MacroRecorder: Recording and Replaying Source Code Transformations

Gustavo SANTOS
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]
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

- Related to:
  - Bug Fixes [Ngu2010]
  - API Updates [Ray2012]
  - Improve the organization

[Ng2010] Recurring Bug Fixes in Object-Oriented Programs. ICSE 2010
Transformation Patterns

- Total of eleven patterns in real software systems

<table>
<thead>
<tr>
<th>Transformation patterns</th>
<th>Number of operators</th>
<th>Pattern occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse (first)</td>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>Eclipse (second)</td>
<td>1</td>
<td>(70)72</td>
</tr>
<tr>
<td>JHotDraw</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>MyWebMarket</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>PackageManager (first)</td>
<td>5</td>
<td>66</td>
</tr>
<tr>
<td>PackageManager (second)</td>
<td>9</td>
<td>19</td>
</tr>
<tr>
<td>PackageManager (third)</td>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>PackageManager (fourth)</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>PetitDelphi</td>
<td>2</td>
<td>(15)19</td>
</tr>
<tr>
<td>PetitSQL</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>VerveineJ</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Transformation Patterns

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

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

Transformation Patterns are complex.
Transformation Pattern

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

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

- Operators can be atomic or aggregated
Systematic Code Changes

- Performing these sequences manually can be:
  - Complex
  - Tedious
  - Error-prone

- Examples in real world systems [San2015]

- Automation is needed
Problem

- Generate customizable, composite, and abstract transformations

- Related Work in Automated Code Transformation

<table>
<thead>
<tr>
<th>Application</th>
<th>Destination of Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sydit [Men2011]</td>
<td>Bug fixes</td>
</tr>
<tr>
<td>Lase [Men2013]</td>
<td>Bug fixes</td>
</tr>
<tr>
<td>Critics [Zha2015]</td>
<td>Bug fixes</td>
</tr>
</tbody>
</table>
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

![Diagram]

- **Development Tool**
  - e.g., Pharo, Eclipse

- **Transformation Tool**
  - e.g., Refactoring, Eclipse
Illustrating Example

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

```java
remove method block() in class Parser
remove class BlockNode
```
Illustrating Example - Record

- performed changes
- changed code
- changed entities
Illustrating Example - Generalize

- Add a new value
- Add a new name, then save
Illustrating Example - Replay

In the development tool, apply it to a new method with my custom name.
Illustrating Example - Replay

Inspect the changes before applying, then execute with the selected method.
Status

- Prototype tool

- Proof-of-concept study on Smalltalk systems
  - covered 92% of new code locations
  - 76% of the resulting code is behavior-equivalent to the developer’s
  - 79% of the resulting code is similar to developer’s manual edition
  - Submitted paper on SCAM
Future Work

▪ Use MacroRecorder on the patterns we found before
  ▪ Submitted paper to SCAM

▪ Improve the automation
  ▪ Applying the changes to all entities in the system (matching a condition)
  ▪ Generalize the parameters automatically