

Gavin Mogan

Code Monkey @ Sauce Labs

@halkeye

<http://www.gavinmogan.com>



Hi

I'm Gavin and I work at one of tonight's sponsors, Sauce Labs.

Why not testing?



WHY NOT TESTING???

Okay I just wanted an excuse to use zoidberg and futurama gifs

But first a little background about me.

First Job

- Low pay
- Develop Live
- No source control
- No testing



Gross

After dropping out of bcit, I pretty much took the first dev job I could find.

It was everyone's worst nightmare. Low pay, coding live on the production server, no source control, and while I didn't realize it was a big deal at the time, no testing.

So many files and folders ending in dot bak.

gavin.pl

```
use Inhouse::Library;
use JSON;

$user = Inhouse::Library::createUser({
    username => "zoidberg",
    password => "doctor"
});

print JSON::to_json($user);
```

gavin.js

```
const Users = require('./models/users.js');

Users.createUser({
    username => "zoidberg",
    password => "doctor"
}).then(function(user) {
    console.log($user);
});
```

At the time, I got in the habit of creating gavin files to test things locally before attempting to deploy them.

Little did I know this would lead down a path of testing.

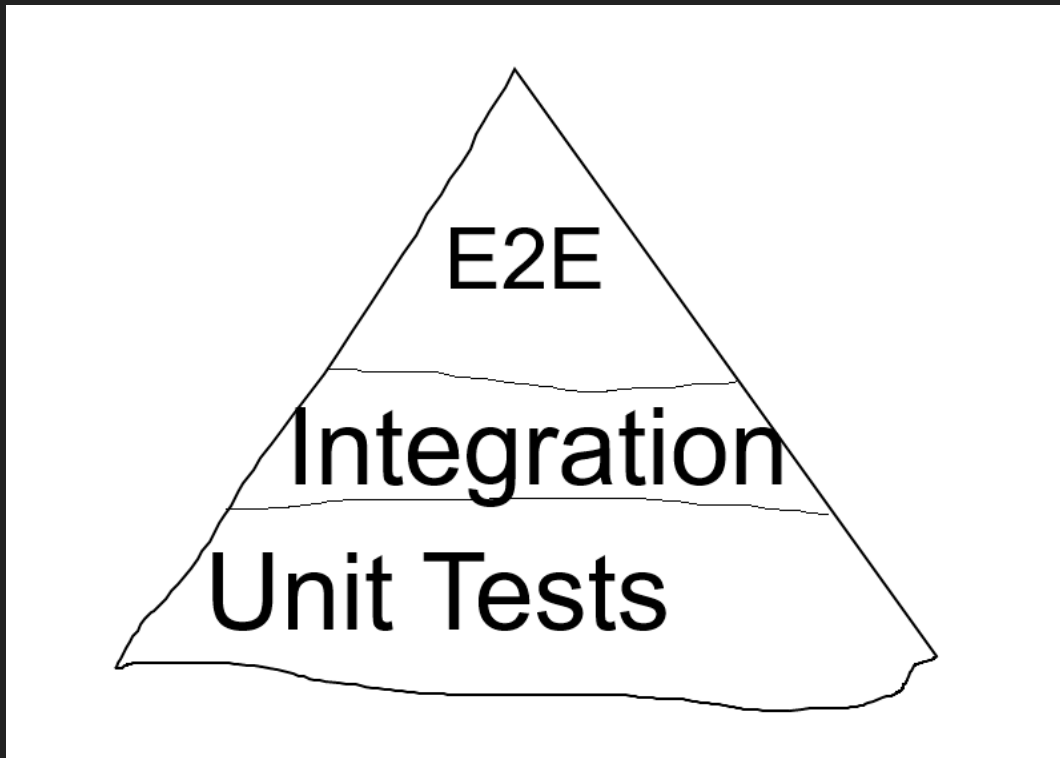
Gavin asks you



Okay.. Time for me to ask you a question.

Hands up if you develop on a daily basis?

Okay, keep your hands up if you actually write tests for your code.



So moving into a bit more formalized topic...

(Gavin draws the best)

Here we have the testing pyramid.

You can find many variations of the same thing but this one I lovingly created.

The idea is that you spend most of your time writing unit tests. They are discrete chunks of tests that confirm small pieces of code.

Next up you have integration tests. Those typically will reach out to other systems. They are slower. Take more time to setup.

Lastly you have the complete end to end tests. They are the slowest to run, so you don't want to do them all the time.

Sauce Labs actually specializes in this layer when you are doing web testing.

Indirect Pros

- Learning Codebases
 - New Hire can write tests
 - Tests are always up to date when compared to documentation
 - When/why is this code supposed to be used.

So why should you write tests?

When I was a developer at telus, I was quickly put on a legacy project and was told to get up to speed.

I was lucky enough to convince them to spend a couple weeks writing tests.

By the time I was finished, not only did I know all the models and db schema inside and out, but I actually found a few bugs and dead code.

"We need time to test"

- Often leads to features being shipped that have never been used
- Blame others (QA) when things don't get caught

Why not testing first?

- Helps you think about design

This is actually one of my biggest pet peeves. If you haven't written any tests, it's really not done yet. So time shouldn't be a factor. But I do understand time crunches and project managers.

- Lets you fail fast

I'm by no means perfect, I leave a bunch of testing till late in my development. But my development these days tends to span a day or two, so testing is still really early considering.

- Write failing test first, then code

- Refactoring later isn't scary

There are some ways to convince people though. There are tons of studies on how writing tests actually increases development speed. I feel like it has a lot to do with peace of mind. Don't need to be afraid of trying something because you have a safety net of tests.

Failing fast means you can try something without working on all the boiler plate around it.

Add function takes two numbers, 1 and 2, does it equal 3? Cool works, does it throw an exception? boo

Failing tests help prevent false positives. Testing javascript callbacks especially lead to this.

SENORGIF.COM



No, no reason, last few slides have just had a ton of text.

How do I run tests?

Javascript

- grunt/gulp/etc
- mocha (with watch)
- qunit
- many many more

Ruby

- guard
- rake
- rspec
- minitest
- unittest
- many many more

Python

- nose
- unittests
- many many more

IDEs

- IdeaJ
- Sublime/Atom

How do I run tests?

- many many more

The list of tooling and frameworks is ever growing bigger. I can't stress enough there's no right way of testing, just the way that works the perfect or best for you.

Generally there are two categories. Often frameworks will fall into both but not always.

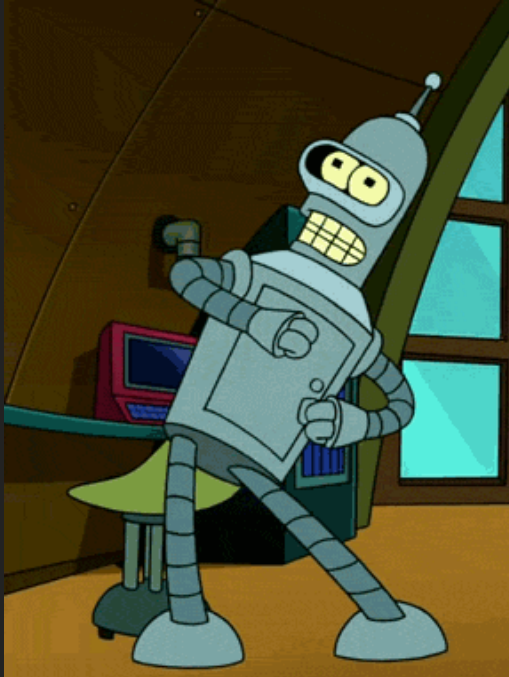
Grunt, gulp, and guard all specialize in running tasks, so they are great at re-running the entire test suite when files are changed. I personally have mine setup to beep at me when things fail so I don't have to keep an eye on it.

Mocha, qunit, rspec, and others are more the testing frameworks. They are how you write the tests. How you see the results, etc.

- ✓ should save article with title
- ✓ should load with the same title

2 passing (66ms)

No, more automated!



- Jenkins
- Travis-ci (.org)
- Circle CI
- Bamboo
- Team City
- Visual Studio Online (VSTS)
- more more more

Lots of people run automated tests. Sometimes referred to as continuous integration.

My personal favourite two are travis ci and jenkins.

Travis-ci is great for open source projects. Its 100% free, and you can do whatever you want.

Jenkins is better for when you have a large collection of projects.

Sauce labs, as well as myself for my own person projects, use a mixture of both.

Where do I start testing

- Convert simple scripts into tests
- Generally anything you care about
- API Return values
- Public methods
- Small discreet units



As I said earlier. Start by taking a script or segment you'd normally try out on its own, and make it into its own full test.

On my team at sauce for our rest apis. If its in a test, we are allow to use it in other components. It's considered a guaranteed return value.

I originally started with bugs. If I had a bug report, I'd make a test for it so I could prove how it happened, and help prevent it from happening again in the future.

In my opinion, Its very tempting to setup a complete environment, fake users, fake posts, fake everything. But the bigger the more complicated the test the harder it will to maintain.

Big tests are really good, but they can come later.

That doesn't mean no bugs occur, but its a start.

At the start, there's really no wrong way of testing. When you start to have a lot of tests, then it makes sense to worry about how long they take, and combining if necessary.

Don't forget bugs

How do I test?

Really easy.

1. Run some code
2. Check the value

```
test "should save article with title" do
  article = Article.new(title => "Gavin's Article")
  assert article.save
end

test "should load with the same title" do
  # create new article
  created_article = Article.create(title => "Gavin's Better Article")
  assert_not_null created_article
  # load from DB
  loaded_article = Article.find(created_article.id)
  # check values
  assert_not_null loaded_article
  assert_equal loaded_article.title, "Gavin's Better Article"
end
```

Okay, so lets see how you actually test.

Step 1, run some code

Step 2, Check the value.

As you can see here, testing actually can be quite simple.

When we refactored the whole system's database engine. We knew that inputting a certain set of data, we'd get a certain set back out. How it worked in between wasn't important.

E2E Tests

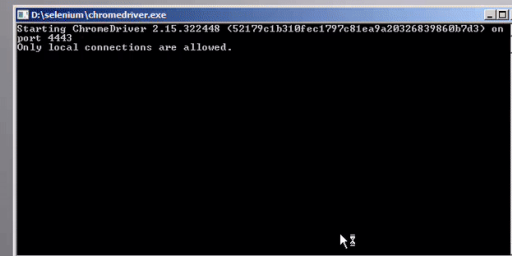
- Usually involve setting up a complete environment
- Websites will actually automate browsers for testing
- Mobile apps will test in emulators or real devices with real apps
- Talking to a fully setup backend. Hopefully not production code, but certainly possible

Example

<https://github.com/saucelabs-sample-test-frameworks/JS-Mocha-WD.js>

```
Running "shell:runTests:chrome" (shell) task
stdout:

stdout:
stdout:   mocha spec examples (chrome)
stdout:
stdout:
stdout:   mocha spec examples (chrome)
stdout:
stdout:   ✓ should get home page (1743ms)
stdout:
stdout:   ✓ should get home page (1888ms)
stdout:
stdout:   ✓ should go to the doc page1 (2152ms)
stdout:
stdout:   ✓ should go to the doc page1 (2080ms)
```



```
D:\selenium\chromedriver.exe
Starting ChromeDriver 2.15.322448 (52179c1b310fec1797c81ea9a20326839860b7d3) on
port 4443
Only local connections are allowed.
```


More Information

- Selenium - Website Testing / Automation
 - <http://www.seleniumhq.org/>
- Appium - Mobile (And more) / Automation
 - <http://appium.io/>
 - Android
 - IOS
 - Universal Windows Platform
- Xamarin
 - (Let other speaker talk about it)

Thanks

Gavin Mogan

@halkeye

<http://www.gavinmogan.com>



Thanks for your time everyone. I hope people learned at least something. Feel free to hit me up by email, or on twitter, or come up and say hi after all the talks.

I'll try and make sure I post my slides to the meetup asap.