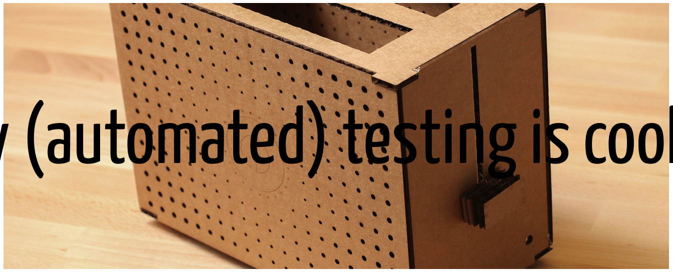

Why (automated) testing is cool ✌️

@adrienjoly

Outline

1. What? Who? Why?
2. Types of tests
3. Tips & tricks

Disclaimer

- Terminology and opinions can vary
- This content of this presentation is based on my experience

What is automated testing for

- Goal: the software works as expected, now and in the future
- Hint: code without tests = legacy code

Who should test

- Anybody who writes evolving software
- Systems that are hard (or expensive) to debug and update (e.g. hardware)
- Web: back-end & front-end

Why testing is good

💎 Sustainable quality and confidence in the codebase

😌 More doc. + fewer bugs ⇒ peace

⭐ Become a better engineer

Why testing is cool

✅ Watching tests pass = satisfying

👹 Hack without fear

🎮 Write code that challenges code = fun

Types of tests

🐜 Unit testing

📦 Functional testing

👫 Integration testing

🐜 Unit testing

- Goal: the system's (pure) functions are reliable
- How: expected outputs for each provided input
- Characteristics: simple to write, fast to run

🐜 Unit test

```
// file: unit.test.js

describe('parseInt', () => {
  it('turns "01" to 1', () => {
    expect(parseInt('01')).toBe(1);
  });
});
```

🐜 Unit test: let's run it!

```
> jest tests/unit.test.js

 PASS  tests/unit.test.js
  parseInteger
    ✓ turns "01" to 1 (2ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.835s, estimated 1s
Ran all test suites.
```

📦 Functional testing

- a.k.a. End-to-end testing
- a.k.a. Acceptance testing
- a.k.a. UI testing

📦 Functional testing

- Goal: the final product does what it's supposed to do
- How: consider the system as a black box, test like a user
- Characteristics: UI tests can be slow and flaky to run

📦 Functional testing

Example: the `/crawled` API endpoint

```
// HTTP GET /crawled => JSON response:

{
  pages: [
    'http://example.com/test-page.html',
    'http://example.com/test-page-2.html'
    // ...
  ]
}
```

📦 Functional testing

A functional test of the `/crawled` API endpoint:

```
// file: functional.test.js

describe('api', () => {
  it('returns the list of crawled pages', async () => {
    await crawler.indexSite('http://example.com');
    // after crawling, check that the page was indexed
    const res = await fetch('http://localhost:8000/crawle
    const json = await res.json();
    expect(json.pages[0]).toMatch(/test-page.html/);
  });
});
```

👫 Integration testing

- Goal: components behave as expected
- How: inject mocks and spies into the system
- More complex, slower to write, need more maintenance

👫 Integration testing

Mocking Algolia's search index component

```
// file: __mocks__/algolia.js

module.exports = {
  objects: [],
  addObject(obj){
    this.objects.push(obj);
  },
  search(query){
    return {
      hits: this.objects
    };
  }
};
```

👫 Integration testing

```
// file: integration.test.js

jest.mock('algolia'); // will inject __mocks__/algolia.js
const algolia = require('algolia');
const worker = require('../src/crawler-worker');

describe('crawler', () => {
  it('indexes one record from test-page.html', () => {
    worker.indexPage('http://localhost/test-page.html');
    expect(algolia.search().hits).toHaveLength(1);
  });
});
```

Let's run the tests: `npm test` ✅

```
> jest --verbose

 PASS  tests/functional.test.js
  api
    ✓ returns the list of crawled pages (23ms)

 PASS  tests/unit.test.js
  parseInteger
    ✓ turns "01" to 1 (2ms)

 PASS  tests/integration.test.js
  crawler
    ✓ indexes one record from test-page.html (1ms)

Test Suites: 3 passed, 3 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.943s, estimated 1s
Ran all test suites.
```

The extra mile: `package.json`

```json
{
  "name": "my-awesome-product",
  "scripts": {
    "test": "jest $@",
    "test:unit": "jest tests/unit.test.js",
    "test:integration": "jest tests/integration.test.js",
    "test:functional": "jest tests/functional.test.js"
  },
  "dependencies": {},
  "devDependencies": {
    "jest": "23.0.0"
  }
}
```

Tips & tricks

- You don't need 100% coverage
- Golden path first
- Use a Continuous Integration (CI) system
- Leave no trace
- Beware flaky tests
- Predictability: no random, no waiting, use fixed dates
- Output of failing test = title of a Github issue
- One PR = at least one test
- Write a failing test before coding
- For each new bug, write a regression test

Tips & tricks // We're done! ✅

- You don't need 100% coverage
- Golden path first
- Use a Continuous Integration (CI) system
- Leave no trace
- Beware flaky tests
- Predictability: no random, no waiting, use fixed dates
- Output of failing test = title of a Github issue
- One PR = at least one test
- Write a failing test before coding
- For each new bug, write a regression test

👉 Sample tests: `http://bit.ly/AJTEST`

🐦 twitter.com/adrienjoly

👋 PS: We're hiring! Ask me about Algolia.