

Systematic Source Code Transformations

Gustavo SANTOS



Presentation

- Bachelor in Computer Science
- Master in Computer Science
- Short visit to RMoD team
- Funded by CAPES (Brazil)
 - Science Without Borders program



Areas of Interest

- Change Impact Analysis
- Modularization
- Information Retrieval
- Quality Metrics
- ...
- Software Evolution

Introduction

Identifying Systematic Code Transformations
Replaying Systematic Code Transformations

Software Evolution

- Software is in constant evolution to remain useful [Leh1980]
- Evolution is composed of changes
 - Performed in distinct moments in time
 - By many developers
- Developers need to reason about code changes [Hat2011]

Refactoring

- Change made to the internal structure to make it [...] cheaper to modify [...] without changing its observable behavior [Fow1999]
- Regular and applied to few entities [Avg2013]

Rearchitecting

- Rearchitecting (large refactoring) [Avg2013]
 - Update APIs
 - Improve the organization
- Less frequent but involves the entire system
- Rearchitecting dataset as product of my Master [San2014]

Software Evolution

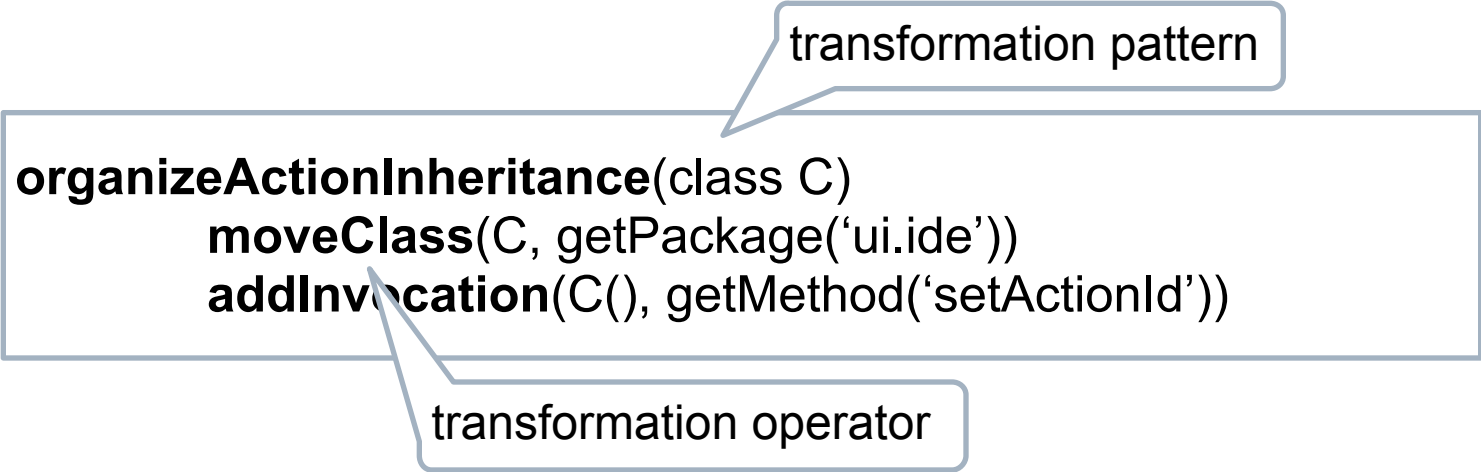
- Systematic Code Changes
- In Eclipse 2.1 → 3.0, for example:

move class C to a package 'ui.ide'
in the initializer of C, **add invocation** to method 'setActionId'

Applied **22** times

Transformation Pattern

- Sequences of transformations that are applied to **similar** code entities



```
organizeActionInheritance(class C)  
  moveClass(C, getPackage('ui.ide'))  
  addInvocation(C(), getMethod('setActionId'))
```

transformation pattern

transformation operator

- Operators can be atomic or aggregated

Conclusions

- Transformation patterns can be:
 - Complex
 - Tedious
 - Error-prone
- Automation is needed

Introduction

Identifying Systematic Code Transformations

Replaying Systematic Code Transformations

Related Work

- We found work concerning such activity

	Application	Destination of changes
[Pan2009]	Bug Fixes	inside methods only
[And2008]	API evolution	inside methods only
[Kim2013]	General	files only
[Mil2014]	General	inside methods only
[Jia2015]	General	inside classes only

Related Work

- No existing work in rearchitecting
- Destination of the changes
 - More complex operators
- Properties of the entities involved
 - More system specific patterns

Investigative Study

- Identify similar changes semi-automatically
- Rearchitecting dataset
 - Performed manually by the developers
 - Systems before and after rearchitecting

Methodology

- Identify similar changes semi-automatically
 - Extract the diff between versions
 - Filter groups of similar changes
 - Manually identify similar properties

move class C to a package 'ui.ide'
in the initializer of C, **add invocation** to method 'setActionId'

C extends eclipse.Action

Transformation Patterns

- Total of eleven patterns in real software systems

Transformation patterns	Number of operators	Pattern occurrences
Eclipse (first)	4	26
Eclipse (second)	1	(70)72
JHotDraw	5	9
MyWebMarket	5	7
PackageManager (first)	5	66
PackageManager (second)	9	19
PackageManager (third)	4	64
PackageManager (fourth)	2	7
PetitDelphi	2	(15)19
PetitSQL	4	6
VerveineJ	2	3

Transformation Patterns are frequent

Transformation Patterns

- In JHotDraw, some operators were not applied
- In other systems, the pattern was not applied at once

System	#Rev.	Date	Occurrences
Eclipse (second)	3.0	06/25/04, 12:08	70
	3.1	06/27/05, 14:35	71
	3.2	06/29/06, 19:05	72
	3.3	06/25/07, 15:00	72
	3.7	06/13/11, 17:36	72
PetitDelphi	210	11/19/14, 14:52	15
	211	11/19/14, 18:56	17
	212	11/26/14, 18:17	18
	213	12/03/14, 18:23	18
	214	12/22/14, 15:55	19

Transformation Patterns are complex

Conclusions

- Automation is needed
 - Perform the transformations correctly
 - Find transformation opportunities
- Generate custom transformations
 - Abstract
 - Replicable
 - System specific
- Submitted paper to ICSME (under review)

Introduction

Identifying Systematic Code Transformations

Replaying Systematic Code Transformations

Problem

- Transformation patterns exist
- Generate custom, abstract transformations
 - Replay in different code locations

Related Work

- Automated Code Transformation

	Application	Destination of Changes
Sydit [Men2011]	Bug fixes	methods only
Lase [Men2013]	Bug fixes	methods only
Critics [Zha2015]	Bug fixes	customizable

Solution

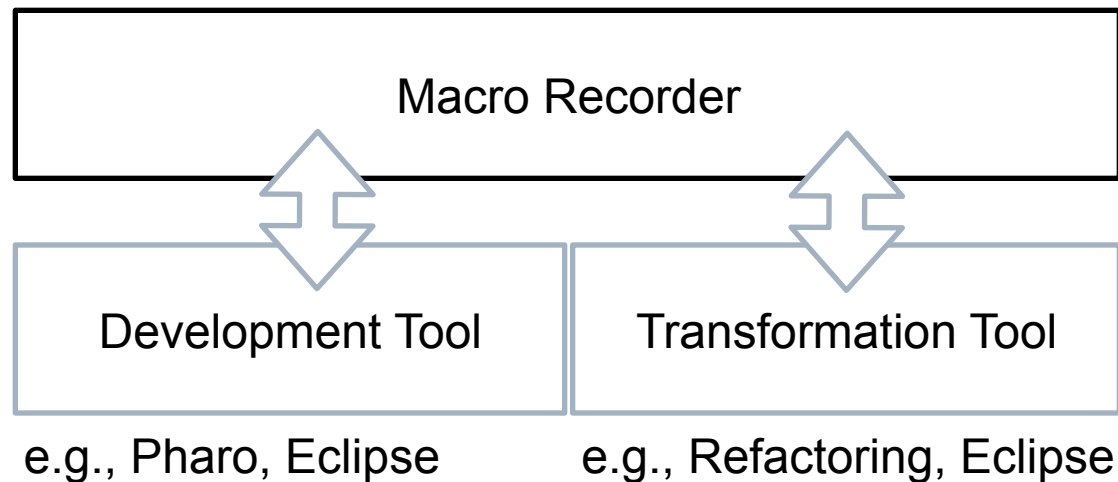
move class StoreAction to a package 'ui.ide'
in StoreAction(), **add invocation** to 'setActionId'

- What if the developer could...
 - Perform the changes manually **once**
 - Generalize the performed changes
 - Replay the changes in other locations

execute it for all class C that **extends** eclipse.Action

Approach

- MacroRecorder
 - For each recorded event in the development tool, generate an equivalent transformation



Illustrating Example

remove method blockNode in class Parser
remove class BlockNode

- Enable recording
- Then perform the change manually
- Stop recording

Illustrating Example - Record

The screenshot shows a window titled "New Transformation Pattern (MRTransformationPattern)". It is divided into two main panes: "Operators List" on the left and "Variables List" on the right. Below these panes is a "Changed Code" section, and at the bottom is a "Name:" field and a "Save" button.

Operators List:

- PDASTDelphiParser » block 22:09
- PDASTBlockNode 22:09

Variables List:

- @method1 #block
- @class1 PDASTDelphiParser
- @className1 #PDASTBlockNode

Changed Code:

```
"protocol: #'as yet unclassified"  
  
block  
  ^ super block ==> [ :parsingItems | PDASTBlockNode new ]
```

Callouts indicate the following changes:

- performed changes:** Points to the "Operators List" pane.
- changed code:** Points to the "Changed Code" section.
- changed entities:** Points to the "Variables List" pane.

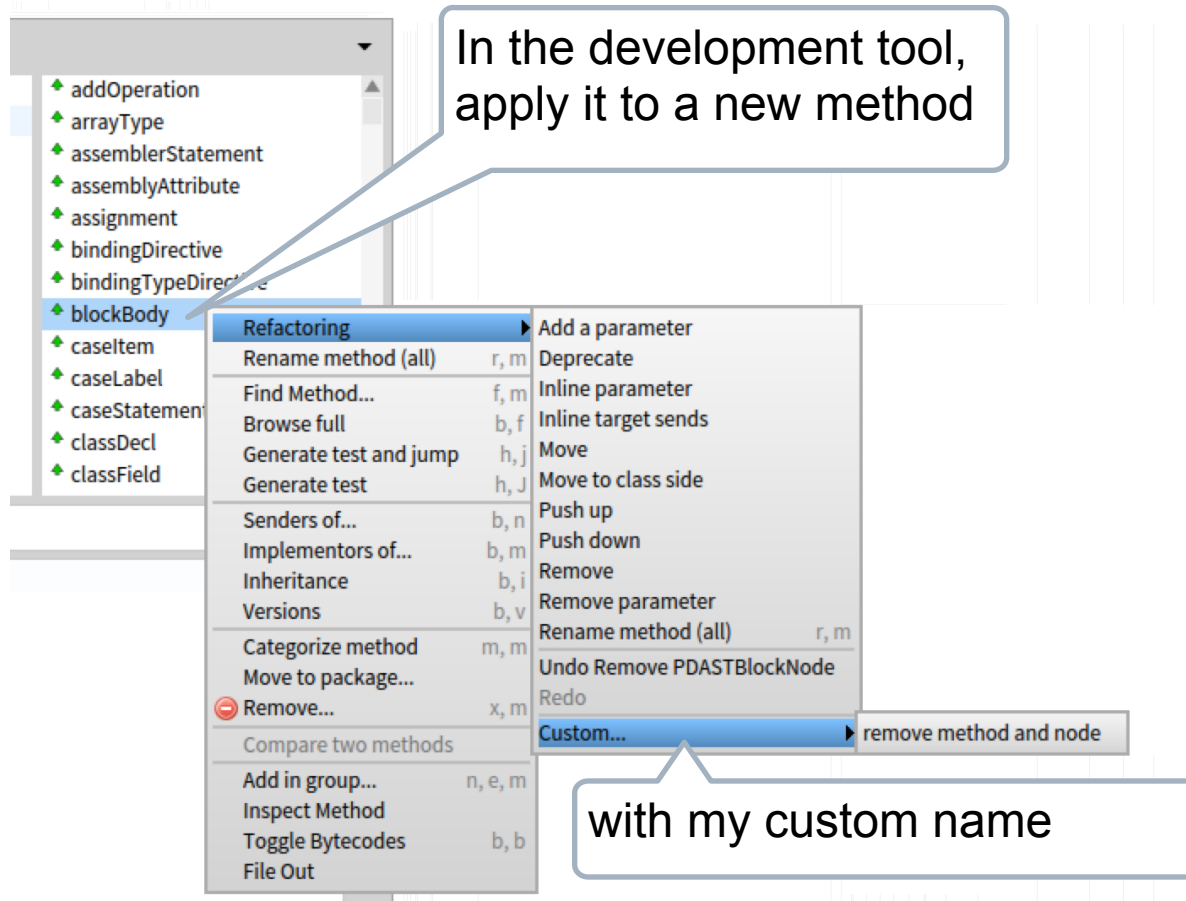
Illustrating Example - Generalize

The screenshot shows a software development environment with a 'New Transformation Pattern (MRTransformationPattern)' dialog box. The dialog has two panes: 'Operators List' and 'Variables List'. The 'Operators List' contains two entries: 'PDASTDelphiParser' with a sub-entry 'block' and a timestamp '22:09', and 'PDASTBlockNode' with a timestamp '22:09'. The 'Variables List' contains three entries: '@method1' with value '#block', '@class1' with value 'PDASTDelphiParser', and '@className1' with value '#PDASTBlockNode'. A context menu is open over the '@className1' entry, showing 'Add Value' and 'Inspect' (with 'Cmd + I' shortcut). A callout bubble points to the 'Add Value' option with the text 'add a new value'. Below the dialog, a 'Define new variable value' dialog is open, displaying a message: 'Please define the new value as a text expression (e.g., the name of a class/method). The value is only changed if the expression is valid'. The text input field contains the expression: `"PDAST" + @method1.name().asCamelCase() + "Node"`. The 'Define new variable value' dialog has 'OK' and 'Cancel' buttons. Below the main dialog, a 'Changed Code' pane shows the following code snippet:

```
block
^ super block ==> [ :parsingItems | PDASTBlockNode new ]
```

 At the bottom of the main dialog, there is a 'Name:' field and a 'Save' button. A callout bubble points to the 'Save' button with the text 'add a new name, then save'.

Illustrating Example - Replay



Illustrating Example - Replay

The screenshot shows a dialog window titled "remove method and node (MRTransformationPattern)". It contains two panes: "Operators List" and "Variables List".

- Operators List:** Contains two entries: "PDASTDelphiParser » block" (timestamp 22:09) and "PDASTBlockNode" (timestamp 22:09).
- Variables List:** Contains three entries: "@method1 #blockBody", "@class1 PDASTDelphiParser", and "@className1 \"PDASTDelphiParser\"".

A "Changes Browser" dialog is overlaid on top, showing a list of changes to be applied:

- Remove PDASTDelphiParser>>#blockBody
- Remove PDASTBlockBodyNode

Below the list, the text "Remove PDASTDelphiParser>>#blockBody" is displayed in green. At the bottom of the "Changes Browser" are "Ok" and "Cancel" buttons.

Callouts provide additional context:

- A callout points to the selected method in the "Variables List" with the text: "with the selected method".
- A callout points to the "Changes Browser" with the text: "Inspect the changes before applying, then execute".
- A callout points to the "Replay!" button at the bottom of the main dialog with the text: "Replay".

The main dialog also features a "Pretty print" checkbox and a "Name:" field containing "remove method and node".

Future Work

- Use MacroRecorder on the patterns we found before
- Check if the examples are correct

