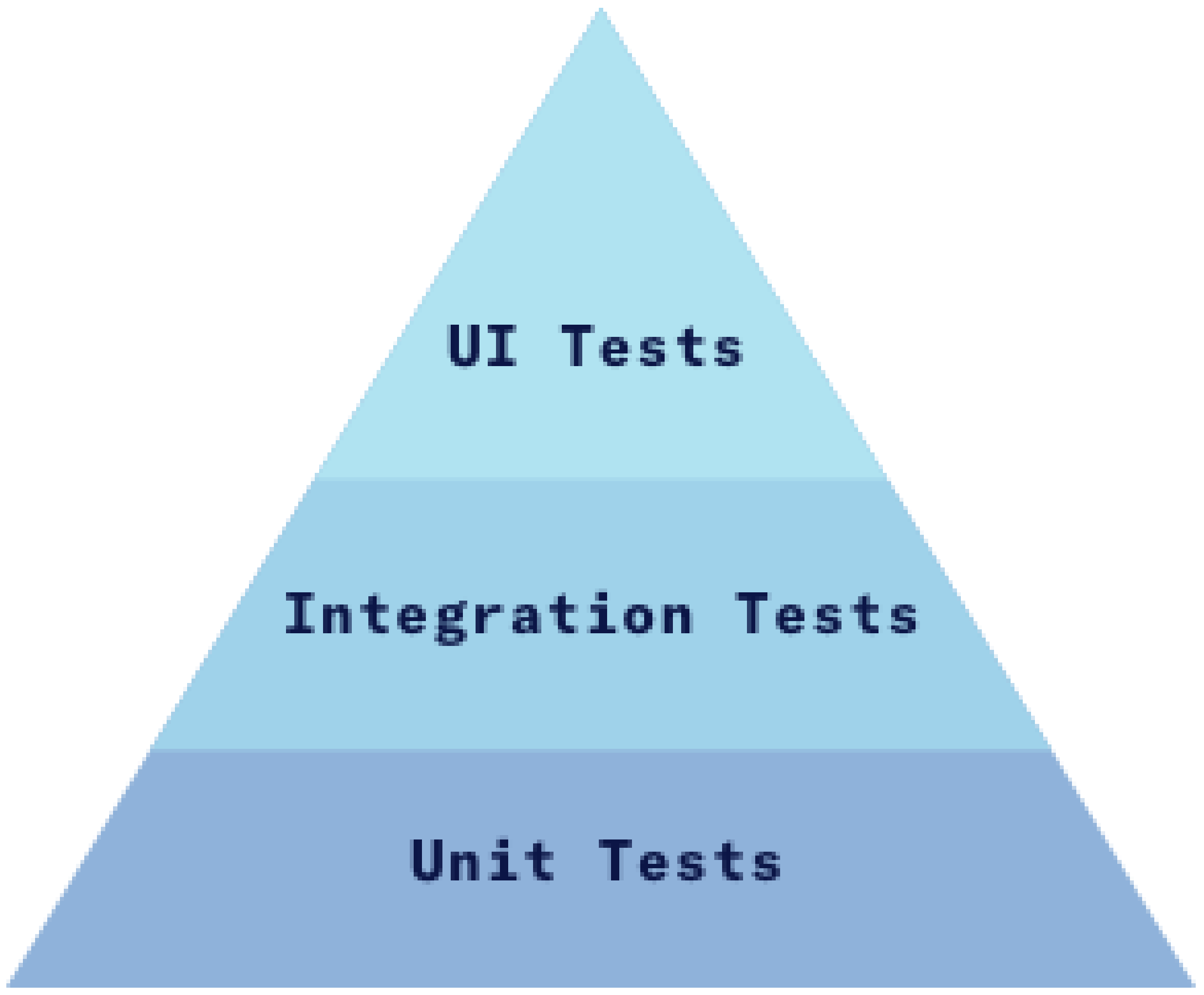
Тестирование Лекция 4

Зачем тестировать ПО?

Пирамида **тестов**



Unit **Testing**

Юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

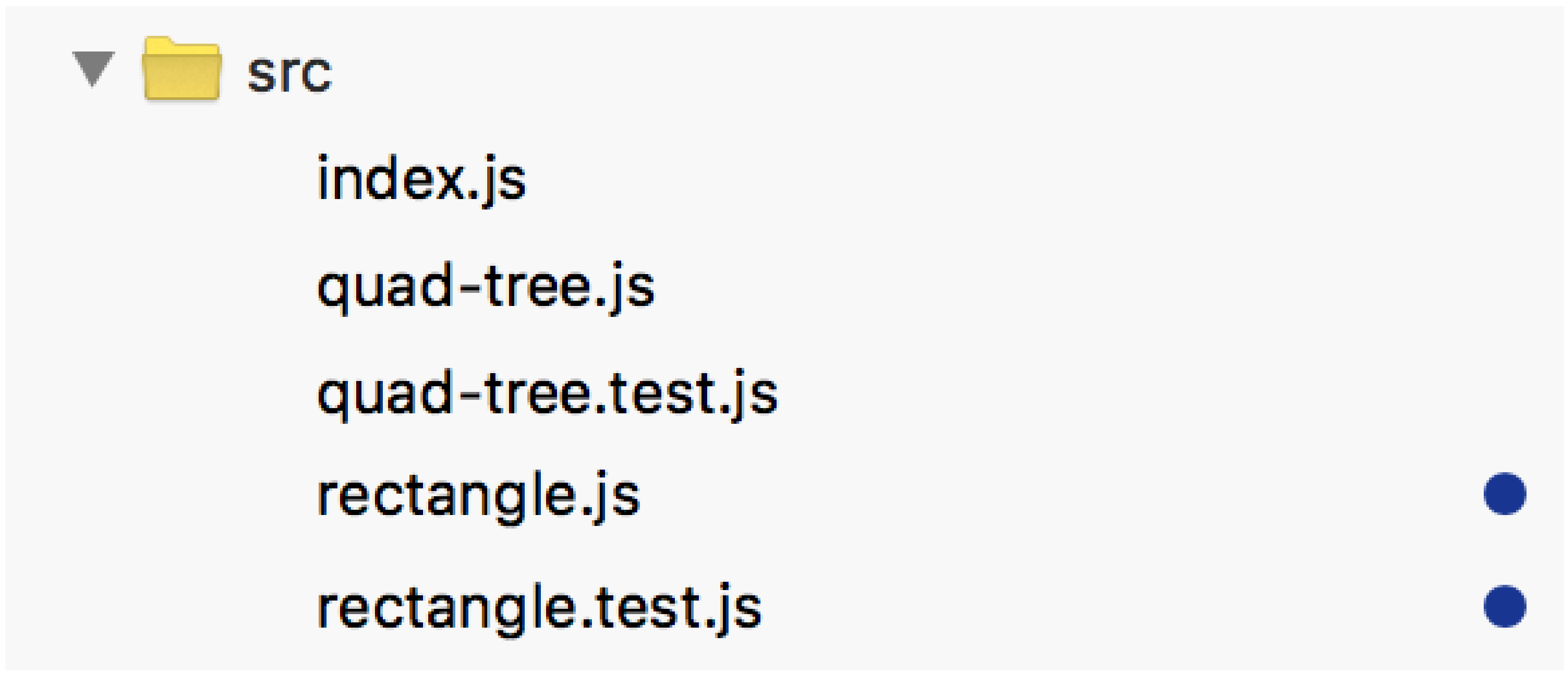
Тесты позволяют проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже протестированных местах программы.

Jest

```
npm install —-save-dev jest
```

Настройка окруженя

```
"scripts": {
  "coverage": "jest --coverage",
  "start": "parcel index.html",
  "test": "jest --verbose",
  "watch": "jest --watch"
},
```



```
▼ src
    index.js
    quad-tree.js
    quad-tree.test.js
    rectangle.js
    rectangle.test.js
```

Пример теста

```
import Rectangle from './rectangle'

describe('Rectangle.contains()', () => {
  it('should returns true if p is inside rect', () => {
    const rect = new Rectangle(0, 0, 3, 2)
    expect(rect.contains({x: 1, y: 1})).toBeTruthy()
  })
})
```

Пример теста

```
import Rectangle from './rectangle'

describe('Rectangle.contains()', () => {
  it('should returns true if p is inside rect', () => {
    const rect = new Rectangle(0, 0, 3, 2)
    expect(rect.contains({x: 1, y: 1})).toBeTruthy()
  })
})
```

Пример теста

```
import Rectangle from './rectangle'

describe('Rectangle.contains()', () => {
  it('should returns true if p is inside rect', () => {
    const rect = new Rectangle(0, 0, 3, 2)
    expect(rect.contains({x: 1, y: 1})).toBeTruthy()
  })
})
```

Пример теста

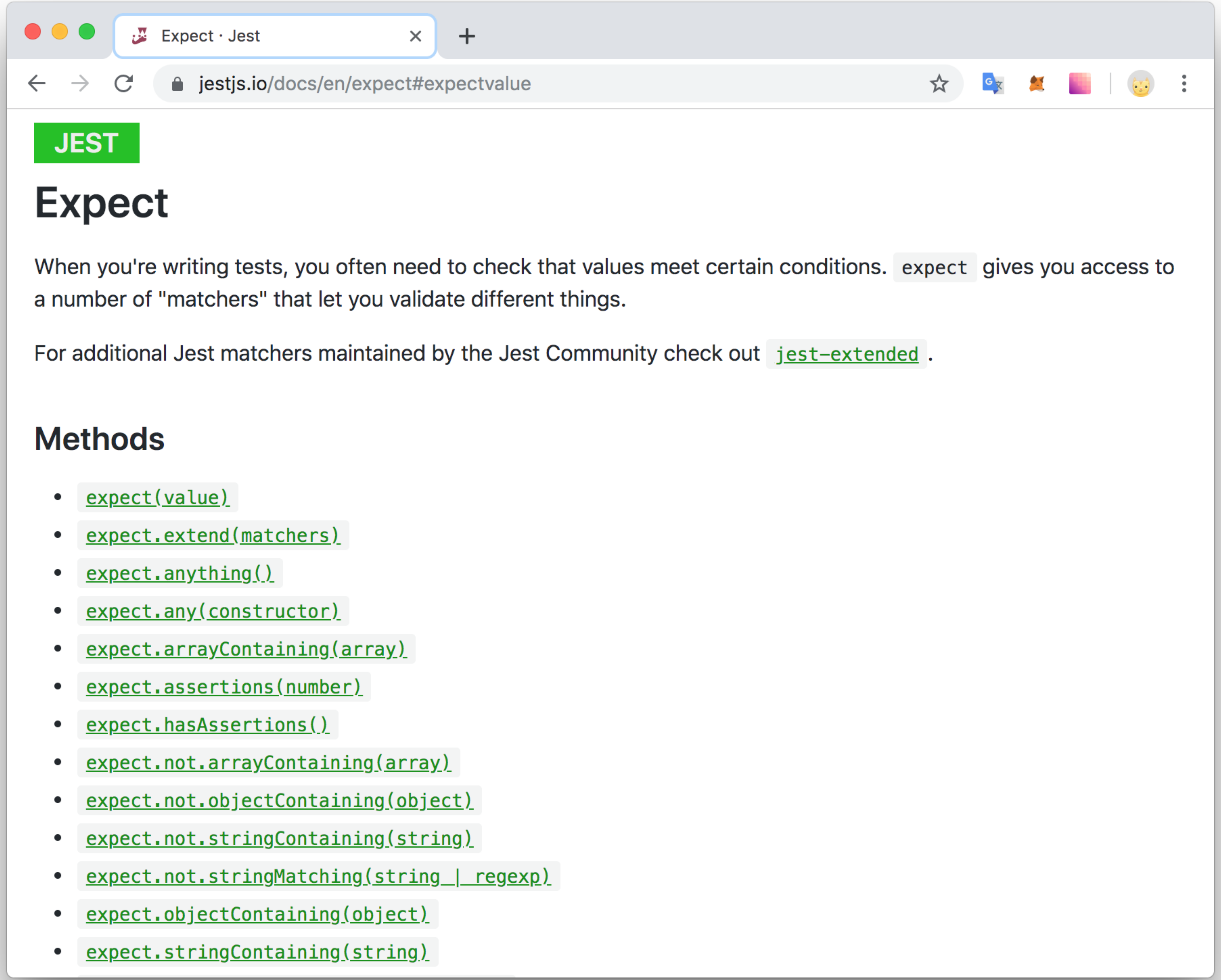
```
import Rectangle from './rectangle'

describe('Rectangle.contains()', () => {
  it('should returns true if p is inside rect', () => {
    const rect = new Rectangle(0, 0, 3, 2)
    expect(rect.contains({x: 1, y: 1})).toBeTruthy()
  })
})
```

Expect

expect - это библиотека **matcher**'ов для проверки выполнения условий.

```
expect(bestSfeduFaculty()).toBe('mmcs')
```

Аргумент, который принимает **expect** должен быть значением, которое генерирует ваш код, а любой аргумент для **match**'ера должен быть ожидаемым значением.



Expect · Jest

jestjs.io/docs/en/expect#expectvalue

JEST

Expect

When you're writing tests, you often need to check that values meet certain conditions. `expect` gives you access to a number of "matchers" that let you validate different things.

For additional Jest matchers maintained by the Jest Community check out `jest-extended`.

Methods

- `expect(value)`
- `expect.extend(matchers)`
- `expect.anything()`
- `expect.any(constructor)`
- `expect.arrayContaining(array)`
- `expect.assertions(number)`
- `expect.hasAssertions()`
- `expect.not.arrayContaining(array)`
- `expect.not.objectContaining(object)`
- `expect.not.stringContaining(string)`
- `expect.not.stringMatching(string | regexp)`
- `expect.objectContaining(object)`
- `expect.stringContaining(string)`

Примеры Matchers

```
expect(recived).toBe(value)
expect(recived).toBeFalsy()
expect(recived).toBeTruthy()
expect(recived).toBeNull()
expect(func).toThrow(error?)
```

Deep **Equal**

```
const obj1 = { a: 42, b: {c: 7} }
const obj2 = { a: 42, b: {c: 7} }

describe('hmmm', () => {
  it('should fail', () => {
    expect(obj1).toBe(can2)
  })
  it('should pass', () => {
    expect(obj1).toEqual(obj2)
  })
})
```

Deep **Equal**

```
const obj1 = { a: 42, b: {c: 7} }
const obj2 = { a: 42, b: {c: 7} }

describe('hmmm', () => {
  it('should fail', () => {
    expect(obj1).toBe(can2)
  })
  it('should pass', () => {
    expect(obj1).toEqual(obj2)
  })
})
```

Deep **Equal**

```
const obj1 = { a: 42, b: {c: 7} }
const obj2 = { a: 42, b: {c: 7} }

describe('hmmm', () => {
  it('should fail', () => {
    expect(obj1).toBe(can2)
  })
  it('should pass', () => {
    expect(obj1).toEqual(obj2)
  })
})
```

Compare **Arrays**

```
const arr1 = [{x: 0, y: 0}, {x: 3, y: 3}]
const arr2 = [{x: 3, y: 3}, {x: 0, y: 0}]


expect(arr1).toEqual(arr2) // ???
```

Compare **Arrays**

```
const arr1 = [{x: 0, y: 0}, {x: 3, y: 3}]
const arr2 = [{x: 3, y: 3}, {x: 0, y: 0}]


expect(arr1).toEqual(arr2) // failed
```

Compare **Arrays**

```
const arr1 = [{x: 0, y: 0}, {x: 3, y: 3}]
const arr2 = [{x: 3, y: 3}, {x: 0, y: 0}]


expect(arr1).toEqual(arr2) // failed
expect(arr1.sort()).toEqual(arr2.sort()) // passed
```

Custom Matchers

Jest - отличный тестовый фреймворк, в котором есть много полезных **API**. Однако бывают случаи, когда полезно иметь дополнительные **match**'еры.

```
describe('meaningOfLife', () => {
  it('should returns even result someday', () => {
    expect(meaningOfLife()).toBeEven()
  })
})
```

Custom Matchers

Jest - отличный тестовый фреймворк, в котором есть много полезных **API**. Однако бывают случаи, когда полезно иметь дополнительные **match**'еры.

```
describe('meaningOfLife', () => {
  it('should returns even result someday', () => {
    expect(meaningOfLife()).toBeEven()
  })
})
```

Custom Matchers

```
expect.extend({
  toBeEven(received) {
    const isEven = (received % 2 === 0)
    return { pass: isEven }
  }
})
```

Custom Matchers

```
expect.extend({
  toBeEven(received) {
    const isEven = (received % 2 === 0)
    return { pass: isEven }
  }
})
```

Custom Matchers

```
expect.extend({
  toBeEven(received) {
    const isEven = (received % 2 === 0)
    return { pass: isEven }
  }
})
```

Custom Matchers

```
expect.extend({
  toBeEven(received) {
    const isEven = (received % 2 === 0)
    return { pass: isEven }
  }
})

describe('meaningOfLife', () => {
  it('should returns even result someday', () => {
    expect(meaningOfLife()).toBeEven()
  })
})
```

Error Message

```
● meaningOfLife › should returns even result someday

  No message was specified for this matcher.

    15 | describe('meaningOfLife', () => {
    16 |   it('should returns even result someday', () => {
  > 17 |     expect(meaningOfLife()).toBeEven()
       |                            ^
    18 |   })
    19 | })
    20 |

    at Object.toBeEven (src/rectangle.test.js:17:29)

Test Suites: 1 failed, 1 passed, 2 total
Tests:       1 failed, 11 passed, 12 total
Snapshots:   0 total
Time:        2.589s
```

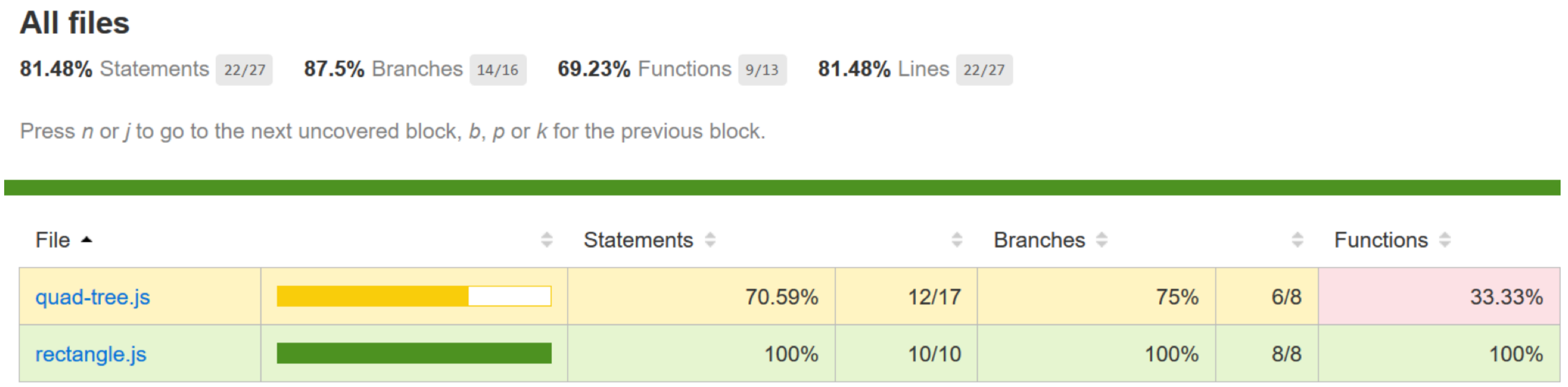
Custom Matchers

```
expect.extend({
  toBeEven(received) {
    const isEven = (received % 2 === 0)
    const toBeOrNotToBe = this.isNot
      ? 'not to be'
      : 'to be'
    const message = () =>
       `expected ${received} ${toBeOrNotToBe} even`

    return { pass: isEven, message: message }
  }
})
```

Error Message

```
  ● meaningOfLife › should returns even result someday

    expected 41 to be even

      15 | describe('meaningOfLife', () => {
      16 |   it('should returns even result someday', () => {
    > 17 |     expect(meaningOfLife()).toBeEven()
         |                            ^
      18 |   })
      19 | })
      20 |

      at Object.toBeEven (src/rectangle.test.js:17:29)

Test Suites: 1 failed, 1 passed, 2 total
Tests:       1 failed, 11 passed, 12 total
Snapshots:   0 total
Time:        2.707s
```

Coverage

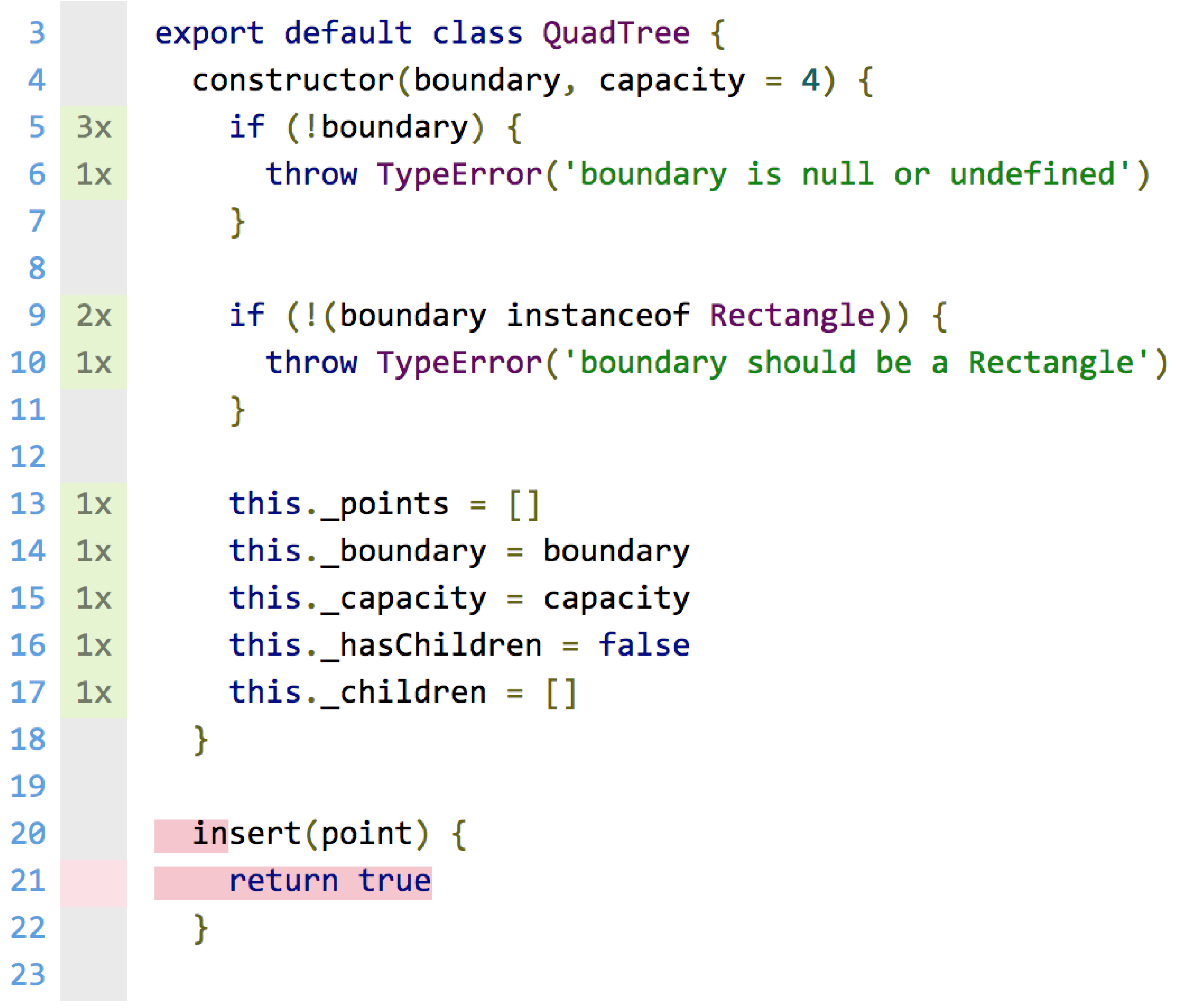


All files

81.48% Statements 22/27 **87.5%** Branches 14/16 **69.23%** Functions 9/13 **81.48%** Lines 22/27

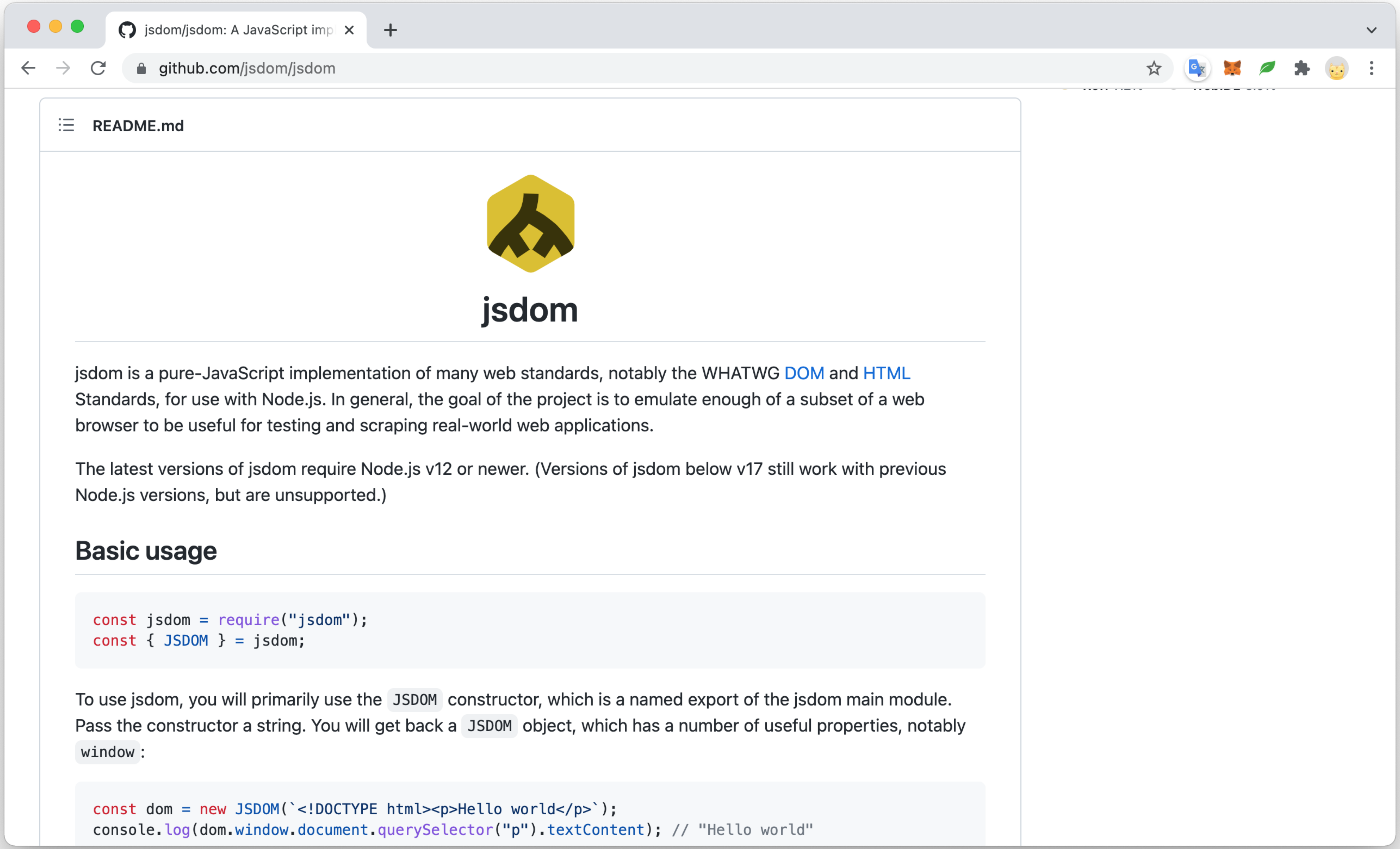
Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File ▲		Statements		Branches		Functions
quad-tree.js		70.59%	12/17	75%	6/8	33.33%
rectangle.js		100%	10/10	100%	8/8	100%



```
     export default class QuadTree {
       constructor(boundary, capacity = 4) {
3x       if (!boundary) {
1x         throw TypeError('boundary is null or undefined')
         }

2x       if (!(boundary instanceof Rectangle)) {
1x         throw TypeError('boundary should be a Rectangle')
         }

1x       this._points = []
1x       this._boundary = boundary
1x       this._capacity = capacity
1x       this._hasChildren = false
1x       this._children = []
       }

       insert(point) {
         return true
       }

```

jsdom



jsdom

jsdom is a pure-JavaScript implementation of many web standards, notably the WHATWG DOM and HTML Standards, for use with Node.js. In general, the goal of the project is to emulate enough of a subset of a web browser to be useful for testing and scraping real-world web applications.

The latest versions of jsdom require Node.js v12 or newer. (Versions of jsdom below v17 still work with previous Node.js versions, but are unsupported.)

Basic usage

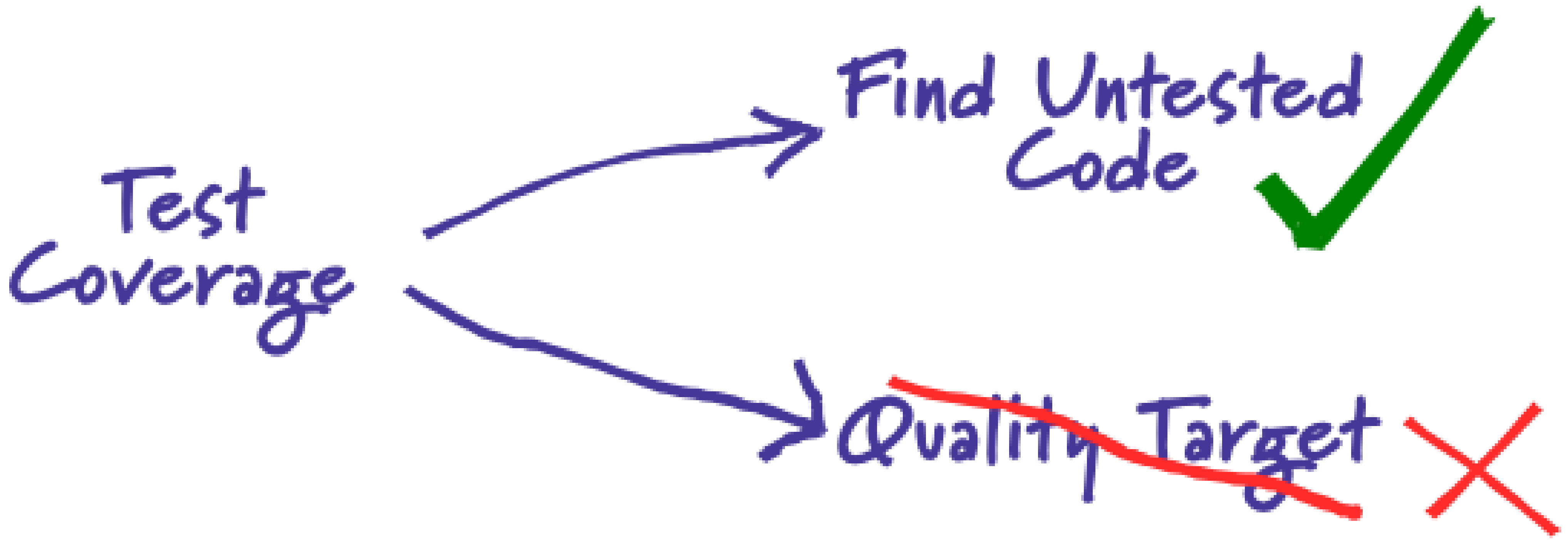
```
const jsdom = require("jsdom");
const { JSDOM } = jsdom;
```

To use jsdom, you will primarily use the `JSDOM` constructor, which is a named export of the jsdom main module. Pass the constructor a string. You will get back a `JSDOM` object, which has a number of useful properties, notably `window`:

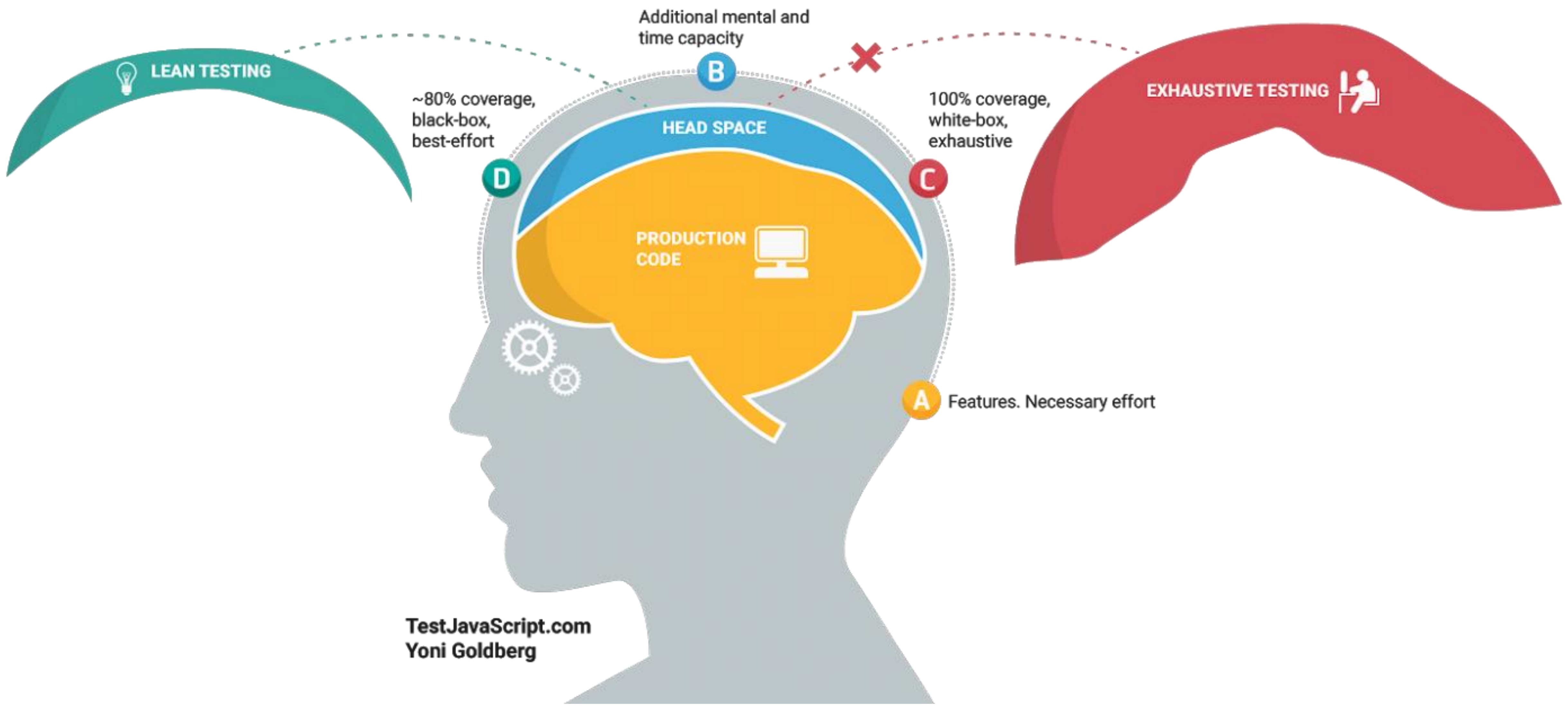
```
const dom = new JSDOM(`<!DOCTYPE html><p>Hello world</p>`);
console.log(dom.window.document.querySelector("p").textContent); // "Hello world"
```

Нужно ли нам **100%** покрытие тестами?

Картинка от **Фаулера**



Lean Testing


Additional mental and time capacity
B
LEAN TESTING
~80% coverage, black-box, best-effort
D
HEAD SPACE
100% coverage, white-box, exhaustive
C
EXHAUSTIVE TESTING
PRODUCTION CODE
A
Features. Necessary effort
TestJavaScript.com
Yoni Goldberg

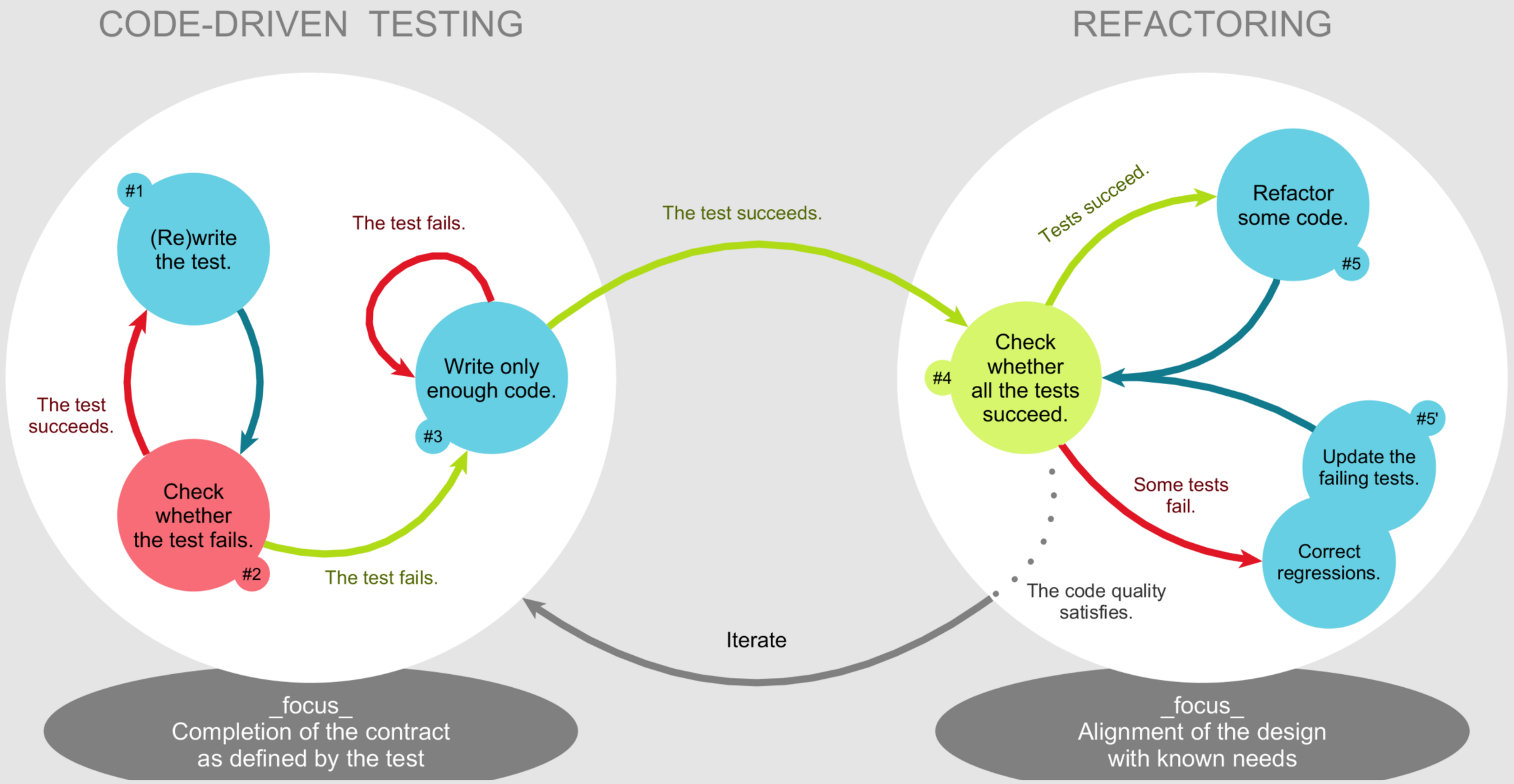
Тестовый код отличается от production-кода.

Тестовый код должен быть простым*.

Test **D**riven **D**evelopment

– Написать тесты
– Выполнить тесты и убедиться, что они не проходят (красные тесты)
– Написать минимальный код, который пройдет тесты
– Выполнить тесты и убедиться, что они проходят (зеленые тесты)
– Выполнить рефакторинг

Test Driven Development



Хорошие практики

– Короткие итерации
– Маленькие модули
– Самодокументируемые тесты
– Отделяйте общую настройку окружения от частной для теста

Плохие практики

- Тесты, зависящие от состояния, созданного другими тестами
- Зависимости между тестами
- Точное тестирование производительности или времени выполнения
- Проектирование всезнающих "оракулов"
- Тестирование реализации
- Медленные тесты

Материалы

https://martinfowler.com/bliki/TestCoverage.html

https://github.com/goldbergyoni/javascript-testing-best-practices/