

# Behavior Driven Development

The Tao of Testable Requirements

*... in 5 minutes*

Aslam Khan  
Technical Director  
PBT Group



# Me

- Day Time
  - [aslamk@pbt.co.za](mailto:aslamk@pbt.co.za)
  - Stuck in Tshwane until July 2008
- Off Time
  - [www.aslamkhan.net](http://www.aslamkhan.net)
  - [aslamkhn@gmail.com](mailto:aslamkhn@gmail.com)



# Thanks

- Dan North
  - <http://dannorth.net/>
  - <http://dannorth.net/introducing-bdd>
  - <http://dannorth.net/whats-in-a-story>



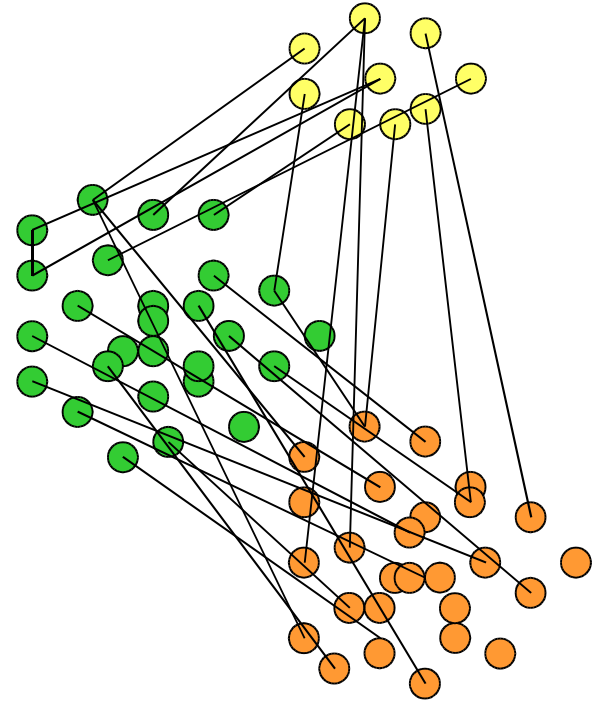
# Disclaimers

- About me
  - Not the world's authority on BDD
- About BDD
  - It works for me
- Code examples shamelessly stolen from Dan's blog



# What's the problem?

- Join the dots between
  - requirements
  - the test code
  - the production code



- Now change a dot or line ... ☹️



# Tell a story

**Story:** Transfer to cash account

**As a** savings account holder

**I want** to transfer money from my savings account to cash account

**So that** I can get cash easily from an ATM



# Tell some more of the story ...

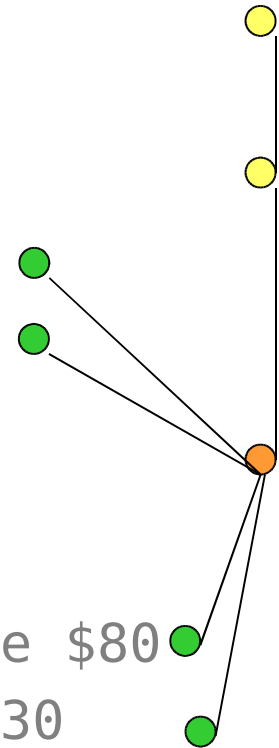
**Story:** Transfer to cash account

**Scenario:** savings account is in credit

**Given** my savings account balance is \$100  
and my cash account balance is \$10

**When** I transfer \$20

**Then** my savings account balance should be \$80  
and my cash account balance should be \$30



# More of the story ...

**Story:** Transfer to cash account

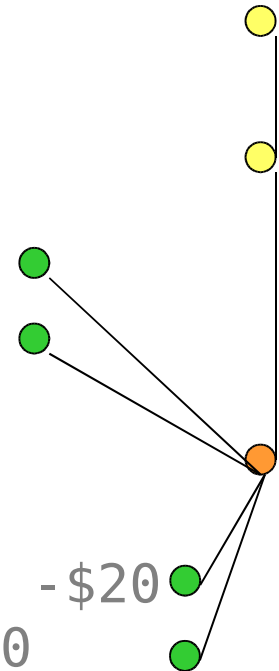
**Scenario:** savings account is overdrawn

**Given** my savings account balance is -\$20  
and my cash account balance is \$10

**When** I transfer \$20

**Then** my savings account balance should be -\$20  
And my cash account balance should be \$10

Now change a dot or line ... 😊



# Write the code...

```
require 'rubygems'
require 'rbehave'
require 'spec' # for "should" method
require 'account' # the actual application code

Story "transfer to cash account",
%(As a savings account holder
  I want to transfer money from my savings account
  So that I can get cash easily from an ATM) do

  Scenario "savings account is in credit" do
    Given "my savings account balance is", 100 do |balance|
      @savings_account = Account.new(balance)
    end
    Given "my cash account balance is", 10 do |balance|
      @cash_account = Account.new(balance)
    end
    When "I transfer", 20 do |amount|
      @savings_account.transfer_to(@cash_account, amount)
    end
    Then "my savings account balance should be", 80 do |expected_amount|
      @savings_account.balance.should == expected_amount
    end
    Then "my cash account balance should be", 30 do |expected_amount|
      @cash_account.balance.should == expected_amount
    end
  end
end
```



# Run the test ...

```
> ruby transfer_funds.rb
```

```
Running 2 scenarios:
```

```
.F  
2 scenarios: 1 succeeded, 1 failed, 0 pending
```

```
FAILURES:
```

```
7) transfer to cash account (savings account is overdrawn) FAILED  
Spec::Expectations::ExpectationNotMetError: expected -20, got  
-40 (using ==)
```

```
...
```

Each . is a test that passed  
Each F is a test that failed  
Each P is a pending test



# Do the round trip ...

```
/>ruby transfer_funds.rb --dry-run --format simple
```

```
Story: transfer to cash account
As a savings account holder
  I want to transfer money from my savings account
  So that I can get cash easily from an ATM
```

```
Scenario: savings account is in credit
  Given my savings account balance is 100
  Given my cash account balance is 10
  When I transfer 20
  Then my savings account balance should be 80
  Then my cash account balance should be 30
```

```
Scenario: savings account is overdrawn
  Given my savings account balance is -20
  Given my cash account balance is 10
  When I transfer 20
  Then my savings account balance should be -20
  Then my cash account balance should be 10
```



# Supporting Acts

- Ruby
  - rspec
  - rbehave
  - Install as a gem
- Java
  - JBehave (<http://jbehave.org/>)
  - Download the jar ☹
- .NET
  - NSpec (<http://nspec.tigris.org/>)
  - NBehave (<http://nbehave.org/>)
  - Ditto ☹



# Back of the mind ...

- Everyone understands “done”
- Avoid gumption traps
  - “That’s not what I asked for”
  - “I forgot to tell you about this other thing”
- Ubiquitous language
- Outside-in style development
  - Send “tell, don’t ask” messages between objects
  - `dog.getBody().getTail().wag()` ☹️
  - `dog.expressHappiness()` 😊



# Where do I use it?

- BDD for
  - Discovering requirements
  - Documenting requirements
  - Release planning
  - Documenting acceptance criteria
  - Writing tests
  - Writing code
  - “done” tracking
  - Focused training for end users



# Wrap up...

- Tell the story
  - As a [role/person]
  - I want [some feature]
  - So that [some benefit]
- Tell the scenarios
  - Given [some context]
  - and [some more context]
  - When [some event]
  - Then [some outcome]
  - and [another outcome]
- Write the test
- Write the code
- Run the test



Q & A

