CODE COVERAGE ISN'T COVERAGE

Wayne Roseberry

Microsoft

Author of "Writing Test Plans Made Easy"

The Rules

- People respond to rewards and punishments
- Don’t Chase The Numbers
- High Code Coverage <> Good
  - *But low coverage == Bad*

PEOPLE WILL DO DUMB THINGS WHEN THE METRIC IS CODE COVERAGE

What To Do?

- Stop Using Code Coverage As A “Good Enough” Marker
- Use Code Coverage as a “Not Good Enough” Gate
- Use Code Coverage To Inspire Test Analysis

Seaside Aquarium, Seaside OR

- Goal: Herring

- Behavior:
  - Spinning
  - Spitting
  - Croaking
  - Clapping

Testers as Sea Lions

- Goal/Reward: High code coverage numbers

- Behavior
  - Build tests for uninteresting code paths
  - Build tests that don’t actually test the feature
  - Build tests that give a false illusion of sufficiency

  Two Examples:
  - Chasing the Numbers
  - Good Isn’t Good

EXAMPLE 1: CHASING THE NUMBERS

Covered	`switch (fld.type)` `{`
Covered	`Case FLD_STRING:` `  return ExtractStringFromField(fld);`
Covered	`Case FLD_DATE:` `  return ExtractDateFromField(fld);`

Not Covered	`Else`
	`  return null;`
	`}`

Tester Conclusion:

1. Create special build with hook *(method was internal)*
2. Call function directly, setting fld.type = <value other than…>

EXAMPLE 1: CHASING THE NUMBERS

Covered	`switch (fld.type)` `{`
Covered	`Case FLD_STRING:` `  return ExtractStringFromField(fld);`
Covered	`Case FLD_DATE:` `  return ExtractDateFromField(fld);`

Not Covered	`Else`
	`  return null;`
	`}`

Bigger Problem…

- Why are all other values of fld.type treated equivalent?
- Why didn't the test code use other values for fld.type?
- The consuming components have never been tested with a NULL return value. What if…?

```
object f = (fieldobject) returnfielddata(foo);
display(f.ToString()); // will throw if return was null
```

MOST BUGS ARE FROM MISSING CODE BLOCKS!

EXAMPLE 2: GOOD ISN'T GOOD

The product code…

```
public class SPQuery
{
// A BUNCH OF PROPERTIES AND METHODS OMITTED
// HERE FOR SAKE OF SLIDES…
  public string Query()
  {
     get {return m_Query;}
     set {m_Query = value;}
  }
}
```

Similar get/set pattern for most other properties.

EXAMPLE 2: GOOD ISN'T GOOD

The test code…

```
// instantiate the object, set its properties
SPQuery qry = new SPQuery();
qry.Query = queryXMLString;

if (qry.Query == queryXMLString)
{Log.Pass("Query matched expected value");}
else
{Log.Fail("Query did not match expected value");}
```

Similar pattern was used for all other properties.

EXAMPLE 2: GOOD ISN'T GOOD

Query Generator – how customers use it…

```
// instantiate the object, set its properties
SPQuery qry = new SPQuery();
qry.Query = queryXMLString; // the actual query
qry.DatesInUTC = true; // change date format
qry.AutoHyperlink = true; // render links as anchors

// fetch the items
SPListItemCollection items = splist.GetItems(qry);

// code continues, reading items
```

EXAMPLE 2: GOOD ISN'T GOOD

Problems:

1. **Testing wrong pattern**
   Customers typically set, then use properties. Test code only checked property persistence.
2. **Missing Properties That Affect Behavior**
   E.g. are results different based on property values?
3. **Missing Large Complex Data Domain**
   Queries act on data that exists already in lists
4. **Missing Large & Complex Data Format & Behavior Domain**
   Query string object complexity, query richness

What to do

- **Treat Code Coverage as a Probe, Not a Goal**
- **Ask “What does this mean?”**
- **Treat Every Missed Block as Bad, Not Every Covered Block as Good**

Covered Blocks	Uncovered Blocks
Increase domain coverage	Identify weak investments in large test domains
Remove superficial tests that don’t find bugs	Look for weakness in integration points
Identify opportunities to borrow, share test code	Correct or replace test patterns that do not use product appropriately
Hunt for missing code blocks for value differences	