

1

Testing in Layers

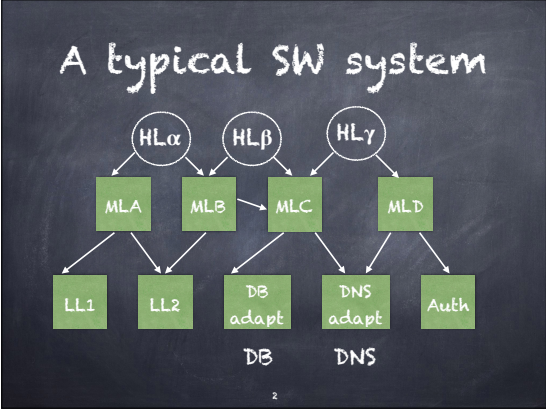
[http://www.aleax.it/pyconit17\\_en.pdf](http://www.aleax.it/pyconit17_en.pdf)



©2017 Google -- aleax@google.com

1

2



3

Why test?

- if you need me to explain this...
- ...then you need my OTHER talks on testing!-) [other people's too]
- i.e, I'm not covering that today!-)

3

## How do we test it?

- ⦿ ancient way: white-box, black-box
- ⦿ too-close modern way...:
  - ⦿ unit-tests: white-boxy, dev-focused
  - ⦿ integration-tests: end-to-end
  - ⦿ maybe: human-in-loop QA

4

QA = Quality Assurance (use a different term than "test", it's TOTALLY different!!!)

## Best way: Layers

- ⦿ unit-tests (fast, run all the time)
  - ⦿ strictly on internal logic
  - ⦿ mock out "every" dependency
  - ⦿ fast above all (as they run all the time)
- ⦿ higher-layer tests
- ⦿ pattern language: match the CODE layers!

5

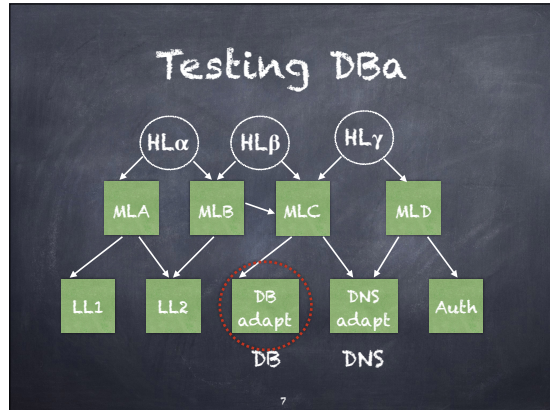
If running automatically in the background, fast allows lower latency for feedback;  
if not, fast means you'll run tests more often; either way -> higher productivity

## The fundamental things apply... e.g:

- ⦿ all tests must be reproducible
  - ⦿ if any randomness, force a seed
  - ⦿ if depends on current time of day, day of week, etc etc, must fake time
- ⦿ test-first approach to fixing bugs

6

...and many other excellent, necessary qualities of automated tests, layered or not.



7

- ### How to test DBa
- pure unit-test: mock out DB
    - fine if we understand DB 100%
  - 2nd layer int-test: emulated DB
    - local, controlled, maybe in-mem
    - including semantic constraints!

8

DB = database

- ### What's a Semantic Constraint?
- e.g: "after conn.close() no other call is allowed, RuntimeError raised"
    - a fake must emulate this behavior
    - ...a mock will not unless you already KNOW all about it (still worth it for maintainers)

9

A fake may also add constraints such as "DB size < 23 MB" — for test uses, should be OK.

```
from unittest import mock # also in later slides

# for pure unit-test:
with mock.patch.object(dba, 'db', autospec=True) as fdb:
    # prepare fdb's side effects under test
    fdb.connect.cursor.side_effect = ...

    # body of tests

# for 2nd layer integration-test:
fdb = fake_db.Fake(...params...)
with mock.patch.object(dba, 'db', new=fdb):
    # populate fdb for the test
    fdb.connect().cursor().execute(...)

    # body of tests

# for full integration-test:
# start and populate real db for the test
# body of tests
```

10

10

## "Body of tests"

- core reusable part
  - exercises all relevant paths
  - including "simulated" exceptions
- optionally followed by (for mocks):
  - check of calls, arguments, ...

11

11

"Body of tests" constant and reusable; difference among test layers is in the preparation for running the "body of tests" (optionally also in extra checks afterwards for mocks).

## Mocks aren't Fakes

- ...nor other kinds of test doubles (see <https://martinfowler.com/articles/mocksArentStubs.html>)
- dummy, fake, stub, spy, mock
- key issue: who owns/maintains

12

12



## Mocks vs Fakes

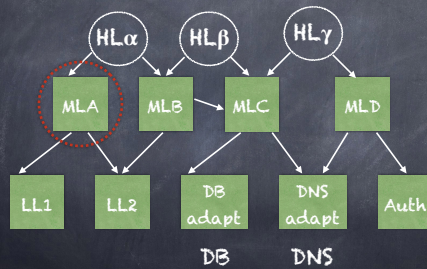
- mock: flexibly emulates anything
  - depending on tests' calls on it
  - lets you check calls, args, ...
- fake: fast, limited simulation
  - of a specific component/module

13

13

Both should also, on request, emulate error situations (e.g. "CPU on fire":-) to check your code handles such disasters gracefully (almost impossible to check w/o simulation!).

## Testing MLA



14

14

## How to test MLA

- pure unit-test: mock out LL1, LL2
  - fastest, mostly fine (team ownership)
- 2nd layer int-test: actual LL1, LL2
  - if fast enough (check w/timeit)...
    - don't need pure unit-test (less work)

15

15

If uncertain, try both ways — timeit!

```

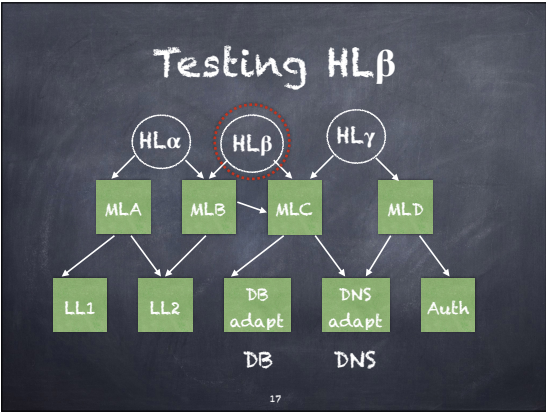
# for pure unit-test:
with mock.patch.object(mla, 'll1', autospec=True) as f1, \
    mock.patch.object(mla, 'll2', autospec=True) as f2:
    # prepare f1's & f2's side effects under test
    # body of tests

# for 2nd layer integration-test:
# prepare ll1's and ll2's for the test
# body of tests

# no further integration-tests in this specific case

```

16



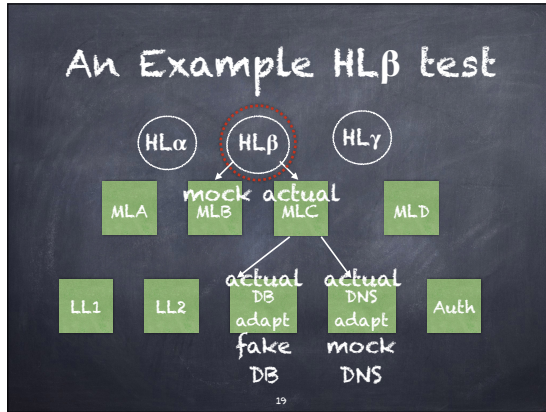
17

- ### How to test HLβ
- ◉ pure unit-test: mock out MLB, MLC
  - ◉ 2nd layer int-tests: actual MLB, MLC
    - ◉ mock DBa, DNSa; mock or real LL2
  - ◉ 3rd layer i-t: act. ML\*, LL2, DBa, DNSa
    - ◉ mock or fake DB, DNS
  - ◉ (pick subset, else, combinatorial explosion!)

18

tradeoffs: mocks may be faster; mostly: ownership of test-double, thus, how much detailed/precise understanding of corner cases is needed

19



20

```

# code for the example:
with mock.patch.object(hbb, 'mlb', autospec=True) as fb:
    # prepare fb's side effects under test
    ...
    fdb = fake_db.Fake(...params...)
    with mock.patch.object(dba, 'db', new=fdb):
        # populate fdb for the test
        fdb.connect().cursor().execute(...)
        with mock.patch.object(dnsa, 'dns', autospec=True) as fd:
            # prepare fd's side effects under test
            ...
            # body of tests
  
```

The code block shows a sequence of mock patches and test preparation steps. The line '# body of tests' is highlighted with a red box. The number 20 is written at the bottom of the code block.

21

### Use mock, fake, or actual module?

- mock: fastest, least accurate
- actual: least work, if fast enough
  - design it to be primeable for speed
- fake: best if there (thorough, deep, fast)
  - coding a good fake is a lot of work

The number 21 is written at the bottom of the text block.

## Check complexity

- e.g., external "DNS" module – what DNS records?
- often: just A records, DN → IP
  - trivial to mock or fake
- or: CNAME, HINFO, MX, NS, PTR, SOA, TXT, ...
- needs careful fake (mock VERY hard to make and keep correct and complete!)

22

22

DNS = Domain Naming System; once mostly a simple "my.host.com" - > 22.33.44.55 mapping, now (over?)grown into much richer functionality (e.g., TXT records to validate ownership)

## Load-test in layers

- actual elapsed-time measurements need end-to-end code paths
- BUT: with intermediate tests you can get (t) time in your code plus (n) number of calls to external systems
- ...and can compute worst case total time as:  $t + n * (\text{ext.sys.'s SLAs})$

23

23

SLA = Service Level Agreement (e.g.: "90% of queries answered in < 33 msec")

## "body of tests" for load testing

- not the same as for other tests
- rather: take correctness for granted;
  - exercise perf-critical paths
- usually best to separate for easier elapsed-time measurement

24

24



## Test refactorings

- within module: all talk applies (keep coverage; edit mocks or fakes ditto)
- moving functionality between modules:
  - at first, unit-tests must fail
  - edit tests, mocks/fakes (check pass!)
  - run interm. int-tests of higher levels
    - versions with actual lower levels!

25

25

In a sense, it's automatically a test-driven situation (since tests must exist before refactoring).

## Tests & Logging

- unit-tests must be fast
- check only what's checkable fast
- for everything else:
  - log/snapshot status in detail
  - later, run batch/b.background jobs to check
- batch sanity checks on logs/snaps: good idea
- including non-testing production runs!

26

26



[http://www.aleax.it/pyconit17\\_en.pdf](http://www.aleax.it/pyconit17_en.pdf)

27