Representation, Analysis, and Refactoring Techniques to Support Code Clone Maintenance

Dissertation Research Defense

Robert Tairas
tairasr@cis.uab.edu
http://www.cis.uab.edu/tairasr

June 15, 2010

Committee:
Dr. Barrett Bryant (Chair)
Dr. Jeff Gray
Dr. Nicholas Kraft
Dr. Marjan Mernik
Dr. Brian Toone
Dr. Chengcui Zhang

This research is supported by NSF grant CPA-0702764
Overview of Presentation

Introduction

Motivation

Research Objectives

Approach

Evaluation & Case Studies

Clone View

Level

Scattered

Clones

Clone Group

Representations

Evaluate Clone

Properties

Code

Clones

Clone Research

Size of

Clone Data

Process

Disconnect

Unify Process

Clone Visualizer

Localized

Representation

CoCloRep

Clone Group

Relationships

Sub-clone

Refactoring

CeDAR

Refactoring
Cloning in Software

- **Code Clones:**
  - A section of code that is duplicated in multiple locations in a program

- **Different granularity levels:**
  - Statements, Block, Method, Class, Program

- **Clone Group:**
  - Clones of the same duplication
Types of Clones

Original code

```c
int main() {
    int x = 1;
    int y = x + 5;
    return y;
}
```

- Exact match (Type I)

- Exact match with differing (parameterized) identifier names (Type II)

- Near exact match (Type III)

---

Bellon et al., 2007
Reason for the Existence of Clones

- A section of code is copied and pasted into another part of the same program
  - Code performs some functionality correctly and copy-and-paste is relatively easy

- Simion† (Similar code fragments)
  - Behaviorally similar
  - Origins not from a common code fragment

†Juergens et al., 2010
Clones in Software Maintenance

- **Clone maintenance:**
  - Fix an error, enhance the functionality, or to improve the structure and/or performance
  - Software maintenance consumes up to 90% of software development effort

- **Clone comprehension:**
  - Knowledge of their existence, where the duplicates are located, and what kind of code is being duplicated
  - Program comprehension consumes at least 50% of maintenance cost

†Erlikh, 2000; ‡Standish, 1984
Clone Detection Techniques and Timeline

- **String**: Baker ‘92, Johnson ‘93, Davey ‘95, Ducasse ‘99
- **Token**: Kamiya ‘02, Li ’04
- **Tree**: Baxter ‘98, Evans ‘05, Jiang ’07, Kraft ‘08
- **Program Dependence Graph**: Krinke ’01, Gabel ‘08
- **Assembler**: Davis, ‘10
- **Metrics**: Mayrand ‘96, Kontogiannis ‘97
Clone Research

Detection

Representation
- Textual
- Visual
- Intermediate

Analysis
- Evolution Properties
- Bug Detection
- Structural / Semantic
- Refactoring

Maintenance
- In-Place
- Duplication Removal
Overview of Presentation

Introduction

Motivation

Clone View Level
Scattered Clones

Size of Clone Data
Process Disconnect
Representation Challenge: Evaluating Clone Groups

- Current representations and visualizations generally provide a system-level view

- Clones can be scattered in multiple source files

<table>
<thead>
<tr>
<th></th>
<th>Same file</th>
<th>Same directory</th>
<th>2nd Cousin</th>
<th>3rd Cousin and more</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone pair distribution in Apache(^\S)</td>
<td>912</td>
<td>135</td>
<td>840</td>
<td>641</td>
<td>2528</td>
</tr>
</tbody>
</table>

\[^\d\S\#\%\*\+\-\=\_\^\{\}\|\}\}\]CCFlnder, 2010; \[^\d\S\#\%\*\+\-\=\_\^\{\}\|\}\}\]Jiang and Hassan, 2007; \[^\d\S\#\%\*\+\-\=\_\^\{\}\|\}\}\]Rieger et al., 2004; \[^\d\S\#\%\*\+\-\=\_\^\{\}\|\}\}\]Kapser and Godfrey, 2005
Simian Output

<table>
<thead>
<tr>
<th>Source File</th>
<th>Starting Line</th>
<th>Ending Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:...\CMPFieldMetaData.java</td>
<td>134-145</td>
<td></td>
</tr>
<tr>
<td>C:...\CMPFieldMetaData.java</td>
<td>117-129</td>
<td></td>
</tr>
<tr>
<td>C:...\UsersRolesLoginModuleTest.java</td>
<td>64-68</td>
<td></td>
</tr>
<tr>
<td>C:...\LoginModulesTest.java</td>
<td>312-316</td>
<td></td>
</tr>
<tr>
<td>C:...\ServerDataCollector.java</td>
<td>230-265</td>
<td></td>
</tr>
<tr>
<td>C:...\Scheduler.java</td>
<td>552-587</td>
<td></td>
</tr>
<tr>
<td>C:...\EJBVerifier11.java</td>
<td>448-480</td>
<td></td>
</tr>
<tr>
<td>C:...\EJBVerifier11.java</td>
<td>1073-1109</td>
<td></td>
</tr>
<tr>
<td>C:...\EJBVerifier11.java</td>
<td>1297-1337</td>
<td></td>
</tr>
</tbody>
</table>

SimScan Output

Clone Group

Clone Group

-found 6 duplicate lines in the following files:
Between lines 201 and 207 in /.../WritableRaster.java
Between lines 1305 and 1311 in /.../Raster.java

-found 6 duplicate lines in the following files:
Between lines 920 and 926 in /.../JFIFMarkerSegment.java
Between lines 908 and 914 in /.../JFIFMarkerSegment.java

...
Analysis Challenge: Large Amounts of Data

- Clone coverage in software of various sizes and languages reported by various clone detection tools
- Detection results can yield large amounts of data

Clone coverage percentages in different programs

<table>
<thead>
<tr>
<th>Program</th>
<th>LoC</th>
<th>% of Clones</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux Kernel</td>
<td>4,365K†</td>
<td>15%</td>
</tr>
<tr>
<td>JDK 1.4.2</td>
<td>2,418K‡</td>
<td>8%</td>
</tr>
<tr>
<td>JDK 1.3.0</td>
<td>570K¤</td>
<td>9%</td>
</tr>
<tr>
<td>Process-Control System</td>
<td>400K§</td>
<td>12%</td>
</tr>
<tr>
<td>JHotDraw 7.0.7</td>
<td>71K¥</td>
<td>19%</td>
</tr>
<tr>
<td>JavaGenes 0.7.68</td>
<td>45K¥</td>
<td>10%</td>
</tr>
</tbody>
</table>

†Li et al., 2004; ‡Jiang et al., 2007; §Kamiya et al., 2002; B Baxter et al., 1998; CloneDR, 2010
Maintaining Clones

After a period of time

Activity | Class Containing Clones | Correction Date
--- | --- | ---
New statement insertion | ClassDiagramModel | March 2002
 | DeploymentDiagramModel | August 2002
Bug fix | SelectionComponentInstance | October 2002
 | SelectionComponent | February 2003

Updates of clones in ArgoUML†

†Aversano et al., 2007
Removing Clones through Refactoring

- Modularizing the code represented by clones through appropriate abstractions may improve code quality
  - Less duplicated code to maintain
  - Ease of future maintenance efforts

- **Refactoring** is one means of improving the quality of code
  - The goal of refactoring is to preserve the external behavior of code while improving its internal structure†

†Fowler, 1999
Refactoring Challenge: Process Disconnect

- Techniques such as ARIES\(^\dagger\) and SUPREMO\(^\ddagger\) can assist in determining clones that can potentially be refactored.
- However, the task of refactoring clones is delegated to the programmer.
- The programmer must either manually refactor the clones or forward the information about the clones to a refactoring engine.

\(^\dagger\)Higo et al., 2004; \(^\ddagger\)Koni-N'Sapu, 2001
Summary of Challenges

- Representation
  - System-level Views / Scattered Clones

- Analysis
  - Large Amounts of Data

- Refactoring
  - Process Disconnect
Overview of Presentation

- Introduction
- Motivation
- Research Objectives

- Clone Group Representations
- Evaluate Clone Properties
- Unify Process
Research Scope

Detection

Representation
- Textual
- Visual
- Intermediate

Analysis
- Evolution Properties
- Bug Detection
- Structural / Semantic
- Refactoring

Maintenance
- In-Place
- Duplication Removal
Research Scope

We focus on supporting two aspects related to the maintenance of code clones:
1) clone comprehension through its representation and analysis
2) clone maintenance with a focus on the removal of the duplication associated with the clones
Research Objectives
Research Objectives

Detection

Representation
- Localized Visualization
- MDE-based DSL

Analysis
- IR-based Relationships
- Historical Refactorings

Maintenance
- Unified Process
- Refactoring Engine Extensions
Research Objectives: Representation

- Contribute novel visualizations of clone groups
- Investigate the utilization of Model-Driven Engineering (MDE) techniques to represent and analyze clone groups

### Representation

- Localized Visualization
- MDE-based DSL

### Analysis

- IR-based Relationships
- Historical Refactorings

### Maintenance

- Unified Process
- Refactoring Engine Extensions
Research Objectives: Analysis

- Discover relationships of clone groups using an Information Retrieval (IR) technique
- Observe relationships of clones and actual historic refactorings

### Representation
- Localized Visualization
- MDE-based DSL

### Analysis
- IR-based Relationships
- Historical Refactorings

### Maintenance
- Unified Process
- Refactoring Engine Extensions
Research Objectives: Refactoring

- Extend the capabilities of an IDE to unify the phases of clone detection, analysis, and refactoring
Research Objectives

- Detection
- Representation
  - Localized Visualization
  - MDE-based DSL
- Analysis
  - IR-based Relationships
  - Historical Refactorings
- Maintenance
  - Unified Process
  - Refactoring Engine Extensions
Overview of Presentation

Introduction

Motivation

Research Objectives

Approach
  Evaluation & Case Studies

Representation
- Clone Visualizer
- Localized Representation
- CoCloRep

Analysis
- Clone Group Relationships
- Sub-clone Refactoring

Refactoring
- CeDAR
Clone Group Representations

Clone group representations

Representation
- Localized Visualization
- MDE-based DSL

Analysis
- IR-based Relationships
- Historical Refactorings

Maintenance
- Unified Process
- Refactoring Engine Extensions
CloViz: Visualization of Clone Detection Results

- Provide an alternative method of viewing clone detection results from the widely used scatter plot
- Extended from the AspectJ Development Tools Visualiser plug-in

Visualization view in CloViz

Logging concern in Tomcat†

†Hilsdale and Kersten, 2004
Comparison with Scatter Plot
Comparison with Scatter Plot

Clone group representation

Extraneous visualization

Availability Timeline...
Visualizer Utilization

- Visualization technique included in clone detection plug-in developed at Technische Universität München
- Part of ConQAT (Continuous Quality Assessment Toolkit)
Refactoring activity requires multiple modal dialog boxes

- Separation between program editing and refactoring tasks

- A solution: visualize refactoring changes directly in the source editor

Screen shot of Refactor! Pro†

†Refactor! Pro, 2010
Localized Clone Representation

- Represent a clone group in a localized manner
- Parameterized differences visualized in representation
Displaying Clones in a Localized Manner

- Localized representation is displayed after a user selects a clone group

- Determine differences among the clones
  - Differences based on first-level statement comparisons
Detecting Parameterized Elements

- A *suffix tree* is generated on the AST nodes representing the statements of a group of clones.
- Elements in nodes containing *allowed* differences are mapped together.

Parameterized elements mapped: 

```
file.getAbsolutePath()  dir.getAbsolutePath()
```

Excerpt of suffix tree:

- Clone 1: 
  - Stmt1
  - Stmt2
  - `file.getAbsolutePath()`
- Clone 2: 
  - Stmt1
  - Stmt2
  - `dir.getAbsolutePath()`

Parameterized elements mapped: 

```
file → dir
```
Statement Similarity Levels

- Comparing two statements of two clones
  - Level 1: Corresponding nodes are identical and match each other exactly
  - Level 2: Corresponding nodes are identical, but can contain allowed parameterized differences
    - MethodInvocation, NumberLiteral, QualifiedName, SimpleName, and StringLiteral
  - Level 3: Corresponding nodes are *not* identical, but both are correspond to types from the Level 2 comparison
Example Representations

- Exact statements, statements with parameterized differences, and non-matching statements

```java
String classname = (String) e.nextElement();
String location
    = classname.replace('.', File.separatorChar) + ".class";
File classFile = new File(config.srcDir, location);
if (classFile.exists()) {
    checkEntries.put(location, classFile);
    log("dependent class: " + classname + " - " + classFile,
         Project.MSG_VERBOSE);
}
```

```java
String classname = (String) e.nextElement();
String filename = classname.replace('.', File.separatorChar);
filename = filename + ".class";
File depFile = new File(basedir, filename);
if (depFile.exists() && parentSet.containsKey(filename)) {
    // This is included
    included.addElement(filename);
}
```
Example Representations

- **Sub-groups of clones**
- **Tighter similarities: Clones 1 and 4 vs. Clones 2 and 3**

```java
for (int i = 0; i < params.length; i++) {
    if (CONTAINS_KEY.equals(params[i].getType())) {
        contains.addElement(params[i].getValue());
    }
}
```

Clone 1

```java
for (int i = 0; i < params.length; i++) {
    if (COMMENTS_KEY.equals(params[i].getType())) {
        comments.addElement(params[i].getValue());
    }
}
```

Clone 4

```java
for (int i = 0; i < params.length; i++) {
    if (LINE_BREAKS_KEY.equals(params[i].getName())) {
        userDefinedLineBreaks = params[i].getValue();
        break;
    }
}
```

Clone 2

```java
for (int i = 0; i < params.length; i++) {
    if (PREFIX_KEY.equals(params[i].getName())) {
        prefix = params[i].getValue();
        break;
    }
}
```

Clone 3
Clone Properties Based on Visualizations

Quick summary of neighboring clones

Clones with small differences

Identifying clone with more difference
Evaluation: Fully Representing Clones

- Considers the number of clone groups (i.e., #CG) that can be appropriately represented
- Evaluated on multiple open source Java projects

<table>
<thead>
<tr>
<th>Project</th>
<th>#CG</th>
<th>Exact (%)</th>
<th>Param (%)</th>
<th>StmtDiff (%)</th>
<th>Mixed (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Ant 1.6.5</td>
<td>429</td>
<td>61 (14%)</td>
<td>152 (35%)</td>
<td>131 (31%)</td>
<td>85 (20%)</td>
</tr>
<tr>
<td>ArgoUML 0.26</td>
<td>650</td>
<td>61 (9%)</td>
<td>214 (33%)</td>
<td>124 (19%)</td>
<td>251 (39%)</td>
</tr>
<tr>
<td>Jakarta-JMeter 2.3.2</td>
<td>377</td>
<td>77 (20%)</td>
<td>158 (42%)</td>
<td>71 (19%)</td>
<td>71 (19%)</td>
</tr>
<tr>
<td>JBoss AOP 2.1.5</td>
<td>159</td>
<td>51 (32%)</td>
<td>81 (51%)</td>
<td>14 (9%)</td>
<td>13 (8%)</td>
</tr>
<tr>
<td>JFreeChart 1.0.10</td>
<td>847</td>
<td>151 (18%)</td>
<td>415 (49%)</td>
<td>168 (20%)</td>
<td>113 (13%)</td>
</tr>
<tr>
<td>JRuby 1.4.0</td>
<td>318</td>
<td>113 (36%)</td>
<td>70 (22%)</td>
<td>63 (20%)</td>
<td>72 (23%)</td>
</tr>
<tr>
<td>EMF 2.4.1</td>
<td>285</td>
<td>54 (19%)</td>
<td>136 (48%)</td>
<td>52 (18%)</td>
<td>42 (15%)</td>
</tr>
<tr>
<td>JEdit 4.2</td>
<td>345</td>
<td>91 (26%)</td>
<td>120 (35%)</td>
<td>88 (26%)</td>
<td>46 (13%)</td>
</tr>
<tr>
<td>Squirrel-SQL 3.0.3</td>
<td>428</td>
<td>78 (18%)</td>
<td>164 (38%)</td>
<td>70 (16%)</td>
<td>116 (27%)</td>
</tr>
</tbody>
</table>
Evaluation: Fully Representing Clones

- “Exact” → Clones that match each other exactly

<table>
<thead>
<tr>
<th>Project</th>
<th>#CG</th>
<th>Exact (%)</th>
<th>Param (%)</th>
<th>StmtDiff (%)</th>
<th>Mixed (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Ant 1.6.5</td>
<td>429</td>
<td>61 (14%)</td>
<td>152 (35%)</td>
<td>131 (31%)</td>
<td>85 (20%)</td>
</tr>
<tr>
<td>ArgoUML 0.26</td>
<td>650</td>
<td>61 (9%)</td>
<td>214 (33%)</td>
<td>124 (19%)</td>
<td>251 (39%)</td>
</tr>
<tr>
<td>Jakarta-JMeter 2.3.2</td>
<td>377</td>
<td>77 (20%)</td>
<td>158 (42%)</td>
<td>71 (19%)</td>
<td>71 (19%)</td>
</tr>
<tr>
<td>JBoss AOP 2.1.5</td>
<td>159</td>
<td>51 (32%)</td>
<td>81 (51%)</td>
<td>14 (9%)</td>
<td>13 (8%)</td>
</tr>
<tr>
<td>JFreeChart 1.0.10</td>
<td>847</td>
<td>151 (18%)</td>
<td>415 (49%)</td>
<td>168 (20%)</td>
<td>113 (13%)</td>
</tr>
<tr>
<td>JRuby 1.4.0</td>
<td>318</td>
<td>113 (36%)</td>
<td>70 (22%)</td>
<td>63 (20%)</td>
<td>72 (23%)</td>
</tr>
<tr>
<td>EMF 2.4.1</td>
<td>285</td>
<td>54 (19%)</td>
<td>136 (48%)</td>
<td>52 (18%)</td>
<td>42 (15%)</td>
</tr>
<tr>
<td>JEdit 4.2</td>
<td>345</td>
<td>91 (26%)</td>
<td>120 (35%)</td>
<td>88 (26%)</td>
<td>46 (13%)</td>
</tr>
<tr>
<td>Squirrel-SQL 3.0.3</td>
<td>428</td>
<td>78 (18%)</td>
<td>164 (38%)</td>
<td>70 (16%)</td>
<td>116 (27%)</td>
</tr>
</tbody>
</table>
Evaluation: Fully Representing Clones

- “Param” ➔ Clone groups with parameterized differences
- Majority of the cases except ArgoUML and JRuby
- Four cases almost half of the instances

<table>
<thead>
<tr>
<th>Project</th>
<th>#CG</th>
<th>Exact (%)</th>
<th>Param (%)</th>
<th>StmtDiff (%)</th>
<th>Mixed (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Ant 1.6.5</td>
<td>429</td>
<td>61 (14%)</td>
<td>152 (35%)</td>
<td>131 (31%)</td>
<td>85 (20%)</td>
</tr>
<tr>
<td>ArgoUML 0.26</td>
<td>650</td>
<td>61 (9%)</td>
<td>214 (33%)</td>
<td>124 (19%)</td>
<td>251 (39%)</td>
</tr>
<tr>
<td>Jakarta-JMeter 2.3.2</td>
<td>377</td>
<td>77 (20%)</td>
<td>158 (42%)</td>
<td>71 (19%)</td>
<td>71 (19%)</td>
</tr>
<tr>
<td>JBoss AOP 2.1.5</td>
<td>159</td>
<td>51 (32%)</td>
<td>81 (51%)</td>
<td>14 (9%)</td>
<td>13 (8%)</td>
</tr>
<tr>
<td>JFreeChart 1.0.10</td>
<td>847</td>
<td>151 (18%)</td>
<td>415 (49%)</td>
<td>168 (20%)</td>
<td>113 (13%)</td>
</tr>
<tr>
<td>JRuby 1.4.0</td>
<td>318</td>
<td>113 (36%)</td>
<td>70 (22%)</td>
<td>63 (20%)</td>
<td>72 (23%)</td>
</tr>
<tr>
<td>EMF 2.4.1</td>
<td>285</td>
<td>54 (19%)</td>
<td>136 (48%)</td>
<td>52 (18%)</td>
<td>42 (15%)</td>
</tr>
<tr>
<td>JEdit 4.2</td>
<td>345</td>
<td>91 (26%)</td>
<td>120 (35%)</td>
<td>88 (26%)</td>
<td>46 (13%)</td>
</tr>
<tr>
<td>Squirrel-SQL 3.0.3</td>
<td>428</td>
<td>78 (18%)</td>
<td>164 (38%)</td>
<td>70 (16%)</td>
<td>116 (27%)</td>
</tr>
</tbody>
</table>
Evaluation: Fully Representing Clones

- “StmtDiff” → Clone groups with statement differences
- “Mixed” → Clone groups containing both “Param” and “StmtDiff”

<table>
<thead>
<tr>
<th>Project</th>
<th>#CG</th>
<th>Exact (%)</th>
<th>Param (%)</th>
<th>StmtDiff (%)</th>
<th>Mixed (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Ant 1.6.5</td>
<td>429</td>
<td>61 (14%)</td>
<td>152 (35%)</td>
<td>131 (31%)</td>
<td>85 (20%)</td>
</tr>
<tr>
<td>ArgoUML 0.26</td>
<td>650</td>
<td>61 (9%)</td>
<td>214 (33%)</td>
<td>124 (19%)</td>
<td>251 (39%)</td>
</tr>
<tr>
<td>Jakarta-JMeter 2.3.2</td>
<td>377</td>
<td>77 (20%)</td>
<td>158 (42%)</td>
<td>71 (19%)</td>
<td>71 (19%)</td>
</tr>
<tr>
<td>JBoss AOP 2.1.5</td>
<td>159</td>
<td>51 (32%)</td>
<td>81 (51%)</td>
<td>14 (9%)</td>
<td>13 (8%)</td>
</tr>
<tr>
<td>JFreeChart 1.0.10</td>
<td>847</td>
<td>151 (18%)</td>
<td>415 (49%)</td>
<td>168 (20%)</td>
<td>113 (13%)</td>
</tr>
<tr>
<td>JRuby 1.4.0</td>
<td>318</td>
<td>113 (36%)</td>
<td>70 (22%)</td>
<td>63 (20%)</td>
<td>72 (23%)</td>
</tr>
<tr>
<td>EMF 2.4.1</td>
<td>285</td>
<td>54 (19%)</td>
<td>136 (48%)</td>
<td>52 (18%)</td>
<td>42 (15%)</td>
</tr>
<tr>
<td>JEdit 4.2</td>
<td>345</td>
<td>91 (26%)</td>
<td>120 (35%)</td>
<td>88 (26%)</td>
<td>46 (13%)</td>
</tr>
<tr>
<td>Squirrel-SQL 3.0.3</td>
<td>428</td>
<td>78 (18%)</td>
<td>164 (38%)</td>
<td>70 (16%)</td>
<td>116 (27%)</td>
</tr>
</tbody>
</table>
CoCloRep: Code Clone Representation

MDE-based clone representation and analysis

**Representation**
- Localized Visualization
- MDE-based DSL

**Analysis**
- IR-based Relationships
- Historical Refactorings

**Maintenance**
- Unified Process
- Refactoring Engine Extensions
CoCloRep: Code Clone Representation

- An investigation into the development of a Domain-Specific Language (DSL) for representing code clones
- Utilizing Model-Driven Engineering (MDE) in the context of clone analysis

MDE is concerned with raising the abstraction level of software development by utilizing models to specify the application
First DSL: Clone Representation

--- clone 1 ---
1: int g;
2: int f = g + 3;
3: i = i + 1;
4: c = f + m

--- clone 2 ---
1: int q;
2: int p = q + 3;
3: c = p + m

--- clone instances ---
1: instance r = cg(f, g) {
2: t {
3: i = i + 1;
4: }
5: };
6: }
7: instance s = cg(p, q);

--- clone group ---
1: clone cg($a, $b) {
2: int $b;
3: int $a = $b + 3;
4: {{ t }}
5: c = $a + m
6: }
Second DSL: Commands

- **Input**

  ```
  variables cg;
  ```

- **Output**

  1. Variable information for clone group cg
  2. Declared variables:
  3.   \( b \)
  4.   \( a \)
  5. Outside assigned variables:
  6.   \( c \)
  7.   \( i \) (in instance \( r \))
  8. Outside non-assigned variables:
  9.   \( m \)
Model Transformation Process

EBNF TS

M3

EBNF

Clones Grammar

Commands Grammar

EBNF TS

MDE Technical Space (TS)

KM3

Clones Metamodel

Commands Metamodel

Variables Metamodel

EBNF TS

M2

Clones on Clones

Commands on Clones

Code Clones

Variables Grammar

M1

Code Clones Model

Commands Model

Variables Model

Injection

Transformation

Extraction
Representation and Analysis in CoCloRep

- Representation of clones (as models)
  - Commonalities stored in clone groups
  - Variabilities stored in clone instances
  - Modified / combined AST of all clone instances

- Analysis of clones (via model transformations)
  - Transformations with both declarative and imperative constructs
  - Requires more complex transformations
    - Not one-to-one
Summary

- **Clone group representation**
  - Representations provide a low-level view of clones and a centralized location to view clone properties

- **Maintenance**
  - Visual representations provide a quick summary of clone properties
    - i.e., location of clones, complexity of clone differences
  - Preliminary investigation of using MDE for clone refactoring
Clone analysis using Information Retrieval

Clustering of code clones based on non-structural properties

Analysis
- IR-based Relationships
- Historical Refactorings

Representation
- Localized Visualization
- MDE-based DSL

Maintenance
- Unified Process
- Refactoring Engine Extensions
static void foo() throws RESyntaxException {
    String a[] = new String[] {"123.400","abc","orange 100"};
    org.apache.regexp.RE pat = new org.apache.regexp.RE("[0-9,]+" );
    int sum = 0;
    for(int i = 0; i < a.length; ++i)
    {
        if(pat.match(a[i]))
        {
            sum += Sample.parseNumber(pat.getParen(0));
        }
    }
    System.out.println("sum = " + sum);
}

static void goo(String[] a) throws RESyntaxException {
    RE exp = new RE("[0-9,]+" );
    for(int i = 0; i < a.length; ++i)
    {
        if(exp.match(a[i]))
        {
            sum += parseNumber(exp.getParen(0));
        }
    }
    System.out.println("sum = " + sum);
}
Clone analysis using Information Retrieval

- Investigate additional relationships among clone groups based on non-structured properties
- Latent Semantic Indexing (LSI) used to cluster clone groups based on the identifier names in the clones
Latent Semantic Indexing (LSI) can be used to provide relationships among terms and documents in a corpus. Document to Document relationships are determined based on terms in documents.
Approach: Clone Group Clustering

Clone Group 1

```java
static void foo() throws RESyntaxException {
    String a[] = new String[] { "123,400", "abc", "orange 100" };
    org.apache.regexp.RE pat = new org.apache.regexp.RE("[0-9,]+");
    int sum = 0;
    for (int i = 0; i < a.length; ++i)
        if (pat.match(a[i]))
            sum += Sample.parseNumber(pat.getParen(0));
    System.out.println("sum = " + sum);
}
```

Term-Document Matrix

|      | CG1 | CG2 | ...
|------|-----|-----|-----
| a    | 3   | X   |    |
| apache | 2  | X   |    |
| foo   | 1   | X   |    |
| getParen | 1  | X   |    |
| i    | 4   | X   |    |
|      | ... | ... | ...

Singular Value Decomposition (SVD)
Information Retrieval-based Process

- Case Study: Microsoft Research Kernel
  - Available for academic teaching and research
  - Basic operating system implementations for the NT Kernel

Diagram:
1. **Original Source Code** → **CCFinder** → **Clones in Clone Groups**
2. **Clones in Clone Groups** → **Filter Clone Groups** → **Filtered Clone Groups**
3. **Filtered Clone Groups** → **srcML** → **Source Code in XML Format**
4. **Source Code in XML Format** → **Generate Term-Document Matrix**
5. **Generate Term-Document Matrix** → **Term-Document Matrix** → **Matlab**
   - **Singular Value Decomposition**
6. **Singular Value Decomposition** → **Cluto** → **Clustered Clone Groups**
7. **Clustered Clone Groups** → **Generate Cluster Information** → **HTML Report**

**Latent Semantic Indexing**
Cluster Observations: Example

- Clones were grouped based on the method of assigning RequiredLength

```
RequiredLength = (ULONG)sizeof(TOKEN_STATISTICS);

while (Index < Token->RestrictedSidCount) {
    RequiredLength += SeLengthSid( Token->RestrictedSids[Index].Sid );
    Index += 1;
}
```
Cluster Observations: Example

- Clones were grouped based on statement sequences
  - Clone Group
  - [Array Initialization]
  - [Code Sequence 1]
  - [Code Sequence 2]
  - Clone
  - Clone Group
  - [Code Sequence 1]
  - [Array Initialization]
  - [Code Sequence 2]

- Clones grouped based on existence of a statement and arguments

```c
1: irp = IoAllocateIrp( deviceObject->StackSize, (...) );
2: if (!irp) {
3:    if (...) {
4:        ExFreePool( event );
5:    }
6:  IopAllocateIrpCleanup( fileObject, (...) );
7:  return STATUS_INSUFFICIENT_RESOURCES;
8: }
9: irp->Tail.Overlay.OriginalFileObject = fileObject;
10: irp->Tail.Overlay.Thread = CurrentThread;
```
Sub-Clone Refactoring

Observing actual refactorings associated with detected clones

**Analysis**
- IR-based Relationships
- Historical Refactorings

**Representation**
- Localized Visualization
- MDE-based DSL

**Maintenance**
- Unified Process
- Refactoring Engine Extensions
Refactoring Clones

public class A {
    public void method() {
        {cloned statements}
        {cloned statements}
        {cloned statements}
        ...
        {cloned statements}
        {cloned statements}
    }
}

Extract-Method Refactoring

public class A {
    public void method() {
        newMethod();
        ...
        newMethod();
    }
    public void newMethod() {
        {cloned statements}
        {cloned statements}
        {cloned statements}
    }
}

public class A {
    public void method() {
        newMethod();
        ...
        newMethod();
    }
    public void newMethod() {
        {cloned statements}
        {cloned statements}
        {cloned statements}
    }
}
Clone Refactoring Process

- Changes between two versions
  - First version contains original code
  - Second version contains refactored code
What are the refactoring characteristics of clones detected by a clone detection tool, if such a tool was used in the clone maintenance process?
Approach: Observing Refactoringings

- Observing actual clone-related refactorings in multiple release versions of JBoss

```java
@@ -2471,13 +2469,7 @@
     scan_position.current_slot = Page.INVALID_SLOT_NUMBER;

     // release the scan lock now that we have saved away the row
-    -    if (scan_position.current_scanpageno != 0)
-    -    {
-    -        this.getLockingPolicy().unlockScan(
-    -            scan_position.current_scanpageno);
-    -        scan_position.current_scanpageno = 0;
-    -    }
+    unlockCurrentScan(scan_position);
    }
```
Refactoring in Clone Ranges

```java
protected String getValue(String name, String value) {
    try {
        String propertyName = value.substring(2, value.length()-1);
        ObjectName propertyServiceON = new ObjectName("...");
        KernelAbstraction kernelAbstraction = KernelAbstractionFactory.getInstance();
        String propertyValue = (String)kernelAbstraction.invoke(...);
        log.debug("Replaced ejb-jar.xml element " + name + " with value " + propertyValue); return propertyValue;
    } catch (Exception e) {
        log.warn("Unable to look up property service for ejb-jar.xml element " + ...);
    }
    String replacement = StringPropertyReplacer.replaceProperties(value);
    if (replacement != null)
      value = replacement;
    return value;
}
```

```java
if (edge instanceof MTransition) {
    MTransition tr = (MTransition) edge;
    FigTrans trFig = new FigTrans(tr);
    // set source and dest
    // set any arrowheads, labels, or colors
    MStateVertex sourceSV = tr.getSource();
    MStateVertex destSV = tr.getTarget();
    FigNode sourceFN = (FigNode) lay...
    FigNode destFN = (FigNode) lay...
    trFig.setSourcePortFig(sourceFN);
    trFig.setSourceFigNode(sourceFN);
    trFig.setDestPortFig(destFN);
    trFig.setDestFigNode(destFN);
    + FigTrans trFig = new FigTrans(tr, lay);
    return trFig;
}
```

- Refactoring performed on only part of the reported clone range
- Sub-clone refactoring
Evaluation: Tool Coverage

- 21 *Extract Method*-type Refactoring in JBoss (v2.2.0–4.2.3)
  - Clones initially detected by Simian
  - Further evaluated with four other tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Exact Coverage</th>
<th>Larger Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. CCFinder</td>
<td>4 (19%)</td>
<td>8 (38%)</td>
</tr>
<tr>
<td>2. CloneDR</td>
<td>6 (28%)</td>
<td>9 (42%)</td>
</tr>
<tr>
<td>3. Deckard</td>
<td>8 (38%)</td>
<td>3 (14%)</td>
</tr>
<tr>
<td>4. Simian</td>
<td>2 (9%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>5. Simscan</td>
<td>6 (28%)</td>
<td>12 (57%)</td>
</tr>
</tbody>
</table>

- These tools mainly look for the maximal sized clone
Evaluation: Focus on Deckard

- Deckard selected due to tree-based tool performance
- JBoss re-evaluated
- Additional artifacts: ArgoUML (v0.10.1–0.26) and Apache Derby (v10.1.1.0–10.5.3.0)

<table>
<thead>
<tr>
<th>Property</th>
<th>JBoss</th>
<th>ArgoUML</th>
<th>Derby</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refactoring Coverage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exact coverage</td>
<td>19</td>
<td>17</td>
<td>12</td>
</tr>
<tr>
<td>Sub-clone coverage</td>
<td>14</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>Coverage Levels</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same level</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>1 level above</td>
<td>9</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>&gt; 1 level above</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Clone Differences</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Refactorable</td>
<td>7</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Not Refactorable</td>
<td>7</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
Evaluation: Focus on Deckard

- Reported clone range mainly the same level or one syntactic level above the actual refactored code
- Possibly to keep some logic in the original location

<table>
<thead>
<tr>
<th>Property</th>
<th>JBoss</th>
<th>ArgoUML</th>
<th>Derby</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refactoring Coverage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exact coverage</td>
<td>19</td>
<td>17</td>
<td>12</td>
</tr>
<tr>
<td>Sub-clone coverage</td>
<td>14</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>Coverage Levels</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same level</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>1 level above</td>
<td>9</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>&gt; 1 level above</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Clone Differences</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Refactorable</td>
<td>7</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Not Refactorable</td>
<td>7</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

- Programmers only refactored a sub-clone even when the entire clone was refactorable
Summary

- Analysis of large amounts of clone data
  - “Super-clones”
    - Clone group clustering based on non-structural information
  - “Sub-clones”
    - Refactoring performed on partial range of clones
- Maintenance
  - Clone groups that are related could be considered for similar updating
  - Support for sub-clone refactoring should be part of maintenance process
CeDAR: Clone Detection, Analysis, and Refactoring

Unifying the process of clone detection, analysis, and refactoring

**Representation**
- Localized Visualization
- MDE-based DSL

**Analysis**
- IR-based Relationships
- Historical Refactorings

**Maintenance**
- Unified Process
- Refactoring Engine Extensions
**Current Refactoring Process**

![Diagram of the refactoring process]

- **M1**: Clones must be detected manually
- **M2**: Clones must be analyzed manually
- **M3**: Each section of code must be manually selected and forwarded to Refactoring Engine

- **A1**: *Extract Method* refactoring limited to local variable name differences
  - Limited to clones in one file
  - Clone information only available after selection for refactoring

---

**Eclipse IDE**

- Original Source Code
- Detect Clones
- Code Clones
- Analyze Clones
- Clones to Refactor
- Select Code for Refactoring
- Code to be Refactored
- Refactoring Engine
- Internal Clone Detector
- Refactored Source Code
Current Approaches

Eclipse IDE

A1 Clones detected automatically

A2 Clones analyzed with automated assistance

M1 Each section of code must be **manually** selected and forwarded to Refactoring Engine
Our Approach: Unified Process

Eclipse IDE

A1 Automated clone detection remains an external process

A2 All clone information forwarded to refactoring engine

Additional parameterized differences such as fields, method calls, and string literals
Parameterized Element Mapping

- Include parameterized values of internal and external fields, method calls, and strings

---

CeDAR Plug-in in Eclipse

Original Source Code

Clone Detection Tool

Clone Group
- Clone 1
- Clone 2
- Clone 3

Clone Information Display

Selected Clone Group

Clone Refactoring

Refactored Source Code

Original Source Code

Clone Detection Tool

Clone Group
- Clone 1
- Clone 2
- Clone 3

Clone Information Display

Selected Clone Group

Clone Refactoring

Refactored Source Code

```java
if (bool1) {
    x = getVal1() + Class1.num1;
}
...

if (bool2) {
    x = getVal2() + Class2.num2;
}
...

if (bool3) {
    x = getVal3() + Class1.num1;
}
...
```
Type II Clones

- “syntactically identical copy; only variable, type, or function identifiers were changed.” [Bellon et al., 2007]

- Fields
  - Include fields that are different between at least two clones
  - Include clones with [field] ←→ [local variable] mappings

```java
public class A {
    int field1;
    int field2;

    public void method() {
        cloned statements
        reference to field1
        cloned statements
    }
    ...  
    cloned statements
    reference to field2
    cloned statements
}
```

```java
public class A {
    int field1;
    int field2;

    public void method() {
        newMethod(field1);
        ...
        newMethod(field2);
    }

    public void newMethod(int field) {
        cloned statements
        reference to field
        cloned statements
    }
}
```
Type II Clones

- **Method calls**
  - Include methods with no arguments
  - Pass method-related expressions if all clones use same expression

- **Strings**
  - Include strings with 1-to-1 correspondence

```java
public void method() {
    ...
    {reference to p}
    {reference to p.call()}
    ...
    {reference to q}
    {reference to q.call()}
    ...
}

public void newMethod(Object a, Object b) {
    {reference to a}
    {reference to b}
}

public void method() {
    ...
    newMethod(p, p.call())
    ...
    newMethod(q, q.call())
    ...
}

public void method() {
    ...
    newMethod(p)
    ...
    newMethod(q)
    ...
}
```
Evaluation: Additional Refactorings

- In half of the software artifacts evaluated, the number of refactoring doubled

<table>
<thead>
<tr>
<th>Project</th>
<th>KLoC</th>
<th>CG</th>
<th>Eclipse</th>
<th>CeDAR</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Ant 1.7.0</td>
<td>67</td>
<td>120</td>
<td>14 (12%)</td>
<td>28 (23%)</td>
<td>+14</td>
</tr>
<tr>
<td>Columba 1.4</td>
<td>75</td>
<td>88</td>
<td>13 (15%)</td>
<td>30 (34%)</td>
<td>+17</td>
</tr>
<tr>
<td>EMF 2.4.1</td>
<td>118</td>
<td>149</td>
<td>8 (5%)</td>
<td>14 (9%)</td>
<td>+6</td>
</tr>
<tr>
<td>Hibernate 3.3.2</td>
<td>209</td>
<td>177</td>
<td>15 (8%)</td>
<td>18 (10%)</td>
<td>+3</td>
</tr>
<tr>
<td>Jakarta JMeter 2.3.2</td>
<td>54</td>
<td>68</td>
<td>3 (4%)</td>
<td>11 (16%)</td>
<td>+8</td>
</tr>
<tr>
<td>JEdit 4.2</td>
<td>51</td>
<td>157</td>
<td>15 (10%)</td>
<td>20 (13%)</td>
<td>+5</td>
</tr>
<tr>
<td>JFreeChart 1.0.10</td>
<td>76</td>
<td>291</td>
<td>29 (10%)</td>
<td>62 (21%)</td>
<td>+33</td>
</tr>
<tr>
<td>JRuby 1.4.0</td>
<td>101</td>
<td>81</td>
<td>23 (28%)</td>
<td>23 (28%)</td>
<td>0</td>
</tr>
<tr>
<td>Squirrel SQL 3.0.3</td>
<td>141</td>
<td>75</td>
<td>8 (11%)</td>
<td>20 (27%)</td>
<td>+12</td>
</tr>
</tbody>
</table>
Parameterized Differences

- Each parameterized difference utilized during *Extract Method* refactoring activity, albeit in varying occurrences

<table>
<thead>
<tr>
<th>Project</th>
<th>Local Variable</th>
<th>Internal Field</th>
<th>External Field</th>
<th>Method Call</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Ant 1.7.0</td>
<td>10</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Columba 1.4</td>
<td>14</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>EMF 2.4.1</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Hibernate 3.3.2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Jakarta JMeter 2.3.2</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>JEdit 4.2</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>JFreeChart 1.0.10</td>
<td>34</td>
<td>19</td>
<td>11</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>Squirrel SQL 3.0.3</td>
<td>12</td>
<td>6</td>
<td>3</td>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>
CeDAR in Eclipse

Parsing clone detection reports
Localized representation
CeDAR in Eclipse

Clone location visualization
CeDAR in Eclipse

Sub-clones
CeDAR in Eclipse

Centralized maintenance
Summary

- Clone maintenance process (detection, analysis, and refactoring) unified within Eclipse through CeDAR
- Extensions incorporate more parameterized differences among clones to enable additional accepted refactorings
- Instances of clone refactoring doubled in many of the evaluated software artifacts
Contributions

- **Representation**
  - Visualization and representation of clones at the clone group level and a transformation-based clone analysis approach

- **Analysis**
  - The discovery of additional clone properties related to the semantic relationships of clone groups, and refactoring of partial clones

- **Refactoring**
  - A unified clone maintenance process that reduces the manual steps required for refactoring and increases support for refactoring of different clone types
Future Research Plan

- **Continued Focus on Clone Maintenance**
  - Increasing refactoring capabilities
  - Incorporating visualizations in the refactoring task
  - Clone models via model weaving

- **Broader Application of Work**
  - Additional clone property analysis (e.g., outlier clones)
  - Information retrieval and model analysis
Publications

Journals

Conferences and Workshops
R. Tairas, F. Jacob, J. Gray, Visualizing Code Clones in a Localized Manner, ACM Symposium on Software Visualization, Salt Lake City, UT, 10/10, under review.
R. Tairas, J. Gray, Sub-clones: Considering the Part Rather than the Whole, Int. Conf. on Software Engineering Research and Practice (SERP), Las Vegas, NV, 07/10, to appear.
R. Tairas, J. Gray, Sub-clone Refactoring in Open Source Software Artifacts, Symp. on Applied Computing (SAC), Sierre, Switzerland, 03/10: 2364-2365.
R. Tairas, Centralizing Clone Group Representation and Maintenance, Student Research Competition, Int. Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Orlando, FL, 10/09: 781-782.
R. Tairas, M. Mernik, J. Gray, Using Ontologies in the Domain Analysis of Domain-Specific Languages, Workshop on Transformation and Weaving Ontologies in Model-Driven Engineering (TWOMDE), Int. Conf. on Model Driven Engineering, Languages, and Systems (MoDELS), LNCS 5421, Toulouse, France, 09/08: 332-342. (Best Paper Award)
R. Tairas, J. Gray, I. Baxter, Visualization of Clone Detection Results, Eclipse Technology Exchange Workshop (ETX), Int. Conf. on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA), Portland, OR, 10/06: 50-54.

Doctoral Symposium
R. Tairas, Clone Detection and Refactoring, Int. Conf. on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA), Portland, Oregon, 10/06: 50-54.

Tool Demonstrations
R. Tairas, J. Gray, Get to Know Your Clones with CeDAR, Int. Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Orlando, FL, 10/09: 817-818.
Code Clones Literature

- http://www.cis.uab.edu/tairasr/clones/literature/
  - Containing 185 research citations (as of June 2010)
  - Includes web sites of tools, events, and research groups
  - Has been cited by several research publications

"I regard your clone detection literature page as the most up-to-date and condensed source of new clone detection papers."  

"...I often visit and make use of (it)."

"Your clone bibliography page ... has been a very useful resource for our work."

"This site was very useful for me when I was studying the clone detection problem. I think, it is the most useful site concerning clone detection on the Internet."
Thank you

- **Personal:**
  - http://www.cis.uab.edu/tairasr

- **Code Clones Literature:**
  - http://www.cis.uab.edu/tairasr/clones/literature

- **SoftCom Laboratory:**
  - http://www.cis.uab.edu/softcom