Probative Programming

Toward the physical unification of code and tests

David A. Black
Director
Ruby Power and Light, LLC
http://www.rubypal.com
dblack@rubypal.com

Emerging Technologies for the Enterprise
March 26-27, 2009

About me

- Programming in Ruby since 2000
- Ruby/Rails consultant and trainer
  - Ruby Power and Light, LLC (rubypal.com)
- A director of Ruby Central
- Author of
  - *Ruby for Rails* (2006)
  - *The Well-Grounded Rubyist* (imminent!)
  - *Rails Choices* (working title; 2009/2010)

But enough about me....

Broad outline

- The problem
- The consequences of the problem
- Ideas for dealing with the problem
- Demonstrations:
  - Literate Programming
  - Cucumber
  - Probative Programming (using Cucumber)

I.

The problem:

Testing is optional

Testing

- A vital practice
  - with a profound technical problem

It is *possible* not to test;
therefore, people don't.

Compare with...

- installing software tools (you have to)
- running interpreter/compiler (you have to)
- setting up permissions for your team (you have to)
- etc. (you have to...)

Black's Law

As long as testing is optional, there will be untested code.

Black's Other Law

Technical problems respond to technical solutions, not to peer pressure.

II.

The consequences

The consequences

- Technical
  - untested code
  - ummm, more untested code
- Social
  - pressure
  - cajoling
  - bragging rights
  - unpleasantness

Show us your test$_e$s!

Where are your tests?

You do have testes,

don't you?

Very tiresome.

P.S. It doesn't work.

τεκνος, not ηθική*

- The problem is technical, not ethical
- No non-technical solution is adequate
  - not cajoling
  - not bragging
  - not "No pain, no gain" mantras

* *Pace* people who actually know Greek

Whatever happened to...

Programming should be

Fun!

Two out of three IS (are?) bad...

Only two of the following can be true:

- Programming is fun
- Testing is not fun
- Programming should include lots of testing

III.

The solution (I think, maybe): Probative Programming

probe

proof

prüfen

pro'-ba-tive

probare

1. serving or designed for test or trial
2. affording proof or evidence

(Random House Dictionary)

probable

prove

The solution to untested code

- Change the flow of energy
- No more muscle flexing
- No more displays of grit
- Let the gravity of the process do its work

The "what"

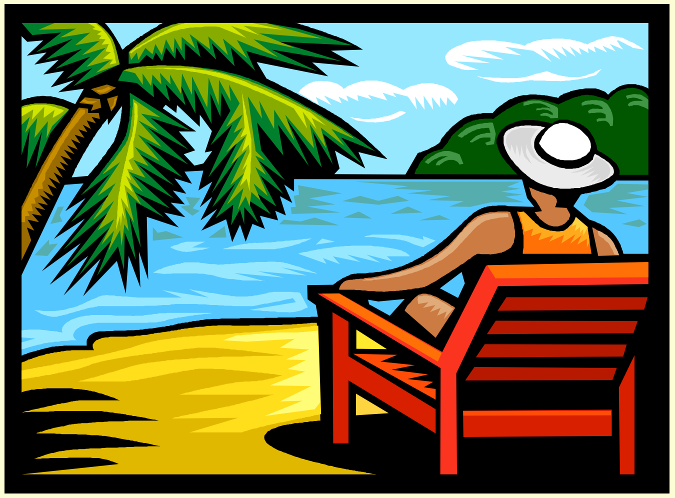
- A file containing code and tests
  - not executable
- A processor/tester
  - applied to file
  - runs tests
  - *iff successful, generates code files*

The *what?!*

- Code does not exist until tests pass
- *Not* running tests no longer an option
- Creates a technical system for ensuring test coverage
  - – taking it out of the realm of willpower

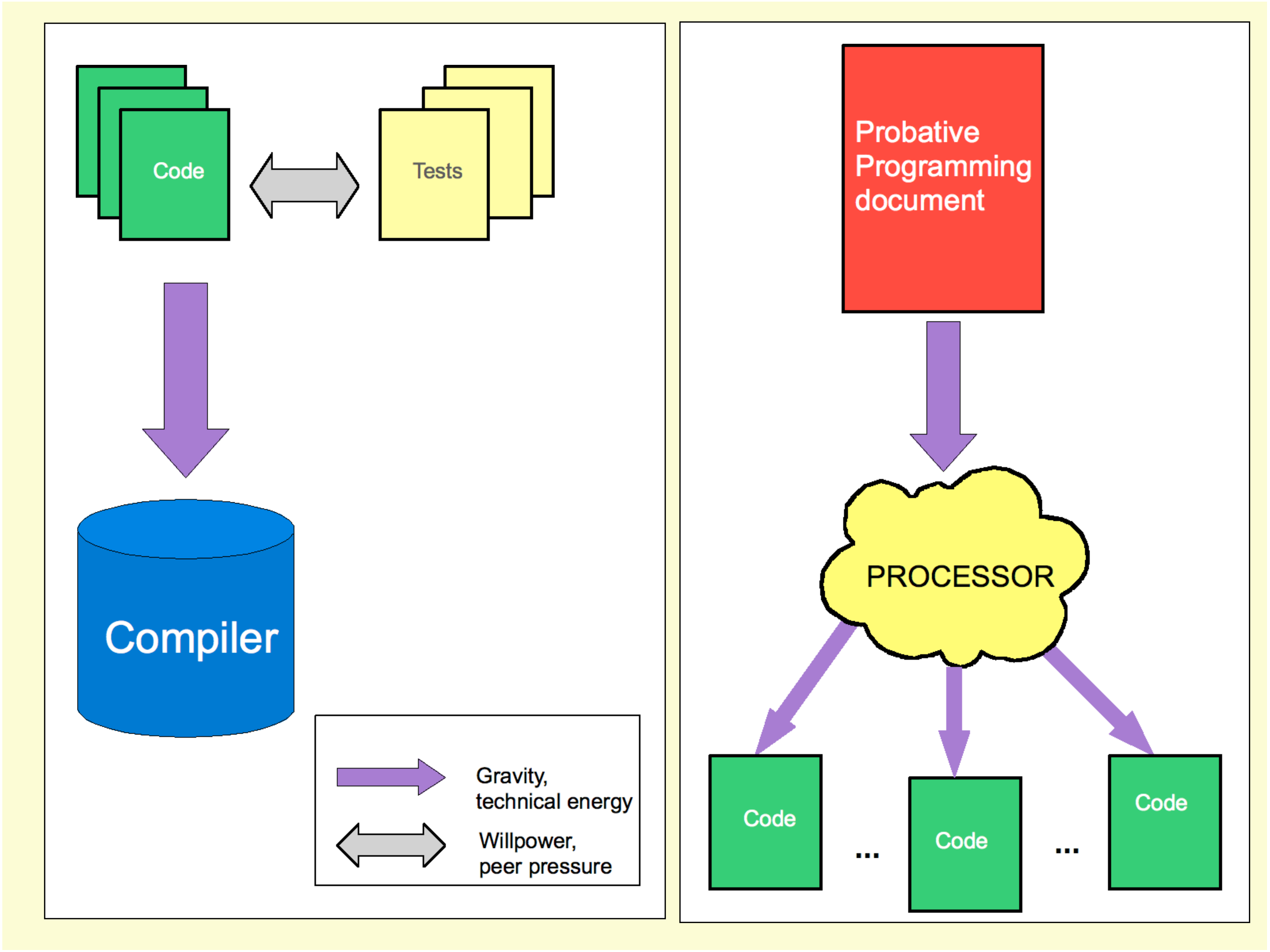
Realign the axes of energy....

Before...



...After!

Or, a little more technically...


Code
Tests
Compiler
Gravity, technical energy
Willpower, peer pressure
Probative Programming document
PROCESSOR
Code
...
Code
...
Code

Inspiration: Literate Programming

- Knuth, 1984
- Goal: combine a programming language and a formatting language
  - typically ((C or Pascal) and TeX)
- Code and documentation in one file
  - not code, not documentation
  - contains the potential for both

Literate Programming...

...does not mean just writing lots of comments

...does not mean just having documentation strings

...is not the same as executable specs

(it's not a spec/test technology)

...quite radically changes what it means to engage in the act of writing a program

Literate programming

- Example

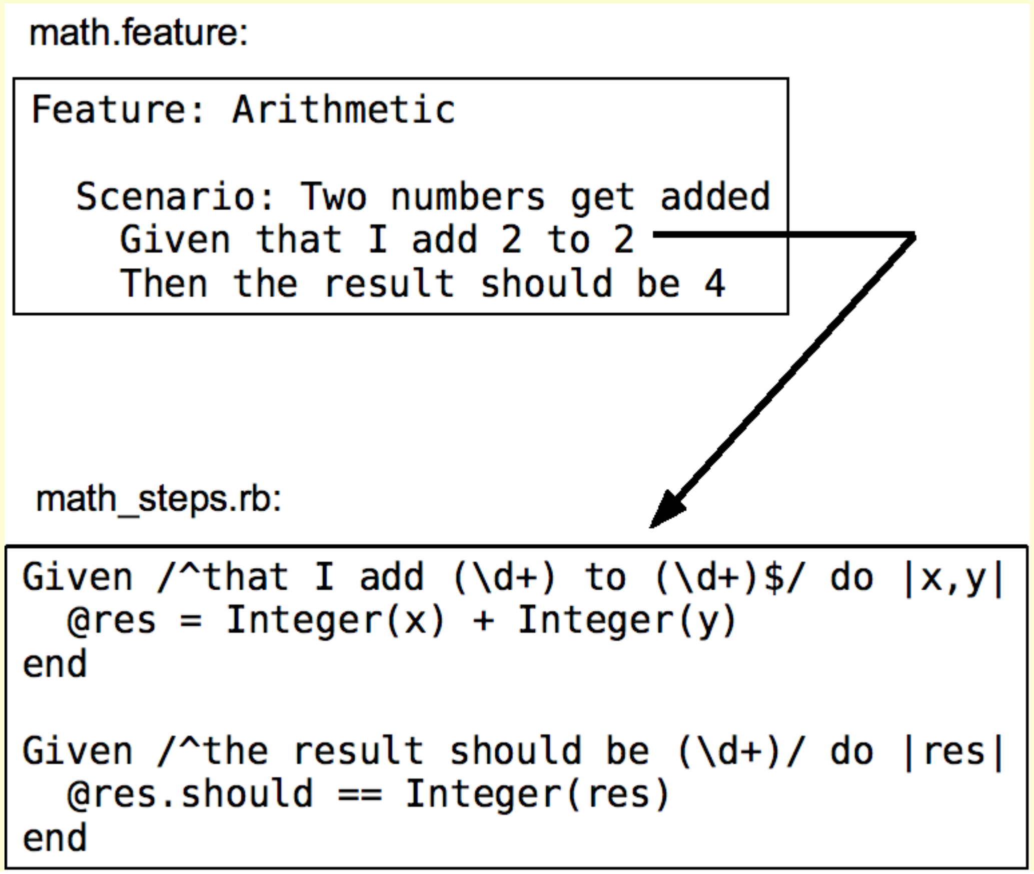
Goal of Probative Programming

- Similar to literate programming *(in my dreams....)*
  - shift the center of gravity
  - instead of code wrapped in tests, both together
  - possibly not even view the code files

The how

- Example, using Cucumber

This is how cucumber works.....



math.feature:

```
Feature: Arithmetic

  Scenario: Two numbers get added
    Given that I add 2 to 2
    Then the result should be 4
```

math_steps.rb:

```
Given /^that I add (\d+) to (\d+)$/ do |x,y|
  @res = Integer(x) + Integer(y)
end

Given /^the result should be (\d+)/ do |res|
  @res.should == Integer(res)
end
```

Wrap-up

- The problem is technical
- The solution has to be technical
- It's not about strength of character