Automated Transplantation and Differential Testing for Clones

Tianyi Zhang, Miryung Kim
University of California, Los Angeles
Problem Statement

• Code clones are common in modern software systems.
• Developers often find it difficult to examine the runtime behavior of clones.
• This problem is exacerbated by a lack of tests. 46% of clone pairs are only partially covered by existing test suites.
• We present Grafter to reuse tests between clones and examine behavior differences.
A pair of similar but not identical clones that are detected by an existing clone detection tool, Deckard [ICSE 2007].

```java
public class Copy extends Task{
    private IncludePatternSet includes;
    public void setIncludes(String patterns){
        ...
        StringTokenizer tok = new StringTokenizer(patterns, "\", "");
        while(tok.hasMoreTokens()){
            includes.addPattern(tok.nextToken);
        }
    }
}

class IncludePatternSet {
    public Set<String> set;
    public void addPattern(String s) { set.add(s); }
}

public class Delete extends Task{
    private ExcludePatternSet excludes;
    public void setExcludes(String patterns){
        ...
        StringTokenizer tok = new StringTokenizer(patterns, "\", "");
        while(tok.hasMoreTokens()){
            excludes.addPattern(tok.nextToken);
        }
    }
}

class ExcludePatternSet {
    public Set<String> set;
    public void addPattern(String s) { set.add(s); }
}

* The example is adapted from Apache Ant 1.9.6 for presentation purposes.
```
A programmer updates the use of StringTokenizer to StringUtils.split in the Copy and Delete classes.

* The example is adapted from Apache Ant 1.9.6 for presentation purposes.
Limitation of Existing Techniques

• Existing test reuse technique for clones works only at a method or class level and requires a reuse plan. [Makady & Walker]

• Existing differential testing or random testing techniques are not geared towards intra method clones [Geno, Diffut, Randoop]

• Existing clone inconsistency detection techniques do not detect behavioral differences between clones [Jiang et al., CBCD, SPA]
Grafter: Automated Test Reuse and Differential Testing

public class Copy extends Task{
    private IncludePatternSet includes;
    ...
    String[] tokens = StringUtils.split(patterns, ",");
    for(String tok : tokens){
        includes.addPattern(tok);
    }
    ...
}

public class Delete extends Task{
    private ExcludePatternSet excludes;
    ...
    String[] tokens = StringUtils.split(patterns, ".");
    for(String tok : tokens){
        excludes.addPattern(tok);
    }
    ...
}

Success?
Grafter: Automated Test Reuse and Differential Testing

```
public class Copy extends Task{
    private IncludePatternSet includes;
    ...
    String[] tokens = StringUtils.split(patterns, "\,");
    for(String tok : tokens){
        includes.addPattern(tok);
    }
    ...
}
```

```
public class Delete extends Task{
    private ExcludePatternSet excludes;
    ...
    String[] tokens = StringUtils.split(patterns, "\.");
    for(String tok : tokens){
        excludes.addPattern(tok);
    }
    ...
}
```
Grafter Approach Overview

Clone Pair

A

B

Test Suite

Variation Identification

Code Transplantation

Data Propagation

Differential Testing

<table>
<thead>
<tr>
<th>Test</th>
<th>Clone A</th>
<th>Clone B</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>pass</td>
<td>fail</td>
</tr>
<tr>
<td>T2</td>
<td>pass</td>
<td>pass</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Clone A</th>
<th>Clone B</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>“string”</td>
<td>“string”</td>
</tr>
<tr>
<td>S2</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>
Grafter Approach Overview

**Variation Identification**
- Name Variation
- Type Variation
- Method Call Variation

**Code Transplantation**
- Declare Variables
- Transform Types
- Declare Methods
- Recursive Calls

**Data Propagation**
- Populate intermediate input data to clone
- Transfer intermediate output back to test

**Differential Testing**
- Test Comparison
- State Comparison
**Grafter Approach Overview**

1. **Variation Identification**
   - Name Variation
   - Type Variation
   - Method Call Variation

2. **Code Transplantation**
   - Declare Variables
   - Transform Types
   - Declare Methods
   - Recursive Calls

3. **Data Propagation**
   - Populate intermediate input data to clone
   - Transfer intermediate output back to test

4. **Differential Testing**
   - Test Comparison
   - State Comparison
Grafter Approach Overview

**Variation Identification**
- Name Variation
- Type Variation
- Method Call Variation

**Code Transplantation**
- Declare Variables
- Transform Types
- Declare Methods
- Recursive Calls

**Data Propagation**
- Populate intermediate input data to clone
- Transfer intermediate output back to test

**Differential Testing**
- Test Comparison
- State Comparison
Grafter Approach Overview

**Variation Identification**
- Name Variation
- Type Variation
- Method Call Variation

**Code Transplantation**
- Declare Variables
- Transform Types
- Declare Methods
- Recursive Calls

**Data Propagation**
- Populate intermediate input data to clone
- Transfer intermediate output back to test

**Differential Testing**
- Test Comparison
- State Comparison
Step 1: Variation Identification

public class Delete extends Task{
    private ExcludePatternSet excludes;
    public void setExcludes(String patterns){
        String[] tokens = StringUtils.split(patterns, ",");
        for(String tok : tokens){
            excludes.addPattern(tok);
        }
    }
}

public class Copy extends Task{
    private IncludePatternSet includes;
    public void setIncludes(String patterns){
        String[] tokens = StringUtils.split(patterns, ",");
        for(String tok : tokens){
            includes.addPattern(tok);
        }
    }
}
public class Copy extends Task{
    private IncludePatternSet includes;
    public void setIncludes(String patterns){
        ...
        String[] tokens = StringUtils.split(patterns, ",");
        for(String tok : tokens){
            includes.addPattern(tok);
        }
    }
    ...
}

public class Delete extends Task{
    private ExcludePatternSet excludes;
    public void setExcludes(String patterns){
        ...
        String[] tokens = StringUtils.split(patterns, ",");
        for(String tok : tokens){
            excludes.addPattern(tok);
        }
    }
    ...
}
Step 2: Code Transplantation

Variation Identification -> Code Transplantation -> Data Propagation -> Differential Testing

```java
public class Delete extends Task{
    private ExcludePatternSet excludes;
    public void setExcludes(String patterns){
        String[] tokens = StringUtils.split(patterns, ",");
        for(String tok : tokens){
            excludes.addPattern(tok);
        }
    }
    ...
}

public class Copy extends Task{
    private IncludePatternSet includes;
    public void setIncludes(String patterns){
        String[] tokens = StringUtils.split(patterns, ",");
        for(String tok : tokens){
            includes.addPattern(tok);
        }
    }
    ...
}
```
Step 2: Code Transplantation

```
public class Delete extends Task{
    private ExcludePatternSet excludes;
    public void setExcludes(String patterns){
        ...
        String[] tokens = StringUtils.split(patterns, ".");
        for(String tok : tokens){
            excludes.addPattern(tok);
        }
    }
    ...
}

public class Copy extends Task{
    private IncludePatternSet includes;
    private ExcludePatternSet excludes;
    public void setIncludes(String patterns){
        ...
        excludes = new ExcludePatternSet();
        String[] tokens = StringUtils.split(patterns, ".");
        for(String tok : tokens){
            excludes.addPattern(tok);
        }
    }
    ...
}
Step 3: Data Propagation

public class Delete extends Task{
    private ExcludePatternSet excludes;
    public void setExcludes(String patterns){
        String[] tokens = StringUtils.split(patterns, ".");
        for(String tok : tokens){
            excludes.addPattern(tok);
        }
    }
}

public class Copy extends Task{
    private IncludePatternSet includes;
    private ExcludePatternSet excludes;
    private ExcludePatternSet excludes;
    public void setIncludes(String patterns){
        String[] tokens = StringUtils.split(patterns, ".");
        for(String tok : tokens){
            excludes.addPattern(tok);
        }
        includes.set = excludes.set;
    }
    public void setExcludes(String patterns){
        String[] tokens = StringUtils.split(patterns, ".");
        for(String tok : tokens){
            excludes.addPattern(tok);
        }
        includes.set = excludes.set;
    }
}
Step 4: Differential Testing

Variation Identification → Code Transplantation → Data Propagation → Differential Testing

<table>
<thead>
<tr>
<th>Test</th>
<th>Copy.java</th>
<th>Delete.java</th>
</tr>
</thead>
<tbody>
<tr>
<td>testCopy</td>
<td>pass</td>
<td>fail</td>
</tr>
</tbody>
</table>

**State-level Comparison**

<table>
<thead>
<tr>
<th>State</th>
<th>Copy.java</th>
<th>Delete.java</th>
</tr>
</thead>
<tbody>
<tr>
<td>patterns</td>
<td>“src/<em>.java, test/</em>.java”</td>
<td>“src/<em>.java, test/</em>.java”</td>
</tr>
<tr>
<td>tokens</td>
<td>[“src/<em>.java”, “test/</em>.java”]</td>
<td>[“src/<em>”, “java, test/</em>”, “java”]</td>
</tr>
<tr>
<td>in(ex)cludes</td>
<td>&lt;IncludePatternSet&gt; &lt;set&gt; [“src/<em>.java”, “test/</em>.java”] &lt;/set&gt; &lt;/IncludePatternSet&gt;</td>
<td>&lt;ExcludePatternSet&gt; &lt;set&gt; [“src/<em>”, “java, test/</em>”, “java”] &lt;/set&gt; &lt;/ExcludePatternSet&gt;</td>
</tr>
</tbody>
</table>

**Test-level Comparison**

Test Copy.java Delete.java

testCopy pass fail
• Our tool & dataset are now publicly available.

Behavioral differences are represented in tables and highlighted for ease of investigation.
Evaluation Dataset

• Our dataset contains 52 pairs of non-identical clones from 3 open source projects.
• Our dataset includes 38 Type II clones and 14 Type III clones, based on a well-known clone taxonomy [Roy et al.].

<table>
<thead>
<tr>
<th>Subject</th>
<th>Version</th>
<th>Description</th>
<th>LOC</th>
<th>Test#</th>
<th>Branch</th>
<th>Stmt</th>
<th>Clone Pair</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Ant</td>
<td>1.9.6</td>
<td>A software build framework</td>
<td>267,048</td>
<td>1,864</td>
<td>45%</td>
<td>50%</td>
<td>18</td>
</tr>
<tr>
<td>Java APNS</td>
<td>1.0.0</td>
<td>A Java client for Apple Push Notification service (APNs)</td>
<td>8,362</td>
<td>103</td>
<td>59%</td>
<td>67%</td>
<td>7</td>
</tr>
<tr>
<td>XML Security</td>
<td>2.0.5</td>
<td>A XML signature and encryption library</td>
<td>121,594</td>
<td>396</td>
<td>59%</td>
<td>65%</td>
<td>27</td>
</tr>
</tbody>
</table>
Research Questions

• RQ1. What is Grafter’s transplantation capability?
• RQ2. How does Grafter compare with a static approach by Jiang et al. in its ability to detect differences in clones?
• RQ3. How sensitive is Grafter in detecting behavioral differences caused by mutants?
RQ1. Transplantation Success and Test Reuse Capability

• Grafter successfully grafts 49 of 52 pairs of clones
• Grafter inserts 6 lines of stub code on average to ensure type safety
• Grafter doubles the test coverage for partially tested clone pairs
RQ2. Behavioral Difference Detection

- Jiang et al. [FSE’07] present a static approach that detects three pre-defined cloning inconsistencies: (1) renaming mistake, (2) control construct inconsistency, and (3) control predicate inconsistency.
- Grafter exposes behavioral differences in 84% clone pairs while Jiang et al. detect syntactic inconsistency in 33% clone pairs only.
RQ3. Robustness

- We systematically injected 361 mutants on 30 pairs of clones with no existing test behavioral differences
- We compare its behavioral differencing capability with a static approach by Jiang et al.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOR</td>
<td>Arithmetic operator replacement</td>
<td>$a + b \rightarrow a - b$</td>
</tr>
<tr>
<td>LOR</td>
<td>Logical operator replacement</td>
<td>$a \land b \rightarrow a \mid b$</td>
</tr>
<tr>
<td>COR</td>
<td>Conditional operator replacement</td>
<td>$a \lor b \rightarrow a &amp; &amp; b$</td>
</tr>
<tr>
<td>ROR</td>
<td>Relational operator replacement</td>
<td>$a == b \rightarrow a &gt;= b$</td>
</tr>
<tr>
<td>SOR</td>
<td>Shift operator replacement</td>
<td>$a \ ? \ b \rightarrow a = b$</td>
</tr>
<tr>
<td>ORU</td>
<td>Operator replacement unary</td>
<td>$\neg a \rightarrow: a$</td>
</tr>
<tr>
<td>STD</td>
<td>Statement deletion operator: delete (omit) a single statement</td>
<td>$\text{foo}(a,b) \rightarrow / / \text{foo}(a,b)$</td>
</tr>
<tr>
<td>LVR</td>
<td>Literal value replacement: replace by a positive value, a negative value or zero</td>
<td>$0 \rightarrow 1$</td>
</tr>
</tbody>
</table>
RQ3. Robustness

public void setType(String type) {
    if (type != null && type.length() == 0) {
        this.type = null;
    } else {
        URI tmpType = null;
        try {
            tmpType = new URI(type);
        } catch (URISyntaxException ex) {
            ...
        }
        this.type = tmpType.toString();
    }
}

public void setEncoding(String encoding) {
    if (encoding == null && encoding.length() == 0) {
        this.encoding = null;
    } else {
        URI tmpEncoding = null;
        try {
            tmpEncoding = new URI(encoding);
        } catch (URISyntaxException ex) {
            ...
        }
        this.encoding = tmpEncoding.toString();
    }
}

Mutation Example from Apache XML Security
RQ3. Robustness

- Grafter detects 36% more mutants using the test-level comparison and almost 2X more mutants using the state-level comparison.
- Grafter is less biased to mutant types than Jiang et al.
Conclusion

• This work introduces the first test transplantation and reuse approach for enabling runtime behavior comparison between clones.

• Grafter’s code transplantation succeeds in 94% of the cases.

• The fine-grained differential testing can detect up to 2X more seeded faults than a baseline static cloning bug finder.
Q&A