

Reifying Refactorings on SCM Systems

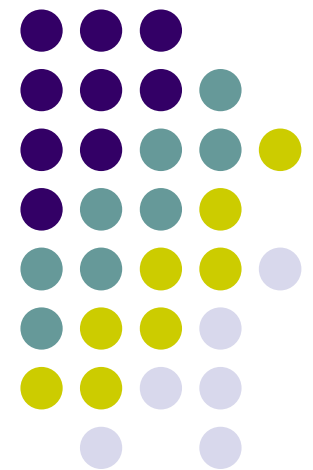
MSc Thesis Project

Author: Diego Tubello (Baufest)

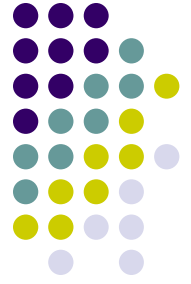
Direction: Hernan Wilkinson (Mercap)

Department of Computer Science

FCEyN – UBA



Smalltalks 2007



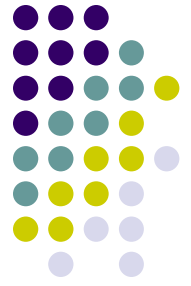
Contents

- Background
- Motivation example
- Objective
- Proposed solution
- Integrating with refactoring information
- Implementation
- Conclusions
- Future work



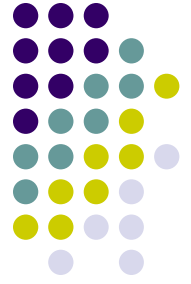
Background: refactorings

- Refactorings are structural changes that do not affect the behavior of the program.
- Usually affect many parts of the system.
- There are tools to perform them automatically.

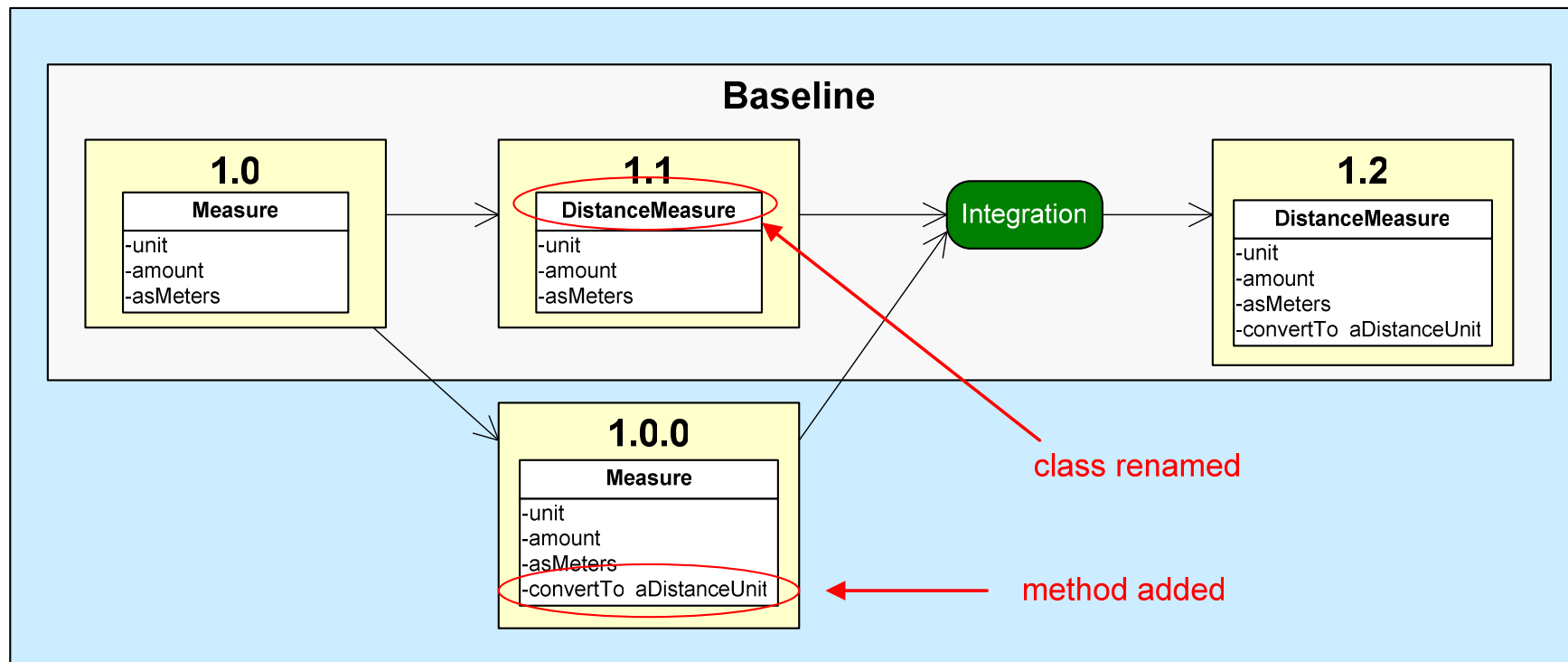


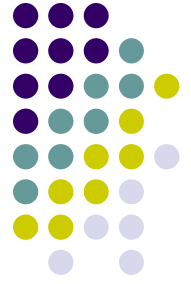
Background: team development

- The work of the team is integrated using SCM tools.
- There is a baseline where developer changes are integrated with the current version.
- Conflicts arise when the same entities such as classes or methods are modified by different developers.
- Example: ENVY, the SCM tool of Visual Age
 - Classes, methods and modules are versioned



Motivation Example

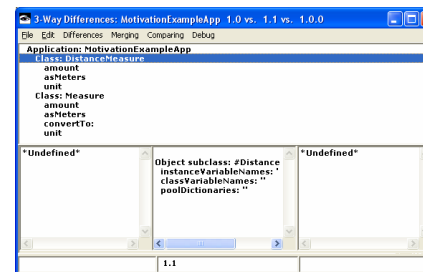




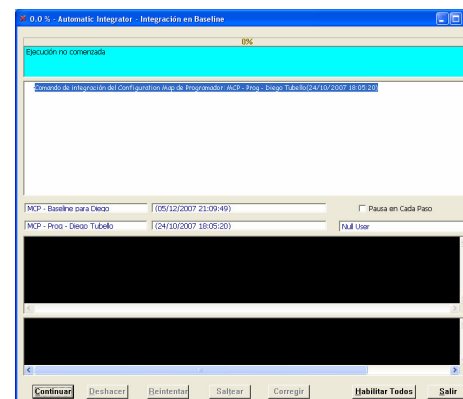
Motivation Example

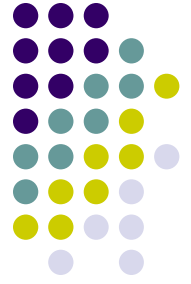
- Demo: integrating this example in Visual Age/ENVY with the current tools:

- 3-way diff



- Automatic Integrator





Motivation Example

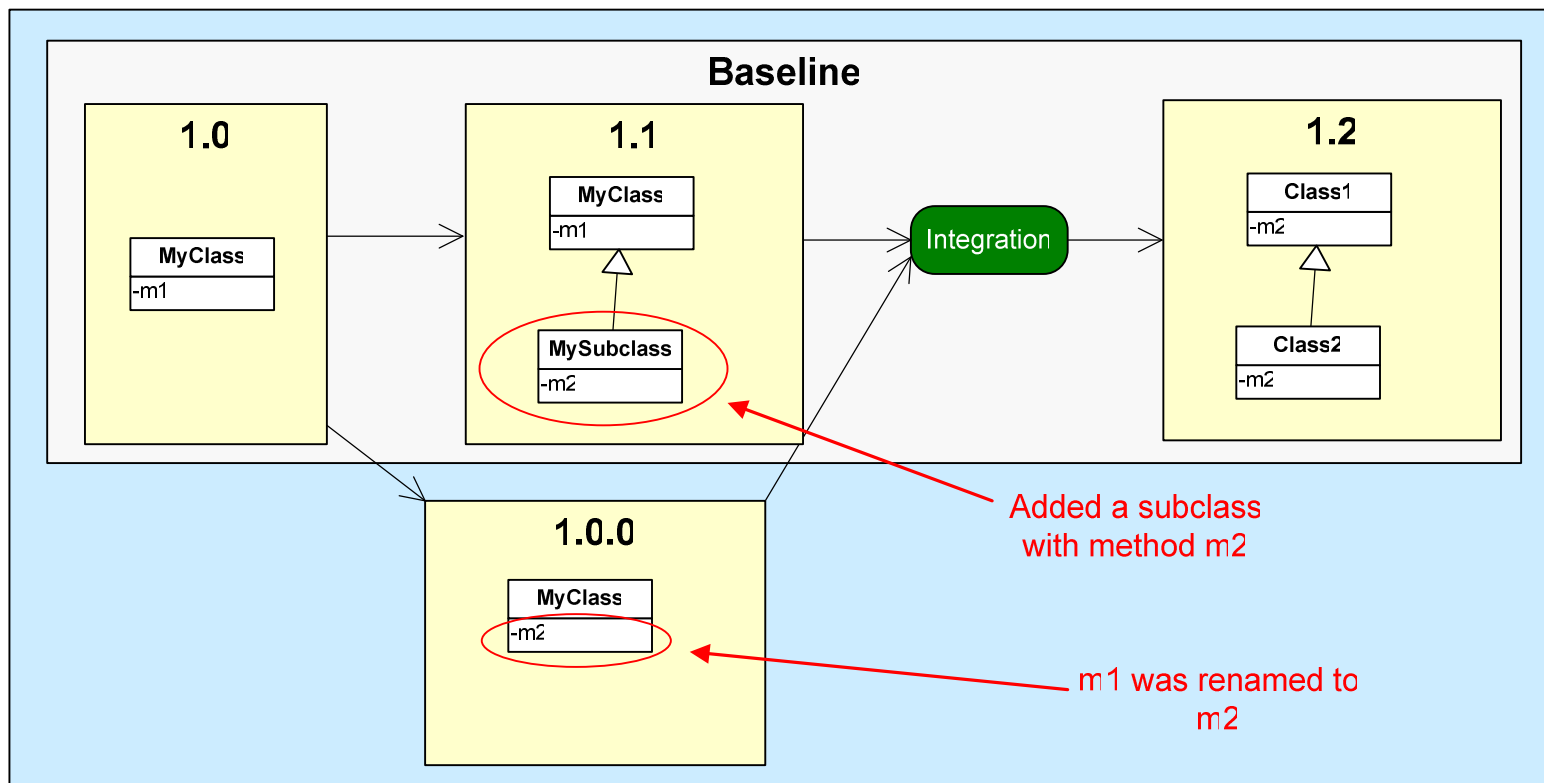
What's missing ?

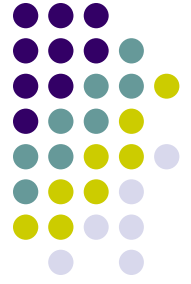
- SCM tools are not aware that a class was renamed.
- There is no way to know that DistanceMeasure was Measure: history is lost.
- Integration tools see the refactoring as a set of distributed and unconnected changes:
 - because SCM stores the changes in that way



Motivation Example

- Undetected conflicts





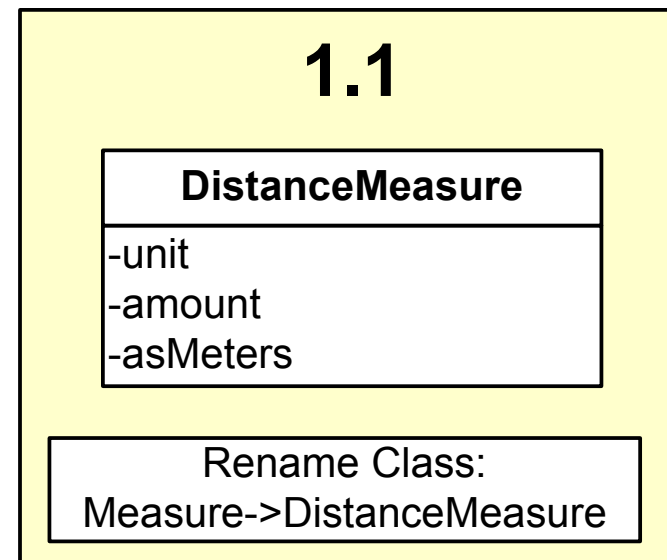
Objectives

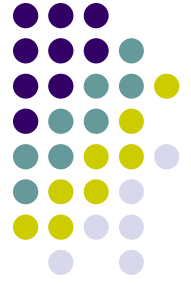
- Reify the concept of refactoring in an SCM tool.
- Modify integration tools to use refactoring information.
- Allow refactoring from different branches to be integrated automatically.
- When automatic integration is not possible, provide all the information necessary for the integrator to take a decision.
- Make an implementation in Visual Age/ENVY. Starting with class and method renames.



Proposed Solution

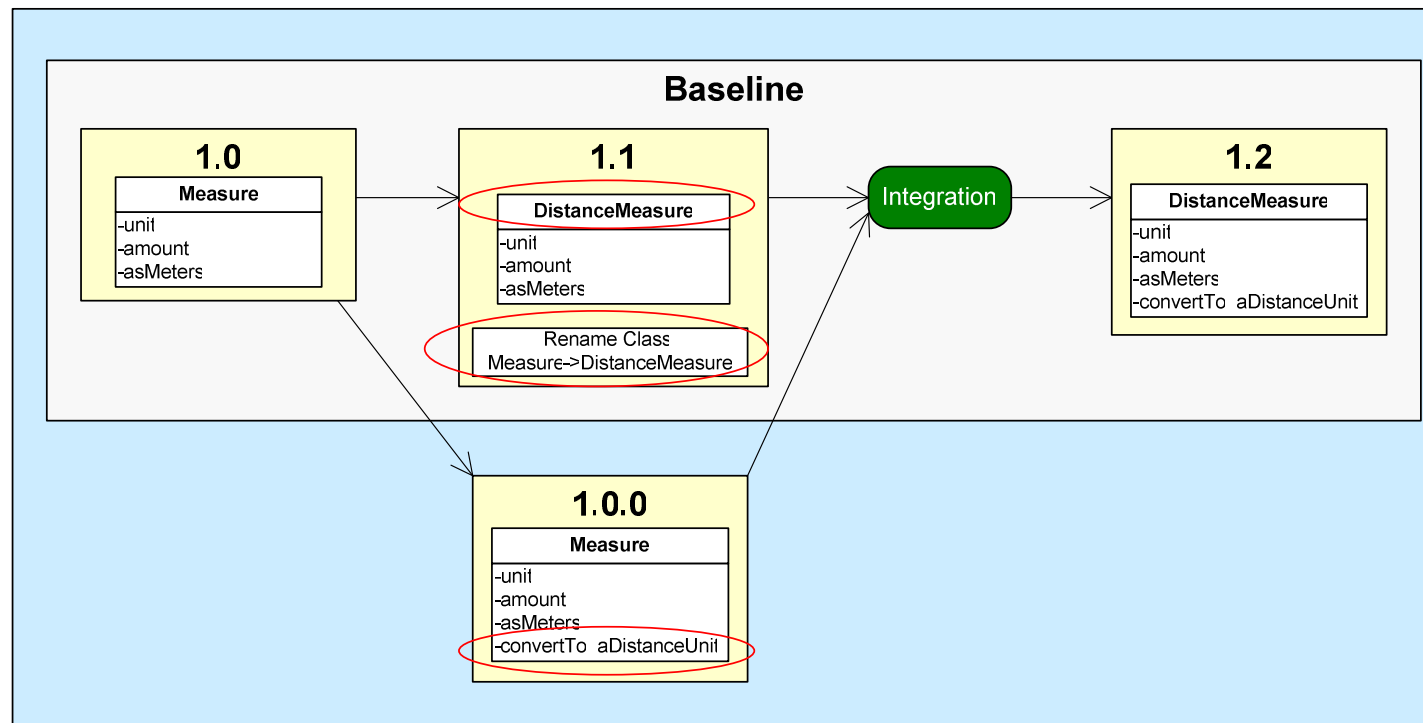
- Record refactorings when made by developers.
- Keep the refactoring information when doing the integration





Proposed Solution

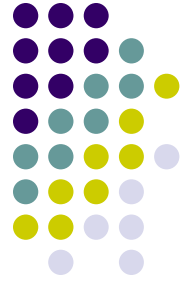
- Use the information of stored refactorings to:
 - integrate new changes
 - track evolution of entities: travel through the entity history





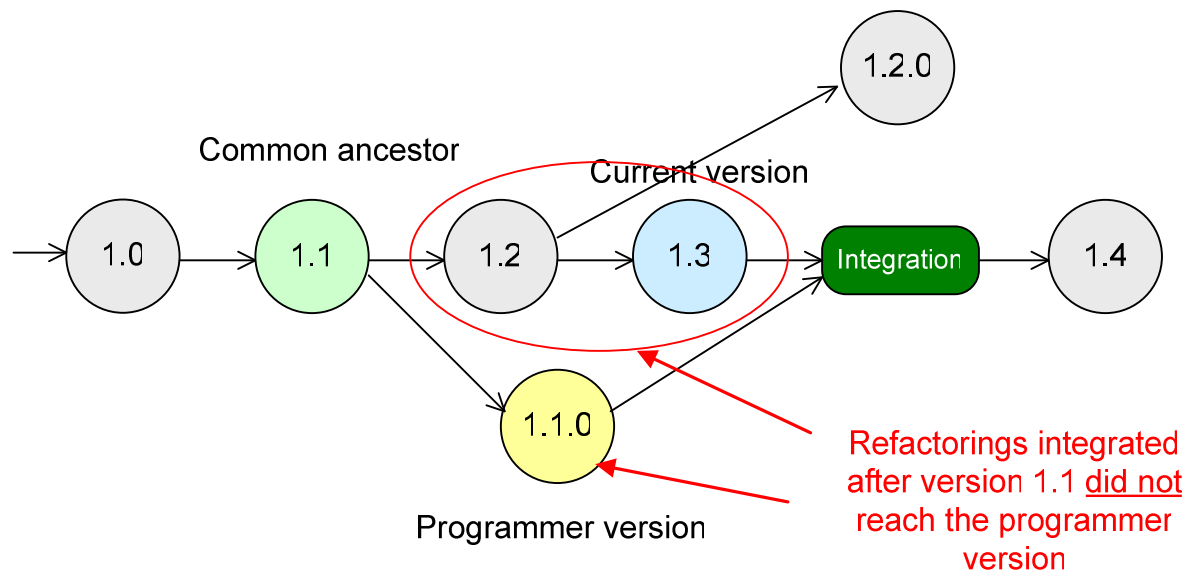
Integrating with refactoring information

- What do we need to consider when integrating changes to a baseline ?
 - Refactorings already integrated in baseline that did not reach the branch
 - Refactorings made in the branch that did not reach the baseline
 - Manual changes



Integrating with refactoring information

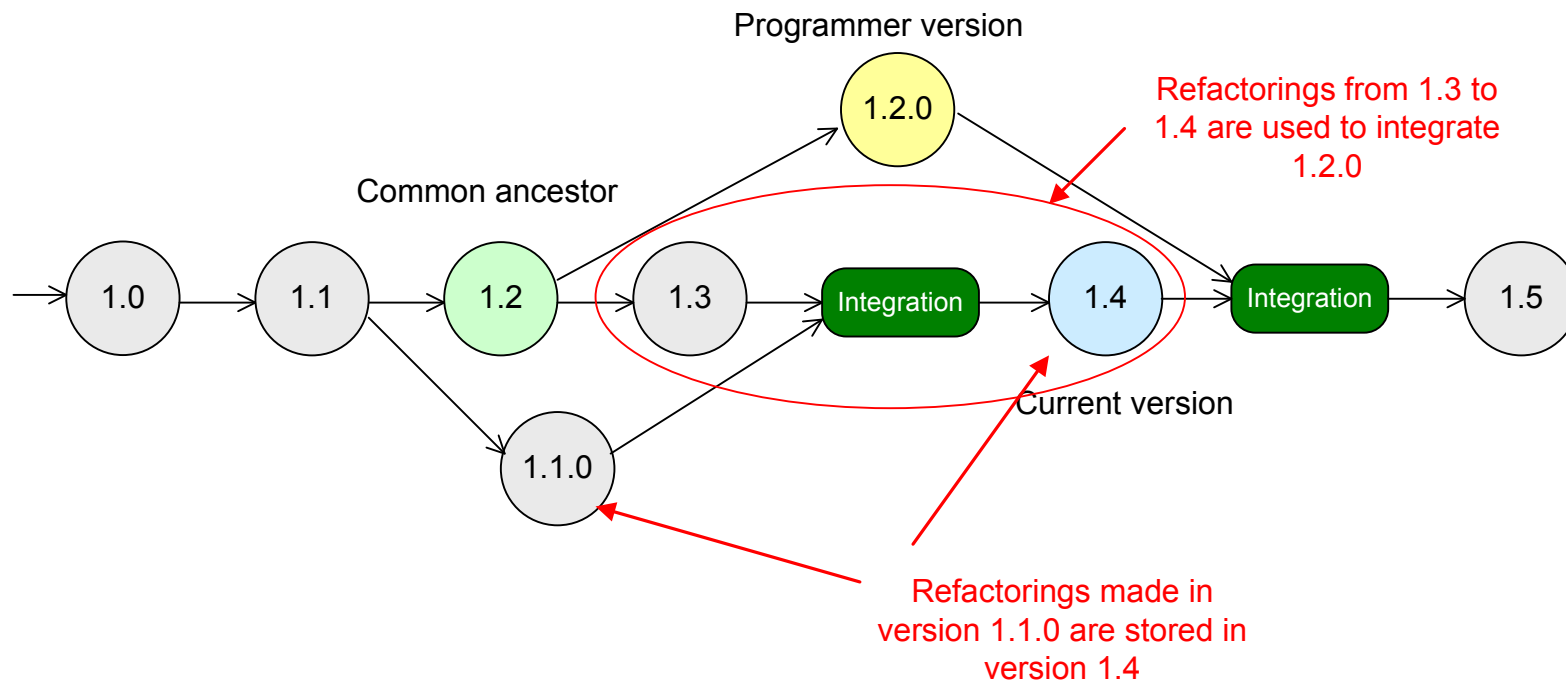
- Refactorings already integrated in baseline that did not reach the branch



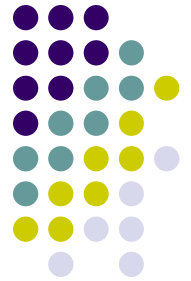


Integrating with refactoring information

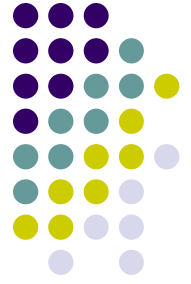
- Refactorings already integrated in baseline that did not reach the branch



Integrating with refactoring information

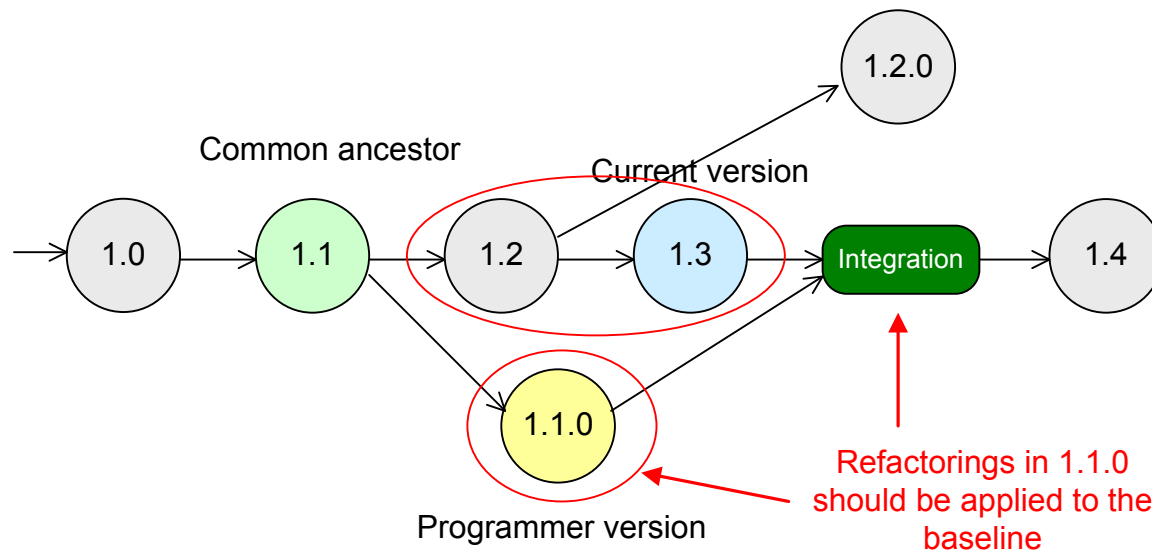


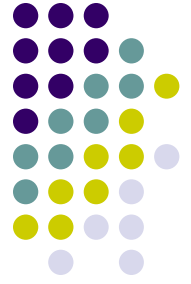
- Refactorings made in the branch that did not reach the baseline:
 - We should find a way to “apply” them to the unreached code
 - It’s not straight forward:
 - We have a different landscape that when the refactoring was applied originally
 - We should extend the concept of refactoring to handle this type of scenario.
 - Refactorings should be applied in the same order that were performed the first time.



Integrating with refactoring information

- Refactorings made in the branch that did not reach the baseline:





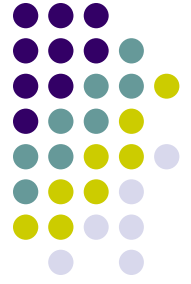
Implementation

- Why Visual Age and ENVY?
 - There are no tools like these for this environment.
 - Visual Age And ENVY are tight integrated.
 - ENVY is suitable for extension: we can associate additional information to versions.
 - We already have tools for automatic integration that can be extended.
- We have two main parts:
 - Store developer refactorings in ENVY together with editing changes
 - Modify the automatic integrator to use refactoring information
- The solution is being developed using TDD.



Implementation

- Store developer refactorings in ENVY together with editing changes.
 - Keep record of refactorings made by the developer
 - Save them together with the developer changes.
 - Make use ENVY's capability of storing any kind of object.



Implementation

- Modify the automatic integrator to use refactoring information:
 - Apply refactorings made by the programmer to baseline
 - Compare entities in different versions using refactorings information to find differences.
 - Store the programmer refactorings in the newly created version in the baseline.

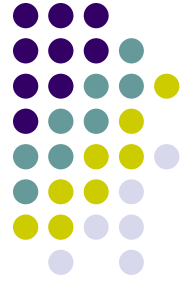


But...

How difficult would be to do implement the something like this for another language or SCM tool ?

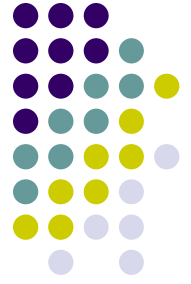
For example, for a widely used combination like:

Java and CVS



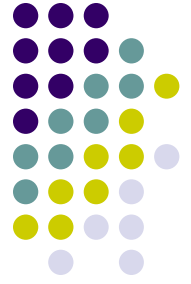
Other languages/SCM tools ?

- CVS is file based, it sees the programs as a set of files that are versioned. It does not know the difference between:
 - A Java class
 - An html page
 - A cooking recipe
- Changes are stored as differences in the files that conform the program, no matter what is inside them.
- How can we make file based tools be aware of a class rename if it does not know what a class is ?
- Versioned model vs versioned files.



Conclusions

- Less conflicts and more automatic integrations
- We have more information to take decisions when real conflicts arise
- More freedom to perform refactorings
- Track the evolution of entities:
 - better understanding of the evolution of the model



Future work

- Finish the implementation in Visual Age for renames
- Test it in a real team environment
- Extend the tool to work with more refactorings.
- Extend tools to browse history to be aware of refactorings.

Questions?

