Detection and Analysis of Near-Miss Software Clones

Chanchal Roy
University of Saskatchewan, Canada
September 22, 2009
Software Clone or Code Clone?

A code fragment which has identical or similar code fragment(s) in source code

Slide from Katsuro Inoue group
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

Type I

Reuse by copy and paste

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

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

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
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

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

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

Near-Miss Software Clone
Renaming Identifiers and Literal Values

Reuse by copy and paste
Renaming of identifiers
Renaming of Literals and Types

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
  sum=sum + i; //s4
  div=sum/2 //s3b
  product = product * i; //s5
  sum=sum + i; //s4
  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
  if  (i % 2 == 0) sum+= i; //s4m
  product = product * i; //s5
  fun(sum, product); } } //s6

Reuse by copy and paste

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

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
  sum=sum + i; //s4
  line deleted
  product = product * i; //s5
  fun(sum, product); } } //s6

Deletions of lines
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
}

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}
Are There Clones in Software?

- Yes, significant amount of cloned code
- Varies depending on the domain and origin.
  - 7%-23%  Baker WCRE’05
  - 12.7%  Baxter et al. ICSM’98
  - 5%-20%  Mayrand et al. ICSM’96
  - 10%-15%  Kapser and Godfrey JSME’06
- Sometimes, even about 50%,  OO COBOL system,  Ducasse et al. JSME’06
Why there are Clones in Software?

- “Copy&paste” is a common practice in software development [Kapser and Godfrey WCRE’06]
  - Existing code is often used as templating
- Time limit assigned to programmers
  - No time for developing new code
- Risk in developing new code [Cordy ICPC’03]
  - Existing code is well tested
- Language limitations
  - Does not allow procedural abstraction
Do Clones Matter?

- Are they Harmful? Possibly, Yes, Juergens et al. ICSE’09
  - “Inconsistent changes to code duplicates are frequent and can lead to severe unexpected behavior”
  - “Number 1 Bad Smell” in the community Beck and Fowler’s “Stink Parade of Bad Smells”

- Increased cognitive effort
  - Difficult to understand differences betw cloned fragments

- Code bloat
  - Produce unnecessarily long code

- However, “could be useful in many ways”, Kapser and Godfrey WCRE’06
Other Applications of Code Similarity Detection?

- Software Maintenance and Evolution
- Software Reuse Process
- Program Comprehension
- Detecting Library Candidates
- Plagiarism Detection
- Aspect Mining
- Virus Detection
- Adapt to Model-Based Development
Contributions of the Thesis

Scenario-based Comparison & Evaluation
- SCP’09, ICPC’08(a)
- Invited for special issue
- 115 page technical report
- Grand survey with Rainer Koschke

Mutation-/Injection-based Evaluation Framework
- C3S2E’08
- Mutation’09 (Best Poster award) (best paper award)

Literature Review
- (250+ papers)

An Editing Taxonomy for Intentional Clone Creation
- C3S2E’08

Today’s Talk
- NICAD: Next Generation Clone Detection Tool
- ICPC’08(b)

Large Empirical Study
- WCRE’08, Invited for special issue to JSME’09 (accepted)
Clone Detection Process

“Software entities are more complex for their size than perhaps any other human construct because no two parts are alike (at least above the statement level). If they are, we make the two similar parts into a subroutine - open or closed. In this respect, software systems differ profoundly from computers, buildings, or automobiles, where repeated elements abound.”

-By Frederick P. Brooks, Jr: No Silver Bullet: Essence and Accidents of Software Engineering

*Slide from Rainer Koschke*
Existing Clone Detection Methods

Because of the importance, 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]
- **Parser-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, Komondoor and Horwitz SAS’01, Dublix [Krinke WCRE’01]
Motivation [Bellon et al. TSE’07, Roy and Cordy Tech Report’07]

- Text-based techniques can find clones
  - with high accuracy and confidence, but-
    - detected clones are often **non-syntactic**
    - sensitive to formatting and any sort of editing changes

- Parser-based techniques can detect
  - syntactically meaningful clones, but-
    - tend to be more **heavy-weight**, requiring a fully-fledged parser and sub-tree comparison
    - experiments also show **low recall** for tree-based techniques

- Furthermore, **neither of them can detect near-miss clones well**.
NICAD: A Hybrid

- Hybrid of text-based and parser-based methods by combining their strengths and overcoming their limitations
- Use benefits of parser, efficient island grammar, agile parsing and source transformation but compares code fragments text-line-wise instead of sub-tree comparison
- Also many other interesting features for near-miss clone detection
- Standalone, only needs TXL grammar
NICAD
conceptual diagram

1. Code Base
   - Potential Clone extractor and standard pretty-printer
   - Pretty-printed Potential Clones
     - Flexible Code Normalization
     - Flexible Code Filtering
     - Flexible Code Pretty-Printing

2. Threshold/Exemplar-based dynamic Clustering of Pre-processed Potential Clones and Text-line Comparison
   - Clone Data
   - Report Generator

3. Mapped back
   - Interactive HTML Report
   - XML Report
 NICAD
conceptual diagram

Potential Clone extractor and standard pretty-printer

Flexible Code Normalization
Flexible Code Filtering
Flexible Code Pretty-Printing

Threshold/Exemplar-based dynamic Clustering of Pre-processed Potential Clones and Text-line Comparison

Clone Data
Report Generator

Interactive HTML Report
XML Report

Mapped back
Structural Extraction

- Use robust island grammars to isolate and extract
  - Extract meaningful units for comparison
    - Example: begin-end block or function block
    - Also get source coordinate of the units

- During extraction we can also apply standard pretty-printing
  - Removes comments and formatting differences
Standard Pretty-Printing

Using TXL’s agile parsing we can include grammatical cues

- preserve tree structure in text

Text line-based
- Not clones
Even “{“ and “}” are added to fragments F2 and F3
Standard Pretty-Printing

Text-line comparison now finds them exactly similar
Form a clone class, \{F1, F2, F3, F4\} as of tree-based method but avoids expensive tree comparison
Because of text-comparison, precision is very high
NICAD
conceptual diagram

Potential Clone extractor and standard pretty-printer

Pretty-printed Potential Clones

Flexible Code Normalization
Flexible Code Filtering
Flexible Code Pretty-Printing

Threshold/Exemplar-based dynamic Clustering of Pre-processed Potential Clones and Text-line Comparison

Clone Data

Report Generator

Interactive HTML Report
XML Report

Mapped back
Flexible Code Normalization

Again, we use TXL rules for context-sensitive structural abstraction/normalization
Flexible Code Normalization

<table>
<thead>
<tr>
<th>Line</th>
<th>F1</th>
<th>F2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if(AnyIfControl) {</td>
<td>if (AnyIfControl) {</td>
</tr>
<tr>
<td></td>
<td>a=n + y;</td>
<td>a=n + y;</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>3</td>
<td>else {</td>
<td>else {</td>
</tr>
<tr>
<td>5</td>
<td>a=n - y;</td>
<td>a=n - y;</td>
</tr>
<tr>
<td>6</td>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

- Could be further scoped, e.g., only the right part of the control
- Language and possibly project specific

Now same and text-line will find \{F1, F2\} as clone pair
### Flexible Code Filtering

<table>
<thead>
<tr>
<th>F1</th>
<th>Line</th>
<th>Line</th>
<th>F2</th>
</tr>
</thead>
<tbody>
<tr>
<td>void foo(){</td>
<td>1</td>
<td>void foo(){</td>
<td>1</td>
</tr>
<tr>
<td>int x=10;</td>
<td>2</td>
<td>int x=10, n=2, y=5, a=0;</td>
<td>2</td>
</tr>
<tr>
<td>Int n=2;</td>
<td>3</td>
<td>if (x&lt;=5) {</td>
<td>3</td>
</tr>
<tr>
<td>Int y=5, a=0;</td>
<td>4</td>
<td>a=n + y;</td>
<td>4</td>
</tr>
<tr>
<td>if (x&lt;==5) {</td>
<td>5</td>
<td>}</td>
<td>5</td>
</tr>
<tr>
<td>a=n + y;</td>
<td>6</td>
<td>else {</td>
<td>6</td>
</tr>
<tr>
<td>}</td>
<td>7</td>
<td>a=n - y;</td>
<td>7</td>
</tr>
<tr>
<td>else {</td>
<td>8</td>
<td>}</td>
<td>8</td>
</tr>
<tr>
<td>a=n - y;</td>
<td>9</td>
<td>}</td>
<td>9</td>
</tr>
<tr>
<td>}</td>
<td>10</td>
<td>Elide unimportant parts for comparison</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>Contextual basis, lang. and project specific</td>
<td></td>
</tr>
</tbody>
</table>
### Flexible Code Filtering

<table>
<thead>
<tr>
<th>F1</th>
<th>Line</th>
<th>Line</th>
<th>F2</th>
</tr>
</thead>
<tbody>
<tr>
<td>void foo() {</td>
<td>1</td>
<td>1</td>
<td>void foo() {</td>
</tr>
<tr>
<td>if (x &lt;= 5) {</td>
<td>2</td>
<td>2</td>
<td>if (x &lt;= 5) {</td>
</tr>
<tr>
<td>a = n + y;</td>
<td>3</td>
<td>3</td>
<td>a = n + y;</td>
</tr>
<tr>
<td>}</td>
<td>4</td>
<td>4</td>
<td>}</td>
</tr>
<tr>
<td>else {</td>
<td>5</td>
<td>5</td>
<td>else {</td>
</tr>
<tr>
<td>a = n - y;</td>
<td>6</td>
<td>6</td>
<td>a = n - y;</td>
</tr>
<tr>
<td>}</td>
<td>7</td>
<td>7</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td>8</td>
<td>8</td>
<td>}</td>
</tr>
</tbody>
</table>

- Text-line comparison now finds them as clone pair with high accuracy
Flexible Pretty-printing

- Example, “for” headers

<table>
<thead>
<tr>
<th>Line</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>for(i=0;i&lt;10;i++)</code></td>
<td><code>for(i=1;i&lt;10;i++)</code></td>
<td><code>for(j=2;j&lt;100;j++)</code></td>
</tr>
</tbody>
</table>

NOT same on text-line comparison

<table>
<thead>
<tr>
<th>Line</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>for(i=0;i&lt;10;i++)</code></td>
<td><code>for(i=1;i&lt;10;i++)</code></td>
<td><code>for(j=2;j&lt;100;j++)</code></td>
</tr>
<tr>
<td>2</td>
<td><code>i=0;</code></td>
<td><code>i=1;</code></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><code>i&lt;10;</code></td>
<td><code>i&lt;10;</code></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><code>i++;</code></td>
<td><code>i++;</code></td>
<td></td>
</tr>
</tbody>
</table>

{F1, F2} 75% same
{F1, F3} 25% Same
{F3, F3} 25% Same
NICAD
conceptual diagram

28 NICAD

Code Base

Potential Clone extractor and standard pretty-printer

Pretty-printed Potential Clones

Flexible Code Normalization

Flexible Code Filtering

Flexible Code Pretty-Printing

Threshold/Exemplar-based dynamic Clustering of Pre-processed Potential Clones and Text-line Comparison

Clone Data

Report Generator

Interactive HTML Report

XML Report

Mapped back
Catching Unanticipated Gaps

<table>
<thead>
<tr>
<th>F1</th>
<th>Line</th>
<th>Line</th>
<th>F2</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (x&lt;=5) {</td>
<td>1</td>
<td>1</td>
<td>if (x&lt;=5) {</td>
</tr>
<tr>
<td>a=n + y;</td>
<td>2</td>
<td>2</td>
<td>a=n + y;</td>
</tr>
<tr>
<td>y=y + 2;</td>
<td>3</td>
<td>3</td>
<td>y=y + 2;</td>
</tr>
<tr>
<td>n=n +1;</td>
<td>4</td>
<td>4</td>
<td>n=n +1;</td>
</tr>
<tr>
<td>functionName(a, y, n) ;</td>
<td>5</td>
<td>5</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td>6</td>
<td>6</td>
<td>else {</td>
</tr>
<tr>
<td>else {</td>
<td>7</td>
<td>7</td>
<td>a=n - y;</td>
</tr>
<tr>
<td>a=n - y;</td>
<td>8</td>
<td>8</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Using LCS we calculate dissimilarity metrics
- If both metrics below certain threshold, we call them clones
Comparing the Potential Clones

Choose Largest Unclassified Potential Clone as Exemplar

Pretty-Printed / Normalized / Filtered Potential Clone Files

Dynamic Cluster Comparable Size Potential Clones

Dissimilarity Threshold

Comparable Size Potential Clone Cluster

Pair Comparison of Exemplar with Potential Clones

Choose Next Exemplar and Repeat the Process

Clone Classes

Choose Next Exemplar and Repeat the Process
NICAD
conceptual diagram

 Potential Clone extractor and standard pretty-printer

 Pretty-printed Potential Clones

 Flexible Code Normalization
 Flexible Code Filtering
 Flexible Code Pretty-Printing

 Threshold/Exemplar-based dynamic Clustering of Pre-processed Potential Clones and Text-line Comparison

 Clone Data

 Report Generator

 Interactive HTML Report

 XML Report

 Mapped back

 Code Base
NICAD’s Output: XML format

<clones systemname="abyss" minclonesize="3" withupi="30%" nfragments="6" nClasses="3">
  <class id="1" nfragments="2">
    <source file="abyss/src/server.c" startline="63" endline="71" withupi="0"/>
    <source file="abyss/src/server.c" startline="53" endline="61" withupi="0.3"/>
  </class>
  <class id="2" nfragments="2">
    <source file="abyss/src/trace.c" startline="59" endline="67" withupi="0"/>
    <source file="abyss/src/trace.c" startline="50" endline="57" withupi="0.3"/>
  </class>
  ...
</clones>
Clone Visualization with NICAD

```c
int cmpfiledates(const TFileInfo **f1, const TFileInfo **f2)
{
    if (!(*f1)->attrib & A_SUBDIR) && !(*f2)->attrib & A_SUBDIR)
        return (-1);
    if (!(*f1)->attrib & A_SUBDIR) && (*f2)->attrib & A_SUBDIR)
        return 1;
    return ((*f1)->time_write - (*f2)->time_write);
}
```

Source file="abyss/src/server.c" startline="63" endline="71" withupi="0" pcid="107"

Report 1: [all clones in class] (2 total)

```c
void TraceExit (char *fmt,...)
{
    va_list argptr;
    va_start (argptr, fmt);
    TraceNMsg (fmt, argptr);
    va_end (argptr);
}
```
Visualization: A Clone Class

```c
int cmpfiledates(const TFileInfo **f1, const TFileInfo **f2)
{
    if (((*f1)->attrib & A_SUBDIR) && ((*f2)->attrib & A_SUBDIR))
        return (-1);
    if (!((*f1)->attrib & A_SUBDIR) && ((*f2)->attrib & A_SUBDIR))
        return 1;
    return ((*f1)->time_write-(*f2)->time_write);
}

Source file="abyss/src/server.c" startline="63" endline="71" withupi="0" pclid="107"
```

```c
int cmpfilenames(const TFileInfo **f1, const TFileInfo **f2)
{
    if (((*f1)->attrib & A_SUBDIR) && ((*f2)->attrib & A_SUBDIR))
        return (-1);
    if (!((*f1)->attrib & A_SUBDIR) && ((*f2)->attrib & A_SUBDIR))
        return 1;
    return strcmp((*f1)->name, (*f2)->name);
}

Source file="abyss/src/server.c" startline="53" endline="61" withupi="0.3" pclid="106"
```
Evaluation of NICAD

- First small empirical study with welttab
  - Shows better than an existing tree-based method
  - Studied effect of flexible code pretty-printing, normalizations, filtering

- Also found better than others in the scenario-based evaluation
Precision and Recall of NICAD

- **Very High Precision**
  - No false positives up to UPI threshold 20%

- **Not quite sure about Recall**
  - Only measured for a system

- **Cannot know Recall for large systems**
  - No standard benchmark available
  - Validation is challenging
    - huge manual work in validating candidate clones
    - no crisp definition of clones, judges disagree
Evaluation Framework

- 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 meta model for clone creation
  - => Model-driven mutation operators for cloning
    - => Framework for tool comparison
Recall of NICAD

- An **Automatic** Mutation-/Injection-Based Framework (C3S2E’08, Mutation’09), results w.r.t Weltab open source system.

<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 of NICAD

- **An Automatic Mutation-/Injection-Based Framework**
  (C3S2E’08, Mutation’09), results w.r.t Weltab open source system.

<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>
Large Empirical Studies

Still not sure whether NICAD can detect clones
- From large systems (e.g., from Linux)
- From systems of different languages

Furthermore, marked lack of in-depth studies
- On the cloning status in open source systems
  - in particular w.r.t. near-miss clones and
  - systems of different languages
Large Empirical Studies
Roy and Cordy WCRE’08, JSME’09(Invited for special issue)

- Comprehensive in-depth evaluation of clone properties
  - In different dimensions including
    - Language (three different languages: 10 C, 7 Java and 6 C#)
    - Clone Size
    - Clone Similarity
    - Clone Location
    - Clone Density (both by LOC and cloned functions)
    - Varying system size: 4 KLOC- 6.3 MLOC
    - Diverse varieties of applications
  - All open source systems including complete Linux Kernel
Future Work

- Further hybridization of NICAD
  - hybrid with IR/program dependency analysis for semantic clones
- NICAD in the development process
  - as a plug-in in the IDEs
- Mutation-based framework
  - mega tool comparison experiment with 3rd party tools
- A comprehensive clone management system
  - a quality-driven taxonomy of clones
  - an optimized refactoring model
Conclusion

- A hybrid clone detection method
  - detects both exact and near-miss clones with high precision and recall.

- An Editing Taxonomy of clones
  - attempts to define a theory of clone creation

- A scenario-based comparison and evaluation
  - helps select tools for specific purpose or build new hybrid method

- A mutation-based framework to evaluate clone detection tools
  - automatically evaluates NICAD for precision and recall

- Large scale empirical studies
  - evaluates NICAD and studies cloning in large systems of different languages
Acknowledgements

Thanks to Jim Cordy

- Introduced me the area of clone detection
- Available anytime
  - Even works in the weekend
- Always praises his students
  - His students are the best!
- Allows freedom to the students
  - Worked in other areas as well
- Cares a lot about the future careers of his students
  - He forced me to apply for jobs
  - Also helped to get an NSERC PDF
Experiences

- Could have done more
  - Was not motivated in the beginning

- Listen carefully to the advisor but not always
  - What you think about the problem/solution?
    - Keep faith in you!
  - Show enough causes to understand your advisor
    - Its your responsibility

- Be serious early of the PhD, have paper published!!
  - Keep 4th year for searching jobs?
Questions?
Contributions of the Thesis

Scenario-based Comparison & Evaluation

- SCP’09, ICPC’08(a) (Invited for special issue)
- 115 page technical report
- Grand survey with Rainer Koschke

Mutation-/Injection-based Evaluation Framework

- C3S2E’08 (Best Poster award) (best paper award)

Literature Review (250+ papers)

- An Editing Taxonomy for Intentional Clone Creation
- C3S2E’08
- ICPC’08(a)

NICAD: Next Generation Clone Detection Tool

- ICPC’08(b)

Large Empirical Study

- WCRE’08, Invited for special issue to JSME’09