Test

```
import unittest
from newscheck.core import find_bad_articles

class TestBadArticleFinder(unittest.TestCase):
    def test_find_bad_articles(self):

        data = {
            "good.txt": ["lorem ipsum dolorum"],
            "bad.txt": ["i am the bad article"]
        }

        def check(lines):
            return lines[0] == "i am the bad article"

        bad_articles = list(find_bad_articles(data.items(), check))
        self.assertEqual(1, len(bad_articles))
        self.assertEqual("bad.txt", bad_articles[0])
```

Hynek Schlawack
@hynek

I feel like the possibility of monkey patching in tests sets new programmers in dynamic languages on the wrong track. A shortcut that can cost you years.

Go got right: define your (narrow) interfaces, make it easy to replace parts with fakes. _Leads_ you right to SRP.

11:29 PM · Sep 11, 2019 · Twitterrific for iOS

17 Retweets **61** Likes

Hynek Schlawack @hynek · Sep 11
Replying to @hynek
You also prob don't need mock(s). Brittle constructs of Mocks-in-Mocks-in-Mocks are neither more readable nor less work than writing a fake or using something more robust like pypi.org/project/preten… Specifically assert_called is rarely appropriate; eg when patching sys.exit.