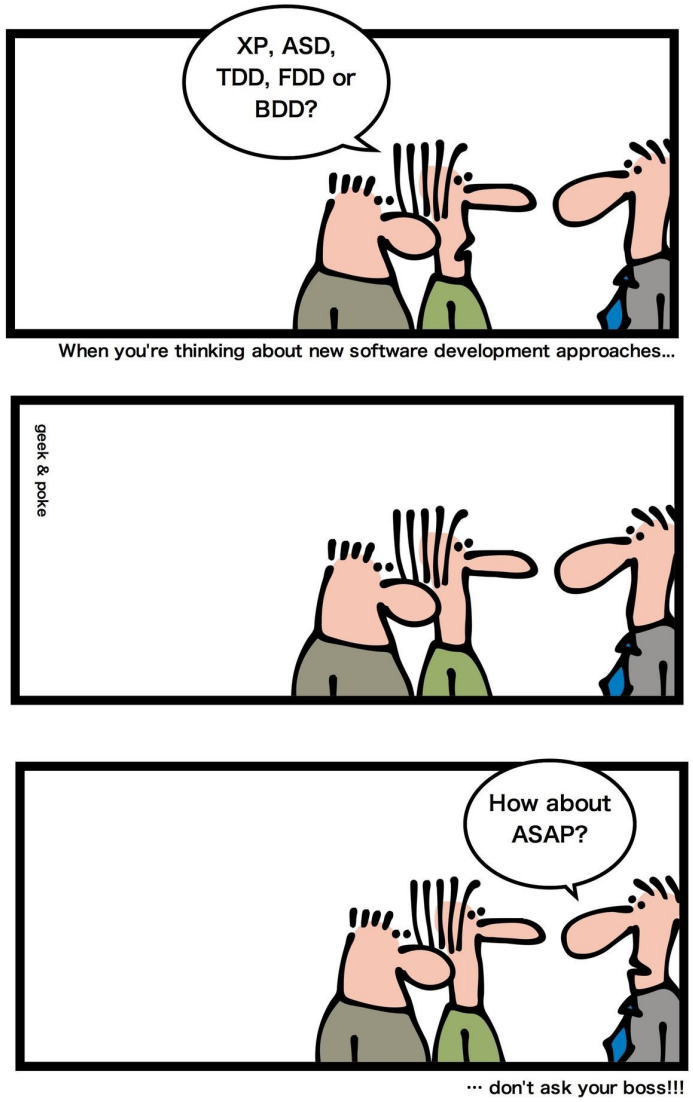
Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Agile Testing Technologies Librairies, and Frameworks **V3**

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

S. Ingram

Dont ask your boss…



Today's plan

- **Small exercice/example on JEST and TDD.**
- **(Pre-requisite) Test Doubles:** Mock, stub, and fake description.
- **Mocha for (Unit, Integration, BDD) Testing:**
  - Comparison with other JS frameworks for unit and integration tests.
  - Mocha Description.
  - Mocha BDD-style assertion styles using Chai.
  - Mocha spying, stubbing, and mocking using Sinon.js.
- **Cucumber for BDD Testing:**
  - Cucumber Description.
  - Mocha vs Cucumber.
  - Cucumber **feature** description with Gherkin.
  - Cucumber test implementation with Cucumber.js.
- **Cypress for Automated UI Testing:**
  - Cypress vs Selenium.
  - Cypress features and bundled tools.
  - Practice with Cypress UI testing.

Small exercice/example on the TDD approach and JEST unit testing library:

- In a previous course edition, I asked students to write a function and its test (or should I say tests!) that can classify the outcome of a PCR-like test based on reference values.
- I am sharing the exercice and two solutions submitted, so we can evaluate them together.

This is the exercice

- Write a function supposed to return the outputs below depending on the input provided.
- Use a **TDD** approach, and document all the different steps followed.

Input	Output
1	PC
0.2	D
0.7	PC
4	C
2	C
0.5	PC

Is this solution submitted correct?

Input	Output
1	PC
0.2	D
0.7	PC
4	C
2	C
0.5	PC

test > TS TDD.test.ts > ...

```ts
import fn from '../src/TDD';

describe('TDD practice', () => {
    it('Should produce correct output', () => {
        const inputs = [1, 0.2, 0.7, 4, 2, 0.5];
        const outputs = ['PC', 'D', 'PC', 'C', 'C', 'PC'];

        inputs.forEach((v, k) => {
            expect(fn(v)).toBe(outputs[k]);
        });
    });
});
```

```
> ts-avl-tst@0.1.0 test C:\Users\Guillaume Hochet\Desktop\ts-avl-tst
> jest

 PASS  test/TDD.test.ts
  TDD practice
    √ Should produce correct output (4ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.877s, estimated 2s
Ran all test suites.
PS C:\Users\Guillaume Hochet\Desktop\ts-avl-tst>
```

Is this one correct?

1) Does the function comply with “business” requirements? (Input/output)
2) Does it use TDD?

Input	Output
1	PC
0.2	D
0.7	PC
4	C
2	C
0.5	PC

test > TS TDD.test.ts > ...

```ts
import fn from '../src/TDD';

describe('TDD practice', () => {
    it('Should produce correct output', () => {
        const inputs = [1, 0.2, 0.7, 4, 2, 0.5];
        const outputs = ['PC', 'D', 'PC', 'C', 'C', 'PC'];

        inputs.forEach((v, k) => {
            expect(fn(v)).toBe(outputs[k]);
        });
    });
});
```

```
> ts-avl-tst@0.1.0 test C:\Users\Guillaume Hochet\Desktop\ts-avl-tst
> jest

 PASS  test/TDD.test.ts
  TDD practice
    √ Should produce correct output (4ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.877s, estimated 2s
Ran all test suites.
PS C:\Users\Guillaume Hochet\Desktop\ts-avl-tst>
```

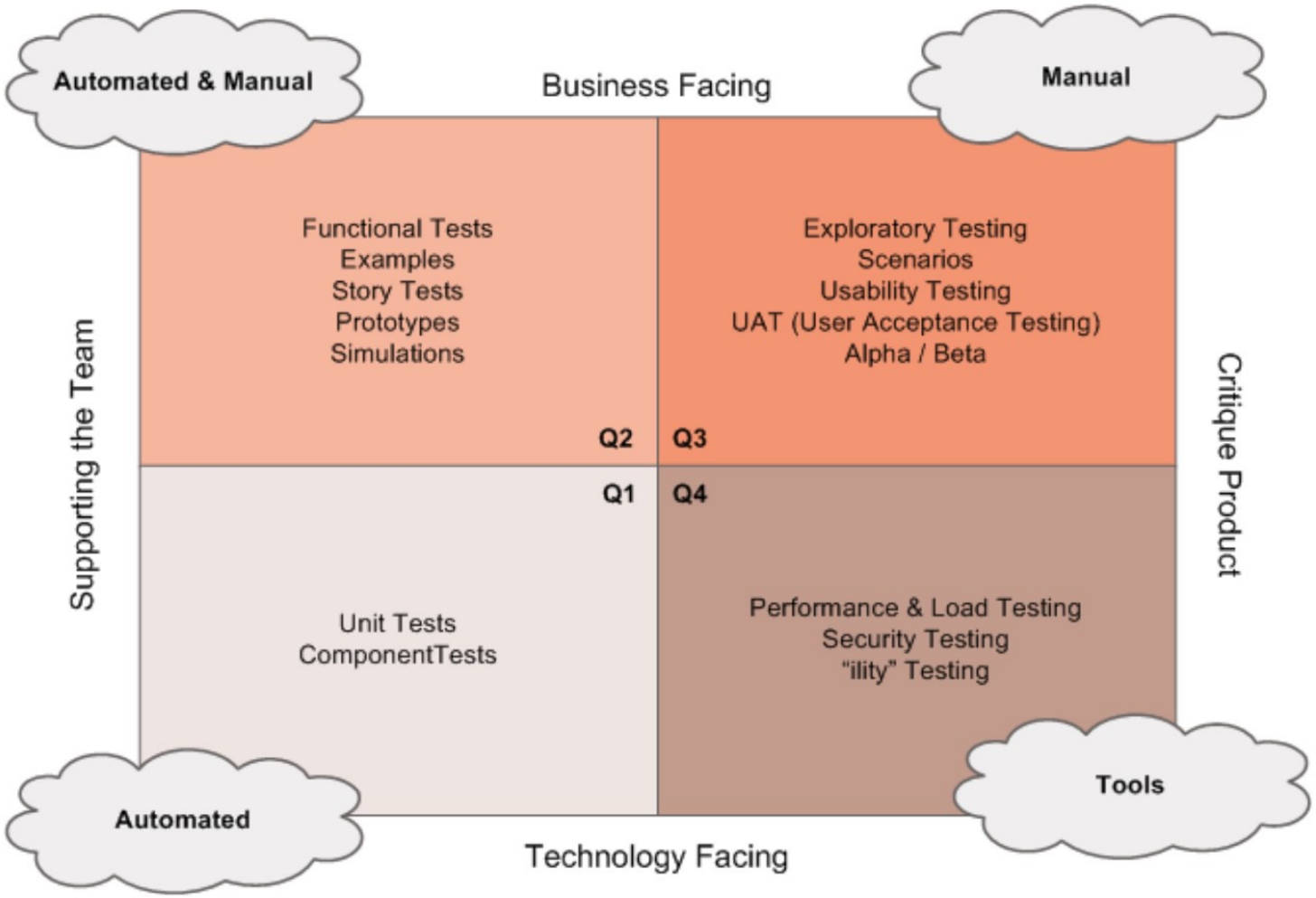
Is this one correct?

```
> jest --verbose ./test_TDD

 PASS  test_TDD/exercise.test.js
  ✓ 1 returns PC (3 ms)
  ✓ 0.2 returns D (1 ms)
  ✓ 0.7 returns PC
  ✓ 4 returns C
  ✓ 2 returns C
  ✓ 0.5 returns PC (1 ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        1.257 s
Ran all test suites matching /.\/test_TDD/i.
```

```
function mysterious(input) {
    var expected = ' ';

    if(input == 1)
        expected = 'PC';
    else if(input == 0.2)
        expected = 'D';
    else if(input == 0.7)
        expected = 'PC';
    else if(input == 4)
        expected = 'C';
    else if(input == 2)
        expected = 'C';
    else if(input == 0.5)
        expected = 'PC';

    return expected;
  }
```

```
test.each([[1, 'PC'], [0.2, 'D'], [0.7, 'PC'], [4, 'C'], [2, 'C'], [0.5, 'PC']])(
    '%f returns %s', (input, expected) => {
    expect(mysterious(input)).toBe(expected);
    },
);
```

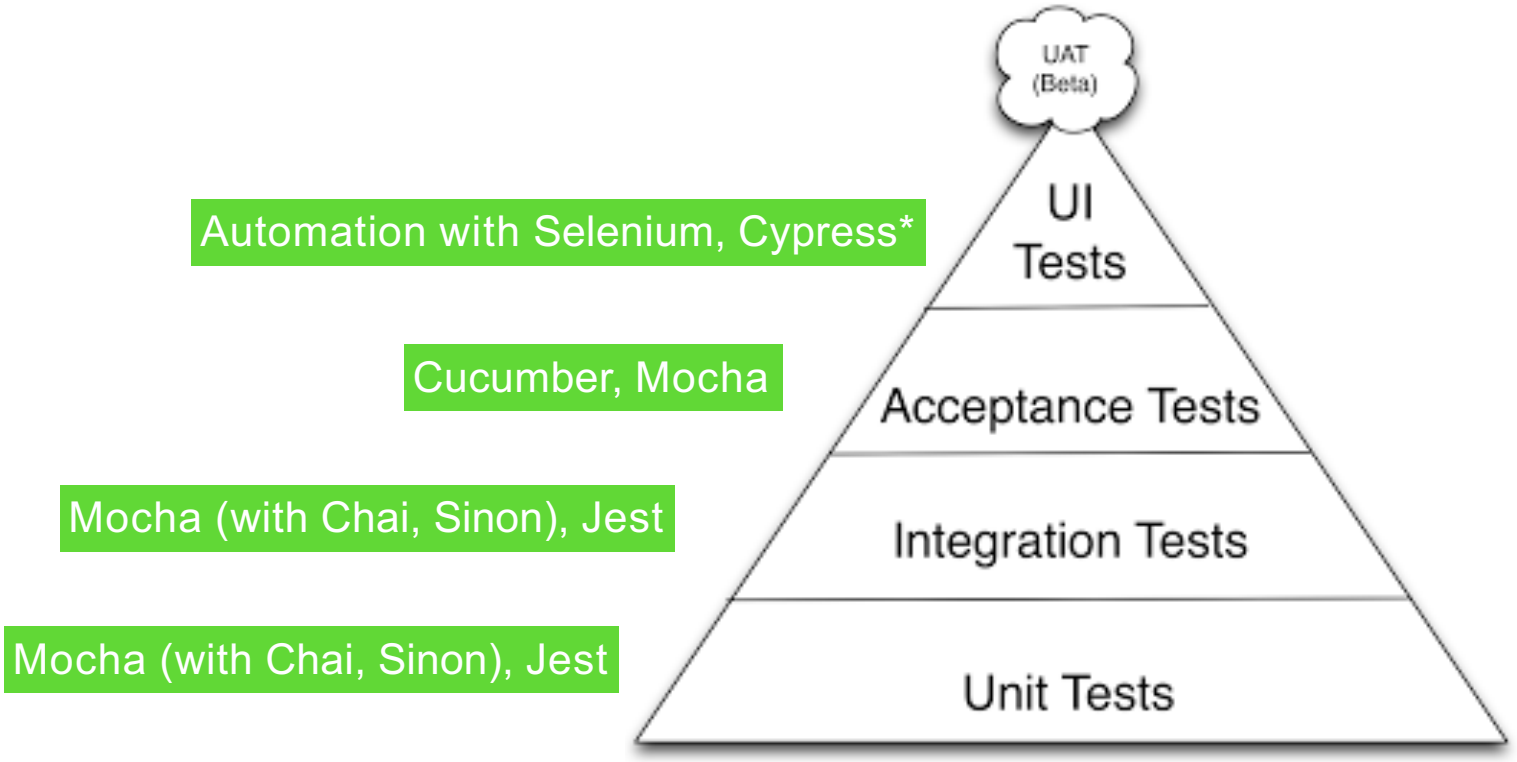
Agile Testing Quadrant



Copyright 2009: Lisa Crispin

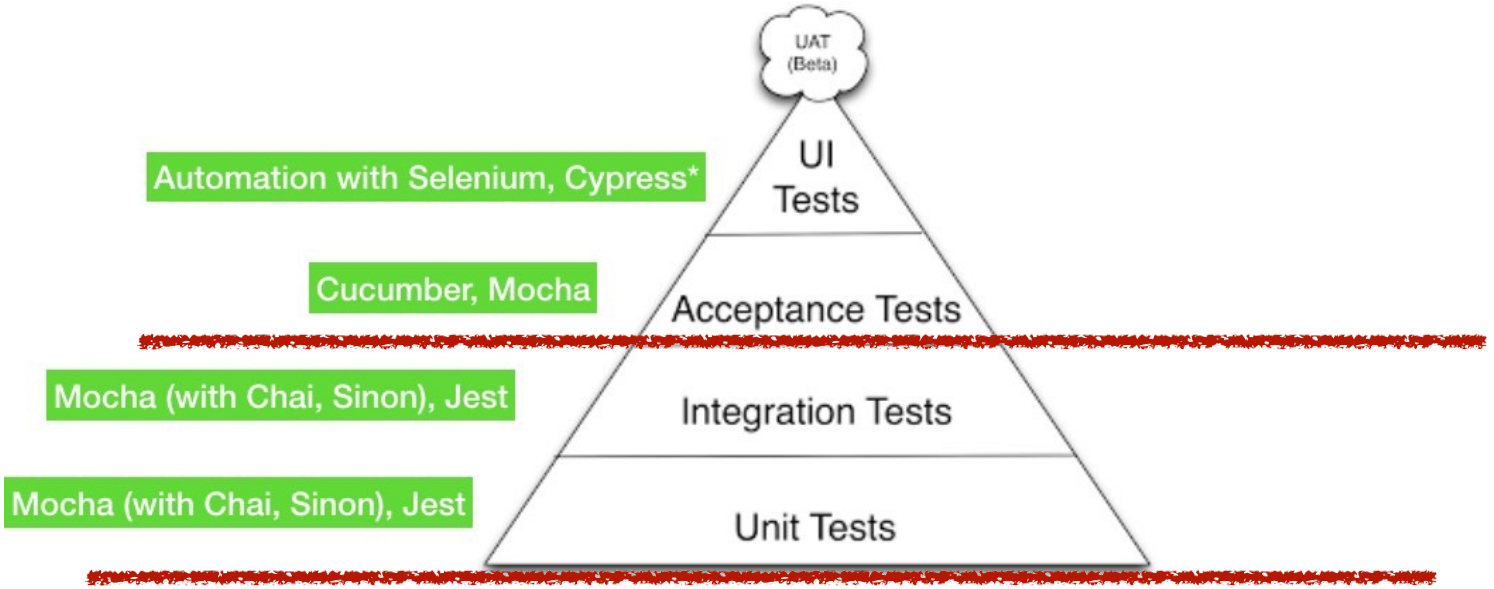
And one more Testing Pyramid

(The best among those shared so far, in my view, can you guess why?)



** Since Cypress integrates Chai and Sinon, it can also be used in lower pyramid sections.*

Test Doubles with Sinon.js



Test Doubles

Test Doubles are similar to actor ‘stunts’ in movies.

https://www.vulyplay.com/img/blog/trampoline-stunts-movies-pyrotechnics.jpg

Test Doubles

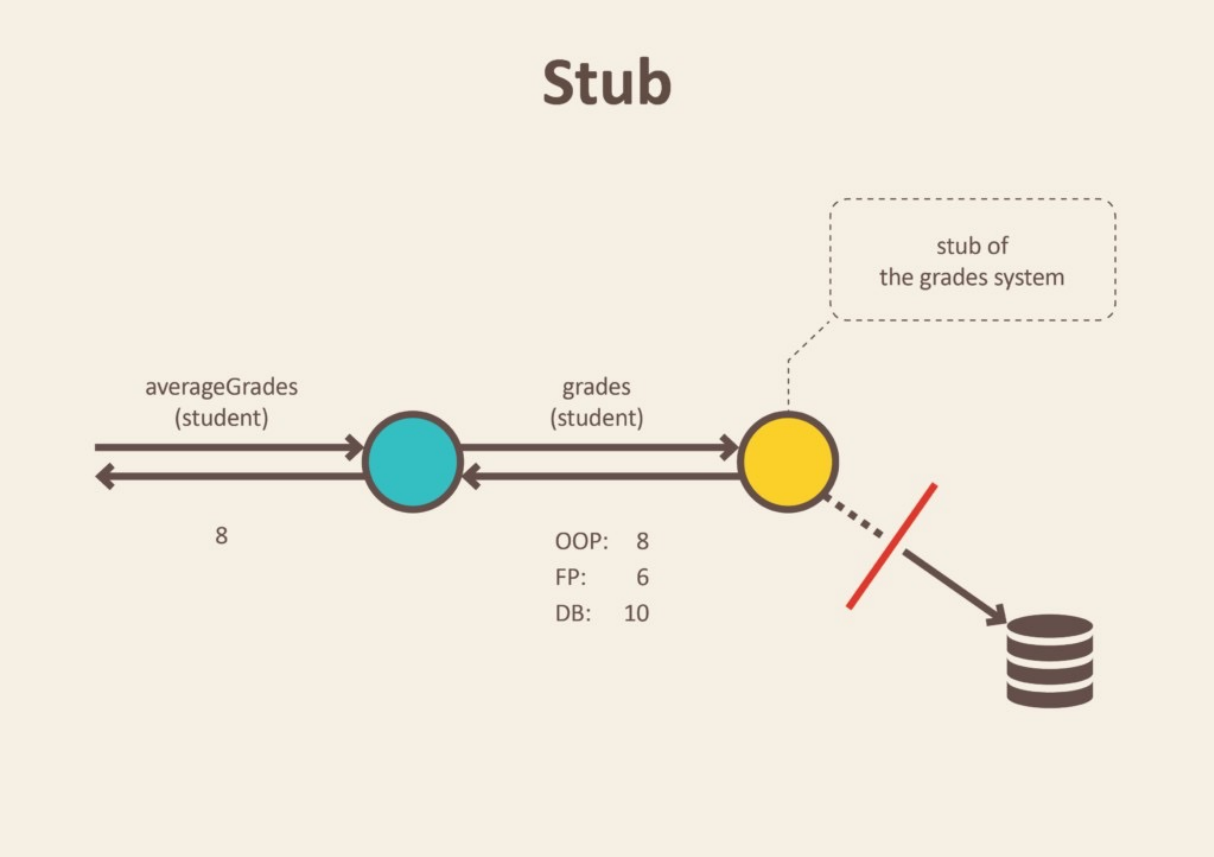
- The term was introduced by Meszaros in his ‘xUnit Test Patterns’ book.
- There are 3 main types of Test Doubles: **Stubs, Fakes, Mocks.**

Stubs

- Stubs contain no logic.
- A stub simply returns the **values** it is asked to return, allowing a test to reach a certain step.

Stubs

Suppose you want to write a unit test a function that computes average grades, you want to test it independently from what it depends on (it is not a integration test!), in an “isolated” way from eventual external errors. so you can give it the data it expects to work properly (here the grades) instead of retrieving them from the real database.

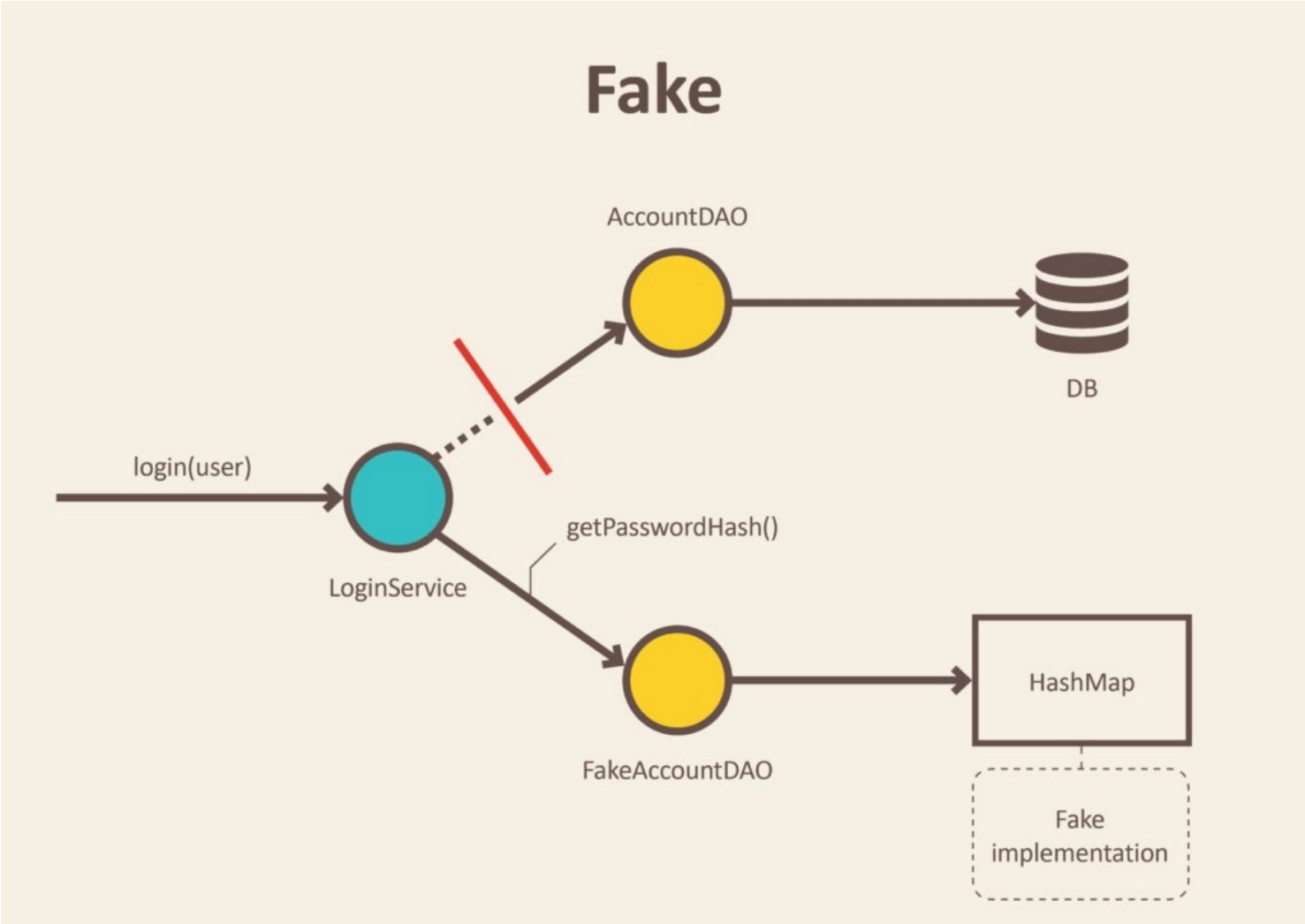


https://blog.pragmatists.com/test-doubles-fakes-mocks-and-stubs-1a7491dfa3da

Fakes

- Compared to stubs, fakes are closer in terms of behaviour to the real objects they replace.
- Usually the person who created the real object, writes its fake.
- Example: replace a database call with a call for an-memory data structure.

Fakes

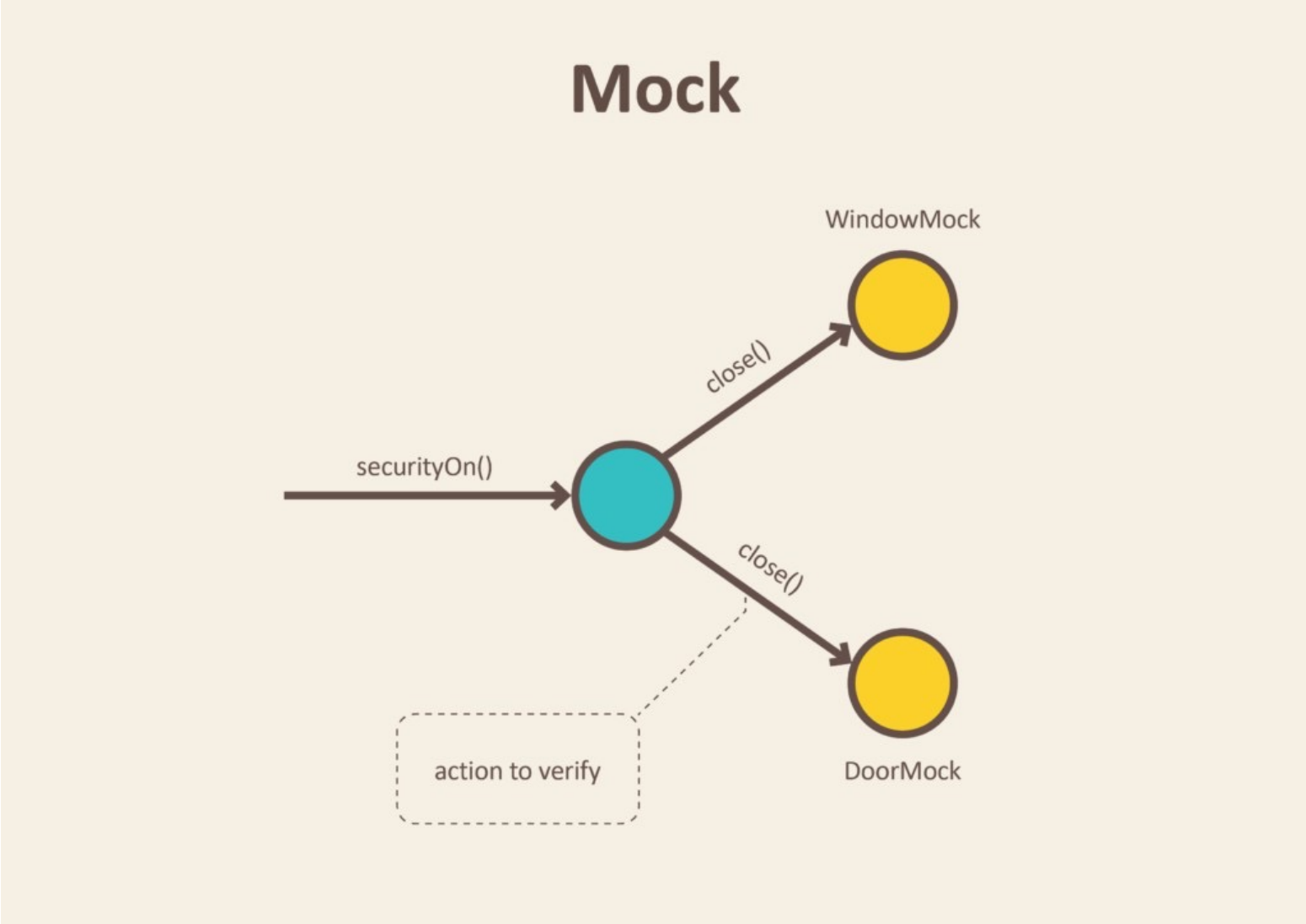


https://blog.pragmatists.com/test-doubles-fakes-mocks-and-stubs-1a7491dfa3da

Mocks

- Mocks are used to test the **interaction** between objects.
- Mocks are useful for functions returning no values.
- The test fails if the mocks object is not **called** as **expected**.

Mocks



https://blog.pragmatists.com/test-doubles-fakes-mocks-and-stubs-1a7491dfa3da

Stub? Mock?

```
public interface MailService {
  public void send (Message msg);
}
public class MailServiceStub implements
MailService {
  private List<Message> messages = new
ArrayList<Message>();
  public void send (Message msg) {
    messages.add(msg);
  }
  public int numberSent() {
    return messages.size();
  }
}
// We can then use state verification on the stub like this.
```

class OrderStateTester...

```
  public void testOrderSendsMailIfUnfilled() {
    Order order = new Order(TALISKER, 51);
    MailServiceStub mailer = new
MailServiceStub();
    order.setMailer(mailer);
    order.fill(warehouse);
    assertEquals(1, mailer.numberSent());
  }
```

•

Class OrderInteractionTester...

```
  public void
testOrderSendsMailIfUnfilled() {
    Order order = new Order(TALISKER, 51);
    Mock warehouse =
mock(Warehouse.class);
    Mock mailer = mock(MailService.class);
    order.setMailer((MailService)
mailer.proxy());

    mailer.expects(once()).method("send");

warehouse.expects(once()).method("hasInven
tory")
       .withAnyArguments()
       .will(returnValue(false));

    order.fill((Warehouse)
warehouse.proxy());
  }
}
```

https://martinfowler.com/articles/mocksArentStubs.html

Stubs and Mock side by side

```
public interface MailService {
  public void send (Message msg);
}
public class MailServiceStub implements
MailService {
  private List<Message> messages = new
ArrayList<Message>();
  public void send (Message msg) {
    messages.add(msg);
  }
  public int numberSent() {
    return messages.size();
  }
}
```

We can then use state verification on the stub like this.

class OrderStateTester...

```
  public void testOrderSendsMailIfUnfilled() {
    Order order = new Order(TALISKER, 51);
    MailServiceStub mailer = new
MailServiceStub();
    order.setMailer(mailer);
    order.fill(warehouse);
    assertEquals(1, mailer.numberSent());
  }
```

•

lass OrderInteractionTester...

```
  public void testOrderSendsMailIfUnfilled() {
    Order order = new Order(TALISKER, 51);
    Mock warehouse = mock(Warehouse.class);
    Mock mailer = mock(MailService.class);
    order.setMailer((MailService)
mailer.proxy());

    mailer.expects(once()).method("send");

warehouse.expects(once()).method("hasInventory"

)
      .withAnyArguments()
      .will(returnValue(false));

    order.fill((Warehouse) warehouse.proxy());
  }
}
```

https://martinfowler.com/articles/mocksArentStubs.html

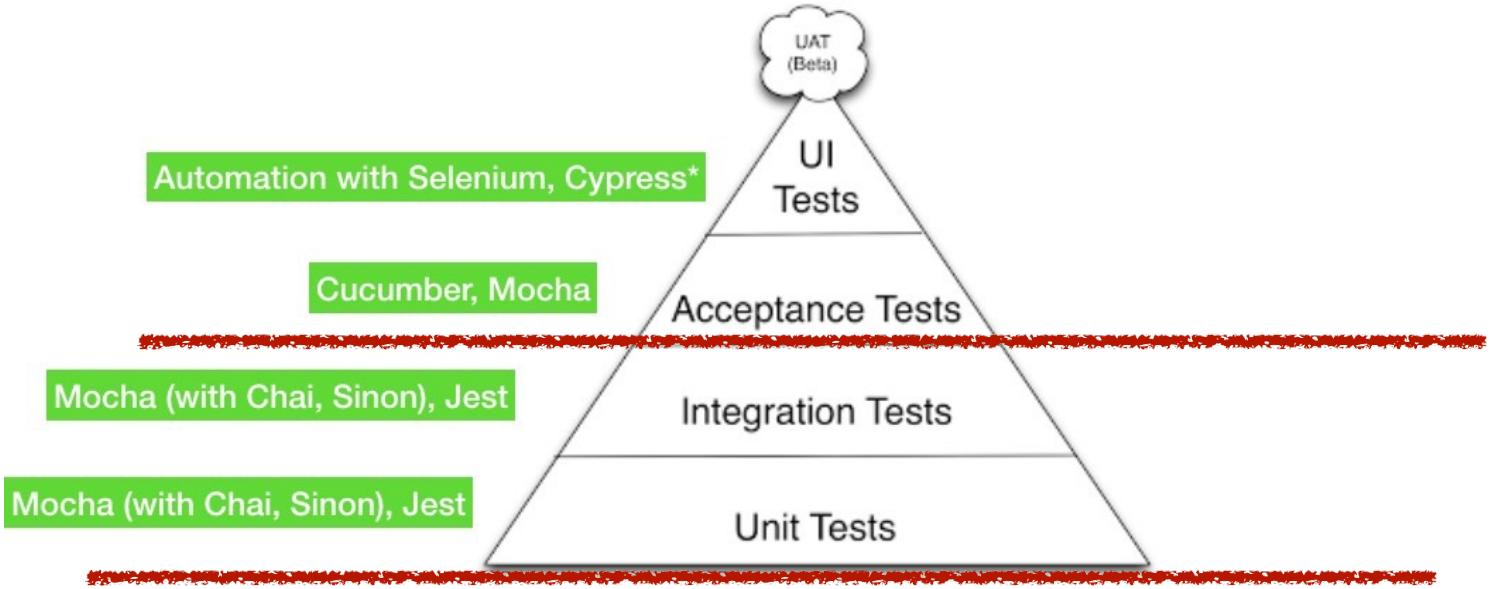
Stubs rely on returned values

Mocks use method calls

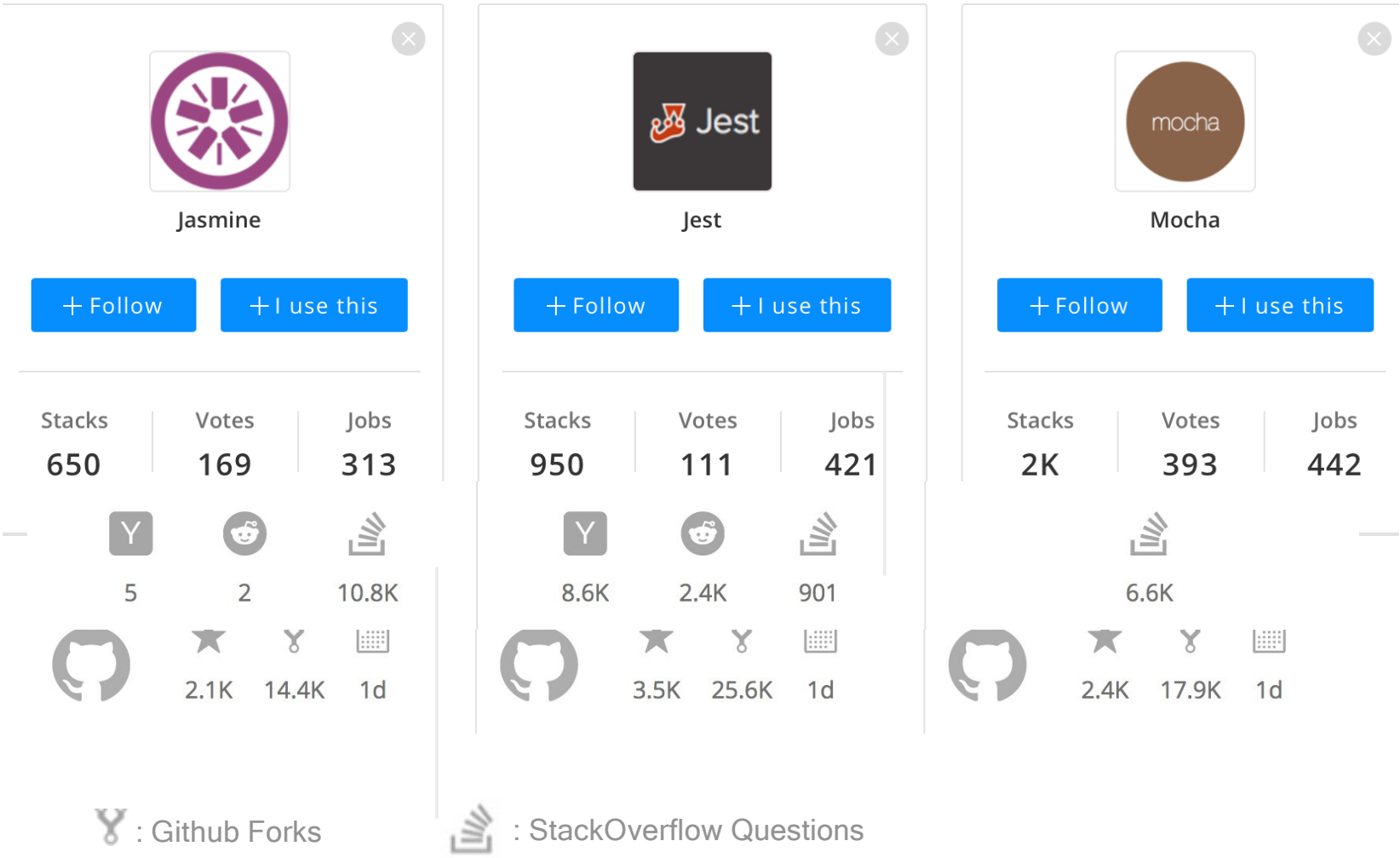
Today's plan

- **Test Doubles:** Mock, stub, and fake description.
- **Mocha for Unit, Integration, BDD Testing:**
  - Comparison with other JS frameworks for unit and integration tests.
  - Mocha Description.
  - Mocha BDD-style assertion styles using Chai.
  - Mocha spying, stubbing, and mocking using Sinon.js.
- **Cucumber for BDD Testing:**
  - Cucumber Description.
  - Mocha vs Cucumber.
  - Cucumber **feature** description with Gherkin.
  - Cucumber test implementation with Cucumber.js.
- **Cypress for Automated UI Testing:**
  - Cypress vs Selenium.
  - Cypress features and bundled tools.
  - Practice with Cypress UI testing.

Unit and Integration tests with Mocha



Libraries and Frameworks for Unit and Integration Tests



https://stackshare.io/stackups/jasmine-vs-jest-vs-mocha

Libraries and frameworks for Unit and Integration Tests

Jasmin

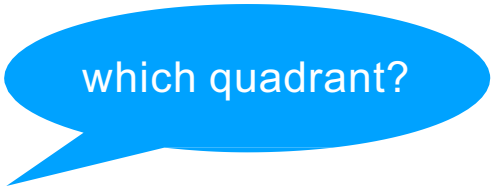
- Created in ‘2008’
- Batteries included’: Complete Testing framework

- Created in 2014, has drastically evolved since then.
- **Easy to set up:** zero configuration.
- Runs **unrelated tests in parallel** (unlike Jasmin and Mocha).
- Offers **Snapshots**.

- Created in 2011
- **No** built-in assertions and test doubles.
- Highly flexible, suitable for large projects.

Testing with Mocha

- Javascript testing framework.
- **BDD**-style syntax.
- Used for unit, integration, and acceptance tests.
- Simplifies testing **asynchronous** code.
- Unlike Jasmin, it does **not** include built-in assertions or stubbing.
  - **Chai** is used for assertions.
  - **Sinon.js** is used for stubbing and spying.



Mocha BDD-style Hooks

```
describe('hooks demo', function() {
  before(function() {
    // runs before all tests in this block
  });

  after(function() {
    // runs after all tests in this block
  });

  beforeEach('some description', function() {
    // runs before each test in this block
    // optional hook description to better understand errors
  });
  afterEach(function() {
    // runs after each test in this block
  });

  // test cases
});
```

Pay attention to the order of execution of hooks

Assertions in Mocha using Chai

- Chai is an **Assertion** Framework used with Mocha, (and Cypress).
- Chai offers human-readable syntax for assertions and error messages.

Assertions in Mocha using Chai

- Chai is an **Assertion** Framework used with Mocha, (and Cypress).
- Chai offers human-readable syntax for assertions and error messages.

http://frontend.turing.io/lessons/module-2/test-driven-development-with-webpack.html

Mocha Test Doubles using Sinon.js

- Sinon.js is a Javascript library that can be used with unit testing framework.
- Fakes, stubs, **spies**, and mocks can be generated using Sinon.js.

https://sinonjs.org/releases/v7.3.2/mocks/

https://sinonjs.org/releases/v7.3.2/fakes/

https://sinonjs.org/releases/v7.3.2/spies/

https://sinonjs.org/releases/v7.3.2/stubs/

Testing Doubles with Sinon.js

- **Fakes** once created, are immutable.
- **Spies** watch functions, and record the arguments those functions receive, their return value and potential exceptions thrown from calling those functions.

Testing Doubles with Sinon.js

- **Fakes**, once created, are immutable.
- **Spies** watch functions, and record the arguments those functions receive, their return value and potential exceptions thrown from calling those functions.
- **Stubs** implement all the Spy APIs and adds APIs to **change** behaviour.
- **Mock** “expectations” implement **both** Spy and Stub APIs.

Example using spies and stubs with Sinon.js

```
"??????????" : function(){
    var message = 'an example message';
    var stub = sinon.stub().throws();
    var spy1 = sinon.spy();
    var spy2 = sinon.spy();

    PubSub.subscribe(message, stub);
    PubSub.subscribe(message, spy1);
    PubSub.subscribe(message, spy2);

    PubSub.publishSync(message, undefined);

    assert(spy1.called);
    assert(spy2.called);
    assert(stub.calledBefore(spy1))
    ;
}
```

What does this test do?

Example using spies and stubs with Sinon.js

The answer to the question of what the function does

```
"test should call all subscribers, even if there are exceptions" : function(){
    var message = 'an example message';
    var stub = sinon.stub().throws();
    var spy1 = sinon.spy();
    var spy2 = sinon.spy();

    PubSub.subscribe(message, stub);
    PubSub.subscribe(message, spy1);
    PubSub.subscribe(message, spy2);

    PubSub.publishSync(message, undefined);

    assert(spy1.called);
    assert(spy2.called);
    assert(stub.calledBefore(spy1));
}
```

Why is a stub used here and not a spy?

Anything you would have done otherwise?

Example using Mocks with Sinon.js

```
"test should call all subscribers even with exceptions": function ()
{
    var myAPI = { method: function () {} };
    var spy = sinon.spy();
    var mock = sinon.mock(myAPI);
    mock.expects("method").once().throws(); // throw an exception
    PubSub.subscribe("message", myAPI.method);
    PubSub.subscribe("message", spy);
    PubSub.publishSync("message", undefined);
    mock.verify();
    assert(spy.calledOnce);
}
```

https://github.com/cypress-io/sinon/blob/master/docs/_releases/v2.4.0/mocks.md

Another example using Mocha with asynchronous code !

NOT FOR THE EXAM

```
describe("Color Code Converter API", function() {

    describe("RGB to Hex conversion", function() {

        var url = "http://localhost:3000/rgbToHex?red=255&green=255&blue=255";

        it("returns status 200", function() {
            request(url, function(error, response, body) {
                expect(response.statusCode).to.equal(200);
            });
        });

        it("returns the color in hex", function() {
            request(url, function(error, response, body) {
                expect(body).to.equal("ffffff");
            });
        });

    });
```

Any problem with this example?

A “better” example with Mocha

NOT FOR THE EXAM

```
describe("Color Code Converter API", function() {

  describe("RGB to Hex conversion", function() {

    var url = "http://localhost:3000/rgbToHex?red=255&green=255&blue=255";

    it("returns status 200", function(done) {
      request(url, function(error, response, body) {
        expect(response.statusCode).to.equal(200);
        done();
      });
    });

    it("returns the color in hex", function(done) {
      request(url, function(error, response, body) {
        expect(body).to.equal("ffffff");
        done();
      });
    });

  });
```

Add a callback function (here “done”) AFTER the last assertion so Mocha waits for the function to be called before executing the test

More on handling asynchronous code: https://mochajs.org/#asynchronous-code

Async/Await can also be used for asynchronous tests

NOT FOR THE EXAM

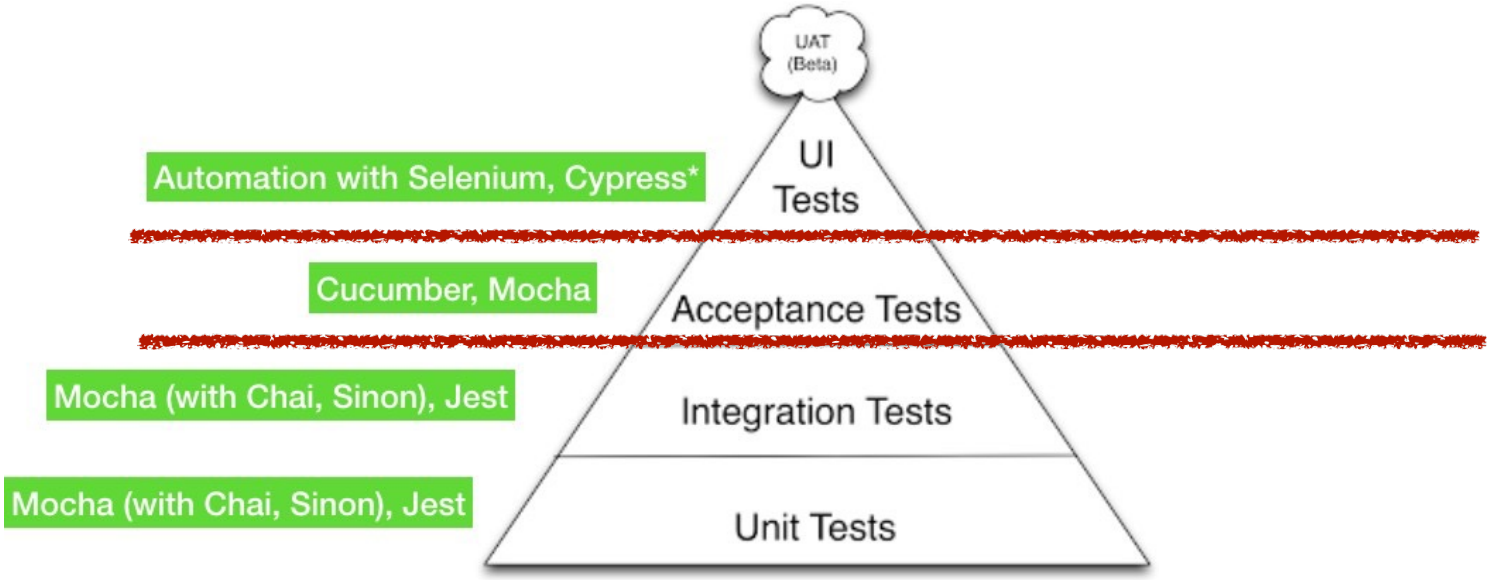
```
beforeEach(async function() {
  await db.clear();
  await db.save([tobi, loki, jane]);
});

describe('#find()', function() {
  it('responds with matching records', async function() {
    const users = await db.find({type: 'User'});
    users.should.have.length(3);
  });
});
```

More on handling asynchronous code: https://mochajs.org/#asynchronous-code

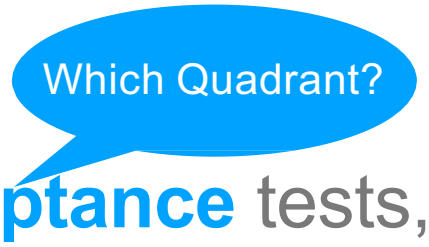
Today's plan

- **Test Doubles:** Mock, stub, and fake description.
- **(Unit, Integration, BDD) Testing with Mocha:**
  - Comparison with other JS frameworks for unit and integration tests.
  - Mocha Description.
  - Mocha BDD-style assertion styles using Chai.
  - Mocha spying, stubbing, and mocking using Sinon.js.
- **BDD Testing with Cucumber:**
  - Cucumber Description.
  - Mocha vs Cucumber.
  - Cucumber **feature** description with Gherkin.
  - Cucumber test implementation with Cucumber.js.
- **Automated UI Testing with Cypress:**
  - Cypress vs Selenium.
  - Cypress features and bundled tools.
  - Practice with Cypress UI testing.

BDD Acceptance Tests with Cucumber



Cucumber



- **BDD** Framework for **acceptance** tests,
- Adapted for non programmers.
- The expected behaviour of tested features is described in a **text** file using the **Gherkin** syntax.
- Tests are then **implemented** in different programming languages (Java, Javascript, Ruby).
- Cucumber integrates well with browser drivers for UI automation (such as Selenium).

Cucumber Feature Description

Feature files are written in Gherkin and focus on What, and not How.

Feature: Users must be able to search for content using “the Search” button.

Scenario: Search for a term.

Given Given I have entered “watir” into the query.

When I click “Search"

Then I should see some result.

https://www.tutorialspoint.com/cucumber/cucumber_ruby_testing.htm

Example of a Cucumber Test implemented with the Ruby programming language

```
require "rubygems"
require "test/unit"
require "watir-webdriver"

class GoogleSearch < Test::Unit::TestCase
def setup
@browser ||= Watir::Browser.new :firefox
end

def teardown
@browser.close
end

def test_search
@browser.goto "google.com"
@browser.text_field(:name => "q").set "watir"
@browser.button.click
@browser.div(:id => "resultStats").wait_until_present assert
@browser.title == "watir - Google Search"
end
end
```

browser automation

https://www.tutorialspoint.com/cucumber/cucumber_ruby_testing.htm

Let’s write a test for a simple addition feature using the Cucumber BDD framework

1) We need to provide the feature description and scenario

$ mkdir cucumber_example
$ cd cucumber_example
$ mkdir features
$ vim addition.feature

```
Feature: Addition
        Test if calculator adds two positive numbers correctly

        Scenario: Addition of two positive numbers
                Given I have number 2 in calculator
                When I entered number 3
                Then I should see result 5
```

3) Define test steps for the addition feature test

$ mkdir step_definitions
$ cd step_definitions
$ vim definition1.js →

```
const { Given, When, Then } =
require('cucumber');
const assert = require('assert')

//// Your step definitions /////
Given(/^I have number (\d+) in
calculator$/, function (num) {
    this.setTo(num);
});

When(/^I entered number (\d+)$/,
function (num) {
    this.incrementBy(num);
});

Then(/^I should see result (\d+)$/,
function (result) {
    assert.equal(this.variable,
parseInt(result));
});
```

4) Implement and specify the code to test

In some cases, it is useful to define its own CustomWorld, including test *instance* properties and methods available to steps and hooks (afterEach, beforeEach) cf. 'Custom World' in

https://github.com/cucumber/cucumber-js/blob/main/docs/support_files/world.md

$ cd features
$ mkdir support
$ cd support
$ vim env.js

env.js

```
const { setWorldConstructor} = require('cucumber')

class CustomWorld {
  constructor() {
  this.variable = 0
  }

  setTo(number) {
    this.variable = number
  }

  incrementBy(number) {
    this.variable += number
  }
}

setWorldConstructor(CustomWorld)
```

All Side by Side

Feature Description File (Gherkin):

```
Feature: Addition
        Test if calculator adds two positive numbers correctly

        Scenario: Addition of two positive numbers
                Given I have number 2 in calculator
                When I entered number 3
                Then I should see result 5
```

Tested module

```
const { setWorldConstructor } = require('cucumber')

class CustomWorld {
  constructor() {
  this.variable = 0
  }

  setTo(number) {
    this.variable = number
  }

  incrementBy(number) {
    this.variable += number
  }
}

setWorldConstructor(CustomWorld)
```

The corresponding test

```
const { Given, When, Then } =
require('cucumber');
const assert = require('assert')
                        Regular expressions
//// Your step definitions /////
Given(/^I have number (\d+) in
calculator$/, function (num) {
    this.setTo(num);
});

When(/^I entered number (\d+)$/, function
(num) {
    this.incrementBy(num);
});

Then(/^I should see result (\d+)$/,
function (result) {
    assert.equal(this.variable,
parseInt(result));
});
```

Execute test

$ npm test

```
> cucumber-js ./features

...

1 scenario (1 passed)
3 steps (3 passed)
0m00.003s
```

Another Simple Feature Description

```
Feature: Bing Search
        This is a sample feature to test search engine

        Scenario: Search something from bing
                Given browse to web site "https://www.bing.com"
                When input keyword "Mars"
                Then click Search button
                And search result should contain "NASA"
```

https://www.codementor.io/cuketest/one-quick-way-to-create-your-cucumber-js-test-script-iq5kxwy8y

Implement the search feature

```
const { Given, When, Then } = require('cucumber');
const assert = require('assert');
const { driver } = require('../support/web_driver');

Given(/^browse to web site "([^"]*)"$/, async function(url) {
    return driver.get(url);
});

When(/^input keyword "([^"]*)"$/, async function (keyword) {
    return driver.findElement({ id: "sb_form_q" }).sendKeys(keyword);
});

Then(/^click Search button$/, async function () {
    return driver.findElement({ id: "sb_form_go" }).click();
});

Then(/^search result should contain "([^"]*)"$/, async function (keyword) {
    await driver.sleep(1000);
let result = await driver.findElement({ id: "b_results" }).getText();
    return assert.ok(result.includes(keyword));
});
```

MochaJS vs Cucumber

MochaJS vs Cucumber

- Both are suited for **BDD** testing.

MochaJS vs Cucumber

- Both are suited for **BDD** testing.
- Cucumber is easier to integration with Web drivers (e.g. Cucumber with Selenium).

MochaJS vs Cucumber

- Both are suited for **BDD** testing.
- Cucumber is easier to integration with Web drivers (e.g. Cucumber with Selenium).
- Mocha is a **Javascript** framework, Cucumber's syntax can be implemented with different languages (e.g. Ruby, Javascript).
-

MochaJS vs Cucumber

- Both are suited for **BDD** testing.
- Cucumber is easier to integrate with Web drivers (e.g. Cucumber with Selenium).
- Mocha is a **Javascript** framework, Cucumber's syntax can be implemented with different languages (e.g. Ruby, Javascript).

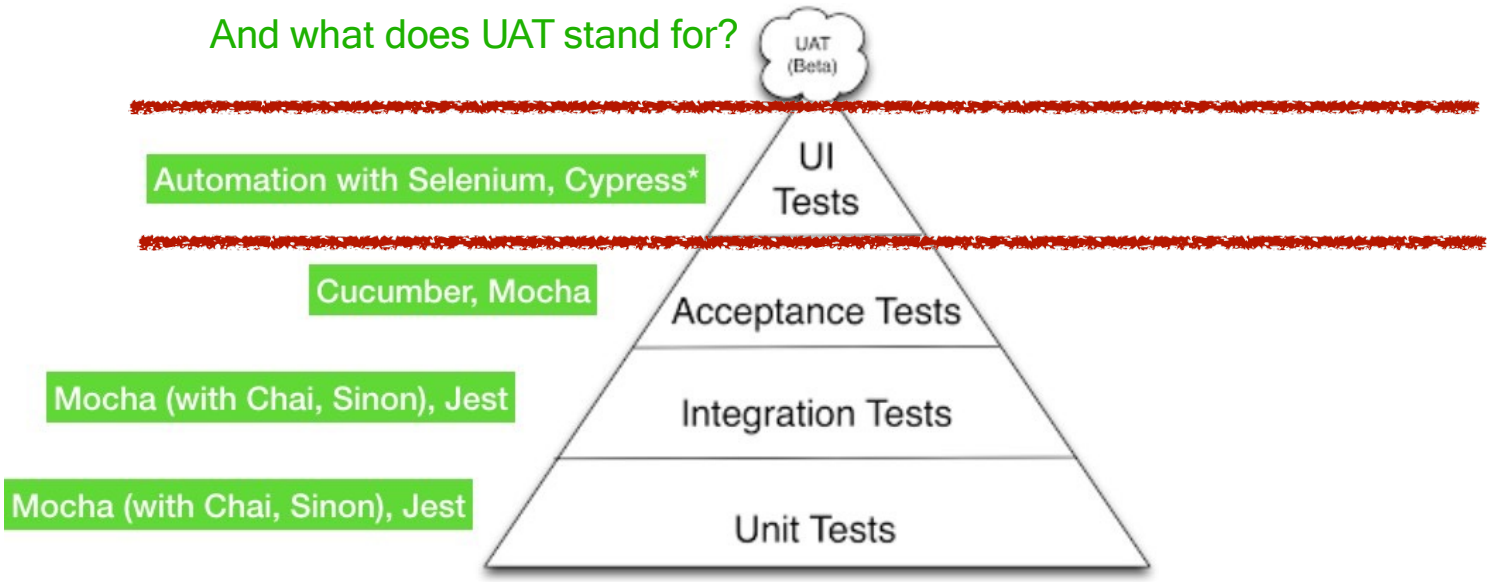
MochaJS vs Cucumber

- Both are suited for **BDD** testing.
- Mocha integration with browser drivers is less tight.
- Mocha is a **Javascript** framework, Cucumber can be implemented with different languages.
- Unlike Mocha, Cucumber separates feature files and the code written to test feature acceptance.
- Cucumber is more human-readable, more adapted to non-programmers.

Today's plan

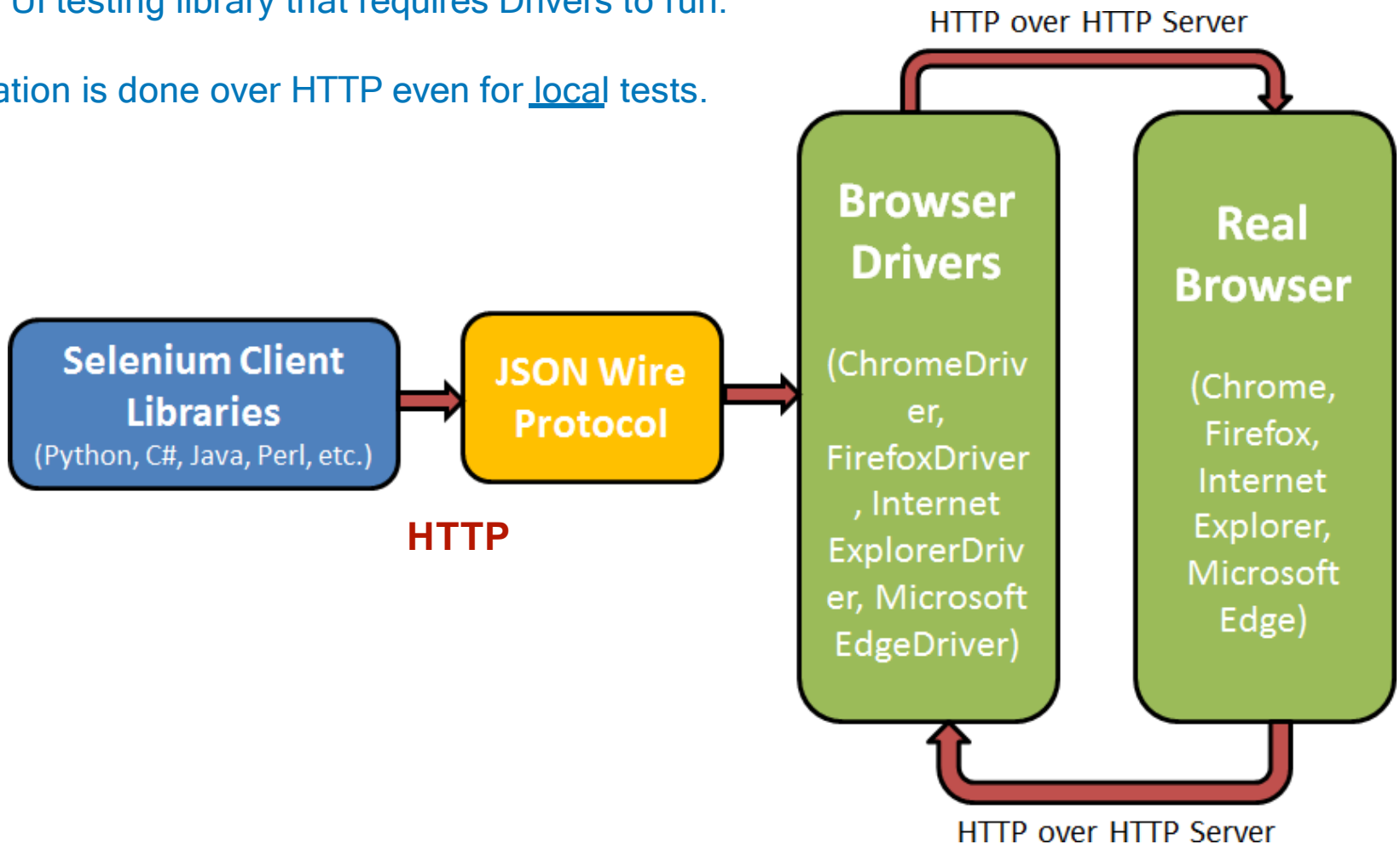
- **Test Doubles:** Mock, stub, and fake description.
- **(Unit, Integration, BDD) Testing with Mocha:**
  - Comparison with other JS frameworks for unit and integration tests.
  - Mocha Description.
  - Mocha BDD-style assertion styles using Chai.
  - Mocha spying, stubbing, and mocking using Sinon.js.
- **BDD Testing with Cucumber:**
  - Cucumber Description.
  - Mocha vs Cucumber.
  - Cucumber **feature** description with Gherkin.
  - Cucumber test implementation with Cucumber.js.
- **Automated UI Testing with Cypress:**
  - Cypress vs Selenium.
  - Cypress features and bundled tools.
  - Practice with Cypress UI testing.

Automated UI Testing with Cypress



Selenium

- Selenium is UI testing library that requires Drivers to run.
- Communication is done over HTTP even for local tests.



https://www.lambdatest.com/blog/test-automation-using-pytest-and-selenium-webdriver/

Cypress

- Cypress is a “**fast, easy, and reliable**” Javascript Testing Framework for automated browser UI testing.
- Cypress adopts Mocha’s BDD syntax for unit and integration tests.

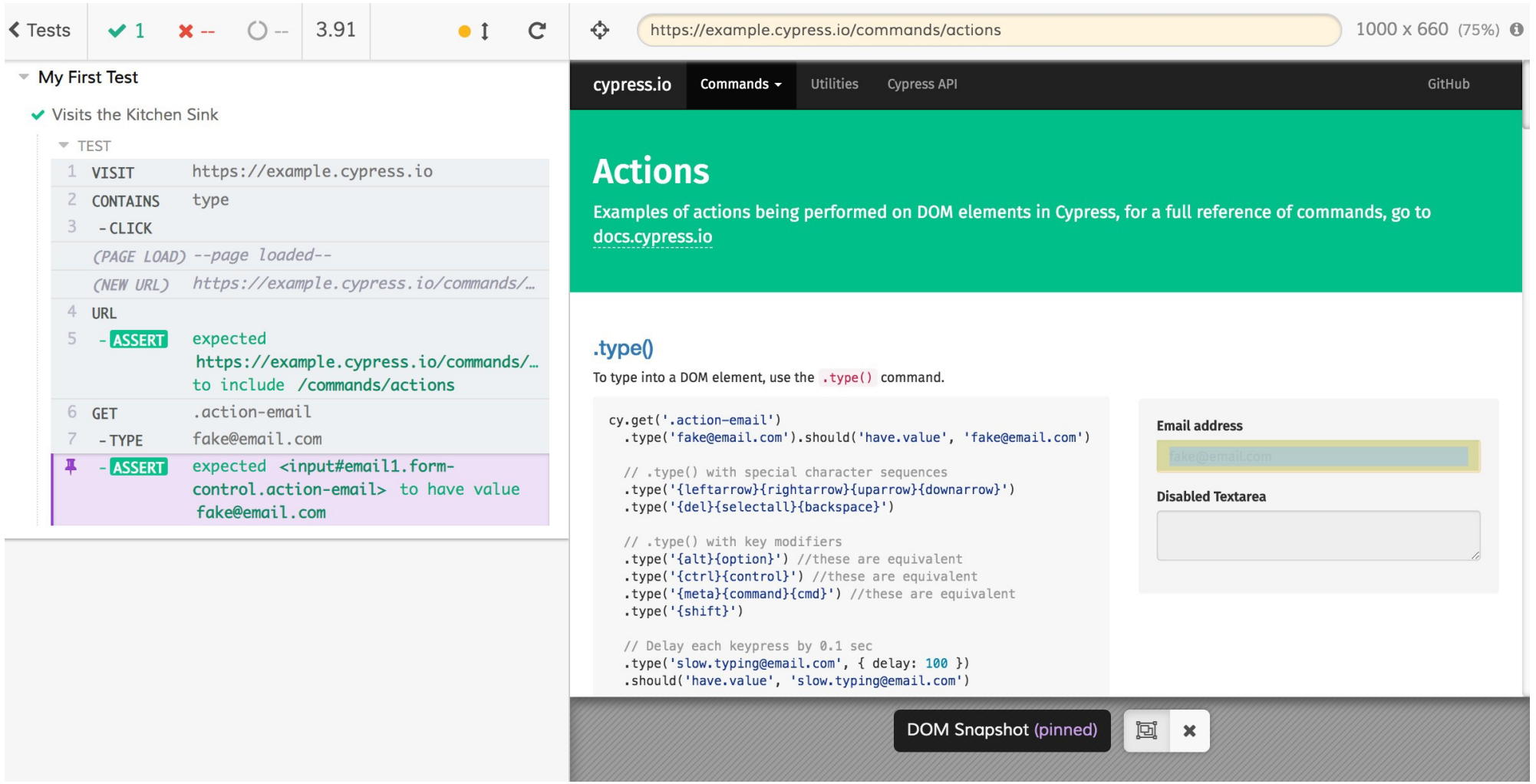
Unlike Selenium, Cypress

- Does not rely on a Web Driver, nor a network communication, it runs directly on the browser with its proper mechanism to **manipulate the DOM**.
- Is a **complete testing** framework:
  - is built on top of **Mocha (**Mocha is built-in).
  - has bundled tools including **Chai** for assertions, **Sinon.js for** stubbing and spying.
- Easy to setup.
- Cypress works for Firefox, Chrome-family browsers (including Edge and Electron) and more recently WebKit (Safari). https://docs.cypress.io/guides/guides/cross-browser-testing
  - Except for Electron, cypress requires installation (locally or in CI environment).

Cypress Test runner

Test Steps

Tested Application Display



More on Cypress Features

- Cypress supports all JQuery selectors.
- Includes a **travel-back-in-time** option, easing debugging.
- Like Jest, Cypress takes DOM **snapshots**, prior to each test step. (on the bottom to facilitate debugging).

Try it Yourself (it's very easy!)

1. **Install** Cypress with npm: https://docs.cypress.io/guides/getting-started/installing-cypress.html

2. **Write** your test(s): https://docs.cypress.io/guides/getting-started/writing-your-first-test.html#Console-output

3. Use Cypress Test Runner for debugging: https://docs.cypress.io/guides/getting-started/testing-your-app.html

4. Execute the tests from the CLI.

5. Remember to create and **integrate a cypress test in your CI/CD pipeline**

References (Optional Readings)

- Scenario Outlines with Cucumber (Scenario Templates): https://cucumber.io/docs/gherkin/reference/#scenario-outline
- Asynchronous code in Mocha: https://mochajs.org/#asynchronous-code
- Anti-pattern Feature-Coupled Step Definition with Cucumber: https://cucumber.io/docs/guides/anti-patterns/#feature-coupled-step-definitions
- Sinon.js to create test spies, mocks, and stubs: https://sinonjs.org/
- xUnit Test Patterns Book: http://xunitpatterns.com/Test%20Double%20Patterns.html
- Test Doubles: https://testing.googleblog.com/2013/07/testing-on-toilet-know-your-test-doubles.html
- Mocha vs Jest vs Jasmin (Picking a testing framework): https://proquest.tech.safaribooksonline.de/book/programming/javascript/9781788477321
- Learning BDD with Javascript (Cucumber.js, Gherkin): https://proquest.tech.safaribooksonline.de/book/programming/javascript/9781784392642
- Example of Cucumber usage with Ruby: https://www.agiratech.com/web-automation-testing-with-ruby-cucumber-watir/
- Building Enterprise Javascript Applications (Writing Unit/Integration Tests): https://proquest.tech.safaribooksonline.de/9781788477321/38b85b06_d091_4751_a2ac_32ca0f98f26b_xhtml#X2ludGVybmFsX0h0bWxWaWV3P3htbGlkPTk3ODE3ODg0NzczMiElMkYzOGI4NWIwNl9kMDkxXzQ3NTFfYTJhY18zMmNhMGY5OGYvNmJfeGh0bWwmcXVlcnk9KChCREQlMiBvZWFjdCkp
- Mocha: https://mochajs.org
- Testing with Chai and Mocha: https://proquest.tech.safaribooksonline.de/book/operating-systems-and-server-administration/9781788835770/the-need-for-testing-lambda-function/204792ad_01b4_4613_87b0_6ca12fcb30c7_xhtml?
- Jest BDD Unit Test from Facebook (React Building Modern Web Applications): https://proquest.tech.safaribooksonline.de/book/web-development/9781786462268/7dot-not-reinventing-the-wheel-tools-for-functional-reactive-programming/ch07s06_html
- Selenium: https://proquest.tech.safaribooksonline.de/book/software-engineering-and-development/software-testing/9781788473576
- Selenium vs Cypress: https://crossbrowsertesting.com/blog/test-automation/selenium-vs-cypress/
- Cypress + Cucumber: https://medium.com/@itortv/how-to-integrate-cypress-and-cucumber-in-your-development-flow-in-just-a-few-weeks-96a46ac9165a
- Jquery selectors: https://www.tutorialsteacher.com/jquery/jquery-selectors.