Automated Testing:
End to End

I SEE YOU TEST YOUR CODE IN PRODUCTION
I TOO LIKE TO LIVE DANGEROUSLY
quickmeme.com

Q & A
at any time
good discussions

Intro to Software Testing

- Why Automated Testing?
- Manual vs Automated Testing
- Types of Automated Tests

Why are people not motivated to do automated testing?

- Code is not testable
- Takes a lot of time (bad code, bad tools)
- Hard to setup (no fixtures, no good tools)
- Learning curve (new frameworks/tools)
- Hard to maintain as the code changes
- Hard to reach a stable always-passing test suite

Why Automated Testing?

- Quality

  happy users + happy business + happy developers = successful business

- Reduced Stress

  no firefighting in production, no escalations

- Reduced Costs

  find bugs in development not in production

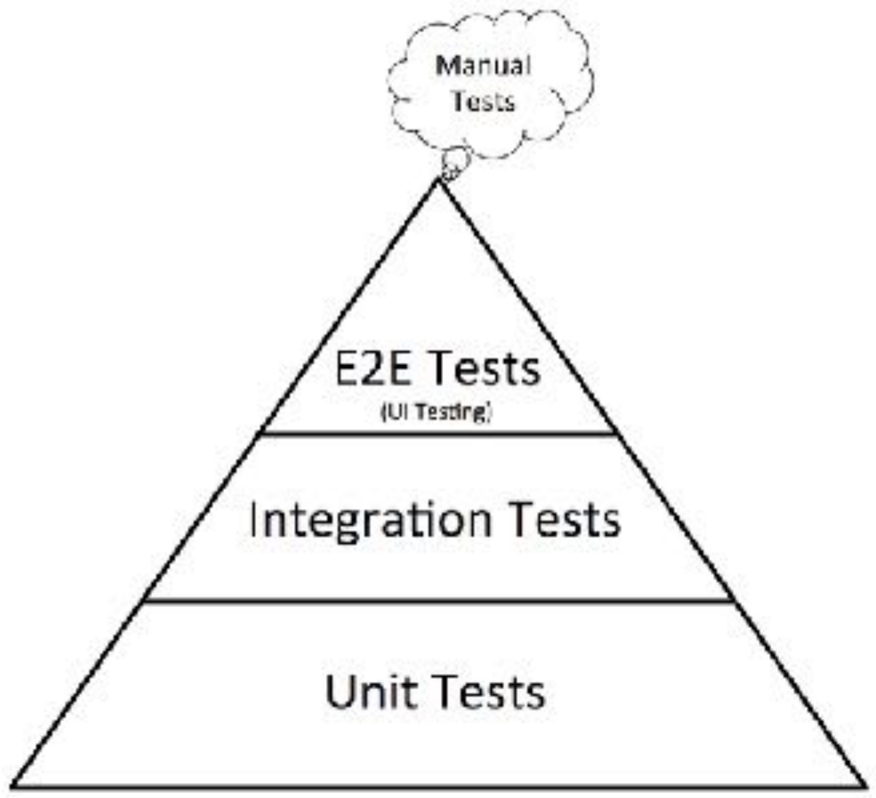
- Documentation

  your test descriptions = your only real up-to-date documentation (use cases, edge cases)

Automated vs Manual Tests

- Manual tests are good for system edges, UI, UX
- Manual testing cannot scale as software grows
- Automated tests are free
- Automated tests are fast
- Automated tests can be run at any time
- No tests vs bad tests

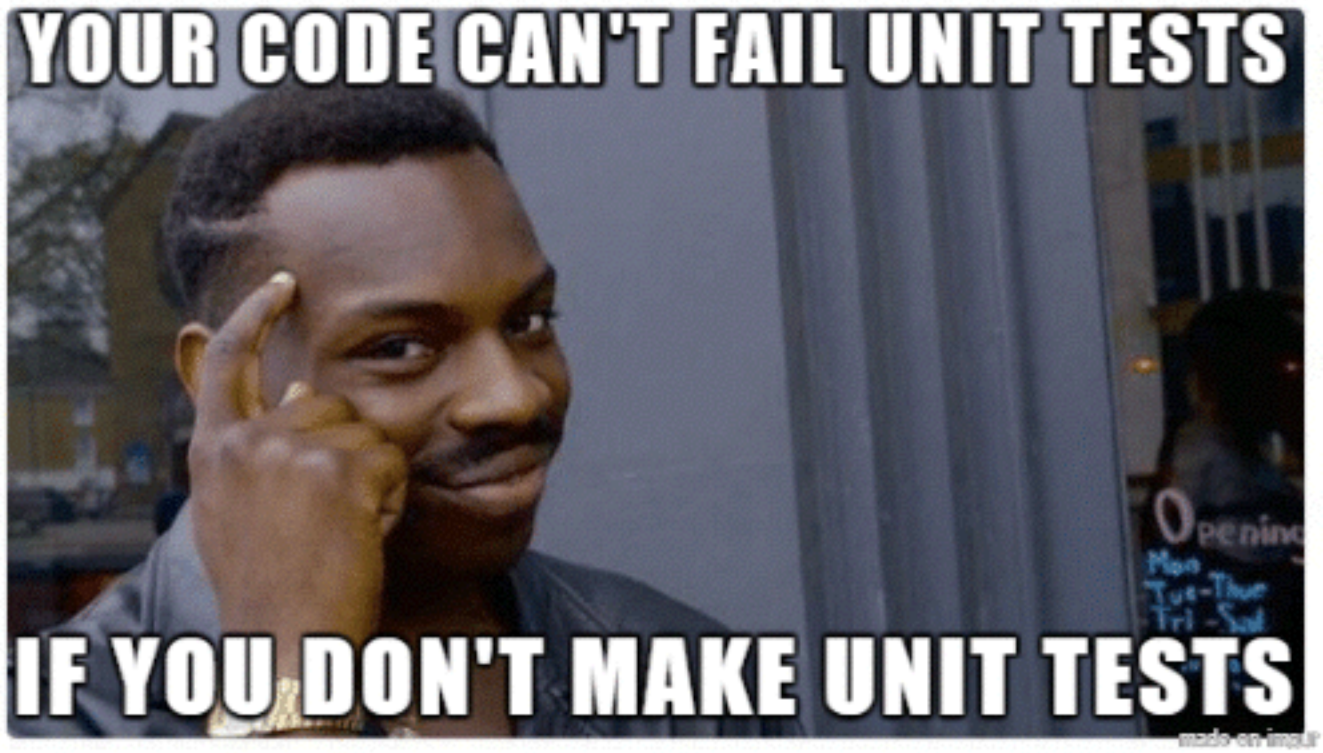
Types of Automated Tests



- End-to-end (UI): test from user perspective (slow, brittle)
- Integration: test that things work well together
- Unit: test things in isolation (fast, stable)

Unit Testing

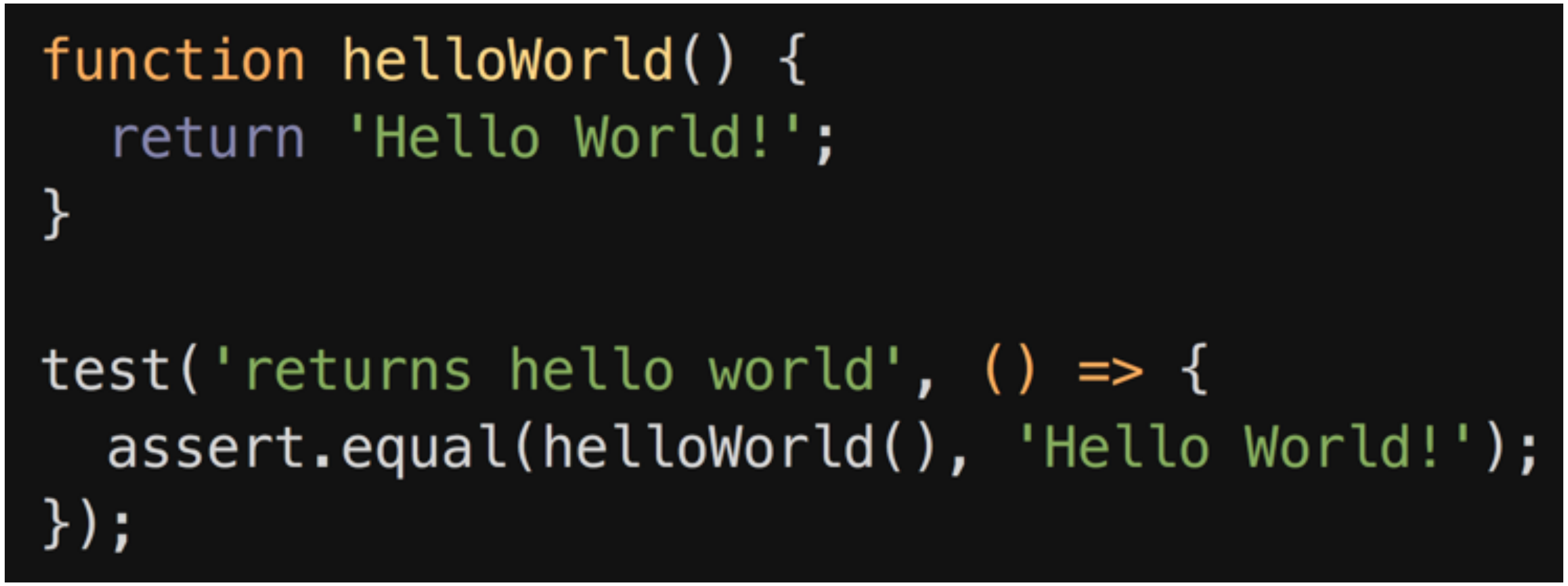
Automated Testing in Isolation



Unit what?

What is Unit Testing?

- Test the behaviour of one module/class/method
- Test only the public API of an object
- Stub all collaborators



```
function helloWorld() {
  return 'Hello World!';
}

test('returns hello world', () => {
  assert.equal(helloWorld(), 'Hello World!');
});
```

Conceptual Phases: AAA

- Arrange
- Act
- Assert

Stubbing: Test Doubles

- Stub: a test double returning a pre-defined answer
- Mock: a test double asserting interaction
- Fake: a test double with fake public API

Good Practices

- Test all logical paths
- Test all edge cases (boundary values)
- Test a single behaviour (one test, one assert)
- Test code should be production grade
- Write a new test for each bug found

Bad Practices

- Big test = bad code
- Test without stubbing collaborators
- Test multiple behaviours (multiple asserts)
- Access to external resources (file system, database, network)
- Sharing state between tests
- Test execution depends on order
- Not repeatable/predictable (depend on time, state)

Integration Testing

What is Integration Testing?

- Testing things work together
- Testing communication between parts of the system
- Testing a logical subsystem/component
- Work with external resources (file system, database)

Why Test in Integration?

- Catch misunderstanding in object interactions
- Test logical subsystem/service/component
- Test interface contracts
- Avoid regressions in component integration (where unit testing fails)

Good Practices

- Think about value vs cost: maintain a small suit
- Production grade code
- Fixtures (prepare database state)
- Write a test for each bug found

Bad Practices

- Testing edge/error cases
- Testing low level behaviour
- Using service layer calls instead of fixtures
- Maintaining a large suite

Levels of Integration Testing

- Test internal components
- Test interaction with external components
- Test internal logical subsystem
- Test application service layer vertically

Functional UI Testing

What is Functional Testing?

- Testing things at the user interface level
- Verify system behaviour at highest level
- Automated Manual Testing (if you wish)

Why Functional Testing?

- Test the whole system as if the user is using it
- Test a complete vertical slice
- Test front-end is wired-up correctly to the backend
- Test UI centric concerns

Good Practices

- Keep the test number to a bare minimum
- One test per feature
- DRY: Page Object Model (POM)
- DRY: Logical Functional Model (LFM)

Bad Practices

- Testing the whole system from the UI
- Testing design cosmetics
- Assertions tightly-coupled with HTML/CSS code
- Using service layer to setup database state

UI Automation Tools

- record & playback
- coded

Unit Testing JavaScript with Jest

What is Jest?

- Jasmine-based unit testing framework by Facebook
- Video: https://youtu.be/HAuXJVI_bUs
- Slides: https://github.com/rogeliog/jest-snapshot-talk

Why Jest?

- Popularity
- Easy setup
- Instant feedback
- Snapshot testing
- Fast
- Build-in code coverage (Istanbul)
- Powerful mocking library (automated mocking)
- Build for React but works with TypeScript (Angular2)
- Used by major players (Facebook, Oculus, Instagram, Twitter, Pinterest)
- No need for headless browser
- Works well with other libraries (Enzyme, Mocha, Chai)
- Jasmine-like API

Setup Jest

```
yarn add -D jest
```

sum() Example

Automatic Mocking

- Mocks all *require* statements
- jQuery, Underscore not mocked
- *jest.dontMock* the SUT
- can be toggled on/off

Why snapshots?

- Drop the manual work
- Quickly adapt to changes in code
- Easy to maintain
- Delegate matching work to Jest

Q & A