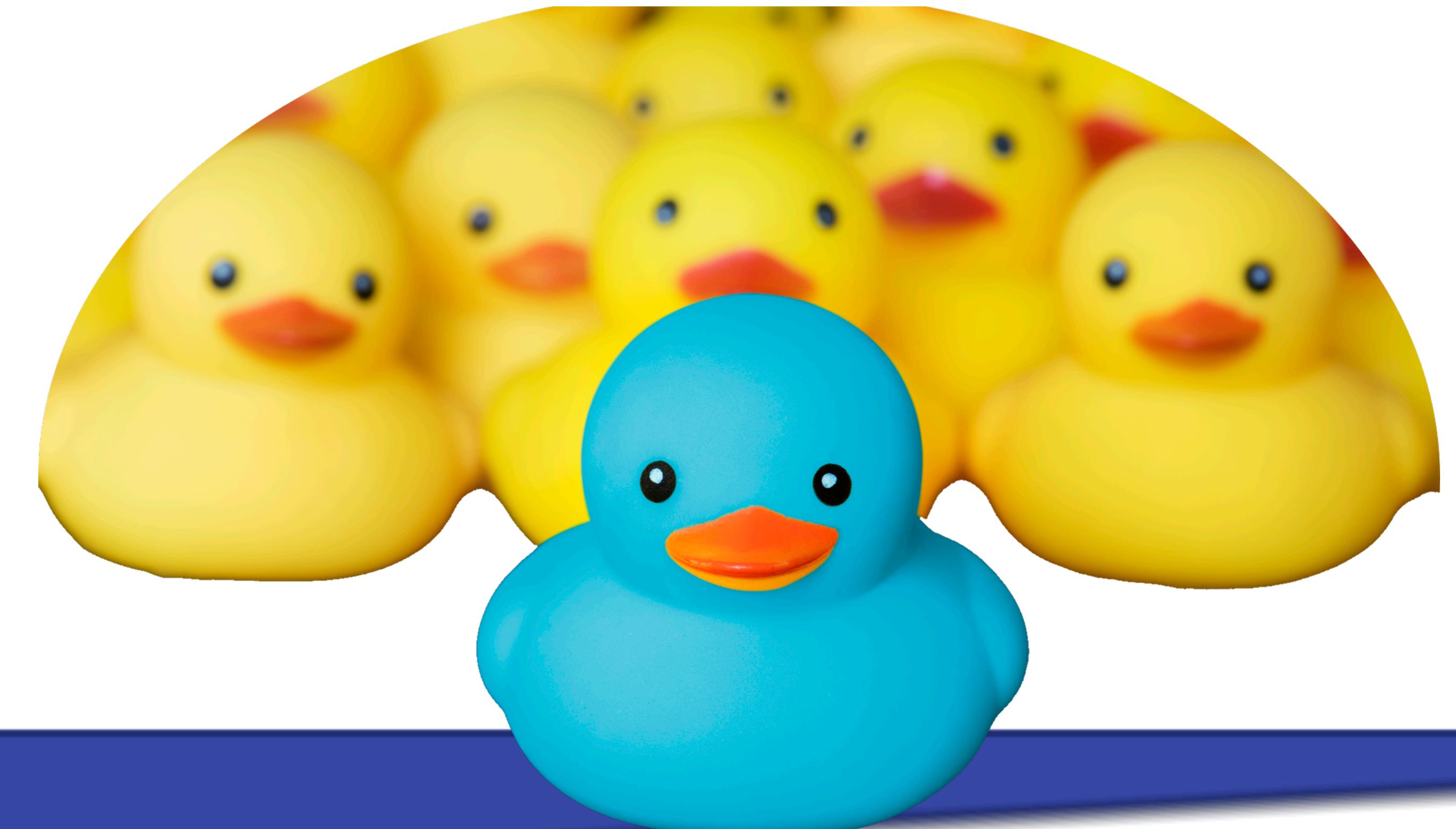


Tests that (Almost) Write Themselves

Hints for Golden Master Testing in Python



Stefan Baerisch, stefan@stbaer.com, 2020-07-20

About

Stefan Baerisch

stbaer.com (German only, sorry)

Software since 2005

Python since 2006

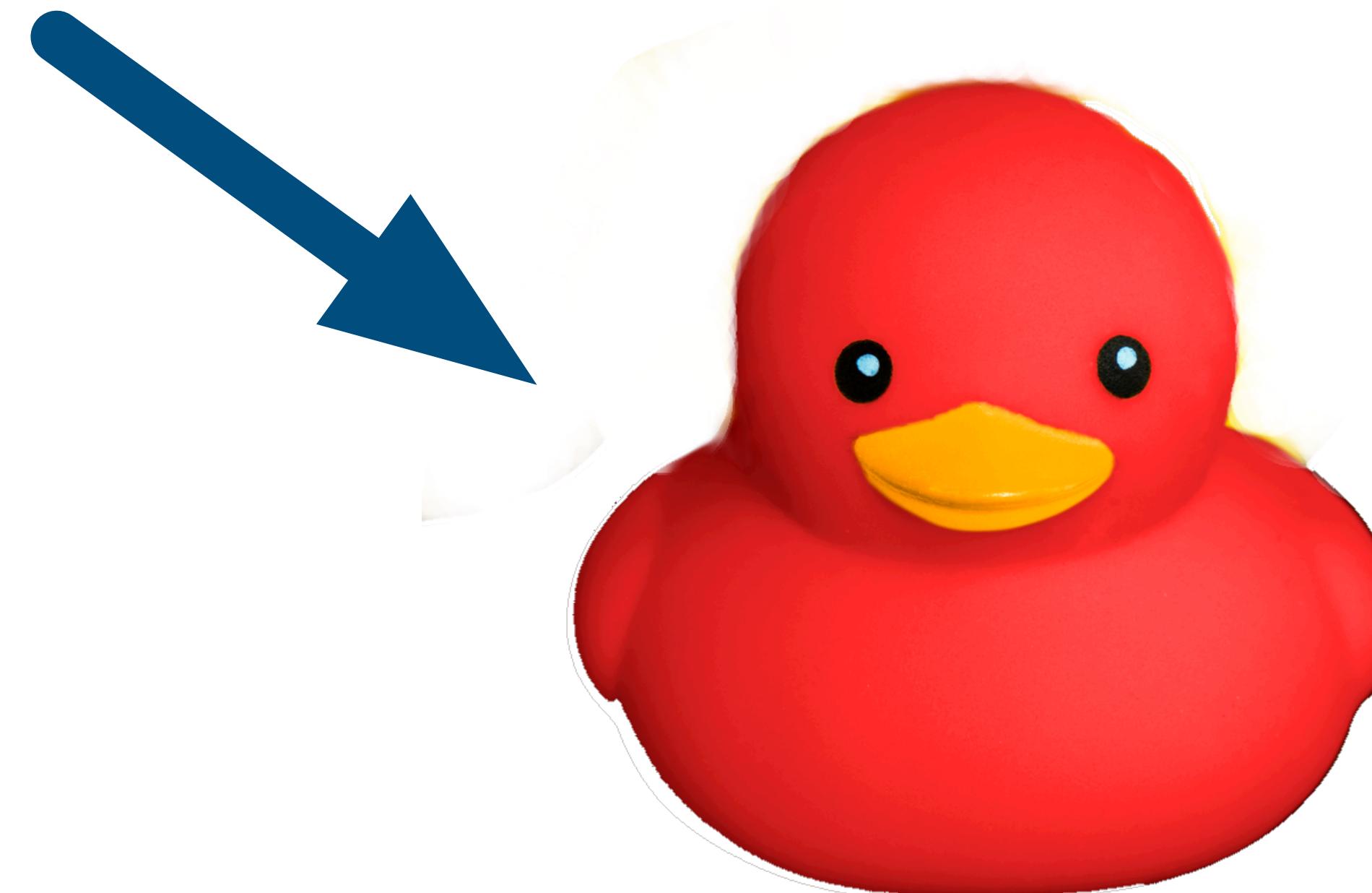
Project management / Test
Management since 2010



Some Background on Golden Master Tests

Testing Ducks

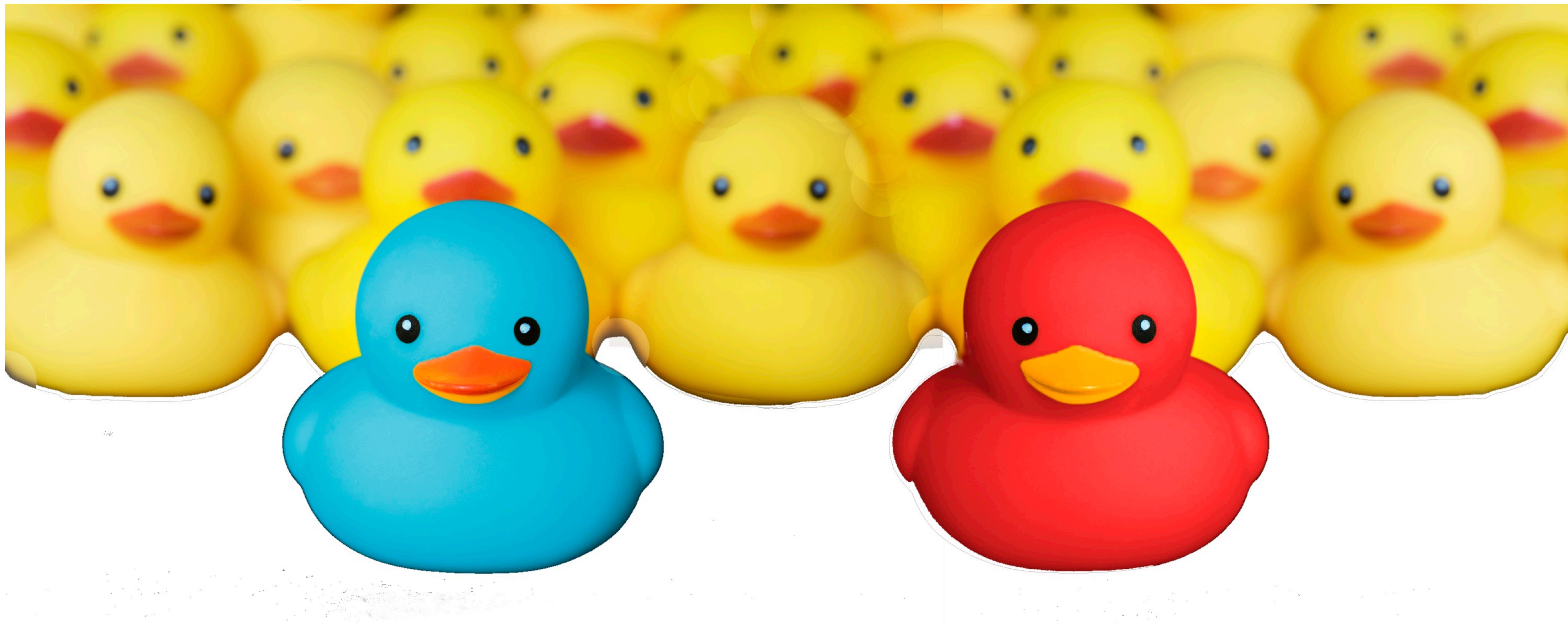
Good Duck?



How would you describe an 'acceptable' rubber duck ?

Do you know it when you see it?

Testing Ducks



How would you describe an ‘acceptable’ rubber duck ?
Do you know it when you see it?

The Idea behind Golden Master Testing

```
▼<body>
  ▼<div class="related" role="navigation" aria-label="related navigation">
    <h3>Navigation</h3>
    ▼<ul>
      ▶<li class="right" style="margin-right: 10px">...</li>
      ▶<li class="right">...</li>
      ▶<li class="right">...</li>
      ▷<li class="right">
        <a href="#">re.html title="re – Regular expression operations" accesskey="P">previous</a>
        "
      </li>
    ▷<li>
      <: mi
    </li>
    ▷<li>
    ▷<li>
      ▷<li class="nav-item nav-item-1">
        <a href="#">index.html
```

```
debug3: sign_and_send_pubkey: signing using rsa-sha2-512
debug3: send packet: type 50
debug3: receive packet: type 52
debug1: Authentication succeeded (publickey).
Authenticated to eviact.com ([142.93.108.97]:22).
debug1: channel 0: new [client-session]
debug3: ssh_session2_open: channel_new: 0
debug2: channel 0: send open
debug3: send packet: type 90
debug1: Requesting no-more-sessions@openssh.com
debug3: send packet: type 80
.
.
.
```

How to implement test if data is unknown or too complex for an assert() ?

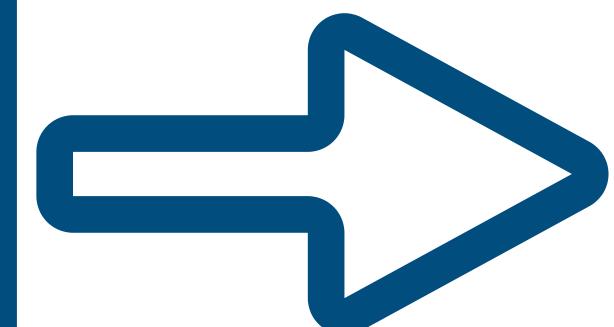
```
debug1: sending packet: type 40
debug1: Sending environment.
debug3: Ignored env NVM_INC
debug3: Ignored env TERM_PROGRAM
debug3: Ignored env NVM_CD_FLAGS
debug3: Ignored env DOTNET_CLI_TELEMETRY_OPTOUT
debug3: Ignored env SHELL
debug3: Ignored env TERM
debug3: Ignored env CLICOLOR
debug3: Ignored env TMPDIR
debug3: Ignored env Apple_PubSub_Socket_Render
debug3: Ignored env TERM_PROGRAM_VERSION
debug3: Ignored env TERM_SESSION_ID
```

```
openssh.com want_reply 0
lback start
```

How does this help?

Scenario

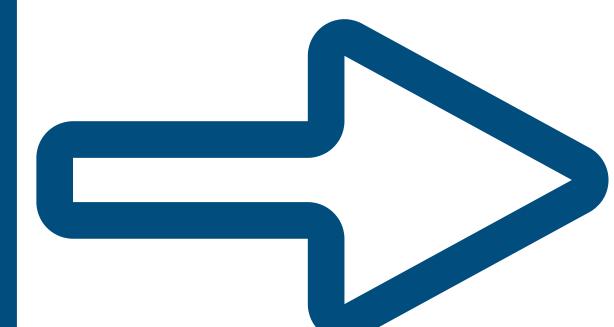
Testing a legacy system



Action

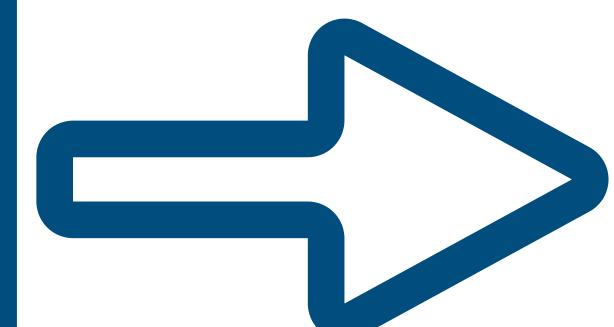
Ensure that changes are visible

Testing complex data



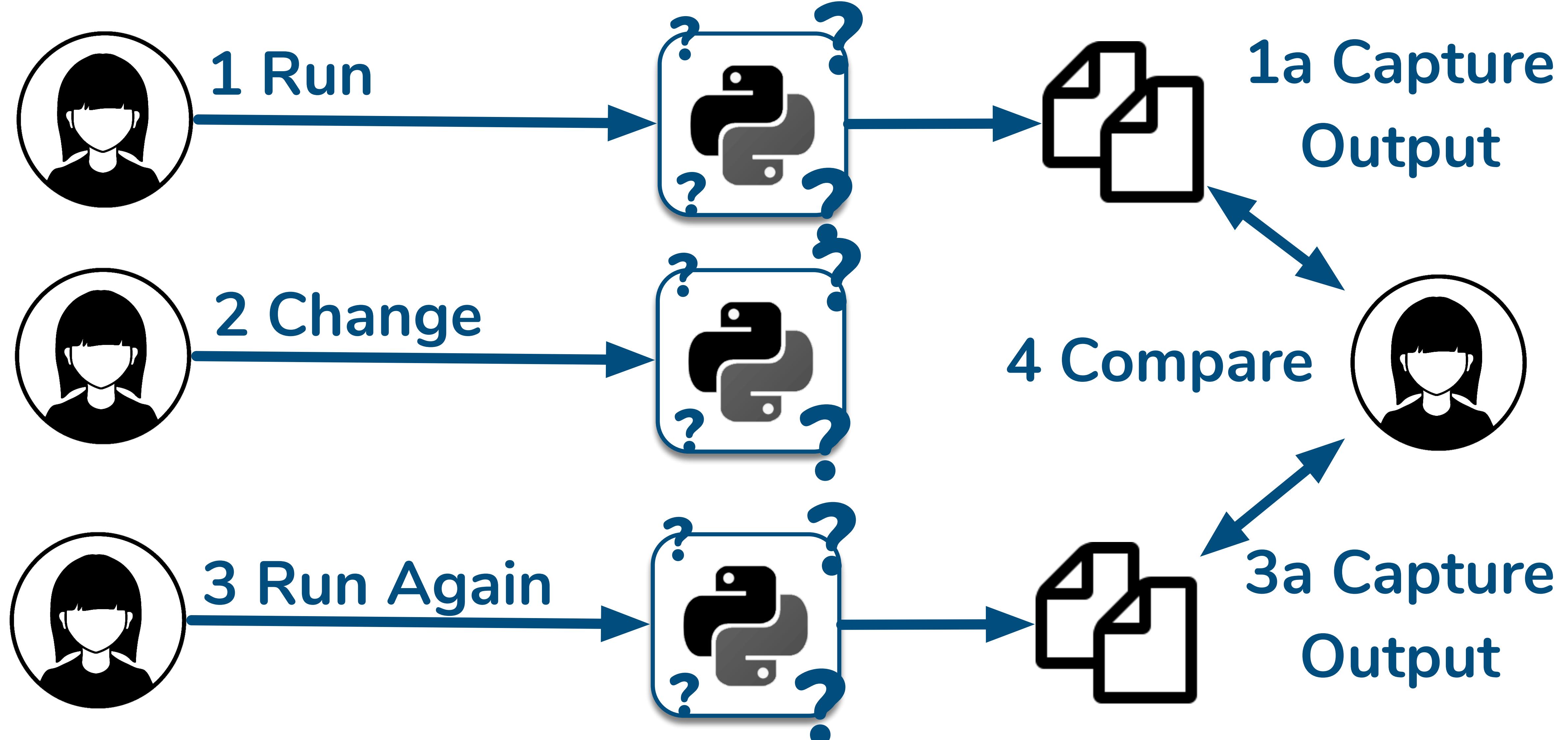
Identify and review changes

Testing complex data with some changes



Select or filter data used for tests

What is Golden Master Testing



Golden Master Testing - An Overview

Golden Master Tests are also known as:

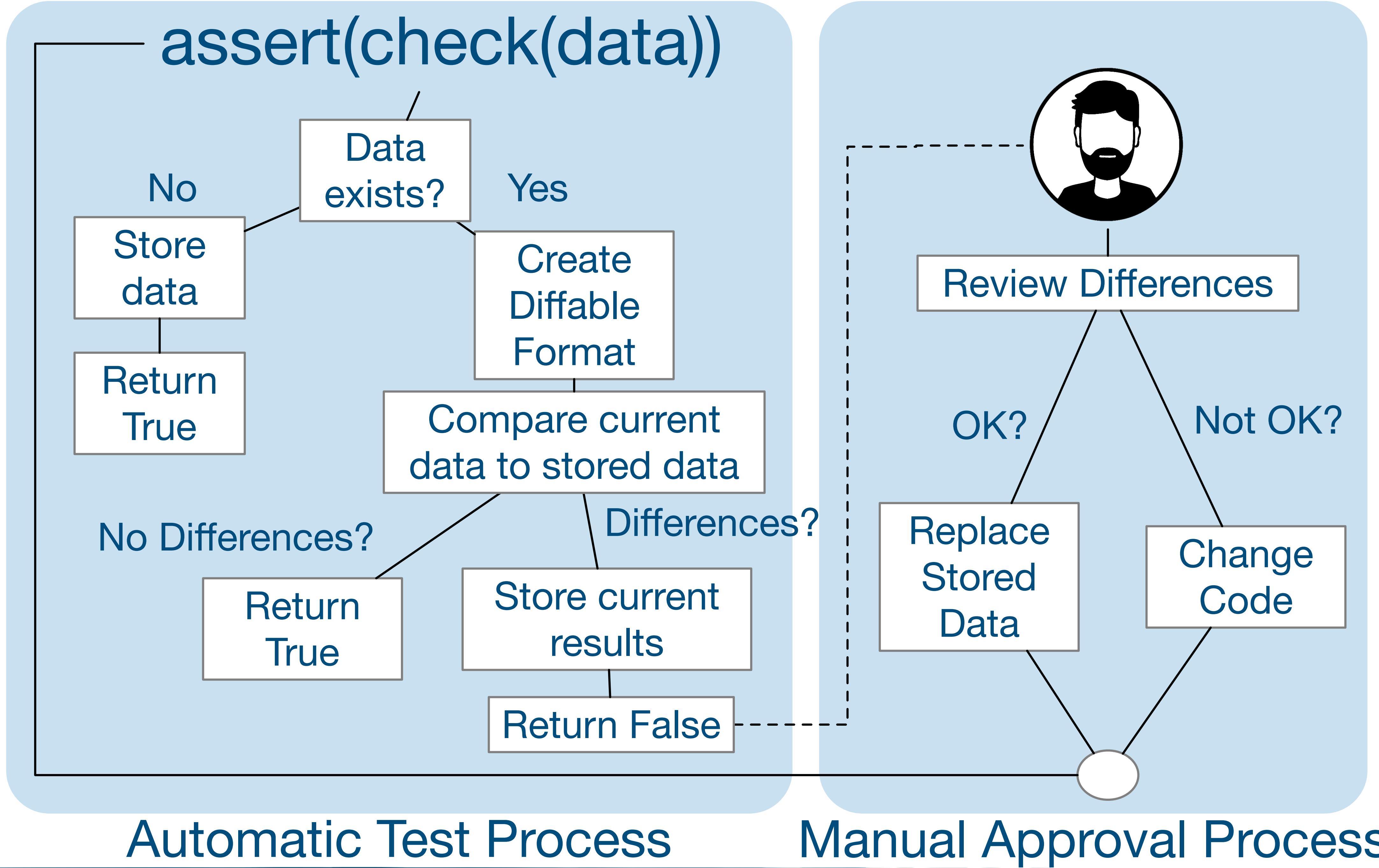
- Characterization Tests
- Approval Tests
- Snapshot Tests

Because changes to the data are ‘approved’

In the Javascript World,
especially Jest

Implementing Golden Master Tests

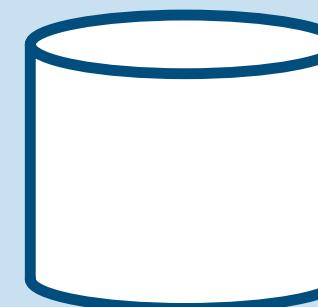
The Process



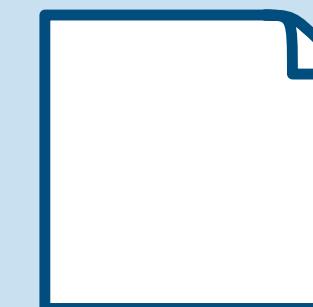
Design Decisions

Test Target

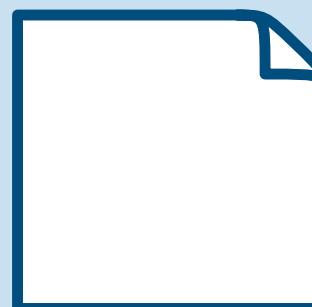
State



Log

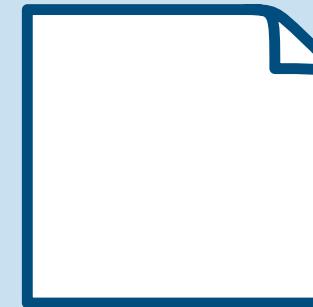


Output

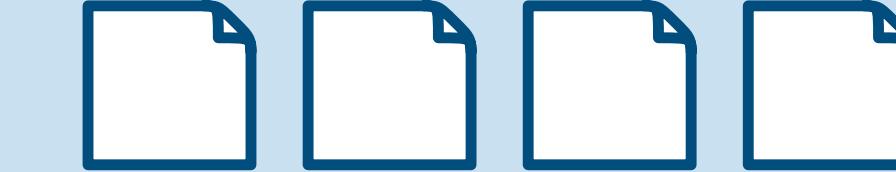


Test Size

Few Large

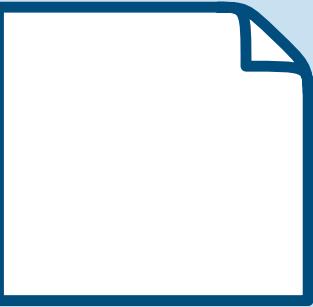


Many Small

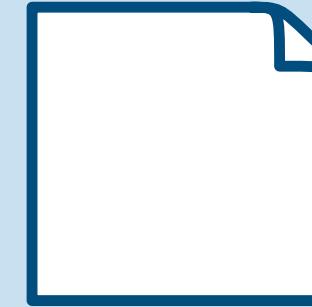


Storage Formats

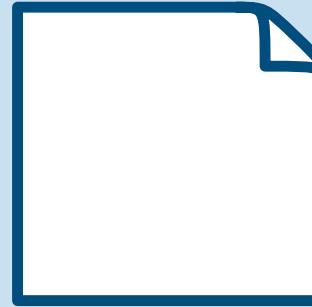
Text



Json/XML

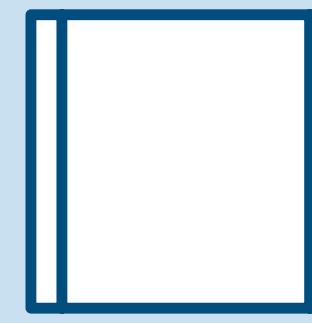


Custom

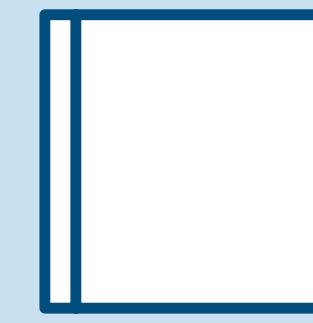


Diff Approach

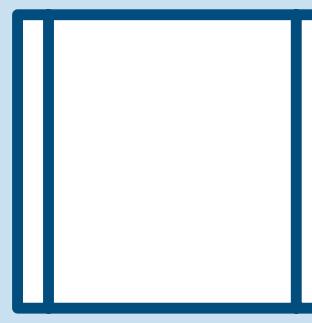
CLI \$...



Lib



Custom



What to Ignore

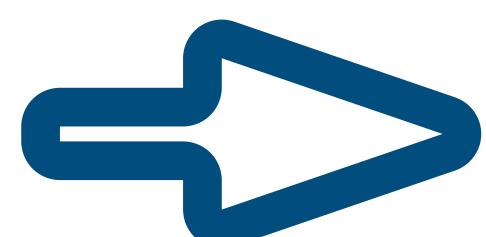
Key Order

Dates

Random Values

Test Target

- Text / Logfiles → Use as is, with Filtering
- Databases → Select and Store as DDL
- JSON → import json
- Objects → custom implementation or...
import jsonpickle or...
import deepdiff
- Images → pillow
- Dataframes → json



Keep it simple. Python makes it quite easy to get a text presentation of data

What to ignore

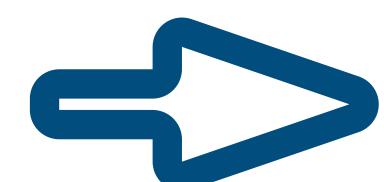
What to ignore?

Dates / timestamps

Key order for non-ordered dicts

Random data (object ids / sequence ids)

Non-relevant data (non significant floating point data)



Needs some pre-processing, depending on use case

Test Size



Storage Formats

How do you store results from past runs?

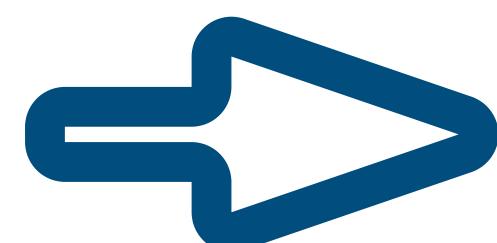
The format should be:

Easy to store in VCSs

Difflable in Python and External Tools

Supported by Editors / Views

Should be structured to support working with Python



If possible, use normalized JSON with linebreaks

An Example Implementation

Design Goals

Keep things simple

Compare Python Objects, — jsonpickle
no extentions

Simple comparision — difflib.unified_diff

Simple storage — store files with tests, explicit naming of tests

Simple operations — check() - store or compare data

list() - show stored data and conflicts

review(name) - show differences

approve(name) - mark the current version as ok

Our Sample Class

```
class MyTestClass:  
    def __init__(self):  
        self.a = "Hello"  
        self.b = 1  
        self.c = 2.33433  
        self.d = [1, 2.3333, 3.3332]  
        self.e = "2020-05-12"  
  
test_data = MyTestClass()  
changed_test_data = MyTestClass()  
changed_test_data.d.pop()  
changed_test_data.c += 0.5
```

Overview

```
checker = checker.Checker(os.path.dirname(__file__)) ← Set storage location
```

```
assert checker.check(test_data, "name1") Save  
assert checker.check(test_data, "name1") Compare ← Success
```

```
assert checker.check(test_data, "name2") Save  
assert checker.check(changed_test_data, "name2") Compare ← Failure
```

```
checker.list() ← Show state of tests / stored data
```

```
checker.review("name2") ← Show diff of "name2" test
```

```
checker.approve("name2") ← Accept the lastest version
```

Overview Results

checker.list()

```
Name: name1 Conflict: False  
Name: name2 Conflict: True  
@@ -1,10 +1,11 @@
```

checker.review("name2") →

```
{  
    "a": "Hello",  
    "b": 1,  
    - "c": 2.83433,  
    + "c": 2.33433,  
    "d": [  
        1,  
        - 2.3333  
        + 2.3333,  
        + 3.3332  
    ],  
    "e": "2020-05-12"  
}
```

checker.approve("name2") →

```
name2.now ⇒ name2.last
```

Implementation - Check

```
def check(self, obj, name):
    jsonpickle.set_preferred_backend('json')
    jsonpickle.set_encoder_options('json', sort_keys=True, indent=4)
    now_text = jsonpickle.encode(obj, unpicklable=False)

    last_filename = self._get_filename(name, LAST)
    if os.path.exists(last_filename):
        last_text = open(last_filename, mode="r", encoding="utf-8").read()
        diff = difflib.unified_diff(now_text.split("\n"), last_text.split("\n"))
        if len(list(diff)) != 0:
            now_filename = self._get_filename(name, NOW)
            self._write_file(now_filename, now_text)
            return False
        else:
            return True
    else:
        self._write_file(last_filename, now_text)
    return True
```

Implementation - List Status

```
def list(self):
    for val in self._get_list():
        print(f"Name: {val['name']} Conflict: {val['conflict']}")

def _get_list(self):
    result = []
    cands = glob.glob(os.path.join(self.path, "*.last"))
    for cand in cands:
        name = os.path.basename(cand).replace(".last", "").lower()
        conflict = os.path.exists(cand.replace(".last", ".now"))
        result.append({"name": name, "conflict": conflict, "filepath": cand})
    return result
```

Implementation - Store

```
def review(self, name=None):
    read = lambda fn: open(fn, mode="r", encoding="utf-8").read().split("\n")
    for cand in self._get_entries(name, True):
        last_name = cand['filepath']
        last_cont = read(last_name)
        now_name = cand['filepath'].replace(".last", ".now")
        now_cont = read(now_name)
        diff = list(difflib.unified_diff(last_cont, now_cont,
                                         tofile=now_name, fromfile=last_name))
        print("\n".join(diff[2:]))

def _get_entries(self, name, only_conflict=False):
    cands = self._get_list()
    if name is not None:
        cands = [c for c in cands if c['name'] == name.lower()
                 and (not only_conflict or c['conflict'])]
    return cands
```

Implementation - Diff

```
def approve(self, name=None):
    for cand in self._get_entries(name, True):
        last = cand['filepath']
        now = last.replace(".last", ".now")
        print(f"{os.path.basename(now)} => {os.path.basename(last)}")
        shutil.move(now, last)
```

Other Existing Libraries

github.com/syrusakbary/snapshottest

Inspired by Javascript's Jess. Nice
Integration with unittest/nose/pytest

github.com/approvals/ApprovalTests.Python

Python implementation of approval
testing,

Summary

Why

Golden Master Tests work by **capturing & comparing** results of program executions. It helps with complex data, especially when we want to monitor for changes.



How

We store results and compare between runs. On differences, we **review & approve** the results

Python

Python has many modules to help us. Some existing implementations exist if we don't want to implement it ourselves



Thank you!

Stefan Baerisch, stefan@stbaer.com, 2020-04-07