Behavior Driven Development

Test First Development in .NET

OdeToCode.com

In This Module

- The evolution of TDD
- Using context
- Writing specifications

Behavior Driven Development

Why?

What's Wrong With Unit Tests?

- The focus on *testing*

What We Care About

Behavior
Experience
Stories
Desirements

What We Focus On

Test Suites
Units
Verification Tests
Implementations

A major difference is vocabulary. – Dave Astels

What is BDD?

- **It has the same mechanics as TDD**
  - We follow the rhythmic chant of red, green, refactor
  - We work in tiny increments
  - We ***design***
- **BDD focuses on behavior**
  - Don't focus on implementation details
  - Don't organize tests around abstractions (no *test fixture per class*)
- **BDD focuses on context**
  - What is the setting for this play?
- **BDD focuses on specifications**
  - Avoid the mindset of writing verification tests
- **BDD focuses on communication**
  - Change your vocabulary – communicate outside of development

What's A Context?

Contexts are natural and recognizable uses of a class or subsystem or web page, etc. – Scott Bellware

Contexts in Code

- **It's not a unit**
  - Unit tests often centered around a class
  - A class should have a single responsibility
  - However, many classes serve varying contexts
- **How many different contexts are in the following code?**

```
public class MovieController : Controller
{
    public ActionResult Index() { … }
    public ActionResult Details(int id) { … }
    public ActionResult Create() { … }
    …
}
```

Specifications

- **Specify the behavior you want to experience in a given context**
  - Stop thinking of tests
  - Think about what the software *should do*
- **Don't write specifications for developers**
  - Think about the expectations of a customer
  - Think about executable specifications or acceptance criteria

… the language you use shapes how you think … and if you want to change how you think it can help to first change your language. – Dave Astels

BDD

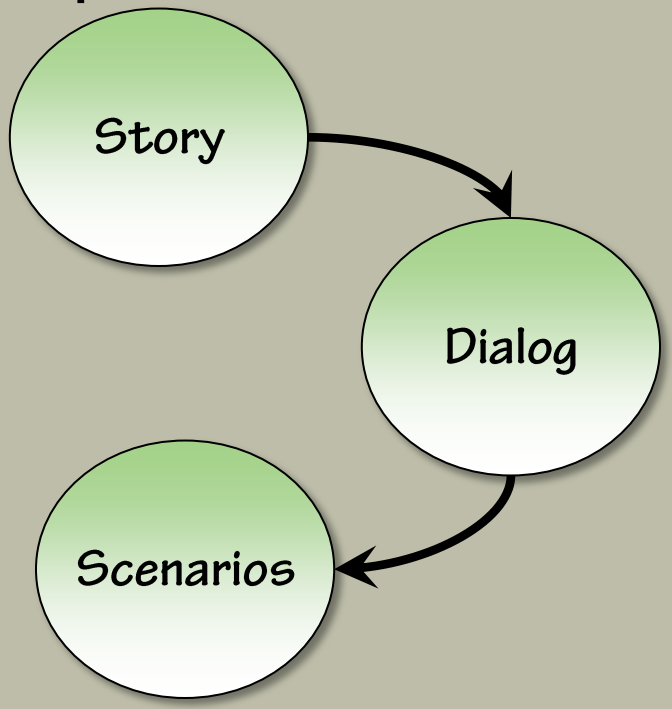
How?

Communication

- **Users and user stories give you raw building material**
  - Stories capture motivation, language, and expectations
- **BDD turns raw material into executable specifications**

As a member of the movie review's site
I want to review a movie
So I can influence the movie's rating

Given a new movie
When adding a new review with a rating of 1
Then the average movie rating should be 1
And the movie should have 1 total review

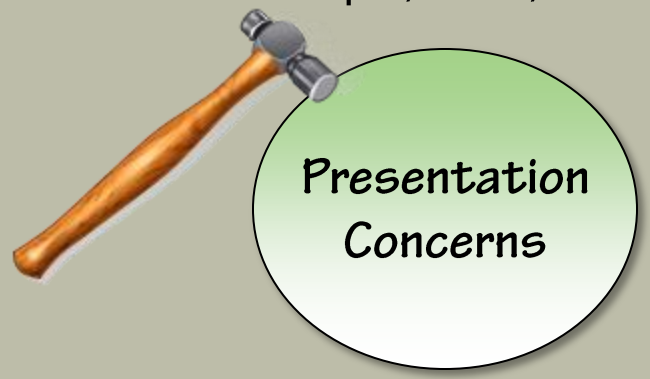


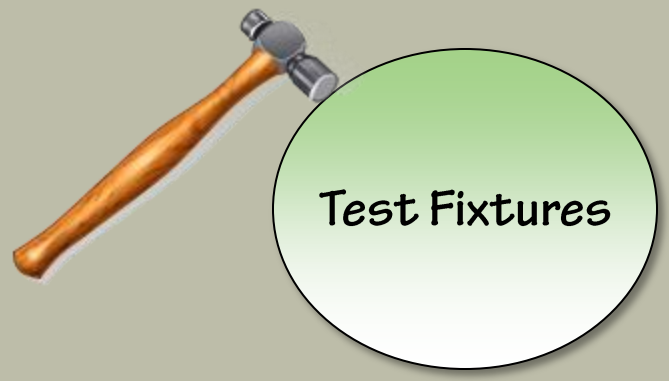
A Simple Start

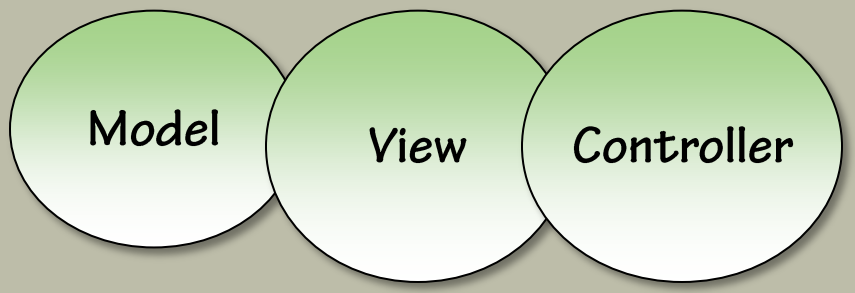
```
[TestFixture]
public class When_adding_first_review_to_movie {
    [SetUp]
    public void Setup() {
        _movie = new Movie();
        var review = new Review {Rating = 1};
        _movie.AddReview(review);
    }

    [Test]
    public void should_have_one_review() {
        Assert.That(_movie.TotalReviews == 1);
    }

    [Test]
    public void should_have_an_average_rating_of_one() {
        Assert.That(_movie.AverageRating == 1);
    }

    private Movie _movie;
}
```

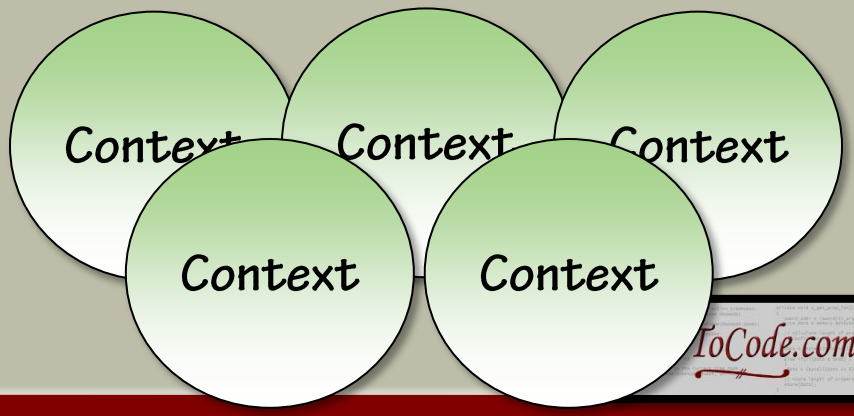
Observations

- **Organize test classes around a specific context**
  - Name the class by what it focuses on
  - Small and focused
- **Describe the experience using specifications**
  - Simple, short, and focused



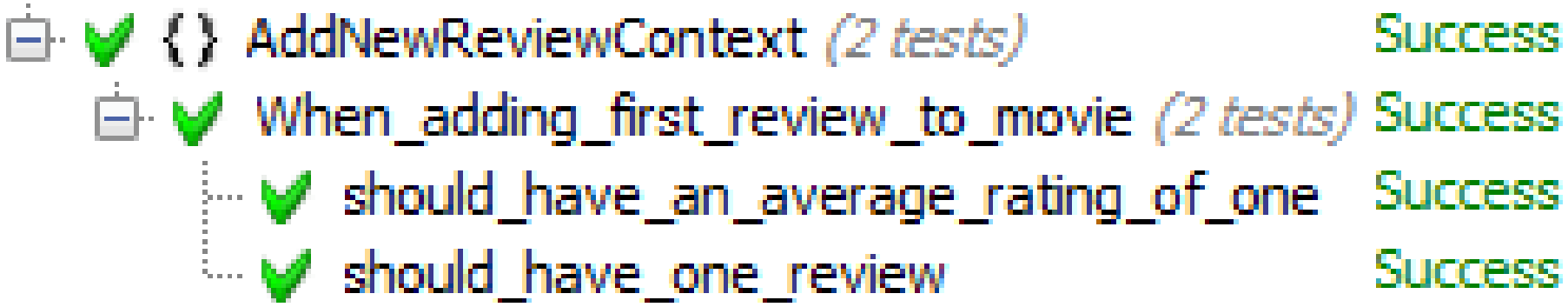






Language Observations

- **Who will find these results useful?**
  - Developers?
  - Testers?
  - Customers?



Next Steps

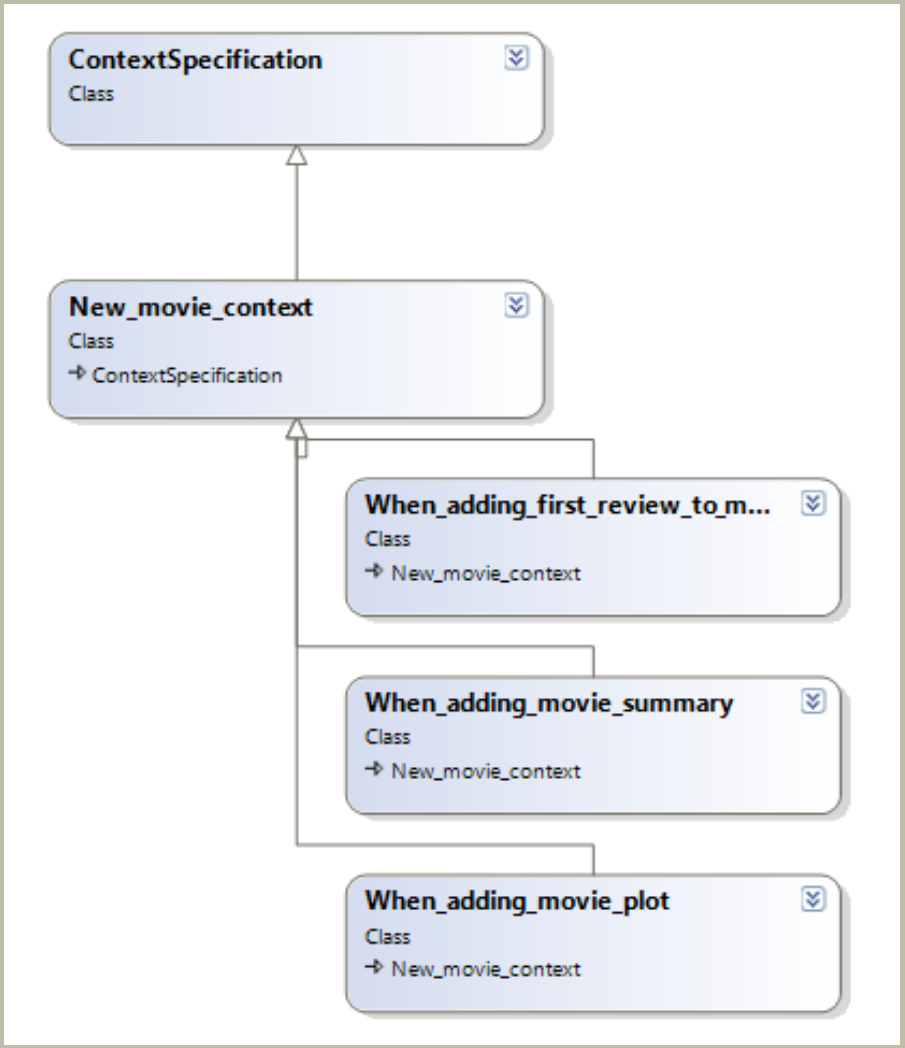
- Base classes for explicit context vocabulary

```
public class ContextSpecification
{
    [SetUp]
    public void Setup()
    {
        Context();
    }

    protected virtual void Context()
    {
    }
}
```

```
public class New_movie_context :
             ContextSpecification
{
    protected override void Context()
    {
        _movie = new Movie();
        ...
    }

    protected Movie _movie;
}
```

Sharing Context

- Keep tests DRY
- Can chain contexts
  - But favor readability



Moving Forward

- **Remember BDD demands a shift in vocabulary**
  - Therefore a shift in mindset
- **Unit testing frameworks use a testing vocabulary**
  - Perhaps there is something better?

MSpec

```
[Subject("Processing payroll")]
public class when_processing_timecard_with_overtime
{
    Establish context = () =>
    {
        _processor = new PayrollProcessor(new FakeLogger());
    };

    Because of = () =>
    {
        _processor.Process();
    };

    It should_handle_vacation_time = () => _vacation.ShouldEqual(2);
    It should_add_overtime_bonus = () => _amount.ShouldEqual(500);
```

- Processing payroll, when processing timecard with overtime *(2 tests)*
  - should add overtime bonus
  - should handle vacation time

SpecFlow

```
Feature: Payroll Processing
    In order to track overtime
    As a manager
    I want the system to calculate the overtime hours

Scenario: Overtime hours by department
    Given I an employee work 10 hours of overtime
    When I run payroll
    Then the result should show 10 hours of overtime
```

Conclusions

- **BDD does not change your mechanics**
  - Incremental design
  - Red, green, refactor
- **BDD changes your vocabulary**
  - Specification, not verification
- **BDD is about:**
  - Behavior
  - Context
  - Specifications
  - Communication to a wider audience