Contents

[**What is Unit Testing?** 1](#_Toc123314661)

[**Unit Testing vs Other Types** 1](#_Toc123314662)

[**Test Driven Development** 2](#_Toc123314663)

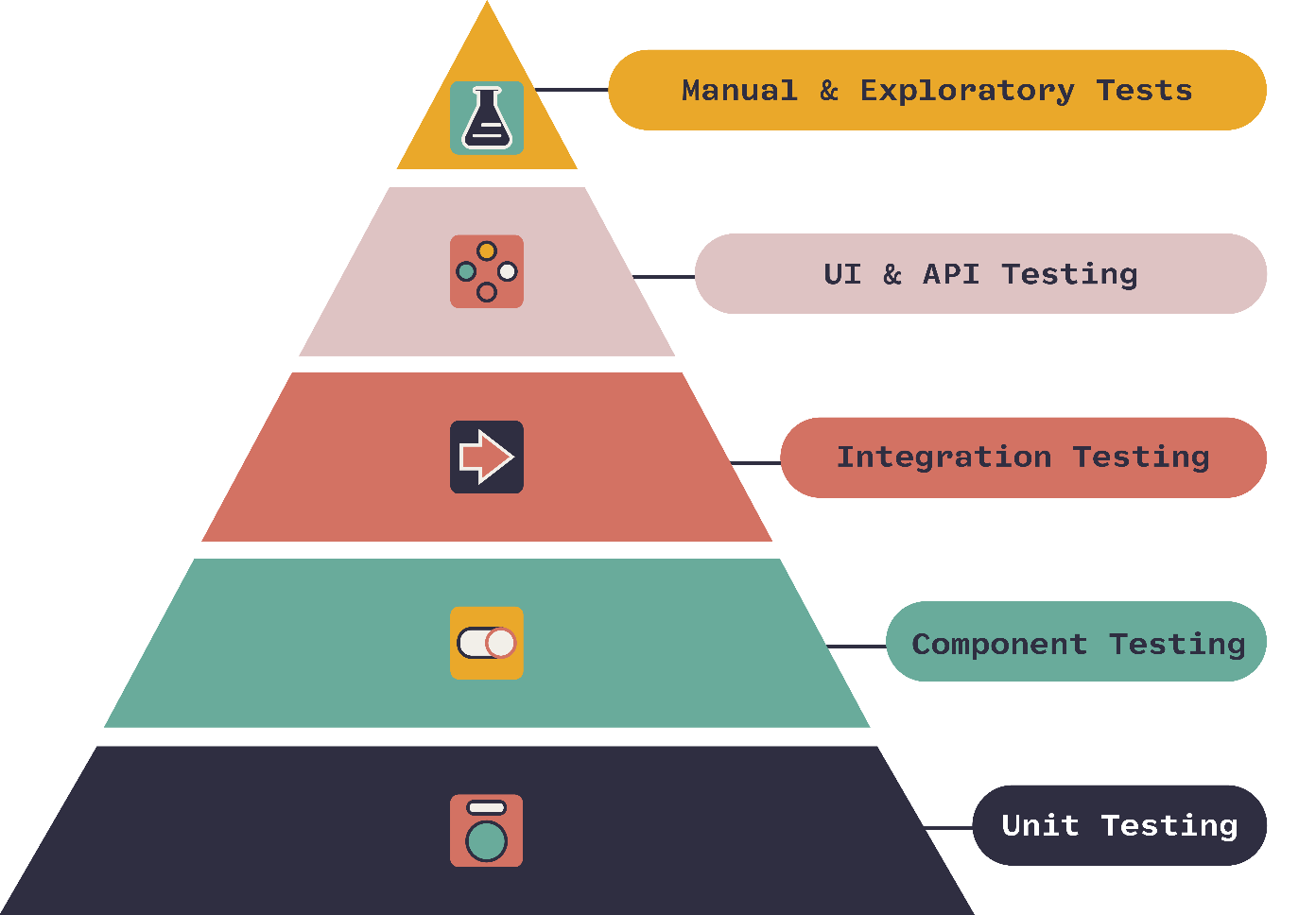
[**JUnit** 3](#_Toc123314664)

[**FIRST Principles of Testing** 3](#_Toc123314665)

[**Sean’s Unit Testing Rules of Thumb** 3](#_Toc123314666)

*'It costs almost five times more to fix a coding defect once the system is released that it does to fix it in unit testing.'*

*– PAUL GROSSMAN, IT CONSULTANT*

[](https://www.onpathtesting.com/blog/qa-testers-what-is-the-agile-testing-pyramid)

## **What is Unit Testing?**

- Unit testing is the process of writing code to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- We should strive to break the function to ensure that it meets quality criteria

- We should also strive for \_\_\_\_\_ code coverage

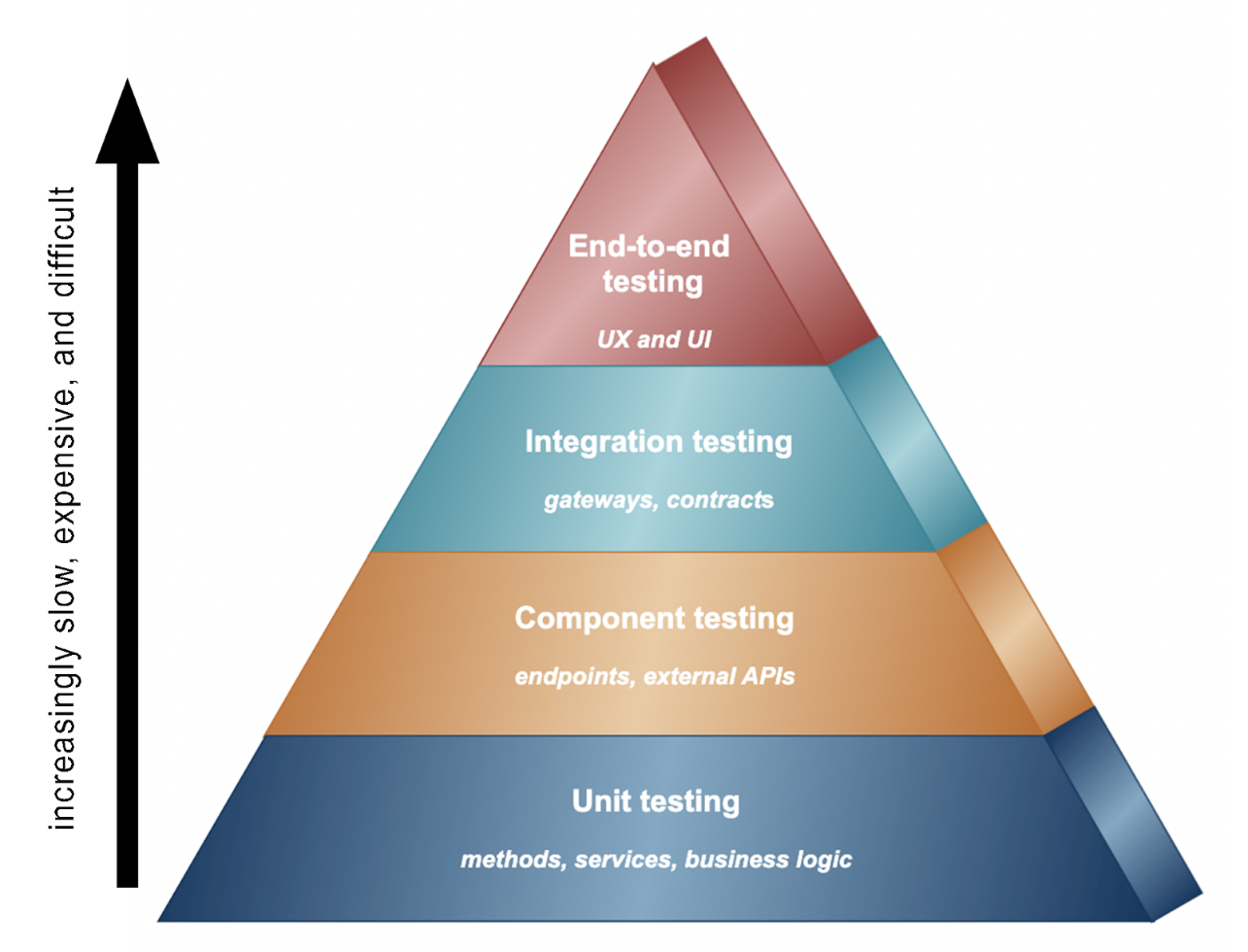
- Every single line of code in our application should be run through our unit tests

- We should also write code with the intention of it being tested (thinking in terms of \_\_\_\_\_\_\_\_)

- Edge cases being \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Ex: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Unit Testing vs Other Types**

[](https://www.cortex.io/content/an-overview-of-the-key-microservices-testing-strategies-types-of-tests-and-the-best-testing-tools)

**Unit Tests** - Tests \_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Component Tests** – Tests \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Integration Tests** - Tests \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**UI Tests**- Tests \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Manual Tests** – Tests \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Regression** **Tests** – Tests \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Stress** - Tests \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Others**: functional, black-box, End-to-End, Fuzz, Penetration, Security, etc.

## **Test Driven Development**

- Test Drive Development (TDD) is this idea you should write code that testable easily

- write tests before write the code

- Yes, the tests will initially fail because the code has not been written

- This makes you think through how the code will be used before how it will be implemented

- TDD was first developed by Kent Beck as part of the Extreme programming movement

- for history of TDD read <https://semaphoreci.com/blog/test-driven-development>

- Test Driven Development is one of several ideas that came from Extreme programming and the Agile programming movement

- We will cover the [Agile Development Manifesto](http://agilemanifesto.org/) more in depth in a later week

1. Individuals and interactions over processes and tools

2. Working software over comprehensive documentation

3. Customer collaboration over contract negotiation

4. Responding to change over following a plan

- It's widely used out in industry because it encourages developers to write good unit tests

## **JUnit**

- JUnit is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- It is the most popular \_\_\_\_\_\_\_\_\_\_\_\_\_\_ in Java

- It supports ­­­­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_ for testing

- I can write specific methods to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- This same principle applies for \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **FIRST Principles of Testing**

* Fast
* Isolated/Independent
* Repeatable
* Self-validating
* thorough

- Fast = ­­­­­­­­­­­­­should take just seconds not minutes, so there is no excuse to not run tests often

“Test early, test often”

- Isolated/Independent = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- follow the 3 A’s

1. Assemble = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Act = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Assert = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Repeatable = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Self validating = automatically comparing the actual with an expected rather than require manually reaching through each test result

- Thorough = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- To read more about the FIRST principle read this [article](https://medium.com/@tasdikrahman/f-i-r-s-t-principles-of-testing-1a497acda8d6)

## **Sean’s Unit Testing Rules of Thumb**

- If a method throws an exception, you should ensure that it DOES throw the exception

- Additionally, the exception should NOT be thrown in valid cases

- Run tests for null values

- When testing numbers, use the rule of "one, none, or some"

- Test that it works with a singular value (works because odd numbers are harder to work with (typically))

- Also, 1 is a prime number

- It's also good to test an even number

-- This ensures that it works for both odd and even

- Test with no values

-- This will handle empty arrays/null values/etc.

- Test with some values

-- Typically, I create a large mock data file (csv) and use the mock data to simulate the actual thing

- Test with negative numbers

-- Often these will go unnoticed in development

-- Handles overflow that way (integer overflow)

- Ask yourself what logical edge cases there are

-- Things like ties, negative numbers, invalid format, business requirements, etc.