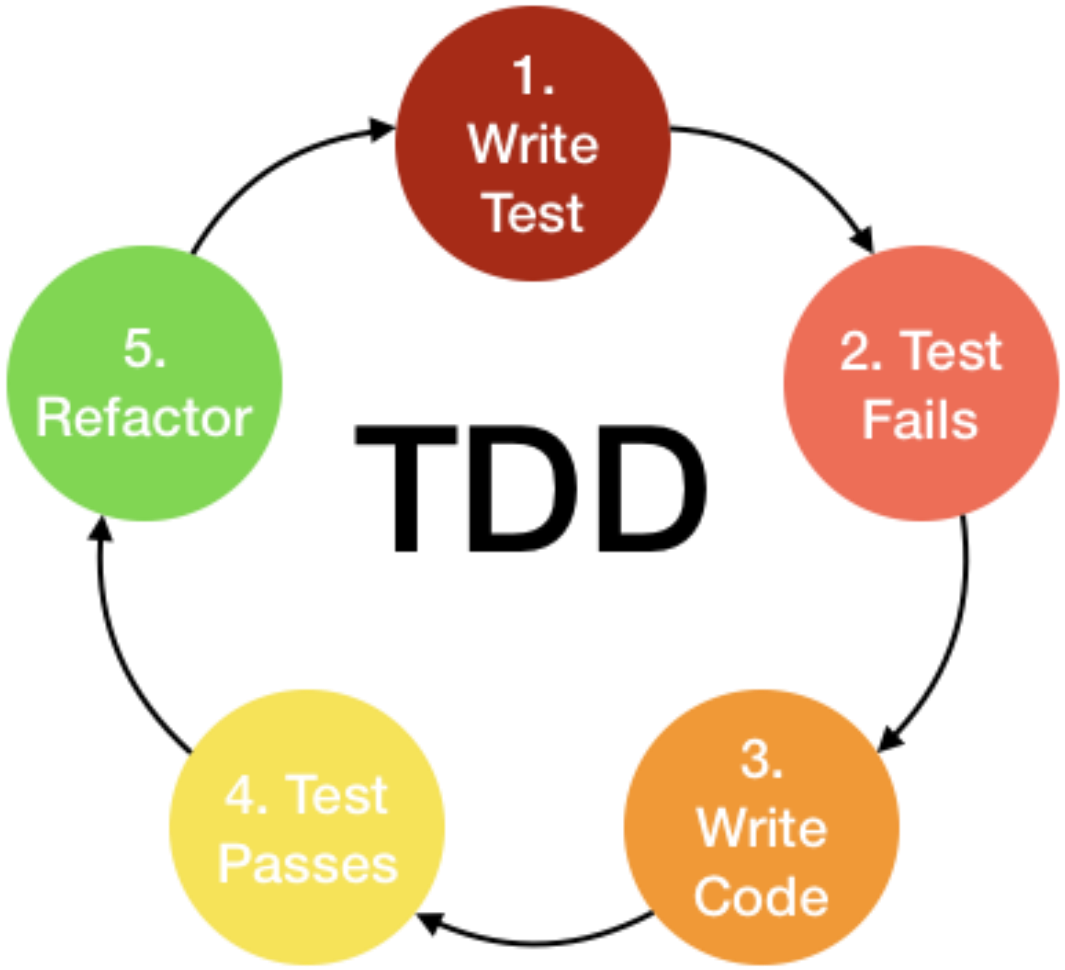
S2 Z5

Getting better

TDD -Test-driven development

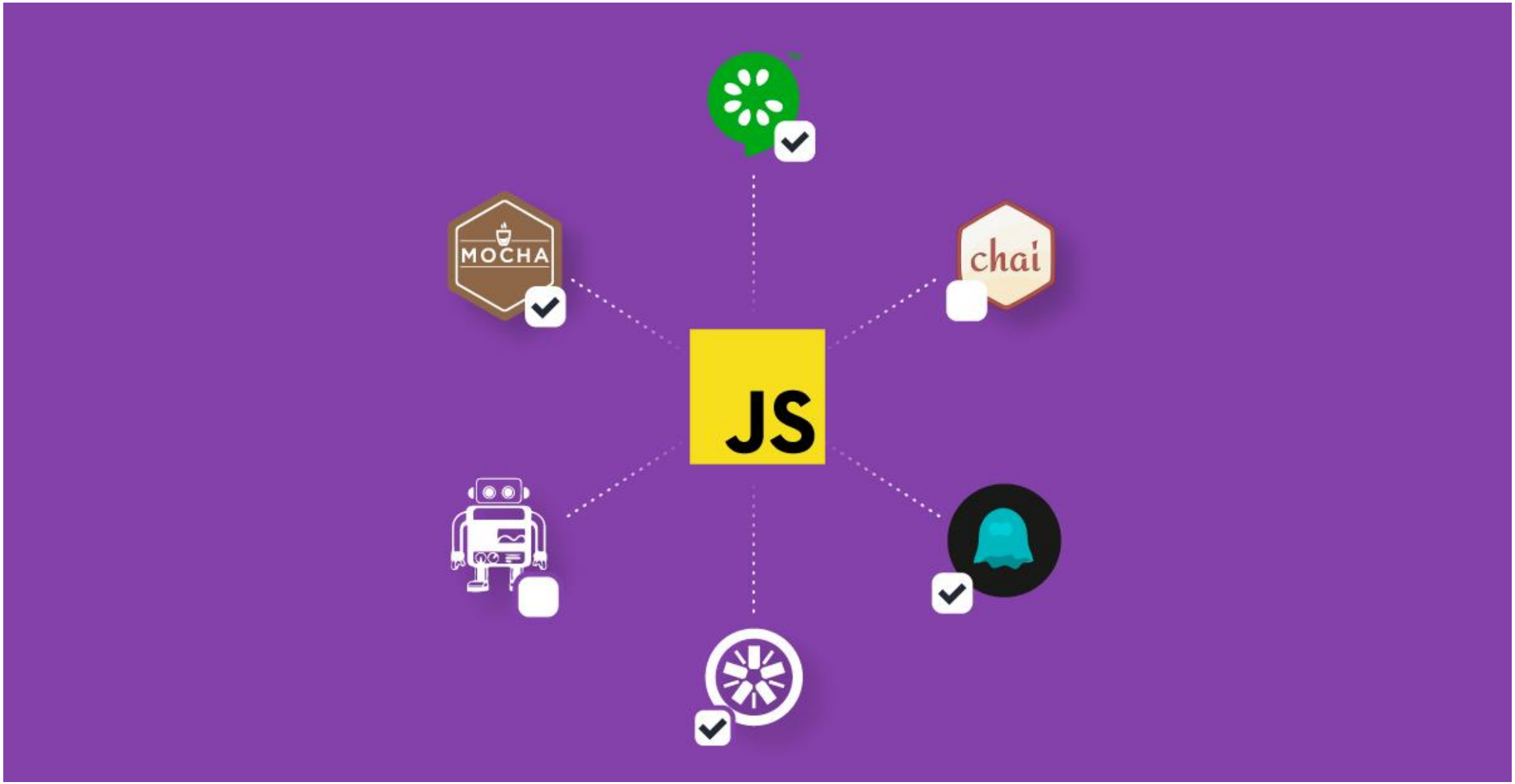
- **Test-driven development (TDD) -** is a software development process that relies on the **repetition** of a very **short** development cycle: requirements are turned into very specific **test cases**, then the code is improved so that the tests pass. This is **opposed** to software development that allows code to be added that is not proven to meet requirements.


1. Write Test
2. Test Fails
3. Write Code
4. Test Passes
5. Refactor
TDD

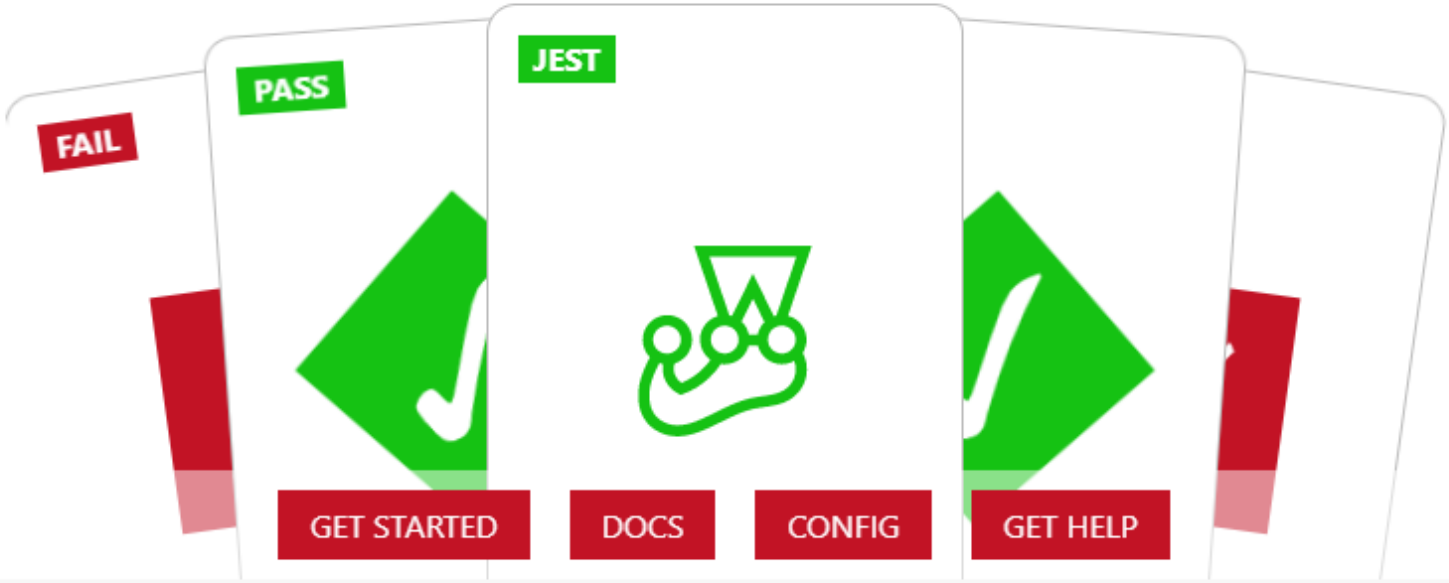
TALK TO ME


TEST
CASE

TEST COVERAGE


MOCHA
chai
JS

JS - jest




```
Jest is a delightful JavaScript Testing Framework
with a focus on simplicity.
```

https://jestjs.io/

Basic Tests

```
test("adds 1 + 2 to equal 3", () => {
    expect(sum(1, 2)).toBe(3);
});

test("adds 2 + 2 to equal 4", () => {
    expect(sum(2, 2)).toBe(4);
});

test("adds 0 + 2 to equal 2", () => {
    expect(sum(0, 2)).toBe(2);
});
```

Basic Tests

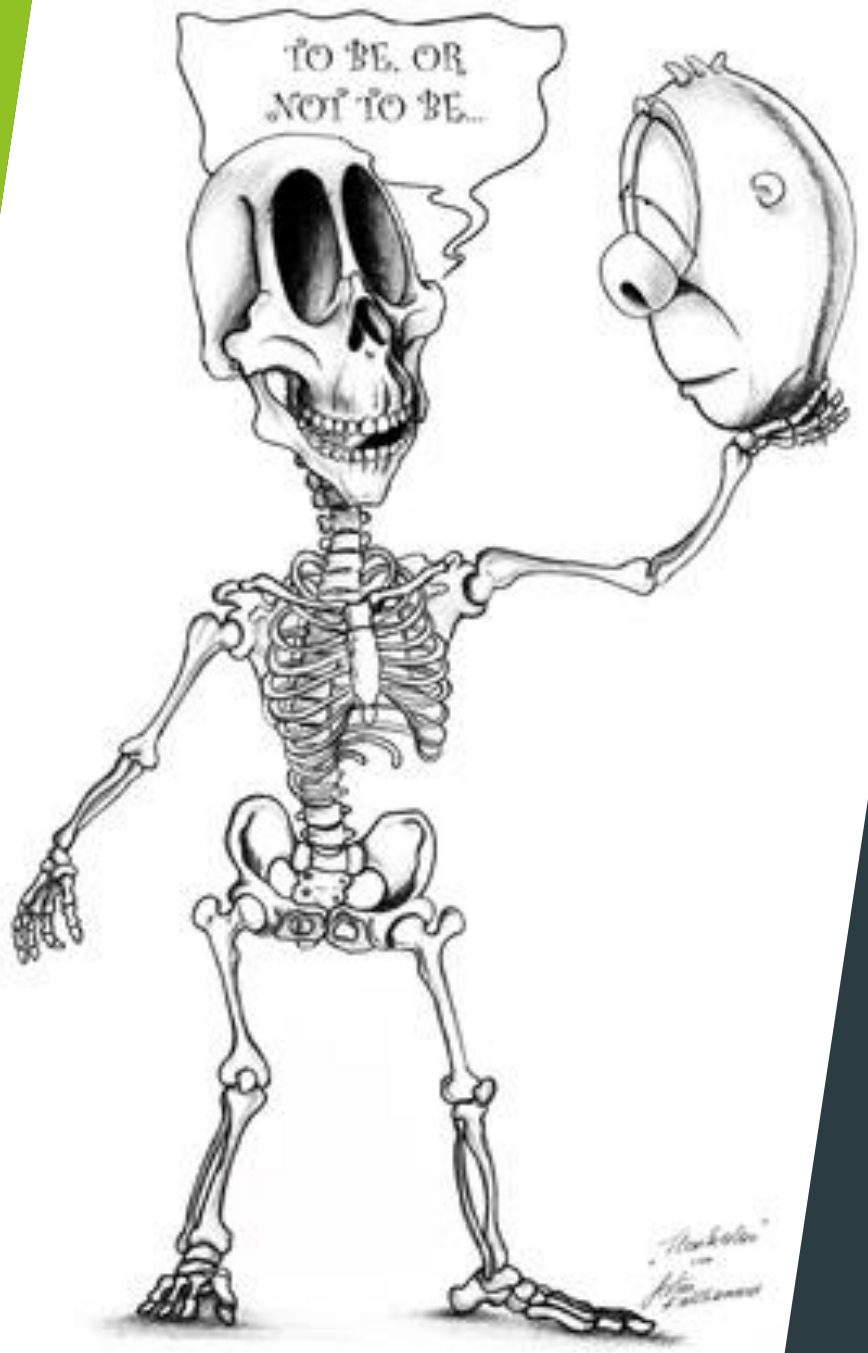
```
test("adds 1 + 2 to equal 3", () => {
    expect(sumLikeMultiply(1, 2)).toBe(3);
});

test("adds 2 + 2 to equal 4", () => {
    expect(sumLikeMultiply(2, 2)).toBe(4);
});

test("adds 0 + 2 to equal 2", () => {
    expect(sumLikeMultiply(0, 2)).toBe(0);
});
```

Precision

- toBeNull
- toBeUndefined
- toBeDefined
- toBeTruthy
- toBeFalsy



Strings

- toMatch()
- Or
- Not.Match()

- toContain()
- Or
- Not.ToContain()

Testing “Errors”

- expect(fn).toThrow();
- expect(fn).toThrow(Error);
- expect(fn).toThrow(“Error text”);

3 more topics

Immutability

```
let x = 1;
x++; //2

const y = 1;
y++; //Assignment to constant variable.
```

Immutability

```
const z = [1, 2];
z = 10; //Assignment to constant variable.
z = [3, 4]; //Assignment to constant variable.
z[0] = 3; // [3,2]
z.length = 0; // []

const w = Object.freeze([1, 2]);
w = 10; //Assignment to constant variable.
w = [3, 4]; //Assignment to constant variable.
w[0] = 3; //[1, 2]
w.length = 0; //[1, 2]
```

Immutability

```
const z = [1, 2];
const w = Object.freeze(z);

z[0] = 3;
w[0] = 3;

console.log(z); //[1, 2]
console.log(w); //[1, 2]
```

Tag Functions

```
function tagFunctionStupid(strings, ...values) {
    return "tagFunctionStupid string";
}

function tagFunctionMimic(strings, ...values) {
    var result = "";

    for (var i = 0; i < strings.length; i++) {
        if (i > 0) result += values[i - 1];
        result += strings[i];
    }

    return result;
}

function tagFunctionAddType(strings, ...values) {
    var result = "";

    for (var i = 0; i < strings.length; i++) {
        if (i > 0) result += `${values[i - 1]}(${typeof values[i - 1]})`;
        result += strings[i];
    }

    return result;
}
```

Tag Functions

```
var firstName = "Karol";
var lastName = "Rogowski";
var yearOfBirth = 1985;

var printMessage = `Hi ${firstName} ${lastName} born at ${yearOfBirth}.`;
printMessage = tagFunctionStupid`Hi ${firstName} ${lastName} born at ${yearOfBirth}.`;
console.log(printMessage);
```

tagFunctionStupid string

```
printMessage = tagFunctionMimic`Hi ${firstName} ${lastName} born at ${yearOfBirth}.`;
console.log(printMessage);
```

Hi Karol Rogowski born at 1985.

```
printMessage = tagFunctionAddType`Hi ${firstName} ${lastName} born at ${yearOfBirth}.`;
console.log(printMessage);
```

Hi Karol(string) Rogowski(string) born at 1985(number).

I'll be back.

Arnold Schwarzenegger

Generator

```
function* generator() {
    console.log(1);
    yield;
    console.log(2);
    yield "two";
    console.log(3);
    return "three";
}

var state = generator();
console.log(state);
console.log(state.next());
console.log(state.next());
console.log(state.next());

for (var f of generator()) {
    console.log(f);
}
```

Generator

```
function* generator() {
    for (var i = 0; i < 5; i++) {
        yield i;
    }
}

for (var f of generator()) {
    console.log(f);
}
```