

Stop writing tests!

Zac Hatfield-Dodds

a provocative but entirely serious talk

zhd.dev | hypofuzz.com



Talking about Testing

*testing is an activity
where we execute code
to find bugs or
check for regressions*

- choose inputs
- run <the thing>
- check behaviour
- repeat as needed



David R. MacIver

@DRMacIver



Every time someone uses reversing a list twice to demonstrate property-based testing, I take a drink.

No, this isn't a drinking game, I'm just being driven to drink by bad examples.

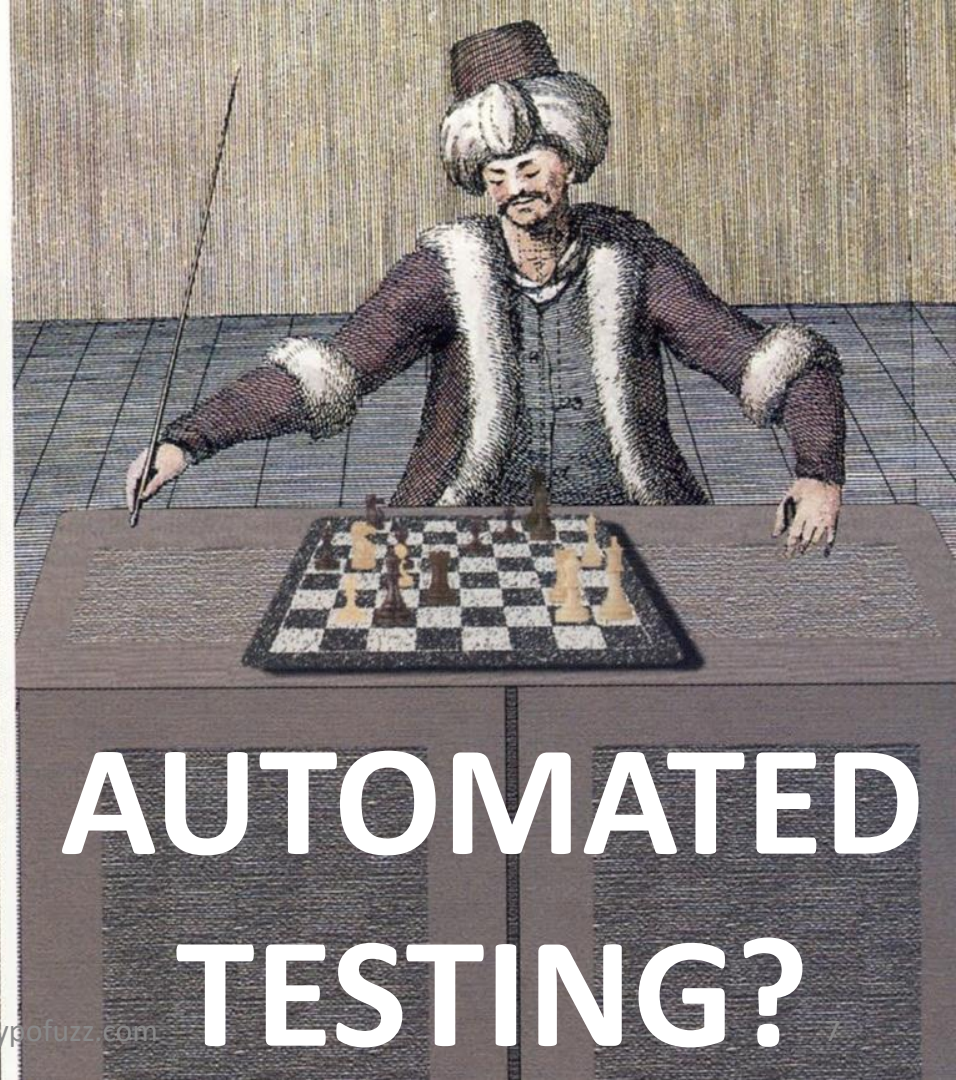
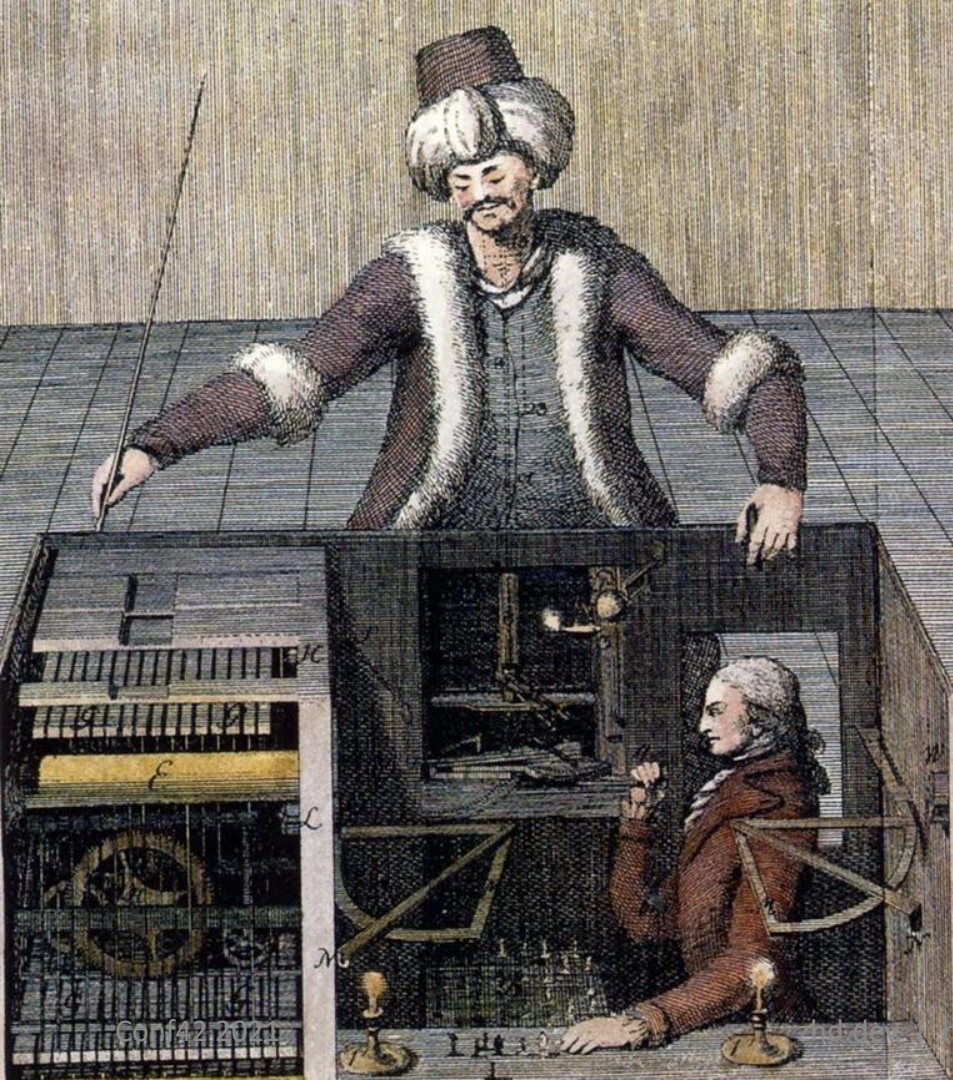
11:42 PM · Jun 17, 2019



```
def test_sorted_manually_ints():  
    assert sorted([1, 2, 3]) == [1, 2, 3]  
  
def test_sorted_manually_floats():  
    assert sorted([3.0, 2.0, 1.0]) == [1.0, 2.0, 3.0]  
  
def test_sorted_manually_strings():  
    assert sorted(["b", "c", "a"]) == ["a", "b", "c"]
```



```
@pytest.mark.parametrize(
    "arg, result",
    [
        ([1, 2, 3], [1, 2, 3]),
        ([3.0, 2.0, 1.0], [1.0, 2.0, 3.0]),
        (["b", "c", "a"], ["a", "b", "c"]),
    ],
)
def test_sorted_parametrized_with_known_results(arg, result):
    assert sorted(arg) == result
```



```
@pytest.mark.parametrize(
    "arg",
    [[1, 2, 3], [3.0, 2.0, 1.0], ["b", "c", "a"]],
)
def test_sorted_parametrized_with_reference(arg):
    assert sorted(arg) == trusted_sort_function(arg)
```




```
@pytest.mark.parametrize(
    "arg",
    [[1, 2, 3], [3.0, 2.0, 1.0], ["b", "c", "a"]],
)
def test_sorted_parametrized_without_known_result(arg):
    result = sorted(arg)
    # Check that the `result` is in sorted order
    for a, b in zip(result, result[1:]):
        assert a <= b
```



```
@pytest.mark.parametrize("arg", ...)
def test_sorted_parametrized_without_known_result(arg):
    result = sorted(arg)
    # Check that the `result` is in sorted order
    for a, b in zip(result, result[1:]):
        assert a <= b
    # Check that `result` has the same elements as `arg`
    assert len(arg) == len(result)
    assert set(arg) == set(result)
```



```
@pytest.mark.parametrize("arg", ...)
def test_sorted_parametrized_without_known_result(arg):
    result = sorted(arg)
    # Check that the `result` is in sorted order
    for a, b in zip(result, result[1:]):
        assert a <= b
    # Check that `result` has the same elements as `arg`
    assert tuple(result) in itertools.permutations(arg)
```



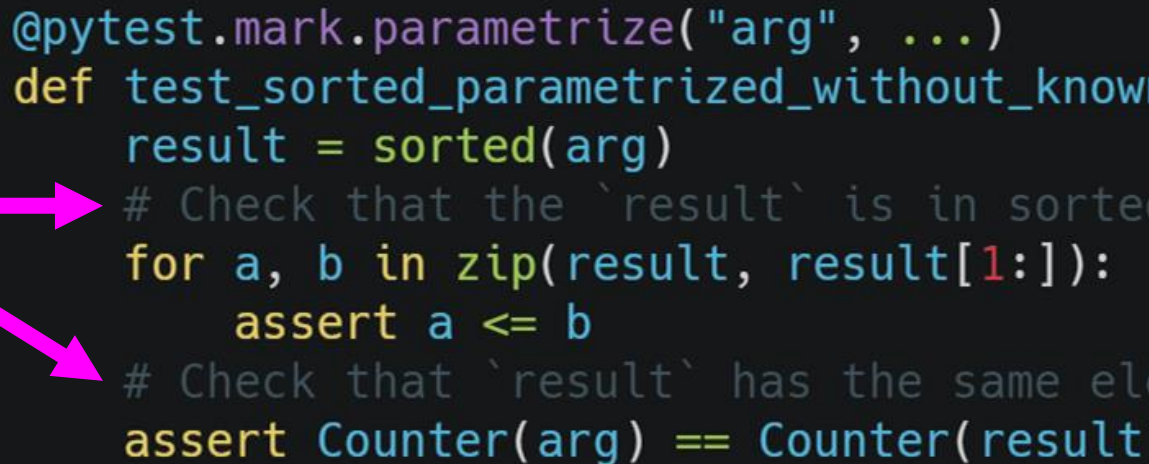
```
@pytest.mark.parametrize("arg", ...)
def test_sorted_parametrized_without_known_result(arg):
    result = sorted(arg)
    # Check that the `result` is in sorted order
    for a, b in zip(result, result[1:]):
        assert a <= b
    # Check that `result` has the same elements as `arg`
    assert Counter(arg) == Counter(result)
```


These are properties!

Sorting is *fully specified* by
just two properties:

1. Output is in order
2. Same elements in both

partial specs are still very
useful for finding bugs :-)



```
@pytest.mark.parametrize("arg", ...)
def test_sorted_parametrized_without_know
    result = sorted(arg)
    # Check that the `result` is in sorted order
    for a, b in zip(result, result[1:]):
        assert a <= b
    # Check that `result` has the same elements as `arg`
    assert Counter(arg) == Counter(result)
```

No matter how rigorous her
analysis or heroic his imagination,

**no person can write a test case
that would never occur to them**

-- Thomas Schelling, almost



```
@given(arg=st.one_of(
    st.lists(st.integers() | st.floats()),
    st.lists(st.text()),
))
def test_sorted_property_based(arg):
    result = sorted(arg)
    # Check that the `result` is in sorted order
    # Check that `result` has the same elements as `arg`
```



```
@given(arg=st.one_of(
    st.lists(st.integers() | st.floats(allow_nan=False)),
    st.lists(st.text()),
))
def test_sorted_property_based(arg):
    result = sorted(arg)
    # Check that the `result` is in sorted order
    # Check that `result` has the same elements as `arg`
```


PROPERTY-BASED TESTING





FOOLPROOF PLAN

1. `pip install hypothesis`
2. Skim <https://hypothesis.rtf.d.io>
3. ~~BUGS!~~ **PROFIT!**

\$ hypothesis write my.tests

an interactive live demo
of the Ghostwriter.



Migrating to PBT

```
def test_checkout_new_branch():  
    """Checking out a new branch makes it the active branch."""  
    with tempfile.TemporaryDirectory() as tmpdir:  
        repo = Repository.initialize(tmpdir.name)  
        repo.checkout("new-branch", create=True)  
        assert "new-branch" == repo.get_active_branch()
```


Migrating to PBT

```
def test_checkout_new_branch(branch_name="new-branch"):
    """Checking out a new branch makes it the active branch."""
    with tempfile.TemporaryDirectory() as tmpdir:
        repo = Repository.initialize(tmpdir.name)
        repo.checkout(branch_name, create=True)
        assert branch_name == repo.get_active_branch()
```

Migrating to PBT

```
@given(branch_name=st.just("new-branch"))
def test_checkout_new_branch(branch_name):
    """Checking out a new branch makes it the active branch."""
    with tempfile.TemporaryDirectory() as tmpdir:
        repo = Repository.initialize(tmpdir.name)
        repo.checkout(branch_name, create=True)
        assert branch_name == repo.get_active_branch()
```

Migrating to PBT

```
def valid_branch_names():  
    # TODO: Improve this strategy.  
    return st.just("new-branch")  
  
@given(branch_name=valid_branch_names())  
def test_checkout_new_branch(branch_name):  
    """Checking out a new branch makes it the active branch."""  
    with tempfile.TemporaryDirectory() as tmpdir:  
        repo = Repository.initialize(tmpdir.name)  
        repo.checkout(branch_name, create=True)  
        assert branch_name == repo.get_active_branch()
```

Migrating to PBT

```
def valid_branch_names():  
    # TODO: read `man git-check-ref-format`  
    return st.text()  
  
@given(branch_name=valid_branch_names())  
def test_checkout_new_branch(branch_name):  
    """Checking out a new branch makes it the active branch."""  
    with tempfile.TemporaryDirectory() as tmpdir:  
        repo = Repository.initialize(tmpdir.name)  
        repo.checkout(branch_name, create=True)  
        assert branch_name == repo.get_active_branch()
```

Migrating to PBT

```
def valid_branch_names():  
    # TODO: probably too narrow, but OK for now.  
    return st.text(alphabet=ascii_letters, min_size=1, max_size=95)  
  
@given(branch_name=valid_branch_names())  
def test_checkout_new_branch(branch_name):  
    """Checking out a new branch makes it the active branch."""  
    with tempfile.TemporaryDirectory() as tmpdir:  
        repo = Repository.initialize(tmpdir.name)  
        repo.checkout(branch_name, create=True)  
        assert branch_name == repo.get_active_branch()
```

Migrating to PBT

```
def valid_branch_names():  
    # TODO: probably too narrow, but OK for now.  
    return st.text(alphabet=ascii_letters, min_size=1, max_size=95)  
  
@given(branch_name=valid_branch_names())  
def test_checkout_new_branch(branch_name):  
    """Checking out a new branch makes it the active branch."""  
    with tempfile.TemporaryDirectory() as tmpdir:  
        repo = Repository.initialize(tmpdir.name)  
        repo.checkout(branch_name, create=True)  
        assert branch_name == repo.get_active_branch()
```


Migrating to PBT

```
@given(branch_name=valid_branch_names(), repo=repositories())
def test_checkout_new_branch(branch_name, repo):
    """Checking out a new branch makes it the active branch."""
    assume(branch_name not in repo.get_branches())
    repo.checkout(branch_name, create=True)
    assert branch_name == repo.get_active_branch()
```

Coverage-guided fuzzing: Atheris


No targets? No problem - target “executed this line of code”!

Atheris is Google’s libfuzzer wrapper for Python

- Designed to run a single function for hours or days
- Great for C-extensions and native code
- Hypothesis integrates well with traditional fuzzers



```
@given(...)  
def test_foo(...):
```



```
atheris.Setup(sys.argv, test_foo.hypothesis.fuzz_one_input)  
atheris.Fuzz()
```

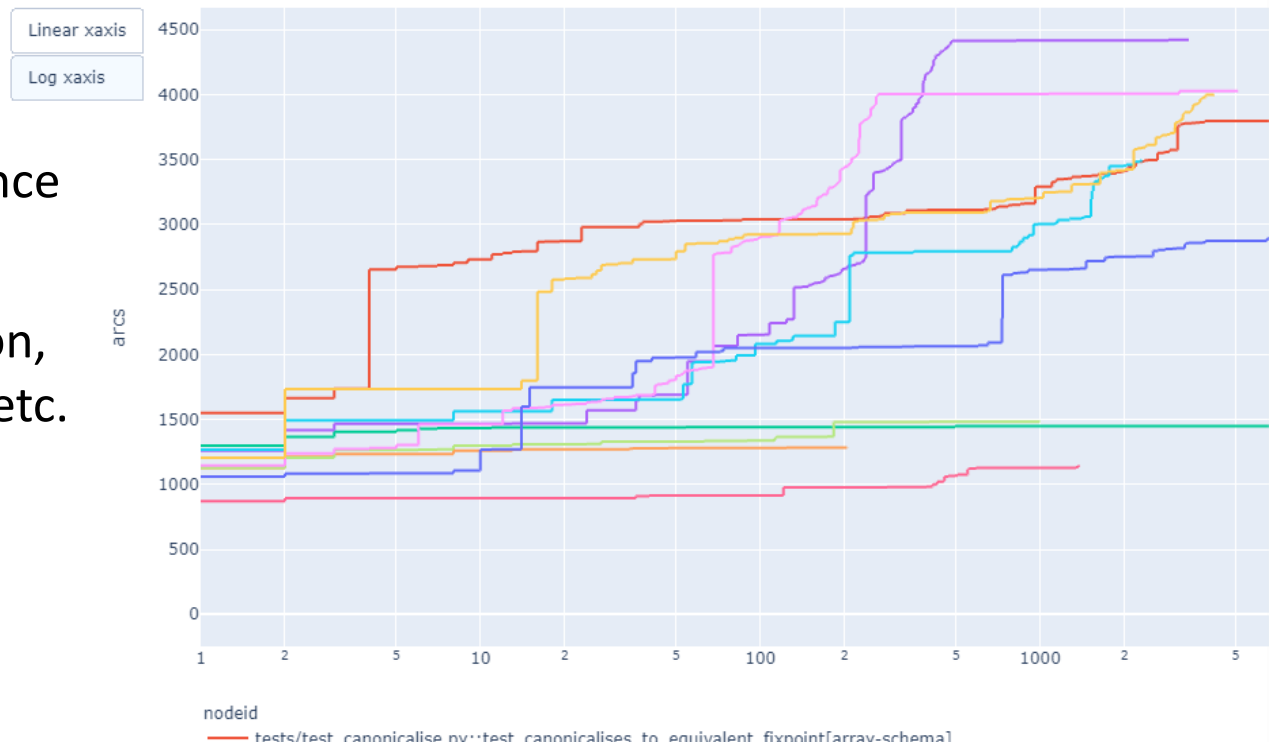
Coverage-guided fuzzing: HypoFuzz

HypoFuzz is Zac's fuzzing engine for Hypothesis *test suites*

- pure-python
- target() and event()
- runs all the tests at once

Better workflow integration,
great database support*, etc.

*but not better than .fuzz_one_input



that was ***Stop
writing
tests!***

Zac Hatfield-Dodds

zhd.dev / hypofuzz.com

