Code Clones

Spiros Mancoridis[1]
Modified by Na Meng

Overview

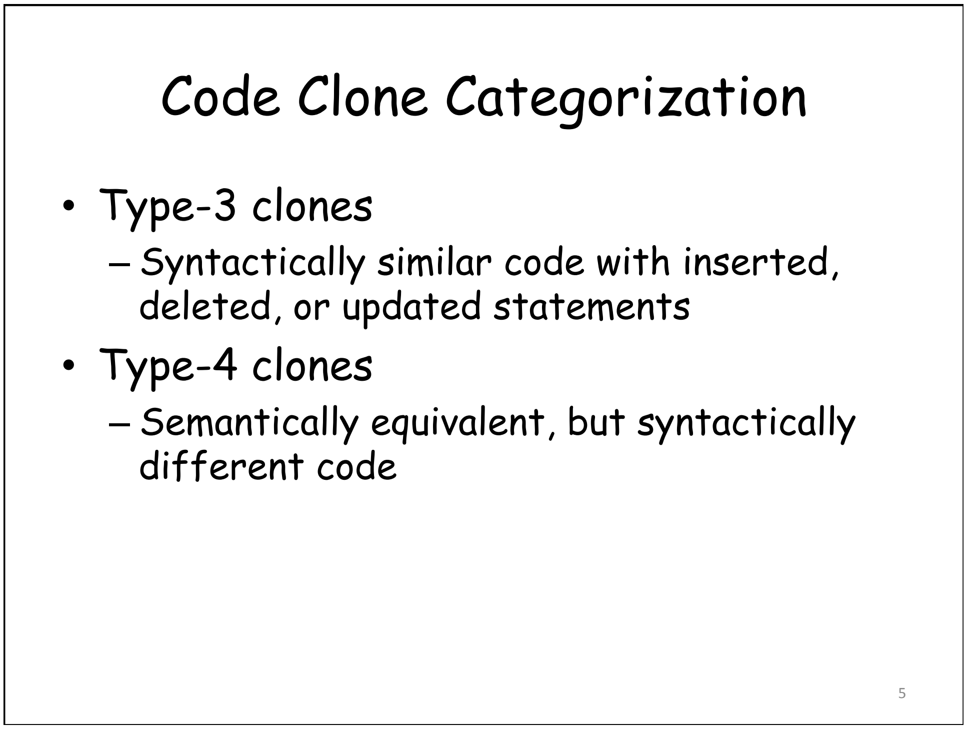
- Definition and categories
- Clone detection
- Clone removal refactoring

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

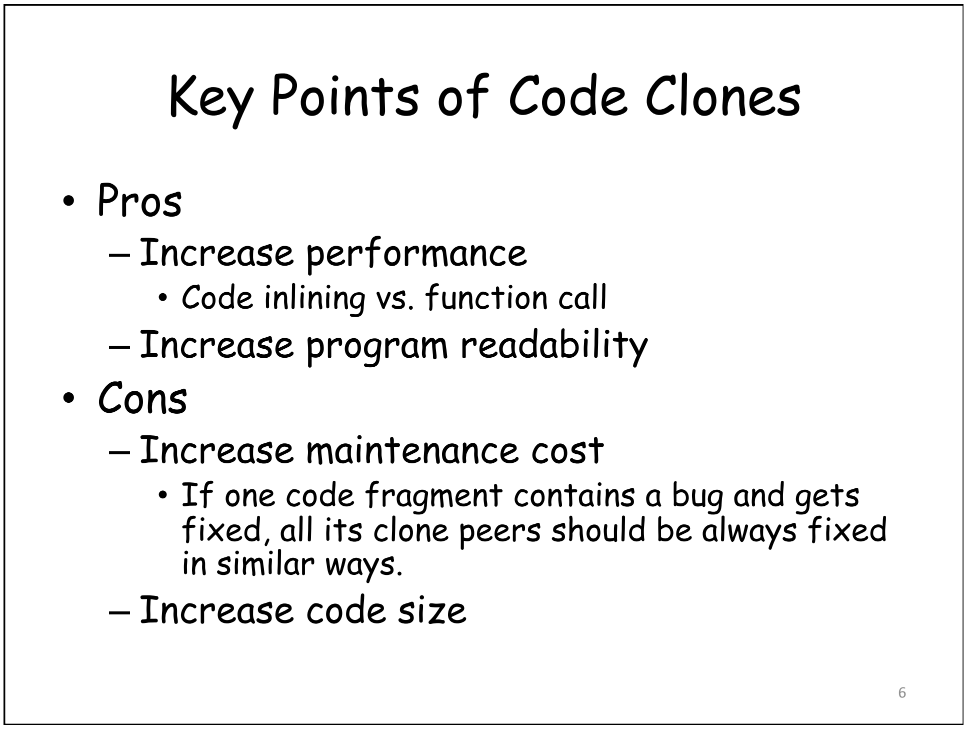
Code Clone Categorization

- 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



Code Clone Categorization

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



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

Clone Detection Strategies

- Text matching
- Token sequence matching
- Graph matching

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

Ambiguous Match

- Longest Common Subsequence match
- N-grams match

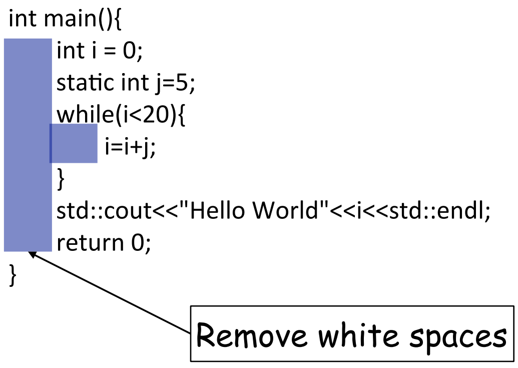
Token Sequence Matching

- 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]

CCFinder

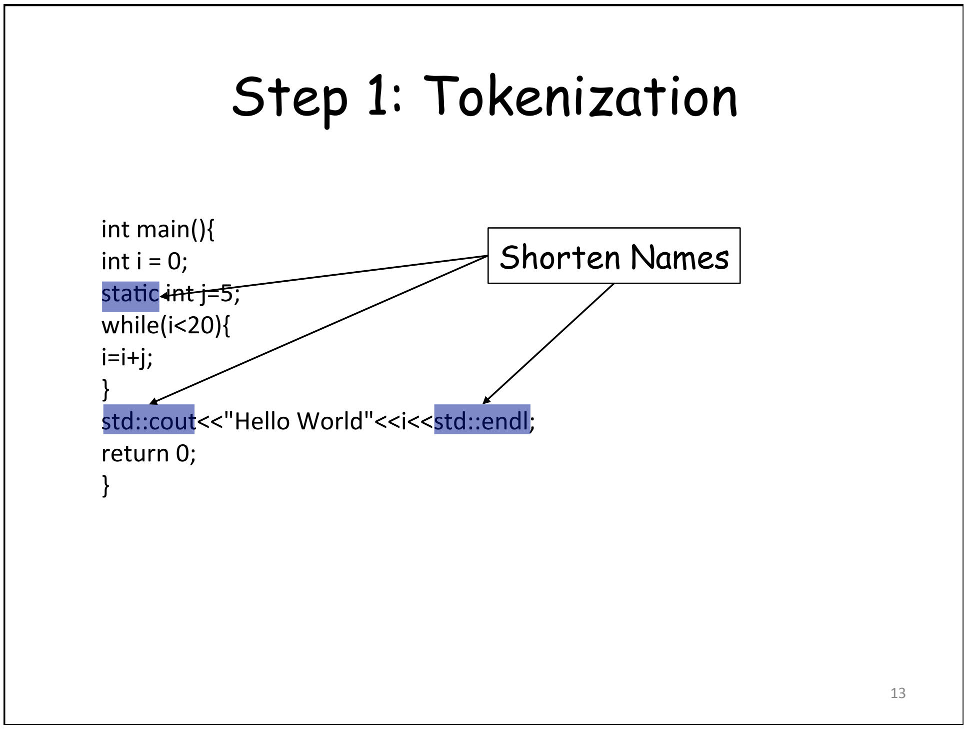
- Step 1: Convert a program with multiple files to a single long token sequence
- Step 2: Find longest common subsequence of tokens

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;
}
```

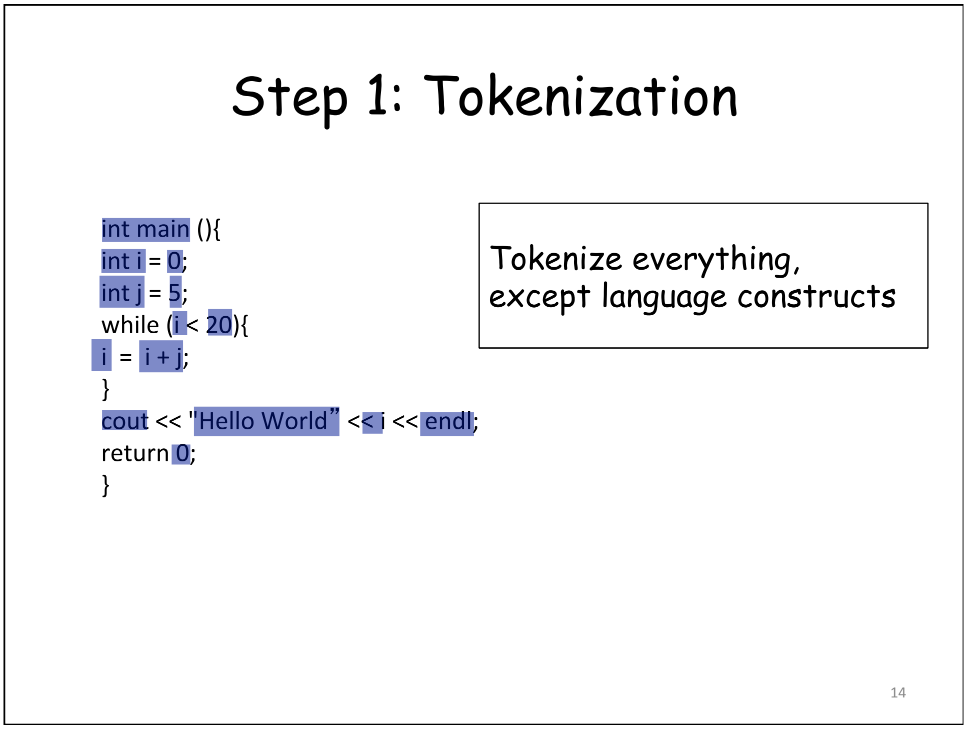
Remove white spaces



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;
}
```

Shorten Names



Step 1: Tokenization

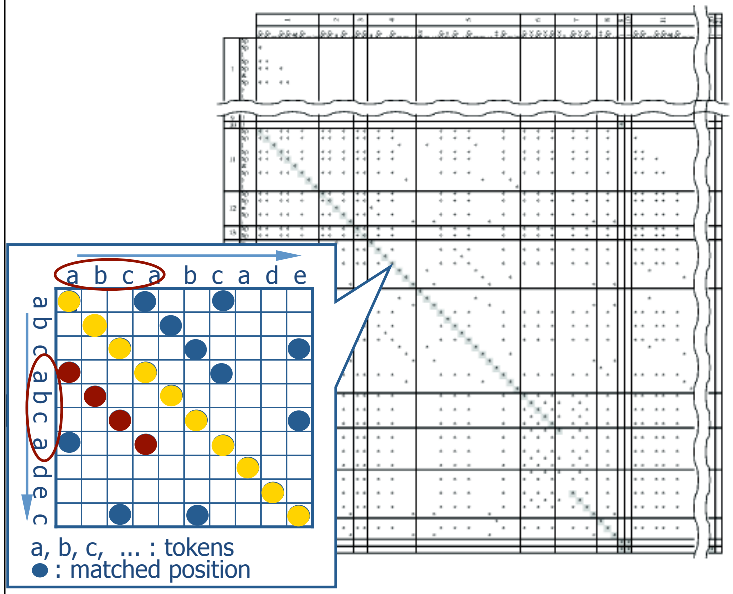
```
int main (){
int i = 0;
int j = 5;
while (i < 20){
 i  =  i + j;
}
cout << "Hello World” << i << endl;
return 0;
}
```

Tokenize everything,
except language constructs

Step 1: Tokenization

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

Step 2: Find Clones



Detected Clone Pair Example[2]

```
static void foo() throws RESyntaxException {
  String a[] = new String [] { "123,400", "abc", "orange 100" };
  org.apache.regexp.RE pat = new
org.apache.regexp.RE("[0-9,]+");
  int sum = 0;
  for (int i = 0; i < a.length; ++i)
    if (pat.match(a[i]))
      sum += Sample.parseNumber(pat.getParen(0));
  System.out.println("sum = " + sum);
}
static void goo(String [] a) throws RESyntaxException {
  RE exp = new RE("[0-9,]+");
  int sum = 0;
  for (int i = 0; i < a.length; ++i)
    if (exp.match(a[i]))
      sum += parseNumber(exp.getParen(0));
  System.out.println("sum = " + sum);
}
```

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

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

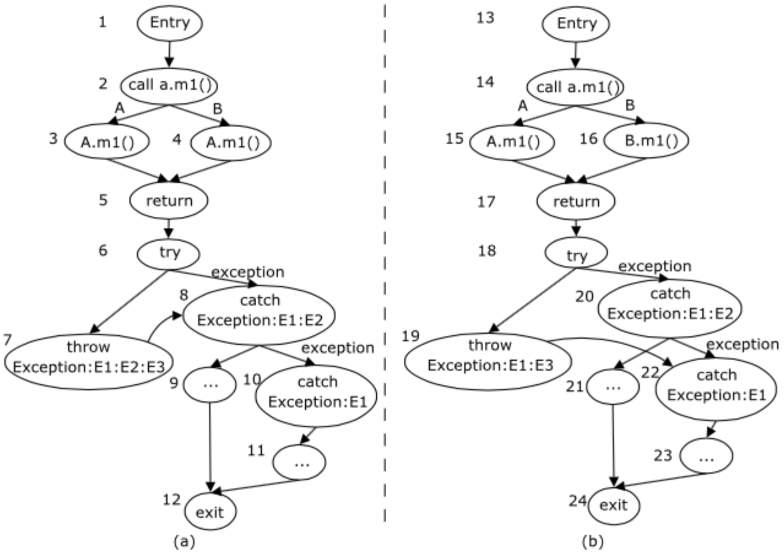
Graph Matching

- 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])

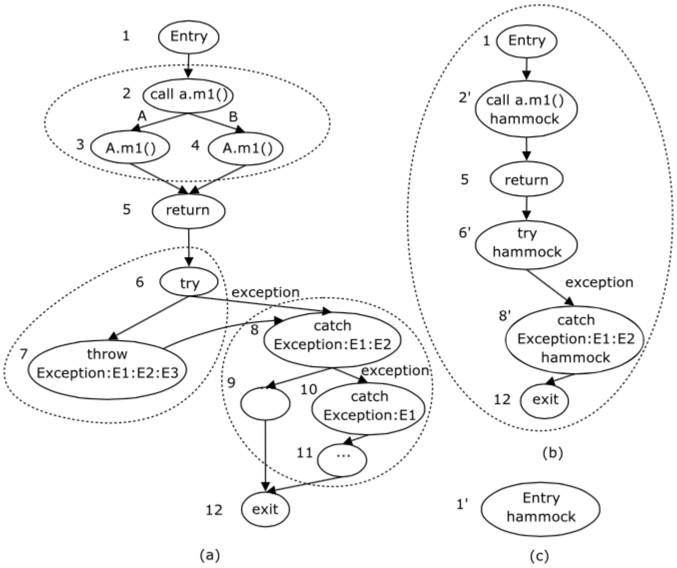
CFG-based Clone Detection[4]

- 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

Example: Enhanced CFG comparison for P and P'



Hammock Graph Creation



Algorithm

- Input: hammock node n, n', look-ahead threshold LH
- Output: set of matched pairs N
- Algorithm

```
expand n and n' one level to graph G and G'
Push start node pair <s, s'> to stack ST
while ST is not empty
    pop <c, c'> from ST
    if c or c' is already matched then
        continue;
    if <c, c'> does not match then
        compare c with LH successors of c' or
        compare c' with LH successors of c until find match
    if a match is found then
        N = N U {c, c', "unchanged"}
    else
        N = N U {c, c', "modified"}
    push the pair's sink node pair on stack
```

Observations

- The look-ahead process is like bounded LCS algorithm
  - It can tolerate statement insertions at the same level
- The algorithm starts from the outmost Hammock, so it is similar to top-down tree-differencing algorithm
- When statements are inserted at the higher level, the algorithm does not work well
  - <c, c', "modified">

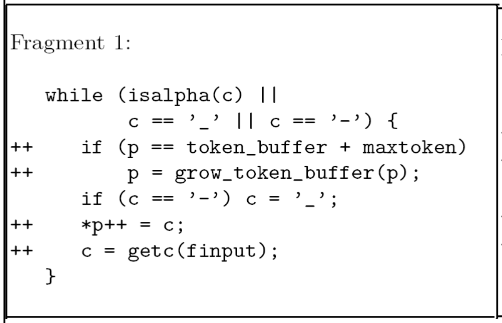
PDG-based Clone Detection [5]

- Using Slicing to Identify Duplication in Source Code
  - Step 1: Partition PDG nodes into equivalence classes based on the syntactic structure, such as while-loops
  - Step 2: For each pair of matching nodes (r1, r2), find two isomorphic subgraphs containing r1 and r2

Algorithm to Find Isomorphic Subgraphs

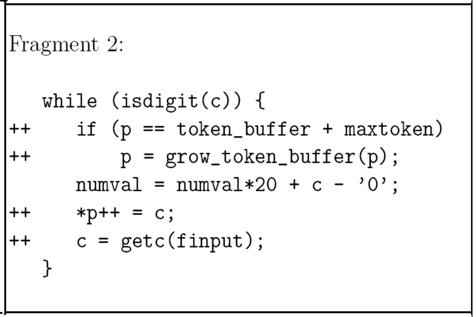
1. Start from r1 and r2, use backward slicing in lock step to add predecessors iff predecessors also match
2. If two matching nodes are loops or if-statements, forward slicing is also used to find control dependence successors (statements contained in the structure)

Example



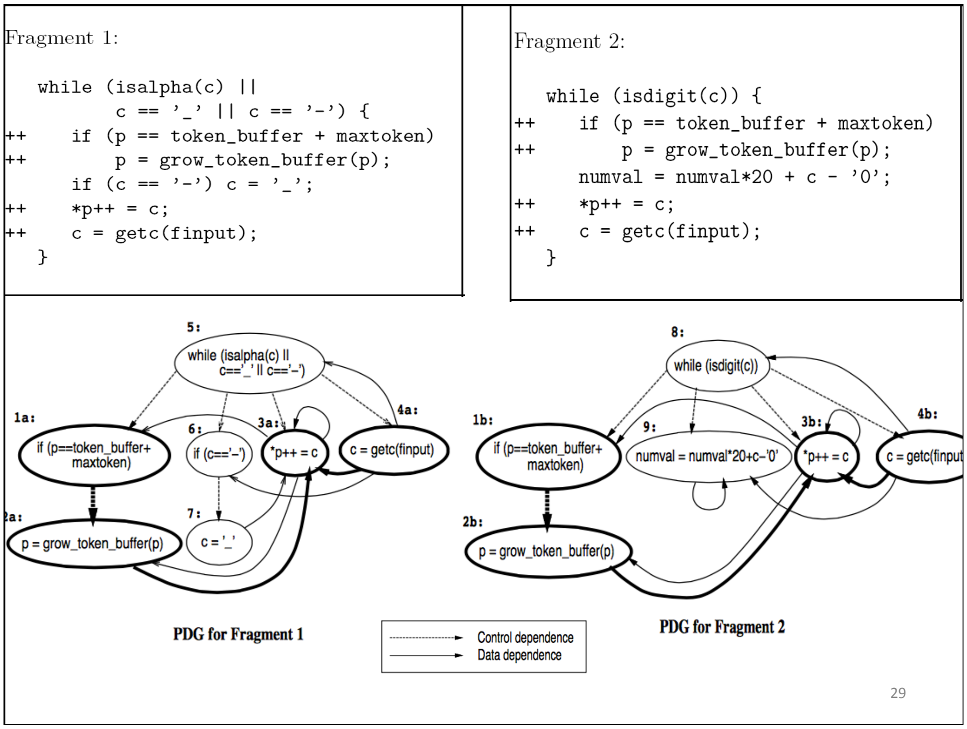
Fragment 1:

```
   while (isalpha(c) ||
          c == ’_’ || c == ’-’) {
++    if (p == token_buffer + maxtoken)
++        p = grow_token_buffer(p);
      if (c == ’-’) c = ’_’;
++    *p++ = c;
++    c = getc(finput);
   }
```



Fragment 2:

```
   while (isdigit(c)) {
++    if (p == token_buffer + maxtoken)
++        p = grow_token_buffer(p);
      numval = numval*20 + c - ’0’;
++    *p++ = c;
++    c = getc(finput);
   }
```



Fragment 1:

```
   while (isalpha(c) ||
          c == ’_’ || c == ’-’) {
++    if (p == token_buffer + maxtoken)
++          p = grow_token_buffer(p);
      if (c == ’-’) c = ’_’;
++    *p++ = c;
++    c = getc(finput);
   }
```

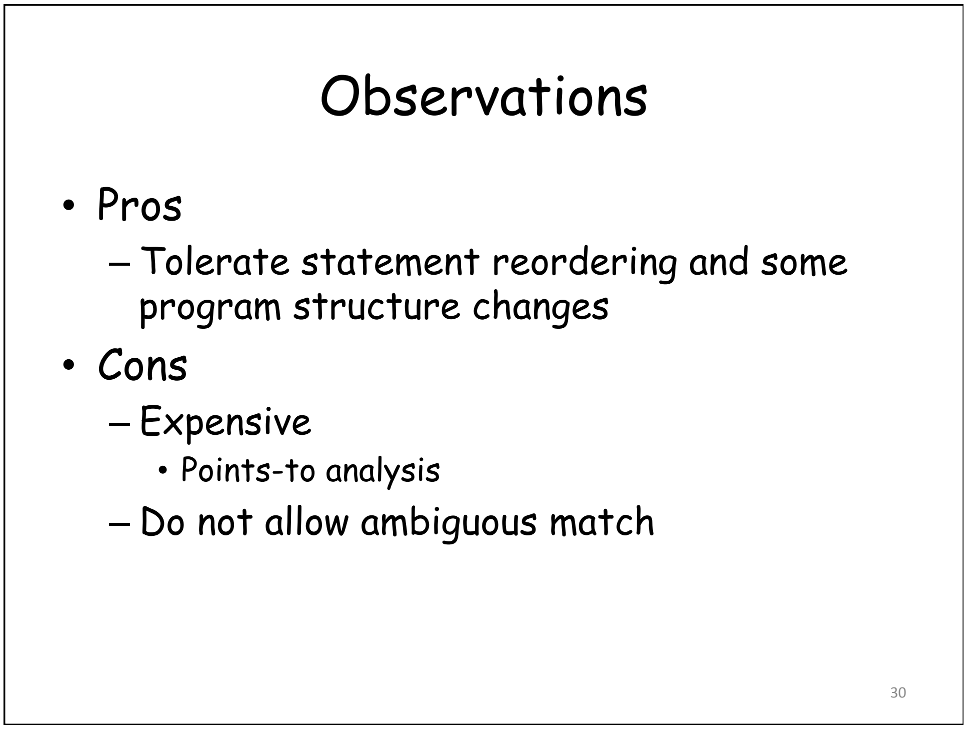
Fragment 2:

```
   while (isdigit(c)) {
++    if (p == token_buffer + maxtoken)
++          p = grow_token_buffer(p);
      numval = numval*20 + c - ’0’;
++    *p++ = c;
++    c = getc(finput);
   }
```

PDG for Fragment 1

PDG for Fragment 2

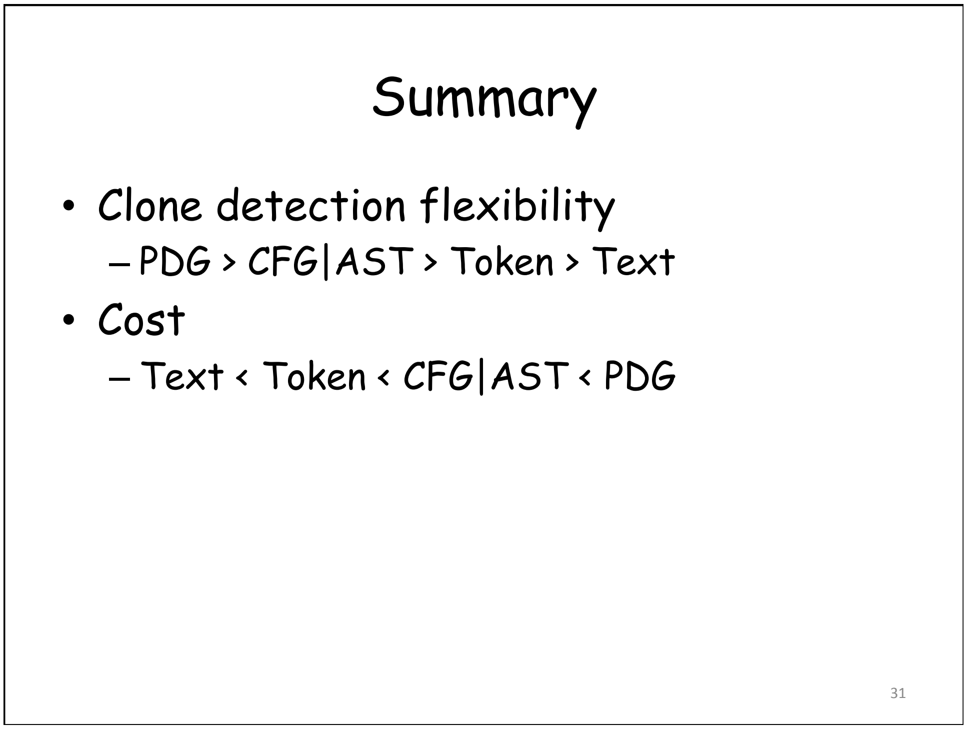
29



Observations

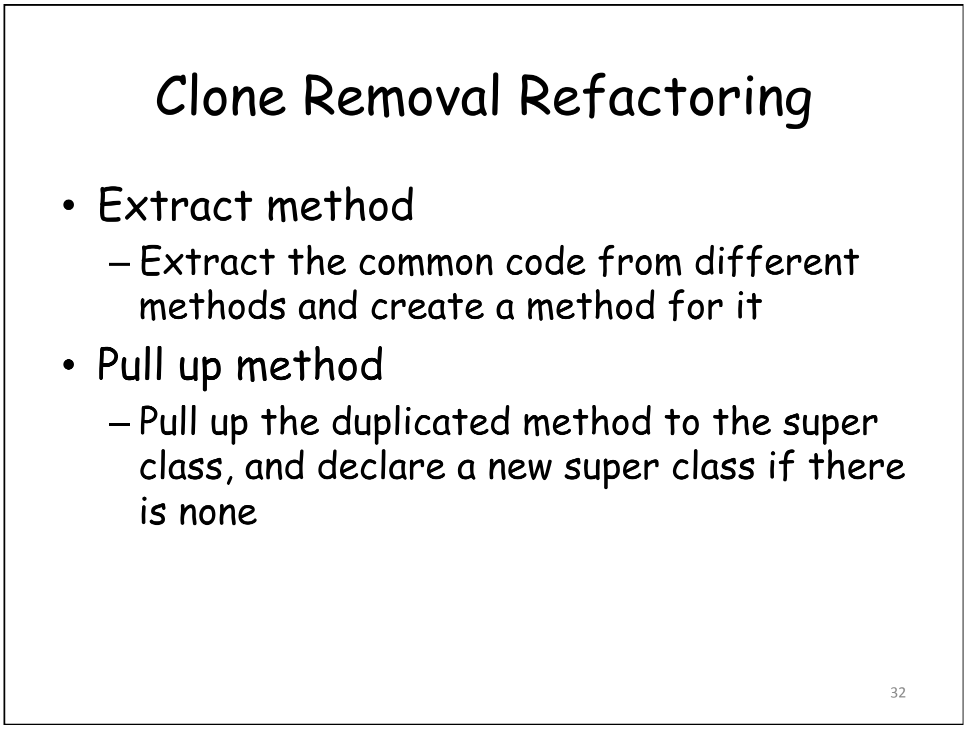
- Pros
  - Tolerate statement reordering and some program structure changes
- Cons
  - Expensive
    - Points-to analysis
  - Do not allow ambiguous match

30



Summary

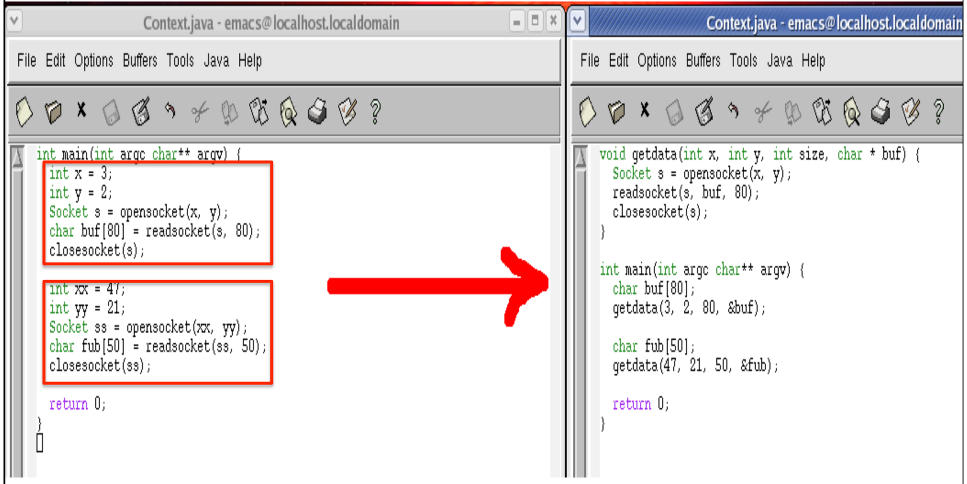
- Clone detection flexibility
  - PDG > CFG|AST > Token > Text
- Cost
  - Text < Token < CFG|AST < PDG



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

Extract Method


Context.java - emacs@localhost.localdomain
File Edit Options Buffers Tools Java Help
Context.java - emacs@localhost.localdomain
File Edit Options Buffers Tools Java Help

```
int main(int argc char** argv) {
  int x = 3;
  int y = 2;
  Socket s = opensocket(x, y);
  char buf[80] = readsocket(s, 80);
  closesocket(s);

  int xx = 47;
  int yy = 21;
  Socket ss = opensocket(xx, yy);
  char fub[50] = readsocket(ss, 50);
  closesocket(ss);

  return 0;
}
```

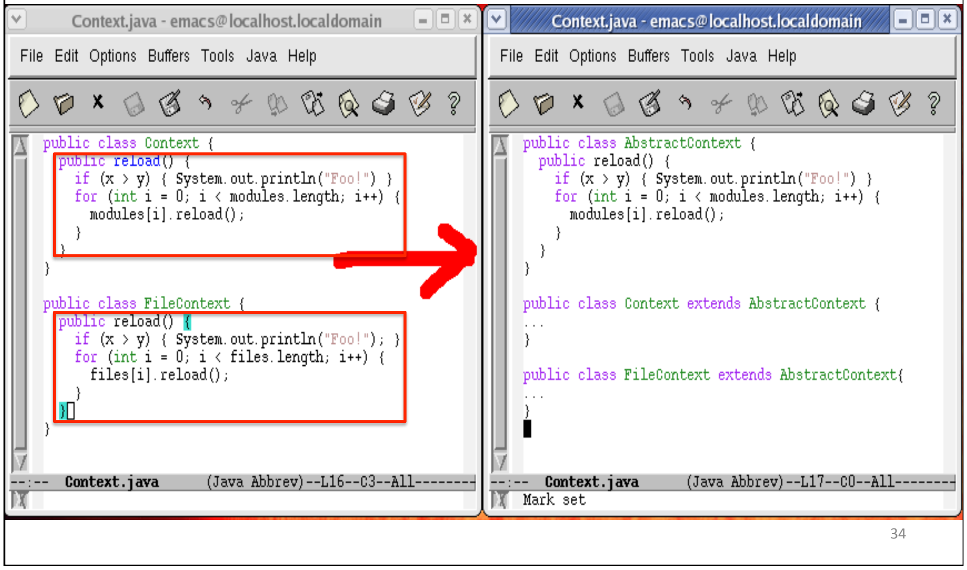
```
void getdata(int x, int y, int size, char * buf) {
  Socket s = opensocket(x, y);
  readsocket(s, buf, 80);
  closesocket(s);
}

int main(int argc char** argv) {
  char buf[80];
  getdata(3, 2, 80, &buf);

  char fub[50];
  getdata(47, 21, 50, &fub);

  return 0;
}
```

Pull Up Method


Context.java - emacs@localhost.localdomain
File Edit Options Buffers Tools Java Help
Context.java - emacs@localhost.localdomain
File Edit Options Buffers Tools Java Help

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

public class FileContext {
  public reload() {
    if (x > y) { System.out.println("Foo!"); }
    for (int i = 0; i < files.length; i++) {
      files[i].reload();
    }
  }
}
--:--  Context.java      (Java Abbrev)--L16--C3--All--------
```

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

public class Context extends AbstractContext {
...
}

public class FileContext extends AbstractContext{
...
}

--:--  Context.java      (Java Abbrev)--L17--C0--All--------
Mark set
```

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

Reference

[4] Taweesup 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