A Mutation / Injection-based Automatic Framework for Evaluating Code Clone Detection Tools

Chanchal K. Roy
James R. Cordy
Queen’s University, Canada
April 4, 2009
Introduction

- What are “Code Clones”?
  - A code fragment which has identical or similar code fragment(s) in source code
Introduction

- Intentional copy/paste is a common reuse technique in software development.
- Previous studies report 7% - 30% cloned code software systems [Baker WCRE’95, Roy and Cordy WCRE’08].
- Unfortunately, clones are harmful in software maintenance and evolution [Juergens et al. ICSE’09].
Introduction: Existing Methods

- In response, many methods proposed:
  - Text-based: Duploc [Ducasse et al. ICSM’99], NICAD [Roy and Cordy, ICPC’08]
  - Token-based: Dup [Baker, WCRE’95], CCFinder [Kamiya et al., TSE’02], CP-Miner [Li et al., TSE’06]
  - Tree-Based: CloneDr [Baxter et al. ICSM’98], Asta [Evans et al. WCRE’07], Deckard [Jiang et al. ICSE’07], cpdetector [Falke et al. ESE’08]
  - Metrics-based: Kontogiannis WCRE’97, Mayrand et al. ICSM’96
  - Graph-based: Gabel et al. ICSE’08, Komondooran and Horwitz SAS’01, Dublix [Krinke WCRE’01]
Introduction: Lack of Evaluation

- Marked lack of in-depth evaluation of the methods in terms of
  - precision and
  - recall

- Existing tool comparison experiments (e.g., Bellon et al. TSE’07) or individual evaluations have faced serious challenges [Baker TSE’07, Roy and Cordy ICPC’08, SCP’09]
Introduction: Precision and Recall

Software System

A: Actual clones?  C: Candidate clones detected by T

D: Detected actual clones

False negatives

False positives

<table>
<thead>
<tr>
<th>D</th>
<th></th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>C</td>
</tr>
</tbody>
</table>

Recall = \frac{|D|}{|A|}

Precision = \frac{|D|}{|C|}
Primary Challenge: Lack of a Reliable Reference Set

We DON’T have this Software System.

A : Actual clones?  C : Candidate clones.

We DON’T have this

We still don’t have this actual/reliable clone set for any system
Challenges in Oracling a System

- No crisp definition of code clones
- Huge manual effort
  - May be possible for small system
- What about for large systems?
  - Even the relatively small cook system yields nearly a million function pairs to sort through
  - Not possible for human to do error-free
Challenges in Evaluation

- Union of results from different tools can give good relative results
  - but no guarantee that the subject tools indeed detect all the clones
- Manual validation of the large candidate clone set is difficult
  - Bellon [TSE'07] took 77 hours for only 2% of clones
- No studies report the reliability of judges
Lack of Evaluation for Individual Types of Clones

- No work reports precision and recall values for different types of clones except,
  - Bellon et al. [TSE’07]: Types I, II and III
  - Falke et al. [ESE’08]: Types I and II
- Limitations reported
  - Baker [TSE’07]
  - Roy and Cordy ICPC’08, SCP’09
In this paper...

- A mutation-based framework that automatically and efficiently measures and compares precision and recall of the tools for different fine-grained types of clones.

- A taxonomy of clones
  - => Mutation operators for cloning
  - => Framework for tool comparison
An Editing Taxonomy of Clones

- Definition of clone is inherently vague
  - Most cases detection dependent and task-oriented

- Some taxonomies proposed
  - but limited to function clones and still contain the vague terms, “similar” and “long differences” [Roy and Cordy SCP’09, ICPC’08]

- We derived the taxonomy from the literature and validated with empirical studies [Roy and Cordy WCRE’08]

- Applicable to any granularity of clones
Exact Software Clones
Changes in layout and formatting

```
void sumProd(int n) { //s0
    int sum=0;       //s1
    int product =1;   //s2
    for (int i=1; i<=n; i++) { //s3
        sum=sum + i;     //s4
        product = product * i; //s5
        fun(sum, product);  //s6
    }
}
```

Changes in whitespace

Changes in comments

Changes in formatting

Reuse by copy and paste

Type I
Near-Miss Software Clone
Renaming Identifiers and Literal Values

```c
void sumProd(int n) { //s0
    int sum = 0; //s1
    int product = 1; //s2
    for (int i = 1; i <= n; i++) { //s3
        sum = sum + i; //s4
        product = product * i; //s5
        fun(sum, product); }
} //s6
```

- `sumProd` => `addTimes`
- `sum` => `add`
- `product` => `times`
- 0 => 0.0
- 1 => 1.0
- `int` => `double`

```c
void addTimes(int n) { //s0
    int add = 0; //s1
    int times = 1; //s2
    for (int i = 1; i <= n; i++) { //s3
        add = add + i; //s4
        times = times * i; //s5
        fun(add, times); }
} //s6
```

```c
void sumProd(int n) { //s0
    double sum = 0.0; //s1
    double product = 1.0; //s2
    for (int i = 1; i <= n; i++) { //s3
        sum = sum + i; //s4
        product = product * i; //s5
        fun(sum, product); }
} //s6
```
Near-Miss Software Clone
Statements added/deleted/modified in copied fragments

void sumProd(int n) { //s0
  int sum=0;               //s1
  int product =1;          //s2
  for (int i=1; i<=n; i++) { //s3
    sum=sum + i;            //s4
    product = product * i;  //s5
  fun(sum, product); } } //s6

Modification of lines

void sumProd(int n) { //s0
  int sum=0;               //s1
  int product =1;          //s2
  for (int i=1; i<=n; i++) { //s3
    if  (i % 2 == 0)
      sum+= i; //s4m
    product = product * i; //s5
  fun(sum, product); } } //s6

Addition of new of lines

void sumProd(int n) { //s0
  int sum=0;               //s1
  int product =1;          //s2
  for (int i=1; i<=n; i++) { //s3
    if  (i % 2 == 0)
      sum+= i; //s4m
  sum=sum + i;            //s4
  product = product * i;  //s5
  fun(sum, product); }} //s6

Reuse by copy and paste

Deletions of lines

void sumProd(int n) { //s0
  int sum=0;               //s1
  int product =1;          //s2
  for (int i=1; i<=n; i++) { //s3
  //line deleted
  sum=sum + i;            //s4
  product = product * i;  //s5
  fun(sum, product); } } //s6

Type III
void sumProd(int n) {     //s0
    int sum=0;                       //s1
    int product =1;                 //s2
    for (int i=1; i<=n; i++) {  //s3
        sum=sum + i;             //s4
        product = product * i; //s5
        fun(sum, product); }   //s6
}

Near-Miss Software Clone
Statements reordering/control replacements

void sumProd(int n) {     //s0
    int product =1;                //s2
    int sum=0;                       //s1
    for (int i=1; i<=n; i++) {  //s3
        sum=sum + i;             //s4
        product = product * i; //s5
        fun(sum, product); }   //s6
}

void sumProd(int n) {     //s0
    int sum=0;                       //s1
    int product =1;                 //s2
    int i = 0;                           //s7
    while (i<=n) {                  //s3'
        sum=sum + i;             //s4
        product = product * i; //s5
        fun(sum, product); }   //s6
    i =i + 1; } }                   //s8

Reordering of Statements
Reuse by copy and paste
Control Replacements

Type IV

Reuse by copy and paste
Mutation Operators for Cloning

- For each of the fine-grained clone types of the clone taxonomy,
  - We built mutation operators for cloning
  - We use TXL [Cordy SCP’06] in implementing the operators

- Tested with
  - C, C# and Java
Mutation Operators for Cloning

- For Type I Clones

<table>
<thead>
<tr>
<th>Name</th>
<th>Random Editing Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>mCW</td>
<td>Changes in whitespace</td>
</tr>
<tr>
<td>mCC</td>
<td>Changes in comments</td>
</tr>
<tr>
<td>mCF</td>
<td>Changes in formatting</td>
</tr>
</tbody>
</table>
mCC: Changes in Comments

<table>
<thead>
<tr>
<th>Line</th>
<th>Original</th>
<th>Mutated</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if (x==5)</td>
<td>if (x==5)</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>{</td>
<td>{</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>a=1;</td>
<td>a=1;</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>}</td>
<td>}</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>else</td>
<td>else</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>{</td>
<td>{</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>a=0;</td>
<td>a=0;</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>}</td>
<td>}</td>
<td>8</td>
</tr>
</tbody>
</table>
### mCF: Changes in formatting

<table>
<thead>
<tr>
<th>Line</th>
<th>Original</th>
<th>Mutated</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if (x==5)</td>
<td>if (x==5)</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>{</td>
<td>a=1;</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>a=1;</td>
<td>}</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>}</td>
<td>else</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>else</td>
<td>{</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>{</td>
<td>a=0;</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>a=0;</td>
<td>}</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>}</td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>
mCF: Changes in formatting

One or more changes can be made at a time
# Mutation Operators for Cloning

For **Type II Clones**

<table>
<thead>
<tr>
<th>Name</th>
<th>Random Editing Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>mSRI</td>
<td>Systematic renaming of identifiers</td>
</tr>
<tr>
<td>mARI</td>
<td>Arbitrary renaming of identifiers</td>
</tr>
<tr>
<td>mRPE</td>
<td>Replacement of identifiers with expressions (systematically/non-systematically)</td>
</tr>
</tbody>
</table>
### mSRI: Systematic renaming of identifiers

<table>
<thead>
<tr>
<th>Line</th>
<th>Original</th>
<th>Mutated</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if (x==5)</td>
<td>if (x==5)</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>{</td>
<td>{</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>a=1;</td>
<td>b=1;</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>}</td>
<td>}</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>else</td>
<td>else</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>{</td>
<td>{</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>a=0;</td>
<td>b=0;</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>}</td>
<td>}</td>
<td>8</td>
</tr>
</tbody>
</table>
mSRI: Systematic renaming of identifiers

<table>
<thead>
<tr>
<th>Line</th>
<th>Original</th>
<th>Mutated</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if (x==5)</td>
<td>if (x==5)</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>{</td>
<td>{</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>a=1;</td>
<td>b=1;</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>}</td>
<td>}</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>else</td>
<td>else</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>{</td>
<td>{</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>a=0;</td>
<td>b=0;</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>}</td>
<td>}</td>
<td>8</td>
</tr>
</tbody>
</table>
mARI: Arbitrary renaming of identifiers

<table>
<thead>
<tr>
<th>Line</th>
<th>Original</th>
<th>Mutated</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if (x==5) { a=1; } else { a=0; }</td>
<td>if (x==5) { b=1; } else { c=0; }</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>
### Mutation Operators for Cloning

- For **Type III Clones**

<table>
<thead>
<tr>
<th>Name</th>
<th>Random Editing Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>mSIL</td>
<td>Small insertions within a line</td>
</tr>
<tr>
<td>mSDL</td>
<td>Small deletions within a line</td>
</tr>
<tr>
<td>mILs</td>
<td>Insertions of one or more lines</td>
</tr>
<tr>
<td>mDLs</td>
<td>Deletions of one or more lines</td>
</tr>
<tr>
<td>mMILs</td>
<td>Modifications of whole line(s)</td>
</tr>
</tbody>
</table>
mSIL: Small Insertion within a Line

<table>
<thead>
<tr>
<th>Line</th>
<th>Original</th>
<th>Mutated</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if (x==5)</td>
<td>if (x==5)</td>
</tr>
<tr>
<td></td>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>2</td>
<td>a=1;</td>
<td>a=1 + x;</td>
</tr>
<tr>
<td>3</td>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>4</td>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td>5</td>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>6</td>
<td>a=0;</td>
<td>a=0;</td>
</tr>
<tr>
<td>7</td>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
mILs: Insertions of One or More Lines

<table>
<thead>
<tr>
<th>Line</th>
<th>Original</th>
<th>Mutated</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if (x==5)</td>
<td>if (x==5)</td>
</tr>
<tr>
<td>2</td>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>3</td>
<td>a=1;</td>
<td>a=1;</td>
</tr>
<tr>
<td>4</td>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>5</td>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td>6</td>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>7</td>
<td>a=0;</td>
<td>a=0;</td>
</tr>
<tr>
<td>8</td>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

mILs: Insertions of One or More Lines

if (x==5)
{
a=1;
}
else
{
a=0;
}
y=y + c;

## Mutation Operators for Cloning

**For Type IV Clones**

<table>
<thead>
<tr>
<th>Name</th>
<th>Random Editing Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>mRDs</td>
<td>Reordering of declaration statements</td>
</tr>
<tr>
<td>mROS</td>
<td>Reordering of other statements (Data-dependent and/or in-dependent statements)</td>
</tr>
<tr>
<td>mCR</td>
<td>Replacing one type of control by another</td>
</tr>
</tbody>
</table>
Mutation Operators for Cloning

- Combinations of mutation operators

Original          Final Mutated

if (x==5)  
  {  
    a=1;  
  }  
else
  {  
    a=0;  
  }

if (x==5)  
  {  
    a=1;  
  }  
else //c
  {  
    a=0;  
  }

if (x==5)  
  {  
    b=1;  
  }  
else //c
  {  
    b=0;  
  }

if (x==5)  
  {  
    b=1+x;  
  }  
else //c
  {  
    b=0;  
  }

mCC + mSRI + mSIL
The Evaluation Framework

- **Generation Phase**
  - Create artificial clone pairs (using mutation analysis)
  - Inject to the code base

- **Evaluation Phase**
  - How well and how efficiently the known clone pairs are detected by the tool(s)
Generation Phase: Base Case

- Code Base
  - Randomly picks a fragment
  - Randomly injects into the code base

- Database
  - Original Fragment
  - Pick an operator (say mCC) and apply
  - Mutated Fragment

- Mutation Operators
  - oCF(f3, 10, 20)
  - mOCF(f1, 2, 12)
  - OP/Type: mCC
  - CP: (oCF, moCF)
Evaluation Phase: Base Case

Subject Tool: T

Clone Report

Database

Mutated Code Base

oCF(f3, 10, 20)
mOCF(f1, 2, 12)
OP/Type: mCC
CP: (oCF, moCF)

Tool Evaluation
Unit Recall

For known clone pair, \((oCF, moCF)\), of type mCC, the unit recall is:

\[
UR_{T}^{(oCF, moCF)} = \begin{cases} 
1, & \text{if } (oCF, moCF) \text{ is killed by } T \\
0, & \text{otherwise}
\end{cases}
\]
Definition of killed(oCF, moCF)

- (oCF, moCF) has been detected by the subject tool, T
  - That is a clone pair, (CF1, CF2) detected by T matches or subsumes (oCF, moCF)
  - We use source coordinates of the fragments to determine this
  - First match the full file names of the fragments, then check for begin-end line numbers of the fragments within the files
Unit Precision

- Say, for moCF, T reports \( k \) clone pairs,
  - \((\text{moCF}, \text{CF}_1), (\text{moCF}, \text{CF}_1), \ldots, (\text{moCF}, \text{CF}_k)\)
- Also let, \( v \) of them are valid clone pairs, then
  - For known clone pair, \((\text{oCF}, \text{moCF})\), of type mCC, the unit precision is:

\[
\text{UP}_{(\text{oCF}, \text{moCF})} = \frac{v}{k}
\]

- \( v \): Total no of valid clone pairs with moCF
- \( k \): Total no of clone pairs with moCF
Automatic Validation of Known Clone Pairs

- Built a clone pair validator based on NICAD (Roy and Cordy ICPC’08)
- Unlike NICAD, it is not a clone detector
  - It only works with a specific given clone pair
  - It is aware of the mutation operator applied
  - Depending on the inserted clone, detection parameters are automatically tuned
Generation Phase: General Case

Random Fragment Selection

Mutator 1

Mutator 2

Mutator N

Randomly Mutated Fragments

Random Fragment Injection

Injected Mutant Source Coordinate Database

Original Code Base

Random Fragments

Randomly Injected Mutant Code Bases

Randomly Mutated Fragments
Evaluation Phase: General Case

Injected Mutant Source Coordinate Database

1

Tool 1

Tool 2

Tool K

Mutant 1 Tool Eval

Mutant 2 Tool Eval

Mutant M Tool Eval

Evaluation Database

Randomly Injected Mutant Code Bases

Tool 1 Mutant 1 Report

Tool 2 Mutant 1 Report

Tool K Mutant 1 Report

Statistical Analysis and Reporting
Recall

With mCC, m fragments are mutated and each injected n times to the code base

\[ R_{mCC}^T = \sum_{i=1}^{m \times n} UR_{(oCF_i, moCF_i)} \]

\[ R_{Type \ I}^T = \sum_{i=1}^{(m \times n) \times (3 + 4)} UR_{(oCF_i, moCF_i)} \]

\[ \frac{T}{m \times n} \]
Overall Recall

I clone mutation operators and c of their combinations applied n times to m selected code fragments, so

\[ R_{\text{overall}} = \frac{\sum_{i=1}^{n} \text{UR}^{(\text{oCF}_i, \text{moCF}_i)}}{T} \]

\[ = \frac{(m \times n) \times (l + c)}{(m \times n) \times (l + c)} \]
With mCC, m fragments are mutated and each injected n times to the code base.

\[ P_{\text{mCC}} = \frac{\sum_{i=1}^{m} n v_i}{\sum_{i=1}^{m} n k_i} \]

\[ P_{\text{Type I}} = \frac{\sum_{i=1}^{m} n (3 + 4) v_i}{\sum_{i=1}^{m} n (3 + 4) k_i} \]
Overall Precision

+ l clone mutation operators and c of their combinations applied n times to m selected code fragments, so

\[ P = \frac{\sum_{i=1}^{m} v_i}{\sum_{i=1}^{m} k_i} \]
Example Use of the Framework

- Select one or more subject systems
- Case one: Evaluate single tool
  - We evaluate NICAD [Roy and Cordy ICPC’08]
- Case two: Compare a set of tools
  - Basic NICAD [Roy and Cordy WCRE’08]
  - Flexible Pretty-Printed NICAD [Roy and Cordy ICPC’08]
  - and Full NICAD [Roy and Cordy ICPC’08]
## Subject Systems

<table>
<thead>
<tr>
<th>Language</th>
<th>Code Base</th>
<th>LOC</th>
<th>#Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>GZip-1.2.4</td>
<td>8K</td>
<td>117</td>
</tr>
<tr>
<td></td>
<td>Apache-httpd-2.2.8</td>
<td>275K</td>
<td>4301</td>
</tr>
<tr>
<td></td>
<td>Weltab</td>
<td>11K</td>
<td>123</td>
</tr>
<tr>
<td>Java</td>
<td>Netbeans-Javadoc</td>
<td>114K</td>
<td>972</td>
</tr>
<tr>
<td></td>
<td>Eclipse-jdtcore</td>
<td>148K</td>
<td>7383</td>
</tr>
<tr>
<td></td>
<td>JHotdraw 5.4b</td>
<td>40K</td>
<td>2399</td>
</tr>
</tbody>
</table>
## Recall Measurement

<table>
<thead>
<tr>
<th>Clone Type</th>
<th>Standard Pt-Printing</th>
<th>Flexible Pt-Printing</th>
<th>Full NICAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type I</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Type II</td>
<td>29%</td>
<td>27%</td>
<td>100%</td>
</tr>
<tr>
<td>Type III</td>
<td>80%</td>
<td>85%</td>
<td>100%</td>
</tr>
<tr>
<td>Type IV</td>
<td>67%</td>
<td>67%</td>
<td>77%</td>
</tr>
<tr>
<td>Overall</td>
<td>84%</td>
<td>87%</td>
<td>96%</td>
</tr>
</tbody>
</table>
## Precision Measurement

<table>
<thead>
<tr>
<th>Clone Type</th>
<th>Standard Pt-Printing</th>
<th>Flexible Pt-Printing</th>
<th>Full NICAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type I</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Type II</td>
<td>94%</td>
<td>94%</td>
<td>97%</td>
</tr>
<tr>
<td>Type III</td>
<td>85%</td>
<td>81%</td>
<td>96%</td>
</tr>
<tr>
<td>Type IV</td>
<td>81%</td>
<td>79%</td>
<td>89%</td>
</tr>
<tr>
<td>Overall</td>
<td>90%</td>
<td>89%</td>
<td>95%</td>
</tr>
</tbody>
</table>
Other Issues

- Time and memory requirements
  - Can report fine-grained comparative timing and memory requirements for subject tools

- Scalability of the framework
  - Can work subject system of any size, depending on the scalability of the subject tools
  - Uses multi-processing to balance the load

- Adapting tools to the framework
  - The subject tool should run in command line
  - Should provide textual reports of the found clones
Related Work: Tool Comparison Experiments

- Baily and Burd SCAM’02
  - 3 CDs + 2 PDs
- Bellon et al. TSE’07
  - Extensive to-date
  - 6 CDs, C and Java systems
- Koschke et al. WCRE’06
- Rysselberge and Demeyer ASE’04
Related Work: Single Tool Evaluation Strategies

- **Text-based**
  - Only example-based evaluation, no precision and recall evaluations except NICAD

- **Token-based**
  - CP-Miner, for precision by examining 100 randomly selected fragments

- **AST-based**
  - Cpdetector, in terms of both precision and recall (Type I & II).
  - Deckard, for precision with examining 100 segments

- **Metrics-based**
  - Kontogianis evaluated with IR-approach, system was oracled manually.

- **Graph-based**
  - Gabel et al., precision examined 30 fragments per experiment.
Conclusion and Future Work

- Existing evaluation studies have several limitations
  - Baker TSE’07, Roy and Cordy SCP’09/ICPC’08
- We provided a mutation/injection-based automatic evaluation framework
  - Evaluates precision and recall single tool
  - Compare tools for precision and recall
- Effectiveness of this framework has been shown comparing NICAD variants
- We are planning to conduct a mega tool comparison experiment with the framework
Acknowledgements

- Inspirations
  - Rainer Koschke for Dagstuhl seminar 2006
  - Stefan Bellon et al., for beginning the road to meaningful evaluation & comparison

- Tool & method authors
  - For useful answers to our questions and worries

- Anonymous referees & colleagues
  - For help in presenting, tuning and clarifying several of the papers of this work.
Questions?