Testing

for JavaScript developers

Martin Kapal

What is the purpose of testing?

- Verify that the product does what it is supposed to do
- Quality software ➡ happy customers

Types of testing

Testing can be categorised by...

Types of testing

Level

unit testing
integration testing
system testing

Automation

manual 💪
automated 🕹️

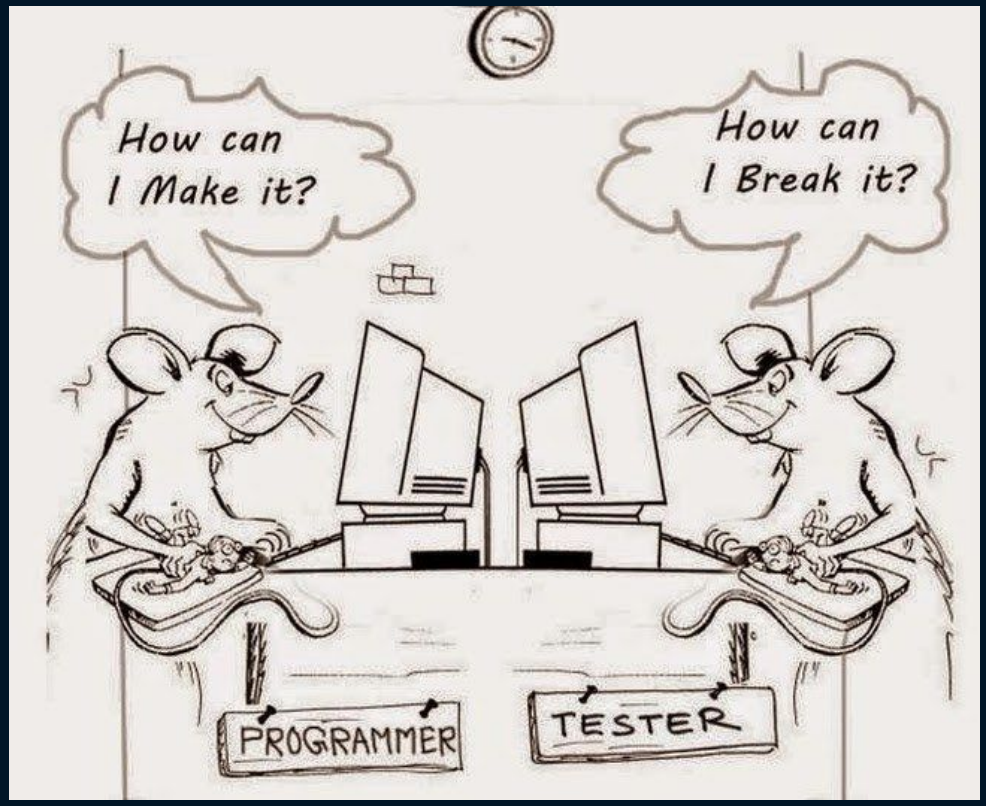
Approach

white-box 👀
black-box 🙈

Requirements

functional
non-functional
(performance, usability...)

* underlined types are usually done by us developers


How can
I Make it?
How can
I Break it?
PROGRAMMER
TESTER

Unit testing

Why should we write unit tests?

- Verify small units of code - functions, classes, components
- Better understand the logic - self-documentation
- Prevent breaking things during refactoring

Unit test structure

- **Arrange** - get the system into the desired state
- **Act** - perform the action you want to test
- **Assert** - verify that the result is what you expected

What to test?

- happy paths 😄 - how a function is usually used
- sad paths 😢 - edge cases, invalid inputs

What makes a good test?

- fast
- isolated
- repeatable
- self-validating
- timely

What makes a good test?

Fast

- avoid using real data sources & services (database, network)
- use *mocking* to create fake external dependencies
- keep performance in mind

What makes a good test?

Isolated

- test can run in any order, at any time
- only one thing should be tested in one test
- one reason to make the test fail

What makes a good test?

Repeatable

- you will always get the same result when running the same test

What makes a good test?

Self-validating

- tests can be run automatically

What makes a good test?

Timely

- you should always write tests
- you should run tests often

So how do I write tests for my app?

Use a test framework!

Test frameworks for JavaScript

- Jest
- Jasmine
- Mocha
- ...and more

Code example time! 👨🏻‍💻

```
function divide(a, b) {
  return a / b;
}
```

Let's test this using Jest

divide.test.js

```
describe('My awesome division function', () => {
  it('should return 0.5 when dividing 1 by 2', () => {
    const result = divide(1, 2);
    expect(result).toEqual(0.5);
  });
});
```

Hurray, the test has passed! 🎉

```
PASS src/__tests__/divide.tests.ts
  My awesome division function
    ✓ should return 0.5 when dividing 1 by 2 (2 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
```

```
describe('My awesome division function', () => {
  it('should return 0.5 when dividing 1 by 2', () => {
    const result = divide(1, 2);
    expect(result).toEqual(0.5);
  });

  it('should return null if trying to divide by zero', () => {
    const result = divide(1, 0);
    expect(result).toEqual(null);
  });
});
```

The tests have failed now 👎

```
FAIL src/__tests__/divide.tests.ts
  My awesome division function
    ✓ should return 0.5 when dividing 1 by 2
    × should return null if trying to divide by zero (1 ms)

  ● My awesome division function › should return null if trying to divide by zero

    expect(received).toEqual(expected) // deep equality

    Expected: null
    Received: Infinity

      11 |   it('should return null if trying to divide by zero', () => {
      12 |     const result = divide(1, 0);
    >           expect(result).toEqual(null);
         |                    ^
      14 |   });
      15 | });
      16 |

      at Object.<anonymous> (src/__tests__/divide.tests.ts:13:20)
```

Let's fix that...

```
function divide(a, b) {
  if (b === 0) {
    return null;
  }

  return a / b;
}
```

Run our tests again...

Nice 👌

```
PASS src/__tests__/divide.tests.ts
  My awesome division function
    ✓ should return 0.5 when dividing 1 by 2 (2 ms)
    ✓ should return null if trying to divide by zero (1 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
```

Test-driven development (TDD)

Write tests first, then implement the functionality

1. Write a test that fails
2. Write the simplest code that passes the new test
3. Refactor existing code
4. ...repeat

Can I test React components?

Yes!

React Testing Library

```jsx
import { render, screen } from '@testing-library/react';
import { Button } from './Button.jsx';

it('renders a disabled button if a prop "disabled" is passed', () => {
  render(<Button disabled>Click me</Button>);

  const button = screen.getByRole('button');

  expect(button).toBeDisabled();
});
```

https://testing-library.com/react

Questions?

Thank you!