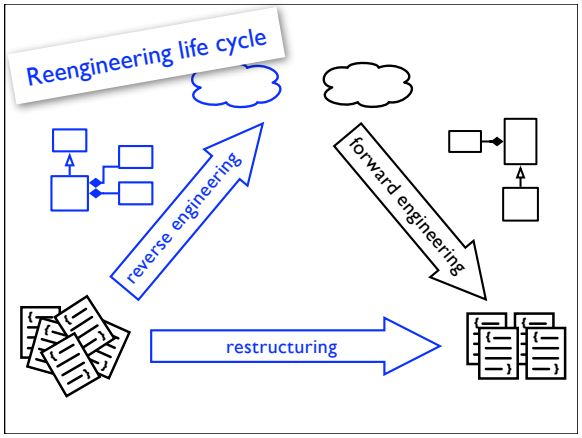
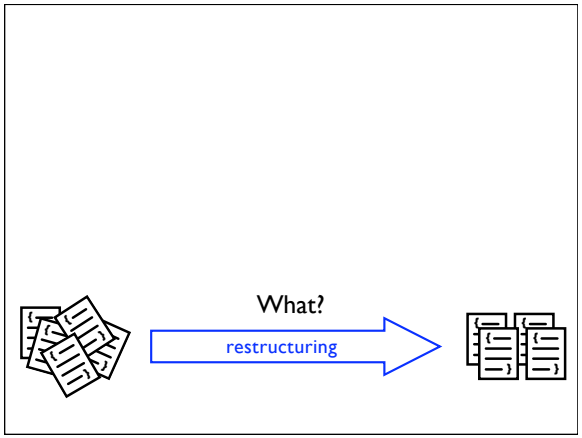
Restructuring

Tudor Gîrba
www.tudorgirba.com



Reengineering ... is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.

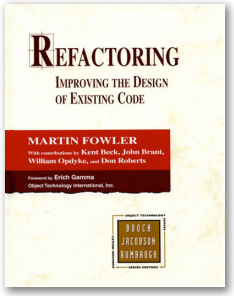
Elliot Chikofsky and James Cross II, “Reverse Engineering and Design Recovery: A Taxonomy,” IEEE Software, vol. 7, no. 1, January 1990, pp. 13—17.



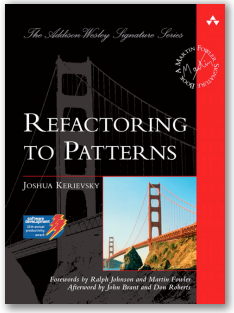
What is restructuring?

Restructuring is often taken as a synonym of refactoring

Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior



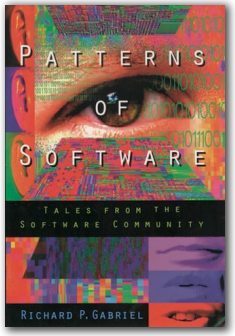
Refactoring is behavior-preserving transformation



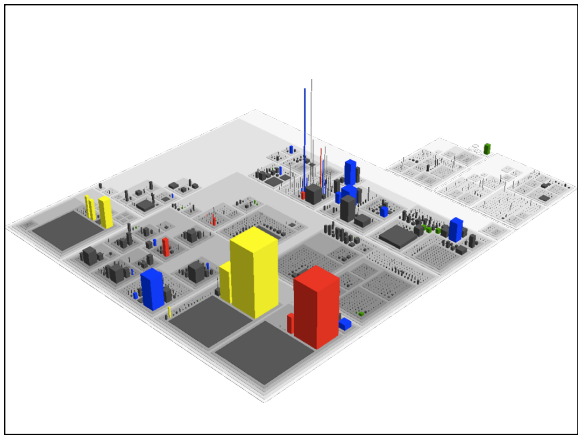
Restructuring is transforming
a program to fit current needs

Why do we need restructuring?

Software should be habitable



Richard Gabriel introduced the term of software “habitability” to denote the idea that software is a “place” that needs being tidy and clean so that we can leave in it.



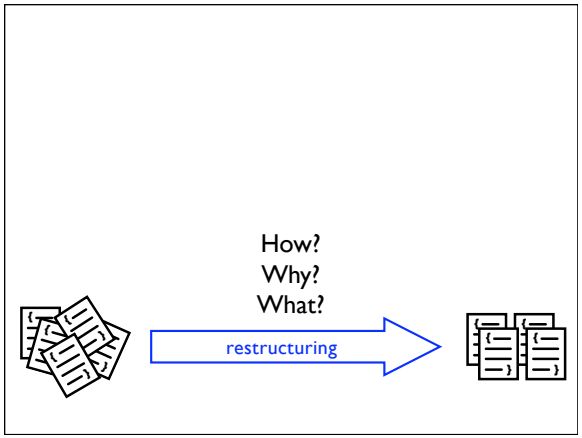
CodeCity takes the idea of habitability and gives software a landscape.

The secret to tidiness is to find the right place for every thing

Markus Denker

If it stinks, change it

Grandma Beck

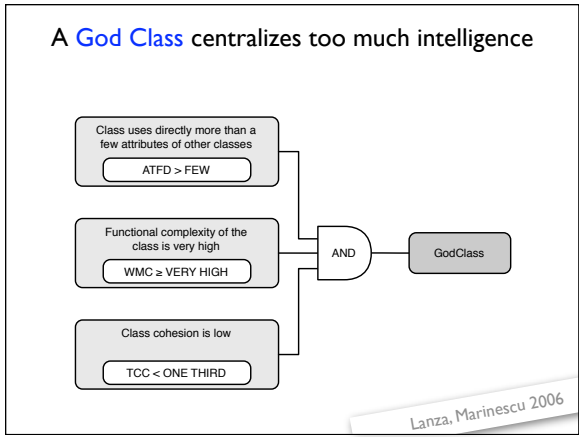


How to restructure?

There is significant literature in describing techniques for restructuring software systems. Here, we show four books:
- Martin Fowler, Kent Beck, John Brant, William Opdyke and Don Roberts, Refactoring: Improving the Design of Existing Code, Addison Wesley, 1999
- Serge Demeyer, Stéphane Ducasse and Oscar Nierstrasz, “Object-Oriented Reengineering: Patterns and Techniques,” 2005, tutorial
- Michele Lanza and Radu Marinescu, Object-Oriented Metrics in Practice, Springer-Verlag, 2006
- Joshua Kerievsky, “Refactoring to Patterns”, Addison Wesley, 2004

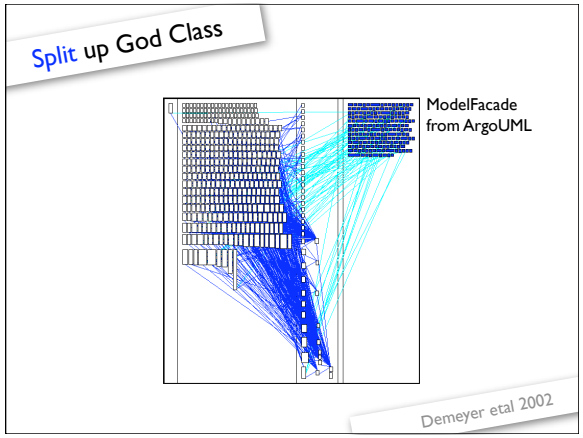
Take a critical look at design

Poems need to rhyme, music needs to sound good, software design needs to feel good.



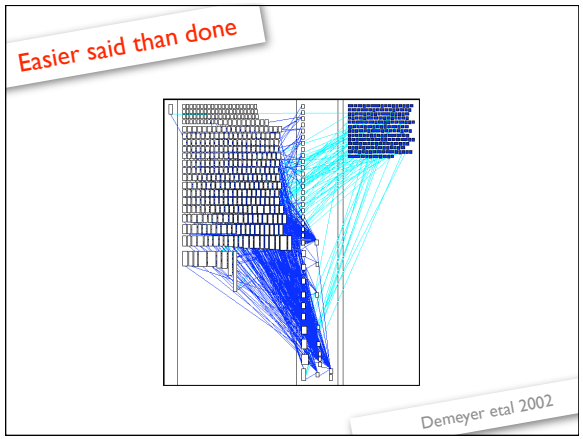
An example of a design problem is the God Class, that is a class that is too complex, uses data from other classes and has low cohesion. How to manually detect such a problem:

- Huge, monolithic class with no clear and simple responsibilities
- One single class contains all the logic and control flow
- Other classes only serve as passive data holders
- Names like “Manager”, “System”, “Root”, “*Controller*”
- Changes to the system always entail changes to the same class

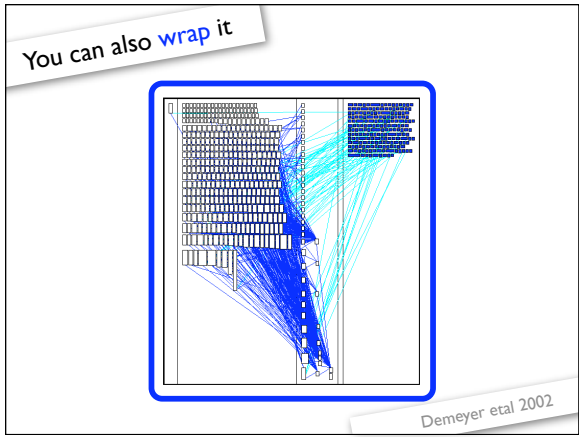


How to eradicate such a problem?

One pattern is Split up God Class that talks about Incrementally distributing responsibilities into slave classes.

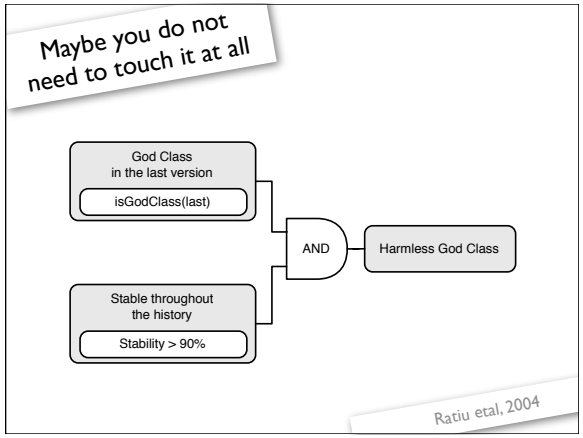


Difficult because God Class is a usually a huge blob
Identify cohesive set of attributes and methods
    Create classes for these sets
Identify all classes used as data holder and analyze how the god class uses them
    Iteratively Move Behavior close to the Data
    Use accessors to hide the transformation
    Use method delegation from the God Class to the providers
    Use Façade to minimize change in clients



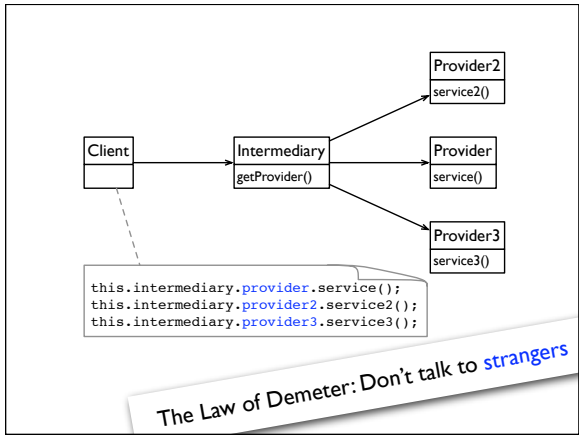
Another solution is to wrap the class and stop the spread of the contamination.

No matter what the choice is, try to always have a running system before decomposing the God Class.

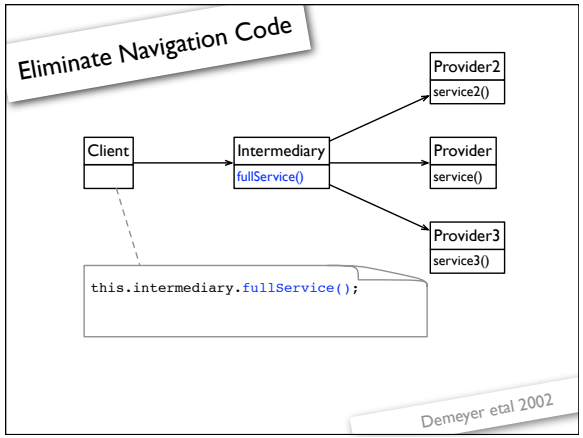


However, before going for eradicating the GodClass, make sure you actually have to do it. For example, if the GodClass was stable for a long period of time, it might not need touching, and an effort invested into it would not pay off.

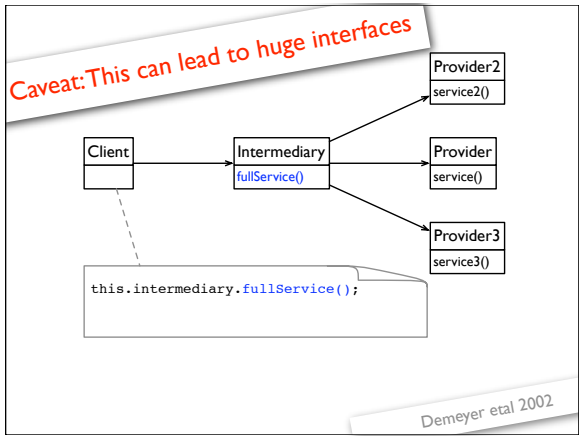
Nevertheless, the eradicating a GodClass problem is a complex one and it is too large to be tackled all at once. It is better to split it up into smaller problems.

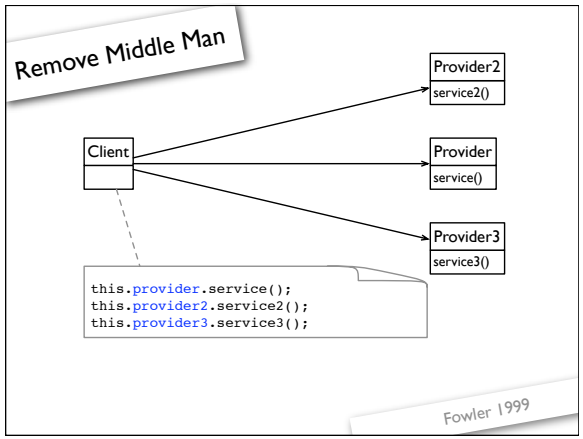


One part of the problem is related to the communication between classes. The Law of Demeter reveal the problem of navigational code which leads to tight coupling in the system.

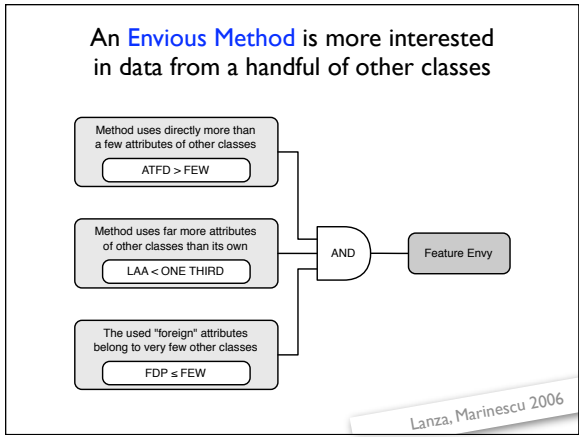


One solution to eliminate navigational code is to provide the service from the intermediary.



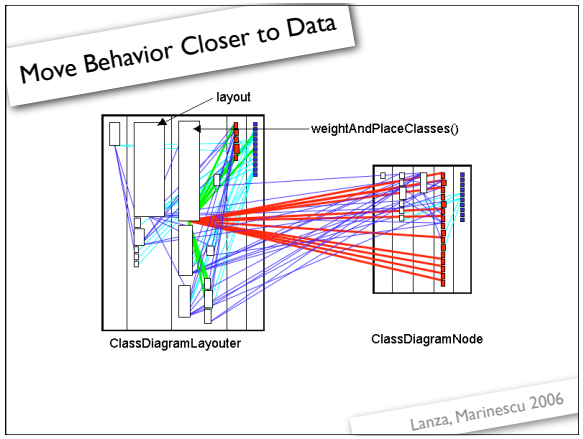


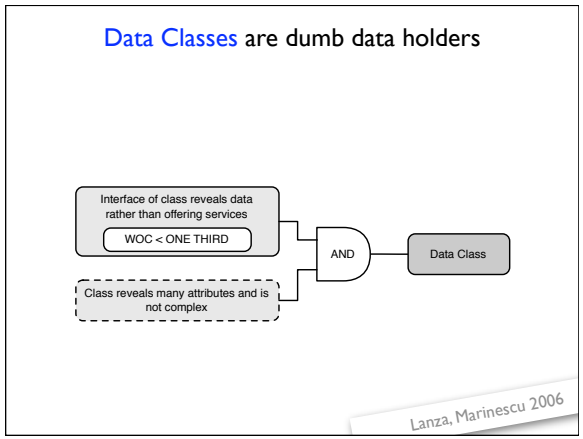
You should not talk to strangers, but you can get introduced to strangers, though. One refactoring that can be used is Remove Middle Man (p 160) so that the client can talk directly with the providers.



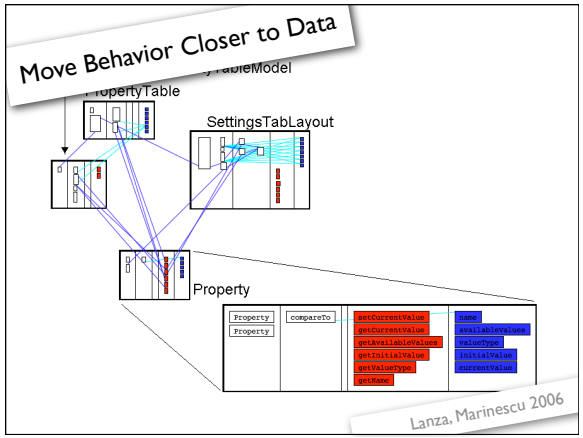
Feature Envy - Martin Fowler, Kent Beck, John Brant, William Opdyke and Don Roberts, Refactoring: Improving the Design of Existing Code, Addison Wesley, 1999.

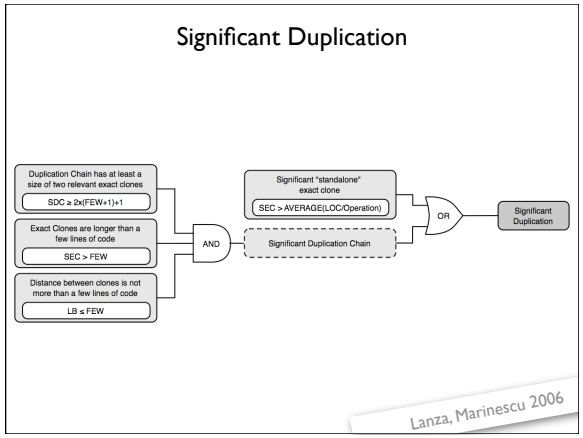
Test methods (and in particular the setUp methods) are typically detected as being envious. Of course, in this case it's not necessarily a design problem.

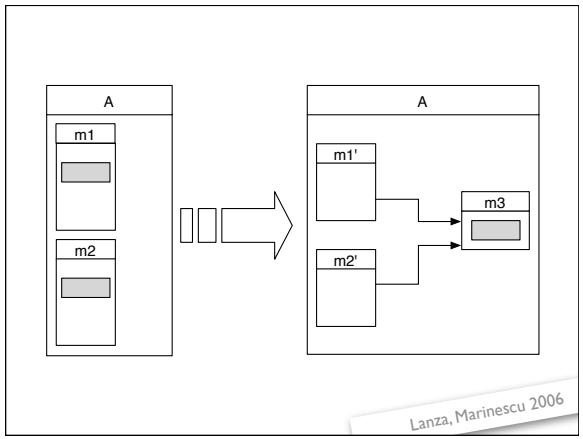


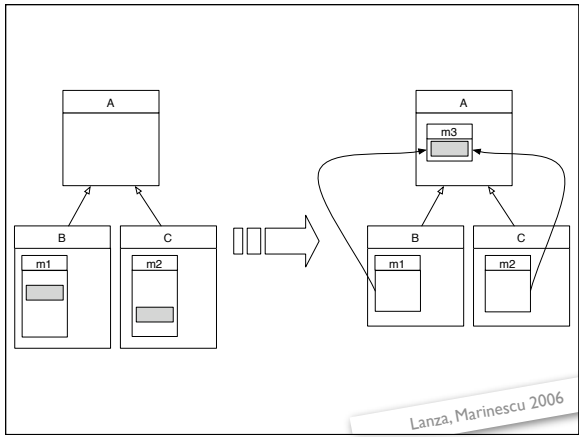


WOC: weight of class

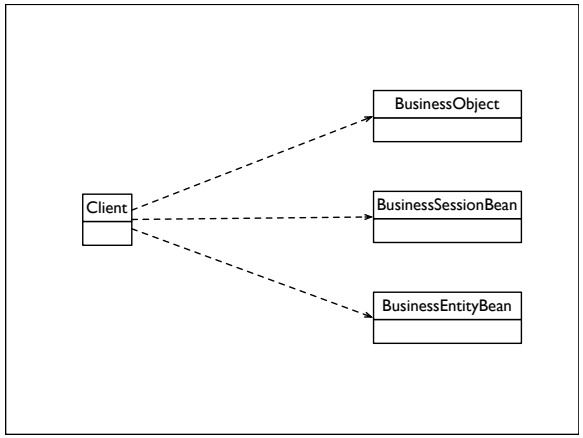

Move Behavior Closer to Data
SettingsTabLayout
Property
Property
Property
compareTo
getCurrentValue
getCurrentValue
getAvailableValues
getInitialValue
getValueType
getName
name
availableValues
valueType
initialValue
currentValue
Lanza, Marinescu 2006


Significant Duplication
Duplication Chain has at least a size of two relevant exact clones
SDC ≥ 2x(FEW+1)+1
Exact Clones are longer than a few lines of code
SEC > FEW
Distance between clones is not more than a few lines of code
LB ≤ FEW
AND
Significant "standalone" exact clone
SEC > AVERAGE(LOC/Operation)
Significant Duplication Chain
OR
Significant Duplication
Lanza, Marinescu 2006


A
m1
m2
A
m1'
m2'
m3
Lanza, Marinescu 2006

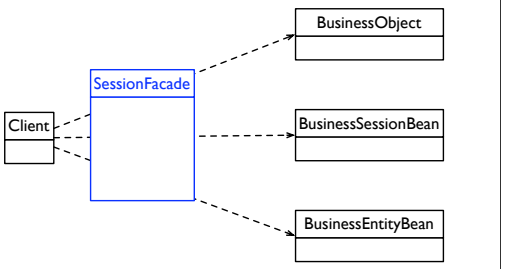


Restructuring can also improve performance

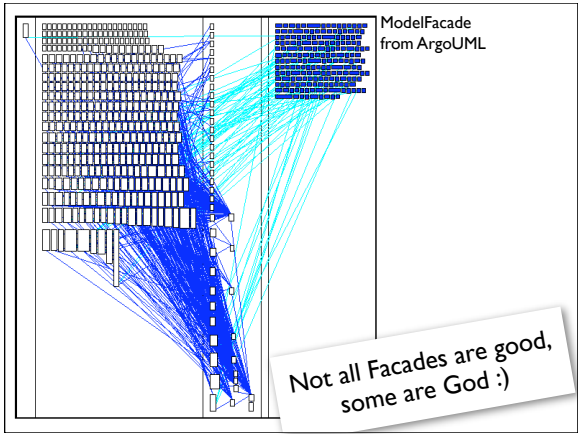


Problem:

- Tight coupling, which leads to direct dependence between clients and business objects;
- Too many method invocations between client and server, leading to network performance problems;
- Lack of a uniform client access strategy, exposing business objects to misuse.



java.sun.com

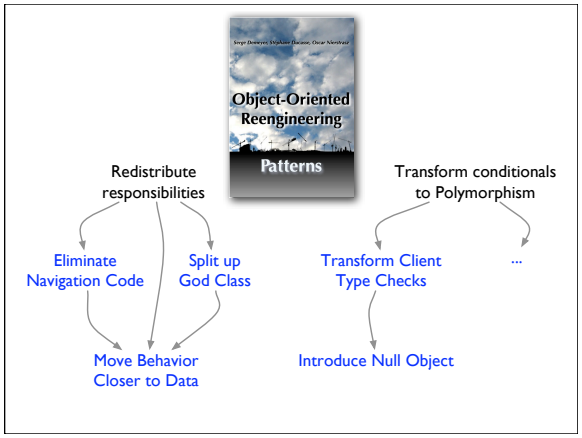


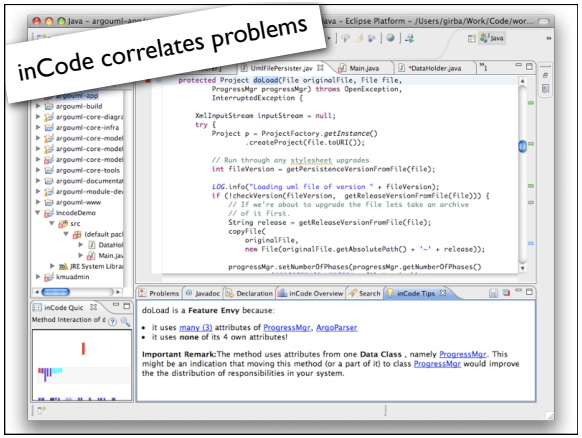
ModelFacade
from ArgoUML

Not all Facades are good,
some are God :)

Trouble never comes alone

Anonymous

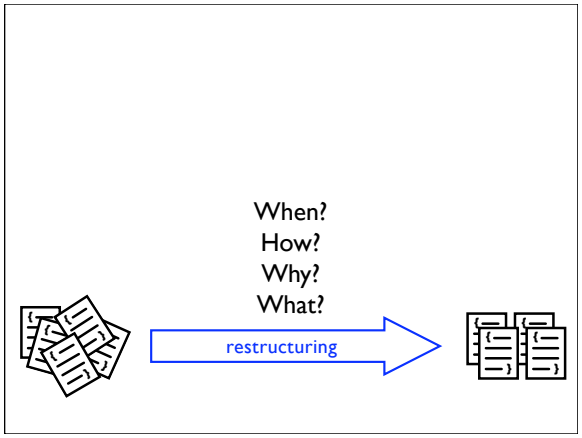




http://www.intooitus.com/inCode.html

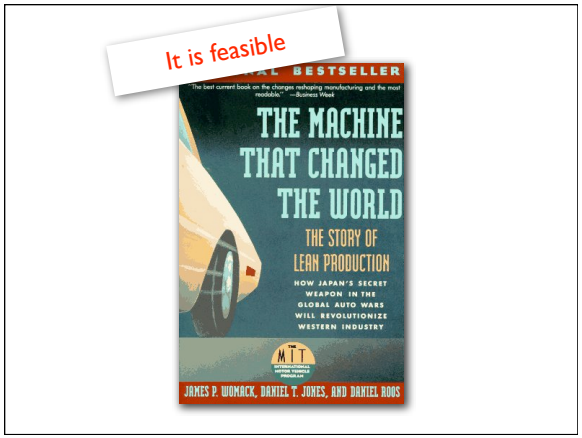
inCode correlates problems and offers advices of how to tackle these problems in concert. Furthermore, when possible it actually offers automatic refactoring options.

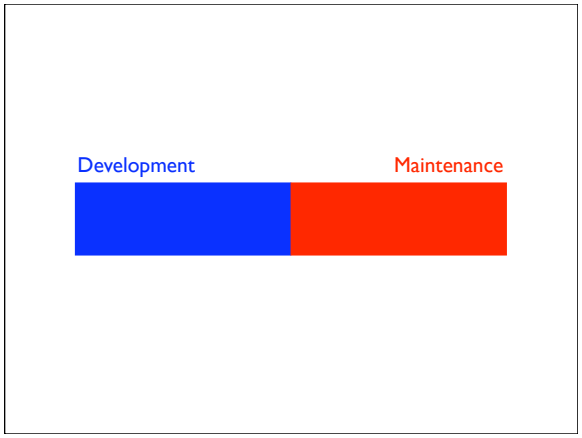
When should we restructure?



Restructure all the time

… hmm, is this feasible?





In the “classical” view on software, development is about not legacy code, and maintenance is about legacy code.
http://users.jyu.fi/~koskinen/smcosts.htm

Most of our effort should be concentrated on dealing with legacy code. Thus, instead of making a distinction between development and maintenance, we better just consider the entire effort as a continuous evolution.

Tudor Gîrba
www.tudorgirba.com