

Sunday, September 23, 2007

Behaviour-Driven Development Explored

Since I last posted I've talked/exchanged emails to a few people interested in the concept of [Behavior-Driven Development](#), so this entry is a slightly less rambling introduction to BDD with some PHP oriented examples later. Any questions can be directed to the comments, or the usual email address.

Behaviour-Driven Development (BDD) is an evolution of Test-Driven Development (TDD). Now, as Noel Darlow pointed out in the earlier comments, any Unit Testing framework can do BDD. This is the biggest barrier to understanding BDD - since it's not completely incorrect (BDD is an evolution, not something sparkly new). Dave Astels (author on two "Practical Guide" books for TDD and XP), boils it down to a simple phrase:

"Behavior-driven development is what you are doing already, if you are doing Test-driven development well."

The phrase is an oversimplification but let's work from there.

The problem? Most people who practice TDD are not doing it well (this is a generally well known problem). Whoa, most people don't do it anyway full stop. A challenge to BDD anywhere is asserting the opposite is true - though honestly I can't see to many people arguing the point.

A core mistake that practitioners, and newcomers, make is entertaining the belief that TDD is all about testing. This is untrue! TDD is not, will not, and never has been, a testing methodology. Tests are a nice, happy side effect - but not the goal of TDD. Test-Driven Development is actually a design methodology - we perform it to improve quality of design (among other benefits).

Yet the main complaints about TDD, are related to...testing. Tests are boring; tests take too much time; I can't test complex code; PHP5 won't let me test private members, yada yada yada. So not only are most people not doing TDD well, far more never adopt TDD because they simply do not understand it from the current TDD literature - which keeps yammering on about "test-first" and other misleading catchphrases. And so the penetration of TDD has remained relatively limited.

Unfortunately, thinking in terms of testing is difficult to avoid. TDD is bombarded with the word. Every book, every article, every forum post, its very name - and every Unit Testing library. Test, test, test, test...oh, and test.

The first rule of BDD? Stop using the word "test" - it's screwing with people's heads!

We do not want to test! What we actually want to be doing, is specifying!

But behold, "test" has allies in this nefarious plot to confuse the mind - like "unit". What is a Unit? Most folk would define a unit as an independent class, method or property isolated from the rest of the system. In TDD, which utilises Unit Testing, we usually end up testing on a per unit basis. Now in a testing approach this is perfect. In a design approach it misses the point.

The second rule of BDD? Stop using the word "unit" - it's the minion of "test"!

Instead, let's pick a word with all sorts of immediate impacts: Behaviour. We do not design code by reference to units alone, we design (and refactor) by reference to the behaviour we wish our code to exhibit. Small, tiny, focused pieces of behaviour. Unit != Behaviour (you see they're spelled different). This is a foundation stone in eXtreme Programming already where prior to commencing coding, we may assemble a collection of User

Stories collected from the client and other stakeholders (like intended users) which our code must implement in order to be useful. Each story is not about units! It's about detailing required behaviour in terms a client (without a programming background) can easily understand and critique.

Next up is "assertion". It too belies a core devoted to oxymoronic forces (to design by testing doesn't make sense to many people). An assertion, in terms of testing, is a tool to verify. Verify what? Code that exists. So another insinuating hint to test - not design.

Third rule? Exile "assertion" to the wilderness. We might as well consign any thoughts of "testing state" (all those who consider testing private properties are wrong anyway) to the same place. In place of assertions, we shall have expectations - expectations of how our selected piece of behaviour will exhibit itself when we get around to writing code.

Now for many people, this will get the immediate response "BDD is just a language change". Yes, it is - but not in isolation because a change in language often precipitates a change in thinking, and the scope of such thought. So let's not get too hung up on the language differences alone. It's how BDD gets you thinking differently that's interesting. Consider the following quick descriptive exercise with the updated terminology:

"In BDD, we describe the behaviour of a system by writing executable specifications."

This, self-summarised, comment captures a goal of BDD in language a lot of experienced TDD users would feel comfortable with (because it's obviously a core facet of TDD completely buried under the test-centric nomenclature). Consider the impact on someone new to TDD - they are now informed about writing concrete examples of desired behaviour. That's a hell of a lot more useful than talking in terms of tests.

But let's not stop here! We have to address one possibly show ending issue - if TDD as described and evolved through BDD makes sense, then how do we escape the fallacy of testing, and move towards the bright light of specifying behaviour? The BDD community of users and advocates (which is growing from its humble beginnings) has answered this with perhaps the most inflammatory of responses:

Stop using Unit Testing frameworks. One prays Marcus and Sebastian let me survive that 

Language is the big barrier. Though one can perform BDD in a testing-centric UT library, it requires an amount of skill to translate between the test oriented language and organisation of Unit Testing to the purely behaviour driven language required by BDD. It's like viewing something in Dutch, and needing to constantly translate it into English (unless you are a native speaker already). It's difficult. So difficult, that TDD is itself by relation difficult to fully grasp and perform well. TDD has had long term exposure - perhaps it's time to do the inevitable: Evolve. Now what?

Use Behaviour Specification frameworks.

Don't be afraid. Remember we're evolving, not expanding the root taxonomy of eXtreme Programming.

A BDD Framework is generally designed from the ground up to orient itself completely to specifying behaviour. As an evolution, it shares undoubtable ancestry with Unit Testing but also borrows from Acceptance Testing. The BDD Framework I'm most familiar with is called [RSpec](#), provided for the Ruby community (variants for Smalltalk, .NET and Java exist). I've been discussing a viable PHP API since my call for PHP BDD tools with others, and we've cooked up a few possibles. Here's one example inspired by RSpec:

```
require_once 'Bowling.php';
```

```

class describeBowlingResultsAfterGutterGame extends Behaviour {
    public function before() {
        $this->_bowling = $this->getSpec('Bowling');
    }
    public function itShouldHaveAScoreOfZero() {
        $this->_bowling->hitGutter();
        $this->_bowling->score->should()->equal(); // by the heavens, it's plain English!
    }
    public function after() {
        unset($this->_bowling);
    }
}

```

Now before we get into a "you can do this in PHPUnit/SimpleTest" declaration of defiance, let's repeat an obvious statement from before. BDD is an evolution of TDD. Okay? So obviously, a BDD framework will appear similar to a TDD framework. But note the differences - remember the language barrier TDD users are forced to deal with?

Now what's going on here? To specify behaviour, that behaviour must be given a context. In Bowling, one could fling that ball into the gutter (because it's a dodgy alley, and we suspect the lane is none too level) or hit a strike. Both are contexts, each giving rise to different behaviour (i.e. the expected score) within the system.

Wow, our normal English explanation looks...familiar. Hey, it's the exact same format as the BDD class! Bingo. The class encourages several guiding conventions, quite literally robbed from RSpec, including class prefixing with "describe" and method prefixing with "itShould". These get your thoughts on track and focused on what matters - describing a behaviour by specifying what it should do. In fact it can be represented in plain English by stringing all the class/method names in what is often called a "spec dox" output:

Bowling results after gutter game
 - should have a score of zero

Here, I trail off and end the entry. With my own thoughts a little more focused, and some more PHP developers hopefully inspired to look deeper at BDD.

I have omitted a few things though - BDD is often looked as an evolution of TDD borrowing additional qualities from Domain-Driven Design and Acceptance Testing Driven Development (I wrote an article on [Acceptance Testing for Zend Devzone](#) during the Summer if looking for a PHP oriented flavour). Not enough room in a blog post to explore everything! One useful aspect, is something DDD calls the "ubiquitous language". The language of behaviours and specs, being an accessible readable format, is not only a programming language - it's equally accessible to everyone from your DBA to your client. Now the utility of this varies - a client may not be so intested in how Bowling works in isolation. But we're seeing something more accessible emerge. Maybe another time I'll discuss another BDD Ruby tool called rbehave (Java has JBehave) which pushes further horizons in the direction of ATDD.

Posted by Pádraic Brady in PHP General, PHP Security at 03:12

Savvy, although I am somewhat skeptical of the feasibility of implementing the example you provide (yes, chained method calls are nice, but PHP's not flexible enough to let us do that with all data-types).


The relevant URL for the Acceptance Testing article is:

<http://devzone.zend.com/article/2242-Acceptance-Testing-of-Web-Applications-with-PHP> Anonymous on Sep 23 2007, 04:30

Hi

I think I'm much more interested than I might have sounded in my earlier post. Although I'm not sure if a whole new term (BDD) is justified I do agree that language is very important. I'll be watching with interest.

Noel Anonymous on Sep 29 2007, 01:11

@Noel: No problem 

. You can find us at PHPSpec.

@Edward: The classy thing is that we're not implementing yet so it's just an implementation detail being ignored until added to an iteration. One possible solution is generating a proxy wrapper around the class so the interface is operating on the proxy, not the underlying object directly. I think that's what you're referring to at least. Anonymous on Sep 29 2007, 01:33