BDD and TDD

These notes came from some professional development training on behavior-driven design (BDD) and test-driven design (TDD). They focus more so on BDD, as TDD is more of a well-understood technique.

The premise for both techniques is that the code is secondary.

- **TDD**: write tests, then write code; more focused on the inner workings of the application
- **BDD**: write executable specifications, then write code; more focused on the outside behavior of the application

My takeaway is that TDD and BDD are techniques for making the specifications of how your software works more explicit by

1. starting with the desired outcome/behavior first rather than just affirming it at the end, and
2. having those specifications confirmed because of executable tests throughout the development process.

The primary benefit is what ends up being produced are more detailed acceptance tests, which improves communication among development, testers, and the customers.

It appears that the BDD area isn't as mainstream/active (at least in .NET) compared to unit testing in general. Many of the Stack Overflow posts and blogs/articles that come up in search results are several years old (2010 to 2013). The Pluralsight videos, however, are more a bit recent (2014 to 2016).

The learning curve for whatever style/tool we chose would be intermediate. I'm not seeing something that would provide a very obvious path forward without a good deal of discipline or having to work around tooling that doesn't help you "fall into the pit of success." I get more of a feeling of "yes, we could technically do this," but it probably wouldn't be easy for a desktop application project. The tooling seems to be a bit more available for web projects.

Another concern is that the examples I've seen are often fairly trivial, which makes the concepts seem approachable. Once you start getting into more meaty application requirements, I imagine it would be tricky to not get lost in the technical "weeds" of matching the what (behavior) with the how (code).

BDD is more in line with an integration test (as opposed to a unit test). Therefore, there are more layers of the software working together to express demonstrable behavior. It is possible to use BDD without having all layers involved. For example, you can specify behavior for a chess game without any UI at all: http://www.daedtech.com/chess-tdd-23-yak-shaving-with-specflow/

If time allows, it would be useful to design a small but non-trivial application from the ground up using this design approach and tooling. It's one thing to do a survey of the landscape (i.e., this document), and another thing to actually get your hands dirty and experience the process.

TDD/BDD as Architectural Tools

https://www.infoq.com/presentations/TDD-BDD-as-Architectural-Tools
By James Kovacs (11 Jan 2012)
1 hr 18 min

Notes

- How do we define an architecture?
  - UML is a common language to create blueprints, but inevitably things fall down
  - Once you start coding, you realize you missed something in the diagram
  - If your design is detailed enough to handle the entire system, you may have well just written the code
  - You end up with a "Big Up Front Design" -- not very Agile
  - "Executable specification" is the holy grail.
- With executing code, we can use the scientific method -- we can probe it. Does it perform well enough? Does it scale? How easy is it to add new things? Is it secure? How does it handle failure?
- We can't keep everything in our heads anymore. We need specifications to delineate our assumptions.
- **Test-driven development** uses the Red/Green/Refactor cycle
  - Red = failing test
  - Green = test now passes
  - Refactor = find opportunity for cleaner code now that everything passes
- **Ping-pong pairing**
  - One computer, two people; one driver, one navigator
  - Person A writes a failing test; Person B makes it pass; Person A can write another test or refactor what Person B just wrote. Repeat.
  - Uses TDD to design your system and have fun; lots of communication, making sure what you're doing is intelligible to the next person
  - Less likely to get interrupted because people see you are busy with someone else
  - More likely to stay focused (not checking e-mail, etc.)
  - Came from Peter Provost and Brad Wilson at the Patterns and Practices group at Microsoft.
- It's an unfortunate naming issue to have the last "D" of TDD/BDD be **development** because it's not about the coding, it's about the **design**.
- **Behavior-driven development**
  - Refinement of TDD; more context and explanation about how you should be doing TDD
  - Dan North is a pioneer in this area
  - Unit tests typically use the **Arrange/Act/Assert** pattern
    - Arrange an empty shopping cart, add an item, assert there is 1 item in the cart
    - Test files are usually associated with the code (e.g., Shopping Cart class); this gets messy quickly because there are lots of contexts in which a shopping cart can be used
  - BDD uses the **Given/When/Then** pattern
    - Given an empty shopping cart, when an item is added, then there should be 1 item in the cart
    - Test files are focused on contexts
- When you have to consider how you're using your class/API (which TDD/BDD brings more focus to in the beginning), you think a bit more about dependencies. You feel the pain of a tightly coupled system because it's hard to create tests.

- Examples of thinking with contexts…
    - Empty shopping cart
        - Add an item, there is 1 item now in the cart
        - Remove an item, an error message is shown
    - Cart with 1 item
        - Add the same item, still have 1 item, quantity is now 2
    - Cart with items
        - User tries to access with HTTP, is redirected to HTTPS
        - Cart is displayed in less than 3 seconds
- QA team can now focus on high-value things (e.g., user interactions, visual arrangements) instead of trying to break the software.
- The tests you write with BDD are essentially unit tests, which help you with regression.
- MSpec is a .NET tool to write BDD-oriented tests
    - Each context is its own class
    - Test runner outputs the specifications in plain English (based on naming conventions in the code)
    - Supports "empty" specifications to be filled in later
    - Comes with lots of examples of how to set up contexts
- You can bend the syntax of existing unit testing frameworks (MSTest, Nunit) to basically do BDD. What you get from tools like MSpec and SpecFlow is that they hide most of the plumbing so you can focus on the behavior, rather than .NET syntax
- BDD tools can integrate with continuous integration tools. For example, MSpec can create an HTML report that shows which specifications are implemented properly.
- Don't write all of your specs up front -- just write enough so you can get started
- Specifications define the architecture, the boundaries of your system, and a playground for your developers to work in. You can actually see how well things are doing -- code coverage, cyclomatic complexity, coupling.
- BDD is not a replacement for TDD; they are complimentary. BDD is about the outside of the system (big behavior), TDD is about the inside of the system.
- BDD is more useful for web controllers (view models?) or for front-end acceptance/integration testing. Focus on business requirements (BDD), rather than developer implementation (TDD). For example, a financial calculator implementation is done with TDD, and the calculator is used is done with BDD.

Tools

- If you want Web integration testing, use WatiN or Selenium
- .NET has MSpec (Machine Specification) and SpecFlow (there are more, but these are the more popular ones)
- Ruby has RSpec and Cucumber

Learning through Search

Things found when searching about BDD concepts, and about how we'd use it on a WPF app

- https://stackoverflow.com/questions/15706130/how-to-write-specflow-against-wpf
- https://stackoverflow.com/questions/10356005/is-bdd-really-applicable-at-the-ui-layer

  Mostly BDD is designed to help you have clear, unambiguous conversations with your stakeholders (or to help you discover the places where ambiguity still exists!) and carry the language into the code. The conversations are much more important than the tools.

  If you use the language that the business use when writing your steps, and keep them at a high level as Dan suggests, they should be far less brittle and more easily maintainable. These scenarios aren't really tests; they're examples of how you're going to use the system, which you can use in conversation, and which give you tests as a nice by-product. Having the conversations around the examples is more important than the automation, whichever level you do it at.
- https://softwareengineering.stackexchange.com/questions/327613/what-advantages-are-there-to-using-a-bdd-test-tool-like-specflow-over-simple-uni

  The term "BDD" was coined by some TDD practitioners to solve a specific problem: when teaching TDD, the most important hurdle to overcome, is to understand that TDD is not about testing.

  Dan North realized that one problem with understanding that TDD is not about testing, is that everything screams "TEST" at you: you are using a test framework to write test cases by inheriting from a class named TestCase and writing test methods whose name starts with test. You use testing terminology everywhere: "assertion", for example. So he thought: what if we simply remove all the words that are about testing and replace them with words related to behavior specification via examples? It's still exactly the same thing, except without the misleading words!

  And that's exactly what a BDD tool is: it's the same as a TDD tool, but without the testing terminology. That's how Dan North wrote JBehave, for example. If you already understand TDD, and understand that TDD is not about testing, then it doesn't matter what tool you use. But I, personally, find it easier to get into the mindset that I'm not writing tests, when I'm, well, not writing tests but rather examples. So, for me, BDD tools are easier to use, when I use TDD tools, I am constantly battling with myself "I know this is called a test, but it isn't one, it's an example, you stupid brain!"

  This is even more pronounced if you show your tests to someone else, e.g. a non-technical stakeholder. You could either explain to them that while this looks like a test, it isn't really a test but rather an example of the behavior … or you could use a framework that makes it look like an example of the behavior. The Great Dream™ that non-technical stakeholders would write executable specifications has not come to pass, but I know of several organizations where the non-technical stakeholders routinely read, and sometimes even edit Gherkin examples, even if they are not writing them from scratch. Obie Fernandez once wrote a web-based structural editor for Cucumber (which probably doesn't work anymore) that would let non-technical users edit Cucumber Scenarios in a structured manner, and immediately execute them against the codebase, and his non-technical customers actually

did use this editor to write new scenarios, and communicate their wishes in the form of executable examples.

tl;dr: TDD is not about testing, BDD is TDD without the testing terminology, BDD tools are testing tools without the testing terminology. If you can manage the mental divorce from testing while using testing terminology, it doesn't matter what you use. However, be aware that you may not be the only client of those "tests", even if you are the only one writing them, you may not be the only one reading them.

- Choosing a BDD framework for .Net
  - Specs aren't written by developers: choose between SpecFlow and NBehave.
  - Specs are written by developers: my personal recommendation NSpec. Runner up StoryQ.
- https://stackoverflow.com/questions/307895/what-is-the-most-mature-bdd-framework-for-net
  - In short, there are two flavors of BDD: xBehave and xSpec
    - xBehave (Given/When/Then) -- start with English, end up with code. Good high-level descriptions, can get tricky mapping this English into code
    - xSpec (Context/Specification) -- start with code, end up with English. More developer-oriented, but not as readable to business folks
  - SpecFlow (xBehave) seems to be the front-runner
  - MSpec (xSpec) seems like a good choice (to me) because of better documentation and Stack Overflow questions
  - This post shows examples of the Bowling Kata in both xBehave and xSpec styles. I found this helpful because it can give your team an idea of which style they'd prefer if choosing the BDD route.
- http://conductofcode.io/post/bdd-frameworks-for-dotnet-csharp/
  - Author tried out several popular BDD frameworks for .NET
  - Shows what the code/specs look like, and how they appear in the test runner

Related (Future) Training

When searching for BDD, Pluralsight had a good deal to offer.

Automating UI Tests for WPF Applications

- https://app.pluralsight.com/library/courses/wpf-applications-automating-ui-tests/
- September 2015
- 2h 56m
- Automating UI tests for WPF applications is not an easy task, especially if you have a large project with reach functionality. Nevertheless, it is possible to create a set of simple and maintainable UI tests which can become a great supplement for your development process. In this course, we will walk through the process of creating a full-fledged automated UI test suite. Along the way, we will use simple yet telling WPF applications with functionality close to that of real-world enterprise applications.

Pragmatic Behavior-driven Design with .NET

- https://app.pluralsight.com/library/courses/pragmatic-bdd-dotnet/
- March 2014
- 2h 52m
- Behavior-driven Design (BDD) is a simple way to structure your tests and development practice, but over the years it's been convoluted by jargon and cargo-cult testing frameworks. In this course, Rob Conery creates a membership library for ASP.NET MVC and brings you along in an "over-the-shoulder," pair-coding style. Along the way concepts will be discussed at length, and at the end you'll appreciate how elegant, simple, and helpful BDD can be.

Business Readable Automated Tests with SpecFlow 2.0

- https://app.pluralsight.com/library/courses/specflow-2-0-business-readable-automated-tests
- 2016 March
- 3h 33m
- There's nothing more frustrating to a developer than building an application only to find out that although the system has no major technical bugs, it's not actually what the customer or business wanted. In this course, Business Readable Automated Tests with SpecFlow 2.0, you'll learn how you can use SpecFlow to allow developers and test automation specialists to produce tests in natural language instead of code. First, you'll learn how to write features and scenarios, then move into Visual Studio. You'll also learn about step definitions, automation code, and data conversion. Finally, you'll discover how to take control of test execution and run any additional code you might need. When this course is finished, you'll be able to create automated tests in SpecFlow that are understandable to any businesses you work with in the future.

More Expressive Testing in .NET with MSpec

- https://app.pluralsight.com/library/courses/expressive-testing-dotnet-mspec
- 2015 August
- 2h 31m
- This course will introduce you to a different way to think about testing your .NET code. Typical unit tests tend to be overly focused on the implementation details of the code instead of specifying the intent of the software. In this course, you will learn how to write specifications for

your code using the Machine.Specifications (MSpec) framework. These specifications can more clearly describe what your software should do instead of how it does it, leading to a more useful and maintainable suite of tests for your system.

SpecFlow Tips and Tricks

- https://app.pluralsight.com/library/courses/specflow-tips-tricks/
- 2014 February
- 1h 28m
- Whether you have recently discovered SpecFlow or have been using it for a while, the tips and tricks in this course will help you create more maintainable SpecFlow test automation solutions.

Syntax Examples

For a more complete set of examples from different tools, see http://conductofcode.io/post/bdd-frameworks-for-dotnet-csharp/.

Mspec

```
[Subject("Making a customer preferred")]
public class when_a_regular_customer_is_made_preferred
{
    Establish context = () =>
    {
        _order = new Order(new[] { new OrderItem(12),
                                   new OrderItem(16) });
        _totalAmountWithoutDiscount = _order.TotalAmount;

        SUT = new Customer(new[] { _order });
    };

    Because of = () =>
        SUT.MakePreferred();

    It should_mark_the_customer_as_preferred = () =>
        SUT.IsPreferred.ShouldBeTrue();

    It should_apply_a_ten_percent_discount_to_all_outstanding_orders = () =>
        _order.TotalAmount.ShouldEqual(_totalAmountWithoutDiscount * 0.9);

    private static Customer SUT;

    private static Order _order;
    private static Double _totalAmountWithoutDiscount;
}

[Subject("Making a customer preferred")]
public class when_a_preferred_customer_is_made_preferred
{
    Establish context = () =>
    {
        _order = new Order(new[] { new OrderItem(12),
                                   new OrderItem(16) });
        _totalAmountWithoutDiscount = _order.TotalAmount;

        SUT = new Customer(new[] { _order });
        SUT.MakePreferred();
    };

    Because of = () =>
        SUT.MakePreferred();

    It should_apply_no_additional_discount_to_the_outstanding_orders = () =>
        _order.TotalAmount.ShouldNotEqual(_totalAmountWithoutDiscount * 0.81);

    private static Customer SUT;

    private static Order _order;
    private static Double _totalAmountWithoutDiscount;
}
```

This yields the following output from the test runner:

Making a customer preferred, when a regular customer is made preferred

» should mark the customer as preferred

» should apply a ten percent discount to all outstanding orders

Making a customer preferred, when a preferred customer is made preferred

» should apply no additional discount to the outstanding orders

Source: https://elegantcode.com/2010/02/19/getting-started-with-machine-specifications-mspec/

SpecFlow

```
Feature: Score Calculation
  As a player
  I want the system to calculate my total score
  So that I know my performance

Scenario: Gutter game
  Given a new bowling game
  When all of my balls are landing in the gutter
  Then my total score should be 0

Scenario: Beginners game
  Given a new bowling game
  When I roll 2 and 7
  And I roll 3 and 4
  And I roll 8 times 1 and 1
  Then my total score should be 32

Scenario: Another beginners game
  Given a new bowling game
  When I roll the following series:       2,7,3,4,1,1,5,1,1,1,1,1,1,1,1,1,1,5,1
  Then my total score should be 40

Scenario: All Strikes
  Given a new bowling game
  When all of my rolls are strikes
  Then my total score should be 300
```

Source: https://github.com/techtalk/SpecFlow-Examples/blob/master/BowlingKata/BowlingKata-MsTest/Bowling.SpecFlow/ScoreCalculation.feature