An Empirical Study of Code Clone Genealogies

Miryung Kim, Vibha Sazawal, David Notkin, and Gail Murphy
University of Washington
University of British Columbia

ESEC/FSE Sept 2005
Conventional Wisdom

**Code clones indicate bad smells of poor design. We must aggressively refactor clones.**

```java
public void updateFrom (Class c ) {
    String cType = Util.makeType(c.Name());
    if (seenClasses.contains(cType)) {
        return;
    }
    seenClasses.add(cType);
    if (hierarchy != null) {
        ....
    }
}...
```

```java
public void updateFrom (ClassReader cr ) {
    String cType = CTD.convertType (c.Name());
    if (seenClasses.contains(cType)) {
        return;
    }
    seenClasses.add(cType);
    if (hierarchy != null) {
        ....
    }
}...
```
Our Previous Study of Copy and Paste Programming Practices at IBM

• Even skilled programmers often 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 of clones.
  - Programmers often apply similar changes to clones.

[Kim et al. ISESE2004]
Research Questions

How do clones evolve over time?
- consistently changed?
- long-lived (or short-lived)?
- easily refactorable?
Previous Studies of Code Clones

- automatic clone detection
  - lexical, syntactic (AST or PDG), metric, etc.

- studies of clone coverage ratio
  - gcc (8.7%), JDK (29%), Linux (22.7%), etc.

- studies of clone coverage change
  - changes of clone coverage in Linux [Antoniol+02], [Li+04]

These studies do not answer how individual clones changed with respect to other clones.
Outline

motivation

- clone genealogy: model and tool
- study procedure and results
Model of Clone Evolution

Clone group

Code snippet

Cloning relationship

Location overlapping relationship

Version i

Version i+1

Version i+2

Version i+3

Add

Consistent Change

Inconsistent Change

Evolution Patterns
Clone genealogy is a set of clone groups connected by cloning relationships over time.

- Consistently changed
- Copied, pasted, and modified

Lineage:

1. A
   - B
   - C
   - D
2. E
   - F
   - G
3. A
   - B
   - D
4. E
   - F
Clone Genealogy Extractor (CGE)

Given multiple versions of a program, $V_k$ for $1 \leq k \leq n$.

- find clone groups in each version using CCFinder.
- find cloning relationships among clone groups of $V_i$ and $V_{i+1}$ using CCFinder.
- map clones of $V_i$ and $V_{i+1}$ using diff based algorithm.
- separate each connected component of cloning relationships (a clone genealogy).
- identify clone evolution patterns in each genealogy.
Outline

motivation

clone genealogy : model and tool

study procedure and results
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 years 2 months</td>
<td>5 years 8 months</td>
</tr>
<tr>
<td>versions</td>
<td>37</td>
<td>224</td>
</tr>
</tbody>
</table>

versions: a set of check-in snapshots that increased or decreased the total lines of code clones
Running CGE on Java Programs

• CCFinder setting
  - minimum token length = 30
  - longest sequence matching

• CGE setting
  - text similarity threshold = 0.3

• false positives
  - repetitive field declaration
  - repetitive static method invocation
  - a series of case switch statements
  - etc.
Consistently Changing Clones

Question: How often do programmers update clones consistently?

Study Method:

- A genealogy has a “consistent change” pattern iff all lineages include at least one consistent change pattern.
- We counted genealogies with a “consistent change” pattern.
Consistently Changing Clones

Results:

• 38% and 36% of genealogies include a consistent change pattern.

Consistent with conventional wisdom, programmers often apply similar changes repetitively to clones.
Volatile Clones

Question: How long do clones survive in the system before they disappear, and how do they disappear?

Study Method:
- A genealogy is “dead” if it does not include clones of the final version.
- We measured the age (lifespan or length) of dead genealogies.
Volatile Clones

Results:

<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>

- 26% and 34% of clone lineages were discontinued because of divergent changes in the clone group.
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.
Locally Unfactorable Clones

Question: How many clones are difficult to refactor?

Study Method:

• 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.

• We manually inspected all genealogies and counted locally unfactorable genealogies.
public void exportObject(Remote obj) throws RemoteException{
    if (TraceCarol.isDebugRmiCarol()) {
        TraceCarol.debugRmiCarol(
            "MultiPRODelegate.exportObject(" ... .
    }
    try {
        if (init) {
            for (Enumeration e = activePtcls.elements(); ... 
               ((ObjDlgt)e.nextElement()).exportObject(obj);
        } 
    } catch (Exception e) {
        String msg = "exportObject(Remote obj) fail";
        TraceCarol.error(msg,e);
        throw new RemoteException(msg);
    }
}

public void unexportObject(Remote obj) throws NoSuchObjectException {
    if (TraceCarol.isDebugRmiCarol()) {
        TraceCarol.debugRmiCarol(
            "MultiPRODelegate.unexportObject(" ... .
    }
    try {
        if (init) {
            for (Enumeration e = activePtcls.elements(); ... 
               ((ObjDlgt)e.nextElement()).unexportObject(obj);
        } 
    } catch (Exception e) {
        String msg = "unexportObject(Remote obj) fail";
        TraceCarol.error(msg,e);
        throw new NoSuchObjectException(msg);
    }
}
Locally Unfactorable Clones

Result:

- 64% and 49% of genealogies are locally unfactorable.

*Contrary to conventional wisdom, refactoring may not remove many clones easily.*
Long-Lived Clones

Question: For clones that live for a long time and tend to change with other clones, can they be easily refactored?

Study Method:

• We measured cumulative proportion of locally unfactorable and consistently changed genealogies.
Long-Lived Clones

Results:

• 51% and 61% of genealogies that lasted more than half of programs’ lifetime are locally unfactorable and consistently changing.

• The proportion of locally unfactorable yet consistently changed genealogies increases with the age of genealogies.

Contrary to conventional wisdom, refactoring cannot help many consistently changed, long-lived clones.
Study Limitations

- clone detection techniques
  - CCFinder vs. other clone detection techniques.
- location tracking techniques
  - diff vs. other location tracking techniques.
- subject programs
  - 20KLOC vs. large scale projects
- time granularity
  - versions vs. editing operations
- language dependency
  - Java vs. other languages
Summary

• We have built a tool that extracts history of code clones from a set of program versions.

• Our study of clone genealogy contradicts some conventional wisdom about code clones.
  - Immediate and aggressive refactoring may be unnecessary for volatile and diverging clones.
  - Refactoring may not help many long-lived and consistently changing clones.

• Our study opens up opportunities for complementary clone maintenance tools.