JUnit Pocket Guide

Kent Beck

Infastructure

Automate tedious parts of tests

Summarize

emotionally more confident

Don't feel rushed

Test-first - confident at all times

Win
  short term
  long term
  with teammates

Δ code later w/o breaking

↓ # of defects

No debugger

test = defect detector

A way to make fewer mistakes

Perfect not a goal, is a verb

Goals

Easy to write

" " learn

Quick

Not effect each other

Any combination

Ordinary Java language

Isolation over speed

Most tests work

Not a script (Too fine grained)

Fixtures

void setUp() { }

Objects/Data used in tests

tearDown() usually skipped

p20

JUnit API

```
subclass TestCase
    assertTrue(x);
or  assertEqual(x, y);
```

5 classes

* 1. Asert

2. Test

* 3. TestCase

4. TestSuite

5. Test Results

Assert

```
✓ values
  assertTrue(x)
  assertNull(x)
  assertSame(x, y)
  asserEquals(x, y)
    "      "   (x, y, Δz)
```

$\underbrace{\Delta z}_{\text{within error}}$

TestCase

Create subclass

method names: test_method

```
test_method(self) {
    Data1
    Data2
    call(Data2)
    assertEquals(D1, call(D2));
```

SetUp(self) for data used n*

Failure ≠ errors

assertion ✓   Bad java in test

p21

Test-First Programming

Separate interface
  from implementation

How does object look from
  the outside?

Add enough design to pass test

Stubs

What you want done in future

Debugging Tests

✓ can reproduce defect

Find object creating #

write test on object

✓ pass test when fixed

↓ chance of unfixing the fix

Learning an API

test show how methods used

Show results

Provide ways to use API

✓ assumptions in how package works

Update, ✓ no bad changes

Story of JUnit

Compiler 5x test code to source

Test framework for small talk

To learn java, ported framework

3hr plane ride

Test infected

p72

less time debugging

more time designing

~~testing~~ in background

Want simple test

indep. objects easier to test

1/3 - 1/2 writing tests

make concrete decisions

Leave something Broken

  Provides a starting place