Can I stop testing yet?
Test adequacy metrics as feedback for automated test generation

Presented by Ramsay Taylor
Test Adequacy

Test Generation

Tests

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

Done?!?
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 done badly

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

dv(A,B) ->
    if (A == 0) and (B > 4) ->
        B;
        true ->
            B / A
    end.
```
Code Coverage done badly

```erlang
-module(abiftest).
-export([dv/2]).

dv(A,B) ->
  if (A == 0) and (B > 4) ->
    B;
    true ->
      B / A
  end.
```
Code Coverage done badly

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

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

** exception error: an error occurred when evaluating an arithmetic expression in function `abiftest:dv/2` (abiftest.erl, line 8)

dv(0,5)  dv(5,5)  dv(0,2)
Code Coverage done better
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
Code Coverage done better

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

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

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

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

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

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

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

- matched: 1
- non-matched: 2

When false:

<table>
<thead>
<tr>
<th>A</th>
<th>matched</th>
<th>non-matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>B &gt; 4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```
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...
Mutation Testing

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