UNSELFISH TESTING

```
JUnit version 4.11
.........
```

```
JUnit version 4.11
......................................................E.E..
```

```
JUnit version 4.11
.E.E..
There were 2 failures:
1) statement(CustomerTest)
org.junit.ComparisonFailure: expected:<...or John
  Godfather 4[ ]9.0
Amount owed is 9...> but was:<...or John
  Godfather 4[ ]9.0
Amount owed is 9...>
2) htmlStatement(CustomerTest)
org.junit.ComparisonFailure: expected:<...</h1>
<p>Godfather 4[ ]9.0</p>
<p>Amount ow...> but was:<...</h1>
<p>Godfather 4[ ]9.0</p>
<p>Amount ow...>

FAILURES!!!
Tests run: 4, Failures: 2
```

@thejayfields

```
JUnit version 4.11
.E.E..
There were 2 failures:
1) statem
org.junit.C                                        n
  Godfath
Amount c
  Godfath
Amount c
2) htmlSta
org.junit.C
<p>Godfa
<p>Amou
<p>Godfa
<p>Amou

FAILURES!!!
Tests run: 4, Failures: 2
```

HELLO
BoB

```
@Test
public void statement() {
  for (int i=0; i<customers.length; i++) {
    assertEquals(
      expStatement(
        "Rental record for %s\n" +
        "%sAmount owed is %s\n"  +
        "You earned %s frequent " +
        "renter points",
        customers[i],
        rentalInfo(
          "\t", "",
          customers[i].getRentals())),
      customers[i].statement());
  }
}
```

```
@Test
public void statement() {
  for (int i=0; i<customers.length; i++) {
    assertEquals(
      expStatement(
        "Rental record for %s\n" +
        "%sAmount owed is %s\n"  +
        "You earned %s frequent " +
        "renter points",
        customers[i],
        rentalInfo(
          "\t", "",
          customers[i].getRentals())),
      customers[i].statement());
  }
}
```

```
@Test
public void statement() {
  for (int i=0; i<customers.length; i++) {
    assertEquals(
      expStatement(
        "Rental record for %s\n" +
        "%sAmount owed is %s\n"  +
        "You earned %s frequent " +
        "renter points",
        customers[i],
        rentalInfo(
          "\t", "",
          customers[i].getRentals())),
      customers[i].statement());
  }
}
```

```
public class CustomerTest {
  Customer john, steve, pat, david;
  String johnName = "John",
    steveName = "Steve",
    patName = "Pat",
    davidName = "David";
  Customer[] customers;
```

```
public class CustomerTest {
  Customer john, steve, pat, david;
  String johnName = "John",
    steveName = "Steve",
    patName = "Pat",
    davidName = "David";
  Customer[] customers;
```

```java
@Before
public void setup() {
  david = ObjectMother
    .customerWithNoRentals(
      davidName);
  john = ObjectMother
    .customerWithOneNewRelease(
      johnName);
  pat = ObjectMother
    .customerWithOneOfEachRentalType(
      patName);
  steve = ObjectMother
    .customerWithOneNewReleaseAndOneRegular(
      steveName);
  customers =
    new Customer[]
    { david, john, steve, pat};
}
```

```
@Before
public void setup() {
  david = ObjectMother
    .customerWithNoRentals(
      davidName);
  john = ObjectMother
    .customerWithOneNewRelease(
      johnName);
  pat = ObjectMother
    .customerWithOneOfEachRentalType(
      patName);
  steve = ObjectMother
    .customerWithOneNewReleaseAndOneRegular(
      steveName);
  customers =
    new Customer[]
    { david, john, steve, pat};
}
```

```
@Test
public void statement() {
  for (int i=0; i<customers.length; i++) {
    assertEquals(
      expStatement(
        "Rental record for %s\n" +
        "%sAmount owed is %s\n"  +
        "You earned %s frequent " +
        "renter points",
        customers[i],
        rentalInfo(
          "\t", "",
          customers[i].getRentals())),
      customers[i].statement());
  }
}
```

```
@Test
public void statement() {
  for (int i=0; i<customers.length; i++) {
    assertEquals(
      expStatement(
        "Rental record for %s\n" +
        "%sAmount owed is %s\n"  +
        "You earned %s frequent " +
        "renter points",
        customers[i],
        rentalInfo(
          "\t", "",
          customers[i].getRentals())),
      customers[i].statement());
  }
}
```

```java
@Test
public void statement() {
  for (int i=0; i<customers.length; i++) {
    assertEquals(
      expStatement(
        "Rental record for %s\n" +
        "%sAmount owed is %s\n"  +
        "You earned %s frequent " +
        "renter points",
        customers[i],
        rentalInfo(
          "\t", "",
          customers[i].getRentals())),
      customers[i].statement());
  }
}
```

```
public static String rentalInfo(
  String startsWith,
  String endsWith,
  List<Rental> rentals) {
  String result = "";
  for (Rental rental : rentals)
    result += String.format(
      "%s%s\t%s%s\n",
      startsWith,
      rental.getMovie().getTitle(),
      rental.getCharge(),
      endsWith);
  return result;
}
```

```
public static String rentalInfo(
  String startsWith,
  String endsWith,
  List<Rental> rentals) {
  String result = "";
  for (Rental rental : rentals)
    result += String.format(
      "%s%s\t%s%s\n",
      startsWith,
      rental.getMovie().getTitle(),
      rental.getCharge(),
      endsWith);
  return result;
}
```

```java
public static String rentalInfo(
  String startsWith,
  String endsWith,
  List<Rental> rentals) {
  String result = "";
  for (Rental rental : rentals)
    result += String.format(
      "%s%s\t%s%s\n",
      startsWith,
      rental.getMovie().getTitle(),
      rental.getCharge(),
      endsWith);
  return result;
}
```

```
@Test
public void statement() {
  for (int i=0; i<customers.length; i++) {
    assertEquals(
      expStatement(
        "Rental record for %s\n" +
        "%sAmount owed is %s\n"  +
        "You earned %s frequent " +
        "renter points",
        customers[i],
        rentalInfo(
          "\t", "",
          customers[i].getRentals())),
      customers[i].statement());
  }
}
```

```java
@Test
public void statement() {
  for (int i=0; i<customers.length; i++) {
    assertEquals(
      expStatement(
        "Rental record for %s\n" +
        "%sAmount owed is %s\n"  +
        "You earned %s frequent " +
        "renter points",
        customers[i],
        rentalInfo(
          "\t", "",
          customers[i].getRentals())),
      customers[i].statement());
  }
}
```

```
@Test
public void statement() {
  for (int i=0; i<customers.length; i++) {
    assertEquals(
      expStatement(
        "Rental record for %s\n" +
        "%sAmount owed is %s\n"  +
        "You earned %s frequent " +
        "renter points",
        customers[i],
        rentalInfo(
          "\t", "",
          customers[i].getRentals())),
      customers[i].statement());
  }
}
```

```java
public static String expStatement(
  String formatStr,
  Customer customer,
  String rentalInfo) {
  return String.format(
    formatStr,
    customer.getName(),
    rentalInfo,
    customer.getTotalCharge(),
    customer.getTotalPoints());
}
```

WORKING EFFECTIVELY WITH UNIT TESTS

@thejayfields

```
; i++) {
+
})),
; i++) {
/em>" +
s" +
em></p>",
})),
);
```

```java
public static String rentalInfo(
  String startsWith,
  String endsWith,
  List<Rental> rentals) {
  String result = "";
  for (Rental rental : rentals)
    result += String.format(
      "%s%s\t%s%s\n",
      startsWith,
      rental.getMovie().getTitle(),
      rental.getCharge(),
      endsWith);
  return result;
}

public static String expStatement(
  String formatStr,
  Customer customer,
  String rentalInfo) {
  return String.format(
    formatStr,
    customer.getName(),
    rentalInfo,
    customer.getTotalCharge(),
    customer.getTotalPoints());
}
```

Tuesday, May 12, 15

@thejayfields

```java
public class CustomerTest {
  Customer john, steve, pat, david;
  String johnName = "John",
    steveName = "Steve",
    patName = "Pat",
    davidName = "David";
  Customer[] customers;

  @Before
  public void setup() {
    david = ObjectMother
      .customerWithNoRentals(
        davidName);
    john = ObjectMother
      .customerWithOneNewRelease(
        johnName);
    pat = ObjectMother
      .customerWithOneOfEachRentalType(
        patName);
    steve = ObjectMother
      .customerWithOneNewReleaseAndOneRegular(
        steveName);
    customers =
      new Customer[]
      { david, john, steve, pat};
  }
```

```java
@Test
public void statement() {
  for (int i=0; i<customers.length; i++) {
    assertEquals(
      expStatement(
        "Rental record for %s\n" +
        "%sAmount owed is %s\n"  +
        "You earned %s frequent " +
        "renter points",
        customers[i],
        rentalInfo(
          "\t", "",
          customers[i].getRentals())),
      customers[i].statement());
  }
}

@Test
public void htmlStatement() {
  for (int i=0; i<customers.length; i++) {
    assertEquals(
      expStatement(
        "<h1>Rental record for " +
        "<em>%s</em></h1>\n%s" +
        "<p>Amount owed is <em>%s</em>" +
        "</p>\n<p>You earned <em>%s" +
        " frequent renter points</em></p>",
        customers[i],
        rentalInfo(
          "<p>", "</p>",
          customers[i].getRentals())),
      customers[i].htmlStatement());
  }
}
```

```java
public static String rentalInfo(
  String startsWith,
  String endsWith,
  List<Rental> rentals) {
  String result = "";
  for (Rental rental : rentals)
    result += String.format(
      "%s%s\t%s%s\n",
      startsWith,
      rental.getMovie().getTitle(),
      rental.getCharge(),
      endsWith);
  return result;
}

public static String expStatement(
  String formatStr,
  Customer customer,
  String rentalInfo) {
  return String.format(
    formatStr,
    customer.getName(),
    rentalInfo,
    customer.getTotalCharge(),
    customer.getTotalPoints());
}
```

1. Statement()

2. Customer[]

3. Customer Assignment

4. Digest Setup

5. OBJECT MOTHER

6. OBJECT MOTHER CUSTOMER VARIATIONS

7. Expected String Building

8. RENTALINFO()

9. RENTALINFO IMPLEMTATION

10. Find and digest expStatement()

ANGER
WORK

why?

we write tests we don’t need with time we don’t have to satisfy people we don’t like.

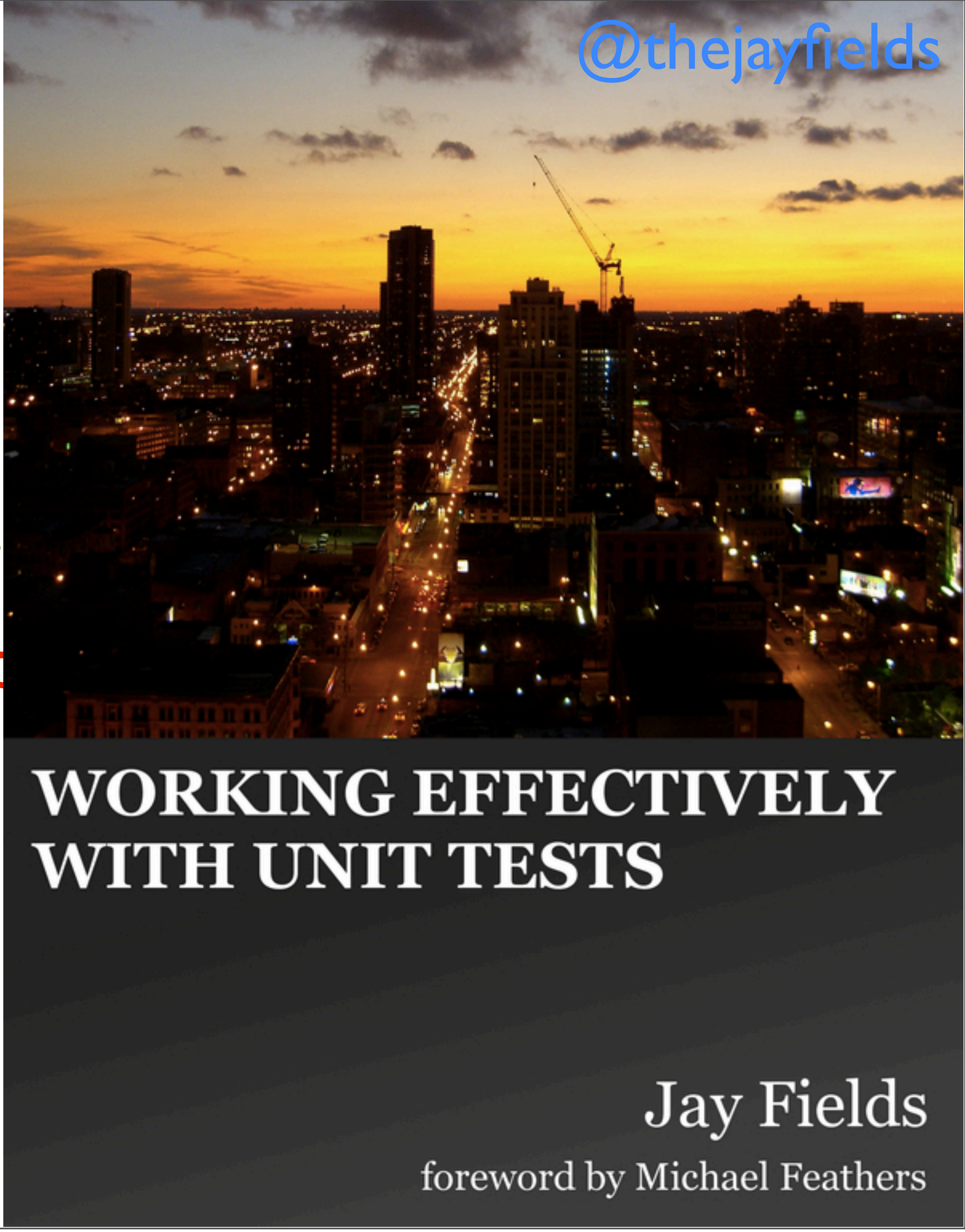
motivators?

motivators?
enable refactoring, immediate feedback, and breaking a problem up into smaller pieces

motivators?
enable refactoring, immediate feedback, and breaking a problem up into smaller pieces

motivators?
enable refactoring, immediate feedback,
and breaking a problem up into smaller
pieces

motivators?
enable refactori
and breaking a p
pieces



WORKING EFFECTIVELY WITH UNIT TESTS

Jay Fields

foreword by Michael Feathers

motivators?
enable refactoring, immediate feedback,
and breaking a problem up into smaller
pieces

-- @delitescere

To create a tiny universe where the software exists to do one thing and do it well.
-- @delitescere

DRY

Suite / Fixture / Test

Suite / ~~Fixture~~ / Test

TESTING UNRELATED CUSTOMERS

CREATING UNRELATED CUSTOMERS

OVEREXTRACTING

RELEVANT
DATA
ONLY

DRY SUITES ,
DRY LOCALS,
ELSE DAMP

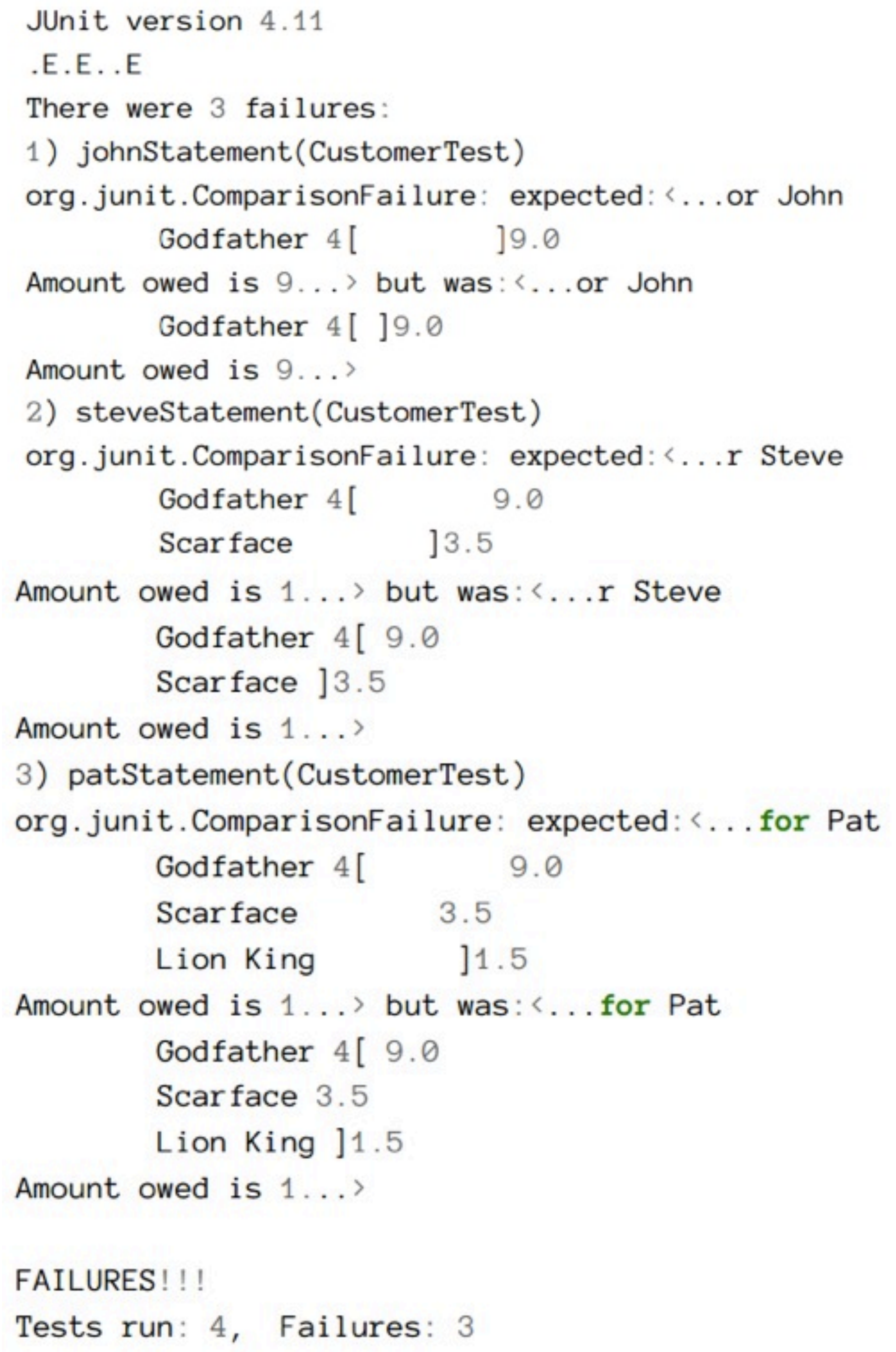
one: replace loop with individual tests

```java
@Test
public void davidStatement() {
  assertEquals(
    expStatement(
      "Rental record for %s\n%sAmount " +
      "owed is %s\nYou earned %s " +
      "frequent renter points",
      david,
      rentalInfo(
        "\t", "", david.getRentals())),
    david.statement());
```

```java
@Test
public void johnStatement() {
  assertEquals(
    expStatement(
      "Rental record for %s\n%sAmount " +
      "owed is %s\nYou earned %s " +
      "frequent renter points",
      john,
      rentalInfo(
        "\t", "", john.getRentals())),
    john.statement());
```

```java
@Test
public void patStatement() {
  assertEquals(
    expStatement(
      "Rental record for %s\n%sAmount " +
      "owed is %s\nYou earned %s " +
      "frequent renter points",
      pat,
      rentalInfo(
        "\t", "", pat.getRentals())),
    pat.statement());
```

```java
@Test
public void steveStatement() {
  assertEquals(
    expStatement(
      "Rental record for %s\n%s" +
      "Amount owed is %s\nYou earned %s " +
      "frequent renter points",
      steve,
      rentalInfo(
        "\t", "", steve.getRentals())),
    steve.statement());
```



```
JUnit version 4.11
.E.E..E
There were 3 failures:
1) johnStatement(CustomerTest)
org.junit.ComparisonFailure: expected:<...or John
        Godfather 4[        ]9.0
Amount owed is 9...> but was:<...or John
        Godfather 4[ ]9.0
Amount owed is 9...>
2) steveStatement(CustomerTest)
org.junit.ComparisonFailure: expected:<...r Steve
        Godfather 4[        9.0
        Scarface        ]3.5
Amount owed is 1...> but was:<...r Steve
        Godfather 4[ 9.0
        Scarface ]3.5
Amount owed is 1...>
3) patStatement(CustomerTest)
org.junit.ComparisonFailure: expected:<...for Pat
        Godfather 4[        9.0
        Scarface        3.5
        Lion King        ]1.5
Amount owed is 1...> but was:<...for Pat
        Godfather 4[ 9.0
        Scarface 3.5
        Lion King ]1.5
Amount owed is 1...>

FAILURES!!!
Tests run: 4,  Failures: 3
```

It's only after we've duplicated everything that we're free to dry anything

two: expect literals

@thejayfields

```java
@Test
public void patStatement() {
  assertEquals(
    expStatement(
      "Rental record for %s\n%sAmount " +
      "owed is %s\nYou earned %s " +
      "frequent renter points",
      pat,
      rentalInfo(
        "\t", "", pat.getRentals())),
    pat.statement());
```

```java
@Test
public void johnStatement() {
  assertEquals(
    expStatement(
      "Rental record for %s\n%sAmount " +
      "owed is %s\nYou earned %s " +
      "frequent renter points",
      john,
      rentalInfo(
        "\t", "", john.getRentals())),
    john.statement());
```

```java
@Test
public void steveStatement() {
  assertEquals(
    expStatement(
      "Rental record for %s\n%s" +
      "Amount owed is %s\nYou earned %s " +
      "frequent renter points",
      steve,
      rentalInfo(
        "\t", "", steve.getRentals())),
    steve.statement());
```

```
@Test
public void patStatement() {
  assertEquals(
    "Rental record for Pat\n\t" +
    "Godfather 4\t9.0\n" +
    "\tScarface\t3.5\n" +
    "\tLion King\t1.5\n" +
    "Amount owed is 14.0\n" +
    "You earned 4 frequent renter points",
    pat.statement());
}
```

```
@Test
public void johnStatement() {
  assertEquals(
    "Rental record for John\n\t" +
    "Godfather 4\t9.0\n" +
    "Amount owed is 9.0\n" +
    "You earned 2 frequent renter points",
    john.statement());
}
```

```
@Test
public void steveStatement() {
  assertEquals(
    "Rental record for Steve\n\t" +
    "Godfather 4\t9.0\n" +
    "\tScarface\t3.5\n" +
    "Amount owed is 12.5\n" +
    "You earned 3 frequent renter points",
    steve.statement());
}
```

three: inline setup

@thejayfields

```java
@Test
public void patStatement() {
  assertEquals(
    "Rental record for Pat\n\t" +
    "Godfather 4\t9.0\n" +
    "\tScarface\t3.5\n" +
    "\tLion King\t1.5\n" +
    "Amount owed is 14.0\n" +
    "You earned 4 frequent renter points",
    pat.statement());
}
```

```java
@Test
public void johnStatement() {
  assertEquals(
    "Rental record for John\n\t" +
    "Godfather 4\t9.0\n" +
    "Amount owed is 9.0\n" +
    "You earned 2 frequent renter points",
    john.statement());
}
```

```java
@Test
public void steveStatement() {
  assertEquals(
    "Rental record for Steve\n\t" +
    "Godfather 4\t9.0\n" +
    "\tScarface\t3.5\n" +
    "Amount owed is 12.5\n" +
    "You earned 3 frequent renter points",
    steve.statement());
}
```

@thejayfields

```java
@Test
public void noRentalsStatement() {
  assertEquals(
    "Rental record for David\nAmount " +
    "owed is 0.0\n" +
    "You earned 0 frequent renter points",
    ObjectMother
    .customerWithNoRentals(
      "David").statement());
}

@Test
public void
newReleaseAndRegularStatement() {
  assertEquals(
    "Rental record for Steve\n\t" +
    "Godfather 4 9.0\n" +
    "\tScarface 3.5\n" +
    "Amount owed is 12.5\n" +
    "You earned 3 frequent renter points",
    ObjectMother
    .customerWithOneNewReleaseAndOneRegular(
      "Steve").statement());
}
```

```java
@Test
public void oneNewReleaseStatement() {
  assertEquals(
    "Rental record for John\n\t" +
    "Godfather 4 9.0\n" +
    "Amount owed is 9.0\n" +
    "You earned 2 frequent renter points",
    ObjectMother
    .customerWithOneNewRelease(
      "John").statement());
}

@Test
public void allRentalTypesStatement() {
  assertEquals(
    "Rental record for Pat\n\t" +
    "Godfather 4 9.0\n" +
    "\tScarface 3.5\n\tLion King 1.5\n" +
    "Amount owed is 14.0\n" +
    "You earned 4 frequent renter points",
    ObjectMother
    .customerWithOneOfEachRentalType(
      "Pat").statement());
}
```

Motivators

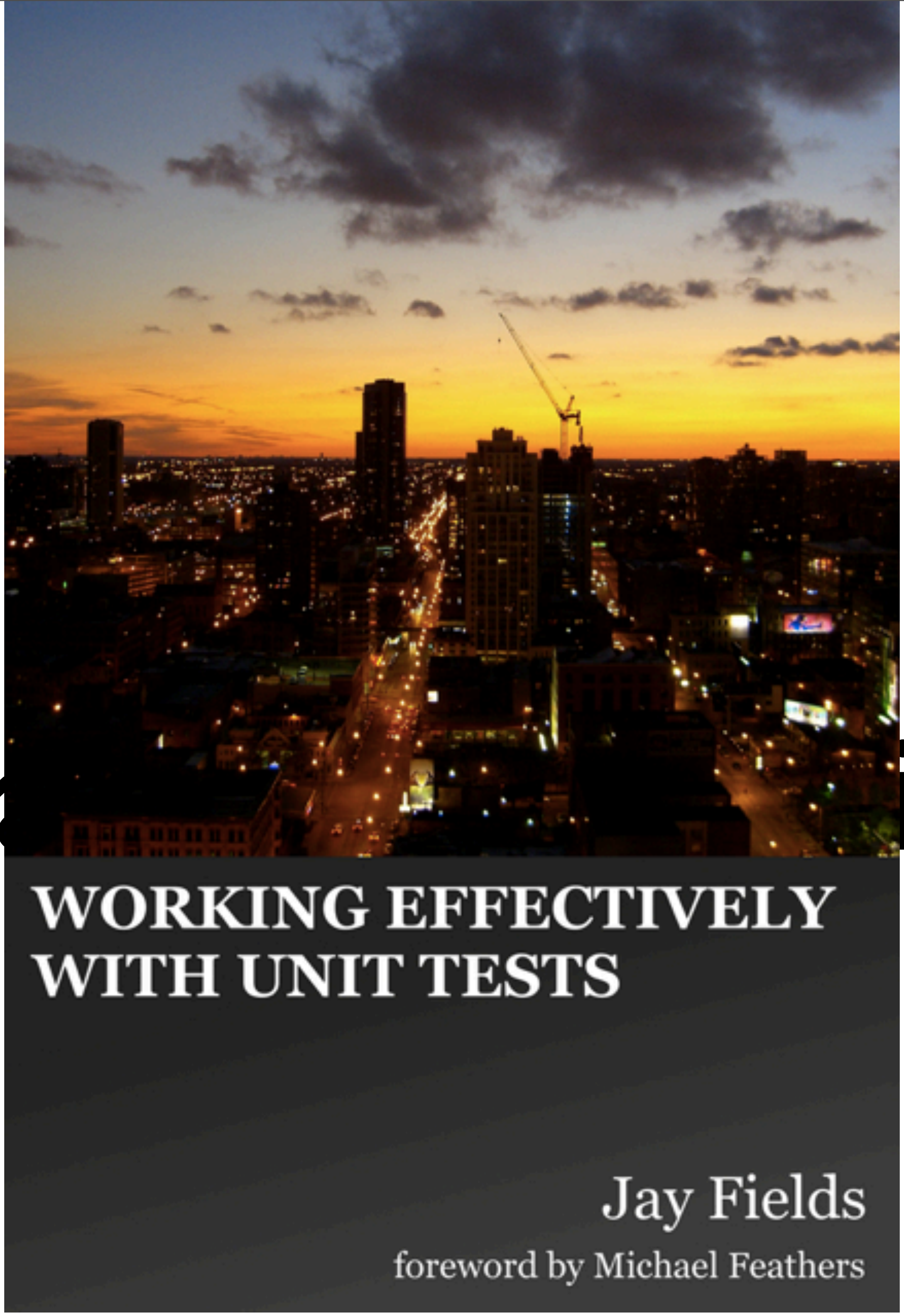
validate the system

feedback

prevent regression

code coverage

en ing



WORKING EFFECTIVELY WITH UNIT TESTS

Jay Fields

foreword by Michael Feathers

enable refactoring

document the system

manager told you to

tdd

break up a problem

improved design

customer acceptance

Any fool can write a test that helps them today. Good programmers write tests that help the entire team in the future.

what can we do?

This is your career and it's ending one test suite run at a time

def: Solitary Unit Test

def: Solitary Unit Test

1. never cross boundaries
2. the Class Under Test should be the only concrete class found in a test

def: Sociable Unit Test

1. any unit test that isn't solitary

+ SOCIABLE

+ SOCIABLE
- SOLITARY

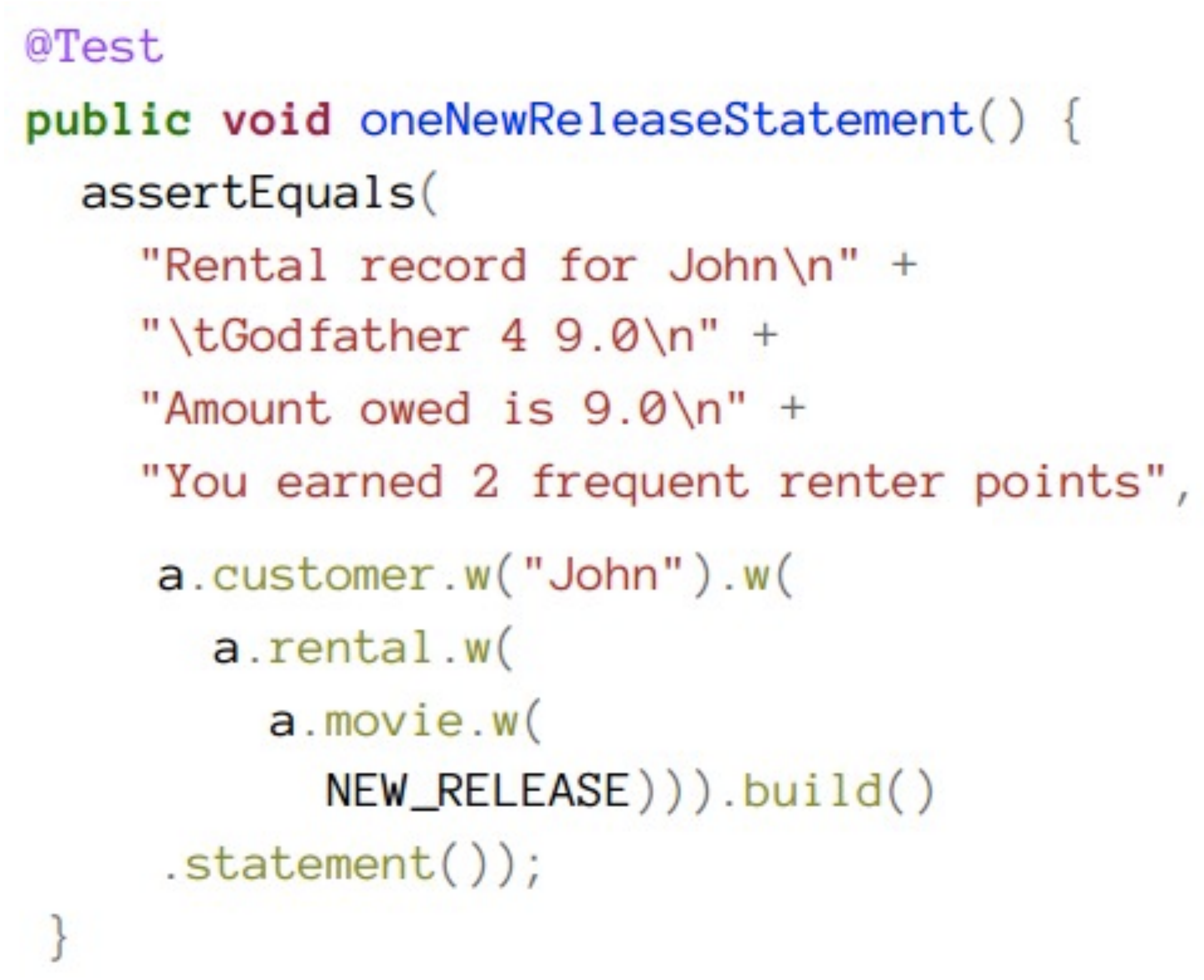
+ SOCIABLE

+ SOLITARY

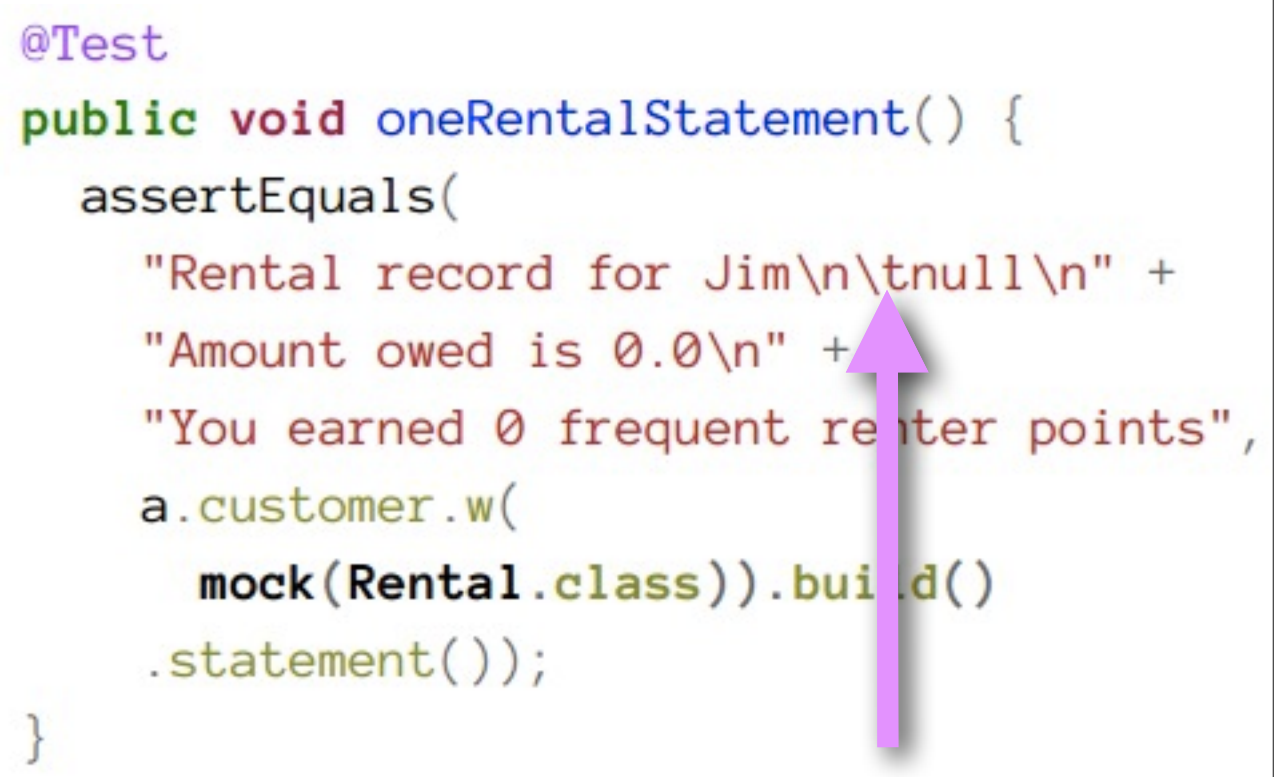
```
@Test
public void oneNewReleaseStatement() {
  assertEquals(
    "Rental record for John\n" +
    "\tGodfather 4 9.0\n" +
    "Amount owed is 9.0\n" +
    "You earned 2 frequent renter points",
     a.customer.w("John").w(
       a.rental.w(
         a.movie.w(
           NEW_RELEASE))).build()
     .statement());
 }
```

```
@Test
public void oneNewReleaseStatement() {
  assertEquals(
    "Rental record for John\n" +
    "\tGodfather 4 9.0\n" +
    "Amount owed is 9.0\n" +
    "You earned 2 frequent renter points",
     a.customer.w("John").w(
       a.rental.w(
         a.movie.w(
           NEW_RELEASE))).build()
     .statement());
 }
```

```
@Test
public void oneRentalStatement() {
  assertEquals(
    "Rental record for Jim\n\tnull\n" +
    "Amount owed is 0.0\n" +
    "You earned 0 frequent renter points",
    a.customer.w(
      mock(Rental.class)).build()
    .statement());
}
```



```
@Test
public void oneNewReleaseStatement() {
  assertEquals(
    "Rental record for John\n" +
    "\tGodfather 4 9.0\n" +
    "Amount owed is 9.0\n" +
    "You earned 2 frequent renter points",

     a.customer.w("John").w(
       a.rental.w(
         a.movie.w(
           NEW_RELEASE))).build()
     .statement());
}
```



```
@Test
public void oneRentalStatement() {
  assertEquals(
    "Rental record for Jim\n\tnull\n" +
    "Amount owed is 0.0\n" +
    "You earned 0 frequent renter points",
    a.customer.w(
      mock(Rental.class)).build()
    .statement());
}
```

```java
@Test
public void threeRentalsCharge() {
  Rental rental = mock(Rental.class);
  when(rental.getCharge()).thenReturn(2.0);
  assertEquals(
    6.0,
    a.customer.w(
      rental,
      rental,
      rental).build().getTotalCharge(),
    0);
}
```

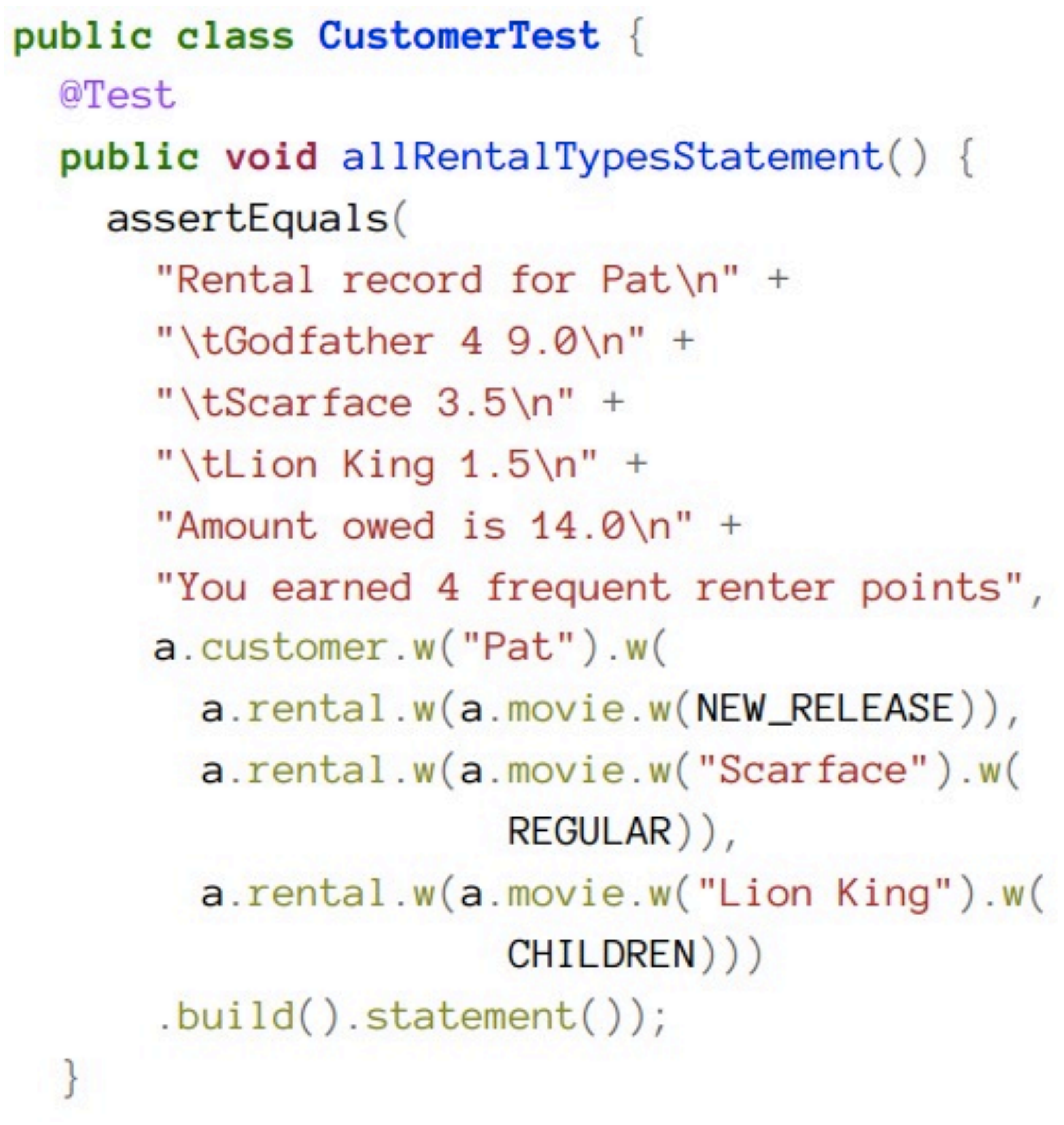
```java
@Test
public void twoRentalsPoints() {
  Rental rental = mock(Rental.class);
  when(rental.getPoints()).thenReturn(2);
  assertEquals(
    4,
    a.customer.w(
      rental,
      rental).build().getTotalPoints());
}
```

```
public class MovieTest {
  @Test
  public void getChargeForChildrens() {
    assertEquals(
      1.5,
      a.movie.w(
        CHILDREN).build().getCharge(1),
      0);
    assertEquals(
      1.5,
      a.movie.w(
        CHILDREN).build().getCharge(2),
      0);
    assertEquals(
      1.5,
      a.movie.w(
        CHILDREN).build().getCharge(3),
      0);
    assertEquals(
      3.0,
      a.movie.w(
        CHILDREN).build().getCharge(4),
      0);
```

CONFIDENCE

ISOLATION

INTEGRATION



```java
public class CustomerTest {
  @Test
  public void allRentalTypesStatement() {
    assertEquals(
      "Rental record for Pat\n" +
      "\tGodfather 4 9.0\n" +
      "\tScarface 3.5\n" +
      "\tLion King 1.5\n" +
      "Amount owed is 14.0\n" +
      "You earned 4 frequent renter points",
      a.customer.w("Pat").w(
        a.rental.w(a.movie.w(NEW_RELEASE)),
        a.rental.w(a.movie.w("Scarface").w(
                             REGULAR)),
        a.rental.w(a.movie.w("Lion King").w(
                             CHILDREN)))
      .build().statement());
  }
```

how else can I improve the test suite?

one assertion per test

```
public class MovieTest {
  @Test
  public void getChargeForChildrens() {
    assertEquals(
      1.5,
      a.movie.w(
        CHILDREN).build().getCharge(1),
      0);
    assertEquals(
      1.5,
      a.movie.w(
        CHILDREN).build().getCharge(2),
      0);
    assertEquals(
      1.5,
      a.movie.w(
        CHILDREN).build().getCharge(3),
      0);
    assertEquals(
      3.0,
      a.movie.w(
        CHILDREN).build().getCharge(4),
      0);
JUnit version 4.11
.E
There was 1 failure:
1) getChargeForChildrens(solitary.MovieTest)
java.lang.AssertionError: expected:<1.5> but was:<3.0>

FAILURES!!!
Tests run: 1,  Failures: 1
```

SELF DESTRUCTION

OUR TESTS ARE
FAILING US

POOR SOLUTION
OR
POOR SOLUTION

OR?

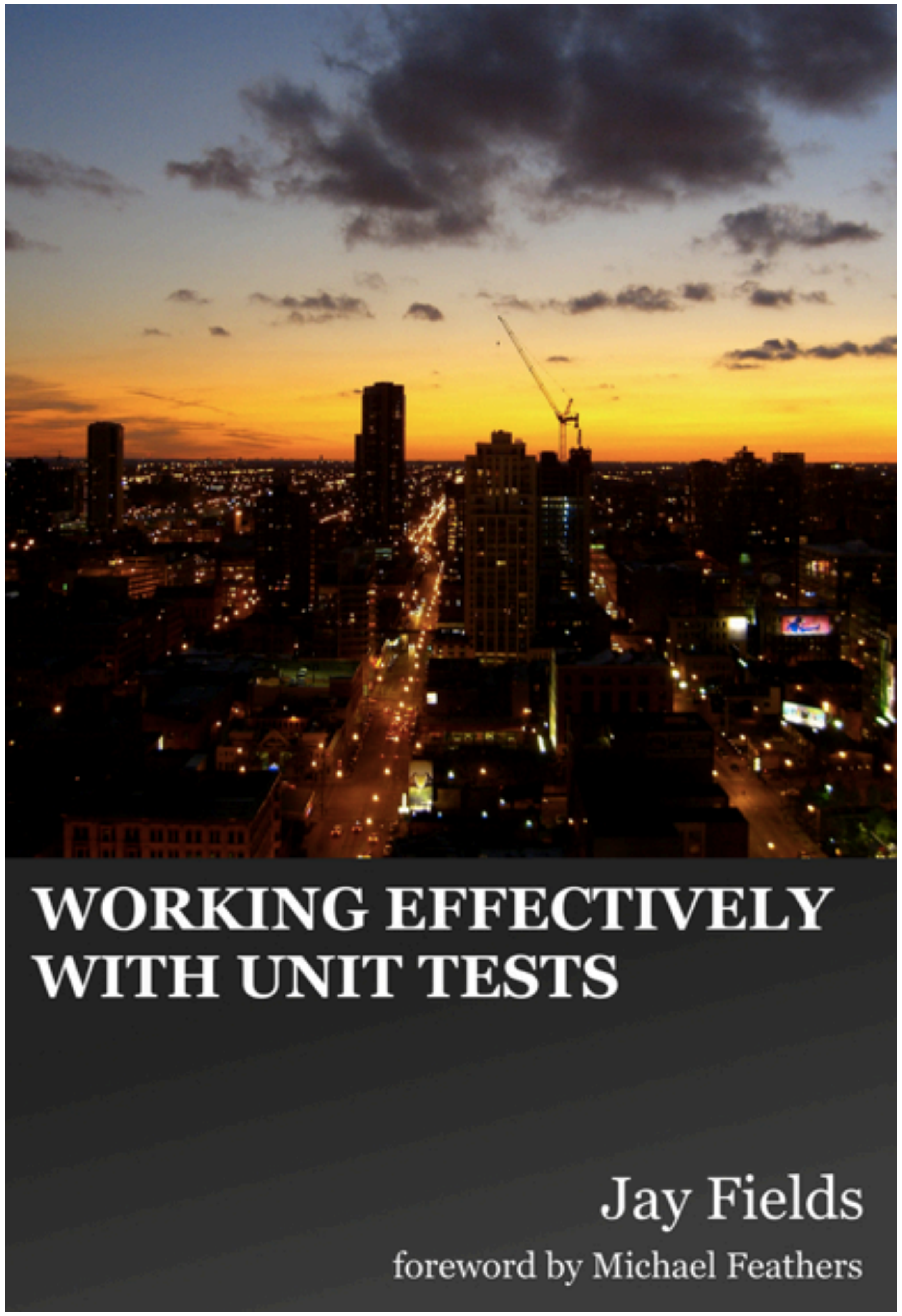
```
public class MovieTest {
  @Test
  public void getChargeForChildrens1Day() {
    assertEquals(
      1.5,
      a.movie.w(
        CHILDREN).build().getCharge(1),
      0);
  }

  @Test
  public void getChargeForChildrens2Day() {
    assertEquals(
      1.5,
      a.movie.w(
        CHILDREN).build().getCharge(2),
      0);
  }
JUnit version 4.11
...E.E.E
There were 3 failures:
1) getChargeForChildrens3Day(solitary.MovieTest)
java.lang.AssertionError: expected:<1.5> but was:<3.0>
2) getChargeForChildrens4Day(solitary.MovieTest)
java.lang.AssertionError: expected:<3.0> but was:<4.5>
3) getChargeForChildrens5Day(solitary.MovieTest)
java.lang.AssertionError: expected:<4.5> but was:<6.0>

FAILURES!!!
Tests run: 5,  Failures: 3
```

ONE ASSERTION
PER TEST

wrapping up

The tests you own end up owning you.

Your tests are not special. They are not beautiful or unique snowflakes. They're the same decaying text as every other test.



WORKING EFFECTIVELY WITH UNIT TESTS

Jay Fields

foreword by Michael Feathers

more:
http://wewut.com

Thank You.
Questions?