W4156

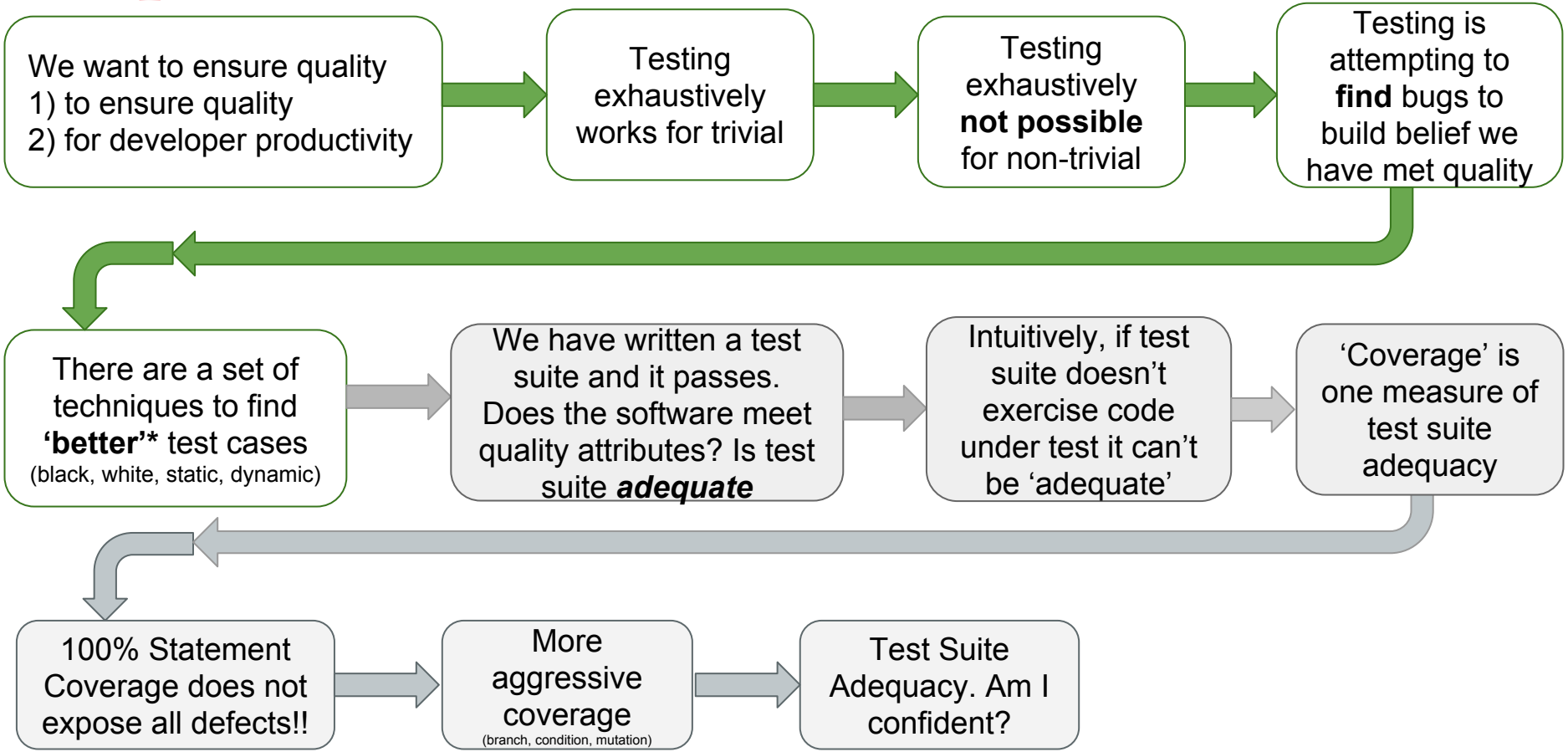
Testing II

Agenda

We are going to cover *theory* in parallel to *practice/code*

- ❑ Test Suite Effectiveness
- ❑ Coverage
  - ❑ Statement
  - ❑ Branch
  - ❑ Condition
- ❑ Testing
- ❑ Testing as Productivity

Recap



* better=higher probability of finding defects

Assessing Test Suite Effectiveness

1. Established we can not exhaustively test
2. We have defined a test suite
3. That test suite passed
4. Should we be confident?

Test Suite Effectiveness: How effective is my test *suite* at finding faults?

The more effective our test suite the more confidence/quality we have assured

Technique III: Coverage

Question 1: Recall our test cases *exercise* our code under test

- What does it tell us if our test cases only exercised 20% of our code under test?
- (Would you fly on a plane where the test cases only exercised 20% of the code?)
  (Congrats! You now understand the basics of white box testing)

Question 2: If you knew how the code worked could you generate new test case?

Assessing Test Suite Effectiveness: Coverage

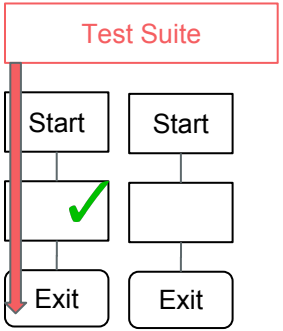
Coverage is the most common measure of test suite effectiveness in industry

Coverage measures how ‘much’ of the code under test is exercised by the test suite

Coverage is a ‘white box’ technique as we rely on knowledge of code internals

We can use control flow graphs to understand code coverage

Coverage Measure I: Statement



- **Statement coverage:** asks whether every *statement* in the code under test has been exercised by the test suite
  - Note: Within a codebase this is all classes/methods
  - As we build up theory we are discussing only methods

- Intuitively, this make sense. How can we hope to have proven that the code works if we have not even run it as part of the test suite?

Statement Coverage

```python
class Nationality:
    American = auto()
    British = auto()


class LegalToDrinkCalculatorWithTwoBugs:
    """..."""

    @staticmethod
    def is_legal(age: int, nationality: Nationality) -> bool:
        """..."""
        legal = True
        if (Nationality.American == nationality and age >= 21) or (Nationality.British and age >= 16):
            legal = True
        return legal
```

```python
class Test100StatementCoverageTwoBugs(unittest.TestCase):

    def test_legal_drinking(self):
        """..."""
        self.assertTrue(LegalToDrinkCalculatorWithTwoBugs.is_legal(21, Nationality.American))
```

Statement Coverage

Coverage.py keeps track of which statements are executed during test suite (green shows stmt executed)

```
9   class LegalToDrinkCalculatorWithTwoBugs:
10      """..."""
14
15      @staticmethod
16      def is_legal(age: int, nationality: Nationality) -> bool:
17          """..."""
24          legal = True
25          if (Nationality.American == nationality and age >= 21) or (Nationality.British and age >= 16):
26              legal = True
27          return legal
```

100% Statement Coverage != Correct

- We created a simple program.
- Wrote a single test case
  - It passed
  - We achieved 100% statement coverage
  - (I wrote it in a way that even with an 'if' statement we did cover 100% of lines)

BUT there are ***STILL TWO*** bugs in the program which the unit test ***DID NOT EXPOSE***

```
class LegalToDrinkCalculatorWithTwoBugs:
    """..."""

    @staticmethod
    def is_legal(age: int, nationality: Nationality) -> bool:
        """..."""
        legal = True
        if (Nationality.American == nationality and age >= 21) or (Nationality.British and age >= 16):
            legal = True
        return legal
```

Statement Coverage Limitation

Remember: Before we added this test case we had achieved 100% statement coverage

Does this test case ‘pass’ (in the sense is_legal returns **False** and assertFalse(**False**) passes)?

```
def test_should_be_illegal_drinking(self):
    """..."""
    self.assertFalse(LegalToDrinkCalculatorWithTwoBugs.is_legal(8, Nationality.American))
```

```
class LegalToDrinkCalculatorWithTwoBugs:
    """..."""

    @staticmethod
    def is_legal(age: int, nationality: Nationality) -> bool:
        """..."""
        legal = True
        if (Nationality.American == nationality and age >= 21) or (Nationality.British and age >= 16):
            legal = True
        return legal
```

Statement Coverage Limitation

We had previously ***achieved*** 100% coverage (and it passed)
And yet! There was an unexposed bug.
This test case exposes that bug.

```
def test_should_be_illegal_drinking(self):
    """ ... """
    self.assertFalse(LegalToDrinkCalculatorWithTwoBugs.is_legal(8, Nationality.American))
```

Run Nosetests in Test100StatementCoverageTwoBugs

2 tests done: 1 failed – 0ms

Test Results 0ms
tests.testing.theory.test_leg 0ms
test_should_be_illegal_dr 0ms

```
Traceback (most recent call last):
  File "/Users/ewanlowe/Documents/developer/w4156-lecture-code/tests/testing/theory/test_legaldrinking.py",
    self.assertFalse(LegalToDrinkCalculatorWithTwoBugs.is_legal(8, Nationality.American))
AssertionError: True is not false

----------------------------------------------------------------------
Ran 2 tests in 0.002s

FAILED (failures=1)
```

It Failed (our code has a bug)

```python
class Test100StatementCoverageTwoBugs(unittest.TestCase):

    def test_legal_drinking(self):
        """..."""
        self.assertTrue(LegalToDrinkCalculatorWithTwoBugs.is_legal(21, Nationality.American))

    def test_should_be_illegal_drinking(self):
        """..."""
        self.assertFalse(LegalToDrinkCalculatorWithTwoBugs.is_legal(8, Nationality.American))
```

Test case fails

```python
class LegalToDrinkCalculatorWithTwoBugs:
    """..."""

    @staticmethod
    def is_legal(age: int, nationality: Nationality) -> bool:
        """..."""
        legal = True
        if (Nationality.American == nationality and age >= 21) or (Nationality.British and age >= 16):
            legal = True
        return legal
```

Culprit?

Limitations of Statement Coverage

- Our initial single test case was is_legal(21, American)
- However, one of the bugs is the initial value of legal is set to **True**
- Uur initial test case we ***did not*** execute path where ‘if’ evaluated **False**

```
@staticmethod
def is_legal(age: int, nationality: Nationality) -> bool:
    """...."""
    legal = False
    if (Nationality.American == nationality and age >= 21) or (Nationality.British and age >= 16):
        legal = True
    return legal
```

Walk through the example and run the tests until you accept with 100% statement coverage there was still a latent defect

100% Statement Coverage != Correct

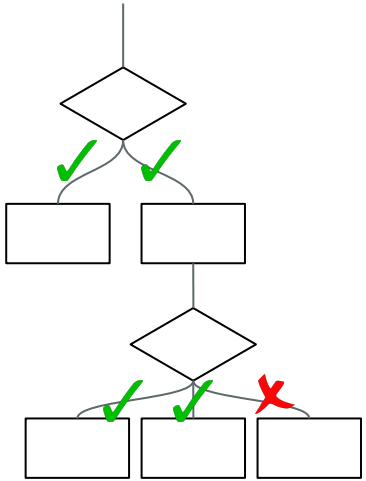
- The finding that 100% statement coverage does not guarantee correctness is important
- We need to understand why this is?
- Program is a state space of inputs and transitions
  - There are a set of possible paths through the program
  - 100% statement coverage ***does not mean*** we have exercised all transitions
  - Defects can remain undetected in other paths

(Even ***without*** an 'if' statement I could have introduced a / 0 error that would not work for certain inputs. Alternatively, I could have two values that only certain combinations caused an overflow etc. Two more scenarios where 100% statement coverage does not mean correctness)

Coverage Measure II: Branch Coverage

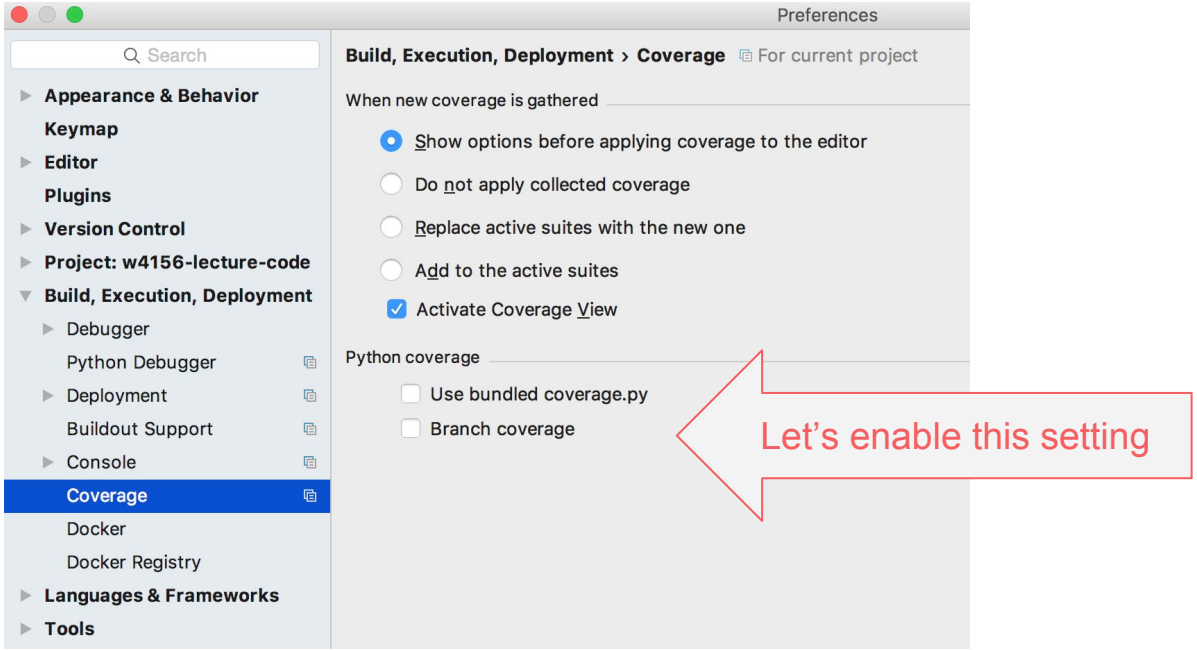
Branch coverage: asks what % of possible paths from control statement were exercised by the test suite.

i.e. : were all branch outcomes executed



Tooling: Enabling Branch Coverage

For some reason PyCharm does not ship with Branch Coverage enabled by default



Rerunning with Branch Coverage

Rerunning the test_legal_drinking **single test case** (21, American), with branch coverage enabled
We see that while we hit stmt coverage we did ***not*** hit branch coverage

This orange says we didn't hit branch coverage

```
9   class LegalToDrinkCalculatorWithTwoBugs:
10      """..."""
14
15      @staticmethod
16      def is_legal(age: int, nationality: Nationality) -> bool:
17          """..."""
24          legal = True
25          if (Nationality.American == nationality and age >= 21) or (Nationality.British and age >= 16):
26              legal = True
27          return legal
```

We even get told why!

```
9   class LegalToDrinkCalculatorWithTwoBugs:
10      """..."""
14
15      @staticmethod
16      def is_legal(age: int, nationality: Nationality) -> bool:
17          """..."""
24          legal = True
            .American == nationality and age >= 21) or (Nationality.British and age >= 16):
```

Line was hit
Line 24 didn't jump to line 27

100% Branch Coverage

```python
class Test100BranchCoverageOneBug(unittest.TestCase):

    def test_legal(self):
        self.assertTrue(LegalToDrinkCalculatorWithOneBug.is_legal(21, Nationality.American))

    def test_illegal(self):
        self.assertFalse(LegalToDrinkCalculatorWithOneBug.is_legal(8, Nationality.Ame
```

Adding a second test case

```python
class LegalToDrinkCalculatorWithOneBugs:

    @staticmethod
    def is_legal(age: int, nationality: Nationality) -> bool:
        """
        :param age: age of person buying alcohol
        :param nationality: nationality of the individual buying the alcohol
        """
        if (Nationality.American == nationality and age >= 21) or (Nationality.British and age >= 16):
            legal = True
        return legal
```

With **branch coverage enabled** we now hit 100%

Recap

1. We wrote some code
2. We wrote a test and achieved 100% statement coverage
3. But there was still a bug
4. We then discovered branch coverage and wrote tests to achieve 100% branch coverage
5. That exposed one of the latent bugs

Are we confident in the code? Are there still latent bugs?

Final Bug

The legal drinking age in the UK is 18 not 16!!!

```python
class LegalToDrinkCalculatorWithOneBugs:

    @staticmethod
    def is_legal(age: int, nationality: Nationality) -> bool:
        """
        :param age: age of person buying alcohol
        :param nationality: nationality of the individual buying the alcohol
        :return:
        """
        legal = False
        if (Nationality.American == nationality and age >= 21) or (Nationality.British and age >= 16):
            legal = True
        return legal
```

Remember, at this stage we have 100% statement and 100% branch coverage and we have still not exposed this bug

Limitations of Branch Coverage

However, although we have now hit 100% <u>branch</u> coverage there is still a bug ...

```
class Test100BranchCoverageOneBug(unittest.TestCase):

    def test_legal(self):
        self.assertTrue(LegalToDrinkCalculatorWithOneBug.is_legal(21, Nationality.American))

    def test_illegal(self):
        self.assertFalse(LegalToDrinkCalculatorWithOneBug.is_legal(8, Nationality.American))

    def test_illegal_british(self):
        self.assertFalse(LegalToDrinkCalculatorWithOneBug.is_legal(17, Nationality.British))
```

Added this test case

And it fails

```
Run  Nosetests in Test100BranchCoverageOneBug
3 tests done: 1 failed – 0ms

Test Results                                0ms
  tests.testing.theory.test_leg             0ms
    test_illegal_british                    0ms

Traceback (most recent call last):
  File "/Users/ewanlowe/Documents/developer/w4156-lecture-code/tests/testing/theory/test_legaldrinking.py",
    self.assertFalse(LegalToDrinkCalculatorWithOneBug.is_legal(17, Nationality.British))
AssertionError: True is not false

----------------------------------------------------------------------
Ran 3 tests in 0.002s

FAILED (failures=1)
```

Limitations of Branch Coverage

Again, branch detection has a gap/flaw. We executed every statement. We also constructed test cases such that the branch evaluated to both **True** and **False.**

```python
class LegalToDrinkCalculatorWithOneBug:

    @staticmethod
    def is_legal(age: int, nationality: Nationality) -> bool:
        """
        :param age: age of person buying alcohol
        :param nationality: nationality of the individual buying the alcohol
        :return:
        """
        legal = False
        if (Nationality.American == nationality and age >= 21) or (Nationality.British and age >= 16):
            legal = True
        return legal
```

- However, the ‘if’ has **two conditions (clauses)**
- If the first condition == True the whole branch is True (and second statement is not evaluated)
- If the first condition == False the second statement is evaluated. In our test cases it was always False
- The British legal drinking age is 18

Even with 100% branch coverage we can have undetected defects

Condition Coverage

Even with 100% branch coverage we can have undetected defects

(You guessed it)

Condition coverage: asks what whether each *conditional* has evaluated to both **True** and **False**

Condition Coverage

```python
class TestConditionCoverageNoBugs(unittest.TestCase):

    def push_assert(self, tple: Tuple):
        legal = LegalToDrinkCalculatorBugFreeIHope.is_legal(tple[0], tple[1])
        self.assertTrue(legal == tple[2])

    def test_legal_drinking(self):
        cases = [(21, Nationality.American, True),    # hits statement coverage
                 (20, Nationality.American, False),   # hits branch coverage
                 (18, Nationality.British, True),     # hits condition coverage (evaluated to false previously)
                 (17, Nationality.British, False),
                 ]

        map(lambda x: self.push_assert(x[0], x[1]), cases)
```

These test cases hit statement, branch and condition coverage

Summary: Coverage Measures

Coverage Measure	Description
Statement	All statements executed
Branch	Every branch evaluated T and F
Condition	Every conditional evaluated T and F

Summary: Coverage Measures

We can now answer the original question we posed ourselves.
Fry was not sure because there was two options:

		Test Suite Adequacy	
		Rigorous (how to define?)	Lousy (how to define?)
Test Suite	Passes	What do we think of quality and defects?	What do we think of quality and defects?
	Fails	At least one bug	

Summary: Coverage Measures

Rigour defined by higher % coverage and more aggressive measures (statement, branch, condition)

		Test Suite Adequacy	
		Rigorous **'Good Coverage'**	Lousy **'Poor Coverage'**
Test Suite	Passes	**Higher confidence in quality Lower defects likely**	**Lower confidence in quality. Higher defects likely**
	Fails	At least one bug	

Do not be confident in your code if your tests pass unless they are rigorous

LPT

Rant: You will often hear “testing takes too long or is too difficult”.

- ***Some*** developers don’t know/care to write good quality/effective tests
  - They write ‘random’ test cases. These test cases ***take time to write***.
  - However, ***we know*** these test cases have a ***low probability*** of exposing defects
  - (They also write code that is difficult to test - we will cover that later)
  - They ***resist testing*** because ***<u>they do</u>*** get a poor return on time
  - The consequence is they will ship ***lower quality*** and spend ***more time*** fixing bugs
- However, ***<u>we</u>*** can ***apply techniques*** quickly:
  - We will then write tests with ***high probability*** of exposing defects
  - We will have an excellent return on our time/investment in testing
  - We will ship ***higher quality*** and spend ***more time writing new functionality*** vs fixing bugs
- But - as always - be pragmatic. Balance design impact, commercials and assess your test strategy

The wrap around counter example is based on an hiring test we gave to every single senior engineer.
<50% of engineers passed this question and this question was a veto

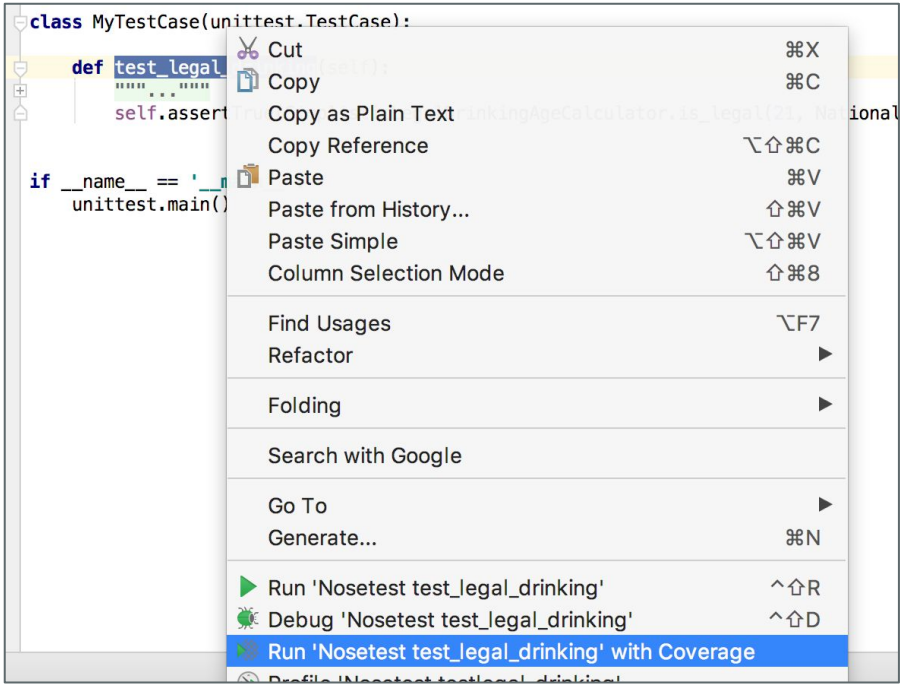
Pop Quiz

Question	Answer
Testing verified the {functional, non-functional, both} behavior of programs?	
Can we exhaustively test a non-trivial program?	
Under normal circumstance can we prove correctness of software?	
A good test case has a <high/low/not relevant> probability of exposing a defect?	
Within black box testing the tester can see <specification, internals, both>?	
Within white box testing the tester can see <specification, internals, both>?	
Once I have written a bunch of tests that pass I am <done, not done, depends>?	
Coverage is a <method of assessing test suite adequacy, way to identify test cases, american and canadian football defensive scheme>	
If I have 100% code coverage my code is correct?	

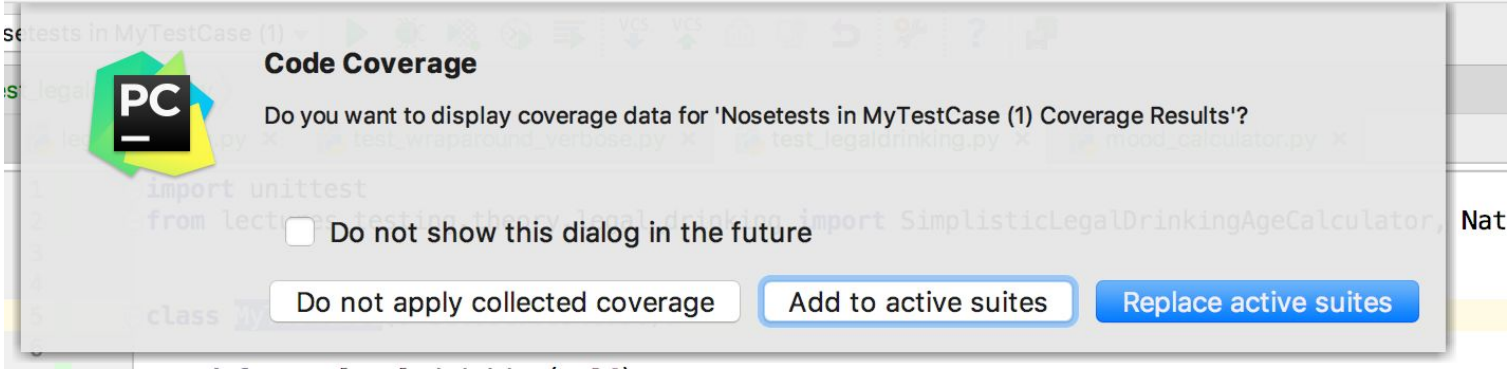
Reading

Material	Optionality
Finish Beginning S.E. Chapter 4 (across this and next lecture)	Required
Understanding Unit Testing	Required

Coverage Tool



Coverage Report



Coverage Report

Coverage Nosetests in MyTestCase (1)

18% files, 94% lines covered

Element	Statistics, %
.idea	
lectures	20% files, 100% lines ...
tests	16% files, 85% lines c...
README.md	

Coverage Nosetests in MyTestCase (1)

25% files, 100% lines covered in 'theory'

Element	Statistics, %
__init__.py	
legal_drinking.py	100% lines covered
mood_calculator.py	not covered
plane_navigation_...	not covered
README.md	
wrap_around_cou...	not covered

Coverage Nosetests in MyTestCase (1)

25% files, 100% lines covered in 'theory'

Element	Statistics, %
__init__.py	
legal_drinking.py	100% lines covered
mood_calculator.py	not covered
plane_navigation_...	not covered
README.md	
wrap_around_cou...	not covered