

1B1b Testing

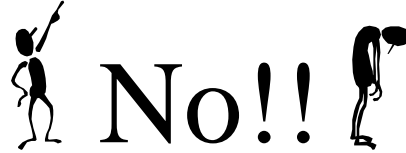
1 Copyright © 2004, Graham Roberts

Department of Computer Science



Perfection

- Do your programs work perfectly?
- Are you perfect?



2 Copyright © 2004, Graham Roberts

Department of Computer Science



Perfection, or Lack of It

- No program is perfect.
- Programs have errors.
- “On average program code has 10 errors per 1000 lines...”

3 Copyright © 2004, Graham Roberts

Department of Computer Science



Two V's

Verification

- “Are we building the system right?”
- Testing

Validation

- “Are we building the right system?”

4 Copyright © 2004, Graham Roberts

Department of Computer Science



Errors

- Anyone claiming to have a completely bug-free program is deliberately lying or is **seriously deluding themselves.**

And this is extremely unprofessional.

5 Copyright © 2004, Graham Roberts

Department of Computer Science



Testing

- Testing is really about trying to find bugs.
- Testing cannot show your program will *always* work properly — only the deluded believe this can be done!
- But it can remove sufficient bugs to make the program “good enough”.

6 Copyright © 2004, Graham Roberts

Department of Computer Science



Testing and Proof

- To prove something we must show:

$$\forall x \bullet P(x)$$

- This implies we have to explore *every* possible state of a program.

7

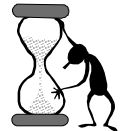
Copyright © 2004, Graham Roberts

Department of Computer Science



Testing and Proof (2)

- Take for example, the `sqrt` method.
- To “prove” it works we have to call it with every possible floating point value.
- So if $2^{64} = 18446744073709551616 \approx 10^{19}$ and we do 10^6 operations per second then this is 10^{13} seconds, which is 10^6 years.



8

Copyright © 2004, Graham Roberts

Department of Computer Science



Testing and Proof (3)

- Let us use some logic:

$$\forall x \bullet P(x) == \neg \exists x \bullet \neg P(x)$$

- For all x, such that P of x implies there does not exist an x, such that not P of x is true.
- This suggests we should look for a state that causes the program to fail.

9

Copyright © 2004, Graham Roberts

Department of Computer Science



Testing and Proof (4)

- The philosophy behind all testing should be the *finding of errors*.
 - Need to identify tests most likely to uncover errors.
- No “proof” can be constructed that no errors exist.
 - Just have the situation that no tests find errors.
 - The next test you add may find an error...

10

Copyright © 2004, Graham Roberts

Department of Computer Science



Questions?

11

Copyright © 2004, Graham Roberts

Department of Computer Science



Making testing `sqrt` manageable

- We still have the problem of 10^{19} possible values that could give us an error.
- Our proof obligation is weaker now though, we only need to find floating point values that cause an error.
- But how do you find them?

12

Copyright © 2004, Graham Roberts

Department of Computer Science



Testing the *sqrt* Method (1)

- We can manage the problem by studying the *domain* of the method.
- *sqrt* partitions the floating point numbers into 3 sets:

$$x < 0$$

$$x = 0$$

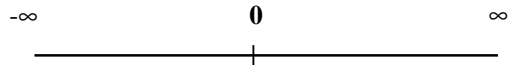
$$x > 0$$

13 Copyright © 2004, Graham Roberts

Department of Computer Science



Testing the *sqrt* Method (2)



14 Copyright © 2004, Graham Roberts

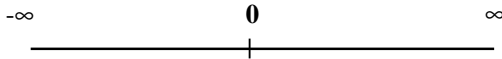
Department of Computer Science



Testing the *sqrt* Method (3)

$$\forall x \bullet x > 0 \wedge \text{sqrt}(x) \in \mathbb{R}$$

$$\forall x \bullet x = 0 \wedge \text{sqrt}(x) = 0$$



$$\forall x \bullet x < 0 \wedge \text{sqrt}(x) \in \mathbb{C}$$

15 Copyright © 2004, Graham Roberts

Department of Computer Science



Testing the *sqrt* Method (4)

- Select representatives from each of the sets to construct the test *data set*.
- Create a *test harness* — a program to call *sqrt* with the elements of the data set.
- Run the program and compare the results with what was expected (which you need to work out some other way!).

16 Copyright © 2004, Graham Roberts

Department of Computer Science



Testing the *sqrt* Method (5)

- Want to focus on boundary conditions:
 - 0.0, 1.0, 2.0, 3.0
 - MIN_DOUBLE, MAX_DOUBLE
 - .3, .33, .333, etc.
 - 0.0000000000001, 0.11111111111111, etc.
 - -0.0, -1.0

17 Copyright © 2004, Graham Roberts

Department of Computer Science



Testing the *sqrt* Method (5)

```
public testSqrt()
{
    System.out.println("sqrt(1.0) = " + Math.sqrt(1.0));
    System.out.println("sqrt(2.0) = " + Math.sqrt(2.0));
    System.out.println("sqrt(3.0) = " + Math.sqrt(3.0));
    System.out.println("sqrt(10.0) = " + Math.sqrt(10.0));
    System.out.println("sqrt(100.0) = " + Math.sqrt(100.0));
    System.out.println("sqrt(1000.0) = " + Math.sqrt(1000.0));
    System.out.println("sqrt(0.0) = " + Math.sqrt(0.0));
    System.out.println("sqrt(-1.0) = " + Math.sqrt(-1.0));
    // etc...
}
```

18 Copyright © 2004, Graham Roberts

Department of Computer Science



Testing the *sqrt* Method (6)

```
sqrt(1.0) = 1.0
sqrt(2.0) = 1.4142135623730951
sqrt(3.0) = 1.7320508075688772
sqrt(10.0) = 3.1622776601683795
sqrt(100.0) = 10.0
sqrt(1000.0) = 31.622776601683793
sqrt(0.0) = 0.0
sqrt(-1.0) = NaN ?
```

19 Copyright © 2004, Graham Roberts

Department of Computer Science



NaN?

- Not a Number.
- This version of `sqrt` will return NaN for any argument < 0 .

(Why? Can you write a method that returns a either a double or a Complex object...)

20 Copyright © 2004, Graham Roberts

Department of Computer Science



Using a different *sqrt* method

```
sqrt(1.0) = 1.0
sqrt(2.0) = 1.4142135623746899
sqrt(3.0) = 1.7320508100147274
sqrt(10.0) = 3.162277665175675
sqrt(100.0) = 10.000000000139897
sqrt(1000.0) = 31.622776601684336
sqrt(0.0) = NaN
sqrt(-1.0) = NaN
```

Different results!
Which are correct?

21 Copyright © 2004, Graham Roberts

Department of Computer Science



But

- This is quickly going to get boring and error prone.
 - Manual checking process.
 - OK for 10 tests,
 - tedious for 100 tests,
 - mind-numbing for 1000 tests.
- Need an automated approach.

22 Copyright © 2004, Graham Roberts

Department of Computer Science



Questions?

23 Copyright © 2004, Graham Roberts

Department of Computer Science



Testing in practice

- Test always and often.
 - Re-run *all* your tests *every* time you edit and compile *any* code.
- This implies that testing is a core activity of the programming process.
- And must be automated.

24 Copyright © 2004, Graham Roberts

Department of Computer Science



Test First

- Write your test code first.
- Then the program code you need to test.
 - If the test code is hard or impossible to write your program design is wrong.
- Use testing to find errors as early as possible.

25 Copyright © 2004, Graham Roberts

Department of Computer Science



Repeatable

- Tests must be *repeatable*.
- Test data should be the same each time a test is run.
- New tests should be added and existing tests retained.

26 Copyright © 2004, Graham Roberts

Department of Computer Science



Automated

- Testing should ideally be automated.
 - Your test code runs the tests *and* checks the results.
- Manual testing is error prone and boring.
 - It won't be done properly.

27 Copyright © 2004, Graham Roberts

Department of Computer Science



The Test Plan

- A *test plan* describes each test and the data set(s) used.
- The plan can be a written document but much more useful if it is embodied in your test code.
 - Separate written documentation wastes time to create and will never be properly maintained.

28 Copyright © 2004, Graham Roberts

Department of Computer Science



Describing a Test

- Purpose – what is being tested and why.
- Test data – data used for testing.
- Test procedure – how the test is carried out.
- Expected results – what you expect to happen.

29 Copyright © 2004, Graham Roberts

Department of Computer Science



Example test for `sqrt`

- Purpose – verify behaviour for positive real numbers.
- Test data – a data set of positive real numbers.
- Test procedure – use method in a test harness, call `sqrt` with the data values.
- Expected results – table of results matching data set, to compare with actual results.

[Repeat for 0 and negative.]

30 Copyright © 2004, Graham Roberts

Department of Computer Science



What data set?

- Can't test every value, so have to be selective.
- Sample of "normal" values (make sure the right answer is actually being given!).
- Also want to find the "edge" conditions.
 - Very large or small values.
 - Values with particular properties (e.g., infinitely recurring decimal places).
 - Values that might result in underflow or overflow while calculating.

31 Copyright © 2004, Graham Roberts

Department of Computer Science



Test Results

- Could keep an "audit trail": the *test log*.
 - Record of all the problems found and bugs fixed.
- But better to automate testing so that results are automatically checked.
- Always fix the problems found immediately before doing anything else.

32 Copyright © 2004, Graham Roberts

Department of Computer Science



Confidence in Results

- The extent of the test data set gives us a level of confidence in the program.
- We know the program has errors but we create a level of confidence of lack of errors in normal use.
- "Good enough" software.
- Keep adding tests and testing until the rate at which errors are found falls below an acceptable threshold.

33 Copyright © 2004, Graham Roberts

Department of Computer Science



Questions?

34 Copyright © 2004, Graham Roberts

Department of Computer Science



Testing Class Based Programs

- All classes must be tested, individually (unit testing) and in collaboration (functional testing).
- The program as a whole is also tested (acceptance testing).

35 Copyright © 2004, Graham Roberts

Department of Computer Science



Unit Testing a Java Class

- In Java, we could give every class a `main` method for testing — except, of course, for the class which holds the `main` that starts the application.
- *But* better to provide a separate test class for *each* class to be tested.

36 Copyright © 2004, Graham Roberts

Department of Computer Science



Testing a Class

- Create and initialise objects.
- Call *public* methods and check results.
 - Either those returned directly,
 - Or by calling another public method to check state of object.
- Private methods/variables are tested indirectly via public methods.
 - If you lack confidence that this is good enough, change your design.

37

Copyright © 2004, Graham Roberts

Department of Computer Science



Example – Person class

```
public class Person
{
    private String name;
    private String phoneNumber;
    public Person(String aName, String aNumber)
    {
        name = aName;
        phoneNumber = aNumber;
    }
    public String getName()
    {
        return name;
    }
    public String getNumber()
    {
        return phoneNumber;
    }
}
```

In this case we need to test that objects are properly initialised.

Note that instance objects are *immutable*.

38

Copyright © 2004, Graham Roberts

Department of Computer Science



Example (2)

```
public class TestPersonClass // Outline of a test class
{
    // Infrastructure to manage tests
    private Person p1, p2; // Person objects to be tested.
    private int count = 0, int fail = 0; // Record of progress
    public void setUp() {...} // Called before each test
    public void finish() {...} // Called after all tests run
    // Comparison method
    public void assertEquals(String s1, String s2, String method) {...}
    // Test methods
    public void testName() {...}
    public void testNumber() {...}
    // Main to run the tests
    public static void main(String[] args) {...}
}
```

39

Copyright © 2004, Graham Roberts

Department of Computer Science



Example (3)

```
public static void main(String[] args)
{
    TestPersonClass t = new TestPersonClass();
    t.setUp();
    t.testName();
    t.setUp(); // Must re-initialise before each test.
    t.testNumber();
    t.finish();
}
```

40

Copyright © 2004, Graham Roberts

Department of Computer Science



Example (4)

```
public void setUp() // Must be called before each test method
{
    count++;
    p1 = new Person("P1", "123");
    p2 = new Person("P2", "456");
}

public void finish() // Called after tests run
{
    System.out.println("\nTests run: " + count);
    System.out.println("Tests OK: " + (count - fail));
    System.out.println("Tests failed: " + fail);
}
```

41

Copyright © 2004, Graham Roberts

Department of Computer Science



Example (5)

```
public void testName()
{
    assertEquals(p1.getName(), "P1", "testName");
    assertEquals(p2.getName(), "P2", "testName");
}

public void testNumber()
{
    assertEquals(p1.getNumber(), "123", "testNumber");
    assertEquals(p2.getNumber(), "456", "testNumber");
    // deliberate fail, for example of what happens when a test fails
    assertEquals(p2.getNumber(), "010", "testNumber");
}
```

42

Copyright © 2004, Graham Roberts

Department of Computer Science



Example (6)

```
public void assertEquals(String s1, String s2, String method)
{
    if (!s1.equals(s2))
    {
        fail++;
        System.out.println("Test failed in method: " + method + " - ");
        System.out.println(s1 + " not equal to " + s2);
    }
    else
        System.out.print(".");
}
```

43 Copyright © 2004, Graham Roberts

Department of Computer Science



Example (7)

- Program output:

```
....
Test failed in method: testNumber - 456
    not equal to 010
Tests run: 2
Tests OK: 1
Tests failed: 1
```

44 Copyright © 2004, Graham Roberts

Department of Computer Science



Extending the Example

- For more complex classes:
 - Add more assert methods used to compare values.
 - Have more test objects.
 - Add test methods.

45 Copyright © 2004, Graham Roberts

Department of Computer Science



JUnit – www.junit.org

- JUnit is a very widely used *unit test* tool.
- Lightweight and straightforward to use.
 - You will be using it next year in 2b11.
- Based on same ideas seen in the example shown on the previous slides but rather better!
- Visit the web site and see what you make of it.
 - For a challenge, use JUnit to test your mini-project.

46 Copyright © 2004, Graham Roberts

Department of Computer Science



Summary

- Testing is used to find bugs and errors, so they can be fixed at the earliest opportunity.
- A proper test strategy is needed.
- Test early, often and always.
- Testing relies on establishing an acceptable degree of confidence, not on “proof”.
- Testing is essential!

47 Copyright © 2004, Graham Roberts

Department of Computer Science

