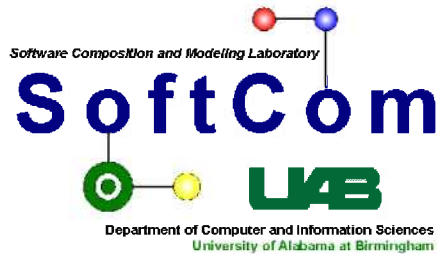


CoCloRep: A DSL for Code Clones

Robert Tairas

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



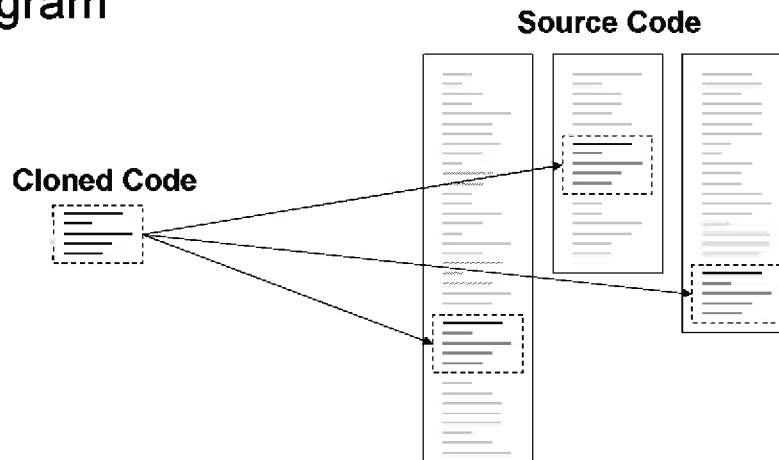
Workshop on Software Language Engineering
MoDELS 2007, Nashville, TN



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

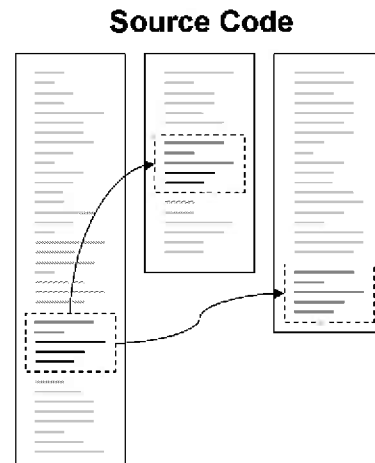
Code Clones

- 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)



3

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)

4

Background: Clone Analysis

- 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 (Golomingi, 2001)
 - Method-level clone grouping (Balazinska, 2000)

5

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 (Kapsner, 2006)

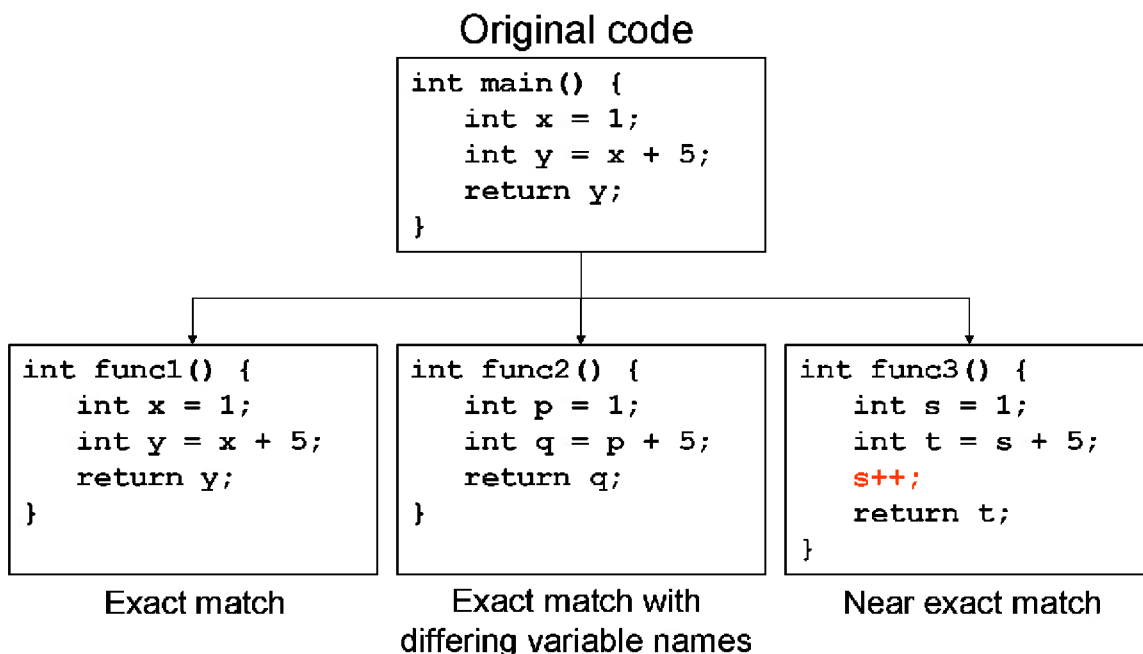
6

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

7

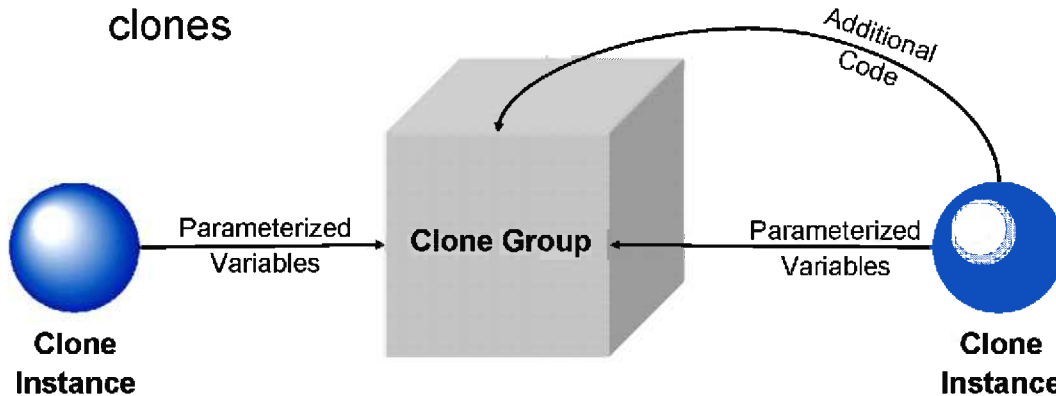
Types of Clones



8

Representing Clones

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



9

First DSL: Clones

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



```
-- clone instances          -- clone group
1: instance r = cg(f, g) {  1: clone cg($a, $b) {
2:   t {                    2:   int $b;
3:     i = i + 1;           3:   int $a = $b + 3;
4:   }                      4:   {{ t }}
5: };                       5:   c = $a + m;
6:                           6: }
7: instance s = cg(p, q);
```

10

First DSL: Clones

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



```
-- clone instances          -- clone group
1: instance r = cg(f, g) {  1: clone cg($a, $b) {
2:   t {                    2:   int $b;
3:     i = i + 1;           3:   int $a = $b + 3;
4:   }                      4:   {{ t }}
5: };                       5:   c = $a + m;
6:                           6: }
7: instance s = cg(p, q);
```

11

Second DSL: Commands

■ Input:

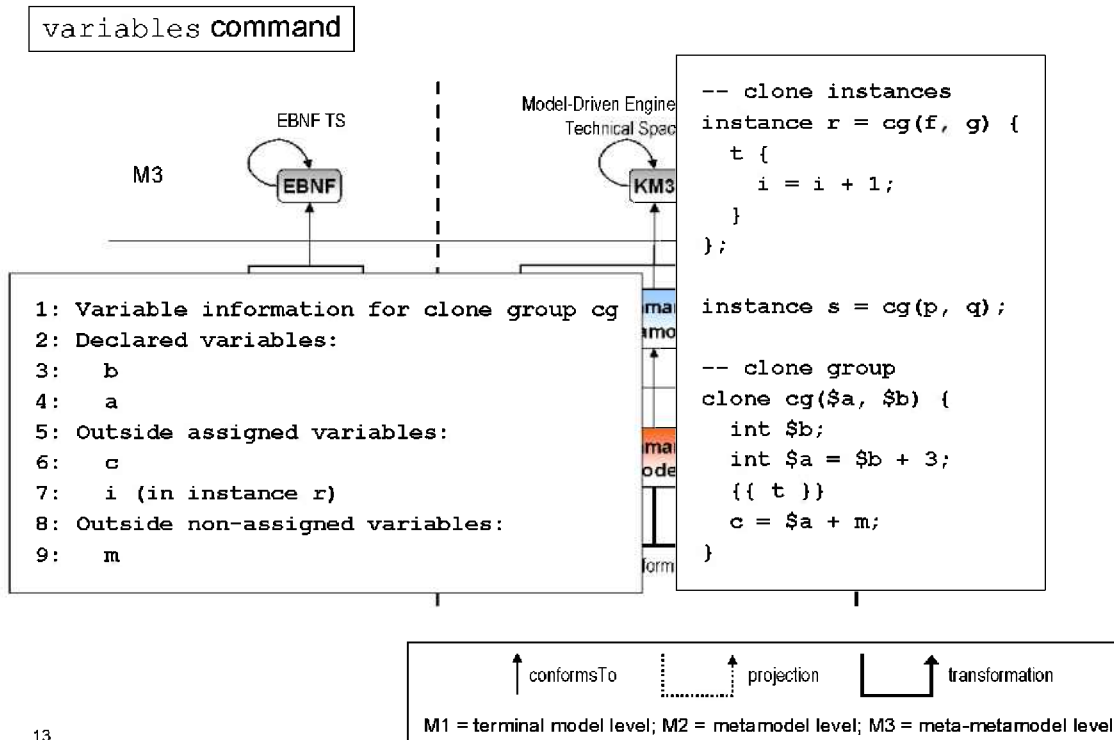
```
variables cg;
```

■ 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
```

12

Model Transformation Process



13

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)

14

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 : CloneGroup;
11:   reference statements[*] ordered container : CloneGroup;
12:  }
13:
14:  -- Clone Instance
15:  class CloneInstance extends LocatedElement {
16:    attribute instanceName : String;
17:    reference cloneName : CloneGroup;
18:    reference arguments[*] ordered container : CloneGroup;
19:    reference boxes[*] ordered container : CloneGroup;
20:  }
```

```
-- 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;
}
```

15

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:   ;
```

Transformation 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.ocIsKindOf(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;
}
```

17

Transformation in ATL (snippet)

```
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.ocIsKindOf(Clones!DeclarationStat) then
12:            h.variable.varName + ' (in ' + f.instanceName + ')'
13:          else
14:            Sequence{}
15:          endif
16:        )
17:      endif
18:    )
19:  );
```

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

instance s = cg(p, q);
```

18

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

19

Future Work

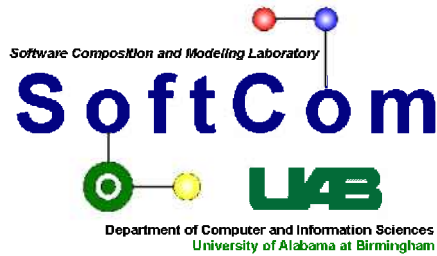
- Reunited
 - Instead of a separate metamodel for clone representation, consider “extending” metamodel of source program
- Comparison
 - More detailed comparisons with existing approaches

20

Thank you. Questions?

- CoCloRep Project: <http://www.cis.uab.edu/tairasr/coclorep>
- 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



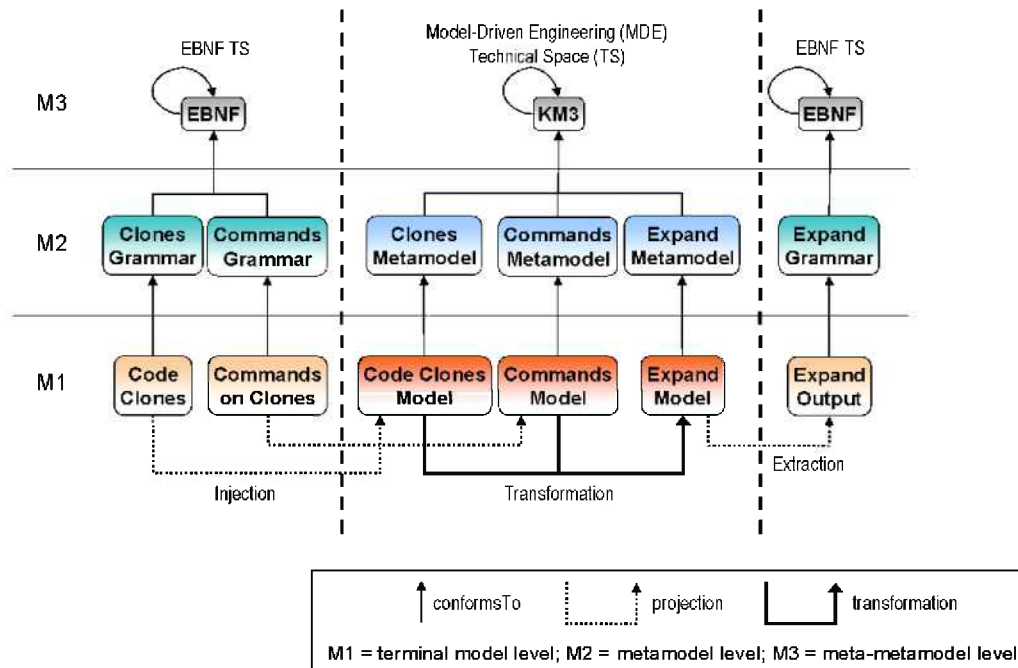
This project is supported by
NSF grant CPA-0702764 and the OpenEmBeDD project

21

Backup Slides

Model Transformation Process

expand command



23

Transformation in ATL (snippet)

```

1: helper def: processVar(t: Clones!Variable): Clones!ActualVar =
2:   if t.ocliIsKindOf(Clones!PlaceholderVar) then
3:     thisModule.PlaceHolderVar2ActualVar(t)
4:   else
5:     thisModule.ActualVar(t)
6:   endif;
...
7: lazy rule AssignStat {
8:   from s : Clones!AssignStat
9:   to t : Expand!AssignStat (
10:     variable <- thisModule.processVar(s.variable),
11:     initExp <- thisModule.processExp(s.initExp)
12:   )
13: }
...
14: lazy rule PlaceHolderVar2ActualVar {
15:   from s : Clones!PlaceholderVar
16:   to t : Expand!ActualVar (
17:     varName <- thisModule.getActualVar(s.varName)
18:   )
19: }

```

Informal Survey

- Copied a section of code (representing some type of functionality);
- Pasted it into another location;
- With the intention to come back and “clean up” the duplicate code later;
- But never did.