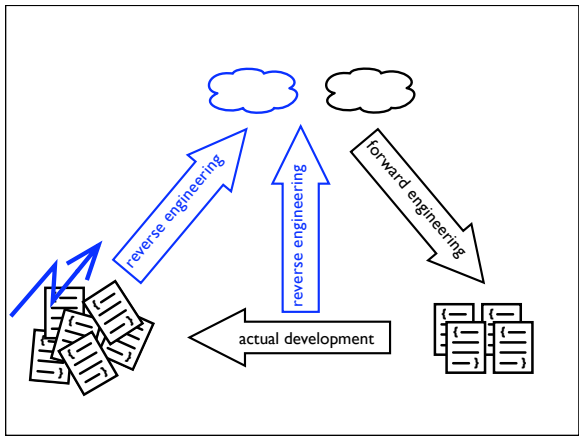


Thus, looking only at the Java code found in the .java files reveals only a part of the story.

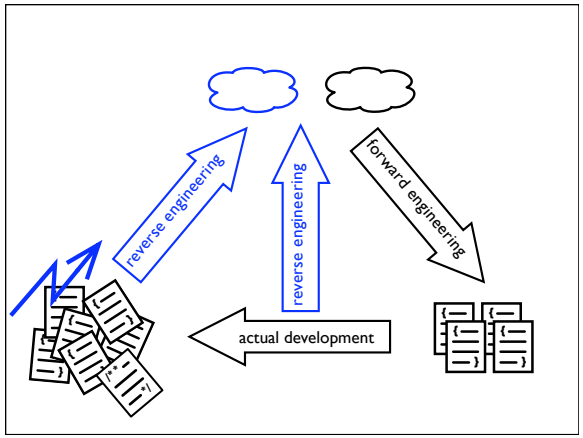


This example shows with blue the calls from the JSP code to the Java classes.

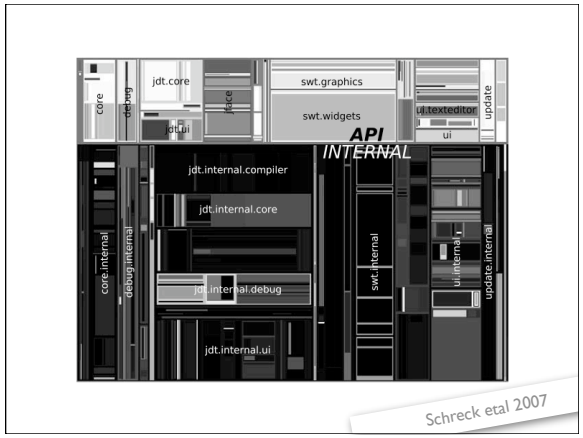
Simple tools can get you far
Queries help reduce the analysis space
Every system is special
Every technology is special
Software systems are more than code



Static, dynamic, history … systems written in different technologies ... What else is there to be taken into account?

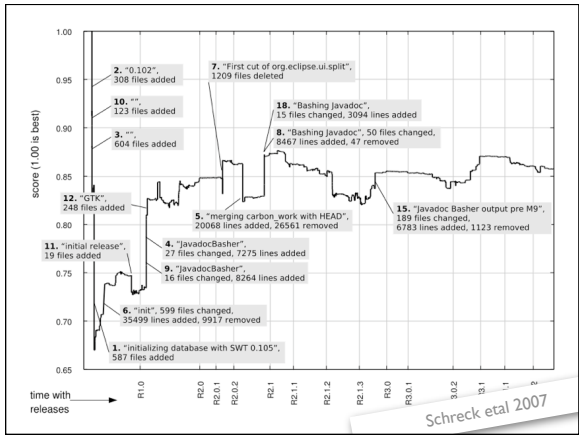


Comments also represent useful sources of information.

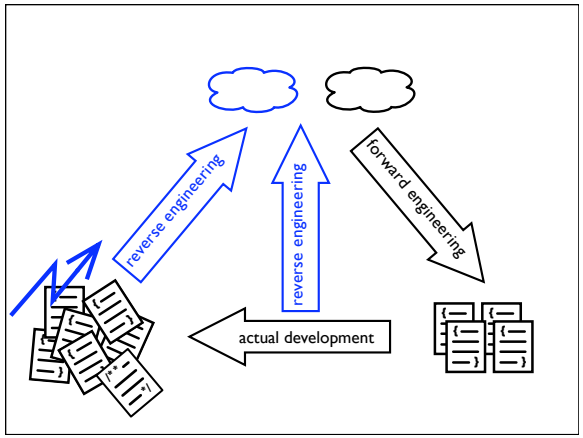


Daniel Schreck, Valentin Dallmeier and Thomas Zimmermann, How documentation evolves over time. In Proceedings of the International workshop on Principles of Software Evolution, p. 4 - 10, 2007.

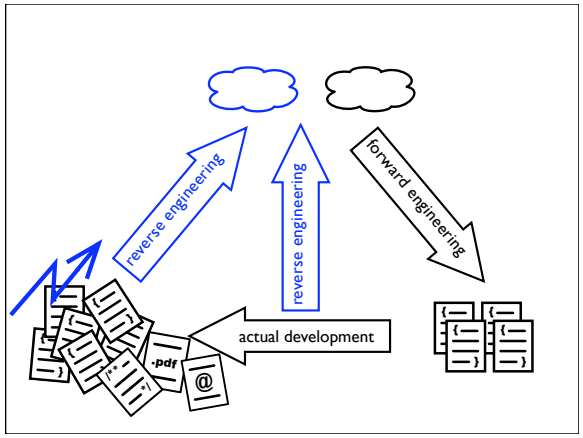
The picture shows a treemap of Eclipse, where the color is given by the coverage of Javadoc comments over methods. The lighter the color, the better covered by comments the part is. We can see how the API is much better commented than the Internal part.



This graph shows how the evolution of the quantity of comments during the Eclipse life time. For each sudden change in comments coverage a summary label is shown.

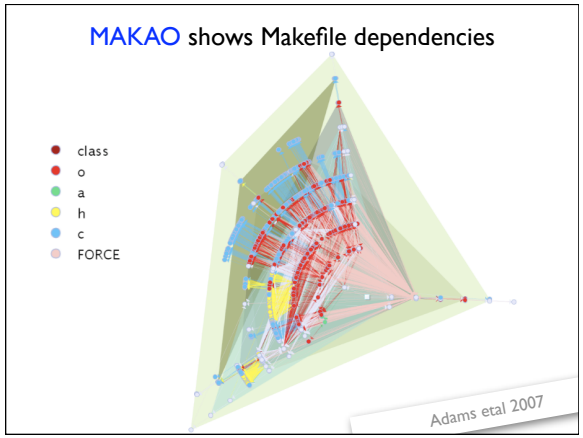


Comments … what else?



External documentation, mails are also useful sources of information.

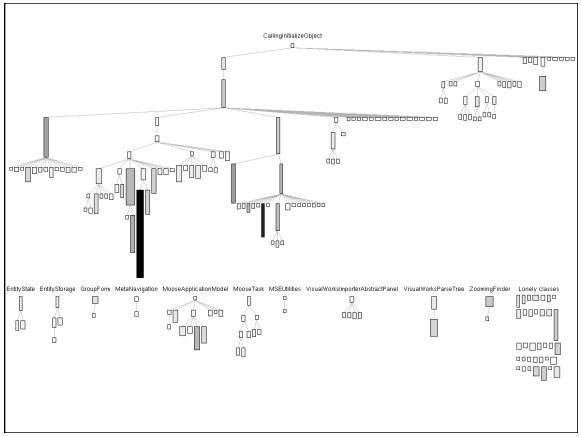
Davor Cubranic and Gail Murphy, “Hipikat: Recommending Pertinent Software Development Artifacts,” Proceedings 25th International Conference on Software Engineering (ICSE 2003), ACM Press, New York NY, 2003, pp. 408—418



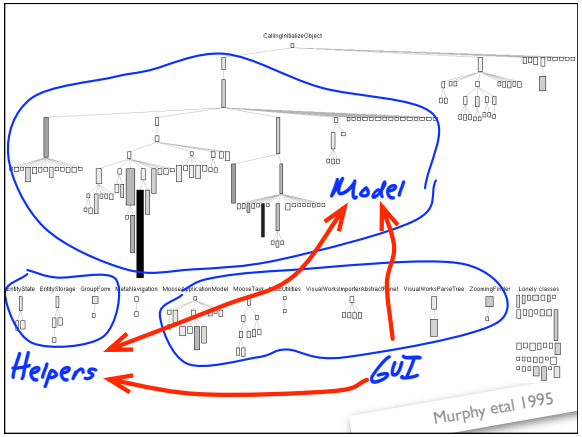
ADAMS, B., DE SCHUTTER, K., TROMP, H. and DE MEUTER, W. (2007). Design recovery and maintenance of build systems, in Proceedings of the 23rd IEEE International Conference on Software Maintenance (ICSM 2007), pages 114-123, IEEE Computer Society, Paris, France, October 2007 http://users.ugent.be/~badams/makao/

Furthermore, a great deal of knowledge is invested into developing and maintaining a scripts that build system. Investigating the build files can reveal dependencies that are otherwise difficult, or impossible to detect by looking at the source code alone.

The sources can tell you how the system looks like, but not why.

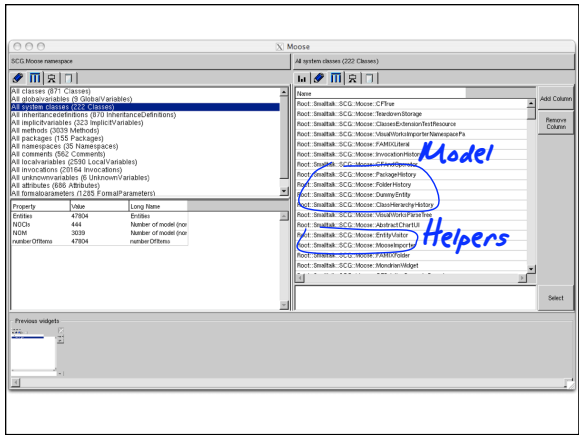


For example, by looking at the source code alone, we cannot know which classes are intended to be part of what logical component.

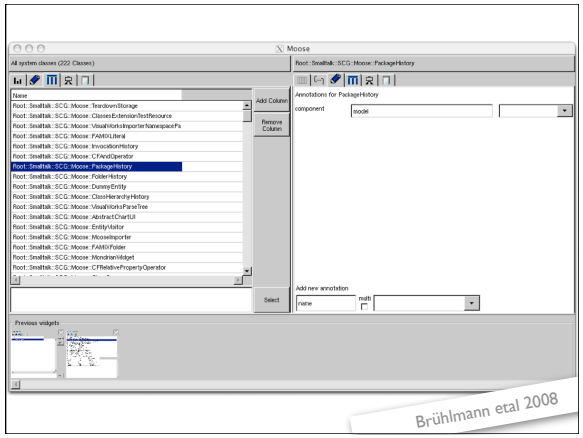


G. Murphy, D. Notkin and K. Sullivan, Software Reflexion Models: Bridging the gap between Source and High-Level Models, Proceedings of SIGSOFT '95, Third ACM SIGSOFT Symposium on the Foundations of Software Engineering, ACM Press, 1995, pp. 18-28.

Nevertheless, the developers possess this information, and we can use it to check the assumptions on the actual code. In this example, it would be useful for the tool to know which classes are part of the Model, the GUI or which are just Helpers.

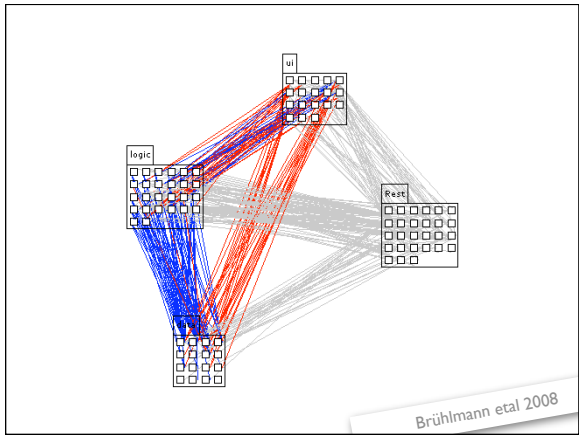


Not only architectural descriptions constitute useful external information. For example, a detection strategy detects code structures that are suspected of being flawed. However, like any automatic detection, it requires manual checking of the results. This manual step is also a means to confront the code with its intention (for example, a generated class should not count in the detection of god classes).



Andrea Brühlmann, Tudor Gîrba, Orla Greevy and Oscar Nierstrasz, “Enriching Reverse Engineering with Annotations,” International Conference on Model Driven Engineering Languages and Systems (Models 2008), Krzysztof Czarnecki et al. (Ed.), LNCS, vol. 5301, Springer-Verlag, 2008, pp. 660-674.

Metanool is a tool that allows the reverse engineer to define during analysis the annotations as properties attached to the entities. At a later point, these annotations can then be taken into account.

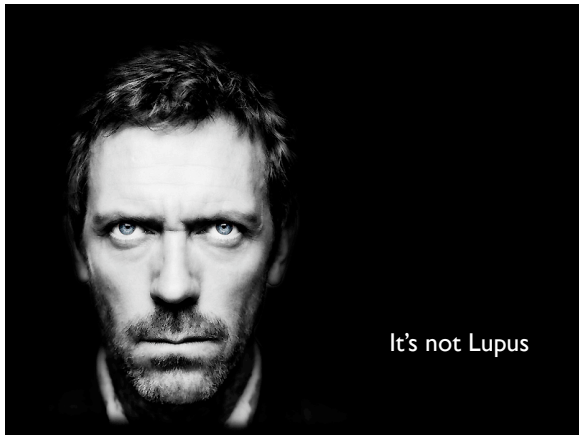


For example, given an architectural description, we can draw a visualization that shows with red the calls that appear to violate the architecture.

Simple tools can get you far
Queries help reduce the analysis space
Every system is special
Every technology is special
Software systems are more than code

Put findings in perspective.

There are many things to look



Fight the temptation of going for the first answer. Consider the uncertainty of data, the uncertainty of now knowing everything about the problem at hand, or the uncertainty inherent in human communication.

It can be Lupus, but it's not necessarily so.

You mainly see what you are looking for.

Tudor Gîrba
www.tudorgirba.com

This lecture aims to cover a wide spectrum of possible things to look for. The more things you know are possible, the more paths you will consider when trying to understand a problem.