# Test Driven Developmet with C# From Padawan to Jedi

Slides at scottsauber.com

scottsauber

## Audience

- .NET Developers
- Anyone interested in testing
- Not sure where to begin with TDD
- Seasoned TDDers
- Basic knowledge of testing

## Agenda

- What different types of tests are there?
- What is TDD?
- Why TDD?
- What do I test?
- Live Demos

## Goals

- Learn “best practices*” for writing tests
- Learn how to TDD with C#

* 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
- Co-organizer of Iowa .NET User Group
- Microsoft MVP
- Friend of Redgate
- Blog at scottsauber.com

# Testing

## Why do we write tests?

- Confidence our application works
- Minimize manual work
- Document behavior through tests

## What kinds of tests are there?

- Unit
- Integration
- End to End
- …and more (Load, Stress, Smoke, etc)

## Unit Tests

- FIRST
- **F**ast
- **I**solated
- **R**epeatable
- **S**elf-validating
- **T**imely
- “Solitary”

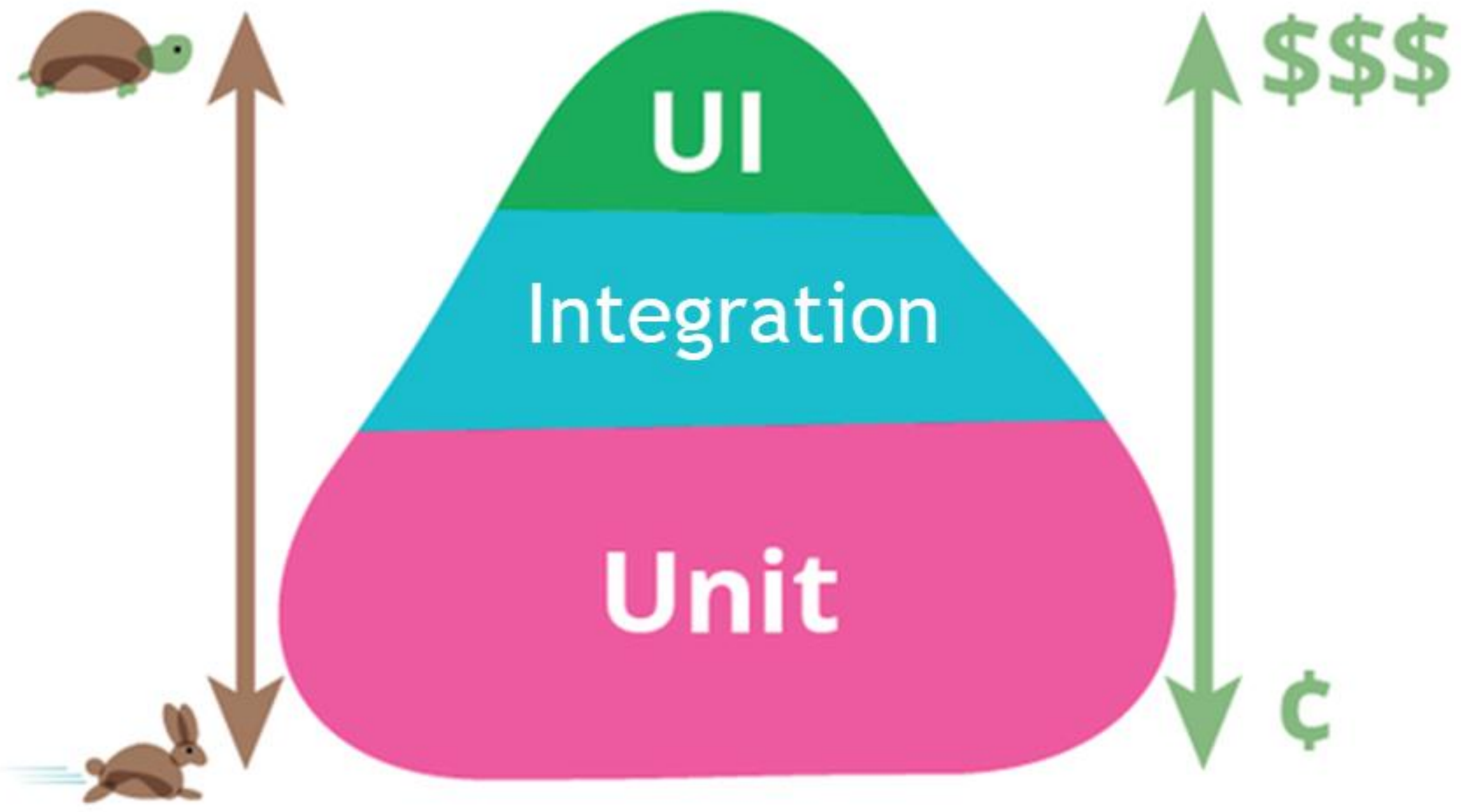
## Integration Tests

- Invokes multiple parts of a system together
- May have external dependencies
- Usually less deterministic if there are uncontrolled dependencies
- Useful for testing legacy code
- “Sociable”

## End to End Tests

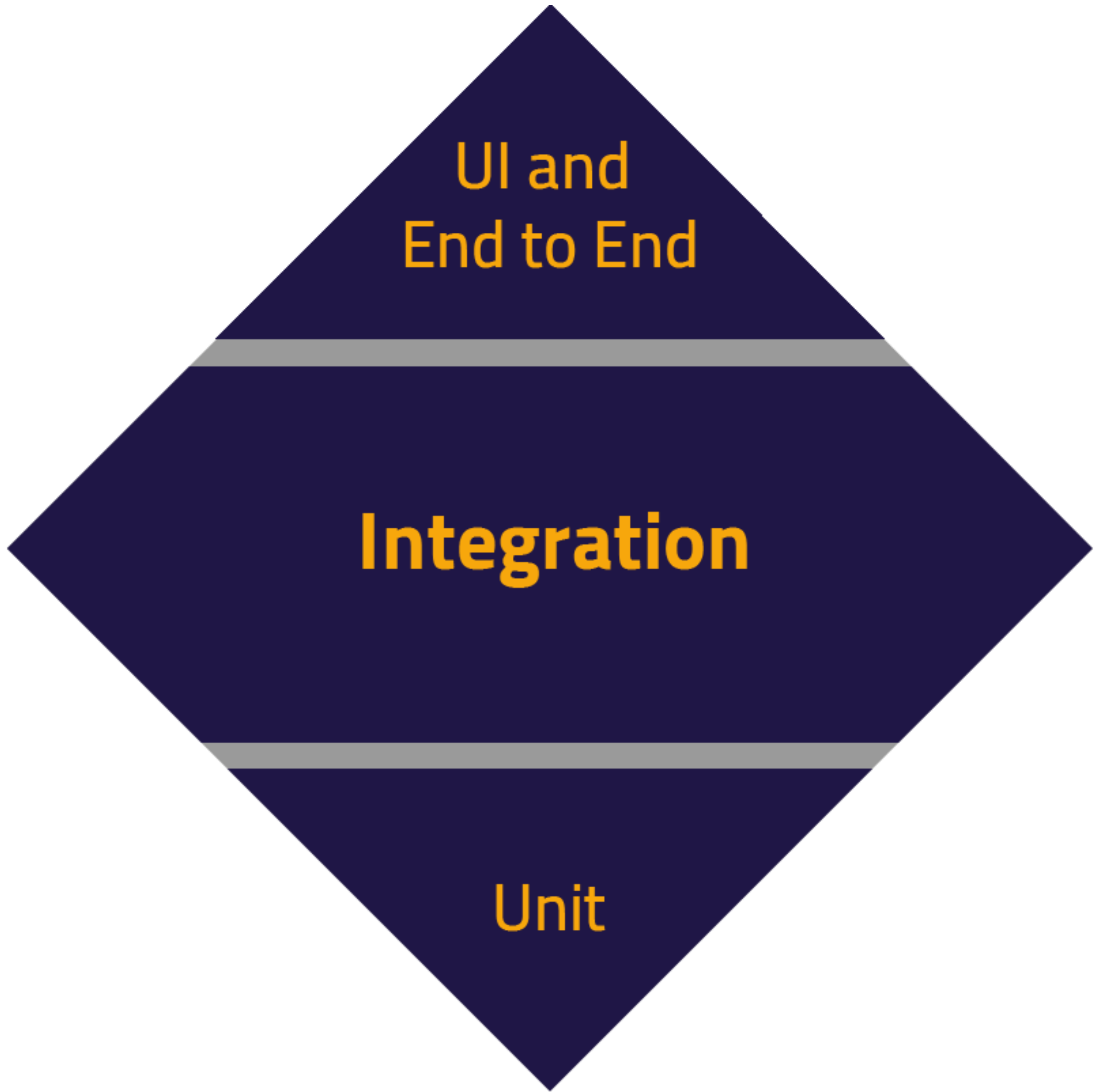
- Type of integration test
- Invokes system from entry point to its end
- Will have external dependencies
- Will not be in memory
- Usually slowest form of integration test
- Usually less deterministic if there are uncontrolled dependencies

## How many do I create of each?



scottsauber

## How many do I create of each?



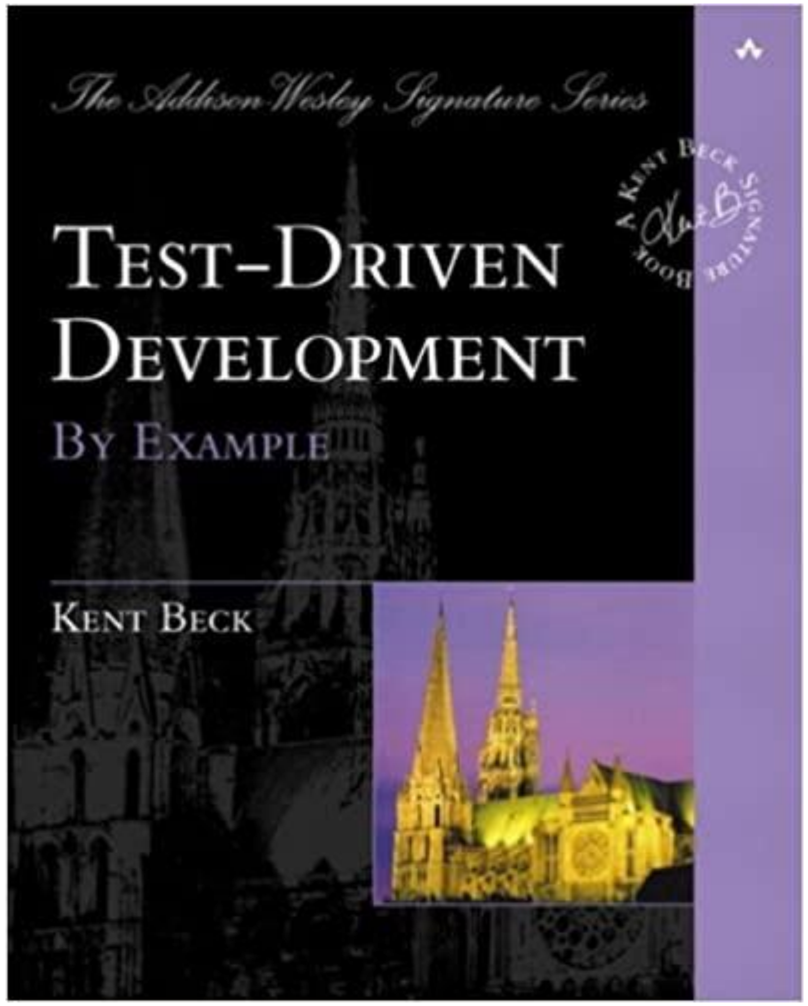
scottsauber

## How many do I create of each?

scottsauber

## Test Driven Development

## What is TDD?

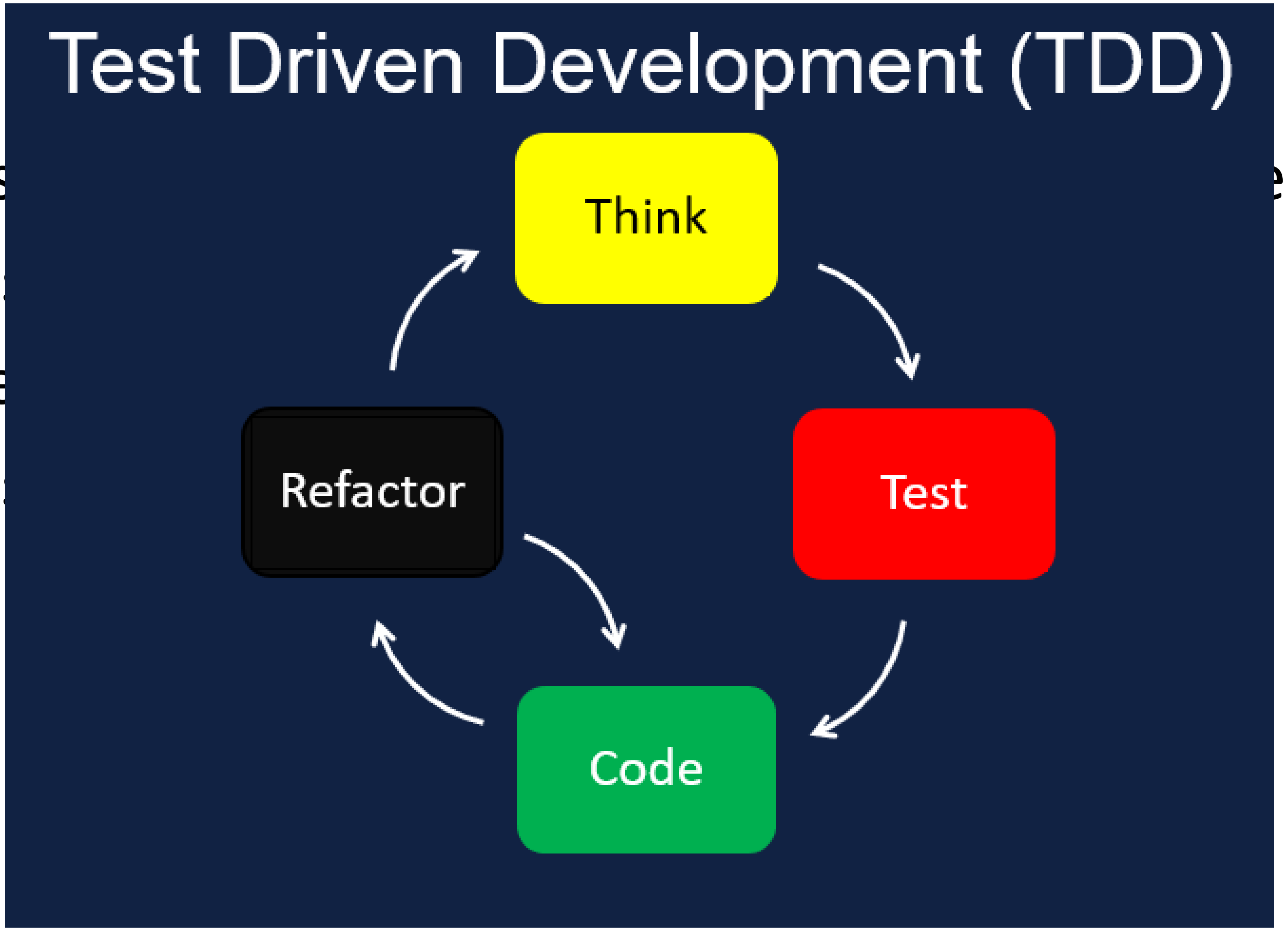

The Addison Wesley Signature Series
A Kent Beck Signature Book
Test-Driven Development
By Example
Kent Beck

## What is TDD?

- A methodology for writing code (not tests)
- You write the test BEFORE you write the production code

## How do I TDD?

- Think
- Write a tes                                        ee
- Run the te
- Write code
- Run the te
- Refactor
- Repeat



scottsauber

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

Eric Evans

## 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

## Why TDD?

- 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 should I test?

- Behavior
- If I can delete code that breaks your app, but no test breaks… ⚠
- If my tests are broken, but my application isn't… ⚠
- Don't use snapshots (…mostly)
- Snapshots don't capture desired behavior
- You can't TDD snapshots
- Snapshots can be useful when doing a huge refactor but output should be the same
- Use them for that - then delete the snapshot

“I’m not a great programmer. I’m a good programmer with great habits.”

Kent Beck

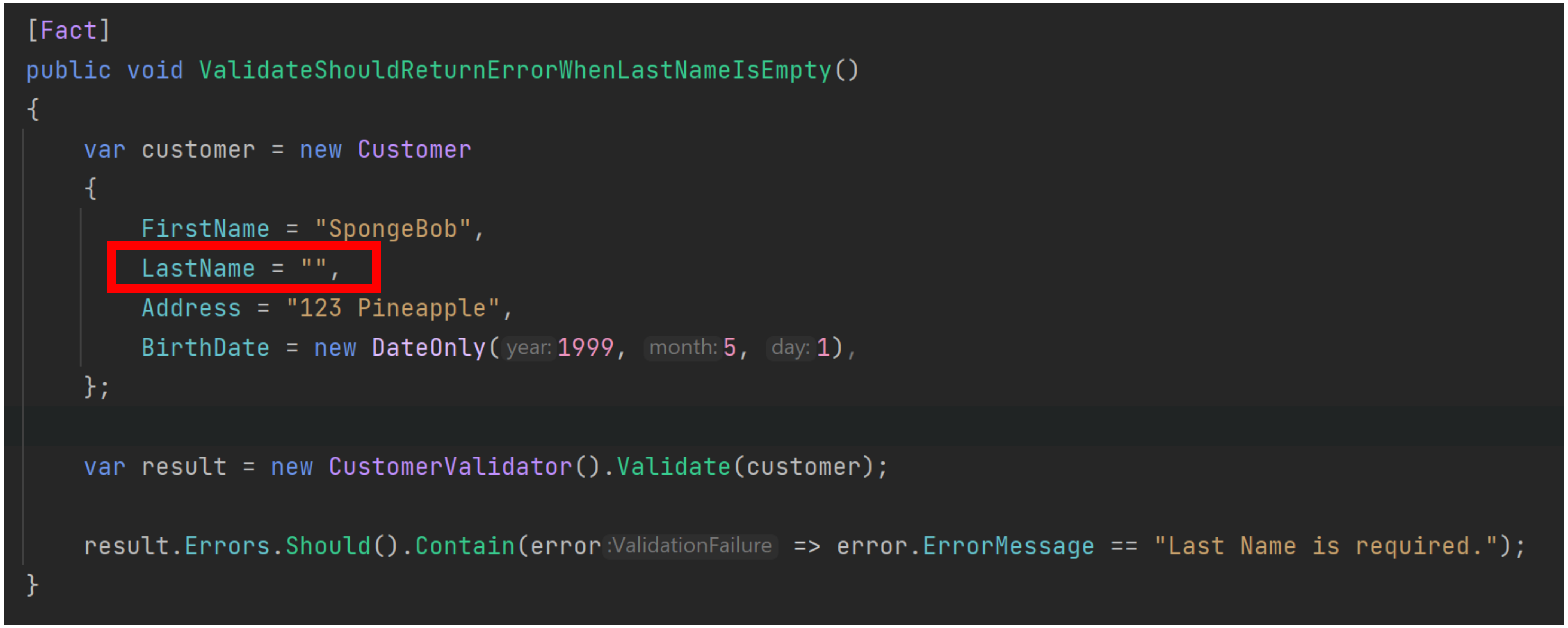
## TDD Demo

## 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



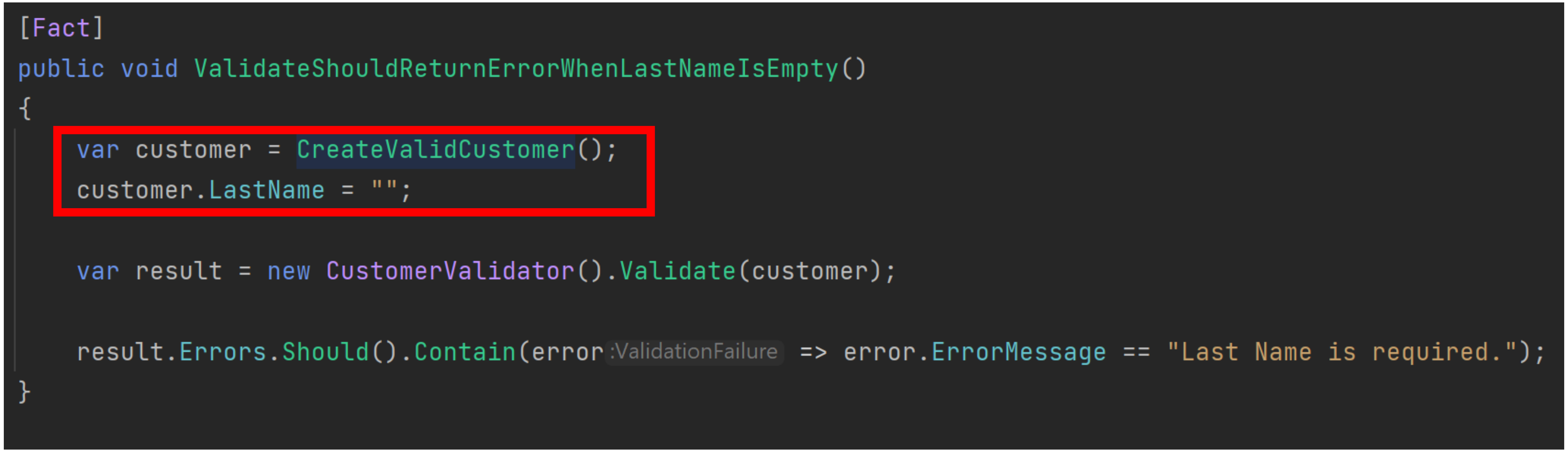
```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    var customer = new Customer
    {
        FirstName = "SpongeBob",
        LastName = "",
        Address = "123 Pineapple",
        BirthDate = new DateOnly(year: 1999, month: 5, day: 1),
    };

    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



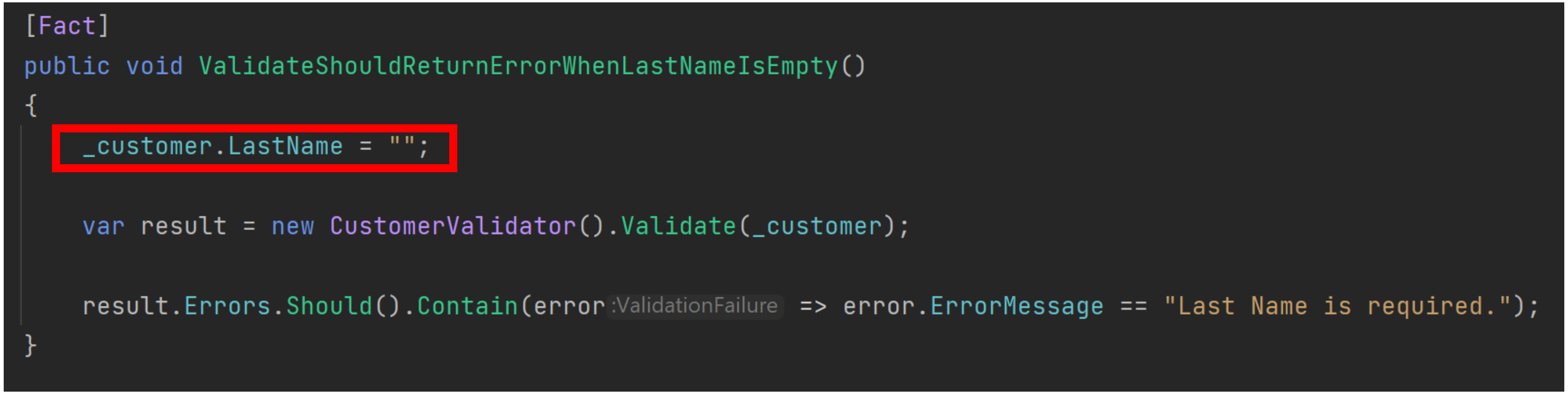
```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    var customer = CreateValidCustomer();
    customer.LastName = "";

    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



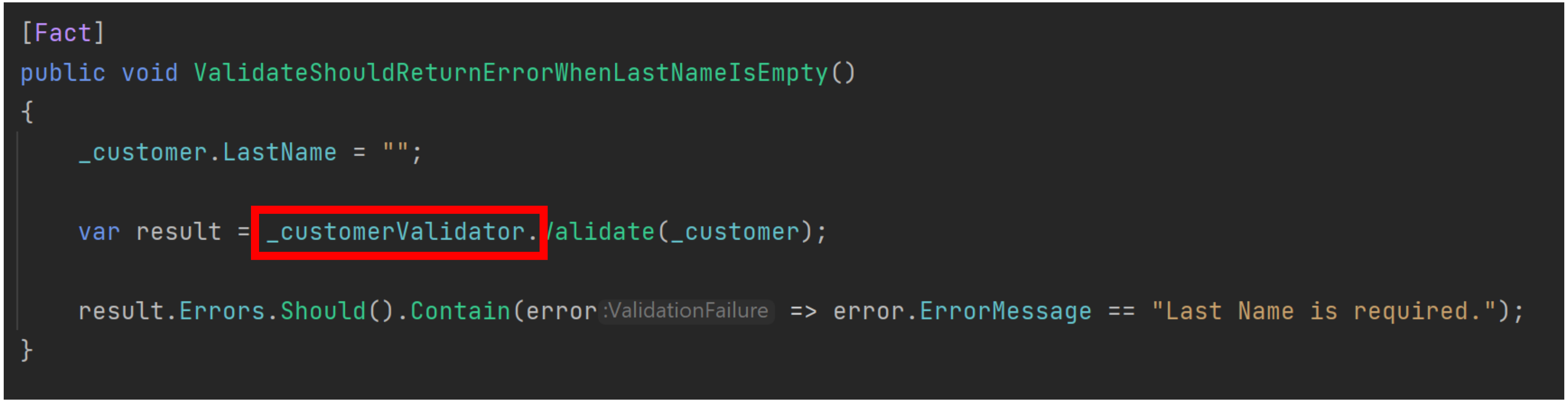
```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    _customer.LastName = "";

    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



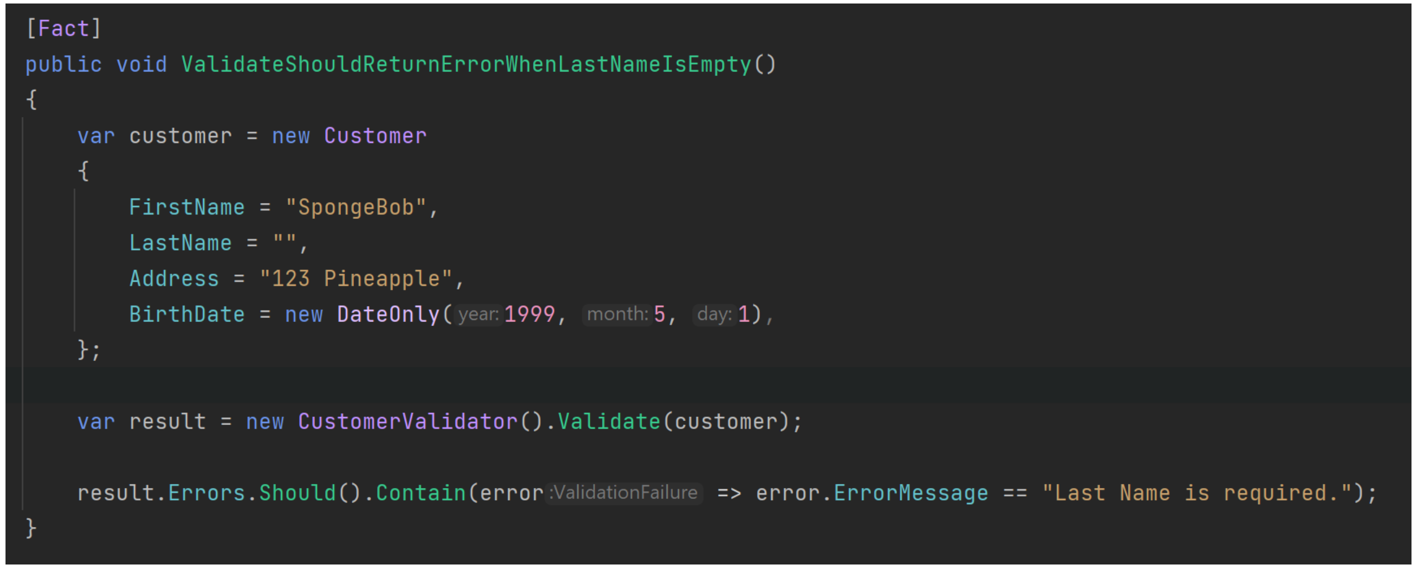
```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    _customer.LastName = "";

    var result = _customerValidator.Validate(_customer);

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

scottsauber

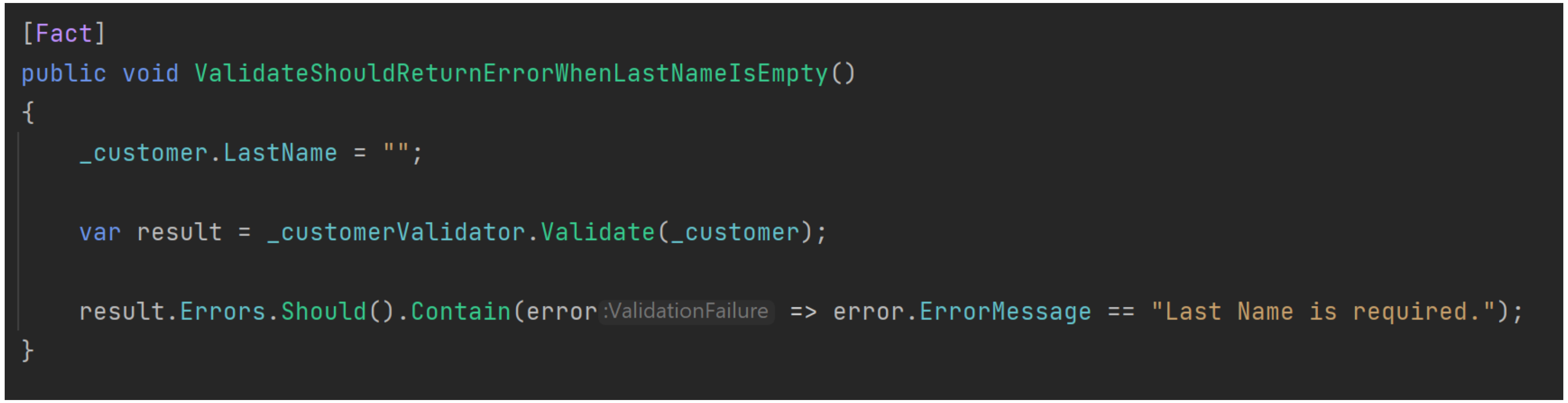
# Chekhov's Gun Applied to Testing



```
[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

## Back to TDD

## Forms of TDD

- “Bottom up” aka Detroit
- “Top Down” aka London aka Acceptance TDD

## Bottom up

- Build the bottom most layers first
- ⚠ Trigger warning ⚠
- Start with repository
- Then work up to Service/Handler
- Then work up to Controller
- In my experience, it’s easier to teach someone bottom up

## Top down

- Start with a failing Integration Test (Acceptance)
- Start with Controller
- Then Service/Handler
- Then Repository
- Very Mock heavy

## Neither style is “correct”

- Pick the style that makes sense for you
- I mix these – I start with a failing acceptance test then do bottom up
- Why? Bottom up makes more sense to me
- Sometimes I delete tests at the end if I don’t think they’re useful to keep around
- That’s ok - they helped me design

# But I don’t have time to write tests!

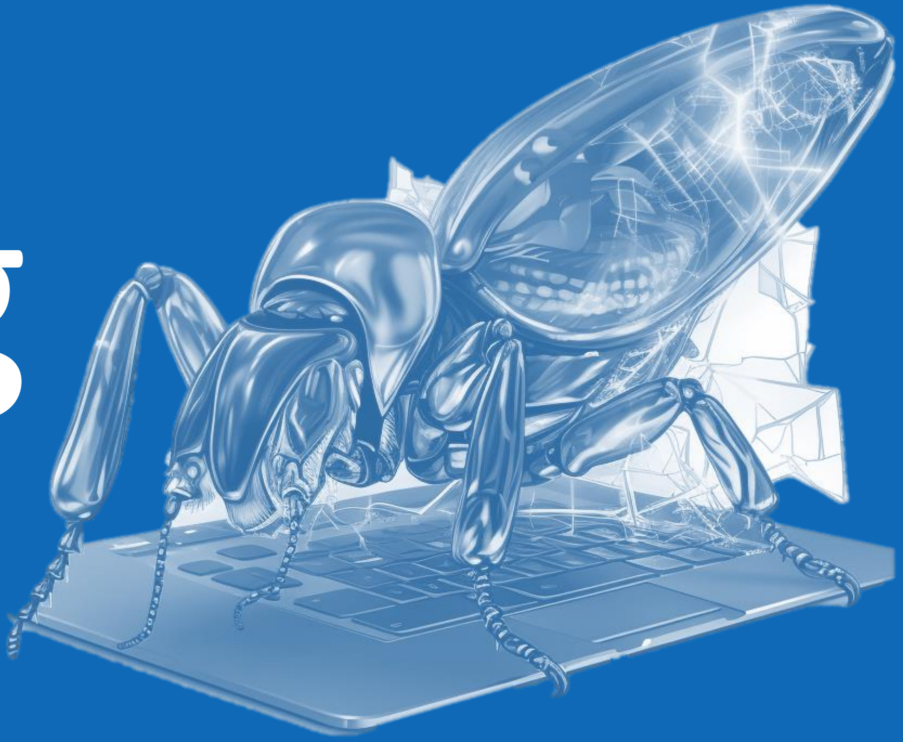
## Why?

My boss won’t let me!

What about this person?

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

The next bug you have is a great starting point



## Takeaways

- What kinds of tests are there
- Why you should TDD
- What to test
- Chekhov’s Gun
- Start doing TDD the next time you have a bug

## Resources

- Slides at scottsauber.com
- TDD for React: https://www.youtube.com/watch?v=oNi7DV7fJcU
- I’m giving a JetBrains Webinar on TDDing Blazor next month!
- Going to be TDDing an entire feature

# Questions?

Contact: ssauber@leantechniques.com

# Thanks!