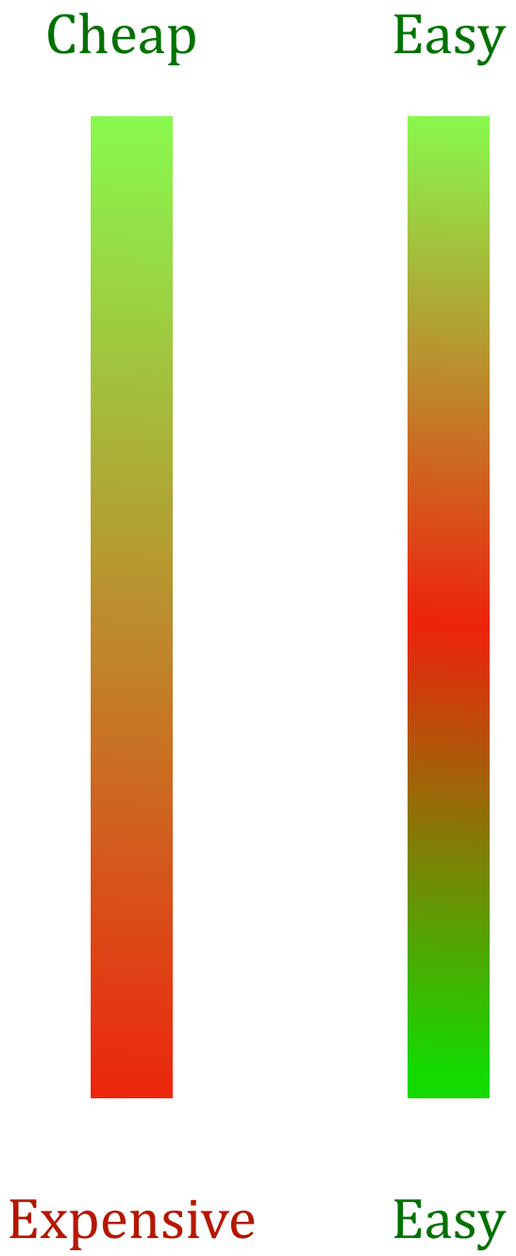
Testing in Duckietown

Challenges for testing in robotics

- Robots live in an "**open world**" that you cannot fully model.
    - No nice "theorems" possible.
- Robots live in **multi-domains**: hardware, software, environment, users are hard to model, and very hard to model in a coherent framework.
- **Variability** is a big problem: of the *hardware*, of the *compute*, of the *environment*, etc.
- **Non-modularity**: hard to test a component in isolation.
- **Expensive experiments**

Testing hierarchy

- **Unit tests**: single function, single module
- **Integration tests**: multiple modules, testing their communication/interaction
- **Functional tests:** multiple modules, testing the end-to-end functionality, often with synthetic data.
- **Regression tests:** multiple modules, testing on real data
- **Simulation tests:** testing in simulation; multiple fidelity levels.
- **Hardware in the loop (HWIL) tests:** tests performance (cpu, network, …)
- **Flight tests**: closed course (controlled conditions), or actual roads



Testing we do in Duckietown

• **Unit tests**: single function, single module	**nosetests / CI**
• **Integration tests**: multiple modules, testing their communication/interaction • **Functional tests:** multiple modules, testing the end-to-end functionality, often with synthetic data. • **Regression tests:** multiple modules, testing on real data	**limited to special cases in DT**
• **Simulation tests:** testing in simulation; multiple fidelity levels.	**AI-DO cloud simulation**
• **Hardware in the loop (HWIL) tests:** tests performance (cpu, network, …)	
• **Flight tests**: closed course (controlled conditions), or actual roads	**AI-DO autolab evaluation**

Nosetests

- **nose** is the simplest way to create unit tests.
- Simply create functions by the name “test*” in files by name “*test*.py”.
- Running “nose package” will run all tests in all files.

mytests.py

```
def test1():
    assert 1 + 2 == 3

def test2():
    assert 0.1 + 0.2 == 0.3
```

$ nosetests mytests.py

```
======================================================================
FAIL: mytests.test2
----------------------------------------------------------------------
Traceback (most recent call last):
  File "lib/python3.7/site-packages/nose/case.py", line 198, in runTest
    self.test(*self.arg)
  File "mytests.py", line 5, in test2
    assert 0.1 + 0.2 == 0.3
AssertionError

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

Code coverage

- Suggested services: **codecov** or **coveralls**
- Code coverage shows which lines are executed or not.

Coverage for **util** : 80%

5 statements | 4 run | 1 missing | 0 excluded

```
1 from math import ceil, pi
2
3 def borkify(i):
4     return int(ceil(i/pi) + 1)
5
6 def fnord(j):
7     return j/2 - 3
```

“dead code”

Example for duckietown-world

https://codecov.io/gh/duckietown/duckietown-world/

/ src / duckietown_world

Files					Coverage
geo	433	387	0	46	89.38%
rules	320	294	0	26	91.88%
seqs	170	140	0	30	82.35%
svg_drawing	401	332	0	69	82.79%
utils	407	226	0	181	55.53%
world_duckietown	1,359	1,246	0	113	91.69%
__init__.py	14	14	0	0	100.00%
Folder Totals (7 files)	3,104	2,639	0	465	85.02%
Project Totals (45 files)	3,104	2,639	0	465	85.02%

Example for duckietown-world

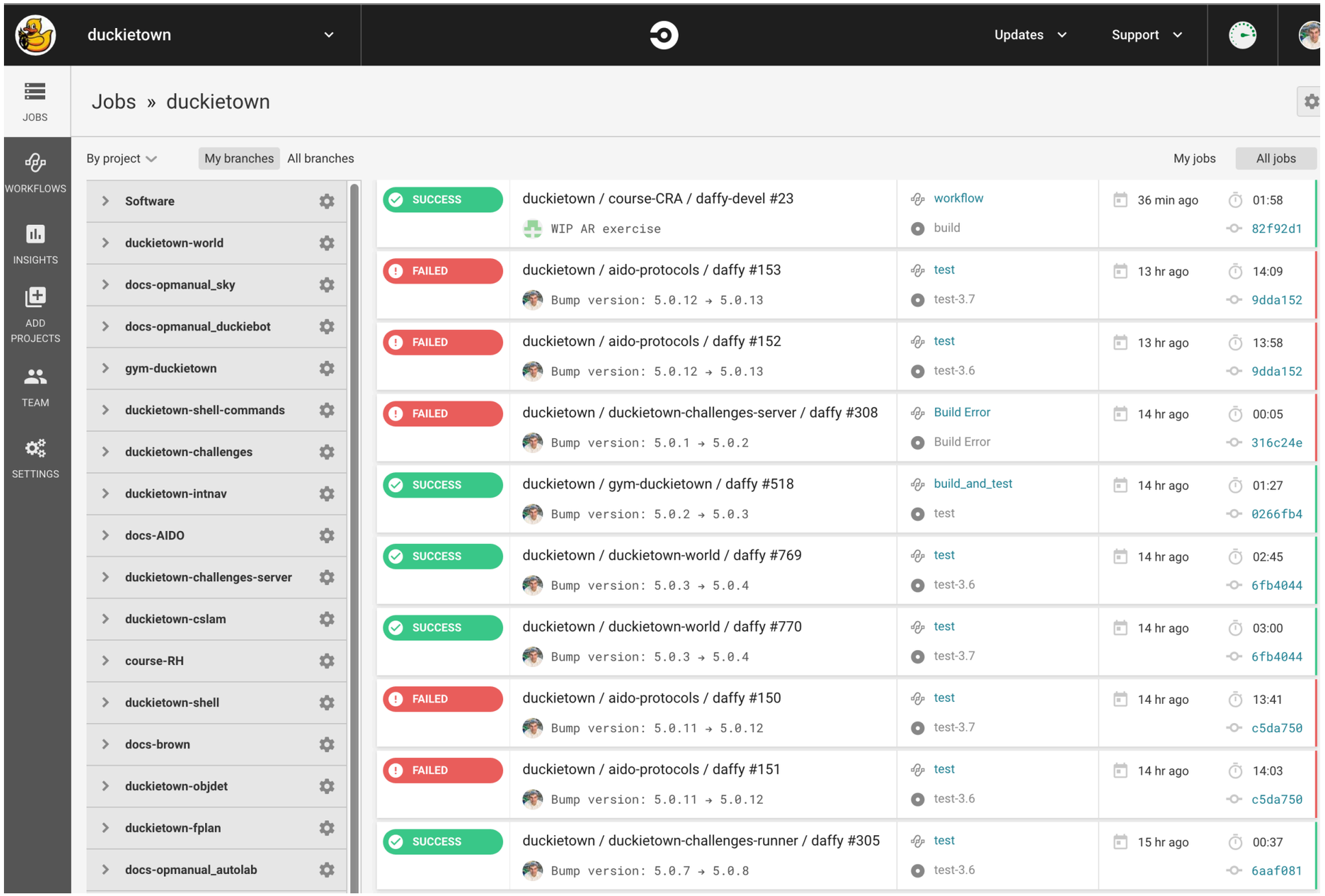
/ src / duckietown_world / world_duckietown / **duckietown_map.py**

72.22% | ø | ø

```
1   # coding=utf-8
2   from duckietown_world.geo import PlacedObject, SE2Transform
3
4   __all__ = [
5       'DuckietownMap',
6   ]
7
8
9   class DuckietownMap(PlacedObject):
10      def __init__(self, tile_size, *args, **kwargs):
11          self.tile_size = tile_size
12          PlacedObject.__init__(self, *args, **kwargs)
13
14      def params_to_json_dict(self):
15          return dict(tile_size=self.tile_size)
16
17      def se2_from_curpos(self, cur_pos, cur_angle):
18          """ Conversion from Duckietown Gym Simulator coordinates z"""
19          H = self.children['tilemap'].H
20          gx, gy, gz = cur_pos
21          p = [gx, (H - 1) * self.tile_size - gz]
22          transform = SE2Transform(p, cur_angle)
23          return transform
24
25      def get_drawing_children(self):
26          children = sorted(self.children)
27          children.remove('tilemap')
28          return ['tilemap'] + children
```

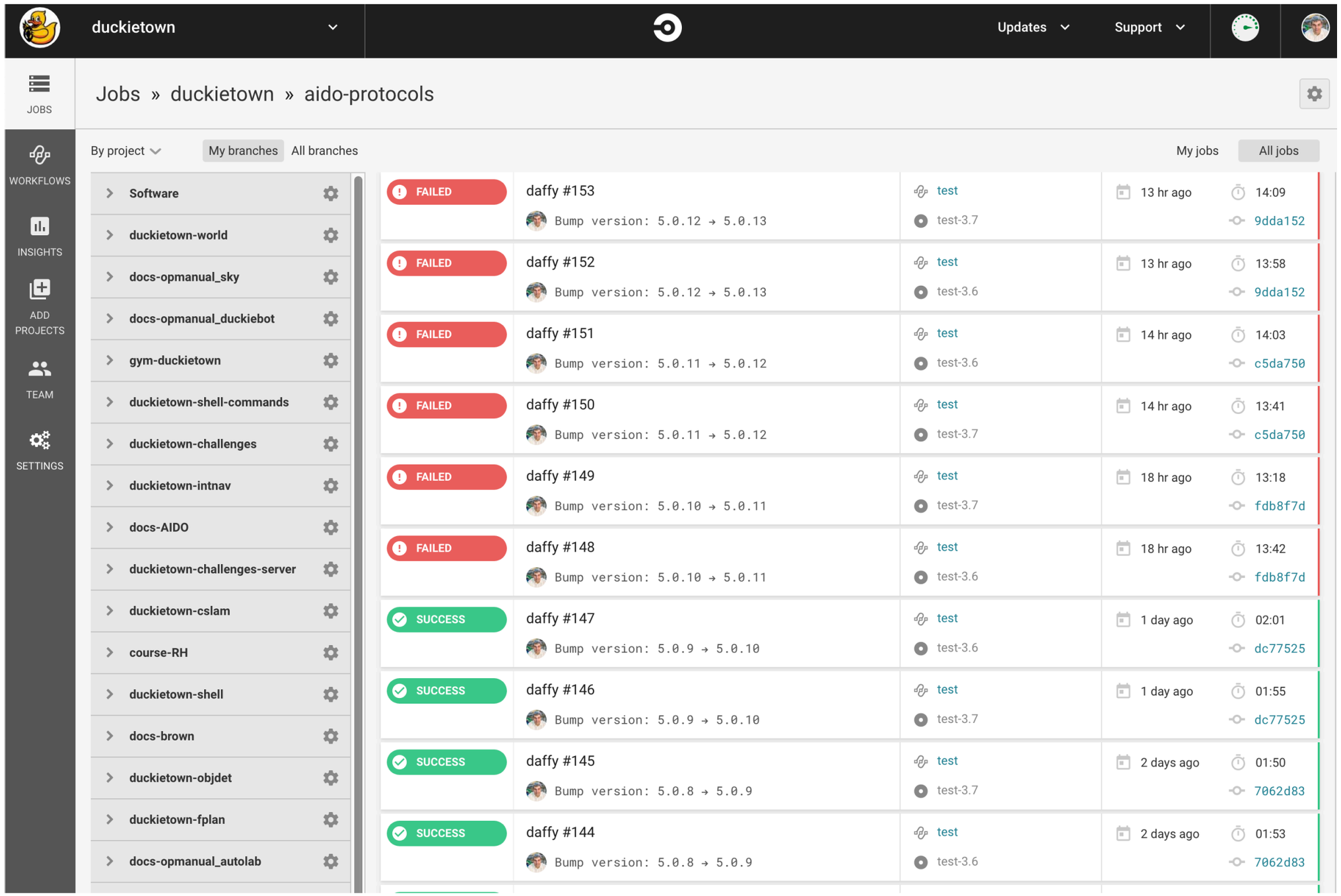
Automatic testing/coverage in the cloud

https://circleci.com/gh/duckietown



Job history shows when the problem was introduced

https://circleci.com/gh/duckietown/aido-protocols



The change that broke the test

https://github.com/duckietown/aido-protocols/commit/fdb8f7dfeaed74dabfb15d1044035b5f11cda000

Search or jump to... / Pull requests Issues Marketplace Explore

duckietown / aido-protocols

Unwatch 3 Star 1 Fork 0

Code Issues 0 Pull requests 0 Security Insights Settings

Bump version: 5.0.10 → 5.0.11

daffy

Browse files

AndreaCensi committed 18 hours ago

1 parent 21467bc commit fdb8f7dfeaed74dabfb15d1044035b5f11cda000

Showing **2 changed files** with **2 additions** and **2 deletions**.

Unified Split

2 .bumpversion.cfg

```
@@ -1,5 +1,5 @@
  [bumpversion]
- current_version = 5.0.10
+ current_version = 5.0.11
  files = src/aido_schemas/__init__.py
  commit = True
  tag = True
```

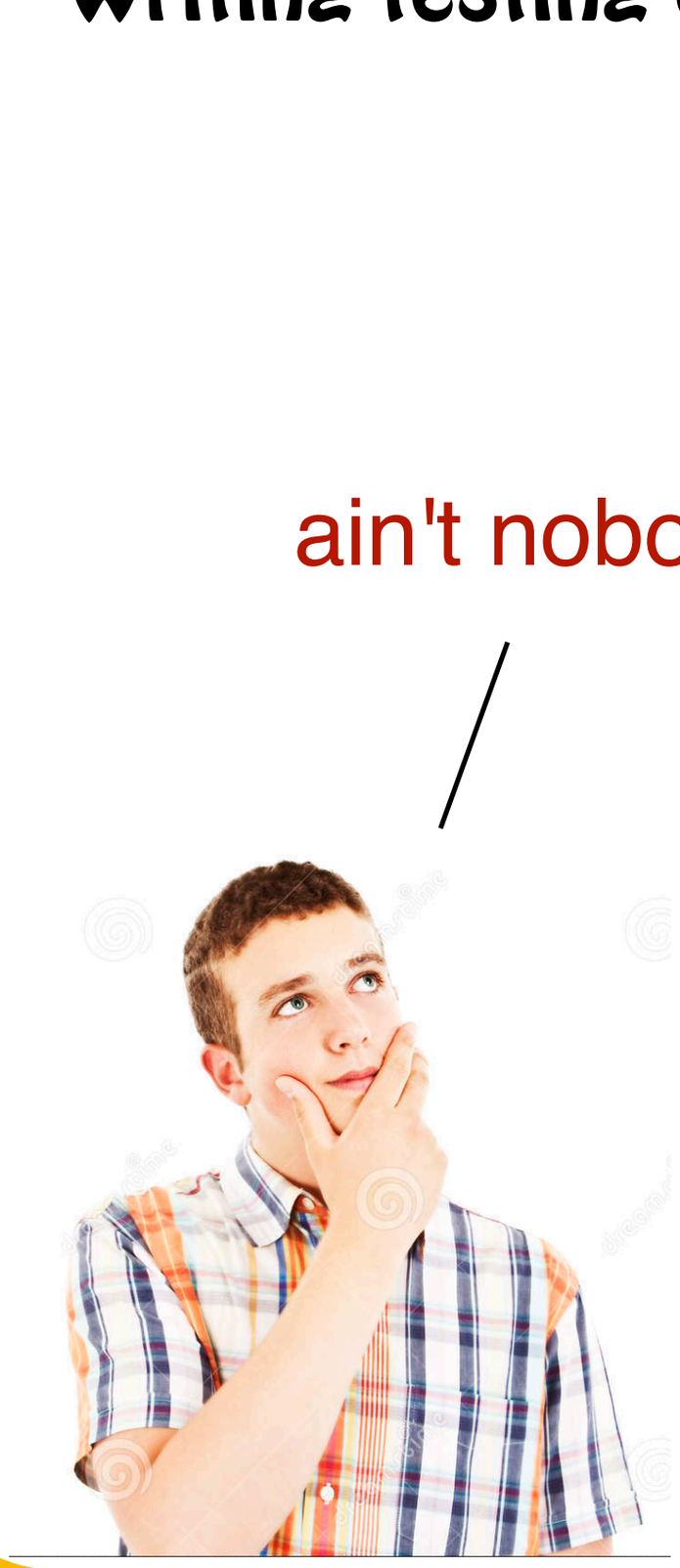
2 src/aido_schemas/__init__.py

```
@@ -1,4 +1,4 @@
- __version__ = "5.0.10"
+ __version__ = "5.0.11"


  import logging
```

Writing testing code

ain't nobody got time for that



Kids only learn that the stove is hot when they put their finger on and they burn it. This, unfortunately, is the limitation of our precious brain.

— Hasso Plattner —

Sometimes if you want to know for sure whether the stove is hot, the only way to find out is to touch it.

— Lee Child —

Bohr's definition of what is "an expert"

An expert is a person who has made all the mistakes that can be made in a very narrow field

Niels Bohr

Corollary: School is about making
as many mistakes as possible
as quickly as possible

...and remembering them with infinite regret.