Using a **Clone Genealogy Extractor** for Understanding and Supporting Evolution of Code Clones

Miryung Kim and David Notkin

May 17th, 2005
Conventional Wisdom

Code clones indicate bad smells of poor design.

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

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

[Kim et al. 2004]

• Even skilled programmers often create and manage code clones with clear intent.
  - programmers cannot refactor clones because of programming language limitations.
  - programmers often apply similar changes to clones.
  - programmers keep and maintain clones until they realize how to abstract the common part of clones.
### Gathering Data from Other Existing Software Projects

<table>
<thead>
<tr>
<th></th>
<th>What we captured for our copy and paste study</th>
<th>What most existing software projects provide in reality</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data source</strong></td>
<td>editing logs captured by an Eclipse plug-in</td>
<td>version control system or release archives</td>
</tr>
<tr>
<td><strong>Scale</strong></td>
<td>a single programmer workspace</td>
<td>multi programmer collaborative environment</td>
</tr>
<tr>
<td><strong>Granularity</strong></td>
<td>key strokes and editing operations such as copy, cut, and paste</td>
<td>check-ins or releases</td>
</tr>
</tbody>
</table>
Outline

☑ motivation
☐ clone genealogy: model and tool
☐ potential applications of clone genealogy
☐ ongoing work
Model of Clone Evolution

Location overlapping relationship

Cloning relationship

Code snippet

Clone group

Version i Version i+1 Version i+2 Version i+3 Version i+4

Add Consistent Change Inconsistent Change Subtract

Evolution Patterns
Clone genealogy reflects how programmers evolved code clones by copying, pasting and modifying code.

consistently changed

copied, pasted, and modified
Clone Genealogy Extractor

- given multiple versions of a program, $V_k$ for $1 \leq k \leq n$.
- find clone groups in each version using a clone detector \textit{(CCFinder)}
- find cloning relationships between clone groups between $V_i$ and $V_{i+1}$, using the same clone detector
- separate each connected component of cloning relationships, a clone genealogy.
- identify clone evolution patterns in each genealogy.
Outline

- motivation
- clone genealogy: model and tool
  - potential applications of clone genealogy
  - ongoing work
Potential Tool: Warning Cloning Related Bug

- Failing to update clones consistently often creates a bug in software.
- Goal: warn programmers about inconsistent update of clones.
Potential Tool: Decision Support for Maintaining Clones

• Refactoring too early vs. refactoring too late

• Goal: suggest when to refactor clones.
Outline

- motivation
- clone genealogy: model and tool
- potential applications of clone genealogy
- ongoing work
Empirical Analysis of Clone Genealogies

- Clone genealogy information *enables* us to study *evolutionary characteristics* of clones.
  - How often do programmers actually make similar changes to clones?
  - What are evolutionary characteristics of clones that are difficult to refactor?
  - How long do clones survive in the system before they disappear?
Summary

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

• We envision that clone genealogy information can be used for potential clone maintenance tools.

• Our previous & ongoing work checks the validity of conventional wisdom using a clone genealogy extractor.

Acknowledgement:
Software engineering laboratory at Osaka University provided CCFinder.
GRASE research group at UCSC provided Kenyon.
Collaborators: Vibha Sazawal and Gail Murphy