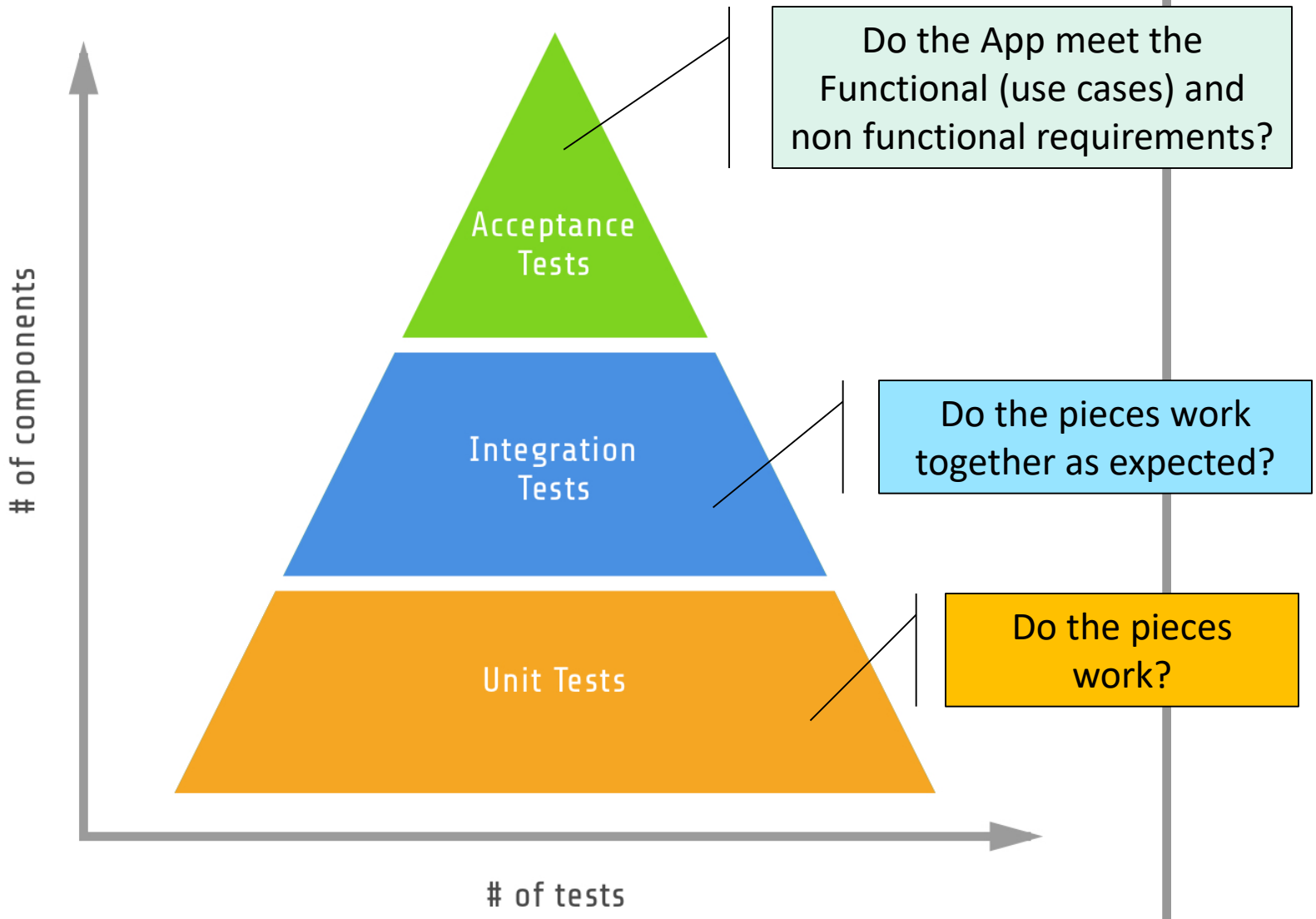
Unit Testing in JavaScript

Dr. Abdelkarim Erradi
Dept. of Computer Science & Engineering
QU

Outline

1. Unit Testing Overview
2. Mocha & Chai
3. Creating a test suites and specs

THE PYRAMID OF TESTS



What is Unit Testing?

- A unit test **tests one unit of work** to verify that it works **as expected**
- A unit test should be:
  - Isolated and Independent from other tests
  - Repeatable
  - Predictable
- Unit tests are added to the code repository along with the code they test
- A unit testing framework is needed
  - e.g., QUnit, Jasmine, **Mocha**
  - We'll use Mocha https://mochajs.org/

Manual Testing

- You may have already done unit testing by without using a Unit Testing framework
- **Manual tests** are less efficient
  - Not structured
  - Not repeatable
  - Not all code covered
  - Not easy to run automatically
- A Unit Testing framework enables better structure of the testing code

Why Unit Tests?

- Help to detect bugs in early stages of the project => **improve code quality**
- Can expose high coupling => encourage refactoring to produce testable code
- Serve as live documentation
- Reduce the cost of change
- Allow change and refactoring with confidence
  (refactoring = change the code structure without changing its functionality)
- Decrease the defect-injection rate due to refactoring / changes

Mocha Overview

- Mocha is a feature-rich framework that helps us write and run unit tests
  - Run in both the browser and on Node.js
  - Can test async code
  - Often used with Chai.js for writing **test assertions** http://chaijs.com/

```
describe('#sum', () => {
  it('when empty array, expect to return 0', () => {
    let actual = sum([])
    expect(actual).to.equal(0)
  })
  it('when with single number, expect the number', () => {
    let number = 6;
    let actual = sum([number]);
    let expected = number;
    expect(actual).to.equal(expected);
  })
})
```

Test Suite & Test Spec

- Test suite describes the functionality to test
  - Using a **describe**() function
  - **xdescribe** to disable a test suite
- Test spec tests the functionality
  - Using an **it**() function to tell the test what **it** should **expect** from running a unit of code
  - Contain one or more expectations (compare **actual** with **expected** results)
  - **xit** to disable a spec

Example

- Test suite is created with the **describe**(name, callbackFunction)
- Test Spec is created with the **it** (name, callbackFunction)

```
describe ('Person Test Suite', () => {
      it('with valid names, expect ok', () => {
      let person = new Person('Ali', 'Faleh’);
      expect(person.firstname()).to.equal('Ali');
      expect(person.lastname()).to.equal('Faleh');
    })
})
```

Chai Expect Assertion

- Expectations are assertions that can be either true or false
- Use the **expect** function within a spec to declare an expectation
  - Receives the actual value as parameter
  - A Matcher is a comparison between the actual and the expected values

```
expect(person.getName()).to.equal('Ali')
expect(person).to.be.a('Person')
expect(person).to.not.be.undefined
expect(person).not.to.be.null
expect(person).to.be.ok
expect(person).not.to.be.ok
```

Matchers

- Used to verify expectations
  - `expect(1 === 1).to.be.true`
  - `expect('b' + 'a' + 'r' ).to.not.equal('bar')`
  - `expect( 10 / 0 ).to.throw(Error)`
  - `expect( { foo: 'bar' })`
    `.to.have.property('foo').and.equal('bar')`
  - `expect( ( ) => x.y.z ).to.throw()`

- More examples @ http://chaijs.com/api/bdd/

Setup and Teardown

- **before** – runs before each test suite
- **after** – runs before each test suite
- **beforeEach** – runs before each test
- **afterEach** – runs after each test

```
before( async () => {
    await mongoose.connect('mongodb://localhost/books')
    await mongoose.connection.db.dropDatabase()
})

beforeEach(() => { ... })

afterEach( () => { ... })

after( async () => {
    await mongoose.disconnect()
})
```

Good Unit Tests

- Test one thing at a time (1 focus per test)
- Test results, not internals
- Test your code for different scenarios
- You want to write positive tests **and** negative tests
  - Positive tests are ones that should pass
  - Negative tests involve values that are outside acceptable ranges

    - You're testing to make sure that they do fail

Summary

- Unit Tests are an important part of any development process
- Mocha library can help you test your JavaScript code
- Mocha uses test suites to organize the tests
  - Tests suites are created with the **describe**
- Specs (tests) are contained in a test suites
  - Tests specs are created with the **it**
- Key guideline for testable code = **Modular code**: simple, single-purpose functions

Resources

- Mocha

https://mochajs.org/

- Chai

http://chaijs.com/