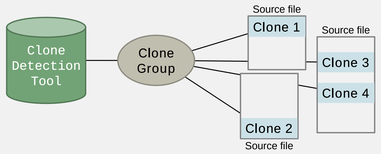
Centralizing Clone Group Representation and Maintenance

Robert Tairas • **tairasr@cis.uab.edu** • **http://www.cis.uab.edu/tairasr/**

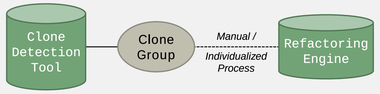
Software Composition and Modeling Laboratory
Department of Computer and Information Sciences
University of Alabama at Birmingham

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

Motivation



- After clone detection, a programmer must observe each clone, which can be scattered in multiple files.
- This motivates the need to provide a *centralized representation* of the clones that displays the properties of each clone and the relationships among them.



- IDE's such as Eclipse provide mechanisms to perform refactoring, but support for refactoring *all clones at once* is still limited.
- This motivates the need to extend the refactoring capabilities *where clones are maintained together in a centralized process*.

Poster Overview

- This poster introduces an Eclipse plug-in called CeDAR (Clone Detection, Analysis, and Refactoring) to demonstrate the benefits of centralizing clone groups for representation and maintenance.

Sample Clone Group

- Four clones detected in Apache-Ant version 1.6.5.
- Clones differ in string values and variable names (i.e., parameterized clones).

```
if (!delete(file)) {
    String message = "Unable to delete file "
        + file.getAbsolutePath();
    if (failonerror) {
        throw new BuildException(message);
    } else {
        log(message, quiet ? Project.MSG_VERBOSE
            : Project.MSG_WARN);
    }
}
```
Clone 1

```
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.MSG_WARN);
    }
}
```
Clone 2

```
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.MSG_WARN);
    }
}
```
Clone 3

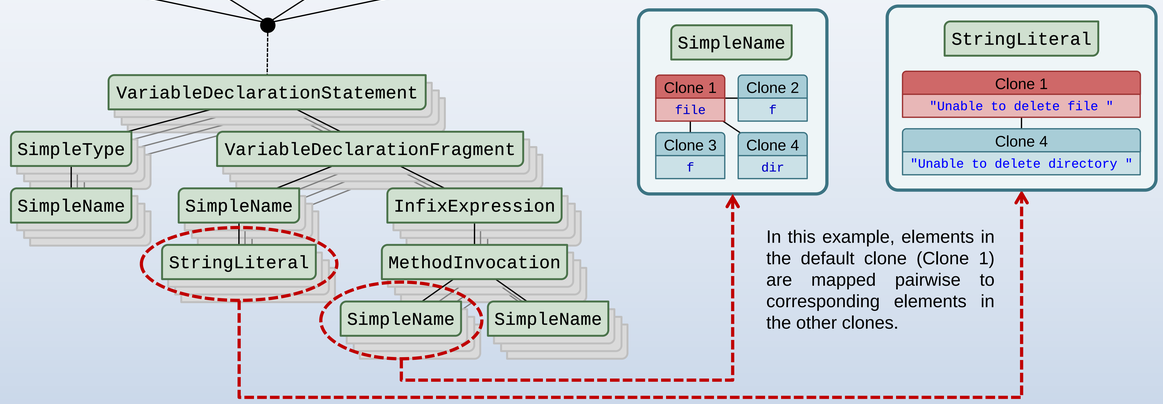
```
if (!delete(dir)) {
    String message = "Unable to delete directory "
        + dir.getAbsolutePath();
    if (failonerror) {
        throw new BuildException(message);
    } else {
        log(message, quiet ? Project.MSG_VERBOSE
            : Project.MSG_WARN);
    }
}
```
Clone 4

Detecting Parameterized Elements

- Detection is performed on a clone group selected by the user.
- AST nodes of one "default" clone are compared with the AST nodes of the remaining clones in the group.
- Differing nodes between the default clone and compared clone are stored as pair-wise relationships.

Current Capabilities

- Can detect nodes of the same type, but with different properties (e.g., variable name or string difference).
- Can detect uses of simple and complex identifiers (e.g., one clone uses a simple variable, another clone uses a method call).
- Clones containing differences in statements are currently not supported, which will require more complex refactoring techniques.



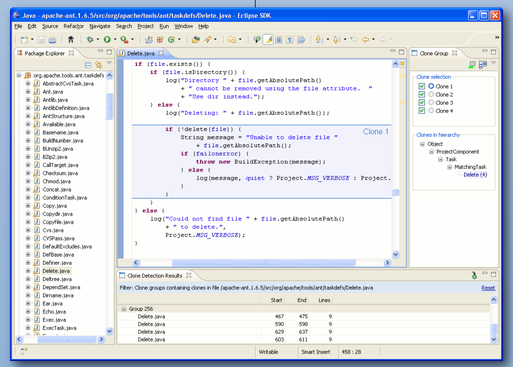
In this example, elements in the default clone (Clone 1) are mapped pairwise to corresponding elements in the other clones.

CeDAR Plug-in

- CeDAR can parse reports from several clone detection tools.
- General clone information from these tools forms the input of the plug-in (i.e., clone location and clone groupings).

Clone detection tools

CloneDR | Deckard | SimScan



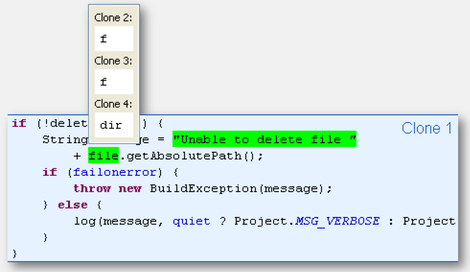
Eclipse plug-in view

Clone Group Representation

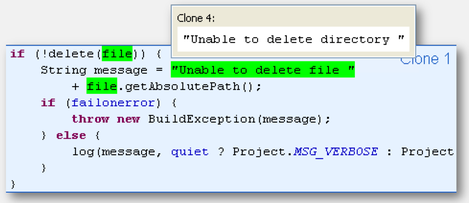
- The detected parameterized elements serve as input to the display of the differences among the clones.
- Parameterized elements are highlighted and alternative values are displayed when moused over.

```
if (!delete(file)) {
    String message = "Unable to delete file "
        + file.getAbsolutePath();
    if (failonerror) {
        throw new BuildException(message);
    } else {
        log(message, quiet ? Project.MSG_VERBOSE : Project
    }
}
```
Clone 1

Parameterized elements highlighted



Hovering over variable `file`



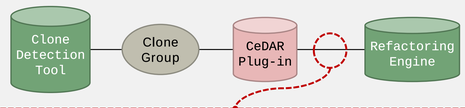
Hovering over string "`Unable to delete file `"

Contribution

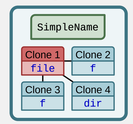
- Clone representation is centralized in one location, which allows the programmer to learn about the clones without the need to open every occurrence of the clone in the file or class system.
- As it relates to removing the duplication, the representation can provide a quick summary of the complexity of the parts of the clones that differ.

Clone Group Maintenance

- Clone information of a selected clone group is passed to the refactoring engine through CeDAR.
- The IDE's internal detection of renamed variables is replaced by the results from a clone detection tool.



The AST nodes representing the clones are passed to identify the sections of code for refactoring.



The detected parameterized elements are utilized to determine, for example in *Extract Method*, what to pass to the newly created method.

Clones replaced by calls to new method

```
dealWithError(file, "Unable to delete file ");
dealWithError(f, "Unable to delete file ");
dealWithError(f, "Unable to delete file ");
dealWithError(dir, "Unable to delete directory ");
```

New extracted method

```
private void dealWithError(File file, String stringVar0) {
    if (!delete(file)) {
        String message = stringVar0
            + file.getAbsolutePath();
        if (failonerror) {
            throw new BuildException(message);
        } else {
            log(message, quiet ? Project.MSG_VERBOSE : Project.MSG_WARN);
        }
    }
}
```

Contribution

- By connecting the results of clone detection tools and extending the refactoring capabilities of Eclipse, maintenance on a group of clones can be done simultaneously in a centralized way.