Test Driven Development with Blazor

Slides up at scottsauber.com

scottsauber

Audience

- Familiar with Blazor
- Interested in learning TDD

Agenda

- What is TDD?
- Why TDD?
- Tools you can use
- What do I test?
- Live Demos

Goals

- Learn “best practices*” for writing frontend tests
- Share with .NET community testing learnings from React community
- Learn how to TDD (with Blazor!)

* Synonym for “Just My Opinions” and I’ll probably find a way I like better in the future

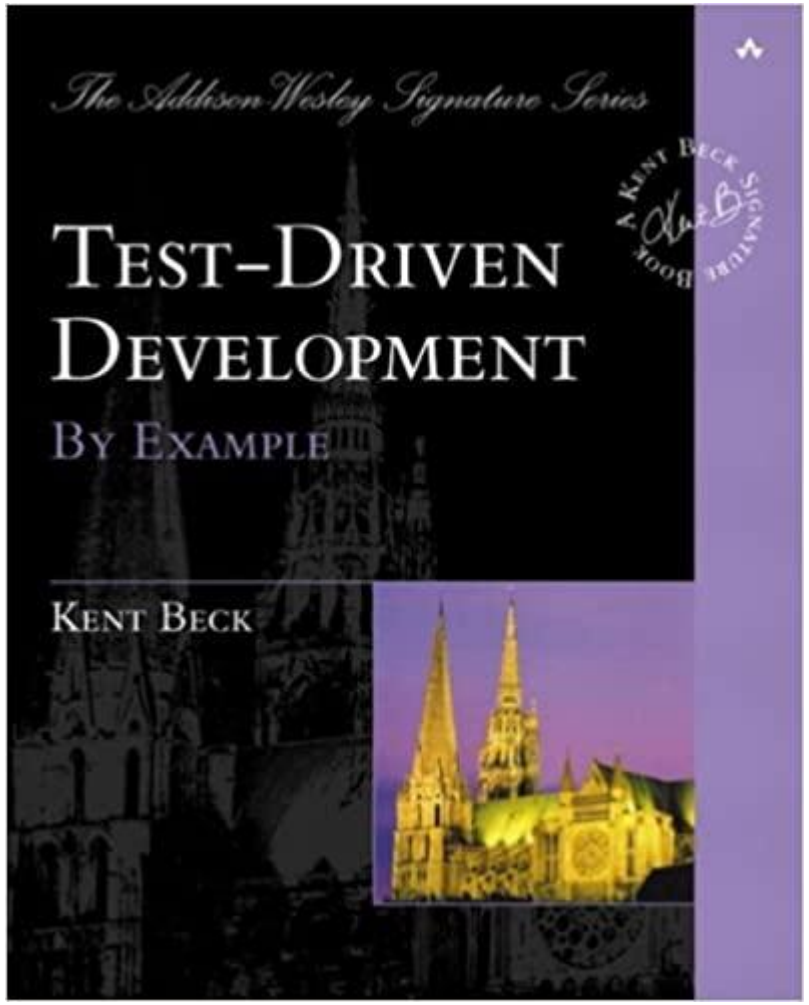
Who am I?

- Director of Engineering at Lean TECHniques
- Microsoft MVP
- Dometrain Author
- Redgate Community Ambassador
- Co-organizer of Iowa .NET User Group
- Used Blazor, Angular, or React since 2015

Why do we write tests?

- We want confidence our application works
- Minimize manual verification
- Document behavior through tests

What is TDD?



scottsauber

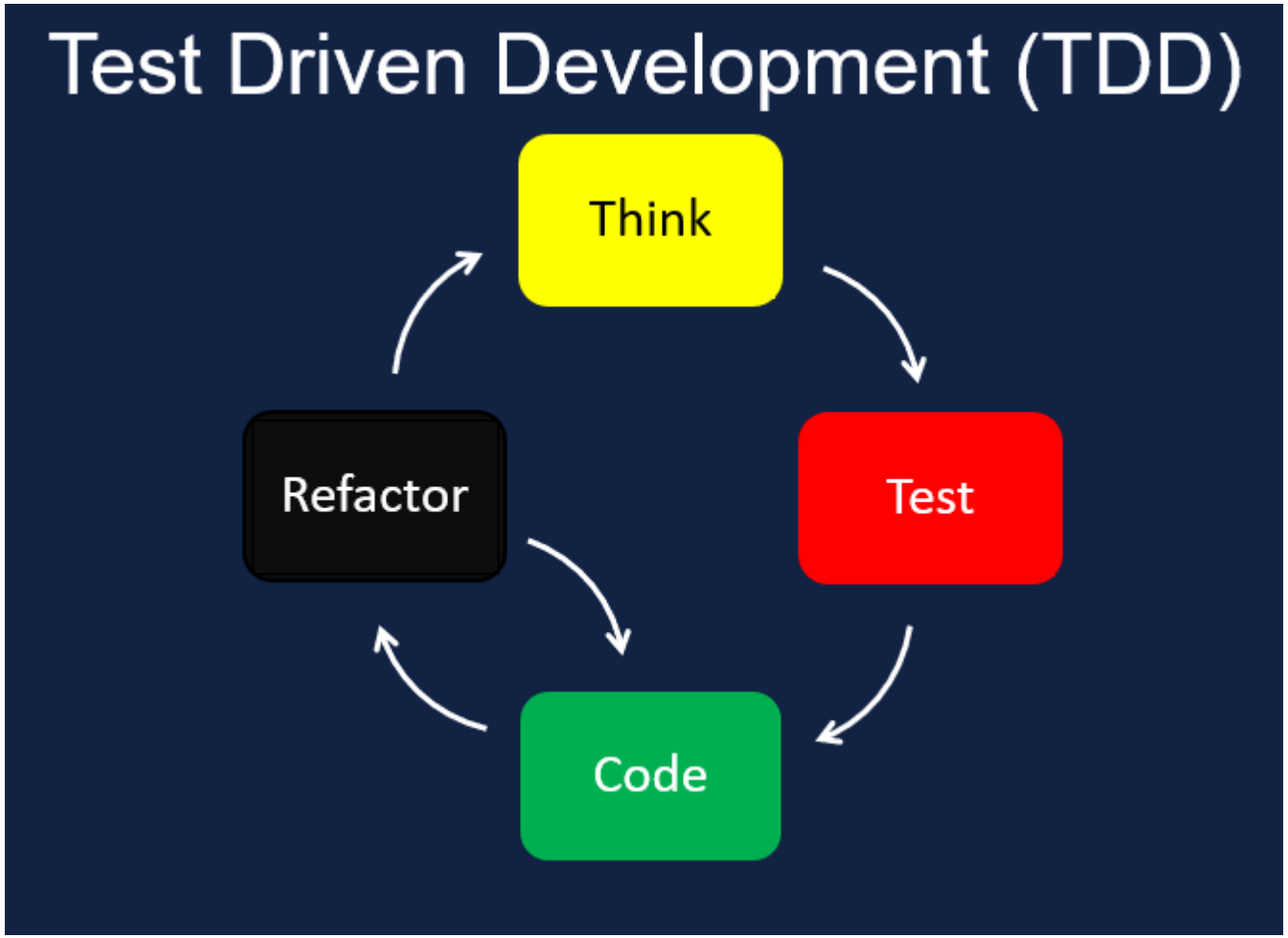
What is TDD?

- Methodology for writing code (not just tests)
- You write the test BEFORE you write the production code

How to TDD?

1. Think
2. Write a test that describes the behavior you want to see
3. Run the test and watch it fail *for the right reason*
4. Write code to make it pass
5. Refactor
6. Repeat

How to TDD?



“If you haven’t seen a test fail, you don’t know if it works.”

Eric Evans

Why Test Driven Development?

- Work in small steps (minimizes waste, minimizes WIP)
- Focus
- Much less time in the debugger
- Thinking through failure states
- Confidence
- Design feedback, hard to write test? Design might be wrong
- Oh yeah… regression tests are nice too

What is NOT TDD?

- TDD is NOT a synonym for writing tests
- TDD is NOT writing multiple tests up front before writing any production code
- TDD does NOT mean no bugs ever (just less)
- TDD is not good for adding tests to existing production code
- TDD zealots do more harm than good

Applying TDD to Blazor

Introduction to Tools

- xUnit
- Shouldly
- bUnit

xUnit

- Test framework
- Used by Microsoft to test .NET

xUnit

```
[Fact]
public void ShouldIncrementCountWhenClickingIncrementButton()
{
    using var testContext = new TestContext();

    var component = testContext.RenderComponent<Counter>();
    var incrementButton :IElement = component.Find( cssSelector: "button");
    incrementButton.Click();

    var currentCount :IElement = component.Find( cssSelector: "[role='status']");
    currentCount.TextContent.ShouldBe( expected: "Current count: 1");
}
```

scottsauber

Shouldly

- Assertion library
- ~80M downloads on NuGet
- Cleaner assertions

Shouldly

```
[Fact]
public void ShouldIncrementCountWhenClickingIncrementButton()
{
    using var testContext = new TestContext();

    var component = testContext.RenderComponent<Counter>();
    var incrementButton :IElement = component.Find( cssSelector: "button");
    incrementButton.Click();

    var currentCount :IElement = component.Find( cssSelector: "[role='status']");
    currentCount.TextContent.ShouldBe( expected: "Current count: 1");
}
```

scottsauber

bUnit

- Helper library for testing Blazor
- Renders components
- Queries for the DOM
- Inject fake dependencies
- Fakes for various things (i.e. NavigationManager)

bUnit

```
[Fact]
public void ShouldIncrementCountWhenClickingIncrementButton()
{
    using var testContext = new TestContext();

    var component = testContext.RenderComponent<Counter>();
    var incrementButton :IElement = component.Find( cssSelector: "button");
    incrementButton.Click();

    var currentCount :IElement = component.Find( cssSelector: "[role='status']");
    currentCount.TextContent.ShouldBe( expected: "Current count: 1");
}
```

scottsauber

What should I test?

- Behavior
- Not that CSS classes exist or any other attributes directly exist
- You can’t prove your app looks good with tests
- Behavior
- If I can delete code that breaks your app, but your tests don’t – that’s a problem
- If my tests break but my application isn’t broken - that’s a problem
  - Flaky Test?
  - Implementation detail?

```
[Fact]
public void ShouldIncrementCountWhenClickingButton()
{
    using var testContext = new TestContext();

    var component = testContext.RenderComponent<Counter>();
    var button:IElement = component.Find(cssSelector:"button");
    button.Click();

❌  component.Instance.CurrentCount.Should().Be(1);
}
```

Current Count is an implementation detail, not behavior

scottsauber

```
[Fact]
public void ShouldIncrementCountWhenClickingButton()
{
    using var testContext = new TestContext();

    var component = testContext.RenderComponent<Counter>();
    var button:IElement = component.Find(cssSelector:"button");
    button.Click();

    var currentCount:IElement = component.Find(cssSelector:"[role='status']");
    currentCount.MarkupMatches("""<p role="status">Current count: 1</p>""");
}
```

The HTML is an implementation detail, not behavior

scottsauber

```
[Fact]
public void ShouldIncrementCountWhenClickingIncrementButton()
{
    using var testContext = new TestContext();

    var component = testContext.RenderComponent<Counter>();
    var incrementButton :IElement = component.Find( cssSelector: "button");
    incrementButton.Click();

    var currentCount :IElement = component.Find( cssSelector: "[role='status']");
    currentCount.TextContent.ShouldBe( expected: "Current count: 1");
}
```

This is the behavior you care about!

scottsauber

```
[Fact]
public void ShouldIncrementCountWhenClickingButton()
{
    using var testContext = new TestContext();

    var component = testContext.RenderComponent<Counter>();
    var button:IElement = component.Find(cssSelector:"button");
    button.Click();

    component.Instance.CurrentCount.Should().Be(1);
}
```

```
[Fact]
public void ShouldIncrementCountWhenClickingButton()
{
    using var testContext = new TestContext();

    var component = testContext.RenderComponent<Counter>();
    var button:IElement = component.Find(cssSelector:"button");
    button.Click();

    var currentCount:IElement = component.Find(cssSelector:"[role='status']");
    currentCount.MarkupMatches("""<p role="status">Current count: 1</p>""");
}
```

```
[Fact]
public void ShouldIncrementCountWhenClickingButton()
{
    using var testContext = new TestContext();

    var component = testContext.RenderComponent<Counter>();
    var button:IElement = component.Find(cssSelector:"button");
    button.Click();

    var currentCount:IElement = component.Find(cssSelector:"[role='status']");
    currentCount.TextContent.Should().Be("Current count: 1");
}
```

scottsauber

“The more your tests resemble the way your software is used the more confidence they can give you.”

Kent C Dodds

react-testing-library creator

What should I NOT test?

- You can’t test if your app looks good
- Do NOT test implementation details
- Avoid using MarkupMatches
- Too many implementation details (i.e. classes, DOM nodes, etc.)
- Avoid using .Instance
- Too many implementation details (i.e. Property, Methods, etc.)

What should I NOT test?

- Avoid using snapshots for your Blazor components… (mostly)
- Snapshots don’t capture desired behavior
- Too many implementation details (i.e. classes, DOM nodes, etc.)
- Results in I see people start blindly accepting changes
- Can’t TDD it because it relies on the final output
- Only use snapshots when doing a total refactor but output should be the same
- Then delete the test

Live Coding!

Slight TDD Detour

“Remove everything that has no relevance to the story. If you say in the first chapter that there is a rifle hanging on the wall, in the second or third chapter it absolutely must go off. If it's not going to be fired, it shouldn't be hanging there.”

Anton Chekhov

Chekhov's Gun Applied to Testing

```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    var customer = new Customer
    {
        FirstName = "SpongeBob",
        LastName = "",
        Address = "123 Pineapple",
        BirthDate = new DateOnly( year: 1999,  month: 5,  day: 1),
    };

    var result = new CustomerValidator().Validate(customer);

    result.Errors.Should().Contain(error :ValidationFailure => error.ErrorMessage == "Last Name is required.");
}
```

scottsauber

Chekhov's Gun Applied to Testing

```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    var customer = CreateValidCustomer();
    customer.LastName = "";

    var result = new CustomerValidator().Validate(customer);

    result.Errors.Should().Contain(error :ValidationFailure => error.ErrorMessage == "Last Name is required.");
}
```

scottsauber

Chekhov's Gun Applied to Testing

```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    _customer.LastName = "";

    var result = new CustomerValidator().Validate(_customer);

    result.Errors.Should().Contain(error:ValidationFailure => error.ErrorMessage == "Last Name is required.");
}
```

scottsauber

Chekhov's Gun Applied to Testing

```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    _customer.LastName = "";

    var result = _customerValidator.Validate(_customer);

    result.Errors.Should().Contain(error:ValidationFailure => error.ErrorMessage == "Last Name is required.");
}
```

scottsauber

Chekhov's Gun Applied to Testing

```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    var customer = new Customer
    {
        FirstName = "SpongeBob",
        LastName = "",
        Address = "123 Pineapple",
        BirthDate = new DateOnly(year: 1999, month: 5, day: 1),
    };

    var result = new CustomerValidator().Validate(customer);

    result.Errors.Should().Contain(error :ValidationFailure => error.ErrorMessage == "Last Name is required.");
}
```

```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    _customer.LastName = "";

    var result = _customerValidator.Validate(_customer);

    result.Errors.Should().Contain(error :ValidationFailure => error.ErrorMessage == "Last Name is required.");
}
```

scottsauber

</ ChekhovsGun>

bunit.web.query

- More ways to query the DOM that are less implementation specific
- React Testing Library style
- Queries promote A11y
- More coming in next 6 months

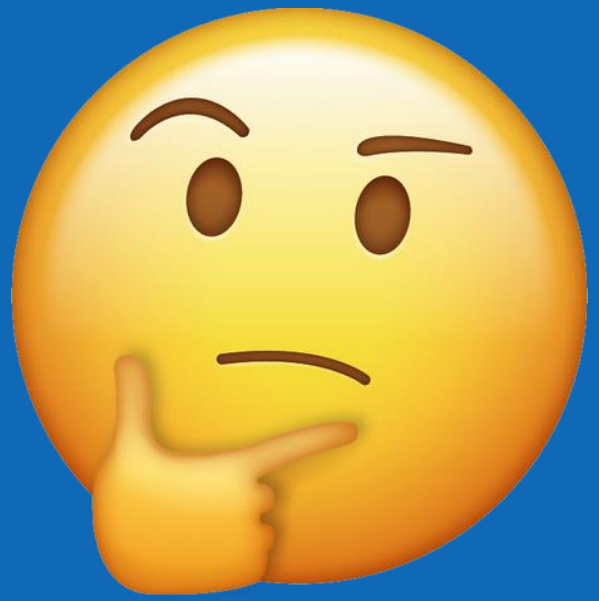
Live Coding!

How can I get started with TDD?

- When you get a bug report coming in
- Write a failing test that proves the bug exists
- Make it pass

But I don’t have time!

Why?



My boss won’t let me!

What about this person?

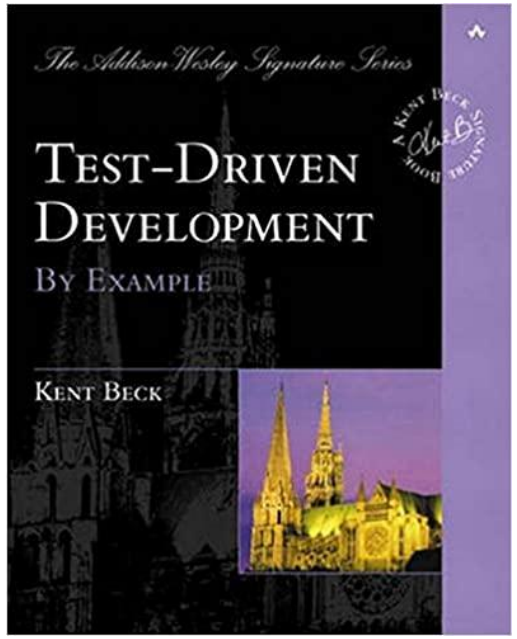
You don’t get better
at TDD
by NOT doing TDD

Takeaways

- Why you should TDD
- How to test Blazor
- What to test in Blazor
- How to get started TDDing Blazor

Resources

- TDD By Example by Kent Beck
- Write Tests blog post by Kent C Dodds
- https://github.com/scottsauber/talks
- This slide deck



Questions?

Email: ssauber@leantechniques.com

Slides up at scottsauber.com

scottsauber

Thanks!

Slides up at scottsauber.com

scottsauber