A Lifecycle of Code Under Test

Bob Fornal

Twitter: @rfornal
https://dev.to/rfornal

This Talk

Overview

1. Define Inputs and Outputs
2. Initial Testing (coverage)
   a. All Branches (paths)
   b. Positive Testing
   c. Negative Testing
3. Handling Bugs (coverage)
4. Refactoring
5. REDESIGN ...
6. Abstraction
7. … Future Work (how are tests affected?)
   a. Black Box Testing
   b. White Box Testing

Programming is like sex: one mistake and you're providing support for a lifetime.
- Michael Sinz

An Overly Complicated Function To Add Two Strings

```
function overlyComplicated(a, b, len) {
  var sum = "";

  if (len < 1) {
    return "";
  }

  for (var i = 0; i < a.length; i++) {
    sum = sum + a[i];
  }
  for (var i = 0; i < b.length; i++) {
    sum = sum + b[i];
  }

  // "INJECTED" BUG HERE
  if (len === 2 || len === 4 || len === 6) {
    return "unexpected";
  }

  return sum.substr(0, len);
}

var oC = overlyComplicated;
```

Define Inputs and Outputs

```
function overlyComplicated(a, b, len) {
  var sum = "";

  if (len < 1) {
    return "";
  }

  for (var i = 0; i < a.length; i++) {
    sum = sum + a[i];
  }
  for (var i = 0; i < b.length; i++) {
    sum = sum + b[i];
  }

  // "INJECTED" BUG HERE
  if (len === 2 || len === 4 || len === 6) {
    return "unexpected";
  }

  return sum.substr(0, len);
}

var oC = overlyComplicated;
```

Inputs

- **a**: string of some length.
- **b**: string of some length.
- **len**: number (integer) of characters of the combined to return.

Outputs

- string of "len" characters.

Examples

- ("abc", "def", 0) returns ""
- ("abc", "def", 1) returns "a"
- ("abc", "def", 3) returns "abc"
- ("abc", "def", 5) returns "abcde"

Initial Testing (coverage)

```
function overlyComplicated(a, b, len) {
  var sum = "";

  if (len < 1) {
    return "";
  }

  for (var i = 0; i < a.length; i++) {
    sum = sum + a[i];
  }
  for (var i = 0; i < b.length; i++) {
    sum = sum + b[i];
  }

  // "INJECTED" BUG HERE
  if (len === 2 || len === 4 || len === 6) {
    return "unexpected";
  }

  return sum.substr(0, len);
}

var oC = overlyComplicated;
```

All Branches (paths)

- No Branches

Positive Testing

- `expect(oC("abc", "def", 1)).toEqual("a");`
- `expect(oC("abc", "def", 3)).toEqual("abc");`
- `expect(oC("abc", "def", 5)).toEqual("abcde");`

Negative Testing

- `expect(oC("abc", "def", 0)).toEqual("");`
- `expect(oC("abc", "def", -1)).toEqual("");`

Handling Bugs (coverage)

```
function overlyComplicated(a, b, len) {
  var sum = “”;

  if (len < 1) {
    return “”;
  }

  for (var i = 0; i < a.length; i++) {
    sum = sum + a[i];
  }
  for (var i = 0; i < b.length; i++) {
    sum = sum + b[i];
  }

  // ”INJECTED” BUG HERE
  if (len === 2 || len === 4 || len === 6) {
    return “unexpected”;
  }

  return sum.substr(0, len);
}

var oC = overlyComplicated;
```

Repeating The Bug In Test Form ...

```
expect(oC(“abc”, “def”, 2)).toEqual(“ab”);
```
- expect “unexpected” to equal “ab”.

```
expect(oC(“abc”, “def”, 4)).toEqual(“abcd”);
```
- expect “unexpected” to equal “abcd”.

```
expect(oC(“abc”, “def”, 6)).toEqual(“abcdef”);
```
- expect “unexpected” to equal “abcdef”.

Handling Bugs (coverage)

```
function overlyComplicated(a, b, len) {
  var sum = "";

  if (len < 1) {
    return "";
  }

  for (var i = 0; i < a.length; i++) {
    sum = sum + a[i];
  }
  for (var i = 0; i < b.length; i++) {
    sum = sum + b[i];
  }

  // "INJECTED" BUG HERE
  // if (len === 2 || len === 4 || len === 6) {
  //   return "unexpected";
  // }

  return sum.substr(0, len);
}

var oC = overlyComplicated;
```

After Fixing The Bug

- `expect(oC("abc", "def", 2)).toEqual("ab");`
- `expect(oC("abc", "def", 4)).toEqual("abcd");`
- `expect(oC("abc", "def", 6)).toEqual("abcdef");`

Refactoring

```
function overlyComplicated(a, b, len) {
  var sum = "";

  if (len < 1) {
    return "";
  }

  sum = a + b;
  sum = sum.substr(0, len);
  return sum;

  // for (var i = 0; i < a.length; i++) {
  //   sum = sum + a[i];
  // }
  // for (var i = 0; i < b.length; i++) {
  //   sum = sum + b[i];
  // }

  // return sum.substr(0, len);
}

var oC = overlyComplicated;
```

After Refactor
... Previous Tests Should Still Pass

Positive Testing

- expect(oCAS("abc", "def", 1)).toEqual("a");
- expect(oCAS("abc", "def", 3)).toEqual("abc");
- expect(oCAS("abc", "def", 5)).toEqual("abcde");

Negative Testing

- expect(oCAS("abc", "def", 0)).toEqual("");
- expect(oCAS("abc", "def", -1)).toEqual("");

Bug Testing

- expect(oCAS("abc", "def", 2)).toEqual("ab");
- expect(oCAS("abc", "def", 4)).toEqual("abcd");
- expect(oCAS("abc", "def", 6)).toEqual("abcdef");

Abstraction

```
function getSum(a, b) {
  return a + b;
}

function overlyComplicated(sumFn, a, b, len) {
  var sum = "";

  if (len < 1) {
    return "";
  }

  sum = sumFn(a, b).substr(0, len);
  // sum = a + b;
  // sum = sum.substr(0, len);
  return sum;
}

function oC(a, b, len) {
  return overlyComplicated(getSum, a, b, len);
}
```

After Abstraction
... Previous Tests Should Still Pass
... Should Add Tests For Abstracted Functionality
... Have Flexibility When Testing Injected Code

Positive, Negative, and Bug Testing

- All Pass

Abstraction

- `expect(getSum("abc", "dev")).toEqual("abcdef");`

Future Work

```
var global = {};

function getSum(a, b) {
  return a + b;
}

function overlyComplicated(sumFn, a, b, len) {
  var sum = "";

  if (len < 1) {
    return "";
  }

  sum = sumFn(a, b).substr(0, len);
  global.sum = sum;
  return sum;
}

function oC(a, b, len) {
  return overlyComplicated(getSum, a, b, len);
}
```

How Are Tests Affected?

- Per **Black-Box Testing,** no test should fail (purely examining inputs to outputs).
- Per **White-Box Testing,** tests should be written to cover the new code.

Future Work Tests

... given

- oC("abc", "def", 1);

... then

- expect(global.sum).toEqual("a");

Handling A/B Tests

- Branching considerations

Overview

1. Define Inputs and Outputs
2. Initial Testing (coverage)
   a. All Branches (paths)
   b. Positive Testing
   c. Negative Testing
3. Handling Bugs (coverage)
4. Refactoring
5. Abstraction
6. … Future Work (how are tests affected?)
   a. Black Box Testing
   b. White Box Testing

Questions ...