UNIT TESTING

In Angular

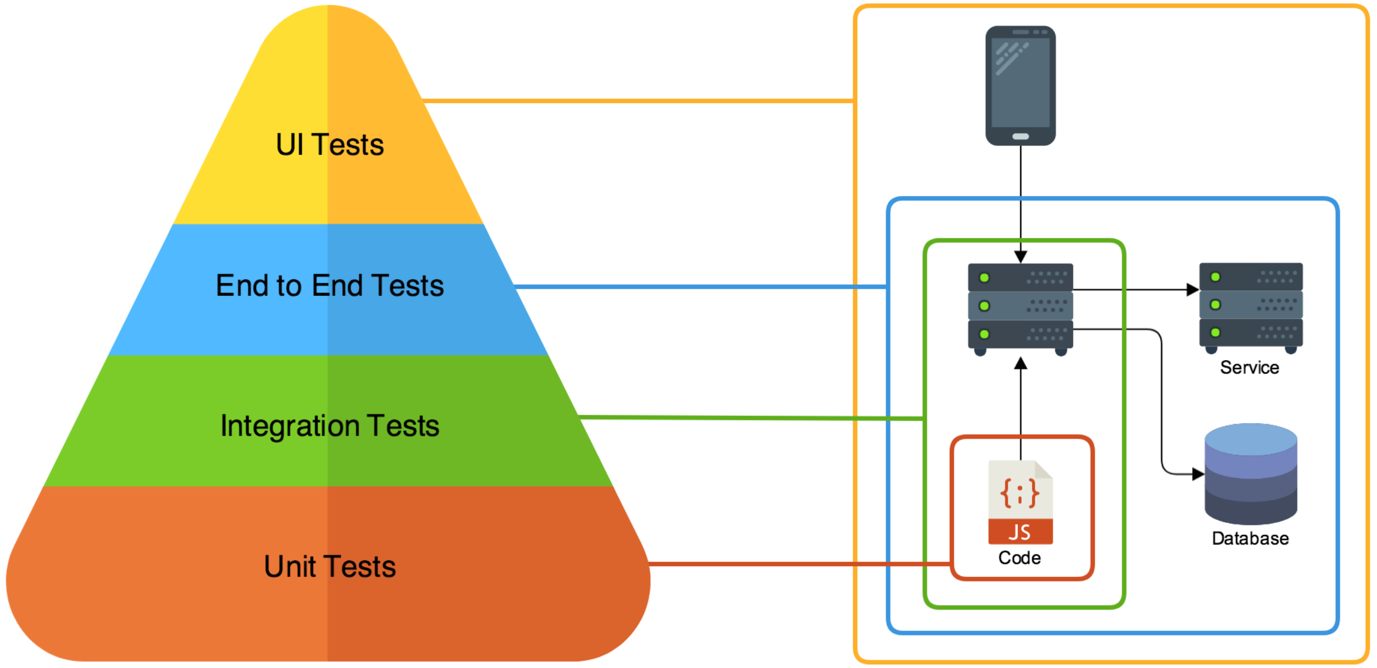
Kristiyan Velkov

Senior Front-end Engineer (Chapter Lead)

LinkedIn

What is Unit Testing?

- **Unit testing** is an action used to validate that separate units of source code remains working properly.



The different types of Unit tests

Unit tests - Testing one thing at a time, “unit” of the application.

- **Isolated tests -** is the process of isolating a class or function and testing it as a regular JavaScript code.

Note!

Integration tests - checking that multiples units are interacting with each other correctly.

- **Shallow**: testing components template without rendering its children.
- **Deep**: include all components parent and children.

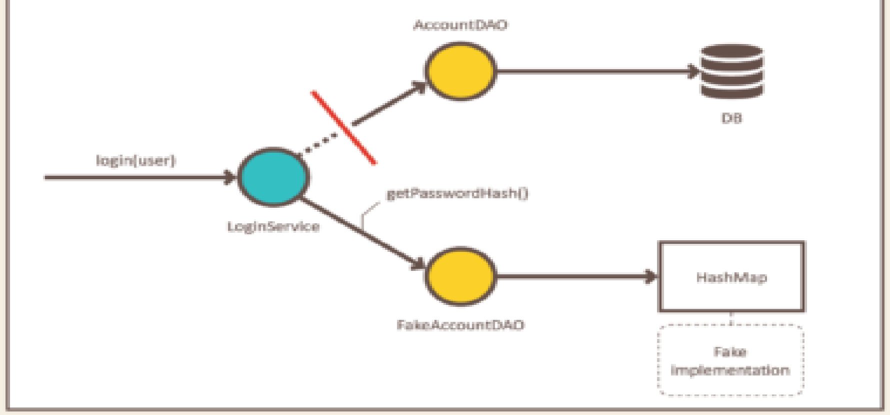
Isolating the Unit in tests

Rule: Only test the unit and not its dependencies.

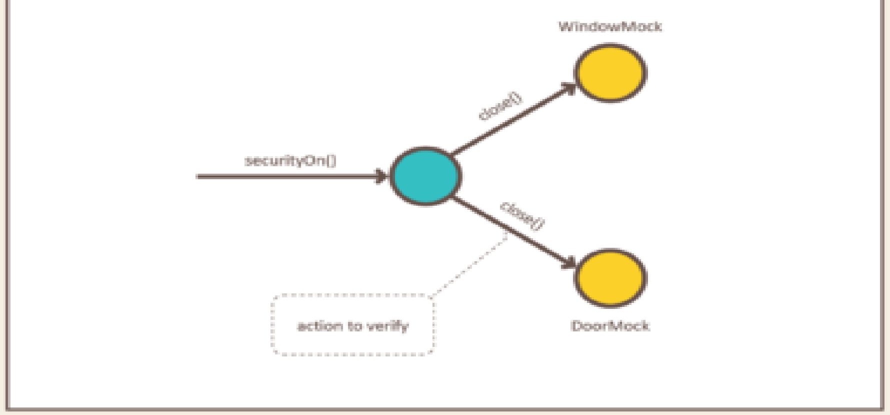
- Use **Test Doubles** to isolate dependencies
  - **Mocks**: Mock functions allow you to test the links between code by replace the actual implementation of a function and capturing calls to the function.
  - **Stubs**: provide already defined answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test. A stub is a way to modify a function and delegate control overs its behavior to you (the programmer). You generally stub a function when it has side effects you are trying to control.
  - **Spies**: Gives you the ability to "spy" on a function, by letting you capture and then assert that the function was called with the right arguments, or that the function was called a certain number of times, or even what the return value was or what context the function was called with.
  - **Fake** objects actually have working implementations, They are not suitable for production. Example : fake api

Test Doubles

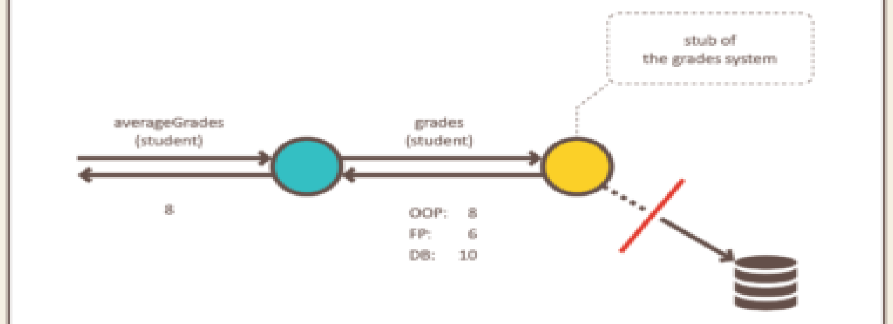
Fake



Mock



Stub



Angular TestBed

- The TestBed is the most important of the Angular testing utilities.
- The TestBed creates a dynamically-constructed Angular test module that emulates an Angular @NgModule.
- The TestBed.configureTestingModule() method takes a metadata object that can have most of the properties of an @NgModule.

TypeScript

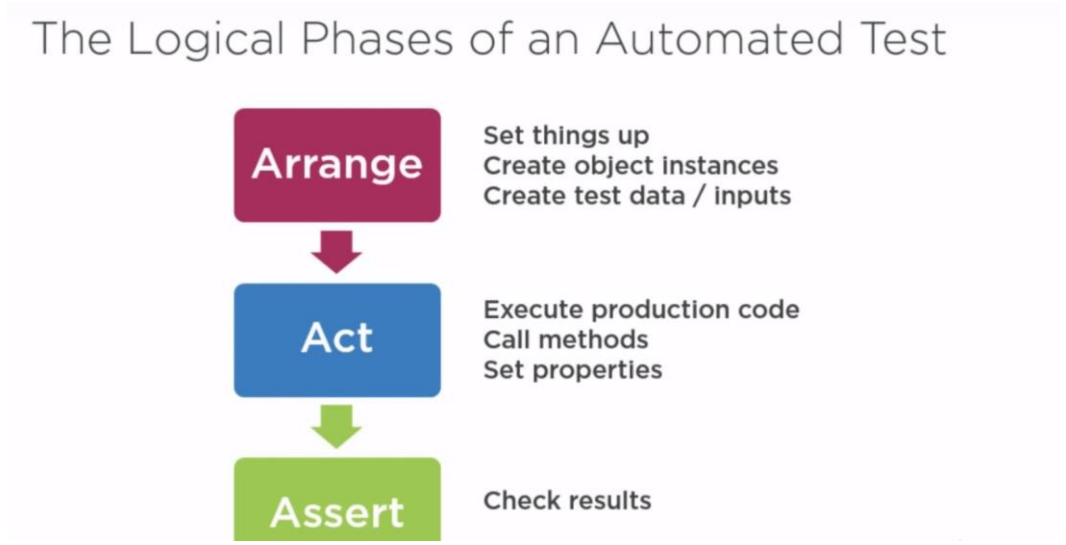
```
import {TestBed, ComponentFixture} from '@angular/core/testing';
import {LoginComponent} from './login.component';
import {AuthService} from "./auth.service";

describe('Component: Login', () => {

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [LoginComponent],
      providers: [AuthService]
    });
  });
});
```

How to structure tests

AAA === **Given-When-Then.**

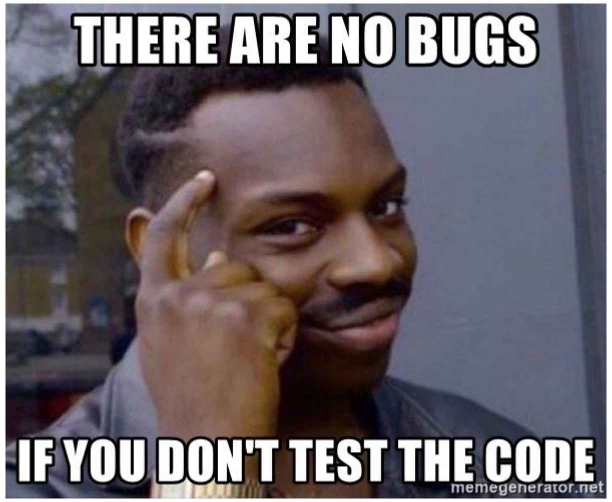
Arrange/Act/Assert (AAA) is a pattern for arranging and formatting code in Unit Test methods.

- **Arrange** – setup the testing objects and prepare the prerequisites for your test.
- **Act** – perform the actual work of the test.
- **Assert** – verify the result.



How to write good unit tests ?

- Minimize logic out of tests (what will test the tests?)
- Name Your Tests Well (the good test will start with keyword “should”)
- Test one thing at a time
- Promotes the removal of duplication in the code
- Isolate change to those parts of the system that must change the original code.
- Repeat yourself if necessary to make it easier to read.
- Tests should be **Readable**
- Make the tests short and fast
- Don’t test already tested things like Angular, React JS, etc.

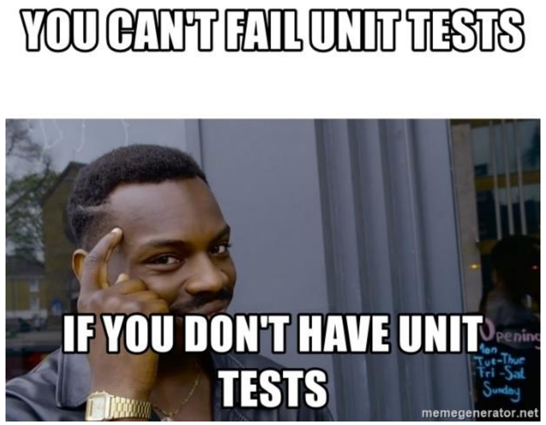


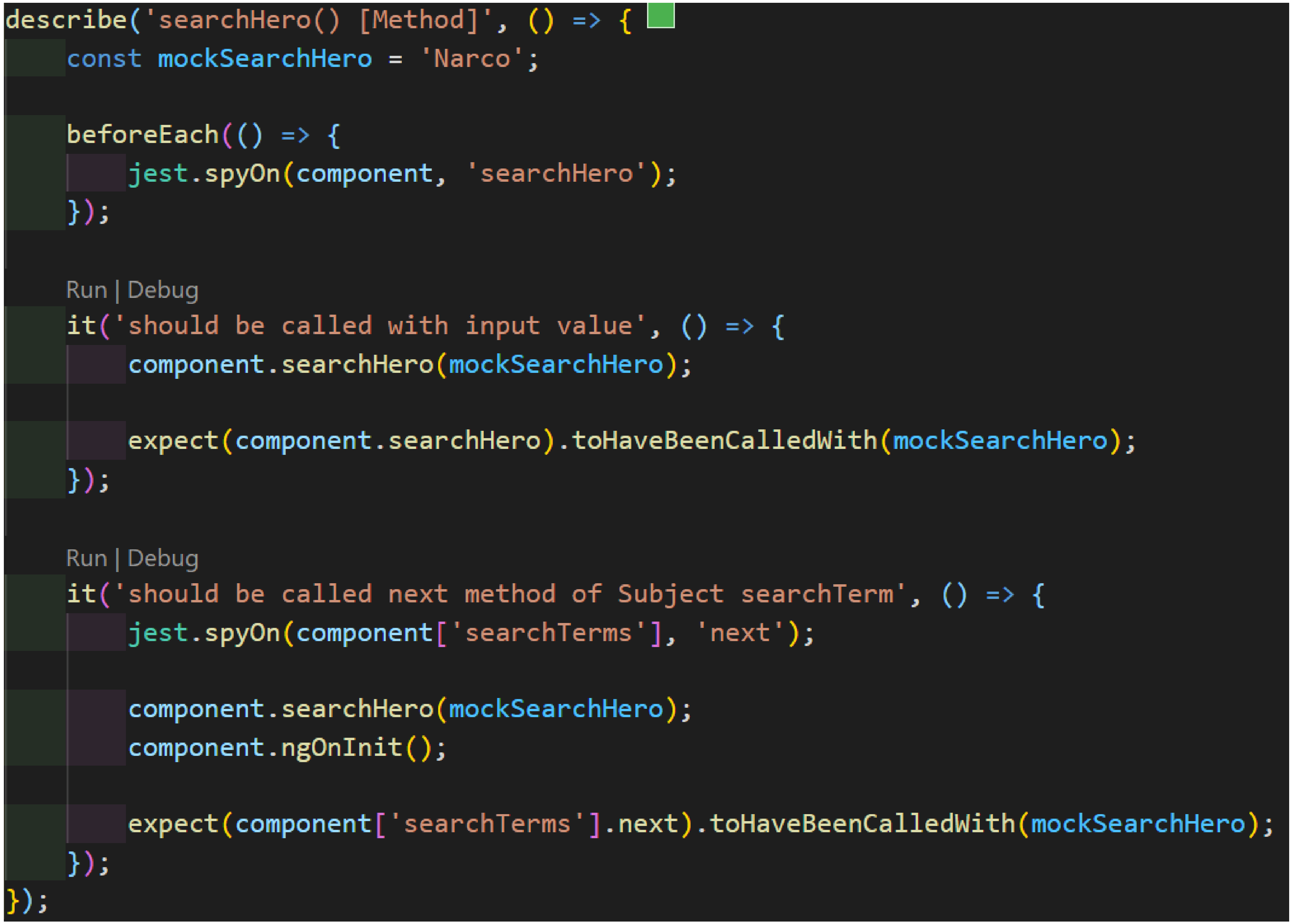
Tell the Story

- A test should be a complete story
- You shouldn't need to look around much to understand the test
- You need to spend only 5 seconds to know how the test work and what is expected
- Unit tests can act as documentation.

Techniques for better unit tests

- Use **afterEach(), beforeEach(), afterAll(), beforeAll()** to structure your test
- Group tests in **describe()**
- **it()** for every new test.
- Include all of the "Act" and "Assert" test parts are in the **it()** clause if is only for particular test else move to **beforeEach()**
- Add metadata for each test like :
  [Method], [Property], [Input], [Output], **[Pipe]** etc. - This will improve the readability of your tests.





```
describe('searchHero() [Method]', () => {
  const mockSearchHero = 'Narco';

  beforeEach(() => {
    jest.spyOn(component, 'searchHero');
  });

  Run | Debug
  it('should be called with input value', () => {
    component.searchHero(mockSearchHero);

    expect(component.searchHero).toHaveBeenCalledWith(mockSearchHero);
  });

  Run | Debug
  it('should be called next method of Subject searchTerm', () => {
    jest.spyOn(component['searchTerms'], 'next');

    component.searchHero(mockSearchHero);
    component.ngOnInit();

    expect(component['searchTerms'].next).toHaveBeenCalledWith(mockSearchHero);
  });
});
```

The Benefits of Unit Tests

- Unit tests help you to find and fix bugs earlier.
- Unit tests can contribute to higher code quality.
- Unit tests might contribute to better application architecture.
- Unit tests can act as documentation.
- Detect code smells in your codebase.

Testing Tools

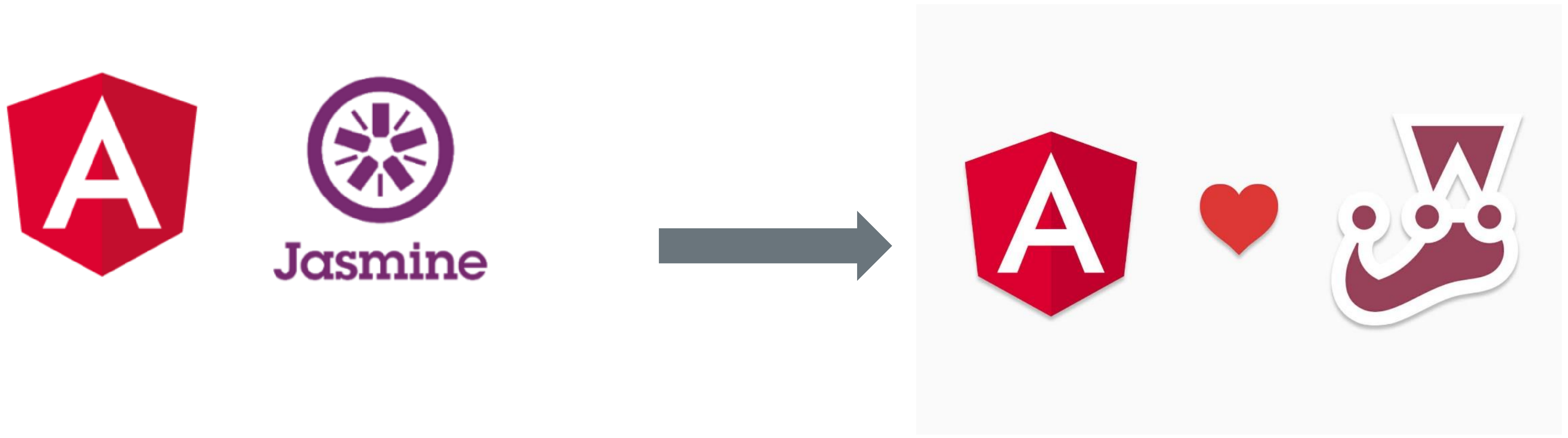
- **Frameworks**
  - Jest
  - Jasmine
  - Cypress
  - Selenium
  - Sinon
  - WebdriverIO
  - Test Café
  - ChaiJs
  - Mocha
  - Puppeteer

- **Mocking Libraries**
  - Jasmine
  - sinon.js
  - Testdouble.js

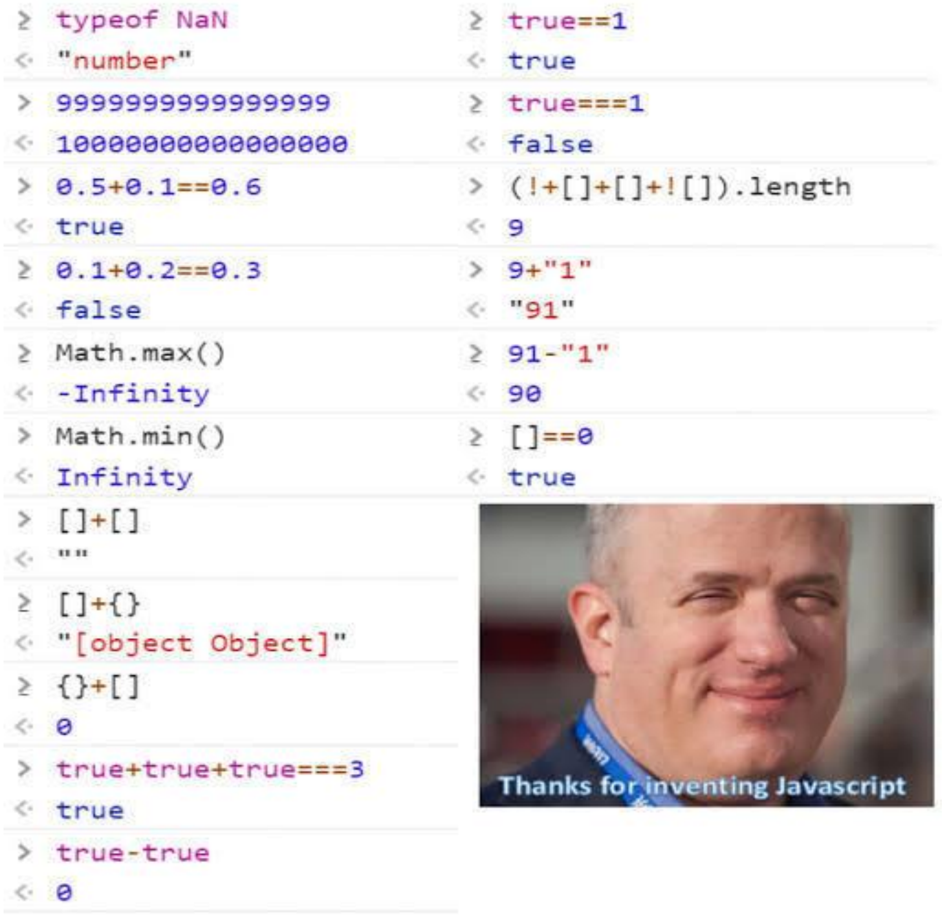
- **Test Runners**
  - Karma
  - Jest
  - Cypress

Migrating from Jasmine to Jest

By default, the Angular CLI provides Karma as a test runner and Jasmine as the test framework. Nx offers the choice of using Jest for both runner and framework. - how to ?



Keep calm and love JavaScript



```
> typeof NaN
< "number"
> 9999999999999999
< 10000000000000000
> 0.5+0.1==0.6
< true
> 0.1+0.2==0.3
< false
> Math.max()
< -Infinity
> Math.min()
< Infinity
> []+[]
< ""
> []+{}
< "[object Object]"
> {}+[]
< 0
> true+true+true===3
< true
> true-true
< 0
```

```
> true==1
< true
> true===1
< false
> (!+[]+[]+![]).length
< 9
> 9+"1"
< "91"
> 91-"1"
< 90
> []==0
< true
```