Correctness

Correctness

March 11, 2019

- How to Show a Programme Correct
- Summary

Introduction

- Airline booking system

Airline booking system

Departs	Arrives	Flight Number
6:25am	12:04pm	NW928
7:50am	1:28pm	NW344
10:15am	3:47pm	NW350
11:30am	5:16pm	NW588
12:40am	6:09am	NW360
3:25pm	9:01pm	NW354
5:00pm	10:31pm	NW358

Introduction

- Airline booking system
- Gym changing room access

Gym changing room access

The gym's response to the problem:

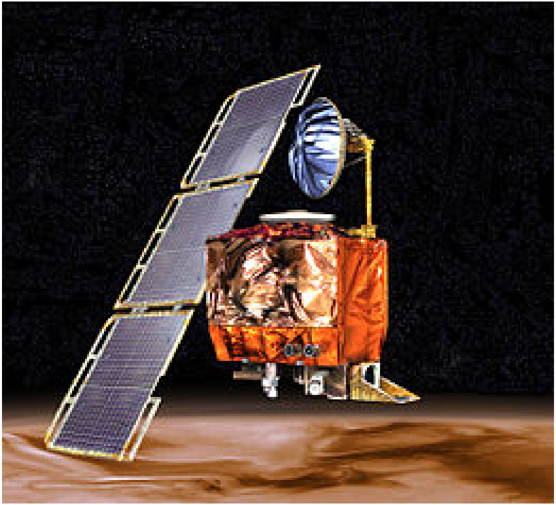
> *"The system is a product that we license and is therefore not something we built this way, but we are working on it. "In the meantime, we have removed the option of 'Dr' as a title to choose from when new members sign up, and are urging anyone who signed up to Pure Gym as a doctor recently to contact our membership team to prevent any issues entering changing rooms."*

They also stated "we have found a bug in the membership system. . . "

Introduction

- Airline booking system
- Gym changing room access
- Mars Climate Orbiter

Mars Climate Orbiter



\$327.6 million (1998 = \$500 million 2018)

Introduction

- Airline booking system
- Gym changing room access
- Mars Climate Orbiter
- Banking mail shot

Banking mail shot

- Actually telecoms company “gold card” mail shot

Banking mail shot

- Actually telecoms company “gold card” mail shot
- Some munged data

Banking mail shot

- Actually telecoms company “gold card” mail shot
- Some munged data
- Only a few letters sent out

Banking mail shot

- Actually telecoms company “gold card” mail shot
- Some munged data
- Only a few letters sent out
- One recipient had the letter framed

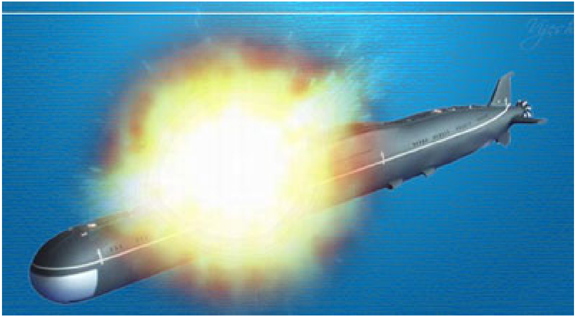
Banking mail shot

- Actually telecoms company “gold card” mail shot
- Some munged data
- Only a few letters sent out
- One recipient had the letter framed
- One non-recipient complained

Introduction

- Airline booking system
- Gym changing room access
- Mars Climate Orbiter
- Banking mail shot
- Torpedo system

Torpedo system



Example

1: How to Show a Programme Correct

1.1: Example: Calculating Squares

```java
public int square(int n) {
   int i = 0, square = 0, twoN = 0;
   while (i != n) {
      square = square + twoN + 1;
      twoN = twoN + 2;
      i++;
   }
   return square;
}
```

Prove the Programme

1.2: Prove the Programme

1.2.1: Writing Test Cases

The exception *proves* the rule

```
public class SquareTest extends junit.framework.TestCase {
    public void test0() {
        assertEquals(0, new Square.square(0));
    }
    public void test1() {
        assertEquals(1, new Square.square(1));
    }
    ⋮
}
```

Prove the Programmes Correct

1.3: Prove the Programme Correct

Use *assertions*

1.3.1: A Simple Example

$\{\mathbf{twoN = 2i}\}$

```
twoN = twoN + 2;
```

$\{\mathbf{twoN = 2(i + 1)}\}$

The Proof

1.3.2: The Proof

1.3.2 A: The aim

```
public int square(int n) {
    int i = 0, square = 0, twoN = 0;
    while (i != n) {
        square = square + twoN + 1;
        twoN = twoN + 2;
        i++;
    }
    6 {square = n^2}
    return square;
}
```

The Proof

1.3.2 B: Introduce the loop variable

```
public int square(int n) {
    int i = 0, square = 0, twoN = 0;
    while (i != n) {
        square = square + twoN + 1;
        twoN = twoN + 2;
        i++;
    }
```

$\boxed{6}\{\mathbf{i = n, square = i^2 = n^2}\}$

```
    return square;
}
```

The Proof

1.3.2 C: Add necessary preconditions

```
public int square(int n) {
    int i = 0, square = 0, twoN = 0;
    $\boxed{1}\{square = i^2\}$
    while (i != n) {
        square = square + twoN + 1;
        twoN = twoN + 2;
        i++;
        $\boxed{5}\{square = i^2\}$
    }
    $\boxed{6}\{i = n, square = i^2 = n^2\}$
    return square;
}

```

The Proof

1.3.2 D: A free assertion

```
public int square(int n) {
    int i = 0, square = 0, twoN = 0;
    $\boxed{1}\{\mathbf{square} = \mathbf{i}^2\}$
    while (i != n) {
        $\boxed{2}\{\mathbf{square} = \mathbf{i}^2\}$
        square = square + twoN + 1;
        twoN = twoN + 2;
        i++;
        $\boxed{5}\{\mathbf{square} = \mathbf{i}^2\}$
    }
    $\boxed{6}\{\mathbf{i} = \mathbf{n}, \mathbf{square} = \mathbf{i}^2 = \mathbf{n}^2\}$
    return square;
}

```

The Proof

1.3.2 E: Add more preconditions

```
public int square(int n) {
    int i = 0, square = 0, twoN = 0;
    $\boxed{1}\{\mathsf{square} = i^2\}$
    while (i != n) {
        $\boxed{2}\{\mathsf{square} = i^2\}$
        square = square + twoN + 1;
        twoN = twoN + 2;
        $\boxed{4}\{\mathsf{square} = (i + 1)^2\}$
        i++;
        $\boxed{5}\{\mathsf{square} = i^2\}$
    }
    $\boxed{6}\{i = n, \mathsf{square} = i^2 = n^2\}$
    return square;
}

```

The Proof

1.3.2 F: Another free assertion

```
public int square(int n) {
    int i = 0, square = 0, twoN = 0;
    $\boxed{1}\{\mathbf{square = i^2}\}$
    while (i != n) {
        $\boxed{2}\{\mathbf{square = i^2}\}$
        square = square + twoN + 1;
        $\boxed{3}\{\mathbf{square = (i + 1)^2}\}$
        twoN = twoN + 2;
        $\boxed{4}\{\mathbf{square = (i + 1)^2}\}$
        i++;
        $\boxed{5}\{\mathbf{square = i^2}\}$
    }
    $\boxed{6}\{\mathbf{i = n, square = i^2 = n^2}\}$
    return square;
}
```

The Proof

1.3.2 G: Rewrite $(\mathtt{i} + 1)^2$

```
public int square(int n) {
    int i = 0, square = 0, twoN = 0;
    $\boxed{1}\{\mathbf{square} = \mathbf{i}^2\}$
    while (i != n) {
        $\boxed{2}\{\mathbf{square} = \mathbf{i}^2\}$
        square = square + twoN + 1;
        $\boxed{3}\{\mathbf{square} = \mathbf{i}^2 + 2\mathbf{i} + 1 = (\mathbf{i} + 1)^2\}$
        twoN = twoN + 2;
        $\boxed{4}\{\mathbf{square} = (\mathbf{i} + 1)^2\}$
        i++;
        $\boxed{5}\{\mathbf{square} = \mathbf{i}^2\}$
    }
    $\boxed{6}\{\mathbf{i} = \mathbf{n}, \mathbf{square} = \mathbf{i}^2 = \mathbf{n}^2\}$
    return square;
}
```

The Proof

1.3.2 H: Some wishful thinking

```
public int square(int n) {
    int i = 0, square = 0, twoN = 0;
    $\boxed{1}\{\mathbf{square} = \mathbf{i}^2\}$
    while (i != n) {
        $\boxed{2}\{\mathbf{square} = \mathbf{i}^2, \mathbf{twoN} = \mathbf{2i}\}$
        square = square + twoN + 1;
        $\boxed{3}\{\mathbf{square} = \mathbf{i}^2 + \mathbf{2i} + \mathbf{1} = (\mathbf{i} + \mathbf{1})^2\}$
        twoN = twoN + 2;
        $\boxed{4}\{\mathbf{square} = (\mathbf{i} + \mathbf{1})^2\}$
        i++;
        $\boxed{5}\{\mathbf{square} = \mathbf{i}^2\}$
    }
    $\boxed{6}\{\mathbf{i} = \mathbf{n}, \mathbf{square} = \mathbf{i}^2 = \mathbf{n}^2\}$
    return square;
}

```

The Proof

1.3.2 I: Add some assertions for `twoN`

```
public int square(int n) {
    int i = 0, square = 0, twoN = 0;
    $\boxed{1}\{\mathbf{square} = \mathbf{i}^2, \mathbf{twoN} = 2\mathbf{i}\}$
    while (i != n) {
        $\boxed{2}\{\mathbf{square} = \mathbf{i}^2, \mathbf{twoN} = 2\mathbf{i}\}$
        square = square + twoN + 1;
        $\boxed{3}\{\mathbf{square} = \mathbf{i}^2 + 2\mathbf{i} + 1 = (\mathbf{i} + 1)^2, \mathbf{twoN} = 2\mathbf{i}\}$
        twoN = twoN + 2;
        $\boxed{4}\{\mathbf{square} = (\mathbf{i} + 1)^2, \mathbf{twoN} = 2(\mathbf{i} + 1)\}$
        i++;
        $\boxed{5}\{\mathbf{square} = \mathbf{i}^2, \mathbf{twoN} = 2\mathbf{i}\}$
    }
    $\boxed{6}\{\mathbf{i} = \mathbf{n}, \mathbf{square} = \mathbf{i}^2 = \mathbf{n}^2, \mathbf{twoN} = 2\mathbf{i}\}$
    return square;
}
```

The Proof

1.3.2 J: The final proof

```
public int square(int n) {
    int i = 0, square = 0, twoN = 0;
    $\boxed{1}\{\mathrm{square} = i^2, \mathrm{twoN} = 2i\}$
    while (i != n) {
        $\boxed{2}\{\mathrm{square} = i^2, \mathrm{twoN} = 2i\}$
        square = square + twoN + 1;
        $\boxed{3}\{\mathrm{square} = i^2 + 2i + 1 = (i+1)^2, \mathrm{twoN} = 2i\}$
        twoN = twoN + 2;
        $\boxed{4}\{\mathrm{square} = (i+1)^2, \mathrm{twoN} = 2i + 2 = 2(i+1)\}$
        i++;
        $\boxed{5}\{\mathrm{square} = i^2, \mathrm{twoN} = 2i\}$
    }
    $\boxed{6}\{i = n, \mathrm{square} = i^2 = n^2, \mathrm{twoN} = 2i\}$
    return square;
}
```

Assertions in Java

1.3.3: Assertions in Java

```
public int square(int n) {
    int square = 0, twoN = 0;
    for (int i = 0; i < n; i++) {
        assert square == i*i :  square; assert twoN == 2*i :
        square = square + twoN + 1;
        assert square == (i+1)*(i+1); assert twoN == 2*i;
        twoN = twoN + 2;
        assert(square == (i+1)*(i+1)); assert(twoN == 2*(i
    }
    assert square == n*n :  square; assert twoN == 2*n;
    return square;
}
```

Improve a Correct Programme

1.4: Improve a Correct Programme

1.4.1: A Simple Example: Recursion Elimination

```
public SomeType f(int n) {
    if (n == 0) {
        return aResult;
    }  else {
        return g(f(n-1));
    }
}
```

An Example

- $\mathtt{f}(0) = \mathtt{aResult}$
- $\mathtt{f}(1) = \mathtt{g}(\mathtt{aResult})$
- $\mathtt{f}(2) = \mathtt{g}(\mathtt{g}(\mathtt{aResult}))$
- ...
- $\mathtt{f}(\mathtt{n}) = \mathtt{g}^{\mathtt{n}}(\mathtt{aResult})$
- ...

An Example

```
public SomeType f(int n) {
    SomeType result = aResult;
    int i = 0;
    while (i != n) {
        result = g(result);
        i++;
    }
    return result;
}
```

First Attempt

1.4.2: The transformation

1.4.2 A: First attempt

1.4.2 A(i): Begin with a simple programme

```
public int square(int n) {
    int square;
    square = $n^2$;
    return square;
}
```

Introduce Recursion

1.4.2 A(ii): Introduce recursion

```
public int square(int n) {
    int square;
    if (n == 0) {
        square = 0;
    } else {
        square = square(n-1) - square(n-1) + $n^2$;
    }
    return square;
}
```

Replace Method Call by Its Value

1.4.2 A(iii): Replace a method call by its value

```
public int square(int n) {
    int square;
    if (n == 0) {
        square = 0;
    }  else {
        square = square(n-1) - $(n-1)^2$ + $n^2$;
    }
    return square;
}
```

Expand

1.4.2 A(iv): Expand

```
public int square(int n) {
    int square;
    if (n = 0) {
        square = 0;
    }  else {
        square = square(n-1) - $n^2$ + $2n$ - 1 + $n^2$;
    }
    return square;
}
```

Simplify

1.4.2 A(v): Simplify

```
public int square(int n) {
   int square;
   if (n == 0) {
      square = 0;
   }  else {
      square = square(n-1) + 2n - 1;
   }
   return square;
}
```

Simplify

1.4.2 A(vi): Standardise

```
public int square(int n) {
    int square;
    if (n == 0) {
        square = 0;
    } else {
        square = square(n-1) + 2(n − 1) + 1;
    }
    return square;
}
```

2n

Eliminate $\mathbf{2(n-1)}$ with:

```
public int twoN(int n) {
    int twoN;
    if (n == 0) {
        twoN = 0;
    } else {
        twoN = twoN(n-1)+2;
    }
}
```

Return a Pair of Values

1.4.2 B: Return a Pair of Values

Rewrite the programme to return a pair of values:

```
public (int square,int twoN) square(int n) {
   (int,int) result;
   if (n == 0) {
      result = (0,0);
   }  else {
      result = (
         square(n-1)→ square + square(n-1)→ twoN + 1,
         square(n-1)→ twoN + 2);
   }
   return square;
}
```

Second Attempt

1.4.2 C: Eliminating the recursion

```
public (int square,int twoN) square(int n) {
    (int square,int twoN) result = (0,0);
    int i = 0;
    while (i != n) {
        result= (
            result→ square + result→ twoN + 1,
            result→ twoN + 2);
        i++;
    }
    return result;
}
```

Final Version

1.4.2 D: Final version

```
public int square(int n) {
    int i= 0, square = 0, twoN = 0;
    while (i != n) {
        square = square + twoN + 1;
        twoN = twoN + 2;
        i++;
    }
    return square;
}
```

Summary

2: Summary

2.1: The three tenets

- Prove a programme
  Can only prove that a programme is incorrect, not that it is correct.
- Prove a programme correct
  Construct the proof *after* the programme has been written. Strongly reliant on the programme being written in a style conducive to proof.
- Improve a correct programme
  The development of the programme *is* the proof of its correctness

Summary

2.2: W?W?W?

- Why do it?
- Would anybody do it?
- Will anybody do it?

For now, “prove” the programme