TESTING

by Anastasiia Derkach

softserve

WHY TO USE

softserve

- **Finds Software Bugs Early**
- **Quality of Code**
- **Makes the Process Agile**
- **Facilitates Changes and Simplifies Integration**
- **Provides Documentation**
- **Debugging Process**
- **Reduce Costs**

softserve

FINDS SOFTWARE BUGS EARLY

Issues are found at an early stage. Since unit testing is carried out by developers who test individual code before integration, issues can be found very early and can be resolved then and there without impacting the other pieces of the code. This includes both bugs in the programmer's implementation and flaws or missing parts of the specification for the unit.

QUALITY OF CODE

Unit testing improves the quality of the code. It identifies every defect that may have come up before code is sent further for integration testing. Writing tests before actual coding makes you think harder about the problem. It exposes the edge cases and makes you write better code.

MAKES THE PROCESS AGILE

One of the main benefits of unit testing is that it makes the coding process more Agile. When you add more and more features to a software, you sometimes need to change old design and code. However, changing already-tested code is both risky and costly. If we have unit tests in place, then we can proceed for refactoring confidently.

Unit testing really goes hand-in-hand with agile programming of all flavors because it builds in tests that allow you to make changes more easily. In other words, unit tests facilitate safe refactoring.

FACILITATION CHANGES AND SIMPLIFIES INTEGRATION

Unit testing allows the programmer to refactor code or upgrade system libraries at a later date and make sure the module still works correctly. Unit tests detect changes that may break a design contract. They help with maintaining and changing the code.Unit testing reduces defects in the newly developed features or reduces bugs when changing the existing functionality.

Unit testing verifies the accuracy of the each unit. Afterward, the units are integrated into an application by testing parts of the application via unit testing. Later testing of the application during the integration process is easier due to the verification of the individual units.

PROVIDES DOCUMENTATION

Unit testing provides documentation of the system. Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit's interface (API).

DEBUGGING PROCESS

Unit testing helps simplify the debugging process. If a test fails, then only the latest changes made in the code need to be debugged.

REDUCE COSTS

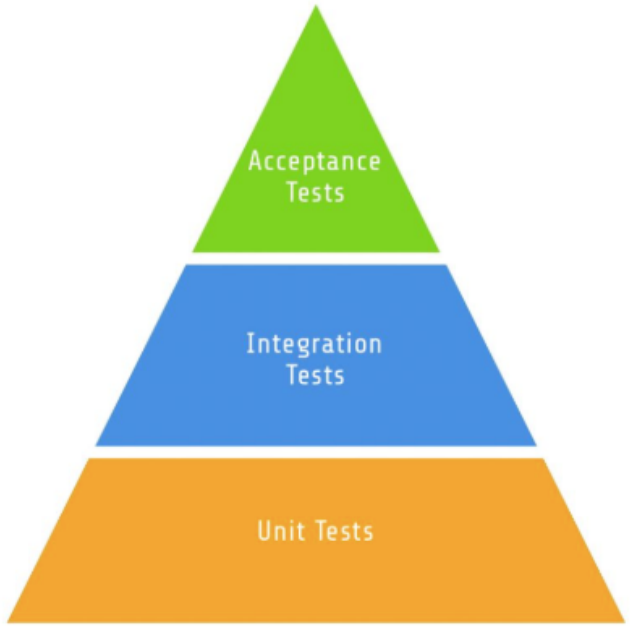
Since the bugs are found early, unit testing helps reduce the cost of bug fixes. Just imagine the cost of a bug found during the later stages of development, like during system testing or during acceptance testing. Of course, bugs detected earlier are easier to fix because bugs detected later are usually the result of many changes, and you don't really know which one caused the bug.

UNIT TESTING

Isolate each part of the program

INTEGRATION TESTS

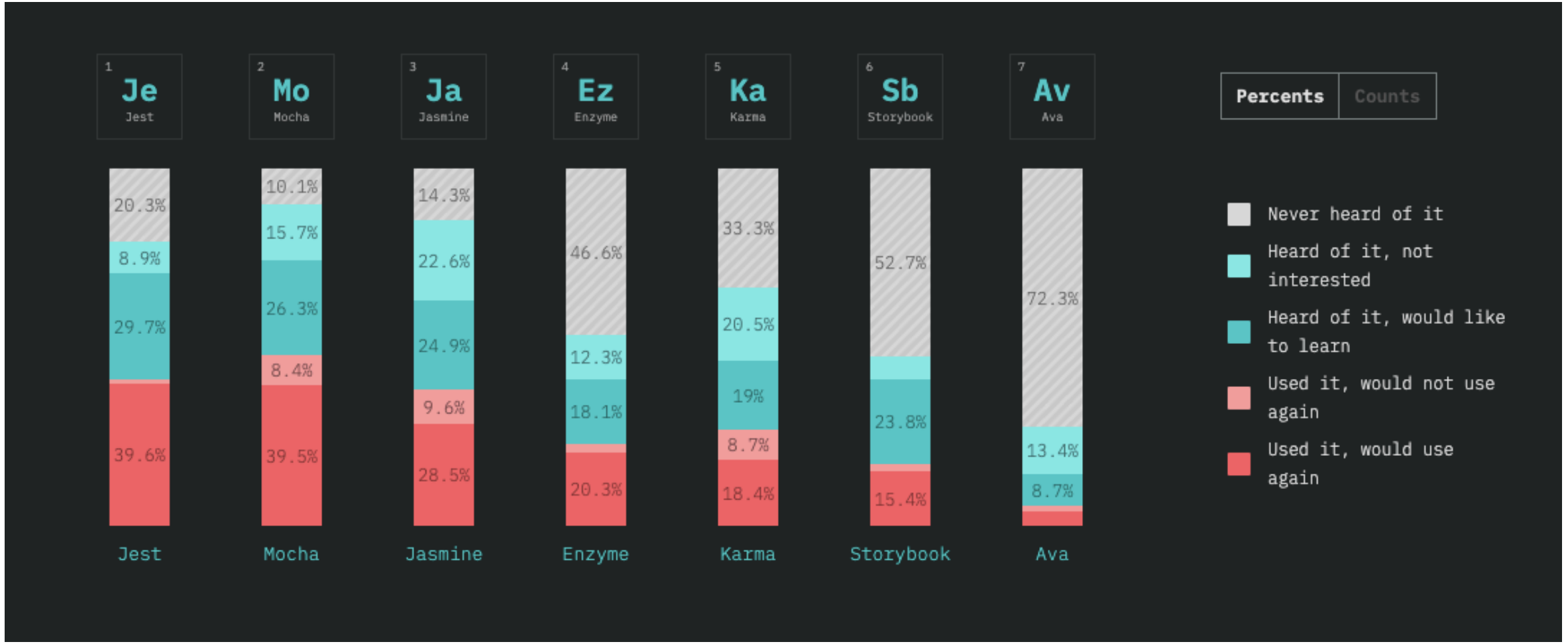
Test the inter-operation of multiple subsystem



softserve

TESTING FRAMEWORK

softserve

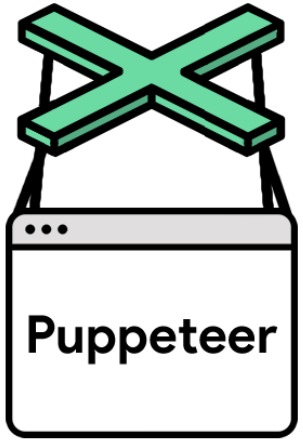

1
Je
Jest
2
Mo
Mocha
3
Ja
Jasmine
4
Ez
Enzyme
5
Ka
Karma
6
Sb
Storybook
7
Av
Ava
Percents
Counts
20.3%
8.9%
29.7%
39.6%
10.1%
15.7%
26.3%
8.4%
39.5%
14.3%
22.6%
24.9%
9.6%
28.5%
46.6%
12.3%
18.1%
20.3%
33.3%
20.5%
19%
8.7%
18.4%
52.7%
23.8%
15.4%
72.3%
13.4%
8.7%
Never heard of it
Heard of it, not interested
Heard of it, would like to learn
Used it, would not use again
Used it, would use again
Jest
Mocha
Jasmine
Enzyme
Karma
Storybook
Ava

softserve

JEST

JASMINE

ENZYME



softserve

Framework	Jasmine	Ava	Jest	Mocha
Open source	YES	YES	YES	YES
In-built coverage reporting	NO	NO	YES	NO
Parallel test running	NO	YES	YES	NO
Snapshots	NO	YES	YES	NO
In-built spies	YES	NO	YES	NO
In-built mocking	YES	NO	YES	NO
In-built assertions	YES	YES	YES	NO
ES2017 support	NO	YES	YES	NO

JEST

JEST is a JavaScript testing framework developed and maintained regularly by FACEBOOK.

It is based on Jasmine but has been greatly improved since the early days.

```
describe("Sum numbers", () => {
  test("it should sum two numbers correctly", () => {
    const sum = 1 + 2;
    const expectedResult = 3;
    expect(sum).toEqual(expectedResult);
  })
});
```

PROS:

- Well documentation
- Easy to set up
- Parallel test running
- Fast – tests are parallelized by running them in their own processes to maximize performance
- It includes features like snapshots, coverage, and test watching

CONS:

- Jest displays multiple error messages for the same error
- It can require more dependencies during initial setup (e.g babel)

JASMINE

JASMINE, on the other hand, has been around for a lot longer. It was developed by Pivotal Labs and released in 2010.

It aims to run on any JavaScript-enabled platform and is highly flexible and compatible with a variety of other testing frameworks and libraries like Sinon and Chai

```
describe("Sum numbers", function() {
  it("should sum two numbers correctly", function() {
    var sum = 1 + 2;
    var expectedResult = 3;
    expect(sum).toEqual(expectedResult);
  });
});
```

PROS:

- Simple to setup. Jasmine has a CLI tool that creates a spec folder and a JSON configuration file. With one command you are ready to start testing your code
- It has been around for a long time and is thoroughly tested, documented, and there are a lot tutorials on how to use it
- It's supported by many CI servers with plugins available for some of those that don't have out of the box support

CONS:

- Unfriendly error logs
- Asynchronous testing can be quite a hustle.
- Test files must have a specific suffix (*spec.js)

MOCHA

MOCHA like Jasmine, has been in existence for quite a while. It was initially released in November 2011.

However, unlike other frameworks like Jest and Jasmine, it relies on third-party assertions, mocking, and spying tools(Spies are objects that keep track of their interaction with other objects or pieces of code.

It is very extensible and has a lot of plugins, extensions, and libraries designed to run on top of it.

PROS:

- Highly extensible and therefore has support for different assertion and mocking libraries

CONS:

- The use of extra libraries can introduce configuration complexity and also increases maintenance work

AVA

Minimalism is the focus of AVA. It has a simple API while still supporting advanced features. It is quite fast and achieves this by running tests in parallel as separate Node.js processes. Unlike other testing frameworks such as Jest and Jasmine, it does not create test globals.

PROS:

- It’s simple and easy to use. To install and setup AVA, all you have to do is run npm init ava
- Parallel test running
- It has built-in support for async functions

CONS:

- AVA is relatively new. The community is still growing and there isn’t a lot of documentation or tutorials like other testing frameworks

CHOOSING FRAMEWORK

softserve

The best framework can vary based on your needs, project size, and other factors. What works now might not work in the future. It is important to take both your current and future needs into consideration when choosing the right framework.

If you want to hit the ground running, you cannot go wrong with Jest. It is an extremely fast framework, easy to set up and has a lot of built-in features to help you with your testing.

RESOURCES USED

- https://2018.stateofjs.com/testing/enzyme
- https://medium.com/@me_37286/yoni-goldberg-javascript-nodejs-testing-best-practices-2b98924c9347
- https://blog.logrocket.com/the-best-unit-testing-frameworks-for-node-js/