Testing For Late Bloomers

dotan@paracode.com, @jondot

You're late to the party

Very late

In Fact...

Its been 12 years

The movement of Earth's techtonic plates changed the party's location

The people attending the party married, had kids, divorced and got married again by the time you arrived.

You should feel bad.

Are you a professional?

Professionals give you a warranty

They can estimate accurately

They will not break your product while fixing it

Or introduce new problems into it

They take pride at the quality of their work

Professionals see quality as a competitive edge.

Its never too late

Unlearn Testing

A test is not code

You do not build a class hierarchy out of your tests

You do not DRY your tests (mostly)

You do not perform refactoring sessions on tests as if it were code

Unlearn Testing

A test is not a console application

It is also not a class (fixture) with 2 methods.

In which the test might be named `test1`, `test2`.

Unlearn Testing

There is no such thing as "test database"

Or special rows in dev database which no one should touch

There should not be tests that go bezerk on databases a-la `DELETE FROM mytable` before each run

"Unit Testing"

Not what you think

"Unit Testing" is both a concept and a method

Unit

Functional/Integation

Acceptance

Test in isolation

Any module that you build has dependencies

Cut them out

Plug them in

Plug fakes and mocks in testing to isolate

Unit Test

Test a single unit, single module

Translate roughly to a "class"

Functional/Integration

Wire up 2 modules, test

Wire up N modules, test

Wire up the dirty parts

Big-bang

Test use cases

Acceptance

When you have a costumer, you must run his tests successfuly

Tests defined by them

For yourself, test in end of iteration

Tests defined by PM

Your Development Lifecycle

Develop

Run tests

Fix failing (+CI)

Add tests

Release

RED

GREEN

REFACTOR

TDD

A money-making buzzword

TDD, seriously now

Drive your design through usage

Chop problems to small chunks

Only write what you need (YAGNI)

F-ck TDD

Let's call it reality-driven-design-with-tests.

RDDWT

Its all about doing it right

And the tools

xUnit 101

Fixture

Test case

Setup

Teardown

```
[TestFixture]
public WhenFlyingAPlan
{
  [SetUp]
  public void SetUp()
  {
    CreatePlane();
  }

  [TearDown]
  public void TearDown()
  {
    //really a bad smell
  }

  [Test]
  public void it_should_crash_alt_given_no_fuel()
  {
    ...
  }
```

Setup runs before **each** test

Order of tests when being run, should be nondeterministic

So kill tests with side effects

There is FixtureSetUp/TearDown but whatever...

Triangles

A triangle can decide if it is:

Equilateral

Isoceles

Scalene

Implementation?

```ruby
class TestTriangles
  def test_equilateral_triangles_have_equal_sides
    assert_equal :equilateral, triangle(2, 2, 2)
    assert_equal :equilateral, triangle(10, 10, 10)
  end

  def test_isosceles_triangles_have_exactly_two_sides_equa
    assert_equal :isosceles, triangle(3, 4, 4)
    assert_equal :isosceles, triangle(4, 3, 4)
    assert_equal :isosceles, triangle(4, 4, 3)
    assert_equal :isosceles, triangle(10, 10, 2)
  end

  def test_scalene_triangles_have_no_equal_sides
    assert_equal :scalene, triangle(3, 4, 5)
    assert_equal :scalene, triangle(10, 11, 12)
    assert_equal :scalene, triangle(5, 4, 2)
  end
end
```

Doing it right

Or how to write good tests

Here's a tip from electronics troubleshooting

IOCP

Input

Output

Control (software: logic)

Power (software: environment & exceptions)

Tooling

Mocks, Fakes, Stubs, Fakers

Test harnesses

Test runners

Coverage

Test Harnesses

You don't have to do it by hand

```
<test id="t1">
  <in>
    <event id="ev001" data="interface connect" />
    <event id="ev002" data="interface connect" />
  </in>
  <process>
    <param id="machine" value="m001"/>
  </process>
  <out>
    <result data="abc123"/>
  </out>
</test>
```

XML Based

Non-trained people (e.g. QA) get intellisense

And familiar tools

VERY easy to implement

Make schema, generate classes and you're done.

```
public void unit_should_fail_given_failures_on_board()
{
    var simulator = Factory.Build()
                    .WithMachine("m001")
                    .AttachToOrder("o001")
                      .Product("iphone")
                      .Board("
                                x x x x
                                P F F P
                                P P P P
                             ")
                      .Order("
                                00 01 09 10
                                02 03 08 07
                                04 05 06 11
                            ");
    simulator.ScanUnit("X23199FW");
    var result = simulator.Fail("bad component");
    Assert(result.[whatever] ..)
}
```

Fluent interface, custom test harness

"Draw" your data on your tests.

Have a test simulator that simulates a real client

A step further

Automatic test runners & watchers

Continuous integration

Continuous coverage

Smoke tests

Automation

BDD

Business value driven development

In your code, you test BV

Expressing BV

Typically via use case

As a User.. in order to..

WhenDoingX..It Should Z..Given Y

Tests Expressing BV

You test BV - outside--in

Force expressing in domain's wording

Ubiquitous language - future DDD session(?)

Natural to read, write -- therefore maintain

```
[TestFixture]
class WhenRegistering
{
  [Test]
  it_should_create_user_as_verified_given_preverify()
  {
    ..
    user.Verified.ShouldBeTrue();
  }
  ..
```

Opinionated Tooling

I <3 'easymock' type mock fwks.

Ruby: rspec, minitest, riot, rr (mocks), faker

C#: NUnit+(own BDD-fu), Moq (mocks), MvcTestHelpers, Machine.Fakes

Javascript: Jasmine (BDD+mocks)

VB: just kidding.

Questions?

Thanks!

dotan@paracode.com, @jondot