

Code Clones

Spiros Mancoridis[1]
Modified by Na Meng

Overview

- Definition and categories
- Clone detection
- Clone removal refactoring

2

Code Clones

- Code clone is a code fragment in source files that is identical or similar to another
- Code clones are either within a program or across different programs
- Clone pair: two clones
- Clone class: a set of fragments which are clones to each other

3

Code Clone Categorization

- Type-1 clones
 - Identical code fragments but may have some variations in whitespace, layout, and comments
- Type-2 clones
 - Syntactically equivalent fragments with some variations in identifiers, literals, types, whitespace, layout and comments

4

Code Clone Categorization

- Type-3 clones
 - Syntactically similar code with inserted, deleted, or updated statements
- Type-4 clones
 - Semantically equivalent, but syntactically different code

5

Key Points of Code Clones

- Pros
 - Increase performance
 - Code inlining vs. function call
 - Increase program readability
- Cons
 - Increase maintenance cost
 - If one code fragment contains a bug and gets fixed, all its clone peers should be always fixed in similar ways.
 - Increase code size

6

Clone Detection Strategies

- Text matching
- Token sequence matching
- Graph matching

7

Text Matching

- Older, studied extensively
- Less complex, and most widely used
- No program structure is taken into consideration
- Type-1 clones & some Type-2 clones
- Two types of text matching
 - Exact string match
 - Diff (cvs, svn, git) is based on exact text matching
 - Ambiguous match

8

Ambiguous Match

- Longest Common Subsequence match
- N-grams match

9

Token Sequence Matching

- A little more complex, less widely used
- No program structure is taken into account, either
- Type-1 and Type-2 clones
- CCFinder[2]
- CP-Miner[3]

10

CCFinder

- Step 1: Convert a program with multiple files to a single long token sequence
- Step 2: Find longest common subsequence of tokens

11

Step 1: Tokenization

```
int main(){
    int i = 0;
    static int j=5;
    while(i<20){
        i=i+j;
    }
    std::cout<<"Hello World"<<i<<std::endl;
    return 0;
}
```

Remove white spaces

12

Step 1: Tokenization

```
int main(){
int i = 0;
static int j = 5;
while(i < 20){
i=i+j;
}
std::cout << "Hello World" << endl;
return 0;
}
```

Shorten Names

13

Step 1: Tokenization

```
int main (){
int i = 0;
int j = 5;
while (i < 20){
i = i + j;
}
cout << "Hello World" << endl;
return 0;
}
```

Tokenize literals, and identifiers of types, methods, and variables.

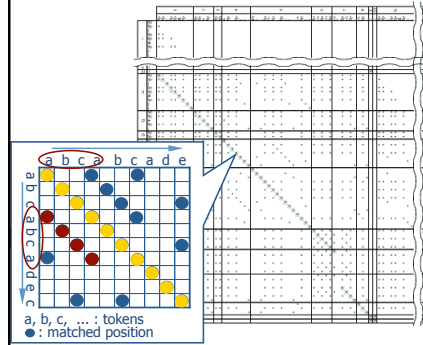
14

Step 1: Tokenization

```
$p $p(){
$p $p = $p;
$p $p = $p;
while($p < $p){
$p = $p + $p;
}
$p << $p << $p << $p;
return $p;
}
```

15

Step 2: Find Clones



16

Detected Clone Pair Example[2]

```
1. static void foo() throws RESyntaxException {
2.   String a[] = new String [] { "123,400", "abc", "orange 100" };
3.   org.apache.regexp.RE pat = new
org.apache.regexp.RE("(0-9),+");
4.   int sum = 0;
5.   for (int i = 0; i < a.length; ++i)
6.     if (pat.match(a[i]))
7.       sum += Sample.parseNumber(pat.getParen(0));
8.   System.out.println("sum = " + sum);
9. }
10. static void goo(String [] a) throws RESyntaxException {
11.   RE exp = new RE("(0-9),+");
12.   int sum = 0;
13.   for (int i = 0; i < a.length; ++i)
14.     if (exp.match(a[i]))
15.       sum += parseNumber(exp.getParen(0));
16.   System.out.println("sum = " + sum);
17. }
```

17

Limitations of CCFinder

- All files are converted into a long token sequence
 - When the program contains millions of lines of code, the tool cannot perform efficiently
- Do not take into account the natural boundary between functions and classes

18

CP-Miner[3]

- Cut the token sequences by considering basic blocks as cutting units
- Calculate a hashcode for each subsequence
- Compare hashcode sequences instead of the original token sequences

19

Graph Matching

- Newer, bleeding edge
- More complex
- Type-1, Type-2, and Type-3 clones
- Syntactic and semantic understanding
 - AST matching (ChangeDistiller)
 - CFG matching (Jdiff[4])
 - PDG matching ([5])

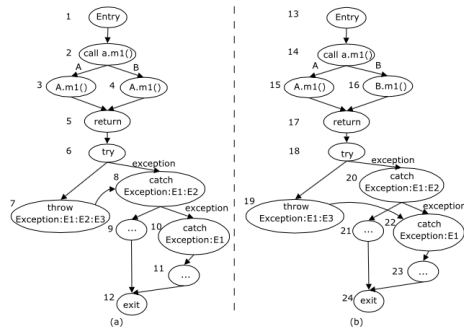
20

CFG-based Clone Detection[4]

- A Differencing Algorithm for Object-Oriented Programs
 - Match declarations of classes, fields, and methods by name
 - Match content in methods by hammock graphs
 - A hammock is a single entry, single exit subgraph of a CFG

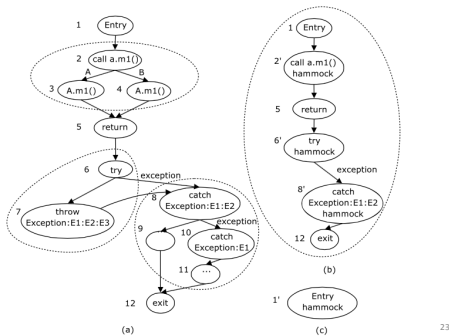
21

Example: Enhanced CFG comparison for P and P'



22

Hammock Graph Creation



23

Algorithm

- Input: hammock node n, n' , look-ahead threshold LH
- Output: set of matched pairs N
- Algorithm
 1. expand n and n' one level to graph G and G'
 2. Push start node pair $\langle s, s' \rangle$ to stack ST
 3. while ST is not empty
 4. pop $\langle c, c' \rangle$ from ST
 5. if c or c' is already matched then
 6. continue;
 7. if $\langle c, c' \rangle$ does not match then
 8. compare c with LH successors of c' or compare c' with LH successors of c until find match
 9. if a match is found then
 10. $N = N \cup \{c, c', \text{"unchanged"}\}$
 11. else
 12. $N = N \cup \{c, c', \text{"modified"}\}$
 13. push the pair's sink node pair on stack

24

Summary

- Clone detection flexibility
 - PDG > CFG|AST > Token > Text
- Cost
 - Text < Token < CFG|AST < PDG

31

Clone Removal Refactoring

- Extract method
 - Extract the common code from different methods and create a method for it
- Pull up method
 - Pull up the duplicated method to the super class, and declare a new super class if there is none

32

Extract Method

```

int main(int argc, char** argv) {
    int x = 1;
    int y = 2;
    Socket s = opensocket(x, y);
    char buf[50] = readsocket(s, 80);
    closesocket(s);

    int sz = 41;
    int px = 41;
    Socket ss = opensocket(xv, yy);
    char fib[50] = readsocket(ss, 50);
    closesocket(ss);

    return 0;
}

void getdata(int x, int y, int size, char * buf) {
    Socket s = opensocket(x, y);
    readsocket(s, buf, 80);
    closesocket(s);
}

int main(int argc, char** argv) {
    char buf[80];
    getdata(1, 2, 80, &buf);

    char fib[50];
    getdata(41, 21, 50, &fib);

    return 0;
}

```

33

Pull Up Method

```

public class Context {
    public reload() {
        if (0 > 9) { System.out.println("Too!"); }
        for (int i = 0; i < modules.length; i++) {
            modules[i].reload();
        }
    }
}

public class FileContext {
    public reload() {
        if (0 > 9) { System.out.println("Too!"); }
        for (int i = 0; i < files.length; i++) {
            files[i].reload();
        }
    }
}

public class AbstractContext {
    public reload() {
        if (0 > 9) { System.out.println("Too!"); }
        for (int i = 0; i < modules.length; i++) {
            modules[i].reload();
        }
    }
}

public class Context extends AbstractContext {
    ...
}

public class FileContext extends AbstractContext {
    ...
}

```

34

Reference

- [1] Spiros Mancoridis, Code Cloning: Detection, Classification, and Refactoring, https://www.cs.drexel.edu/~spiros/teaching/CS675/slides/code_cloning.ppt.
- [2] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue, CCFinder, A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code, TSE '02
- [3] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou, CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code, OSDI '04

35

Reference

- [4] Taweewat Apiwattanapong, Alessandro Orso, and Mary Jean Harrold, A Differencing Algorithm for Object-Oriented Programs, ASE '04
- [5] Raghavan Komondoor, Susan Horwitz, Using Slicing to Identify Duplication in Source Code, SAS '01

36