Context-Based Detection of Clone-Related Bugs

Lingxiao Jiang, Zhendong Su, Edwin Chiu
University of California at Davis
Outline

- Introduction and samples of cloning errors
- Definitions of inconsistencies
- Inconsistency filters for bug detection
- Empirical evaluation
- Conclusion and future work
Code Clones are Common

- Replicating similar functionalities helps improve development productivity
- Many clone detection techniques exist
  - CCFinder: 29% cloned code in JDK
  - CP-Miner: 23% cloned code in the Linux kernel 2.6.6
- Clone removal may not always be beneficial
  - Kim, Sazawal, Notkin, ESEC/FSE 2005
  - Rajapakse, Jarzabek, ICSE 2007
Clone-Related Bugs

- Copy-paste may create many inconsistencies among code clones
- Inconsistencies among clones can be indications of potential defects
TableItem item = Get...... ;

for (int i=0; i<colCnt; i++)
{
    Point start = new Point (item...... ;
    start =
        fTableViewer.getTable()......;

    ...... if (start.x < point.x && end.x > point.x)
    return i;
}

TableItem item = Get...... ;
if (item != null)
{
    for (int i=0; i<colCnt; i++)
    {
        Point start = new Point (item...... ;
        start =
            fTableViewer.getTable()......;

        ...... if (start.x < point.x && end.x > point.x)
        return i;
    }
}
Clone-Related Bugs - Examples 2/3

if ( length >= 9 && 
    strncmp (buffer, "EESOXSCSI", 9) 
    == 0 )
{
    buffer += 9; 
    length -= 9;
    if ( length >= 5 && 
        strncmp(buffer, "term=" , 5) == 0) 
    {
        ...... 
    }
    else
    ret = -EINVAL;
}
else
ret = -EINVAL;

if ( length >= 11 &&
    strcmp (buffer, "CUMANASCSI2") 
    == 0)
{
    buffer += 11;
    length -= 11;
    if ( length >= 5 &&
        strncmp(buffer, "term=" , 5) == 0) 
    {
        ...... 
    }
    else
    ret = -EINVAL;
}
else
ret = -EINVAL;
Clone-Related Bugs - Examples 3/3

if ( l_stride != NULL )
    mpz_cdiv_q ( X1, X1, l_stride->value.integer );

if ( l_stride != NULL )
    mpz_cdiv_q ( X2, X2, r_stride->value.integer );
Outline

- Introduction and samples of cloning errors
- Definitions of inconsistencies
- Inconsistency filters for bug detection
- Empirical evaluation
- Conclusion and future work
Types of Inconsistencies 1/3

Type-1:
- Given two clones, their surrounding contextual nodes are different types
- E.g., function-def vs. if-statement

```
TableItem item = Get...... ;
for (int i=0; i<colCnt; i++)
{
    Point start = new Point (item...... ;
    start =
        fTableViewer.getTable()......;
    ..... if (start.x < point.x && end.x > point.x)
        return i;
}
```

```
TableItem item = Get...... ;
if (item != null)
{
    for (int i=0; i<colCnt; i++)
    {
        Point start = new Point (item...... ;
        start =
            fTableViewer.getTable()......;
        ..... if (start.x < point.x && end.x > point.x)
            return i;
    }
}
Types of Inconsistencies 2/3

Type-2:
- Given two clones, their surrounding conditions are different
- E.g., `strn cmp` vs. `str cmp`

```c
if ( length >= 9 &&
    strn cmp (buffer, "EESOXSCSI", 9) == 0 )
{
    buffer += 9;
    length -= 9;

    if ( length >= 5 &&
        strn cmp (buffer, "term=", 5) == 0 )
    {
        ......
    }
    else
        ret = -EINVAL;
} else
    ret = -EINVAL;
```

```c
if ( length >= 11 &&
    str cmp (buffer, "CUMANASCSI2") == 0 )
{
    buffer += 11;
    length -= 11;

    if ( length >= 5 &&
        strn cmp (buffer, "term=", 5) == 0 )
    {
        ......
    }
    else
        ret = -EINVAL;
} else
    ret = -EINVAL;
```
Types of Inconsistencies 3/3

Type-3:
- Given two clones, the number of unique identifiers within them are different
- E.g., 6 vs. 7

```c
if ( l_stride != NULL )
    mpz_cdiv_q ( X1, X1, l_stride->value.integer );
```

```c
if ( l_stride != NULL )
    mpz_cdiv_q ( X2, X2, r_stride->value.integer );
```
Outline

- Introduction and samples of cloning errors
- Definitions of inconsistencies
- Inconsistency filters for bug detection
- Empirical evaluation
- Conclusion and future work
Inconsistency Filters

- Not all inconsistencies are created equal
- Many may not be bug indicators
  - Spatially close clones
  - Intentional (w/ big differences) clones
Inconsistency Filters - Examples 1/5

Contextual nodes are \texttt{loop vs. function-def}

\begin{tabular}{|l|l|}
\hline
for ( x = A; x <= B; x++ ) & for ( y = C; y <= D; y++ ) \\
\multicolumn{2}{|c|}{\ldots more similar code\ldots} \\
\hline
\end{tabular}
Inconsistency Filters - Examples 2/5

Contextual nodes are **loop vs. if-statement**

<table>
<thead>
<tr>
<th>for ( x = A; x &lt;= B; x++ )</th>
<th>if ( y != D )</th>
</tr>
</thead>
<tbody>
<tr>
<td>...similar code...</td>
<td>...similar code...</td>
</tr>
</tbody>
</table>
Inconsistency Filters - Examples 3/5

Contextual conditions are significantly different

\[
\begin{align*}
\text{if ( x } & \leq B ) \\
& \quad \ldots \text{similar code}\ldots \\
\end{align*}
\quad
\begin{align*}
\text{if ( y+z } & \neq C*D ) \\
& \quad \ldots \text{similar code}\ldots \\
\end{align*}
\]
Identifiers within clones are significantly different

```c
if ( data != NULL )
{
    data->in[ix] = read8(data, VT_REG1211);
    data->in_min[ix] = read8(data, VT_REG1211);
    data->in_max[ix] = read8(data, VT_REG1211);
    ......  
}
```

```c
If ( data != NULL )
{
    data->in[i] = read_data(client, ADM1025_REG);
    data->in_min[i] = read_data(client, ADM1025_MIN);
    data->in_max[i] = read_data(client, ADM1025_MAX);
    ......  
}
```
Inconsistency Filters - Examples 5/5

Clones are spatially very close

| Line 1: | the code piece |
| Line 2: | is from line 1 |
| Line 3: | to line 9, |
| Line 4: | and it's similar |
| ...... | to the code from |
| Line 8: | line 11 |
| Line 9: | to line 19 |
| Line 10: | |

Clones with a small difference

| Line 11: | the code piece |
| Line 12: | from line 11 |
| Line 13: | to line 19 |
| Line 14: | has a little inconsistency |
| ...... | against the code from |
| Line 18: | line 1 |
| Line 19: | to line 9 |
Outline

- Introduction and samples of cloning errors
- Definitions of inconsistencies
- Inconsistency filters for bug detection
- Empirical evaluation
- Conclusion and future work
Empirical Evaluation

Based on DECKARD
- A tree-based, accurate, scalable clone detection tool
- Parameters for clone detection:
  - Similarity
  - Minimum token number
  - Stride

<table>
<thead>
<tr>
<th>Application</th>
<th># Files</th>
<th># LoC</th>
<th># Cloned LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux kernel 2.6.19 (C)</td>
<td>8733</td>
<td>5639833</td>
<td>358331</td>
</tr>
<tr>
<td>Eclipse (Java)</td>
<td>8320</td>
<td>1832332</td>
<td>70455</td>
</tr>
</tbody>
</table>
Code Reduction for Inspection

![Graph showing code reduction for inspection]

- Linux kernel 2.6.19:
  - Total LoC: 5640
  - LoC After filtering: 358
  - Code Reduction: 6.35%

- Eclipse:
  - Total LoC: 1832
  - LoC After filtering: 70
  - Code Reduction: 3.85%

Legend:
- # LoC
- # Cloned LoC
- # LoC (Inconsistencies)
- # LoC After filtering
Bug Detection Rates

# Clone groups

- **Linux kernel:** 7852
- **Eclipse:** 2246

Number

- **Linux kernel 2.6.19:**
  - Total # Inconsistencies: 881
  - After filtering: 396
  - # Bugs After filtering: 33
  - # Clone groups: 11.2%

- **Eclipse:**
  - Total # Inconsistencies: 461
  - After filtering: 265
  - # Bugs After filtering: 15
  - # Clone groups: 20.5%

Legend:
- Total # Inconsistencies
- After filtering
- # Bugs After filtering
- Total # Bugs
Inconsistency Types vs. Bugs

After filtering

8.7% 7.2% 17.4% 1.4%

5.4% 2.7%

Linux kernel 2.6.19

Eclipse

# Type-1 Incons. # Type-1 Bugs
# Type-2 Incons. # Type-2 Bugs
# Type-3 Incons. # Type-3 Bugs
False positives

- Syntactic nature
  - Deckard
  - Inconsistencies
  - Filters

- Hypothesized for further pruning
  - Semantic-based clone detection
  - Semantic-aware inconsistencies
## Breakdown of Discovered Bugs

<table>
<thead>
<tr>
<th>No.</th>
<th>Category</th>
<th># Bugs (Linux)</th>
<th># Bugs (Eclipse)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Total</td>
<td>41</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>Missed conditional checks</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Negated conditions</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Inappropriate conditions</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Off-by-one</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Inappropriate scoping</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>Missed or inappropriate qualifiers</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>Wrong variables</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>Missed or inappropriate locks</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>Inappropriate logic for corner cases</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>Unhandled cases or exceptions</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>Wrong function calls</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>Wrong data fields</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>Wrong macros</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Additional Evaluation: Comparison with CP-Miner

- CP-Miner
  - Token-based clone detection
  - Identifier mapping-based bug detection
- On Linux kernel 2.6.19

- CP-Miner: 251 reports
- 396 reports
- 13
- 42
Outline

- Introduction and samples of cloning errors
- Definitions of inconsistencies
- Inconsistency filters for bug detection
- Empirical evaluation
- Conclusion and future work
Conclusion and Future work

- Proposed a general notion of inconsistencies among code clones that can help reveal program errors

- Complement program analyses
  - Reduce the amount of code for analysis
  - Provide hints for possible causes of defects

- Extend to semantic-based contextual inconsistencies
  - Semantic-based (PDG-based) clones
  - Semantic inconsistency inference
    - E.g., Dillig, Dillig, Aiken, PLDI 2007
Thank you!

Questions?
jiangl@cs.ucdavis.edu