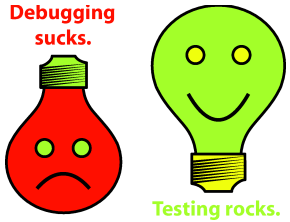

Testing on the Toilet

Better Stubbing in Python

January 22, 2007

So you've learned all about method stubs, mock objects, and fakes. You might be tempted to stub out slow or I/O-dependent built-ins. For example:

```
def Foo(path):
  if os.path.exists(path):
    return DoSomething()
  else:
    return DoSomethingElse()

def testFoo(self):                # Somewhere in your unit test class
  old_exists = os.path.exists
  try:
    os.path.exists = lambda x: True
    self.assertEqual(Foo('bar'), something)
    os.path.exists = lambda x: False
    self.assertEqual(Foo('bar'), something_else)
  finally:
    os.path.exists = old_exists    # Remember to clean-up after yourself!
```

Congratulations, you just achieved 100% coverage! Unfortunately, you might find that this test fails in strange ways. For example, given the following `DoSomethingElse`, which checks the existence of a different file:

```
def DoSomethingElse():
  assert os.path.exists(some_other_file)
  return some_other_file
```

`Foo` will now throw an exception in its second invocation because `os.path.exists` returns False so the assertion fails.

You could avoid this problem by stubbing or mocking out `DoSomethingElse`, but the task might be daunting in a real-life situation. Instead, it is safer and faster to parameterize the built-in:

```
def Foo(path, path_checker=os.path.exists):
  if path_checker(path):
    return DoSomething()
  else:
    return DoSomethingElse()

def testFoo(self):
  self.assertEqual(Foo('bar', lambda x: True), something)
  self.assertEqual(Foo('bar', lambda x: False), something_else)
```

More information, feedback, and discussion:

http://googletesting.blogspot.com