CoCloRep: A DSL for Code Clones

Robert Tairas

with Shih-Hsi Liu, Frédéric Jouault, and Jeff Gray

Workshop on Software Language Engineering
MoDELS 2007, Nashville, TN

This project is supported by
NSF grant CPA-0702764 and the OpenEmbeDD project
Code Clones

- Code clone: a sequence of statements that are duplicated at multiple locations in a program
Clones in Source Code

- Origin: copy-and-paste parts of code from one location to another
  - The copied code already works correctly
  - No time to be efficient

- Research shows that 6-8% of large-scale application code are clones (Jiang, 2006)
Background: Clone Detection Tools

- **String**
  - (Johnson, 1994), (Ducasse, 1999)

- **Token**
  - Dup (Baker, 1996), CCFinder (Kamiya, 2002), (Basit, 2007)

- **Abstract Syntax Tree**
  - CloneDR (Baxter, 1998), (Koschke, 2006), (Evans, 2007)

- **Program Dependence Graph**
  - (Komondoor, 2001), (Krinke, 2001)

- **Metrics**
  - (Mayrand, 1996), (Kontogiannis, 1997)
The detection stage can report a lot of clones

What to do with the clones
- Keep them as-is
- Refactor

Clone analysis in the Grammarware space
- Metrics based on variable properties and distance (Higo, 2004)
- Relationships in the class hierarchy (Golombingi, 2001)
- Method-level clone grouping (Balazinska, 2000)
Related Work: Clones as Models

- **Clone Region Descriptors**
  - Representation of clones for future monitoring and modification (Duala-Ekoko, 2007)

- **Clone-Pair Model**
  - Relationship of each clone pair in the group (Giesecke, 2006)

- **Regional Group of Clones**
  - Based on software entity locations used for determining structural relationships (Kapser, 2006)
Goal

- Representation and analysis of clones using Model-Driven Engineering (MDE)
  - Representation of clones as models via a Domain-Specific Language (DSL)
  - Analysis of these models through model transformations
Types of Clones

Original code

```c
int main() {
    int x = 1;
    int y = x + 5;
    return y;
}
```

Exact match

```c
int func1() {
    int x = 1;
    int y = x + 5;
    return y;
}
```

Exact match with differing variable names

```c
int func2() {
    int p = 1;
    int q = p + 5;
    return q;
}
```

Near exact match

```c
int func3() {
    int s = 1;
    int t = s + 5;
    s++;
    return t;
}
```

Types of Clones

Original code

```c
int main() {
  int x = 1;
  int y = x + 5;
  return y;
}
```

```
int func1() {
  int x = 1;
  int y = x + 5;
  return y;
}
```

Exact match

```
int func2() {
  int p = 1;
  int q = p + 5;
  return q;
}
```

Exact match with differing variable names

```
int func3() {
  int s = 1;
  int t = s + 5;
  s++;
  return t;
}
```

Near exact match

Types of Clones

**Original code**

```c
int main() {
    int x = 1;
    int y = x + 5;
    return y;
}
```

**Exact match**

```c
int func1() {
    int x = 1;
    int y = x + 5;
    return y;
}
```

**Exact match with differing variable names**

```c
int func2() {
    int p = 1;
    int q = p + 5;
    return q;
}
```

**Near exact match**

```c
int func3() {
    int s = 1;
    int t = s + 5;
    s++;
    return t;
}
```

Types of Clones

Original code

```c
int main() {
    int x = 1;
    int y = x + 5;
    return y;
}
```

- **Exact match**
  ```c
  int func1() {
    int x = 1;
    int y = x + 5;
    return y;
  }
  ```

- **Exact match with differing variable names**
  ```c
  int func2() {
    int p = 1;
    int q = p + 5;
    return q;
  }
  ```

- **Near exact match**
  ```c
  int func3() {
    int s = 1;
    int t = s + 5;
    s++;
    return t;
  }
  ```

Clone group: a set of clones representing the same duplication of code
- A clone group contains the commonalities of the clones
First DSL: Clones

-- clone 1

1: int g;
2: int f = g + 3;
3: i = i + 1;
4: c = f + m;

-- clone 2

1: int q;
2: int p = q + 3;
3: c = p + m;

-- clone instances

1: instance r = cg(f, g) {
2:   t {
3:     i = i + 1;
4:   }
5: };
6: 
7: instance s = cg(p, q);

-- clone group

1: clone cg($a, $b) {
2:   int $b;
3:   int $a = $b + 3;
4:   {{ t }}
5:   c = $a + m;
6: }
# First DSL: Clones

<table>
<thead>
<tr>
<th>-- clone 1</th>
<th>-- clone 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: int g;</td>
<td>1: int q;</td>
</tr>
<tr>
<td>2: int f = g + 3;</td>
<td>2: int p = q + 3;</td>
</tr>
<tr>
<td>3: i = i + 1;</td>
<td>3: c = p + m;</td>
</tr>
<tr>
<td>4: c = f + m;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>-- clone instances</th>
<th>-- clone group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: instance r = cg(f, g) {</td>
<td>1: clone cg($a, $b) {</td>
</tr>
<tr>
<td>2: t {</td>
<td>2: int $b;</td>
</tr>
<tr>
<td>3: i = i + 1;</td>
<td>3: int $a = $b + 3;</td>
</tr>
<tr>
<td>4: }</td>
<td>4: {{ t }}</td>
</tr>
<tr>
<td>5: };</td>
<td>5: c = $a + m;</td>
</tr>
<tr>
<td>6:</td>
<td>6: }</td>
</tr>
<tr>
<td>7: instance s = cg(p, q);</td>
<td></td>
</tr>
</tbody>
</table>
First DSL: Clones

```
-- clone 1
1: int g;
2: int f = g + 3;
3: i = i + 1;
4: c = f + m;

-- clone 2
1: int q;
2: int p = q + 3;
3: c = p + m;

-- clone instances
1: instance r = cg(f, g) {
2:   t {
3:     i = i + 1;
4:   }
5: }
6: }
7: instance s = cg(p, q);

-- clone group
1: clone cg($a, $b) {
2:   int $b;
3:   int $a = $b + 3;
4:   {{ t }}
5:   c = $a + m;
6: }
```
First DSL: Clones

-- clone 1
1: int g;
2: int f = g + 3;
3: i = i + 1;
4: c = f + m;

-- clone 2
1: int q;
2: int p = q + 3;
3: c = p + m;

-- clone instances
1: instance r = cg(f, g) {
2:   t {
3:     i = i + 1;
4:   }
5: }
6:
7: instance s = cg(p, q);

-- clone group
1: clone cg($a, $b) {
2:   int $b;
3:   int $a = $b + 3;
4:   {{ t }}
5:   c = $a + m;
6: }

1: int g;
2: int f = g + 3;
3: i = i + 1;
4: c = f + m;
1: int q;
2: int p = q + 3;
3: c = p + m;
First DSL: Clones

-- clone 1
1: int g;
2: int f = g + 3;
3: i = i + 1;
4: c = f + m;

-- clone 2
1: int q;
2: int p = q + 3;
3: c = p + m;

-- clone instances
1: instance r = cg(f, g) {
2:   t {
3:     i = i + 1;
4:   }
5: };
6: }
7: instance s = cg(p, q);

-- clone group
1: clone cg($a, $b) {
2:   int $b;
3:   int $a = $b + 3;
4:   {{ t }}
5:   c = $a + m;
6: }

Second DSL: Commands

- **Input:**

  ```
  variables cg;
  ```

- **Output:**

  1: Variable information for clone group cg
  2: Declared variables:
  3:   b
  4:   a
  5: Outside assigned variables:
  6:   c
  7:   i (in instance r)
  8: Outside non-assigned variables:
  9:   m
Model Transformation Process

variables command

M1 = terminal model level; M2 = metamodel level; M3 = meta-metamodel level
Model Transformation Process

variables command

-- clone instances
instance r = cg(f, g) {
    t {
        i = i + 1;
    }
}

instance s = cg(p, q);

-- clone group
clone cg($a, $b) {
    int $b;
    int $a = $b + 3;
    {{ t }}
    c = $a + m;
}

M1 = terminal model level; M2 = metamodel level; M3 = meta-metamodel level
Model Transformation Process

variables command

Model-Driven Engineering (MDE) Technical Space (TS)

M1 = terminal model level; M2 = metamodel level; M3 = meta-metamodel level
Model Transformation Process

variables command

Model-Driven Engineering (MDE)
Technical Space (TS)

M1 = terminal model level; M2 = metamodel level; M3 = meta-metamodel level
Model Transformation Process

variables command

1: Variable information for clone group cg
2: Declared variables:
3:   b
4:   a
5: Outside assigned variables:
6:   c
7:   i (in instance r)
8: Outside non-assigned variables:
9:   m
Model Transformation Process

variables command

M1 = terminal model level; M2 = metamodel level; M3 = meta-metamodel level
AMMA Platform
(ATLAS Model Management Architecture)

Platform for defining and transforming models through the following DSL’s:

- KM3: Abstract syntax
- TCS: Concrete syntax
- ATL: Transformations

Related Tutorial at MoDELS 2007:
“Putting MDA to Work on Eclipse with the AMMA Tool Suite”
Tuesday afternoon (October 2)
-- Root

class Root extends LocatedElement {
    reference cloneGroup[*] ordered container : CloneGroup;
    reference cloneInstance[*] ordered container : CloneInstance;
}

-- Clone Group

class CloneGroup extends LocatedElement {
    attribute cloneName : String;
    reference parameters[*] ordered container : Variable;
    reference statements[*] ordered container : Statement;
}

-- Clone Instance

class CloneInstance extends LocatedElement {
    attribute instanceName : String;
    reference cloneName : CloneGroup;
    reference arguments[*] ordered container : Variable;
    reference boxes[*] ordered container : Box;
}
Abstract Syntax in KM3 (snippet)

1:    -- Root
2:    class Root extends LocatedElement {
3:        reference cloneGroup[*] ordered container : CloneGroup;
4:        reference cloneInstance[*] ordered container : CloneInstance;
5:    }
6:
7:    -- Clone Group
8:    class CloneGroup extends LocatedElement {
9:        attribute cloneName : String;
10:       reference parameters[*] ordered container : Variable;
11:       reference statements[*] ordered container : Statement;
12:    }
13:
14:    -- Clone Instance
15:    class CloneInstance extends LocatedElement {
16:        attribute instanceName : String;
17:        reference cloneName : CloneGroup;
18:        reference arguments[*] ordered container : Variable;
19:        reference boxes[*] ordered container : Box;
20:    }
21
27

```plaintext
-- clone group
cg $a, $b {
    int $b;
    int $a = $b + 3;
    {{ t }}
    c = $a + m;
}  
```
Abstract Syntax in KM3 (snippet)

```
class Root extends LocatedElement {
    reference cloneGroup[*] ordered container : CloneGroup;
    reference cloneInstance[*] ordered container : CloneInstance;
}

class CloneGroup extends LocatedElement {
    attribute cloneName : String;
    reference parameters[*] ordered container : Variable;
    reference statements[*] ordered container : Statement;
}

class CloneInstance extends LocatedElement {
    attribute instanceName : String;
    reference cloneName : CloneGroup;
    reference arguments[*] ordered container : Variable;
    reference boxes[*] ordered container : Box;
}

-- clone instances
instance r = cg(f, g) {
    t {
        i = i + 1;
    }
};
instance s = cg(p, q);
```
Concrete Syntax in TCS (snippet)

1:    -- Root
2:    template Root main
3:      :  cloneGroup cloneInstance
4:      ;
5:      
6:    -- Clone Group
7:    template CloneGroup context addToContext
8:      :  "clone" cloneName "(" parameters{separator = ","} ")"
9:        "{" statements "}"
10:      ;
11:      
12:    -- Clone Instance
13:    template CloneInstance context addToContext
14:      :  "instance" instanceName "=" cloneName{refersTo = cloneName}
15:        "(" arguments{separator = ","} ")"
16:        (isDefined (boxes) ? "{" boxes{separator = ","} "}" ) ";
17:      ;
Tranformation in ATL (snippet)

1:  lazy rule DeclaredVar {
2:    from s : Clones!CloneGroup
3:    to t   : Variables!DeclaredVar {
4:      variables <- Sequence {
5:        s.statements->collect(e |
6:          if e.oclIsKindOf(Clones!DeclarationStat) then
7:            e.variable.varName
8:          else
9:            Sequence{}
10:        endif
11:      ),
12:      thisModule.processBox1(s.cloneName)
13:    } } }
14: }
15: }

-- clone group
clone cg($a, $b) {
  int $b;
  int $a = $b + 3;
  {{ t }}
  c = $a + m;
}
Tranformation in ATL (snippet)

```java
1:  helper def: processBox1(t: String): Sequence(OclAny) =
2:    Clones!CloneInstance.allInstancesFrom('IN1')
3:      ->select(e | e.cloneName.cloneName = t)
4:      ->iterate(f;
5:        test : Sequence(OclAny) = Sequence{} | test->including(
6:          if f.boxes->size() = 0 then
7:            Sequence{}
8:          else
9:            f.boxes->collect(g | g.statements)->flatten()
10:          ->collect(h |
11:            if h.oclIsKindOf(Clones!DeclarationStat) then
12:              h.variable.varName + ' (in ' + f.instanceName + ')
13:            else
14:              Sequence{}
15:            endif
16:        )
17:      )
18:    );
19:    -- clone instances
20:    instance r = cg(f, g) {
21:      t {
22:        i = i + 1;
23:      }
24:    };
25:    instance s = cg(p, q);
```
Summary

Initial effort of representation and analysis of clones using MDE:

- **Representation of clones (as models)**
  - Commonalities stored in clone groups
  - Uniqueness stored in clone instances

- **Analysis of clones (via model transformations)**
  - Transformations with both declarative and imperative constructs
Future Work

- Reunited
  - Instead of a separate metamodel for clone representation, consider “extending” metamodel of source program

- Comparison
  - More detailed comparisons with existing approaches
Thank you. Questions?

- Clone Detection Literature: http://www.cis.uab.edu/tairasr/clones/literature
- SoftCom Laboratory: http://www.cis.uab.edu/softcom

Robert Tairas

`tairasr@cis.uab.edu`

Software Composition and Modeling Laboratory

Department of Computer and Information Sciences
University of Alabama at Birmingham

This project is supported by NSF grant CPA-0702764 and the OpenEmbeDD project.