

Large Scale Restructuring

Nicolas Anquetil

Motivation

- Large scale “refactoring”
 - Once in a while, systems need to be completely redesigned
 - e.g. Apparition of internet, cloud, tablets
 - New business opportunity

Programming in the large VS Programming in the small

PROGRAMMING-IN-THE LARGE
VERSUS
PROGRAMMING-IN-THE-SMALL

Frank DeRemer
Hans Kron

University of California, Santa Cruz

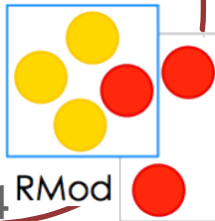
Key words and phrases

Module interconnection language, visibility, accessibility, scope of definition, external name, linking, system hierarchy, protection, information hiding, virtual machine, project management tool.

long and is easily comprehensible by a single person who understands the intended environment and function of the module.

We argue that structuring a large collection of

F.DeRemer, H.Kron, "Programming-in-the-Large versus Programming-in-the-small", ACM SIGPLAN Notices, Volume 10, Issue 6, June 1975



Programming in the large vs Programming in the small

PROGRAMMING-IN-THE LARGE
VERSUS
PROGRAMMING-IN-THE-SMALL

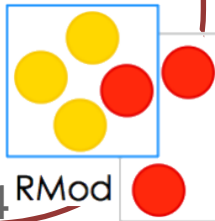
Frank DeRemer
Hans Kron

University of California, Santa Cruz

We need languages for **programming-in-the-small**, i.e. languages not unlike the common programming languages of today, **for writing modules**. We also need a **"module interconnection language"** **for knitting those modules together** into an integrated whole and for providing an overview that formally records the intent of the programmer(s) and that **can be checked for consistency by a compiler**.

Key words
Module
access
linking
hidina.

single per-
ment and
ction of



Refactoring in the large vs in the small

- Current IDEs offer refactorings in the small, at the level of single entities (class, method, variable)
 - Rename
 - Move
 - Extract
 - ...

Refactoring in the large vs Refactoring in the small

- We want “refactoring” tools in the large
 - At the level of system, or multi-entities
 - Rename in batch
 - Move several entities to create a new module
 - Extract a (small) class from a big one
 - Complex “refactorings” involving various steps
 - ...

Large scale “refactorings”

G. Santos, A. Etien, N. Anquetil, S. Ducasse, M.T. Valente. “Recording and Replaying System Specific, Source Code Transformations”. SCAM 2015

- Help programmers perform systematic code transformations

“Refactoring” with macros

- PackageManager 0.58 → 0.59

Applied **19** times

- platform

- package addPlatformRequirement: #'pharo'.
- package addProvision: #'Grease-Core-Platform'

+ platformRequirements

- + ^ #(#'pharo')
- + **provisions**
- + ^ #(#'Grease-Core-Platform')

- platform

- package addPlatformRequirement: #'pharo2.x'.
- package addProvision: #'Seaside-Canvas-Platform'

+ platformRequirements

- + ^ #(#'pharo2.x')
- + **provisions**
- + ^ #(#'Seaside-Canvas-Platform')

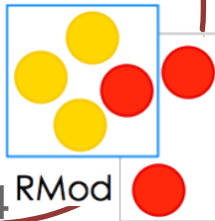


“Refactoring” with macros

- Problems:
 - Complex
 - Tedious
 - Error prone
- Proposition:
 - Manually perform the changes once + record
 - Generalize the recorded changes
 - Replay the changes in other locations

Large scale “refactorings”

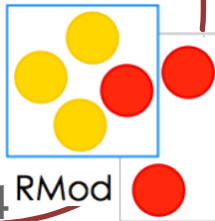
- Refactoring + intelligent follow-up
 - Refactoring in-the-small
 - + additional checks on the system:
 - Did you notice that ... ?
 - Would you like to also ... ?



System restructuring (1)

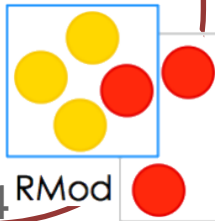
B. Govin (Thales)

- Help programmers re-define the architecture of a system
- Extract component software architecture from a real time, embedded system



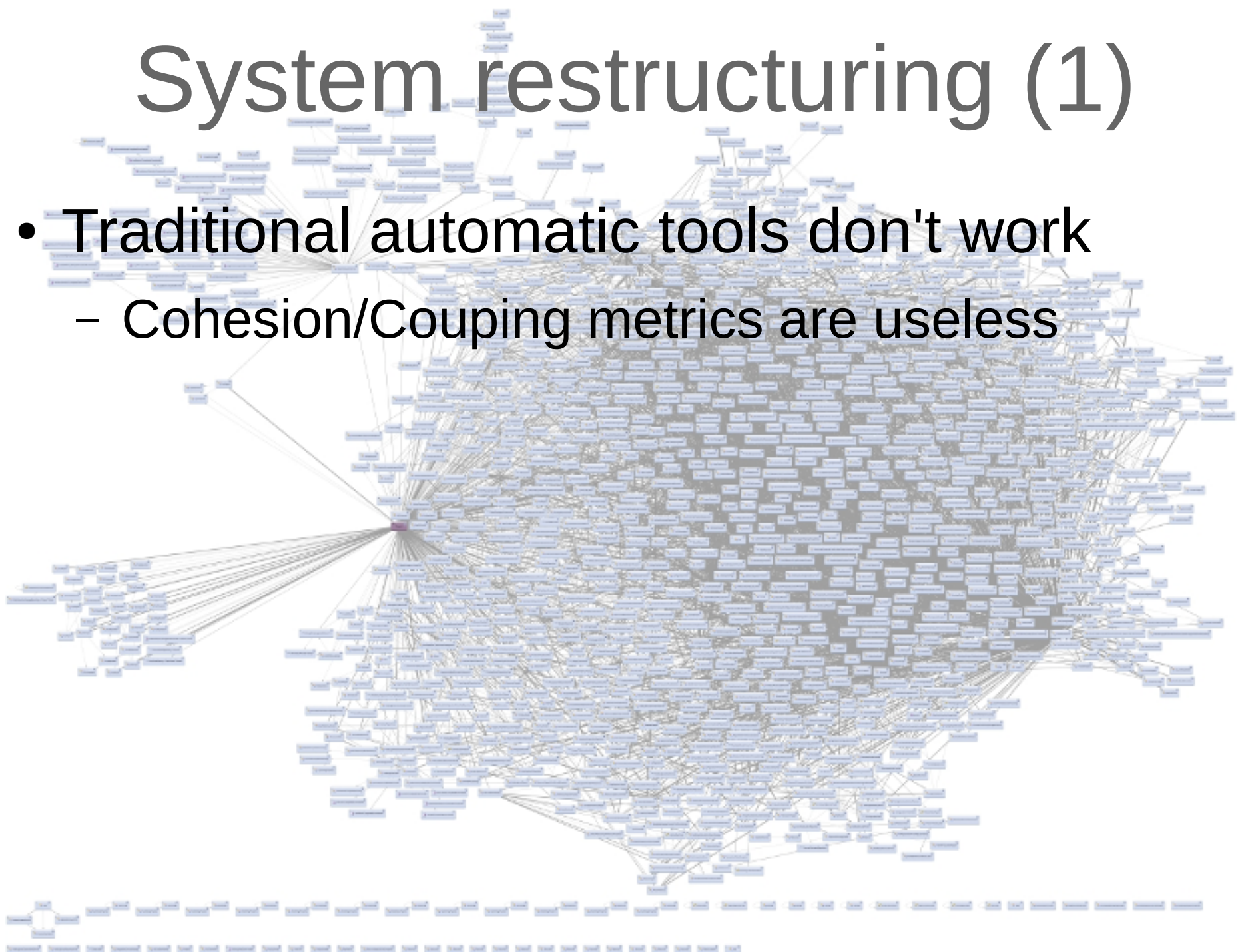
System restructuring (1)

- Help programmers re-define the architecture of a system
 - Process and tools to help engineers
 - Work at all levels:
from packages to individual function calls



System restructuring (1)

- Traditional automatic tools don't work
 - Cohesion/Coupling metrics are useless



System restructuring (1)

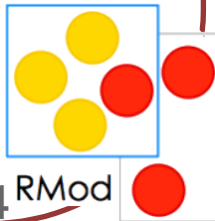
- Traditional automatic tools don't work
 - Cohesion/Coupling metrics are useless
- Try using engineers knowledge
 - Identify “core” elements of components
 - + agglomerate elements around cores



System restructuring (2)

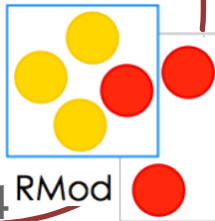
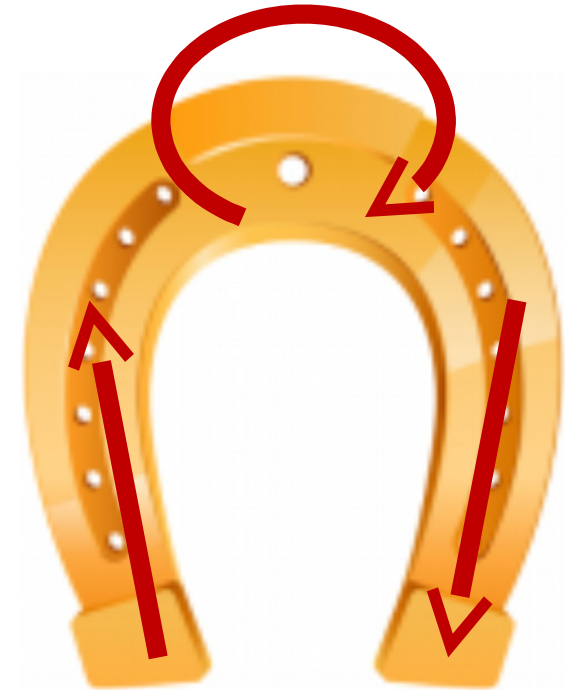
G. Santos, N. Anquetil, A. Etien, S. Ducasse, M.T. Valente. “OrionPlanning: Improving Modularization and Checking Consistency on Software Architecture”. VISOFIT 2015

- Help programmers restructure the architecture of a system



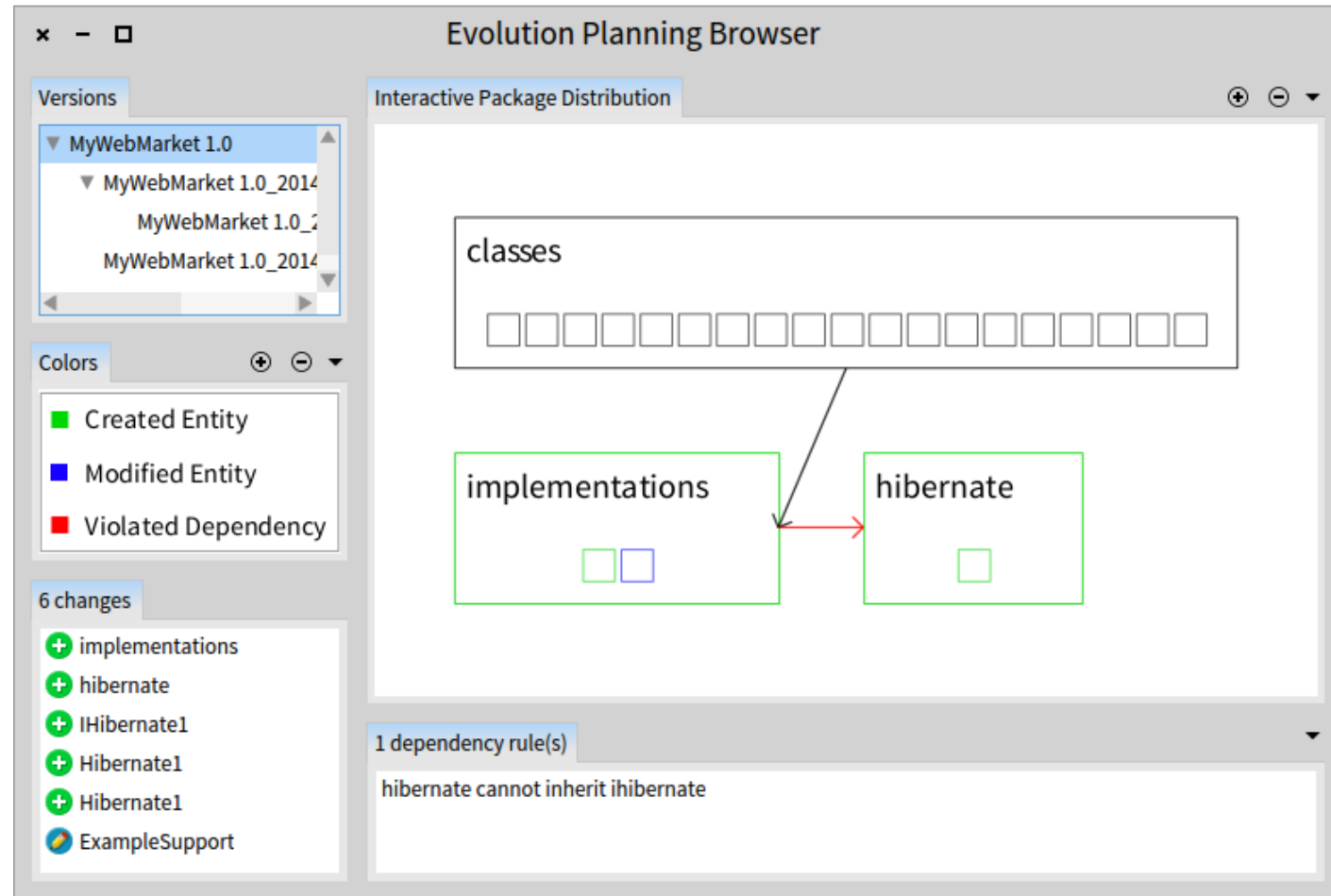
System restructuring (2)

- Horseshoe approach:
 - From source code to model
 - Work on the model
 - Propagate changes back to the code



System restructuring (2)

- Orion planner
 - Define wished architecture
 - Try things
 - Check validity



System restructuring (2)

- Orion planner
 - Define wished architecture
 - Try things

Model of the system
(architecture)
editable

The screenshot displays the Evolution Planning Browser interface. On the left, there are panels for 'Versions' (showing a tree of MyWebMarket 1.0 sub-versions), 'Colors' (with a legend for Created Entity in green, Modified Entity in blue, and Violated Dependency in red), and '6 changes' (listing additions for implementations, hibernate, and IHibernate1). The main area, titled 'Interactive Package Distribution', shows a diagram with a 'classes' box at the top containing 15 empty slots. Below it are two boxes: 'implementations' (containing a green and a blue square) and 'hibernate' (containing a green square). A red arrow points from the 'implementations' box to the 'hibernate' box. At the bottom, a '1 dependency rule(s)' panel lists the rule: 'hibernate cannot inherit ihibernate'. A blue arrow from the text above points to the main diagram area.

System restructuring (2)

- Orion planner
 - Try things

Versions of model

The screenshot displays the Evolution Planning Browser interface. On the left, a 'Versions' tree shows a hierarchy starting with 'MyWebMarket 1.0', which is expanded to show sub-versions: 'MyWebMarket 1.0_2014', 'MyWebMarket 1.0_2', and 'MyWebMarket 1.0_2014'. A blue arrow points from the text 'Versions of model' to the 'MyWebMarket 1.0' node. Below the tree is a 'Colors' section with three categories: 'Created Entity' (green square), 'Modified Entity' (blue square), and 'Violated Dependency' (red square). At the bottom left, a '6 changes' list shows: '+ implementations', '+ hibernate', '+ IHibernate1', '+ Hibernate1', '+ Hibernate1', and '+ ExampleSupport'. The main area, titled 'Interactive Package Distribution', contains a dependency graph. At the top is a 'classes' box with 15 empty slots. Below it are two boxes: 'implementations' (containing a green and a blue square) and 'hibernate' (containing a green square). A black arrow points from the 'classes' box to the 'implementations' box, and a red arrow points from the 'implementations' box to the 'hibernate' box. At the bottom, a '1 dependency rule(s)' section lists the rule: 'hibernate cannot inherit ihibernate'.

System restructuring (2)

- Orion planner
 - Check validity

Validation
(architectural rules)

The screenshot displays the Evolution Planning Browser interface. On the left, a 'Versions' tree shows 'MyWebMarket 1.0' with sub-versions 'MyWebMarket 1.0_2014'. Below it, a 'Colors' section lists 'Created Entity' (green), 'Modified Entity' (blue), and 'Violated Dependency' (red). A '6 changes' list includes '+ implementations', '+ hibernate', '+ IHibernate1', '+ Hibernate1', '+ Hibernate1', and '+ ExampleSupport'. The main area, 'Interactive Package Distribution', shows a diagram with a 'classes' box at the top, 'implementations' and 'hibernate' boxes below. A red arrow points from 'implementations' to 'hibernate'. At the bottom, a '1 dependency rule(s)' section shows the error: 'hibernate cannot inherit ihibernate'. A blue arrow points from the text 'Validation (architectural rules)' to this error message.

System restructuring (2)

- Still missing: “Do-it” button

