Unit Testing
continued

ERIN KEITH

Goals

1. Retrospective
2. Best Practices
3. Types of Coverage

Retrospective

What do you think is hard about testing?

1. Where did you get stuck?
2. Did you have to make changes to the project code to be able to test it?
   - Consider what tests should **NOT** involve

Do *not* test

A testing unit should focus on one tiny bit of functionality and “prove it correct”.

- Unit tests are simple and localized
- They should **NOT** involve
  - multi-threading
  - I/O
  - database connections
  - web services
  - these would be considered integration tests
- They should be fast
- They run in isolation
- Order shouldn’t matter
- They should not depend on global state

Code Coverage

Test the individual units of software.

- **Code Coverage**
  - percentage of total code covered by testing
- **Function Coverage**
  - there should be at least one test calling each function
- **Statement Coverage**
  - "ideally" each statement should be executed
- **Edge Coverage**
  - As many edge cases as is reasonable should be included
- **Branch Coverage**
  - each control structure path should be explored
- **Condition Coverage**
  - boolean sub expressions

Example 1

```
def return_sum_type(a, b):
    result = a + b

    if result > 0:
        return "positive"

    else:
        return "negative"
```

What kind of coverage should we apply here?

Example 1

```
def return_sum_type(a, b):
    result = a + b

    if result > 0:
         return "positive"

    else:
         return "negative"
```

- Function Coverage
- Statement Coverage
- Edge Coverage
- Branch Coverage

Example 1

```
class TestReturnSumType(unittest.TestCase):

  def test_positive(self):
    assertEqual(return_sum_type(1, 2), "positive")

  def test_negative(self):
    assertEqual(return_sum_type(-1, -2), "negative")

  def test_negative_zeros(self):
    assertEqual(return_sum_type(0, 0), "negative")
```

Example 2

```
def is_palindrome(str):
    for i in xrange(0, len(str)/2):
        if str[i] != str[len(str)-i-1]:
            return False
    return True
```

What kind of coverage should we apply here?

Example 2

```
def is_palindrome(str):
    for i in xrange(0, len(str)/2):
        if str[i] != str[len(str)-i-1]:
            return False
    return True
```

- Function Coverage
- Statement Coverage
- Edge Coverage
- Branch Coverage

Example 2

```
class TestIsPalindrome(unittest.TestCase):

    def test_palindrome(self):
        assertTrue(is_palindrome("tacocat"))

    def test_not_palindrome(self):
        assertFalse(is_palindrome("hotdog"))

    def test_empty_string(self):
        assertTrue(is_palindrome(""))
```

Example 3

```
if (A and B) or (B and C):
    return True
```

What kind of coverage should we apply here?

Example 3

```
if (A and B) or (B and C):
    return True
```

- Edge Coverage
- Condition Coverage

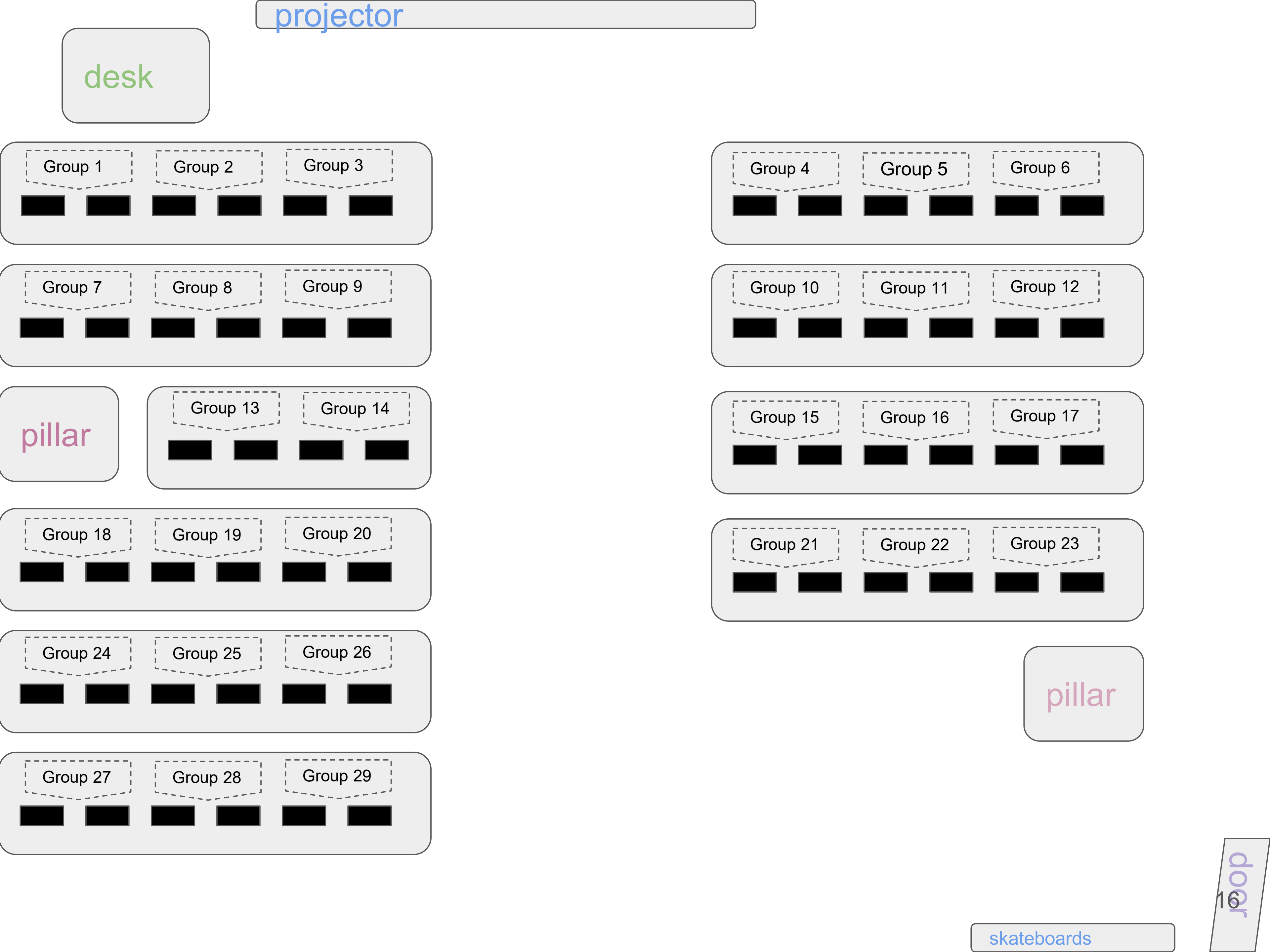
Example 3

```
class TestABandC(unittest.TestCase):
    def test_all_true(self):
      assertTrue(a_b_and_c(1, 1, 1))

    def test_a_b_true(self):
      assertTrue(a_b_and_c(1, 1, 0))

    def test_b_c_true(self):
      assertTrue(a_b_and_c(0, 1, 1))

    def test_a_c_true(self):
      assertFalse(a_b_and_c(0, 1, 0))

    def test_all_false(self):
      assertFalse(a_b_and_c(0, 0, 0))
```

Calculating Code Coverage

- How much of your code base is tested.
- Useful tool for finding untested parts of a codebase
- Not useful as a numeric statement of how good your tests are
- Sweet spot: around 70-85% coverage
- Many tools are available to analyze code coverage

- Consider “test coverage” instead as **a measurement of the degree to which a test or testing suite actually checks the full extent of a program’s functionality**.


projector
desk
Group 1
Group 2
Group 3
Group 4
Group 5
Group 6
Group 7
Group 8
Group 9
Group 10
Group 11
Group 12
pillar
Group 13
Group 14
Group 15
Group 16
Group 17
Group 18
Group 19
Group 20
Group 21
Group 22
Group 23
Group 24
Group 25
Group 26
pillar
Group 27
Group 28
Group 29
skateboards
door