Lecture 15
Refactoring Reconstruction
Today’s Agenda

- Motivation for Refactoring Reconstruction
- Refactoring Reconstruction
- UMLDiff: some slides borrowed from Zhenchang Xing (U. Alberta)
Today's Agenda

- Synthesis of Refactoring Reconstruction Techniques
- API Evolution Support
- Bug Cache (MSR Part II)
Motivation for Reconstructing Refactorings from Two Versions
Motivation for Reconstructing Refactorings from Two Versions
Motivation for Reconstructing Refactorings from Two Versions

1. Detecting Possible Sources of Errors
   - Incomplete refactorings can be sources of errors
   - e.g. BarChart.draw() and PieChart.draw() override Chart.draw()
   - e.g. Chart.draw() and PieChart.draw() were renamed to Chart.paint() and PieChart.paint() but not BarChart.draw().
Motivation for Reconstructing Refactorings from Two Versions

2. Capturing Intent of Changes
   - Better empirical studies of code changes
   - Reduce # of conflicts in version merging
Motivation for Reconstructing Refactorings from Two Versions

3. Capturing and Replaying Changes
   - Automated update of client code: e.g. if a parameter was added to an API, then method invocations in program code using the API is automatically adapted.

4. Longer, continuous evolution history
   - eRose system: when identifying related changes, inferred renamings can be used to combine rules of the previous instance and rules of the new instance.
Motivation for Reconstructing Refactorings from Two Versions

5. Relation to Software Metrics

- Assess what kinds of refactorings increase what kinds of quality metrics

[Source: Identifying Refactorings from Source-Code Changes, Peter Weissgerber and Stephan Diehl ASE 2006]
Design Evolution Analysis
in support of
Evolutionary Software Development

Zhenchang Xing
University of Alberta

Supported by
public class Course {
    private MonitorableQueue waitingList = new MonitorableQueue();

    private Queue backupTAs = new Queue();

    public void addToWaitingList(Collection waitingStudents) {
        waitingList.offerMany(waitingStudents);
    }

    public void enrollFromWaitingList(int howmany) {
        List list = this.waitingList;
    }

    public void notifyBackupTAs() {
        for (Iterator iterator = backupTAs.listIterator(); iterator.hasNext();) {
        }
    }

    public void reportEnrolmentStatistics() {
        int historyHigh = waitingList.maxSize();
    }
}
What I Will Tell Him

Queue is a List

MonitorableQueue is a Queue

SimpleQueue contains a List

MonitorableQueue contains a Queue

Publisher
-subsribers : Queue
+update() : void

ArrayList
+iterator(): Iterator

Queue
+poll(): Object
+offer( Object): void
+offer( many : Object[]): void
+elementData : List
+size() : int

MonitorableQueue

HighWaterMark : int = 0
+resetHighMark(): void
+resetLowMark(): void
+ObservableQueue(): Observable

SimpleQueue

FastQueue

MonitorableQueue

Note: The diagram represents relationships between classes and interfaces, indicating how they are related through inheritance and implementation.
The Research Questions

• What exactly has been changed in the design context and how?

• Why has it been changed in the way it has?

• How can this information be used to support developers and in what tasks?
The Methodology

Extract

Model Differencing

- UMLDiff

Mining

- Change Pattern Detection
- Sequential Pattern Analysis
- Co-evolution Pattern Mining

Supporting

- Diff-CatchUp
- Design Mentor
Model differencing with UMLDiff

The 20th ACM/IEEE International Conference on Automated Software Engineering, 2005

What exactly has been changed and How?
Heuristics in UMLDiff

• Additions and removals are easy
• Renamings are difficult
  – Lexical similarity of names and comments:
    ▪ LCS, Adjacent pair
  – Structural similarity of relations
• Moves are even harder
  – The context from and to which elements are moved
    ▪ Relationships: inheritance, containment, usage
    ▪ Lexical and structural similarity of source and target contexts
  – The number of potential moves
• What if a set of elements are all renamed and/or moved?
  – Multiple rounds of renaming/move recognition
UMLDiff Process

- **Input**: Model\textsubscript{before} and Model\textsubscript{after}
- **UMLDiff** is a heuristic differencing algorithm
  1. Mapping model elements
     - Lexical and structural similarity
  2. Mapping relationships
     - The same relation type and the model elements they relate are mapped
  3. Recognizing extract/inline operations (not limited to class internals)
     - Usage dependency changes
  4. Compare attributes of mapped model elements
- **Output**: A set of elementary design change facts
  - Additions, removals, matches, renamings, moves of model elements
  - Extract and inline operations
  - Changes to relationships (inheritance, association, usage)
  - Changes to attributes (visibility, deprecation-status, …)
Evaluation

• How did they create the ground truth?
  • Use a very low threshold 1% and manually inspect all of them
  • Changes identified by UMLDiff and the ones UMLDiff missed, which were manually added through their manual inspection using JDEvAn tool

• Precision
• Recall
Precision vs. Recall

What a tool finds

What it should find

False positives

Correct predictions

False negatives

precision = % of returned entities are relevant
recall = what % of relevant entities are returned
### How good is UMLDiff?

<table>
<thead>
<tr>
<th></th>
<th>HtmlUnit</th>
<th>JFreeChart</th>
<th>Eclipse JDT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Unit testing framework for web apps</td>
<td>Java library for drawing charts</td>
<td>IDE and Plugin-based framework</td>
</tr>
<tr>
<td>Major releases</td>
<td>11 (~4 years)</td>
<td>31 (~5 years)</td>
<td>6 (~3 years)</td>
</tr>
<tr>
<td>Average #Class</td>
<td>~200</td>
<td>~450</td>
<td>~4000</td>
</tr>
<tr>
<td>Renamings* (Precision)</td>
<td>97.2%</td>
<td>95.2%</td>
<td>93.8%</td>
</tr>
<tr>
<td>[Threshold 0.3] (Recall)</td>
<td>98.5%</td>
<td>96.4%</td>
<td>96.6%</td>
</tr>
<tr>
<td>Moves* (Precision)</td>
<td>99.5%</td>
<td>91.1%</td>
<td>84.8%</td>
</tr>
<tr>
<td>[Threshold 0.4] (Recall)</td>
<td>99.9%</td>
<td>97.1%</td>
<td>90.3%</td>
</tr>
</tbody>
</table>

*Results with heuristics: Name, Comment, Structure, Src/TrgContext, #PotentialMoves, TransitiveUsage, Round=3
JDEvAn in Eclipse
JDEvAn Viewer in Eclipse
## Synthesis of Refactoring Reconstruction Techniques

<table>
<thead>
<tr>
<th>Method</th>
<th>Program Element Characteristics</th>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin Analysis 2005</td>
<td>name similarity, code metrics, calls</td>
<td>two complete versions selected manually</td>
</tr>
<tr>
<td>UMLDiff 2005</td>
<td>name similarity, code relationships</td>
<td>two complete versions selected manually</td>
</tr>
<tr>
<td>M. Kim et al. 2007</td>
<td>name similarity</td>
<td>two complete versions selected manually</td>
</tr>
<tr>
<td>S. Kim et al. 2005</td>
<td>name similarity, code metrics, calls, textual similarity</td>
<td>two complete versions selected manually</td>
</tr>
<tr>
<td>Dig et al. 2006</td>
<td>syntactical similarity, code relationships</td>
<td>two complete versions selected manually</td>
</tr>
<tr>
<td>Weissgerber et al. 2006</td>
<td>structural and code clone differences</td>
<td>all change sets between two versions</td>
</tr>
<tr>
<td>SemDiff 2008</td>
<td>structural and outgoing call differences</td>
<td>all change sets between any versions</td>
</tr>
</tbody>
</table>

[Source: Recommending Adaptive Changes from Framework Evolution, Barthelemy Dagenais and Martin Robillard, ICSE 2008]
API-Evolution Support with Diff-CatchUp

IEEE Transactions on Software Engineering, 2007

How can this information be used to support developers and in what tasks?
**Diff-CatchUp Approach**

- Automatically recover the evolution of framework APIs
  - UMLDiff and change-pattern queries

- Suggest ways to migrate client applications
  - Refactored API
    - Present the refactorings that the API is involved in and its renaming/move counterparts in new version if any
  - Removed (deprecated, visibility-restricted, no-longer-inherited, and class-made-abstract) API
    - Locate “voluntary” migration examples
    - Recommend replacing APIs
Migrate to Refactored API

- **RenameMethod** (maxSize(), highWaterMark())
  
  Prob #1: The method maxSize() is undefined for the type MonitorableQueue.
  Reason: The method name changed.
  Solution: Update the method call with new name.

- **ChangeParamType** (offerMany(...), Collection, Object[])
  
  Prob #2: The method offerMany(Object[]) in the type MonitorableQueue is not applicable for the argument (Collection).
  Reason: Parameter type changed.
  Solution: Obtain Object[] from Collection (e.g. Collection.toArray()).
Migrate to Refactored API

- **RenameClass**(Queue, SimpleQueue)
- **ExtractInterface**(SimpleQueue, Queue)
- **AddAbstraction**(FastQueue, Queue)
- **AddAbstraction**(MonitorableQueue, Queue)

Prob #3: Cannot instantiate the type Queue
Reason: The Queue represents a newly introduced interface in the new version.
The original class Queue is renamed as SimpleQueue.
Solution: Create SimpleQueue object, or
See if the interface Queue's other implementation classes can be used as well.
Migrate to Refactored API

- **ReplaceInheritanceWithDelegation** (MonitorableQueue, SimpleQueue, internalQueue, Queue)
- **ReplaceInheritanceWithDelegation** (SimpleQueue, ArrayList, elementData, List)
- **ExtractInterface** (SimpleQueue, Queue)
- **AddAbstraction** (MonitorableQueue, Queue)

Prob #4: Type mismatch: cannot convert MonitorableQueue to List
Reason: MonitorableQueue is no longer SimpleQueue, which is no longer List
Solution: Stop using MonitorableQueue as List object
May use it as a Queue object
Migrate to “Removed” API

- ReplaceInheritanceWithDelegation(SimpleQueue, ArrayList, elementData, List)

Prob #6: The method listIterator() is undefined for the type Queue
Reason: The original Queue class used to be a List; it inherits listIterator() from its superclass ArrayList, but no longer doing so.

This is essentially a “removed” API. How am I going to replace it?
Diff-CatchUp in Eclipse

![Image of Eclipse IDE window showing Diff-CatchUp feature]

The image shows a screenshot of the Eclipse IDE with the Diff-CatchUp feature enabled. The screenshot includes code snippets and API documentation related to the Eclipse IDE and its debugging tools.
### How Good is Diff-CatchUp?

<table>
<thead>
<tr>
<th>Type of problem</th>
<th>#broken API</th>
<th>#success proposal</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JFreechart</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ImportNotFound</td>
<td>17</td>
<td>17</td>
<td>100</td>
</tr>
<tr>
<td>UndefinedType+ImportNotFound+UndefinedName</td>
<td>254</td>
<td>247</td>
<td>97.2</td>
</tr>
<tr>
<td>InvalidClassInstantiation</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>UndefinedMethod/Constructor</td>
<td>180</td>
<td>151</td>
<td>83.9</td>
</tr>
<tr>
<td>ParameterMismatch</td>
<td>54</td>
<td>54</td>
<td>98.1</td>
</tr>
<tr>
<td>UndefinedField+UndefinedName</td>
<td>33</td>
<td>29</td>
<td>87.9</td>
</tr>
<tr>
<td>UsingDeprecatedType</td>
<td>3</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>UsingDeprecatedMethod/Constructor</td>
<td>35</td>
<td>34</td>
<td>97.1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>577</strong></td>
<td><strong>535</strong></td>
<td><strong>92.7</strong></td>
</tr>
<tr>
<td></td>
<td>HTMLUnit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UndefinedType</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>UndefinedMethod/Constructor</td>
<td>11</td>
<td>9</td>
<td>81.8</td>
</tr>
<tr>
<td>ParameterMismatch</td>
<td>3</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>UsingDeprecatedType</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UsingDeprecatedMethod/Constructor</td>
<td>10</td>
<td>7</td>
<td>70.32</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>26</strong></td>
<td><strong>20</strong></td>
<td><strong>76.9</strong></td>
</tr>
</tbody>
</table>

**Evaluation**

- **JFreechart**
- **HTMLUnit**
My thought on Refactoring Reconstruction Research

- Promising ways to allow programmers to understand code changes at a high level
- Still long ways to go to automatically reconstruct design intent from source code
- It can be applied to mining software repository research.
- This is a challenging problem:
  - heuristics-based, often requiring many similarity thresholds
  - hard to evaluate this type of work in general.
Preview for Monday after Spring Break

• First of all--- have a fun & productive spring break!

• Crosscutting Concerns
  • Why some code changes are crosscutting?
  • Read Visitor Pattern from Design Patterns book--- We may have a quiz on crosscutting concerns (using the visitor pattern code example) on Monday.