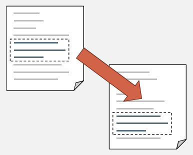
Sub-clone Refactoring in Open Source Software Artifacts

Robert Tairas
Software Composition and Modeling Laboratory
Department of Computer and Information Sciences
University of Alabama at Birmingham, USA
tairasr@cis.uab.edu

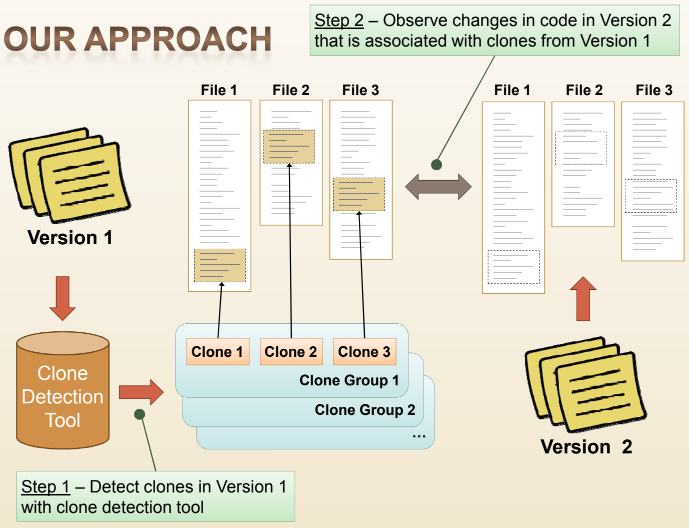
Jeff Gray
Department of Computer Science
University of Alabama, USA
gray@cs.ua.edu

INTRODUCTION

- *Code clones* represent duplicate sections of code found in a program. Clones representing the same duplication are grouped into *clone groups*.
- Clone detection tools provide automated support for discovering duplicated sections of code, the results of which can inform the process of clone removal through refactoring activities.
- We studied how clones reported by clone detection tools would have been refactored by observing actual refactorings associated with the clones.
- The actual refactorings were found by evaluating code changes between two versions of the source code of open source software artifacts.



OUR APPROACH


Step 2 – Observe changes in code in Version 2 that is associated with clones from Version 1
File 1
File 2
File 3
File 1
File 2
File 3
Version 1
Clone Detection Tool
Clone 1
Clone 2
Clone 3
Clone Group 1
Clone Group 2
...
Version 2
Step 1 – Detect clones in Version 1 with clone detection tool

OBSERVED CHANGES

- The figure below shows an example of the clone ranges reported by five tools (i.e., CCFinder, CloneDR, Deckard, Simian, and SimScan) for statements refactored between versions 4.0.5 and 4.2.0 of JBoss in the `EjbJarDDObjectFactory` class.
- The clone range detected by each tool is marked by the number assigned in Table 1.
- The refactoring is represented by the sequence of deleted lines (i.e., identified with '-') being replaced by the sequence of added lines (i.e., identified with '+').

```
1 2   4 5   protected String getValue(String name, String value) {
1 2   4 5     if (value.startsWith("${") && value.endsWith("}")) {
1 2 3 4 5 -     try {
1 2 3 4 5 -       String propertyName = value.substring(2, value.length()-1);
1 2 3 4 5 -       ObjectName propertyServiceON = new ObjectName("...");
1 2 3 4 5 -       KernelAbstraction kernelAbstraction = KernelAbstractionFactory.getInstance();
1 2 3 4 5 -       String propertyValue = (String)kernelAbstraction.invoke(...);
1 2 3   5 -       log.debug("Replaced ejb-jar.xml element " + name + " with value " + propertyValue);
1 2 3   5 -       return propertyValue;
1 2 3   5 -     } catch (Exception e) {
1 2 3   5 -       log.warn("Unable to look up property service for ejb-jar.xml element " + ...);
1 2 3   5 -     }
          +     String replacement = StringPropertyReplacer.replaceProperties(value);
          +     if (replacement != null)
          +       value = replacement;
1 2     5     }
1 2     5     return value;
1 2     5   }
```

Red text represent deleted code
Green text represent added code

SimScan
Simian
Deckard
CloneDR
CCFinder

Three tools (CCFinder, CloneDR, and SimScan) reported the entire method. Simian reported the method's signature and its first six lines. Only Deckard reported the exact code range that was actually refactored.

REFACTORINGS IN JBOSS

- Table 1 documents clone range coverages by the five tools for 21 *Extract Method*-type refactorings in JBoss (ver. 2.2.0–4.2.3) related to clones originally detected by Simian.
- Clone ranges that exactly covered the refactored code ranges account for less than half of the instances for each tool.
- Clone ranges that represent a larger coverage of the actual refactored code range occurred several times.
- These observations suggest refactoring on only part of the reported clone, what we call ***sub-clone refactoring***.

Table 1. Coverage of Refactorings

Tool	Exact Coverage	Larger Coverage
1. CCFinder	4	8
2. CloneDR	6	9
3. Deckard	8	3
4. Simian	2	0
5. Simscan	6	12

REFACTORINGS IN DECKARD CLONES

- Clone-related refactorings were observed for ArgoUML, Apache Derby, and JBoss using the Deckard clone detection tool, as Deckard gave the best results from the previous study (Table 1).

Table 2. Coverage of Refactorings of Deckard Clones

Artifact	Versions	Exact Coverage	Sub-clone Coverage
ArgoUML	0.10.1–0.26	17	**9**
Derby (Apache)	10.1.1–10.5.3	12	**15**
JBoss	2.2.0 – 4.2.3	19	**14**

- In Table 2, the total instances of sub-clone refactoring were comparable to the instances of clones representing the exact coverage of the refactored code.

SUB-CLONE REFACTORING PROPERTIES

- Refactorings related to the "Sub-clone Coverage" instances in Table 2 were further evaluated to determine characteristics that may have influenced sub-clone refactoring.

Table 3. Sub-clone Refactoring Properties

Property`		JBoss	ArgoUML	Derby
Coverage Levels	Same level	4	4	6
	1 level above	9	2	8
	> 1 level above	1	3	1
Clone Differences	Refactorable	7	4	8
	Not Refactorable	7	5	7

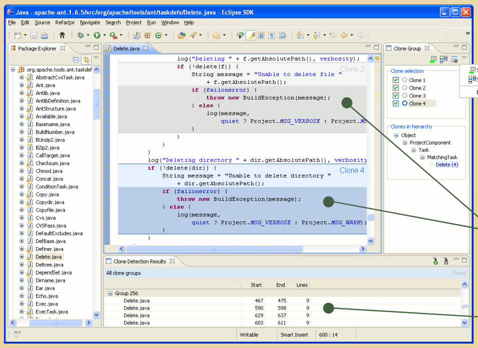
- *Deckard results* – Deckard includes smaller clone groups of the maximal sized groups allowing more exact ranges to be found.
- Table 3 (Coverage Levels) shows that in most cases in JBoss and Derby refactoring was still performed under the syntactic level of the reported clone.

```
 if (edge instanceof MTransition) {
    MTransition tr = (MTransition) edge;
-   FigTrans trFig = new FigTrans(tr);
-   // set source and dest
-   // set any arrowheads, labels, or colors
-   MStateVertex sourceSV = tr.getSource();
-   MStateVertex destSV = tr.getTarget();
-   FigNode sourceFN = (FigNode) lay...
-   FigNode destFN = (FigNode) lay...
-   trFig.setSourcePortFig(sourceFN);
-   trFig.setSourceFigNode(sourceFN);
-   trFig.setDestPortFig(destFN);
-   trFig.setDestFigNode(destFN);
+   FigTrans trFig = new FigTrans(tr, lay);
    return trFig;
  }
```

- *Excluded statements* – In Table 3 (Coverage Levels), JBoss and ArgoUML consisted of four instances of clones being at the same level as the refactored code. Apache Derby consisted of six instances.
- In these cases, some statements were not included as part of the refactoring. An example can be seen on the left, where the first and last statements in the *If-block* were not refactored although they were part of the clone.
- *Clone Differences* – We consider the scenario of selecting the entire clone for refactoring rather than the sub-clone to determine if the entire clone could have been refactored.
- In Table 3 (Clone Differences), approximately half of instances showed that the entire clone could have been refactored, but the programmer still decided to refactor only the sub-clone.

CONCLUSION

- This poster has described an analysis of code clone refactoring by considering the scenario where a clone detection tool was used to automatically identify the clones.
- Based on our evaluation, the refactoring of parts of clones (i.e., sub-clones) is evident and we conclude that there is a need to consider instances of sub-clone refactoring.
- Based on the evaluation of sub-clones and their related refactorings, a mechanism that can select sub-clones for refactorings should allow a programmer to:
  - Select a sub-clone that is one or more syntactic levels below the main clone.
  - Be able to include/exclude bordering statements.
- We are incorporating sub-clone refactoring support in our Eclipse plug-in called CeDAR (Clone Detection, Analysis, and Refactoring), shown below.



```
            log("Deleting " + f.getAbsolutePath(), verbosity);
            if (!delete(f)) {
                String message = "Unable to delete file "
                    + f.getAbsolutePath();
                if (failonerror) {
                    throw new BuildException(message);
                } else {
                    log(message,
                        quiet ? Project.MSG_VERBOSE : Project.M
                }
            }
        }
    }
    log("Deleting directory " + dir.getAbsolutePath(), verbosity
    if (!delete(dir)) {
        String message = "Unable to delete directory "
            + dir.getAbsolutePath();
        if (failonerror) {
            throw new BuildException(message);
        } else {
            log(message,
                quiet ? Project.MSG_VERBOSE : Project.MSG_WARN);
        }
    }
```

	Start	End	Lines
Group 256			
Delete.java	467	475	9
Delete.java	590	598	9
Delete.java	629	637	9
Delete.java	603	611	9

Clone group can be refactored at the same time.

A sub-clone can be selected inside of the originally reported clone range.

Clones obtained from clone detection tool results.

This material is based upon work supported by the National Science Foundation under Grant No. 0702764.