Can I stop testing now??

Test adequacy metrics beyond cover

Ramsay Taylor
Test Adequacy

Erl
Test Adequacy
Test Adequacy

Tests
--------
--------
--------
--------

Erl

Results
------ok
------ok
------ok
------ok
------ok
Test Adequacy

Tests
-----
-----
-----
-----
-----

ErI

Results
----- ok
----- ok
----- ok
----- ok
----- ok

Done?!?
Test Adequacy
What we want from Test Adequacy?

• Have we tested all of the code?

• Have we tested it in all meaningful ways?

• If the answer to either question is “no”, how can I do better?
In this talk

• Code Coverage
  - Testing all of the code that you have written
  - Testing it in meaningful ways

• Mutation Testing
  - Testing the code you might have written…
  - Testing the code in novel ways
  - Actually checking the answers!

• Model Inference
Code Coverage

-module(abitfittest).
-export([dv/2]).

dv(A,B) ->
  if (A == 0) and (B > 4) ->
    B;
  true ->
    B / A
  end.
-module(abififtest).
-export([dv/2]).

dv(A,B) ->
    if (A == 0) and (B > 4) ->
        B;
    true ->
        B / A
end.

dv(0,5)
Code Coverage

-module(abifstest).
-export([[dv/2]]).

dv(A,B) ->
  if (A == 0) and (B > 4) ->
    dv(0,5)
    dv(5,5)
  B;
  true ->
    B / A
  end.
** exception error: an error occurred when evaluating an arithmetic expression in function abiftest:dv/2 (abiftest.erl, line 8)
Modified Condition/Decision Coverage

• Instrument not just what got called, but in what way

• Focus on decision points not large blocks of sequential lines

• Measure/require all (reasonable) ways of taking or not taking a branch
-module(abiftest).
-export([dv/2]).

dv(A,B) ->
  if (A == 0) and (B > 4) ->
    B;
  true ->
    B / A
end.

dv(0,5)
dv(5,5)
-module(abiftest).
-export([dv/2]).

dv([A,B]) ->
    if (A == 0) and (B > 4) ->
        B;
    true ->
        B / A
    end.

dv(0,5)
dv(5,5)
dv(5,0)
-module(abiftest).
-export([dv/2]).

dv([A,B]) ->
  if (A == 0) and (B > 4) ->
    B;
    true -> B / A
  end.

(A == 0) and (B > 4)

- matched: 1
- non-matched: 2

When false:

<table>
<thead>
<tr>
<th></th>
<th>matched</th>
<th>non-matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>A == 0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>B &gt; 4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Pattern Matching

-module(abtest).
-export([[dv/2]]).

dv([0,5]) -> 5;

dv([A,B]) -> B / A.

[0..5]

- matched: 1
- non-matched: 2

When non-matched:

<table>
<thead>
<tr>
<th>matched</th>
<th>non-matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
Pattern Matching

```prolog
-module(abcasetest).
-export([dv/2]).

dv([A,B]) ->
  case {A,B} of
    {0,5} -> B;
    _ -> B / A
  end.
```

\{0,5\}

- matched: 1
- non-matched: 2

When non-matched:

<table>
<thead>
<tr>
<th></th>
<th>matched</th>
<th>non-matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Pattern Matching

```prolog
-module(ablisttest).
-export([[dv/1]]).

dv(Arg) ->
  case Arg of
    [0,5] -> 0;
    [A,B] -> B / A
  end.
```

[0, 5]

- matched: 1
- non-matched: 2

When non-matched:

<table>
<thead>
<tr>
<th></th>
<th>matched</th>
<th>non-matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>empty_list</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>list_size_mismatch</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>not_a_list</td>
<td>0</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Pattern Matching

dv_proc() ->
receive
   {From, {A,B}} when (A == 0) and (B > 4) ->
      From ! B;
   {From, {A,B}} ->
      From ! B / A
end.
Code coverage limitations

• Only assess the code that you have written, not the code you should have written…

• Says nothing except that the code has been executed and maybe didn’t crash.
Mutation Testing

• Deliberately break the code and see if the tests “notice”

• Try to simulate common faults
  - with the system
  - with the programmer…
Test results per mutant

- Fails - Good! It found the fault
- Passed - Bad! It didn’t notice the change
  - unless its “semantically equivalent”
mu2 Framework

• Allows domain-specific operators to be supplied
• Uses the Wrangler refactoring library to allow rich and subtle mutation operators
mu2 Operators

{plus_to_minus,
  ?MUTATION_MATCH("X@ + Y@"),
  ?MUTATION_EXCHANGE("X@ + Y@", "X@ - Y@")}
{swap_case_order,
  ?MUTATION_MATCH("if Guards@@@ -> Body@@@ end"),
  ?MUTATION("if Guards@@@ -> Body@@@ end",
    begin
      A = random:uniform(length(Guards@@@)),
      B = random_not_n(length(Guards@@@), A),
      NewGuards@@@ = swap(Guards@@@, A, B),
      NewBody@@@ = swap(Body@@@, A, B),
      ?TO_AST("if NewGuards@@@ -> NewBody@@@ end")
    end)
}
mu2 Operators

{decrease_timeout,
?MUTATION_MATCH("receive
  Pats@@@ when Guards@@@@ -> Body@@@@
  after APats@@@@ -> ABody@@@@
  end"),

?MUTATION("receive
  Pats@@@@ when Guards@@@@ -> Body@@@@
  after APats@@ -> ABody@@ end",
begin
  NewAPats@@ = lists:map(fun(Pat@) ->
    ?TO_AST("(Pat@ / 100)")
  end,
  APats@@),

?TO_AST("receive
  Pats@@@@ when Guards@@@@ -> Body@@@@
  after NewAPats@@@@ -> ABody@@@@
  end")
end)}
Mutation testing limitations

• Have to compile lots of mutants
• Have to run the test set lots of times
Model Inference
Conclusions

• You should be testing your tests
  - but don’t ask me to recurse again ;)

• Code coverage is cheap so use it
  - but do it properly!

• Mutation testing is a useful complement
  - but its expensive so use it wisely…

• Model inference is cool!
  - look into it
Prototypes…

https://github.com/ramsay-t/Smother

https://github.com/ramsay-t/mu2

http://statechum.sourceforge.net/
Questions?