A Differencing Algorithm for Object-Oriented Programs

Taweesup (Term) Apiwattanapong
Alessandro Orso
Mary Jean Harrold

College of Computing
Georgia Institute of Technology

National Science Foundation awards CCR-0306372, CCR-0205422, CCR-9988294, CCR-0209322, and SBE-0123532 to Georgia Tech
2c2
< float n;
---
> double n;

6a7,9
> public void m1() {
> ... 
> }

9a13
> int i=0;

21c25
< public class E2 extends E1 {
---
> public class E2 extends Exception {

Modified Version (P')

```
try {
    n = 1/n;
    a.m1();
}
```
Outline

• Introduction
• Differencing Algorithm
  • Representation
  • Matching
• Empirical Studies
• Related Work
• Conclusions
Overview of Differencing Algorithm

Phase 1
- Match classes and interfaces
  - New classes and interfaces
  - Matched class and interface pairs
  - Deleted classes and interfaces

Phase 2
- Match methods
  - New methods
  - Matched method pairs
  - Deleted methods

Phase 3
- New statements
  - Match statements
    1. create Differencing Graphs (DiGs)
    2. compare statements

Matched class and interface pairs

New classes and interfaces

Deleted classes and interfaces

New methods

Matched method pairs

Deleted methods

New statements

Match statements

Phase 1

Phase 2

Phase 3

Match statements
void m1()

float n

d = 1/n;

void m2()

void m3(A a)

try {
...
}
catch (E1 e) { ... }
catch (Exception e) { ... }
n_float = 1/n_float

A

B

A.m1()

A.m1()

return

call a.m1()

catch Exception: E1, Exception: E1:E2

try

... catch Exception

exit

A

float n

void m1()

B

void m2()

void m3(A a)

Exception

E1

E2

n = 1/n;
a.m1();
try {
    ...
} catch (E1 e) { ... } catch (Exception e) { ... }
DiG: Simplify the extended CFG

G (DiG for m3 in P)

Entry

n_float = 1/ n_float

call a.m1()

A.m1()

return

try

EX

catch Exception: E1, Exception: E1: E2

... catch Exception

... exit

A

B
DiG: Simplify the extended CFG

G (DiG for m3 in P)

Entry

n_float = 1/ n_float

call a.m1()

HM1

A

A.m1()

B

A.m1()

return

try

HM2

EX

catch Exception:E1,
Exception:E1:E2

...

exit

...

catch Exception
DiG: Simplify the extended CFG

G (DiG for m3 in P)

Entry

n_float = 1/ n_float

call a.m1()

A

B

A.m1()

A.m1()

return

try

HM1

EX

catch Exception: E1, Exception: E1:E2

HM2

exit

G' (DiG for m3 in P')

Entry

n_double = 1/ n_double

int i_int=0;

call a.m1()

A

B

A.m1()

A.m1()

return

try

HM3

EX

catch Exception: E1

HM4

exit

...
Matching

G
Entry

n_float = 1/ n_float

unchanged

modified

call a.m1()

unchanged

return

try

catch Exception:E1, Exception:E1:E2

exit

G'
Entry

n_double = 1/ n_double

unchanged

int i_int=0;

unchanged

modified

call a.m1()

unchanged

A.m1()

unchanged

B.m1()

unchanged

A.m1()

unchanged

B.m1()

unchanged

A.m1()

unchanged

B.m1()

unchanged

Look-ahead limit: 1
Similarity threshold: 0.5

4 unchanged matched (67)
6 compared (.67)
public class A {
    double n;
    public void m1() { ... } }
public class B extends A {
    public void m1() {
        ...
    }
    public void m2() { ... }
    public void m3() {
        n = 1/n;
        int i = 0;
        a.m1();
        try {
            ...}
catch (E1 e) { ... }
catch (Exception e) { ... }
}
public class E1 extends Exception {
    ...
}
public class E2 extends Exception {
    ...
}
Outline

• Introduction
• Differencing Algorithm
  • Representation
  • Matching
• Empirical Studies
• Related Work
• Conclusions
Empirical Studies

Experimental Setup

- JDiff: A Java implementation of our technique
- Subject: Jaba
  - A Java bytecode analysis tool
  - 60KLOC (550 classes, 2800 methods)
  - 2 sets of 4 consecutive versions
    - Low activity: v1, …, v4 (3-20 changes)
    - High activity: va, …, vd (15-150 changes)

Studies

1. Efficiency of our algorithm
2. Effectiveness of our algorithm in matching
3. Effectiveness of our algorithm for a maintenance task
Study 1: Efficiency of Our Algorithm

Goal: Measure the efficiency of our algorithm for various look-ahead limits and hammock similarity thresholds

Method:
1. Run JDiff
   - Low-activity versions (v1-v2, v1-v3, and v1-v4)
   - Various look-ahead limits (0-50)
   - Various similarity thresholds (0-1)
2. Collect the running times.
Study 1: Efficiency of Our Algorithm

- $v_1 - v_2$ (S>0.2)
- $v_1 - v_3$ (S>0.2)
- $v_1 - v_4$ (S>0.2)
- $v_1 - v_2$ (S=0)
- $v_1 - v_3$ (S=0)
- $v_1 - v_4$ (S=0)
Study 3: Effectiveness for a Maintenance Task

Goal:
Assess the effectiveness of our algorithm for a maintenance task (coverage estimation)

Method:
- Jaba’s regression test suite (~60% coverage)
- Both low- and high-activity versions (v1-v2, v1-v3, v1-v4, va-vb, va-vc, and va-vd)
- For each pair (vi-vj),
  1. Collect coverage for vi
  2. Run JDiff on vi-vj to get mappings
  3. Get estimated coverage of vj based on mappings
  4. Collect actual coverage for vj
  5. Compare actual and estimated coverage of vj
### Study 3: Effectiveness for a Maintenance Task

<table>
<thead>
<tr>
<th>Pair ( vi,vj )</th>
<th>Avg. Correctly estimated for ( vj ) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v1,v2 )</td>
<td>98.57</td>
</tr>
<tr>
<td>( v1,v3 )</td>
<td>98.46</td>
</tr>
<tr>
<td>( v1,v4 )</td>
<td>98.03</td>
</tr>
<tr>
<td>( va,vb )</td>
<td>96.25</td>
</tr>
<tr>
<td>( va,vc )</td>
<td>86.08</td>
</tr>
<tr>
<td>( va,vd )</td>
<td>84.70</td>
</tr>
</tbody>
</table>

#### High-activity period

#### Low-activity period
Outline

• Introduction
• Differencing Algorithm
  • Representation
  • Matching
• Empirical Studies
• Related Work
• Conclusions
Related Work

Textual
- E.W. Myers. *Algorithmica* 1986 (UNIX diff)

Control-flow graph based

Dependence graph based
- S. Horwitz. *PLDI* 1990
- D. Binkley. *ICSM* 1992

Abstract syntax tree based
- Raghavan et al. *ICSM* 2004 (Dex)

Input-output dependence based
- D. Jackson. *ICSM* 1994 (Semantic diff)
Conclusions

Contributions

• A differencing algorithm that
  • Based on a new graph representation which models object-oriented features
  • Uses several strategies to increase matching capability
• A tool that implements our technique (JDiff)
• A set of studies that show the efficiency and effectiveness of the approach

Future Directions

• To improve matching results
  • Investigate additional heuristics
  • Use common change patterns
• Test-suite augmentation
  • Create new test cases based on changes in the program
Questions?