Challenges of the Dynamic Detection of Functionally Similar Code Fragments

Florian Deissenboeck, Lars Heinemann, Benjamin Hummel, Stefan Wagner
Technische Universität München

CSMR 2012, Szeged, Hungary
Similarities in Software

- Negative effects on maintenance
  - unnecessary code size increase
- Similarities due to copy & paste (code clones) well-researched
- But what about similarities of independent origin?
Functional Similarities

• Not necessarily syntactically similar

```java
public static int factorial(int n) {
    if (n == 0) {
        return 1;
    } else if (n > 0) {
        return n * factorial(n - 1);
    }
    throw new IllegalArgumentException();
}
```

```java
public static int factorial(int n) {
    if (n < 0) {
        throw new IllegalArgumentException();
    }
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result = result * i;
    }
    return result;
}
```

• simion: functionally similar code fragment regarding I/O behavior

• Cannot be found with clone detection [1]

Motivation for this Research

• *Own experience*: functionality is reimplemented in large code bases

• Automated detection could be as beneficial as clone detection for QA

• Encouraged by results of existing approach for C systems (Jiang&Su, ISSTA 2011)

• *Goal*: Transfer to object-oriented Java systems
Disclaimer

• Detection results of approach unsatisfactory (few simions detected & many false-positives)

• **But**: Valuable insights regarding challenges

DON‘T TRY THIS AT HOME!
General Idea

- Execute candidate code fragments on random input and compare output
Pipeline

Source Code

Chunking

i_1, i_2, ...
o_1, o_2, ...

i_1, i_2, ...
o_1, o_2, ...

i_1, i_2, ...
o_1, o_2, ...
Pipeline

- Deterministic random values for primitive types
- Recursive algorithm for object types
- Parameter permutation
- 100 input vectors for each chunk
Pipeline

- Source Code
  - Chunking
  - Input Generation
  - Execution

- Execute each chunk with all input vectors
- Security Manager to prevent “harmful“ operations
Pipeline

Source Code → Chunking → Input Generation → Execution → Comparison

- Compare outputs for each output variable
- Simion if inputs and outputs for all executions are equal
- Filter clones and “trivial“ I/O behavior
Case Study: Study Objects

<table>
<thead>
<tr>
<th>System</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commons Lang</td>
<td>17,504</td>
</tr>
<tr>
<td>Freemind</td>
<td>51,762</td>
</tr>
<tr>
<td>Jabref</td>
<td>74,586</td>
</tr>
<tr>
<td>Jetty</td>
<td>29,800</td>
</tr>
<tr>
<td>JHotDraw</td>
<td>78,902</td>
</tr>
</tbody>
</table>

“Info1“: Collection of 109 student implementations of an e-mail address validator (each 8..55 statements)
Number of Chunks

• Each chunk has to pass through pipeline
• Direct impact on processing time
• Experiments with 3 chunking strategies

```c
void foo ()
{
    
    
}
```

method

```c
void foo ()
{
    ___________
    
    
}
```

sliding window

```c
void foo ()
{
    
    
    
}
```

intent
## Number of Chunks (2)

<table>
<thead>
<tr>
<th>System</th>
<th>Method</th>
<th>Sl. Wind.</th>
<th>Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commons Lang</td>
<td>1,538</td>
<td>7,940</td>
<td>1,843</td>
</tr>
<tr>
<td>Freemind</td>
<td>1,984</td>
<td>80,816</td>
<td>20,632</td>
</tr>
<tr>
<td>Jabref</td>
<td>2,085</td>
<td>133,556</td>
<td>21,388</td>
</tr>
<tr>
<td>Jetty</td>
<td>1,457</td>
<td>22,006</td>
<td>7,713</td>
</tr>
<tr>
<td>JHotDraw</td>
<td>2,813</td>
<td>211,283</td>
<td>16,221</td>
</tr>
</tbody>
</table>
Technical Challenges

• Parameter types are interfaces, abstract classes or untyped collections (37%-94% of chunks)
• Call to UI, networking or I/O API functions (5%-60% of chunks)
Effectiveness of Approach

• Full detection pipeline
• How many chunks “pass” each pipeline step?
• How many simions are detected?
Effectiveness (Student programs)

<table>
<thead>
<tr>
<th>Pipeline Step</th>
<th>#Chunks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chunk Extraction</td>
<td>240</td>
</tr>
<tr>
<td>Input Generation</td>
<td>134</td>
</tr>
<tr>
<td>Execution</td>
<td>133</td>
</tr>
<tr>
<td>Comparison</td>
<td>105</td>
</tr>
</tbody>
</table>

= #simions

method-based chunking
# Effectiveness (Jabref)

<table>
<thead>
<tr>
<th>Pipeline Step</th>
<th>#Chunks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chunk Extraction</td>
<td>2,085</td>
</tr>
<tr>
<td>Input Generation</td>
<td>1,542</td>
</tr>
<tr>
<td>Execution</td>
<td>621</td>
</tr>
<tr>
<td>Comparison</td>
<td>55</td>
</tr>
</tbody>
</table>

= #simions

**method-based chunking**
# Manual Inspection of Simions

[not in paper]

<table>
<thead>
<tr>
<th>System</th>
<th>Detected</th>
<th>Confirmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commons Lang</td>
<td>54</td>
<td>33 (61%)</td>
</tr>
<tr>
<td>Freemind</td>
<td>11</td>
<td>7 (64%)</td>
</tr>
<tr>
<td>Jabref</td>
<td>55</td>
<td>9 (16%)</td>
</tr>
<tr>
<td>Jetty</td>
<td>15</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>JHotDraw</td>
<td>18</td>
<td>2 (11%)</td>
</tr>
<tr>
<td><strong>Overall</strong></td>
<td><strong>153</strong></td>
<td><strong>51 (33%)</strong></td>
</tr>
</tbody>
</table>
Detected Simions

```java
public static int stringToInt(String str, int defaultValue) {
    try {
        return Integer.parseInt(str);
    } catch (NumberFormatException nfe) {
        return defaultValue;
    }
}
```

```java
public static int stringToInt(String str, int defaultValue) {
    return toInt(str, defaultValue);
}
```

```java
public static int toInt(String str, int defaultValue) {
    if (str == null) {
        return defaultValue;
    }
    try {
        return Integer.parseInt(str);
    } catch (NumberFormatException nfe) {
        return defaultValue;
    }
}
```
Detected Simions (2)

```java
public static boolean toBoolean(int value, int trueValue, int falseValue) {
    if (value == trueValue) {
        return true;
    } else if (value == falseValue) {
        return false;
    } // no match
    throw new IllegalArgumentException("The Integer did not match either specified value");
}
```

```java
public static boolean toBoolean(Integer value, Integer trueValue, Integer falseValue) {
    if (value == null) {
        if (trueValue == null) {
            return true;
        } else if (falseValue == null) {
            return false;
        } else if (value.equals(trueValue)) {
            return true;
        } else if (value.equals(falseValue)) {
            return false;
        } // no match
        throw new IllegalArgumentException("The Integer did not match either specified value");
    }
```
Differences to Jiang&Su

- Input generation complicated for Java (e.g. interfaces)
- Jiang&Su replace function calls with random values
- We filter “trivial“ simions, e.g. identity function (occurs often)
- Linux kernel code vs. Java GUI apps
Discussion

• Low detection results: no simions or flaws in detection?
• What is the recall? (infeasible to determine manually)

• Main limitation: random testing approach
  – no input or generated input does achieve sufficient code coverage
  – can potentially addressed with white-box methods
• Notion of I/O similarity may not be suitable
  – e.g. different data types or signatures
Thank you.
Questions?