Let’s talk about unit tests.

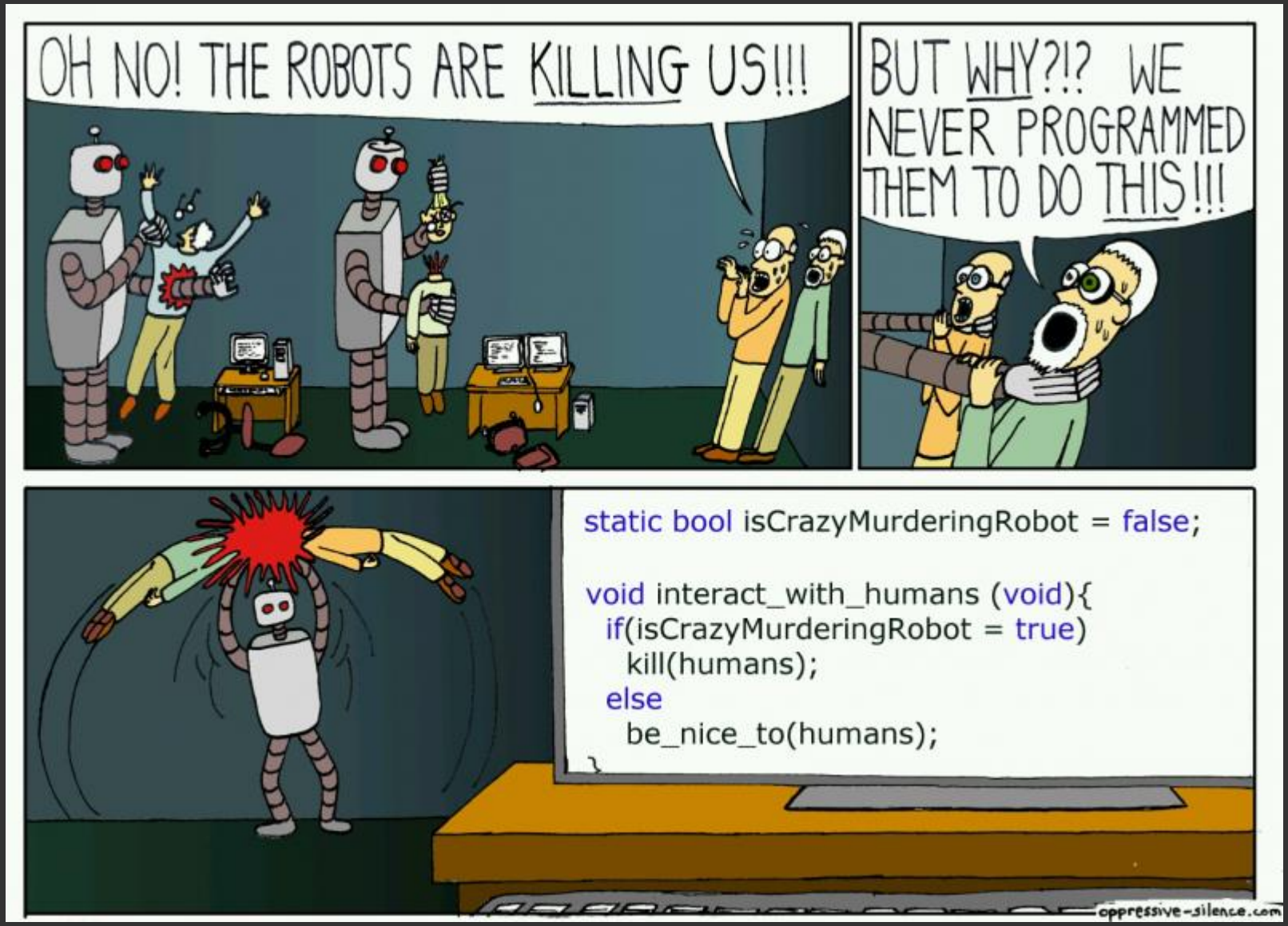
April 2018

Danielo **JEAN-LOUIS**
Front-end developer

What’s unit tests?

*“Method of testing each **unit** of code to **test** if they work well as an **unit**”.*

Why unit tests?



Why unit tests?

- Reduce the number of tickets / bugs
- Improve code quality / reduce complexity
- Encourage modularity
- Ensure old functionalities behaviors even if the code change (!)
- Document code

Sources

- https://medium.com/javascript-scene/what-every-unit-test-needs-f6cd34d9836d

But your code has to be ready for it

Pure function is the first step

Pure function

- No side effects
  - Doesn’t update variables outside its own scope
- Always return the same results
  - E.g.: Math.random() (js) is impure

Sources

- http://www.nicoespeon.com/en/2015/01/pure-functions-javascript/
- https://medium.com/javascript-scene/master-the-javascript-interview-what-is-a-pure-function-d1c076bec976

Example

Impure function

```
let a = 2;
export const sumImpure = (b) => {
    a = a + b; // Access var outsite func
    return a
}
```

Let’s test it...

with Jest

Jest

- Unit test framework
- Created by facebook
- Agnostic but made to work well with React
- Inspired by Jasmine
- Allows snapshot
- Jest = Jasmine + sinon + istanbul

Sources

- https://facebook.github.io/jest/

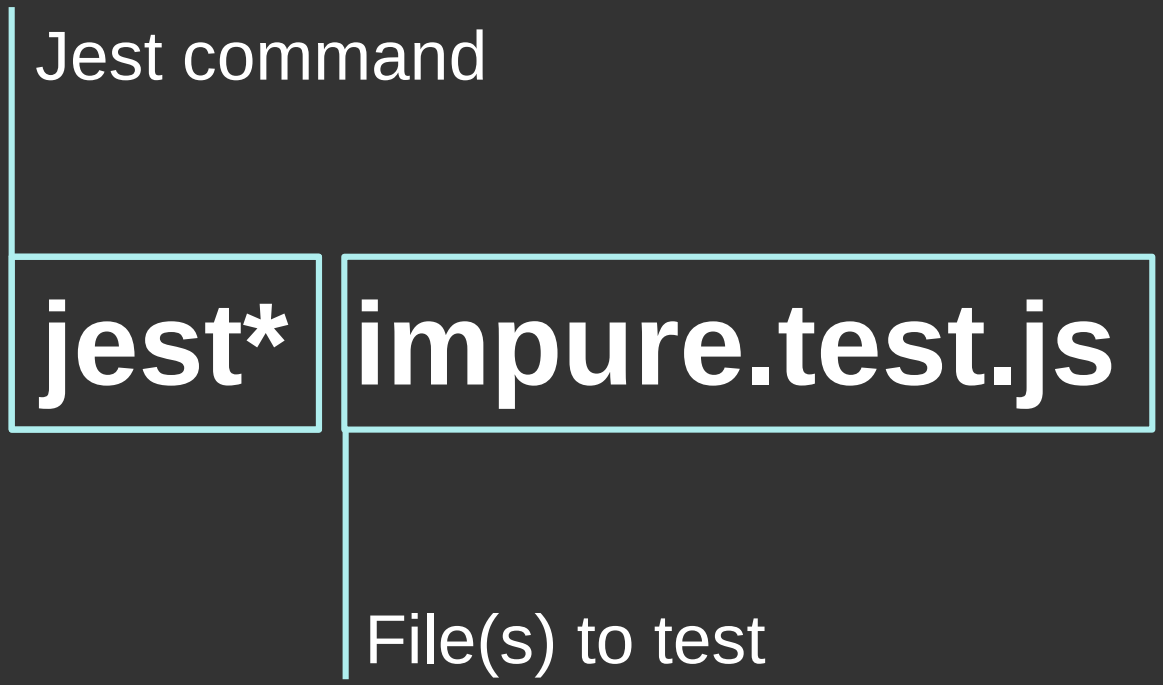
Now test our function

Let’s analyze the command

jest* impure.test.js

***** Assuming jest is installed in global or run from our npm scripts.
If not you’ll need to prefix it with ‘node_modules/.bin/’

Let’s analyze the command



***** Assuming jest is installed in global or run from our npm scripts.
If not you’ll need to prefix it with ‘node_modules/.bin/’

Example

Pure function

```
export const sumPure = (a=0, b=0) => {
    return a + b;
}
```

It works well but...

Did we manage every case?

Code coverage

- Works hand in hand with unit tests
- Detects useless code
- Forces to manage every case / scenario
- Returns the percentage of code executed

- “--coverage” param with Jest
- Jest relies on istanbul for it

Let’s check the coverage

Code coverage details

Statements (Stmts):

Proportion of statements / instructions executed

Examples:

```
const a = 42; // It's a statement
const b = 42; console.log(b) // It's two statements
```

Branches (Branch):

Conditional statements executed (if / else, switch...)

Sources

- https://www.w3schools.com/js/js_statements.asp
- http://2ality.com/2012/09/expressions-vs-statements.html
- https://github.com/dwyl/learn-istanbul
- https://github.com/gotwarlost/istanbul/issues/639

Code coverage details

Functions (Funcs):

Proportion of functions / methods called

Lines (lines):

Proportion of lines executed

Sources

- http://2ality.com/2012/09/expressions-vs-statements.html
- https://github.com/dwyl/learn-istanbul

Let’s improve the coverage

Code coverage details

- /coverage/ to get more details about the latest code coverage

100 % code coverage,
it’s not a dream.
It can be a reality.

Best practices

- Be explicit (You know what you’re testing)
- Don’t try to replace QA / functional tests
- Add them to your precommit
- TDD (Write your test then code)
- Define a naming convention
- Create a mocks folder for your mocked data
- Considerate **unit tests** as a task

Best practices - Mocking

```
export const listItemsTpl = (apiRes) => (
    apiRes
        .filter(entry => entry.name)
        .map(entry => (
            `<li>${entry.name}</li>`
        ))
)
```

Best practices - Mocking

- Avoid (real) API calls
- Improves unit tests' speed
- Enforce the concept:
  One function, one functionality / Single Responsibility Principle

Sources

- https://en.wikipedia.org/wiki/Single_responsibility_principle

Conclusion

- **Unit testing** allows to improve code quality / reduces number of bugs
- **Code coverage** detects useless code
- Unit tests **mustn’t** replace QA. Functional tests exist for this.

More sources

- Presentation + examples :
https://github.com/DanYellow/presentations/tree/master/unit-tests

Questions ?