Testing

Oct 4, 2019

“Testing shows the presence, not the absence of bugs.”

–Edsger W. Dijkstra, 1969

Testing

- Execute the program with selected inputs in a controlled environment.
- Goals
    - Reveals bugs, so they can be fixed (main goal)
    - Assess quality
    - Clarify the specification, documentation

What to test?

- Functional correctness of a method (e.g., computations, contracts)
- Functional correctness of a class (e.g., class invariants)
- Behavior of a class in subsystem/multiple subsystems/the entire system
- Behavior when interacting with the world
  - files, networks, sensors, …
  - nondeterminism, parallelism
  - interaction with users
- Other qualities (performance, robustness, usability, security, …)

Automated testing

- Execute a program with specific inputs, check output for expected values
- Easier to test small pieces than testing user interactions
- Set up testing infrastructure
  - Execute tests regularly
    - After every change!

Selecting test cases: common strategies

- Read specification first
- Write test for (1) representative cases (2) Invalid cases (3) Boundary conditions
- Stress tests
- Think like an attacker
- How many tests should you write?

Example

```
/**
 * computes the sum of the first len values of the array
 *
 * @param array array of integers of at least length len
 * @param len number of elements to sum up
 * @return sum of the array values
 */
int total(int array[], int len);
```

What should you test?

- Empty array
- Array of length 1 and 2
- Array with negative numbers
- Invalid length (negative, longer than array.length)
- Null as array
- Test with an extremely long array
- …

Testable Code

```
//700LOC
public boolean foo() {
    try {
        synchronized () {
            if () {
            } else {
            }
            for () {
                if () {
                    if () {
                        if () {
                            if ()?
                            {
                                if () {
                                    for () {
                                    }
                                }
                            }
                        } else {
                            if () {
                                for () {
                                    if () {
                                    } else {
                                    }
                                    if () {
                                    } else {
                                        if () {
                                        }
                                    }
                                    if () {
                                        if () {
                                            if () {
                                                for () {
                                                }
                                            }
                                        }
```

- Think about testing when writing code
- Separate parts of code to make them independently testable
- Abstract functionality behind interface, make it replaceable
- Test-Driven Development
    - A design and development method in which you write tests before you write the code

Unit Test

- Tests for small units: functions, classes, subsystems
  - Smallest testable part of a system
  - Test parts before assembling them
  - Intended to catch local bugs
- Typically written by developers
- Many small, fast-running, independent tests
- Little dependencies on other system parts or environment
- Insufficient but a good starting point, extra benefits:
  - Documentation (executable specification)
  - Design mechanism (design for testability)

unittest for Python

- Built into Python standard library
- Easy to use
- Good tool support

```python
# this is the function that we will be testing
def hello_world():
    return "Hello, World!"


# import the unittest module, which we will use to write our tests
import unittest


# With unittest, tests are grouped as methods of classes.
# Each such class must be a sub-class of 'unittest.TestCase'.
# And that's about all you need to know about these classes!
class TestHelloWorld(unittest.TestCase):
    """Tests for the hello_world() function"""

    # Each test is written as a method with a name beginning with "test_"
    def test_return_value(self):
        # Writing a doc-string for each test, explaining what it tests,
        # is a good idea.
        """test that hello_world() returns 'Hello, World!'"""

        # self.assertEqual() will make the test fail if the arguments are not equal.
        self.assertEqual(hello_world(), "Hello, World!")

        # If no assertions fail, the test passes successfully. Note that this
        # happens automatically; we don't have to return a value or anything
        # of the sort.
```

Common assertXXX

- assertEqual <-> assertNotEqual
- assertTrue <-> assertFalse
- assertIs <-> assertIsNot
- assertIsNone <-> assertIsNotNone
- assertGreater, assertGreaterEqual, …

```
import unittest

class TestMethods(unittest.TestCase):
    def test_1(self):
        self.assertTrue(3 > 4)

    def test_2(self):
        self.assertGreater(3, 4)

if __name__ == "__main__":
    unittest.main()
```

```
FF
======================================================================
FAIL: test_1 (__main__.TestMethods)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 5, in test_1
    self.assertTrue(3 > 4)
AssertionError: False is not true

======================================================================
FAIL: test_2 (__main__.TestMethods)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 8, in test_2
    self.assertGreater(3, 4)
AssertionError: 3 not greater than 4

----------------------------------------------------------------------
Ran 2 tests in 0.000s

FAILED (failures=2)
```

Exercise: Writing Unit Tests

- Download recitation material: https://cmu.box.com/s/zt8f7czudt0wmzdqt6a1inlokz2idpu6
- Use your favorite source code editing tool to finish TODOs in
  - exercise/assertion_comparison.py
  - exercise/assertion_container.py (use assertCountEqual, what does it do?)
- If using command line, add the following:

```
if __name__ == '__main__':
    unittest.main()
```

Test Coverage

When to stop writing tests?

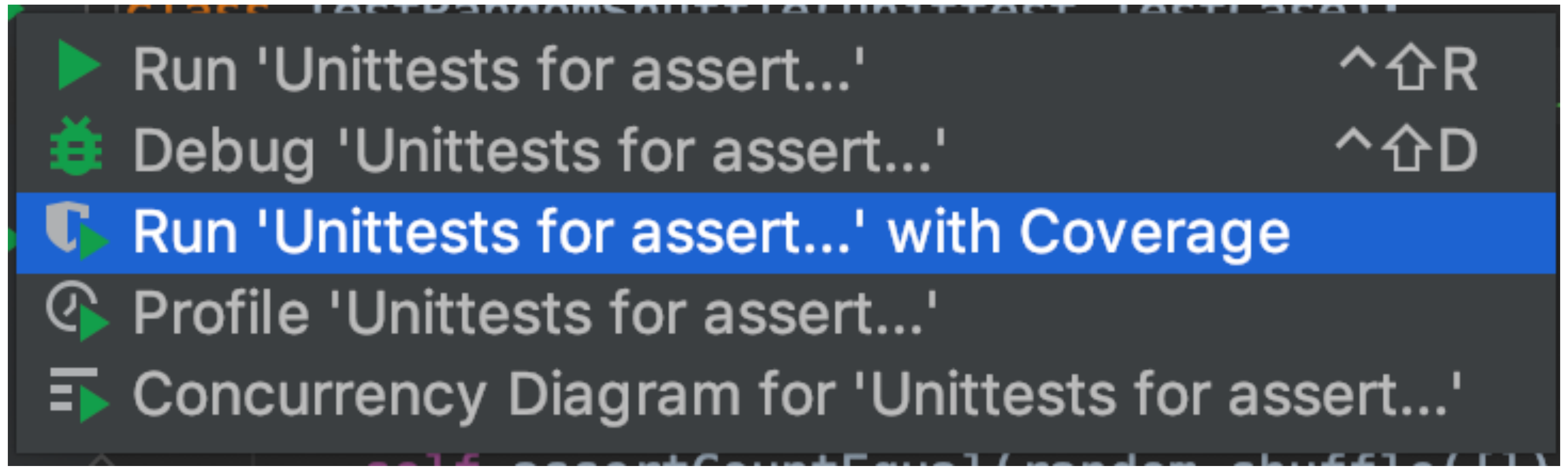
- Outlook: Statement coverage
  - Trying to test all parts of the implementation
  - Execute every statement, ideally

Does 100% coverage guarantee correctness?

Exercise: 100% Statement Coverage

- Open exercise/grade.py with your favorite code editing tool
- Write test cases to achieve 100% statement coverage
- Did you spot the bug while writing test cases?

IntelliJ



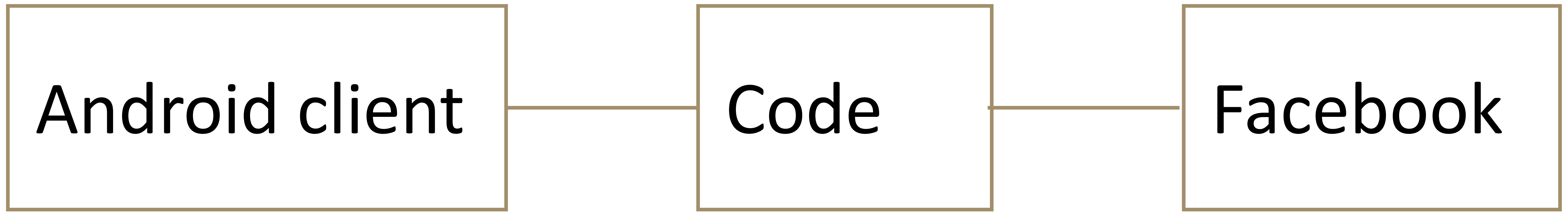
Run 'Unittests for assert...' ^⇧R
Debug 'Unittests for assert...' ^⇧D
Run 'Unittests for assert...' with Coverage
Profile 'Unittests for assert...'
Concurrency Diagram for 'Unittests for assert...'

Command Line

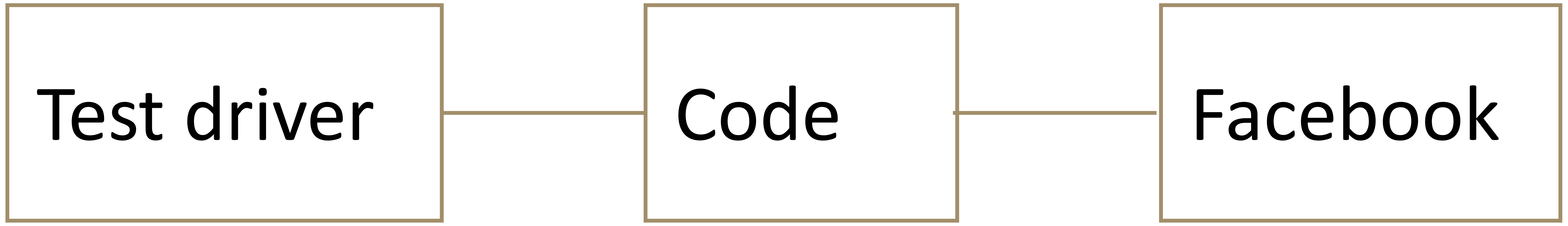
https://coverage.readthedocs.io/en/v4.5.x/#quick-start

Testing with Stub

When you can’t see the entire picture, imaging it!

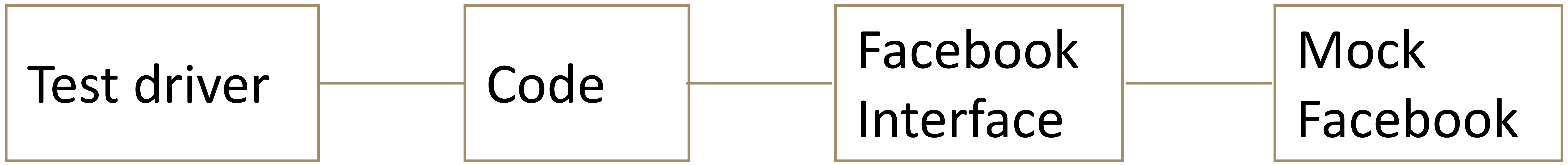


```
void buttonClicked() {
   render(getFriends());
}
List<Friend> getFriends() {
   Connection c = http.getConnection();
   FacebookAPI api = new FacebookAPI(c);
   List<Node> persons = api.getFriends("john");
   for (Node person1 : persons) {
       for (Node person2 : persons) {
      …
       }
   }
   return result;
}
```



```
@Test void testGetFriends() {
   assert getFriends() == ...;
}
List<Friend> getFriends() {
   Connection c = http.getConnection();
   FacebookAPI api = new FacebookAPI(c);
   List<Node> persons = api.getFriends("john");
   for (Node person1 : persons) {
       for (Node person2 : persons) {
      …
       }
   }
   return result;
}
```

This will not quite work



```
@Test void testGetFriends() {
   assert getFriends() == …;
}
List<Friend> getFriends() {
   Connection c = http.getConnection();
   FacebookAPI api = new MockFacebook(c);
   List<Node> persons = api.getFriends("john");
   for (Node person1 : persons) {
       for (Node person2 : persons) {
       …
        }
   }
   return result;
}
```

```
class MockFacebook implements FacebookInterface {
    void connect() {}
    List<Node> getFriends(String name) {
        if ("john".equals(name)) {
            List<Node> result=new List();
            result.add(…);
            return result;
        }
    }
}
```

Stubs

- A dummy stand-in for testing purposes
- Simplest case: an object that returns a default value
- Example
  - Kafka stream

Mocks

- Object configured at runtime to behave in a certain way under certain circumstances
- Often needs mocking framework support
- unittest.mock: https://docs.python.org/3/library/unittest.mock.html