# Testing

- Automated
- Manual
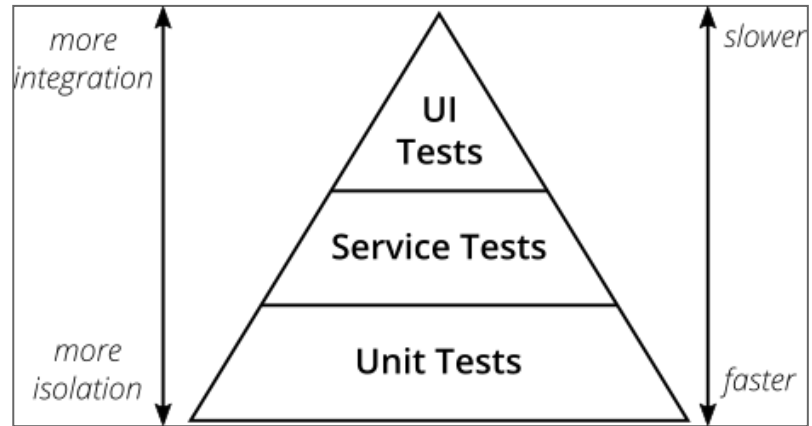
Ultimately about **confidence**

Almost *every* quality job will expect automated testing

# Testing Pyramid

Various Names
- E2E/UI
- Service/Integration
- Unit Tests



Testing Pyramid from **https://martinfowler.com/articles/practical-test-pyramid.html**

# Unit Tests

Most important tests:

- Easier to automate
- A "unit" is an indivisible piece of code
    - function, module, component
- Tested in **isolation**
    - means: by itself, without other parts
- Fast to run
    - usually seconds
- Durable
    - Code changes makes test fail only if input/output changes

# What does a unit test test?

The test:

- Confirms the input/output for the unit
- Defines behavior for bad input
    - If you don't test it, it is undefined
- Can serve as documentation for devs
- **Confirms behavior after change**
    - Allows you to change with confidence

# Integration Tests

- Fewer Integration Tests than Unit tests
- Slower to run (10s of seconds to minutes)
- Confirms assumptions of unit testing
    - Connects units into bigger pieces
        - or with other systems
            - DB
            - browsers
            - service calls
- More "brittle" than unit tests
    - tests can fail after changes for several reasons
        - requires you update the tests

# End-to-End Tests (E2E)

- Fewest tests
- Complete start-to-finish tests
- Slowest to run (minutes to 10s of minutes)
- Most "Brittle"
    - Can break for many reasons
- Hard to debug
- Only guarantee that everything works for end user

# Test Driven Development (TDD)

Practice of writing tests, then code

- Red-Green-Refactor cycle
- Easier when covering a mostly-known interface
- Harder when doing exploratory code
- Builds good habits
    - Create code that is easily testable
- Takes time to adjust to
    - You will code slower for ~6 months
    - Very rough estimate!
    - Then you are as fast or faster!
- Generates a lot of confidence

# Writing Tests After

- Fairly common practice
    - Write code until it mostly works
    - Then write tests
        - Check for all cases
        - Detect problems in the future
- Can have problems if your code is "hard to test"
    - Ex: uses hidden/private/internal methods
    - Ex: needs state only set by series of steps

# Unit Tests with JS

Multiple test systems exist:

- Jest (JS or Components)
- Karma (Technically *not* unit tests)
- Mocha/Chai, Jasmine
- Tape

Process for all:

- Load code
- Organize Tests
- Each test calls code with input
  - defines expected output

# Example unit test

```javascript
import compare from './compare';

describe('compare', () => {
  it('requires 2 inputs of the same length', () => {
    expect(compare()).toThrow('invalid input');
    // more checks here
  });

  it('returns the number of matching letters', () => {
    expect(compare('eat', 'eat')).toBe(3);
    expect(compare('eat', 'hop')).toBe(0);
  });

  it('works regardless of case', () => {
    expect(compare('eat', 'EAT')).toBe(3);
    expect(compare('EAT', 'eat')).toBe(3);
    expect(compare('EaT', 'eAT')).toBe(3);
  });
  //...
});
```

# So much more with testing!

- Web E2E tests automate the browser
    - Type and click for user
    - Several options
        - Selenium/webdriver
        - puppeteer
        - Cypress
- Automate for different browsers
    - Ex: Saucelabs
- Continuous Integration (CI)
    - Automates running tests on commit
        - Can automate release of next version!