Research based on Clone Detection

Overview

• An empirical study of code clone genealogies [1]
• A case study of cross-system porting in forked projects [2]
An empirical study of code clone genealogies

Based on Miryung Kim’s lecture [4]

Problem Statement

- People believe that code clones indicate bad smells of poor design
  - programmers may introduce bugs when maintaining code clones inconsistently
- Is that true?
Findings in Previous Study[3]

• Even skilled programmers create and manage code clones with clear intent
  – Programmers cannot refactor clones because of programming language limitations
  – Programmers keep and maintain clones until they realize how to abstract the common part
  – Programmers often apply similar changes to clones

Research Questions

• How do clones evolve over time?
  – Consistently changed?
  – Long-lived (or short-lived)?
  – Easily refactorable?
Model of Clone Evolution

Clone Group Evolution Pattern

- Same: NG = OG
- Add: at least a new clone is added to NG
- Subtract: at least an old clone is removed from OG
- Consistent Change: all clones are consistently changed
- Inconsistent Change: clones are changed inconsistently
- Shift: at least one clone in NG partially overlap with a clone OG
• There can also be some other evolution pattern, e.g., copy-paste-modify to generate a whole new clone group

Clone genealogy

• A set of clone groups connected by cloning relationship over time

Diagram: A set of clones A, B, C, D, E, F, G connected by cloning relationships. Clones A, B, C, D are consistently changed, while clones E, F, G are copied, pasted, and modified.
Clone Genealogy Extractor (CGE)

- Given multiple versions of a program
  - Find clone groups in each version using CCFinder
  - Find cloning relationship between clone groups across versions based on text similarity
  - Identify a clone genealogy for each set of connected clone groups
  - Identify clone evolution behaviors in each genealogy

Experiment Settings

- Two Java subject programs

<table>
<thead>
<tr>
<th>Program</th>
<th>carol</th>
<th>dnsjava</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>7878 - 23731</td>
<td>5756 - 21188</td>
</tr>
<tr>
<td>duration</td>
<td>2 yrs. 2 mos.</td>
<td>5 yrs. 8 mos.</td>
</tr>
<tr>
<td>versions</td>
<td>37</td>
<td>224</td>
</tr>
</tbody>
</table>
Detected Clone Genealogies

<table>
<thead>
<tr>
<th># of genealogies</th>
<th>carol</th>
<th>dnsjava</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>122</td>
<td>140</td>
</tr>
<tr>
<td>false positive</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>true positive</td>
<td>109</td>
<td>125</td>
</tr>
<tr>
<td>locally unfactorable</td>
<td>70 (64%)</td>
<td>61 (49%)</td>
</tr>
<tr>
<td>consistently changed</td>
<td>41 (38%)</td>
<td>45 (36%)</td>
</tr>
</tbody>
</table>

RQ1: How often do programmers change clones consistently?

• Approach
  – A genealogy has a “consistent change” pattern iff all lineages include at least one consistent change pattern

• Result
  – 38% and 36% of genealogies include a consistent change pattern
RQ2: What is the life time of clones?

• Separate live genealogies from dead genealogies
  – Dead genealogies: those which do not contain clones in the final version
• Calculate the life span of each dead genealogies

Result

• Among 109 clone genealogies of carol, 53 are dead
• Among 125 clone genealogies of dnsjava, 107 are dead
• Among the dead genealogies:

<table>
<thead>
<tr>
<th>disappeared within</th>
<th>carol</th>
<th>dnsjava</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 versions</td>
<td>52%</td>
<td>35%</td>
</tr>
<tr>
<td>5 versions</td>
<td>75%</td>
<td>36%</td>
</tr>
<tr>
<td>10 versions</td>
<td>79%</td>
<td>48%</td>
</tr>
</tbody>
</table>
How do lineages disappear?

<table>
<thead>
<tr>
<th>reasons</th>
<th>carol</th>
<th>dnsjava</th>
</tr>
</thead>
<tbody>
<tr>
<td>divergent changes</td>
<td>26%</td>
<td>34%</td>
</tr>
<tr>
<td>refactoring or removal</td>
<td>67%</td>
<td>45%</td>
</tr>
<tr>
<td>cut off by the threshold</td>
<td>7%</td>
<td>21%</td>
</tr>
</tbody>
</table>

Contrary to conventional wisdom, immediate refactoring may be unnecessary or counterproductive in some cases.

RQ3: Are clones easily refactorable?

- A clone group is locally unfactorable if
  - programmers cannot use standard refactoring techniques, or
  - programmer must deal with cascading non-local changes, or
  - programmers cannot remove duplication due to programming language limitations.
Example

public void exportObject(Remote obj) throws RemoteException{
    if (TraceCarol.isDebugEnabled()) {
        TraceCarol.debugRmiCarol("MultiPRODelegate.exportObject(" ... .
    }
    try {
        if (init) {
            for (Enumeration e = activePatch.elements();
                e.hasMoreElements();
                ((ObjDigi)e.nextElement()).exportObject(obj);
        }
    }
    catch (Exception e) {
        String msg = "exportObject(Remote obj) fail";
        TraceCarol.error(msg,e);
        throw new RemoteException(msg);
    }
}

Summary

• Immediate and aggressive refactoring may be unnecessary for volatile and diverging clones
• Refactoring may not help many long-lived and consistently changing clones
• Q: Do you have other observations?
A Case Study of Cross-System Porting in Forked Projects [2]

Based on Baishakhi Ray’s slides

Problem Statement

• Software forking is important
  – Developers create a variant product by copying and modifying an existing product
  – E.g., FreeBSD, OpenBSD, and NetBSD evolve from the same code base
• What is the characteristic of code changes ported between peer projects?
Research Questions

- How often do developers port edits between projects?
- Are ported changes more defect-prone than others?
- How many developers are involved in patch porting?
- How long does it take for a patch to propagate across projects?
- Where is the porting effort focused on?

Methodology

- Repertoire: Detect ported edits by finding code clones in diff files using CCFinder
- Accuracy measurement
  - Construct a ground truth set of known ported edits, and use it to evaluate precision and recall of Repertoire
  - 94% precision and 84% recall
RQ1: How often do developers port edits?

\[
\text{avg\_porting\_rate} = \frac{\sum_{\text{releases}} \text{ported\_edits}}{\sum_{\text{releases}} \text{total\_edits}}
\]

Result

Porting is significant in the BSD family evolution, and it is not necessarily decreasing over time.
RQ2: Are ported edits more error-prone than others?

<table>
<thead>
<tr>
<th></th>
<th>CLOC</th>
<th>Ported CLOC</th>
<th>Non-Ported CLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeBSD</td>
<td>475482</td>
<td>654858</td>
<td>4100004</td>
</tr>
<tr>
<td>Correlation with bugs</td>
<td>0.26</td>
<td>0.15</td>
<td>0.25</td>
</tr>
<tr>
<td>p-value</td>
<td>&lt; 2.2e-16</td>
<td>&lt; 2.2e-16</td>
<td>&lt; 2.2e-16</td>
</tr>
<tr>
<td>NetBSD</td>
<td>4097338</td>
<td>636006</td>
<td>3461332</td>
</tr>
<tr>
<td>Correlation with bugs</td>
<td>0.41</td>
<td>0.36</td>
<td>0.42</td>
</tr>
<tr>
<td>p-value</td>
<td>&lt; 2.2e-16</td>
<td>&lt; 2.2e-16</td>
<td>&lt; 2.2e-16</td>
</tr>
<tr>
<td>OpenBSD</td>
<td>4728360</td>
<td>507810</td>
<td>4220550</td>
</tr>
<tr>
<td>Correlation with bugs</td>
<td>0.37</td>
<td>0.32</td>
<td>0.38</td>
</tr>
<tr>
<td>p-value</td>
<td>&lt; 2.2e-16</td>
<td>&lt; 2.2e-16</td>
<td>&lt; 2.2e-16</td>
</tr>
</tbody>
</table>

- **CLOC**: Cumulative number of changed lines
- The correlation between bug fixes and ported edits is weaker than that between bug fixes and non-ported edits
- Q: Any improvement for the experiment?

RQ3: How many developers are involved in porting patches from other projects?

A significant portion of active committers port changes, but some do more porting work than others.
RQ4: How long does it take for a patch to propagate to different projects?

While most ported changes migrate to peer projects in a relatively short amount of time, some changes take a very long time to propagate to other projects.

RQ5: Where is the porting effort focused on?

Ported changes affect about 12% to 19% of modified files and porting effort is concentrated on specific parts of the BSD codebase.
Top 4 directories with the largest amount of ported changes

<table>
<thead>
<tr>
<th>Rank</th>
<th>FreeBSD</th>
<th>NetBSD</th>
<th>OpenBSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>src/crypto/openssl</td>
<td>src/sys/arch</td>
<td>src/sys/dev</td>
</tr>
<tr>
<td>2</td>
<td>src/crypto/openssh</td>
<td>src/sys/dev</td>
<td>src/lib/libssl</td>
</tr>
<tr>
<td>3</td>
<td>src/crypto/heimdal</td>
<td>src/crypto/dist</td>
<td>src/sys/arch</td>
</tr>
<tr>
<td>4</td>
<td>src/sys/dev</td>
<td>src/gnu/dist</td>
<td>src/usr.sbin/ppp</td>
</tr>
</tbody>
</table>

Implications

- Call for automated approaches for cross-system porting (implied from RQ1)
- Call for tools to notify developers of potential collateral evolution and cross-system change impact analysis (implied from RQ5)
- Q: Any research questions you want to ask and make implications based on that?
Reference