



Software Testing with QuickCheck

Lecture 1
Properties and Generators

Testing



- How do we know software works?
 - We test it!
- "lists:delete removes an element from a list"

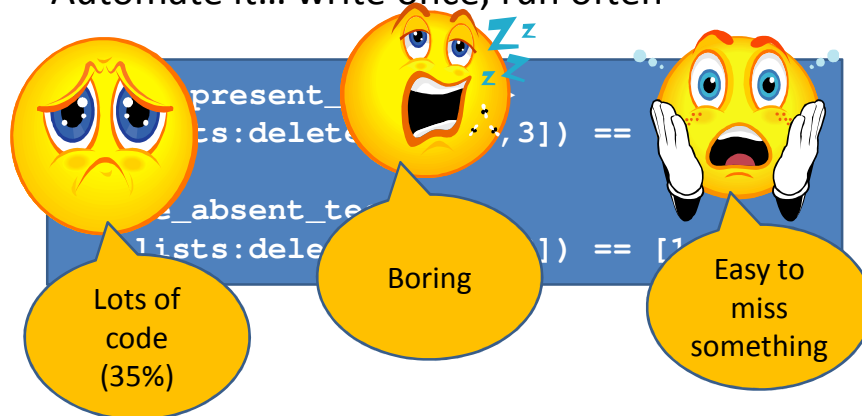
```
4> lists:delete(2, [1, 2, 3]) .  
[1, 3]  
5> lists:delete(4, [1, 2, 3]) .  
[1, 2, 3]
```

- ... seems to work!

Automated Testing



- Testing accounts for ~50% of software cost!
- Automate it... write once, run often



Property-based testing



- Generalise test cases

$\forall \{I, L\} \in \text{int()} \times \text{list}(\text{int}())$

```
prop_delete() ->
  ?FORALL({I, L},
    {int(), list(int())},
    not lists:member(I,
      lists:delete(I, L)) .
```

```
21> eqc:quickcheck(examples:prop_delete()) .
```

```
.....
.....
OK, passed 100 tests
```

Properties

Q...

Bound variable

?FORALL(N, int(), N*N >= 0)

Test case generator

Test oracle

- We test directly against a formal specification

More tests...

Q...

```

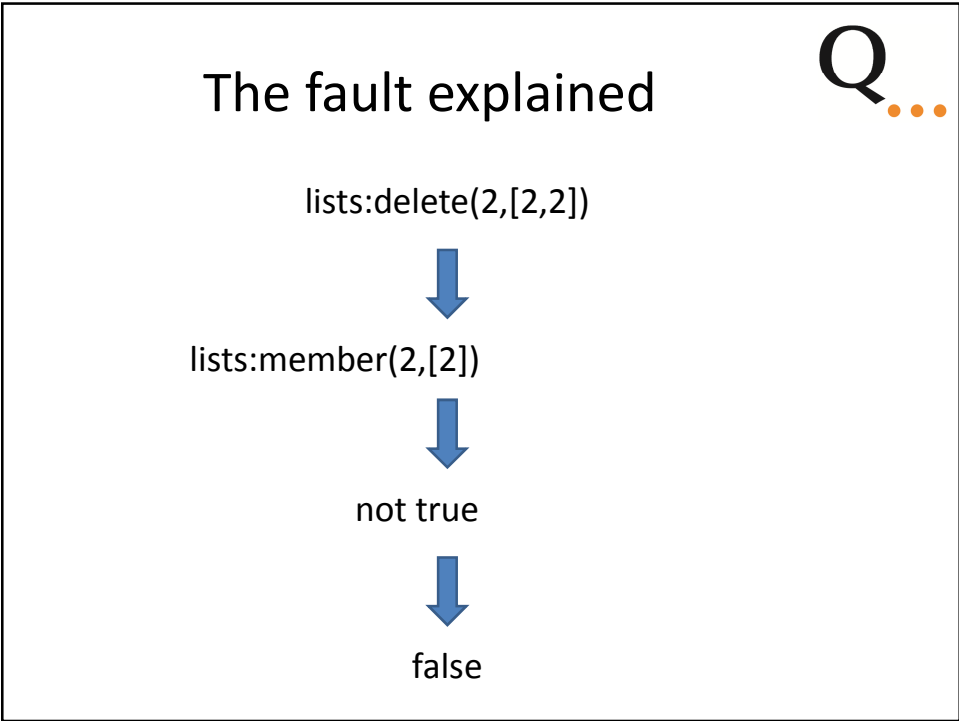
29> eqc:quickcheck (eqc:numtests (1000, examples:prop_delete()) ).
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
...Failed! After 346 tests.
{2, [-7, -13, -15, 2, 2]}
Shrinking. (1 times)
{2, [2, 2]}
false

```

A failed test

A simplest failing test

c.f. ?FORALL ({ I, L }, ..., ...)



Properties with preconditions Q...

- The property has no duplicates

```
no_duplicates(L) ->
  lists:usort(L)
  == lists:sort(L).
```

```

prop_delete() ->
  ?FORALL({I,L},
    {int(),list(int())},
    ?IMPLIES(no_duplicates(L),
      not lists:member(I,lists:delete(I,L)))).
    
```

```

39> eqc:quickcheck(examples:prop_delete()).
.....x.....x.....x.x
.....xx.....x...x...xx...x.x.....x.
x.....x..
OK, passed 100 tests
    
```

Skipped tests

Custom generators



- Why not *generate* lists without duplicates in the first place?

```
ulist (Elem) ->
  ?LET (L, list (Elem),
        lists:usort (L)) .
```

First:
generate a
list L

- Use as ?FORALL (L, ulist (int ()))
- Generators are an *abstraction*
- ?LET for sequencing

Then: sort it
and remove
duplicates

Why was the error hard to find?



- $I \in \text{int}()$
 - $L \in \text{list}(\text{int}())$
- } What is the probability that I occurs in L —*twice*?

```
prop_delete () ->
  ?FORALL ({I, L},
           {int (), list (int ())},
           collect (lists:member (I, L),
                   not lists:member (I, lists:delete (I, L))) .
```

```
34> eqc:quickcheck (examples:prop_delete ()) .
.....
OK, passed 100 tests
88% false
12% true
true
```

Usually I doesn't even
occur *once*

Generate relevant tests



- Ensure that **I** is a member of **L**
 - Generate it *from L*

```
prop_delete_2() ->
  ?FORALL(L, list(int)),
  ?FORALL(I, elements(L),
    not lists:member(I, lists:delete(I,L))) .
```

```
45> eqc:quickcheck(examples:prop_delete_2()).
..xx.x.x.xx...x.x...x...x.....xx.....Failed! After 28 tests.
[-8,0,7,0]
0
Shrinking...(3 times)
[0,0]
0
```

Documenting misconceptions



- Useful to record that an expected property is *not true*

```
prop_delete_misconception() ->
  fails(
    ?FORALL(L, list(int)),
    ?IMPLIES(L /= [],
      ?FORALL(I, elements(L),
        not lists:member(I, lists:delete(I,L)))) .
```

```
49> eqc:quickcheck(examples:prop_delete_misconception()).
x...x.x...x.....OK, failed as expected. After 19
tests.
```

Good distribution ensures we falsify the property quickly

Remember!

Q...

- We test against a formal specification!
 - Often it is the *specification* which is wrong!
- We don't see the test data!
 - 100 passing tests can give a false sense of security
- Collect statistics!
 - Ensure a good test case distribution



QuviQ

Exercises