JS testing

tsevdos.me / @tsevdos

Agenda

- what is testing
- js / node testing
- jest
- testing basics

Rules

Feel free to interrupt me for:

- questions
- relevant comments

Testing 101

- any QA / tester in the room?
- anyone who write tests?
- has anyone break a build?
- has anyone break the production?
- how much harm have you done to your company / users?

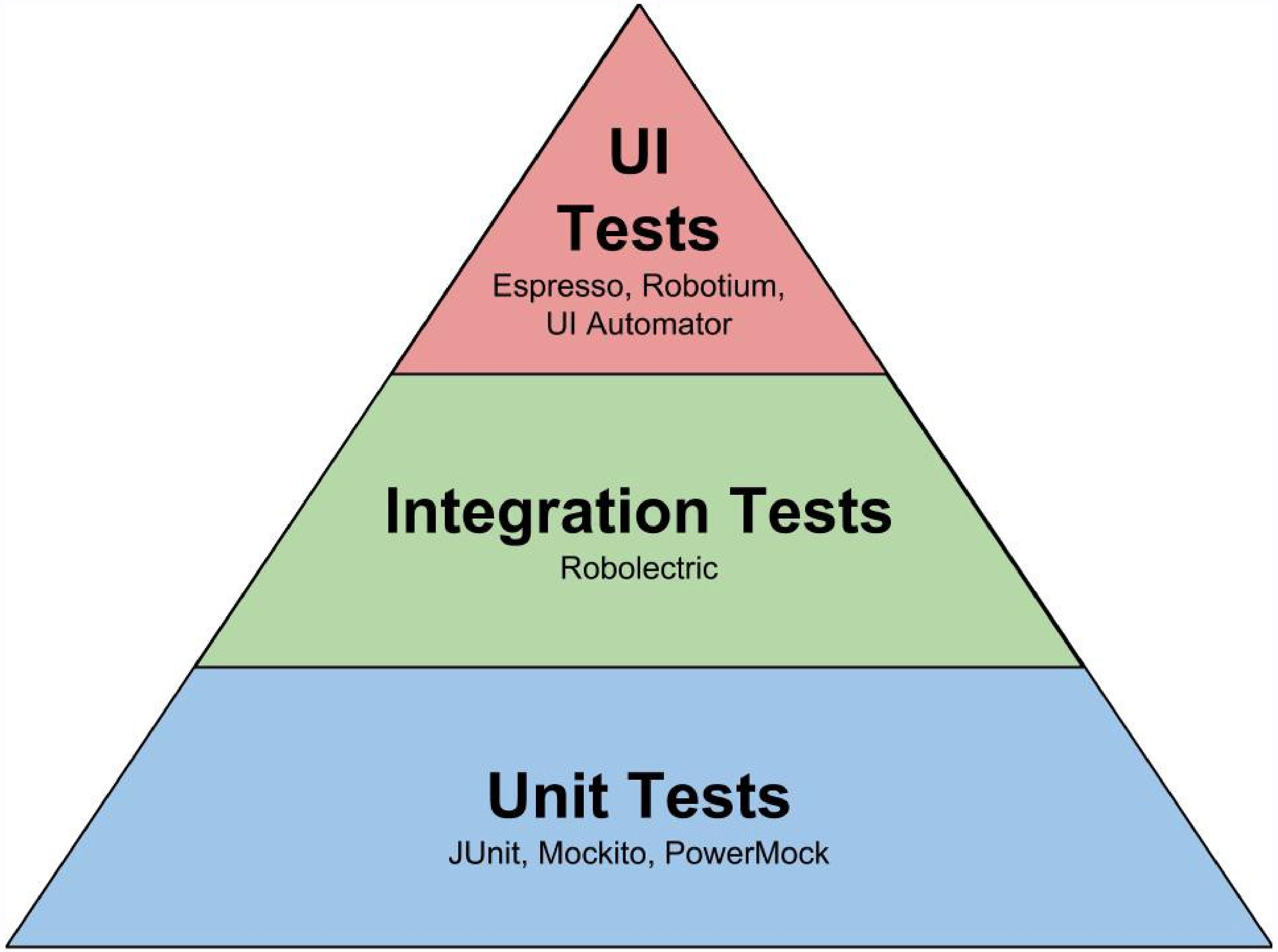
Testing 101: why we test?

- confidence
  - prove our code works
  - refactoring / easier and faster additions / changes
- code quality and design
- documentation
- think about issues and edge cases
- reduce technical debt

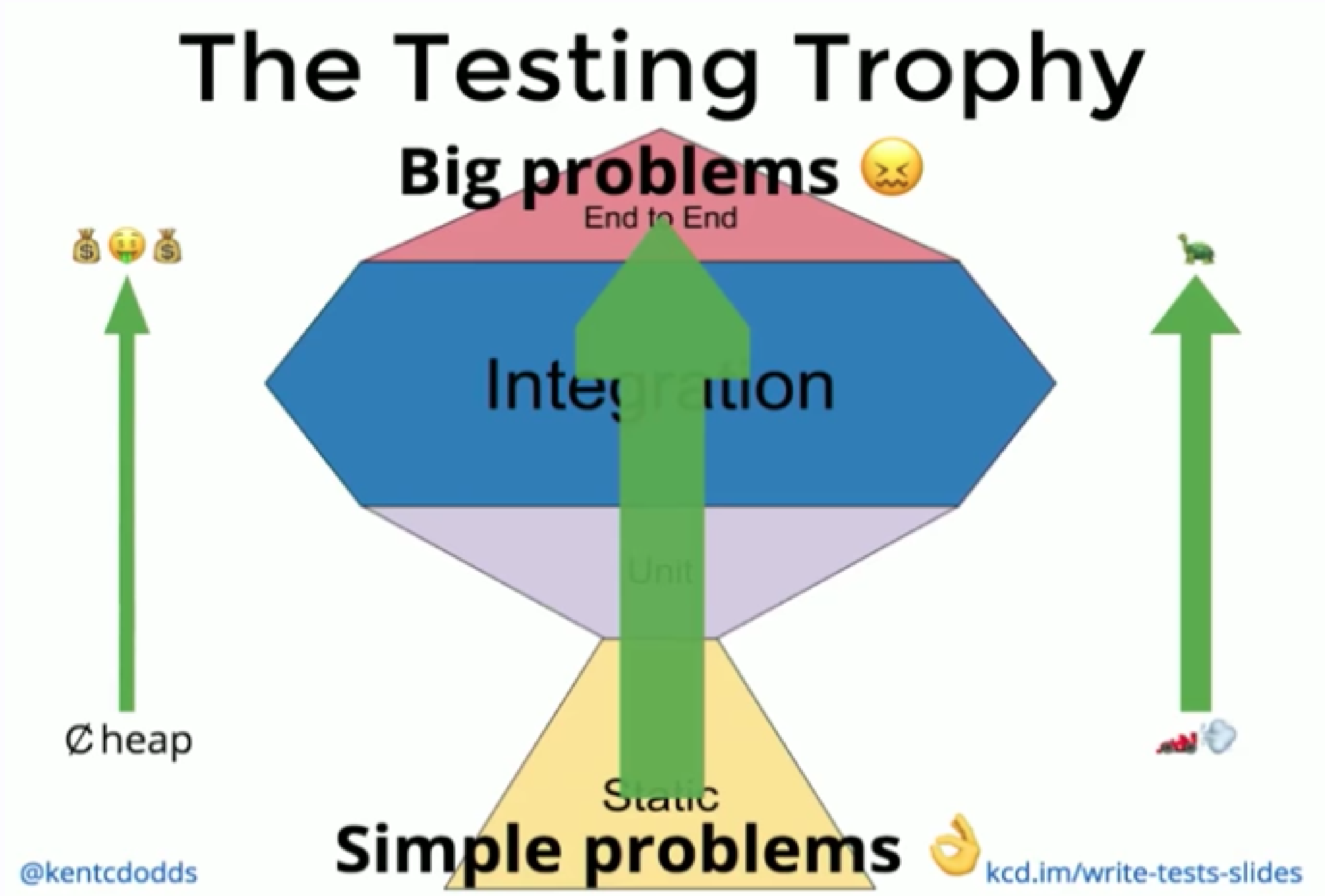
Types of tests

- unit tests: test one isolated unit / piece of code
- integration tests: test the combination of features
- end-to-end tests: test a full interaction path in your app

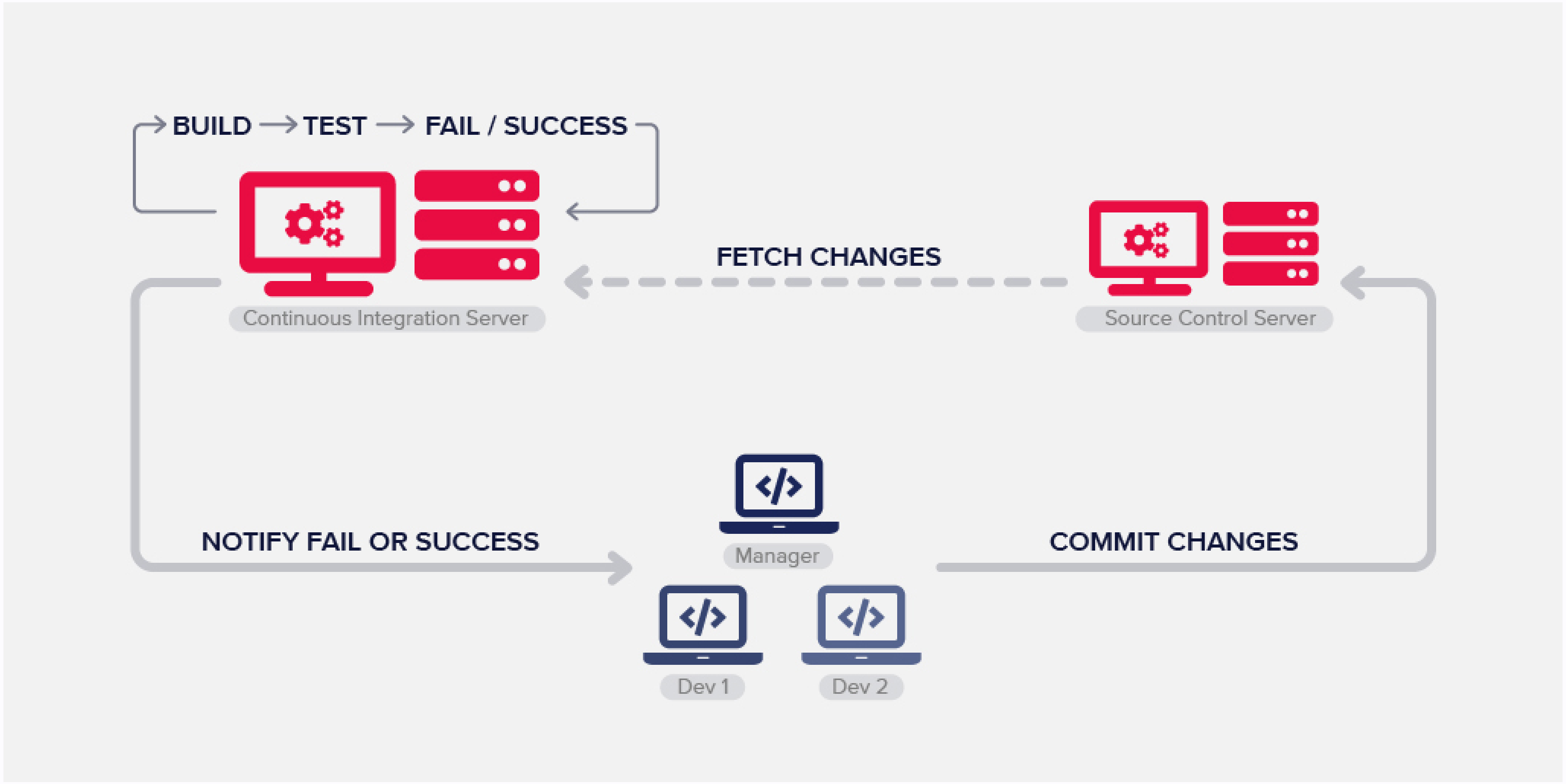
Testing pyramid



Testing Trophy



Continuous Integration



What CI does for us?

- detect and fix issues early (hopefully)
- avoid "integration hell"
- improve quality and testability
- continuous (automated) feedback
- code metrics
- increase transparency and communication
- running tests frequently
- confidence you’re building a solid solution

Testing tools: Test runner

- executes tests and summarizes the results in the terminal
- Jest, Mocha, Jasmine, AVA, etc.

Testing tools: Assertion library

- provide BDD and / or TDD styles for testing
  - it('works', () => { expect(2).toEqual(2); })
  - test('works', () => {expect(2).toEqual(2); })
- Jest, Chai, Jasmine etc.

Jest

- not just a test runner
- super fast (runs tests in parallel using workers)
- built-in mocks and spies
- built-in code coverage report
- snapshot testing
- zero configuration
- many more...

Usefull jest commands / flags

- `jest` (runs the tests)
- `jest --watch`
- `jest --coverage`

Writing Jest tests

1. TDD / BDD syntax
2. skipping tests
3. setup helpers

`./src/examples/00/writingTests.test.js`

Jest matchers: core

all matchers

- expect(value).toBe(value)
- expect(value).toEqual(value)
- expect(value).not.toBe(value)

Jest matchers: strings

- expect(value).toMatch(regexpOrString)

Jest matchers: numbers

- expect(value).toBeLessThan(number)
- expect(value).toBeLessThanOrEqual(number)
- expect(value).toBeGreaterThan(number)
- expect(value).toBeGreaterThanOrEqual(number)

Jest matchers: booleans

- expect(value).toBeTruthy()
- expect(value).toBeFalsy()

Jest matchers: objects

- expect(value).toBeInstanceOf(Class)
- expect(value).toMatchObject(object)
- expect(value).toHaveProperty(keyPath, value)

Jest matchers: arrays

- expect(value).toContain(item)
- expect(value).toHaveLength(number)

Jest matchers: misc

- expect(value).toBeUndefined()
- expect(value).toBeDefined()
- expect(value).toBeNull()
- expect(value).toBeNaN()

Jest matchers: errors

- expect(value).toThrow(error)

Jest matchers

`./src/examples/00/matchers.test.js`

Unit testing phases

1. setup / arrange
2. act
3. assert
4. (teardown?)

Unit test examples

- fn `./src/examples/01/wordCount.js`
- class `./src/examples/02/Calculator.js`

Unit test exercises

(100% code coverage)

- fn `./src/examples/03/isLeapYear.js` in order to pass all tests.
- class `./src/examples/04/Person.js`

Asynchronous testing

async code

- HTTP requests
- DB
- external services

Asynchronous testing examples

- `fn` `./src/examples/05/getHTMLUserTodos.js`
- `class` `./src/examples/06/asyncPerson.js`

Asynchronous testing exercises

(100% code coverage)

- fn `./src/examples/07/getHtmlTodo.js`
- class `./src/examples/08/User.js`

Asynchronous testing drawbacks

- don't block the tests (especially the unit tests)
- unit tests MUST be fast

Mocks and mocking

Mocking: The concept of mocking is primarily used in unit testing. You’d use mocking for isolating code by simulating the behavior of real objects and replacing the real object with the mocked object or function. For example, you can use a mock to make a function throw an error to evaluate how the function you’re testing handles this error.

Spies and spying

Spying: A spy allows you to catch function invocations so you can later verify if the object got called with the right arguments. A spy won’t change the behavior of a function.

Stubs and stubbing

Stubbing: Stubs are similar to spies. Instead of spying on a function, you can use a stub to control the behavior of a function. For example, a function makes an HTTP call to an external API. As you want predictable behavior for unit tests that don’t rely on unpredictable outcomes from an external API, we can stub the API call with a predefined value to make the test predictable again.

Jest's mocked function

```
const mockFn = jest.fn();
```

Mocking return values and promises

```
const mockFn = jest.fn();
mockFn.mockReturnValue(42);


// mock promises (async / await)
const mockAsyncFn = jest.fn();
mockAsyncFn.mockResolvedValue('Async Result');
```

Mocking implementation

```
const mockFn = jest.fn(() => 'Mocked Implementation');
```

Checking calls

```
const mockFn = jest.fn();

mockFn('arg1', 'arg2');

expect(mockFn).toHaveBeenCalledTimes(1);
expect(mockFn).toHaveBeenCalledWith('arg1', 'arg2');
```

Resetting a mock

```
const mockFn = jest.fn();

mockFn();

expect(mockFn).toHaveBeenCalled();

mockFn.mockReset();

expect(mockFn).not.toHaveBeenCalled();
```

Jest mock a module

```
jest.mock("../path/to/module");
```

Jest mock with specific implemetation

```
jest.mock("../path/to/module", () => ({
  ...jest.requireActual("../path/to/module"),
  getAge: jest.fn().mockReturnValue(42);
  getTodos: jest.fn().mockResolvedValue([
    { id: 1, title: "Learn React" },
    { id: 2, title: "Go to Code.Hub" },
    { id: 3, title: "Go out for a drink" },
  ]),
}));
```

Jest mocks examples

- fn `./src/examples/09/getHTMLUserTodos.js`
- class `./src/examples/10/asyncPerson.js`

Jest mocks exercises

(using mocks - 100% code coverage)

- fn `./src/examples/11/getHtmlTodo.js`
- class `./src/examples/12/User.js`

That's all folks

Questions / Discussions?