Contents

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

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

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

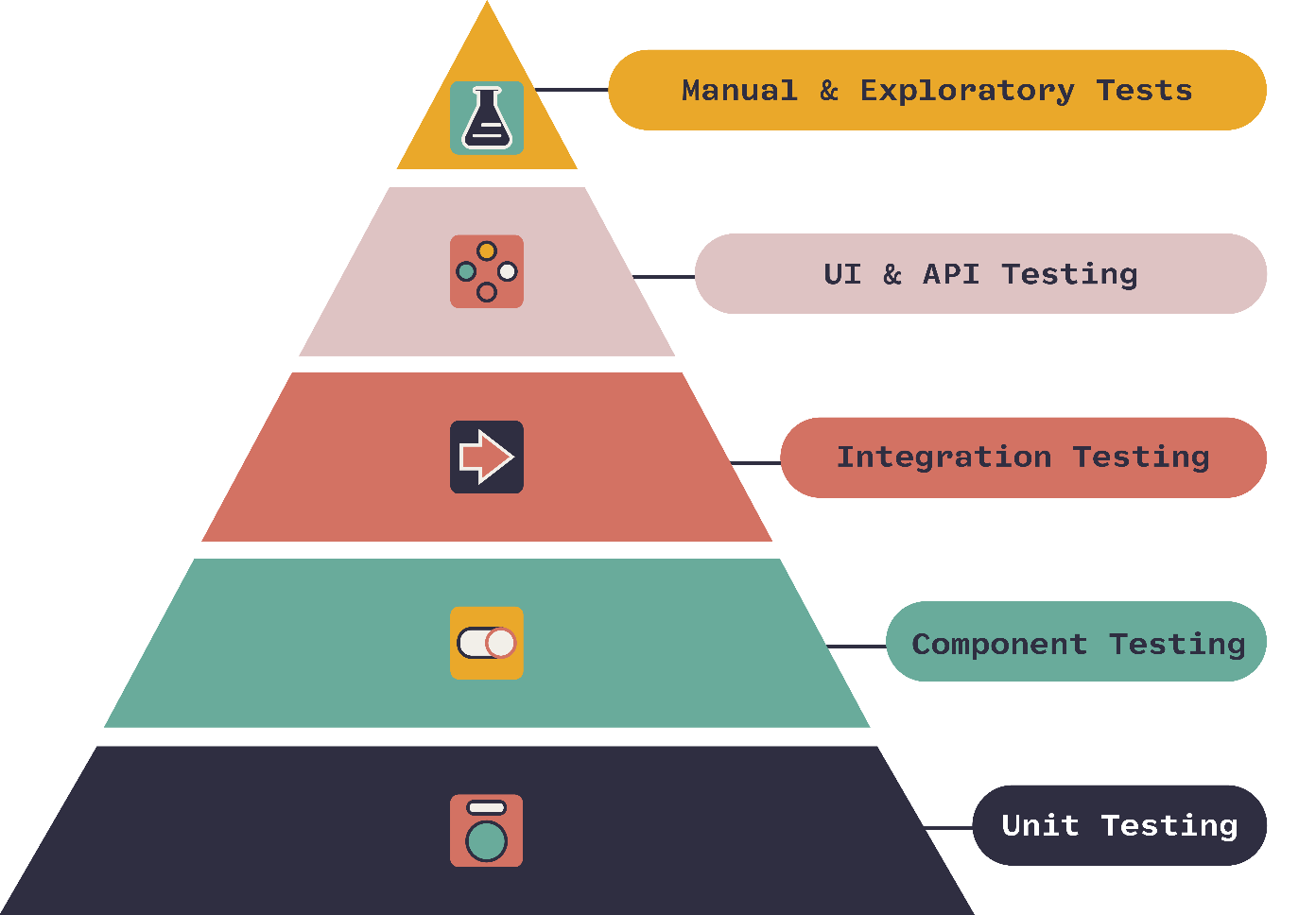
[**JUnit** 3](#_Toc123314664)

[**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 test a small “unit”/portion of the code

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

- We should also strive for 100% code coverage

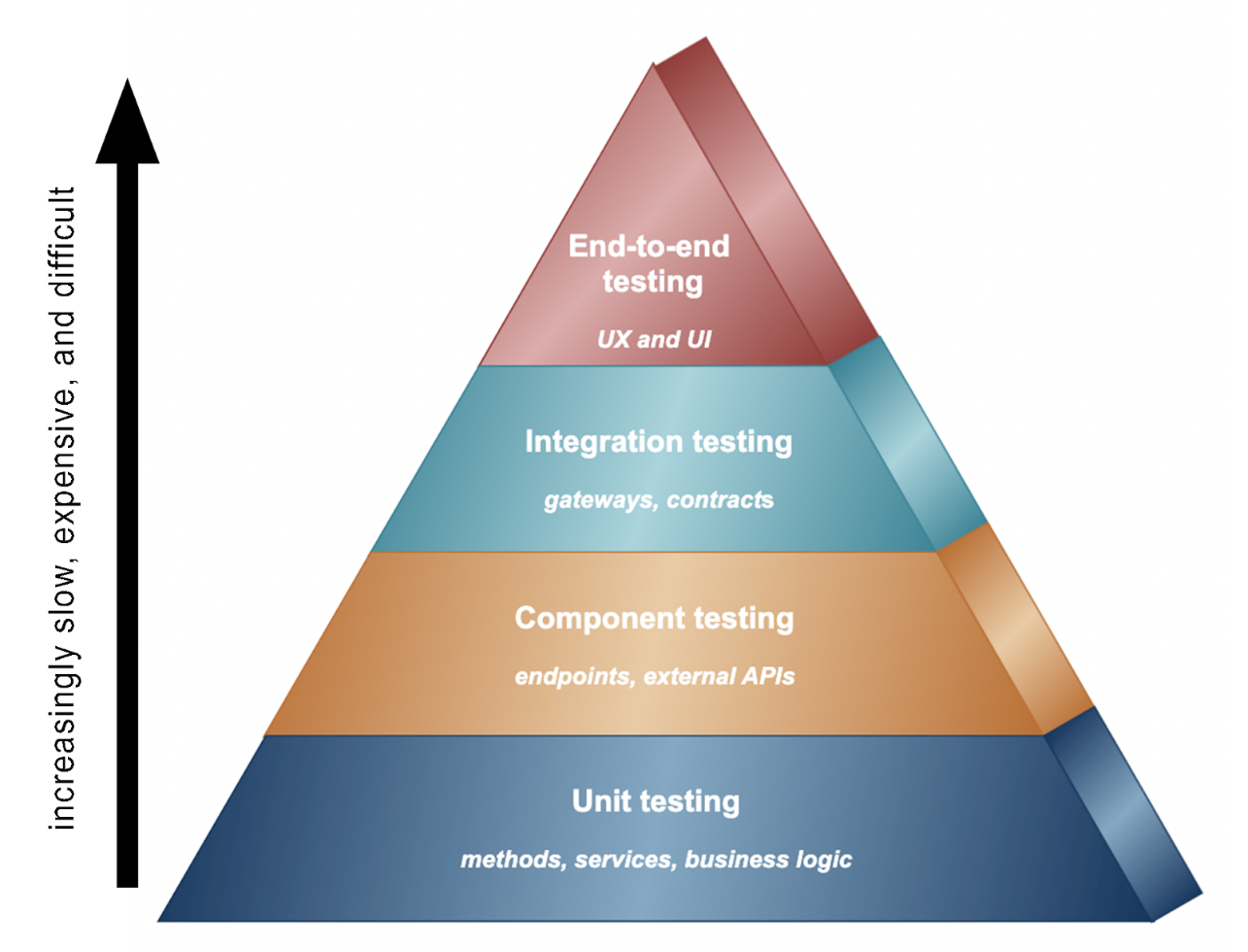
- Every single line of code in our application should be run through our unit tests

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

- Edge cases being maximum and minimum values, array.length, null, empty, zero, 1, prime, etc.

- Ex: integer overflow handling

## **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 small pieces (“units”) of the code

**Component Tests** – Tests larger parts of the code

**Integration Tests** - Tests how the parts of the code interact

**UI Tests**- Tests to make sure the user interface looks the way it is supposed to

**Manual Tests** – Tests are run by hand (vs automated tests)

**Regression** **Tests** – Tests are run when the code changes to make sure no old code breaks

**Stress** - Tests your application running on a server with lots of users or lots of data etc.

**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 a testing framework for writing and running automated unit tests in Java

- It is the most popular unit testing framework in Java

- It supports ­­­­­­­­­­­build up and tear down for testing (@BeforeAll, @AfterAll, @BeforeEach, @AfterEach)

- I can write specific methods to be automatically invoked before all tests or between all tests

- This same principle applies for after all tests or after each test

## **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 = do not depend on the data resulting from one test being used by the next test, you cannot depend on what order the tests will be run in,

ex: I should not have a test called listContainsLetterA() depend on a test that addsLetterAtoList() running first and succeeding. Instead set up a fresh list for the listContainsLetterA() to use.

- follow the 3 A’s

1. Assemble = Set up whatever data/objects you need to run the test

2. Act = call whatever methods you are testing

3. Assert = compare what you expect with the actual result of the methods executed

- **Repeatable** = you should be able to run the test more than once and get the same result (assuming nothing has changed in your test or project code), you should be able to use the test today, tomorrow, next year, etc. on my computer, on your computer, etc. does not depend on a live data set that may change, should instead be using a stable test database

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

- **Thorough** = aim for every line of code being tested, so in an if-else, there should be a test for both cases, switch should have a test for each case and when no case is satisfied, etc.

- 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**

1. 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

2. 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

3. It's also good to test an even number

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

4. Test with no values

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

5. Test with some values

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

6. Test with negative numbers

-- Often these will go unnoticed in development

-- Handles overflow that way (integer overflow)

7. Ask yourself what logical edge cases there are

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