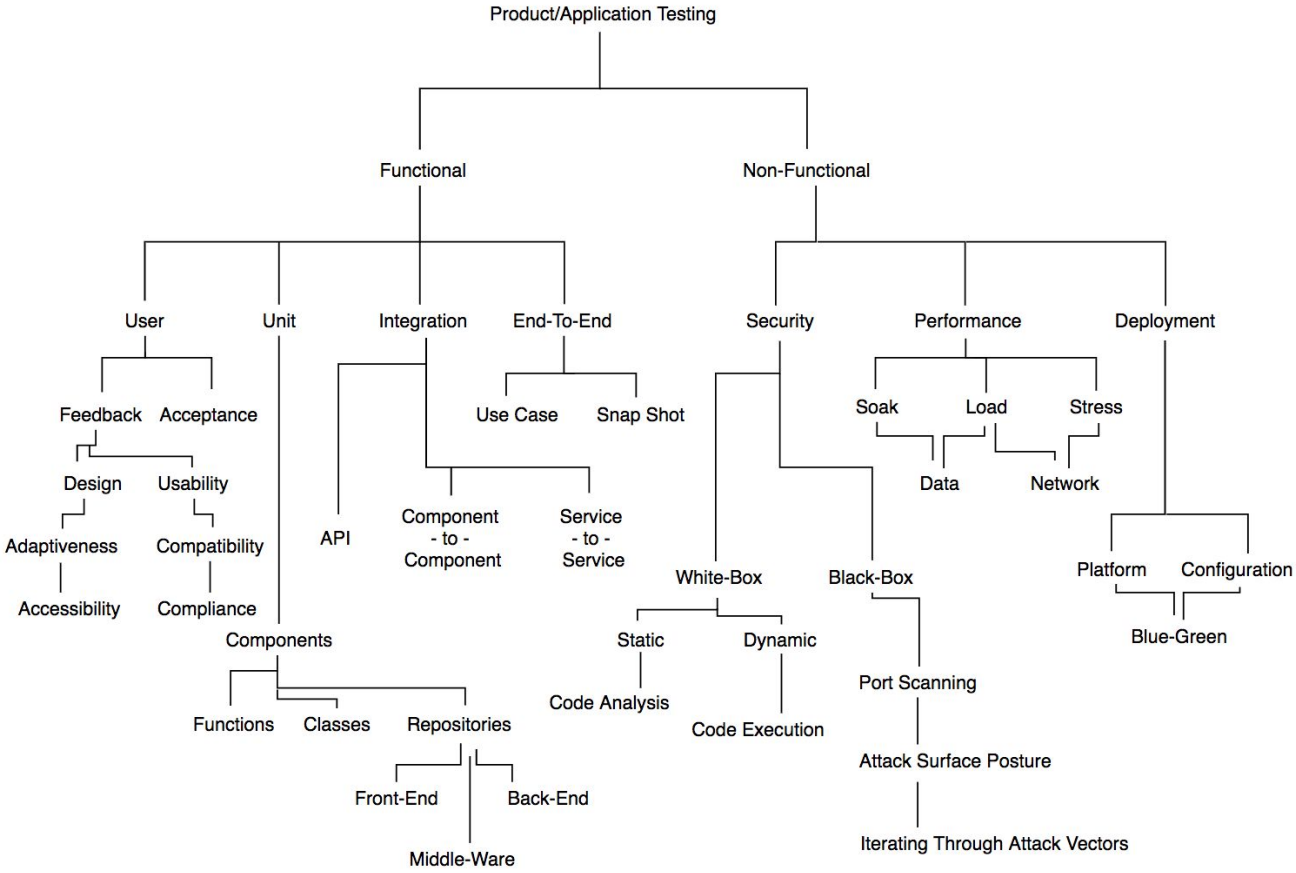
How Much Testing Is Enough Testing

Samantha Wong
GovTech
STACK 2020 - Dec 2020

You’ve Heard Of These Before

- ➔ Functional Testing
- ➔ Load Testing
- ➔ Integration Testing
- ➔ User Testing
- ➔ Smoke Testing
- ➔ Sanity Testing
- ➔ Blind Testing
- ➔ Whitebox Testing
- ➔ Blackbox Testing
- ➔ Unit Testing
- ➔ End to End Testing

Just because there is a name for it, doesn’t mean it exists


Product/Application Testing
Functional
Non-Functional
User
Unit
Integration
End-To-End
Security
Performance
Deployment
Feedback
Acceptance
Design
Usability
Adaptiveness
Compatibility
Accessibility
Compliance
API
Component - to - Component
Service - to - Service
Use Case
Snap Shot
Components
Functions
Classes
Repositories
Front-End
Back-End
Middle-Ware
White-Box
Black-Box
Static
Dynamic
Code Analysis
Code Execution
Soak
Load
Stress
Data
Network
Platform
Configuration
Blue-Green
Port Scanning
Attack Surface Posture
Iterating Through Attack Vectors

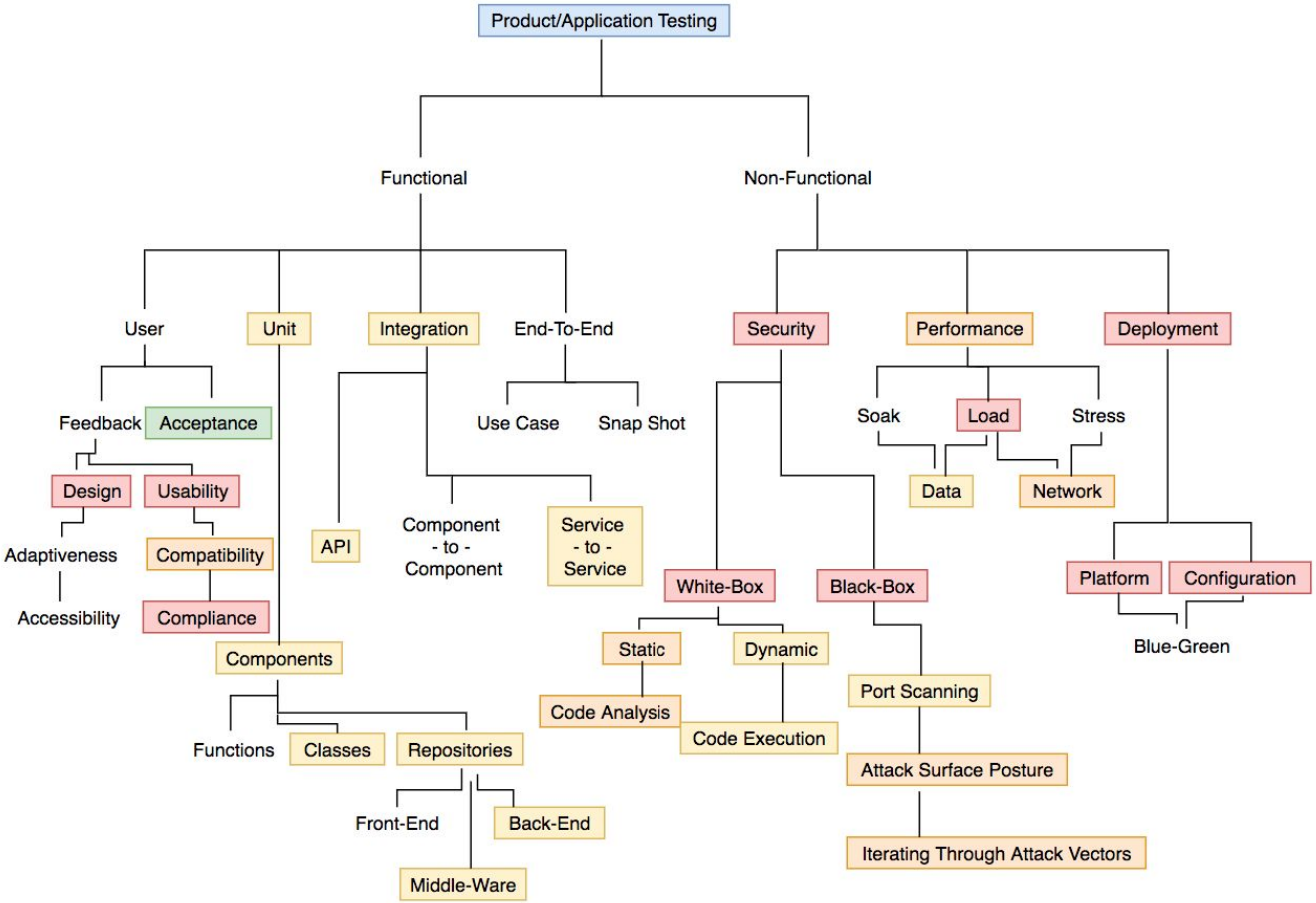
So, how?

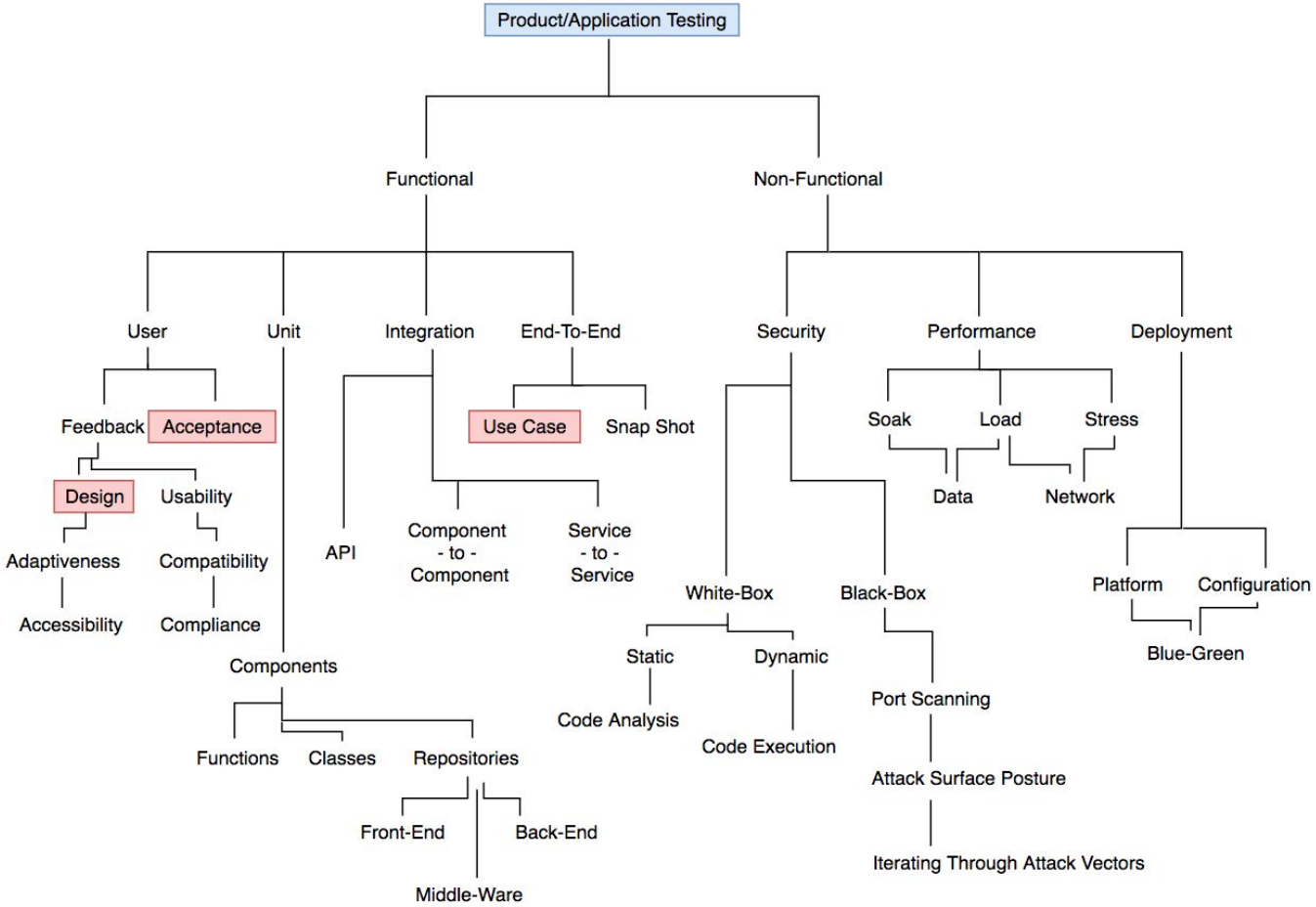
You should choose which testings to focus on based on your acceptable risks

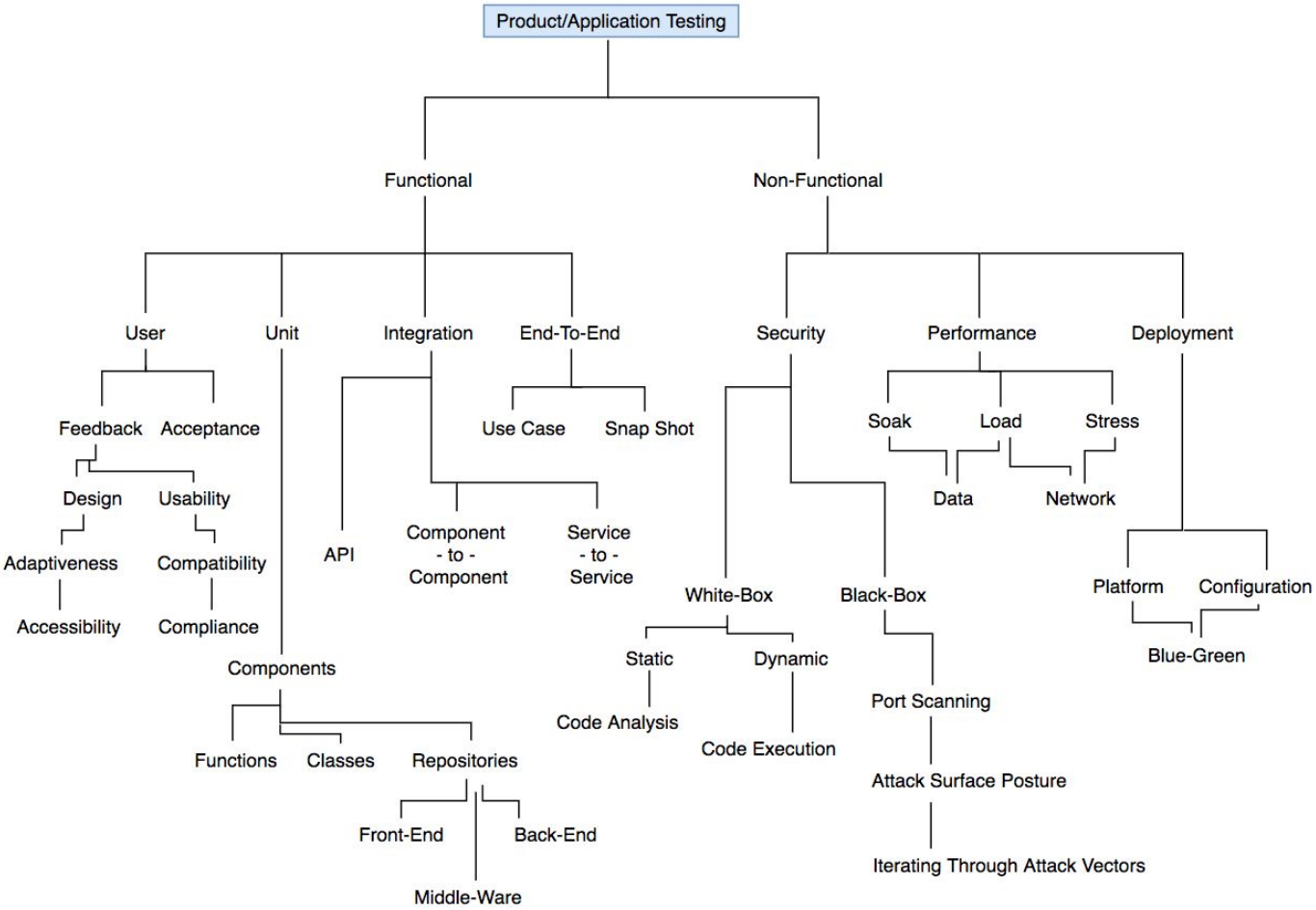
High-traffic user flows like a promotional flow on an e-commerce website

You should choose which testings to focus on based on the nature of your application

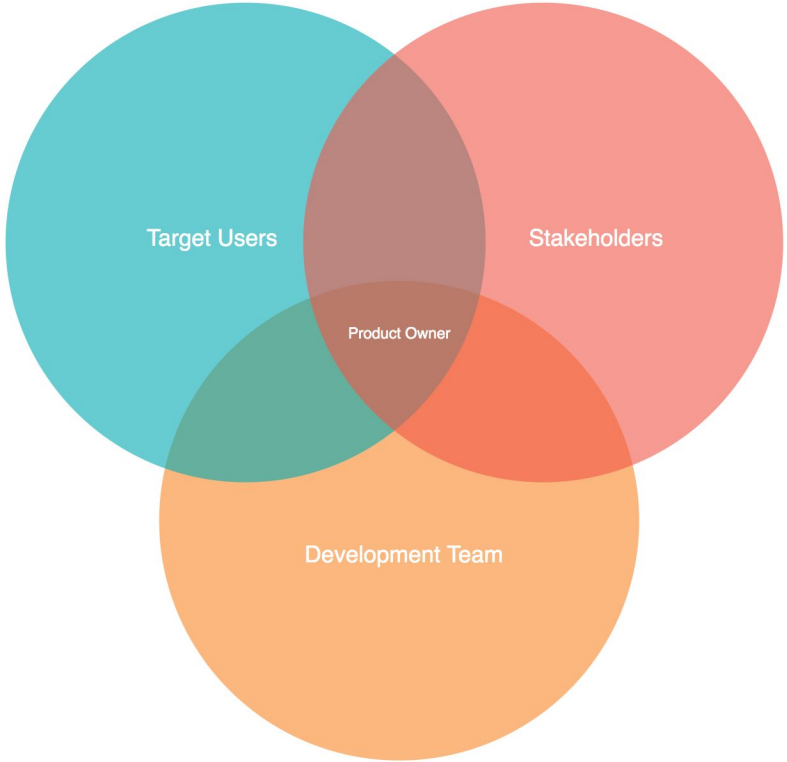
End-to-end testings with your third party agencies


Product/Application Testing
Functional
Non-Functional
User
Unit
Integration
End-To-End
Security
Performance
Deployment
Feedback
Acceptance
Use Case
Snap Shot
Soak
Load
Stress
Design
Usability
Data
Network
Adaptiveness
Compatibility
API
Component - to - Component
Service - to - Service
White-Box
Black-Box
Platform
Configuration
Accessibility
Compliance
Static
Dynamic
Blue-Green
Components
Code Analysis
Code Execution
Port Scanning
Functions
Classes
Repositories
Attack Surface Posture
Front-End
Back-End
Iterating Through Attack Vectors
Middle-Ware


Product/Application Testing
Functional
Non-Functional
User
Unit
Integration
End-To-End
Security
Performance
Deployment
Feedback
Acceptance
Design
Usability
Adaptiveness
Accessibility
Compatibility
Compliance
Components
Functions
Classes
Repositories
Front-End
Back-End
Middle-Ware
API
Component - to - Component
Service - to - Service
Use Case
Snap Shot
White-Box
Black-Box
Static
Dynamic
Code Analysis
Code Execution
Port Scanning
Attack Surface Posture
Iterating Through Attack Vectors
Soak
Load
Stress
Data
Network
Platform
Configuration
Blue-Green


Product/Application Testing
Functional
Non-Functional
User
Unit
Integration
End-To-End
Security
Performance
Deployment
Feedback
Acceptance
Use Case
Snap Shot
Soak
Load
Stress
Design
Usability
Data
Network
Adaptiveness
Compatibility
API
Component - to - Component
Service - to - Service
Platform
Configuration
Accessibility
Compliance
White-Box
Black-Box
Blue-Green
Components
Static
Dynamic
Port Scanning
Code Analysis
Code Execution
Functions
Classes
Repositories
Attack Surface Posture
Front-End
Back-End
Iterating Through Attack Vectors
Middle-Ware

People Who Define The Application and Its Acceptable Risks



How to guarantee the quality of tests

Designing (good) tests are hard!

Tests check that expectations are met

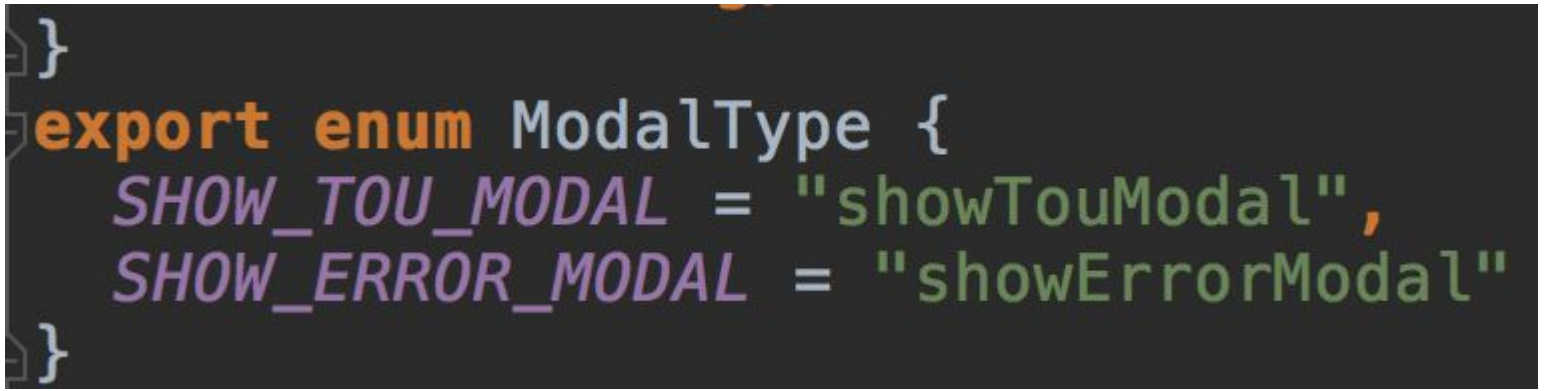
Tests alert when expectations are not met

Know What You Are Testing

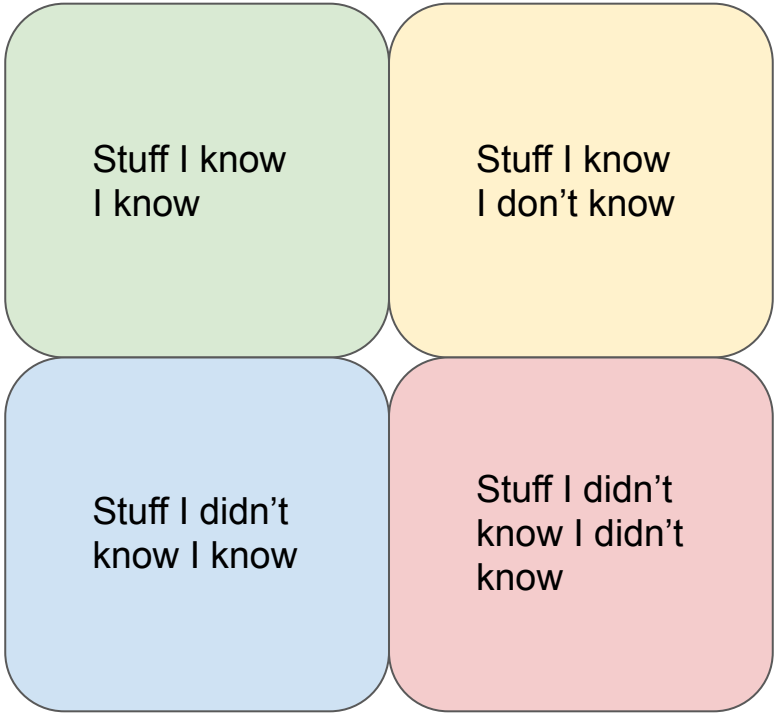
More Doesn’t Mean Better

```
it( title: "should show modal for the term of use when termsOfUseAccepted is false ", fn: async () => {
  // @ts-ignore
  URLSearchParamsPolyfill.mockImplementation(() => mockGetNull);
  const propsWithFalseTermsOfUse = {
    ...props,
    termsOfUseAccepted: false
  };
  wrapper = await shallow(<Login {...propsWithFalseTermsOfUse} />);

  expect(wrapper.update().state( key: "showTouModal")).toBe( expected: true);
});
it( title: "should show modal for handling error when returning state is wrong ", fn: async () => {
  // @ts-ignore
  URLSearchParamsPolyfill.mockImplementation(() => mockGetNull);
  const propsWithFalseTermsOfUse = {
    ...props,
    termsOfUseAccepted: false
  };
  wrapper = await shallow(<Login {...propsWithFalseTermsOfUse} />);

  expect(wrapper.update().state(ModalType.SHOW_TOU_MODAL)).toBe( expected: true);
});
```

Because...Sometimes They Are The Same



Write Tests Before Development



The Spheres of Knowing

Verify Your Tests Do Something

```java
@Test
public void sumOfOneAndThreeShouldBeFour() throws Exception {
  int result = new Calculator().sum(1,3);

  assertEquals(4, result)
}
```

https://jeroenmols.com/blog/2017/11/28/coveragproblem/

```
public class Calculator {
  public int sum(int a, int b) {
    return 4;
  }
}
```

https://jeroenmols.com/blog/2017/11/28/coveragproblem/

```
@Test
function test_sum(input1, input2) {
    assert(sum(1, 3) == 4)
    assert(sum(0, 0.0) == 0)
    assert(sum(-1, 0) == -1)
    assert(sum(1, -100.0) == -99)
}
```

Good tests **should** break when things fail

One more thing..

Don’t trust code coverage tools

They are a source of information,
not a source of quality.

End Slide