Automatically Generating Refactorings to Support API Evolution

Jeff Perkins

MIT CSAIL
Outline

● Library evolution
  ○ Libraries evolve
  ○ Clients often don’t track library changes
  ○ Contributions
● Mechanism to automatically upgrade clients
● Non-behavior preserving changes
● Applicability
● Comparison with other approaches
● Conclusion
Libraries evolve

- APIs change
  - Refactorings
  - Bug fixes
  - New functionality
  - Design changes

- Deprecated methods, classes, fields, etc are retained for backwards compatibility
Clients often don’t track library changes

- Laziness
- Fear
- Problems result
  - Improvements are missed
  - Code may fail (if old methods are removed)
  - Libraries must maintain deprecated methods (or old versions of the library) indefinitely
Contributions

- Use information already in the library to upgrade the client
- Upgrade information is specified in code (precise, machine readable)
- Mechanism to test library improvements without changing client
- Analysis of applicability in two libraries
Outline

- Library evolution
- Mechanism to automatically upgrade clients
  - Upgrading Methods
  - Upgrading Classes
  - Upgrading fields
- Non-behavior preserving changes
- Applicability
- Comparison with other approaches
- Conclusion
Upgrading Methods

- Deprecated code normally includes documentation with a suggested change. For example:

  ```
  /** Use getSize() instead of size() **/
  ```

- This can be expressed more precisely in the body of the method:

  ```java
  @Deprecated public int size() { return getSize(); }
  ```

- A tool can update client code accordingly.
Upgrading Classes

- The deprecated class indicates its replacement by extending the new class

```java
class NewClass {
    public void m1() { ... }
    public void m2() { ... }
    ... 

    @Deprecated class OldClass extends NewClass {
    }
```

- The tool could replace uses of OldClass with NewClass
Upgrading fields

- A deprecated `static final` field’s replacement is the value of the deprecated field

  ```java
  class Old {
      static final int CURSOR = New.CURSOR;
      ...
  }
  ```

- The tool could replace instances of `Old.CURSOR` with `New.Cursor`

- Other fields don’t have as straightforward a substitution. Annotations could be used in these cases.
Outline

- Library evolution
- Mechanism to automatically upgrade clients
- Non-behavior preserving changes
  - Library evolution may result in semantic changes
  - Run time selection between implementations
  - Advantages of run time selection
- Applicability
- Comparison with other approaches
- Conclusion
Library evolution may result in semantic changes

- Reasons for semantic changes
  - More precise or accurate results may be returned.
  - New exceptions or errors may be thrown
  - Exceptions that were previously thrown may no longer be thrown

- Clients may rely on the old behavior

- Library developers will retain and deprecate the previous version for compatibility
Run time selection between implementations

- The deprecated method contains two implementations
  - Original implementation
  - A call to the replacement method

- For example:

```java
deprecated int old_method (Object x) { if (complete_backwards_compatibility) { // old code } else { return new_method (x); } }
```
Advantages of run time selection

- Code with semantic changes can be handled
- Preferred update to client code is precisely expressed
- Client can test changes without modifying their code
- Client can update only if testing indicates that the change is compatible for them.
Outline

- Library evolution
- Mechanism to automatically upgrade clients
- Non-behavior preserving changes
- Applicability
  - Library test cases
  - Results
- Comparison with other approaches
- Conclusion
Library test cases

- java.awt and Apache Byte Code Engineering Library (BCEL)
- Examined deprecated methods to determine the types of modifications
- Modifications that our approach supports
  - Rename method
  - Method arguments changed
  - Method semantics changed
  - Rename static final field
- Modifications that our approach would not support
  - Replace constructor with factory
  - Redesign required
### Results

<table>
<thead>
<tr>
<th>Change Type</th>
<th>AWT</th>
<th>BCEL</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rename method</td>
<td>73</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>Method arguments changed</td>
<td>9</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>Method semantics changed</td>
<td>1</td>
<td>4</td>
<td>Yes</td>
</tr>
<tr>
<td>Rename static final field</td>
<td>13</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>Replace constructor with factory</td>
<td>0</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>Redesign required</td>
<td>26</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>Total deprecated methods/fields</td>
<td>122</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Outline

- Library evolution
- Mechanism to automatically upgrade clients
- Non-behavior preserving changes
- Applicability
- Comparison with other approaches
  - Other approaches
  - Refactoring disadvantages
- Conclusion
Other approaches

• Chow and Notkin (1996) - annotate changed functions with update rules in the language of Sorcerer
• Borland (2004) - team refactoring tool
• Lund University (2004) - similar team refactoring tool for Eclipse
• Henkel and Diwan (2005) - capture refactorings to an XML file for later replay
Refactoring disadvantages

- Changes are limited to refactorings.
- A new language or file format is required
- An additional artifact must be shipped
- A special tool must be used by developers to record refactorings
Outline

- Library evolution
- Mechanism to automatically upgrade clients
- Non-behavior preserving changes
- Applicability
- Comparison with other approaches
- Conclusion
  - Conveying updates in code is a potentially useful approach
Conveying updates in code is a potentially useful approach

- Library developers explicitly and precisely indicate suggested replacements
- Replacement information is specified and edited in the original programming language
- Replacement information is encoded directly into the distributed code
- Clients can test changes before updating
- The use of particular development environments is not required
- 79% of examined cases are applicable
Questions