Behaviour Driven Development

A new Look at Test Driven Development

BDD – Motivation and core ideas

TDD

BDD – Motivation and core ideas

BULLSHIT !!!

Testing

A typical TDD Learning Process

1. Start writing unit tests around code
2. Enjoy a strongly increased sense of confidence
3. Insight that writing the tests before writing the code, helps to focus on writing only the code that is needed.
4. Notice that tests serve to document how the code works.

True benefits of TDD

5. Realise that writing tests in this way helps to 'discover' the API to the code

True benefits of TDD

6. Realise that TDD is about defining **Behaviour** rather than testing

BDD – Motivation and core ideas

Evolution

BDD – Motivation and core ideas

Language

Units

BDD – Motivation and core ideas

Behaviour

BDD – Motivation and core ideas

State

BDD – Motivation and core ideas

Interaction

Assertion

BDD – Motivation and core ideas

Expectation

Verification vs. Specification

'testing is all about making sure that your code functions correctly (verifying by stating assertions) ...

... while specifying is all about defining what it means for your code to function correctly (stating Expectations)'

(Robert C. Martin)

BDD – Motivation and core ideas

Specification Frameworks

Specification of a Stack

Frank: What's a stack?

Linda: It's a container that collects objects in a first in, last out manner.

It **should** provide the possibility to push and pop objects.

Sometimes you'll want to peek the last added element, as well ...

Frank: What does *push* do?

Linda: *push* takes an input object, say *foo*, and places it onto the stack.

push **should** return the successfully pushed object.

The stack **should** contain that object afterwards.

Specification of a Stack

Frank: What if I *push* two things, like *foo* and then *bar* ?

Linda: The second object, *bar*, should be on top of the conceptual stack (containing at least two objects),

so that if you call *pop*, *bar* should come off instead of the first object, which, in your case, is *foo*.

If you called *pop* again, then *foo* should be returned

and the stack should be empty
(assuming there wasn't anything in it before you added the two objects).

Specification of a Stack

Frank: So *pop* removes the most recent item placed into the stack?

Linda: Yes, *pop* should remove the top item (assuming there are items to remove).

peek follows the same rule, but the object isn't removed.
peek should leave the top item on the stack.

Specification of a Stack

Frank: What if I call *pop* without having pushed anything?

Linda: *pop* should throw an exception indicating that nothing has been pushed yet.

Frank: What if I push() null?

Linda: The stack should throw an exception because null isn't a valid value to push().

Describing Expectations using 'should'

Equality	should(be (..)
Negation	should(not (*predicate*(..)))
Types	should(be (ofType(*class*)))
Counts	should(have (atLeast(3, *class*)))) should(have (atMost(5, *class*))))
P.Match.	should(match(regexp))
...	

Summary

We need to start thinking in terms of

behavior specifications
(instead of verification tests)

Current TDD vocabulary shapes a mindset that takes us in a wrong direction

TDD should focus on design
TDD should focus on behaviour
TDD should focus on documentation

A new name for this new way of working:

Behaviour Driven Development