Contextual data flow analysis for structural testing of object oriented systems

Mauro Pezzè
University of Lugano
Contextual data flow analysis for structural testing of object oriented systems
“Structural” testing

Judging test suite thoroughness based on the structure of the program itself

Also known as white-box, glass-box, or code-based testing

To distinguish from functional (requirements-based, black-box) testing
Why structural testing?

One way of answering the question “What is *missing* in our test suite?”

If part of a program is not executed by any test case in the suite, faults in that part cannot be exposed.

what’s a “part”?

A control flow element or combination:
- Statements, Branches, Conditions, Paths,…

Complements functional testing: Another way to recognize cases that are treated differently.
Contextual **data flow analysis** for structural testing of object oriented systems
Data flow concept

Value of \( x \) at 6 could be computed at 1 or at 4

Bad computation at 1 or 4 could be revealed only if they are used at 6

(1,6) and (4,6) are def-use \((DU)\) pairs

defs at 1,4

use at 6
**Definition-clear path**

1,2,3,5,6 is a definition-clear path from 1 to 6

x is not re-assigned between 1 and 6

1,2,4,5,6 is not a definition-clear path from 1 to 6

the value of x is “killed” (reassigned) at node 4

(1,6) is a DU pair because 1,2,3,5,6 is a definition-clear path
Adequacy criteria

All DU pairs:
   Each DU pair is exercised by at least one test case

All DU paths:
   Each *simple* (non looping) DU path ...

All definitions:
   For each definition, there is at least one test case which exercises a DU pair containing it
   (Every computed value is used somewhere)

...
Contextual data flow analysis for structural testing of object oriented systems
Is data flow testing the right choice for OO-SW?

DU associations of instance vars across methods identify relevant **sequences** of methods invocation

A DU association is tested by executing paths from the assignment (definition) to the point where the variable is used (use)

- **Msg(), getInfo()**
- **Msg(), setInfo(byte b), getInfo()**
key to testing OO software: state dependent behavior

object-oriented software: **simple individual methods** create complex behavior through **sequences** of interactions on **encapsulated state**

Implications for testing:

Focus on **encapsulated state**

**Data flow** testing: Right idea (interaction through state), but needs extension for OO software

Focus on interactions:

**sequences** of method calls **in context of a call hierarchy**
Moving to integration testing...

...Classic data flow testing is not enough

What if an object incorporates another object in its state?

```java
class Msg {
    private byte info;
    public Msg() { info = 0; }
    public void setInfo(byte b) { info = b; }
    public byte getInfo() { return info; }
}

class Storage {
    private Msg msg;
    private byte stored;
    public Storage() {
        msg = new Msg();
        stored = 0;
    }
    public void setStored(byte b) { stored = b; }
    public byte getStored() { return stored; }
    public void recvMsg(Msg m) {
        byte recv = m.getInfo();
        msg.setInfo(recv);
    }
    public void storeMsg() {
        byte b = msg.getInfo();
        setStored(b);
    }
}
```

Definition 1

Use 1

Diagram: Message flow and class interactions.
Contextual definitions and uses

**Contextual definition/use** for a variable $v$: $<d, cd> <u, cu>$

$d =$ **Definition** of a variable $v$: location of the assignment

$u =$ **Use** of a variable $v$: location of the use

A **context** $(cd, cu)$ is the chain of method invocations that leads to the definition/use.

```java
class Storage {
    private Msg msg;
    private byte stored;
    public Storage(){
        msg = new Msg();
        stored = 0;
    }
    public void setStored(byte b){stored = b;}
    public byte getStored(){return stored;}
    public void recvMsg(Msg m){
        byte recv = m.getInfo();
        msg.setInfo(recv);
    }
    public void storeMsg(){
        byte b = msg.getInfo();
        setStored(b);
    }
}
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Method</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>Msg()</td>
<td>Msg()</td>
</tr>
<tr>
<td>msg.info</td>
<td>Msg()</td>
<td>Storage():Msg()</td>
</tr>
<tr>
<td>info</td>
<td>setInfo()</td>
<td>setInfo()</td>
</tr>
<tr>
<td>msg.info</td>
<td>setInfo()</td>
<td>rcvMsg():setInfo()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Method</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>getInfo()</td>
<td>getInfo()</td>
</tr>
<tr>
<td>msg.info</td>
<td>getInfo()</td>
<td>storeMsg():getInfo()</td>
</tr>
</tbody>
</table>
Contextual DU associations

**DU association** for a variable \(v\): \(<d,u>\)

**Contextual DU association** for a variable \(v\): \(<d,u,cd,cu>\)

### Defs in Message (Msg)

<table>
<thead>
<tr>
<th>Field</th>
<th>Method</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>Msg()</td>
<td>Msg()</td>
</tr>
<tr>
<td>info</td>
<td>setInfo()</td>
<td>setInfo()</td>
</tr>
</tbody>
</table>

### Uses in Message (Msg)

<table>
<thead>
<tr>
<th>Field</th>
<th>Method</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>getInfo()</td>
<td>getInfo()</td>
</tr>
</tbody>
</table>

### Defs in Storage

<table>
<thead>
<tr>
<th>Field</th>
<th>Method</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg.info</td>
<td>Msg()</td>
<td>Storage():Msg()</td>
</tr>
<tr>
<td>msg.info</td>
<td>setInfo()</td>
<td>rcvMsg():setInfo()</td>
</tr>
</tbody>
</table>

### Uses in Storage

<table>
<thead>
<tr>
<th>Field</th>
<th>Method</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg.info</td>
<td>getInfo()</td>
<td>storeMsg():getInfo()</td>
</tr>
</tbody>
</table>

### DU associations in Message (Msg)

<table>
<thead>
<tr>
<th>Field</th>
<th>Def context</th>
<th>Use context</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>Msg()</td>
<td>getInfo()</td>
</tr>
<tr>
<td>info</td>
<td>setInfo()</td>
<td>getInfo()</td>
</tr>
</tbody>
</table>

### DU associations in Storage

<table>
<thead>
<tr>
<th>Field</th>
<th>Def context</th>
<th>Use context</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg.info</td>
<td>Storage():Msg()</td>
<td>storeMsg():getInfo()</td>
</tr>
<tr>
<td>msg.info</td>
<td>rcvMsg():setInfo()</td>
<td>storeMsg():getInfo()</td>
</tr>
</tbody>
</table>
Is contextual data flow testing the right choice for OO-SW?

**Scalability:**
  Can we analyze large programs?

**Feasibility:**
  How many infeasible contextual def-use associations?

**Effectiveness:**
  Can we reveal failures?
## Scalability

Analysis of sample open source programs of increasing complexity

<table>
<thead>
<tr>
<th></th>
<th>No. classes</th>
<th>SLOC</th>
<th>No. fields</th>
<th>Max no. assoc. (no outliers)</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JEdit</td>
<td>910</td>
<td>92,213</td>
<td>2,975</td>
<td>381</td>
<td>70</td>
</tr>
<tr>
<td>Ant</td>
<td>785</td>
<td>80,454</td>
<td>4,081</td>
<td>331</td>
<td>60</td>
</tr>
<tr>
<td>BCEL</td>
<td>383</td>
<td>23,631</td>
<td>929</td>
<td>792</td>
<td>24</td>
</tr>
<tr>
<td>Lucene</td>
<td>287</td>
<td>19,337</td>
<td>1,013</td>
<td>410</td>
<td>17</td>
</tr>
<tr>
<td>JTopas</td>
<td>63</td>
<td>5,359</td>
<td>196</td>
<td>49</td>
<td>8</td>
</tr>
<tr>
<td>NanoXML</td>
<td>25</td>
<td>3,279</td>
<td>39</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Siena</td>
<td>27</td>
<td>2,162</td>
<td>66</td>
<td>35</td>
<td>5</td>
</tr>
</tbody>
</table>
Feasibility

amount of infeasible associations

JUnit

- 96% Feasible
- 4% Infeasible

99 feasible assoc
3 infeasible assoc

Lucene

- 20% Feasible
- 80% Infeasible

280 feasible assoc.

74 infeasible assoc.

Structural coverage criteria feasibility is usually between 75% and 95%.
Effectiveness

Compare the **original test suite** of JUnit and a **test suite that covers 100% of feasible contextual associations**.

The original test suite misses some important checks:

- Consistency of the initial state of objects
- Some features
- States that violate the preconditions of the methods
Effectiveness

seed **83 faults**

Compare different test suites:

- Original test suite of JUnit
- Test suite that covers 100% of feasible statements
- Test suite that covers 100% of feasible contextual def-use associations

![Bar chart showing effectiveness comparison](chart.png)

- original
- 100% statement
- 100% du associations
M. J. Harrold, G. Rothermel,
Performing data flow testing on classes
FSE 1994

L. Souter, L. L. Pollock,
The Construction of Contextual Def-Use Associations for Object-Oriented Systems
IEEE TSE 2003

G. Denaro, A. Gorla, M. Pezzè,
Contextual Integration Testing of Classes
ETAPS/FASE 2008