Test Driven Design meets Design by Contract

-- Experiments in Specification --

Jim Weirich

RubyGems

- RubyGems

- Rake

- RubyGems

- Rake

- Ruse

Test Driven Development

Red (write test)

Green (fix code)

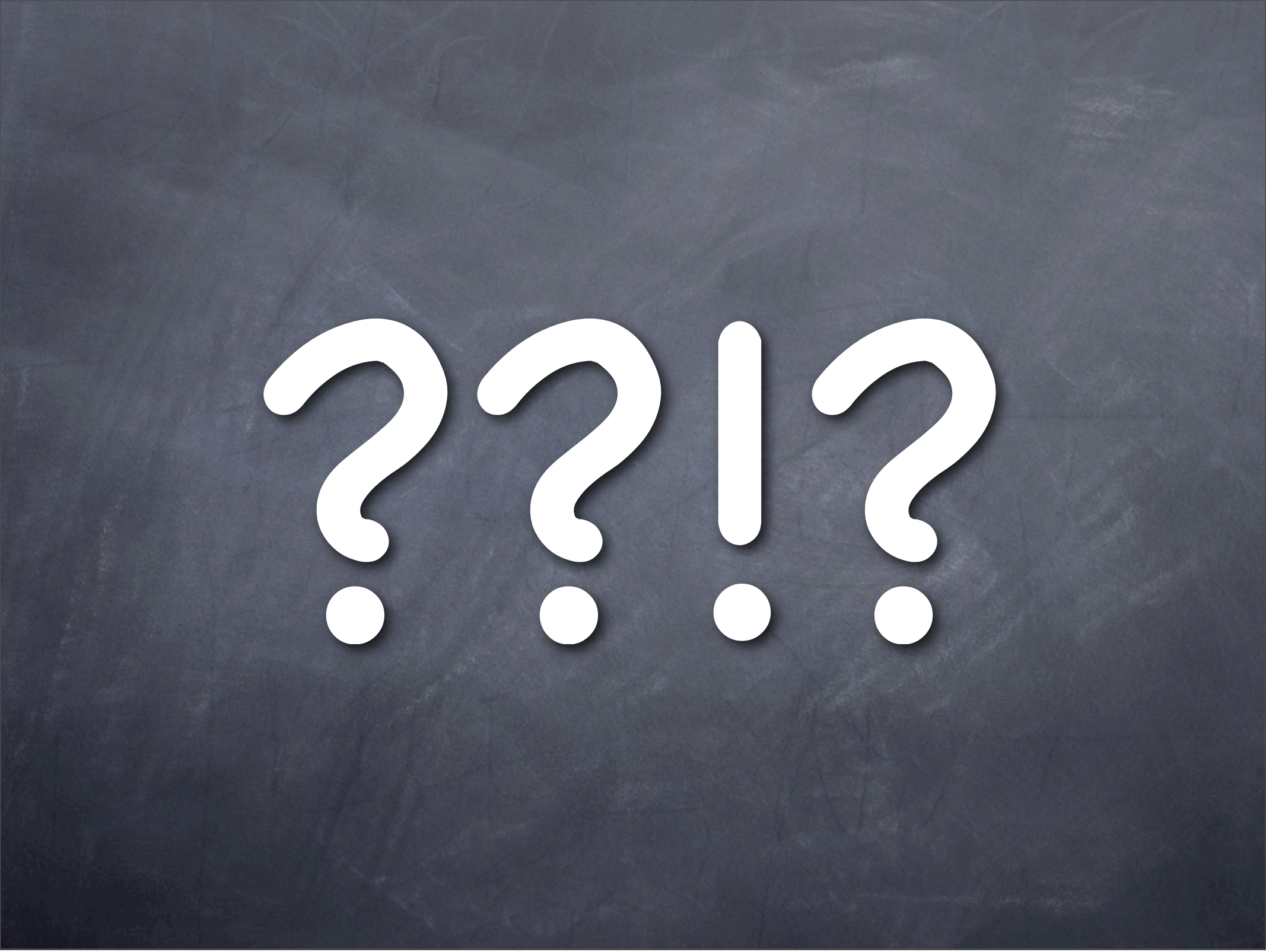
Refactor (clean up)

What's the Fuss?

Dave Astels

(Author of
"Test Driven Development:
A Practical Guide")

"This is not a book about Testing ..."



Design

Test Cases

Unit Tests

Terminology

assert_equal
assert_match

is

Test Suites

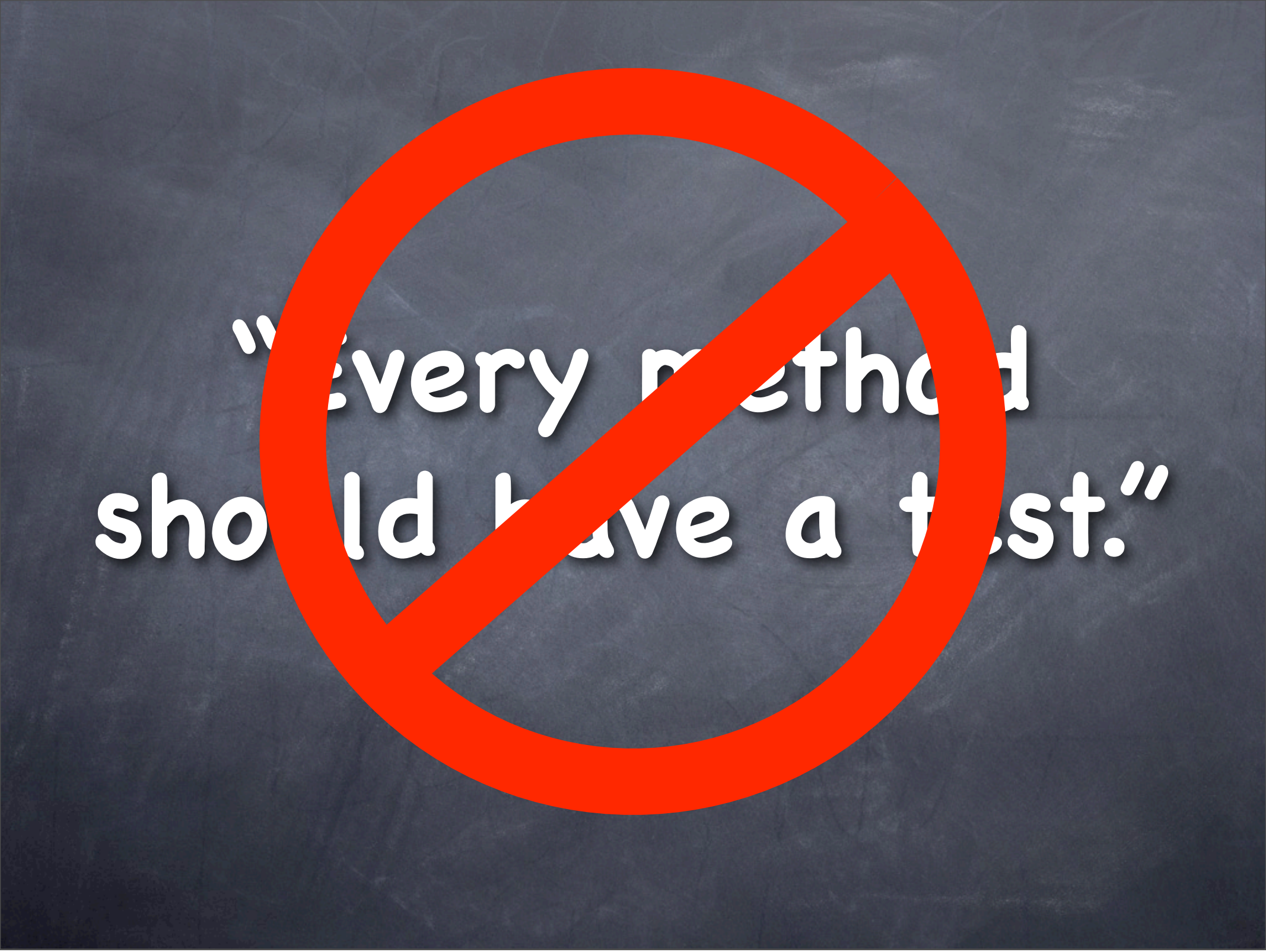
Inappropriate

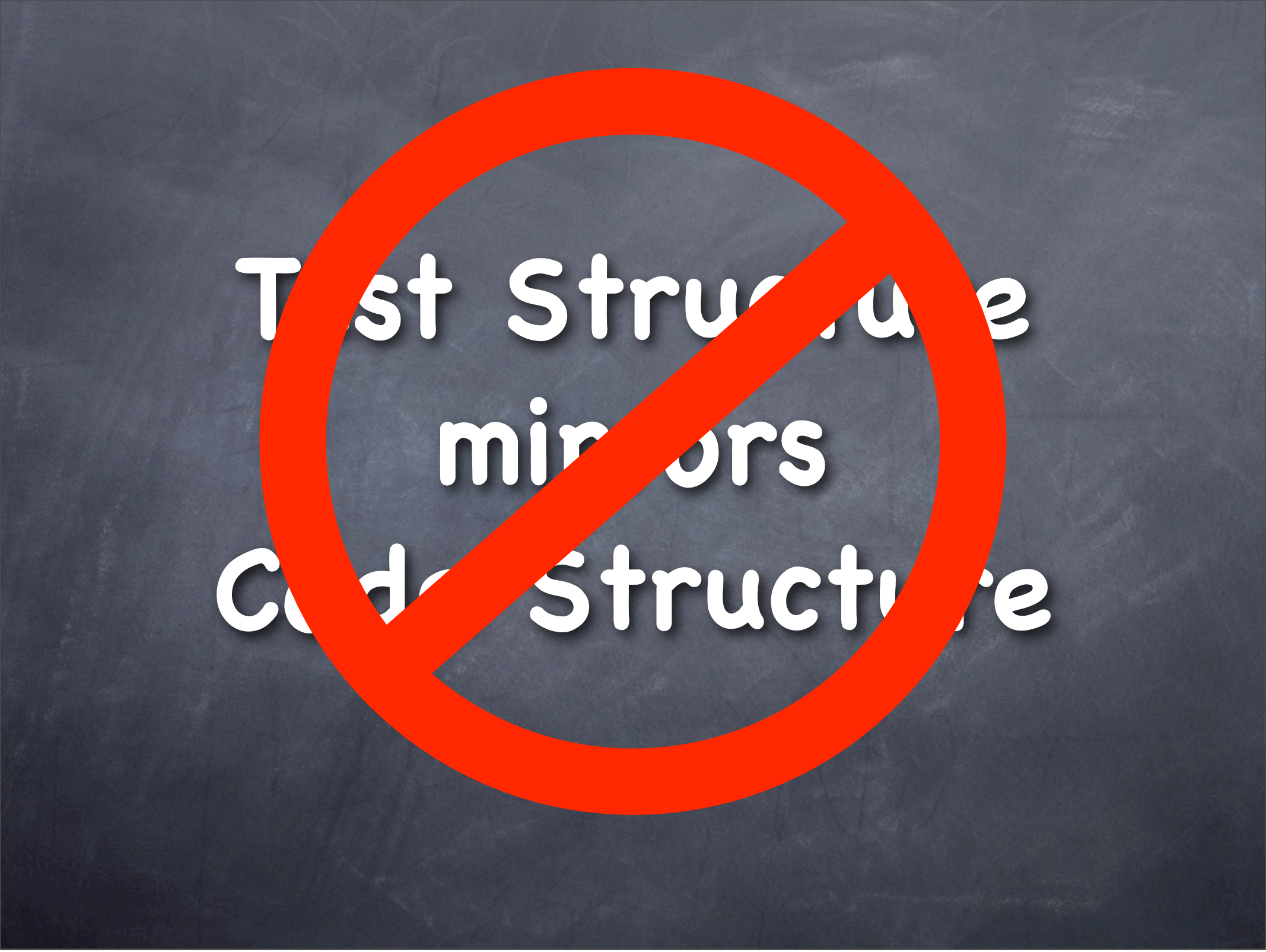
assert_nil
assert_not_nil

Test Fixtures


Verification

Bad Habits


"Every method
should have a test."


Test Structure
mirrors
Code Structure


Over Emphasis
of
State

Test Driven Design

Behavior Driven Design

Tests

Specifications

Assertions

Expectations

Executable Specifications

(rSpec)

```
class TestBowling < Test::Unit::TestCase
  def setup
    @scorer = BowlingScorer.new
  end
  def test_initial_score
    assert_equal 0, @scorer.score
  end
end
```

```
class TestBowling < Test::Unit::TestCase
  def setup
    @scorer = BowlingScorer.new
  end
  def test_initial_score
    assert_equal 0, @scorer.score
  end
end
```

```
context "A New Bowling Scorer" do
  setup do
    @scorer = BowlingScorer.new
  end
  specify "should have a zero score" do
    @scorer.score.should.be 0
  end
end
```

"A New Bowling Scorer"

"should have a zero score"

- A New Bowling Scorer
  - should have a zero score

- A New Bowling Scorer
  - should have a zero score
- A Scorer after one frame
  - should have the number of pins
- A Scorer after a spare
  - should add the pins
  - should the frame to the following
- A Scorer after a strike
  - ...

away from

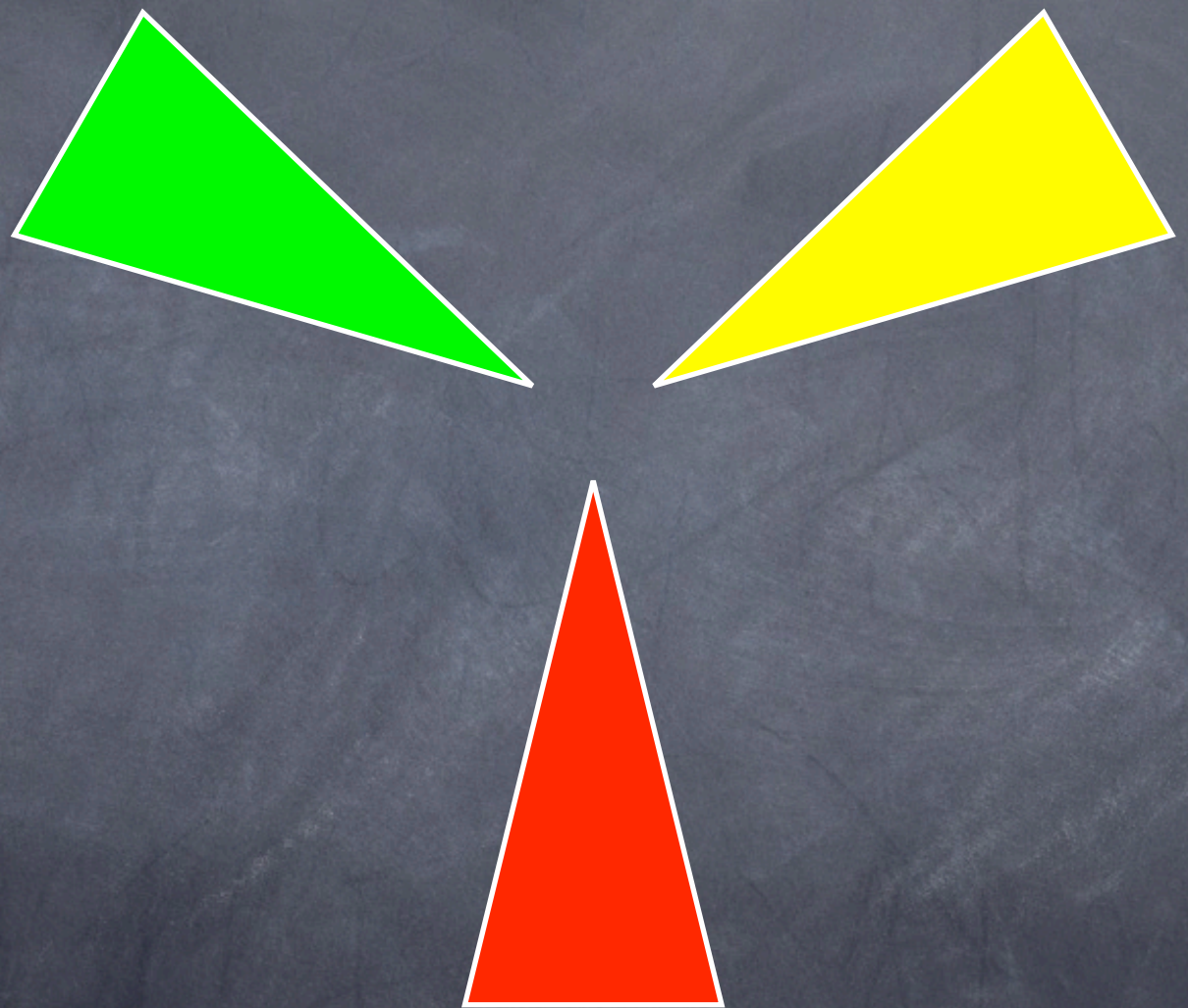
Verification

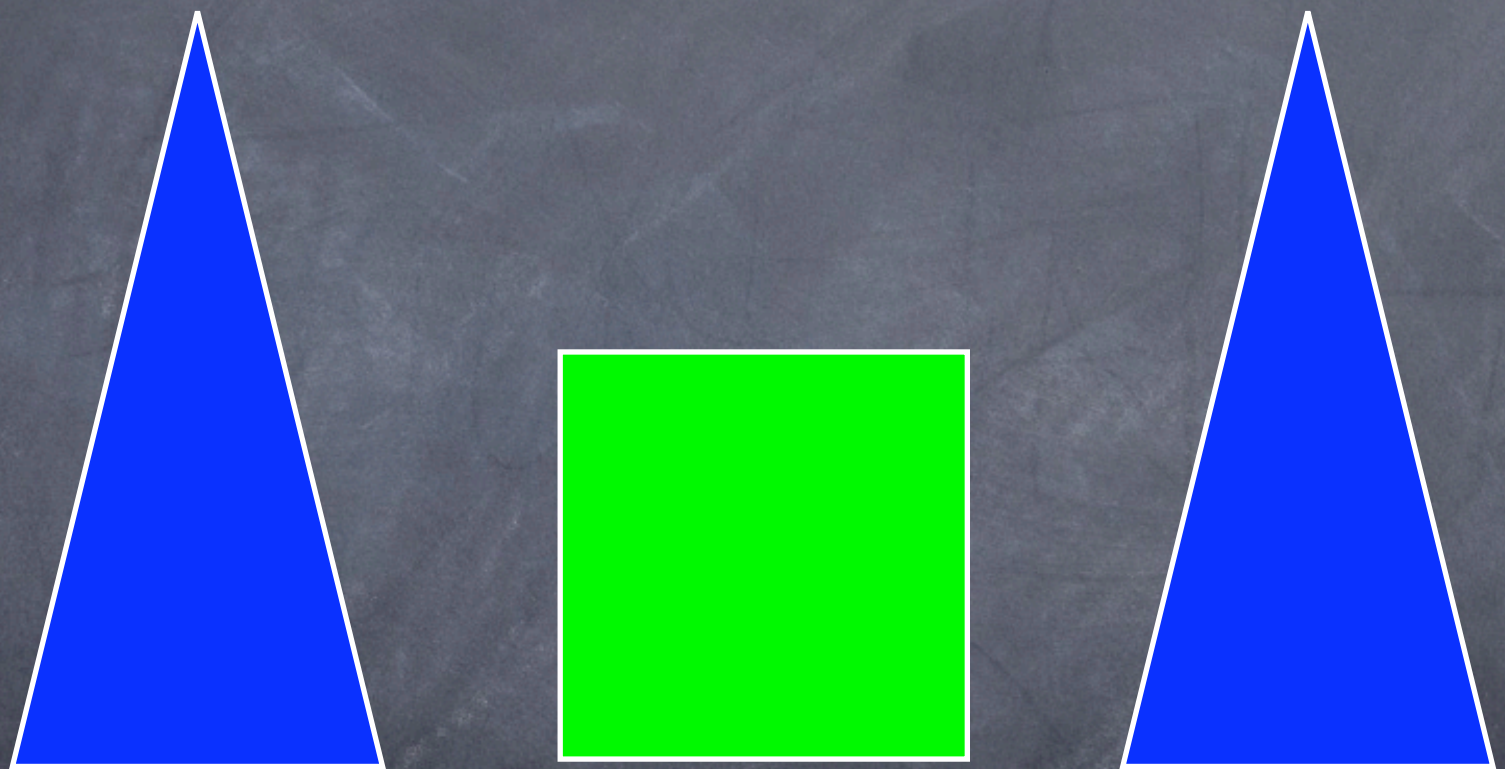
toward Design and Specification

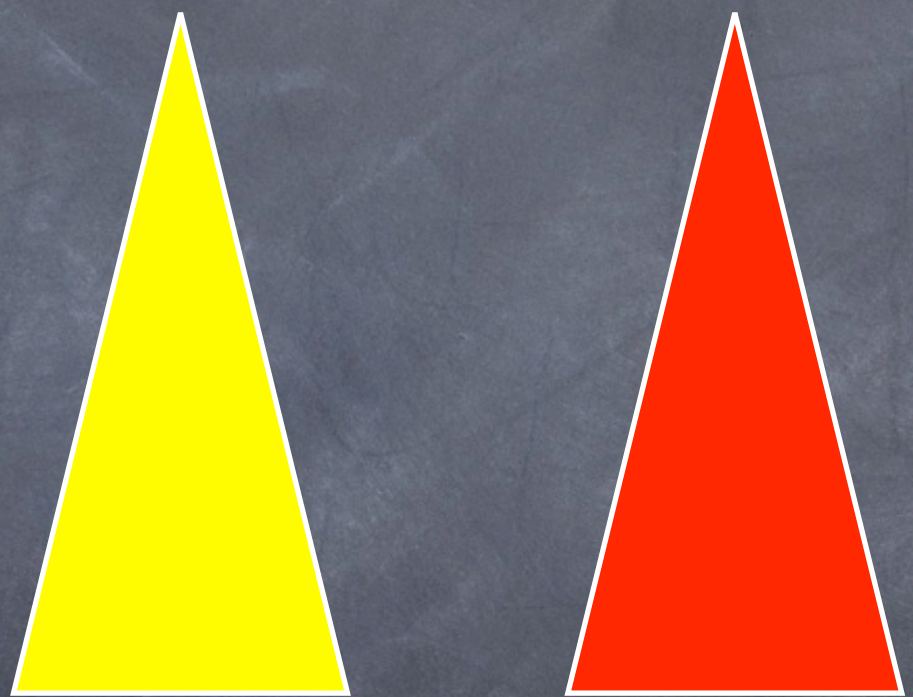
Questions?

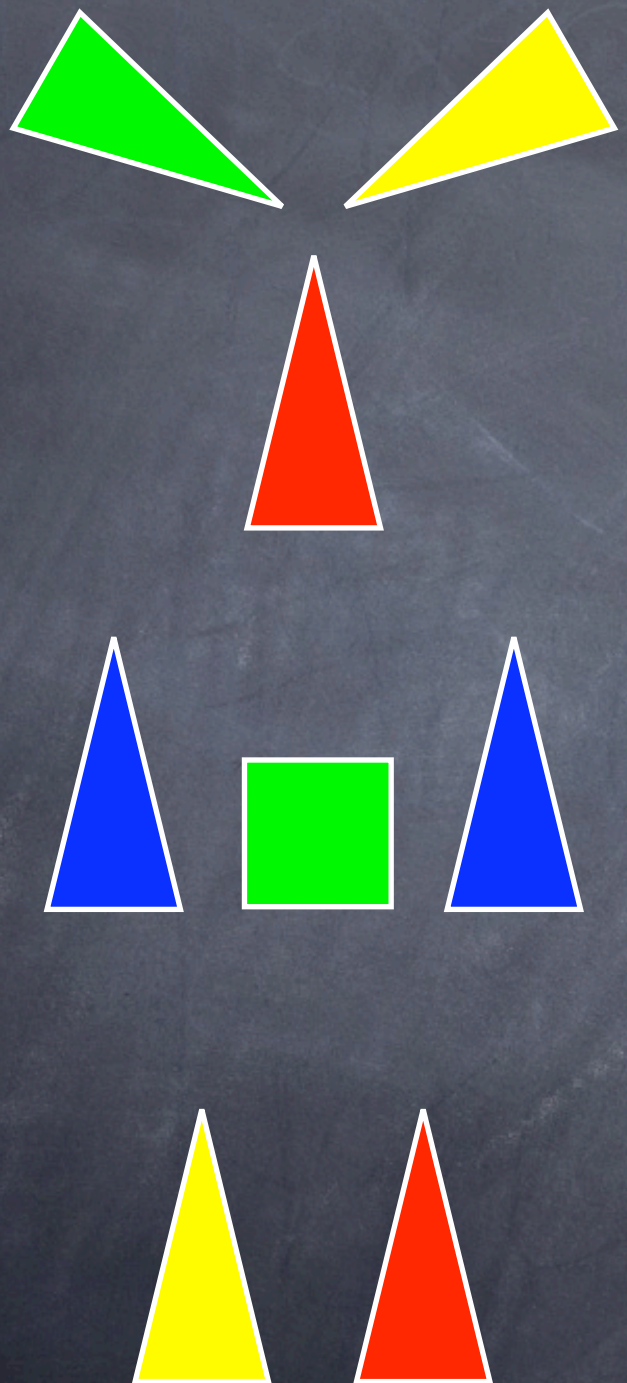
Specification

by Example









What is the rule?

At least two triangles?

or maybe:

Even number of upright triangles

1 3 5 7 9 11 ...

Odd Numbers

```
! n.even?
```

2 3 5 7 11 ...

Prime Numbers

```
n.factors.size == 1
```

3 10 5 16 8 ...

Wondrous Numbers

```
(n.even?) ? n/2 : 3*n+1
```

3 10 5 16 8 4 2 1

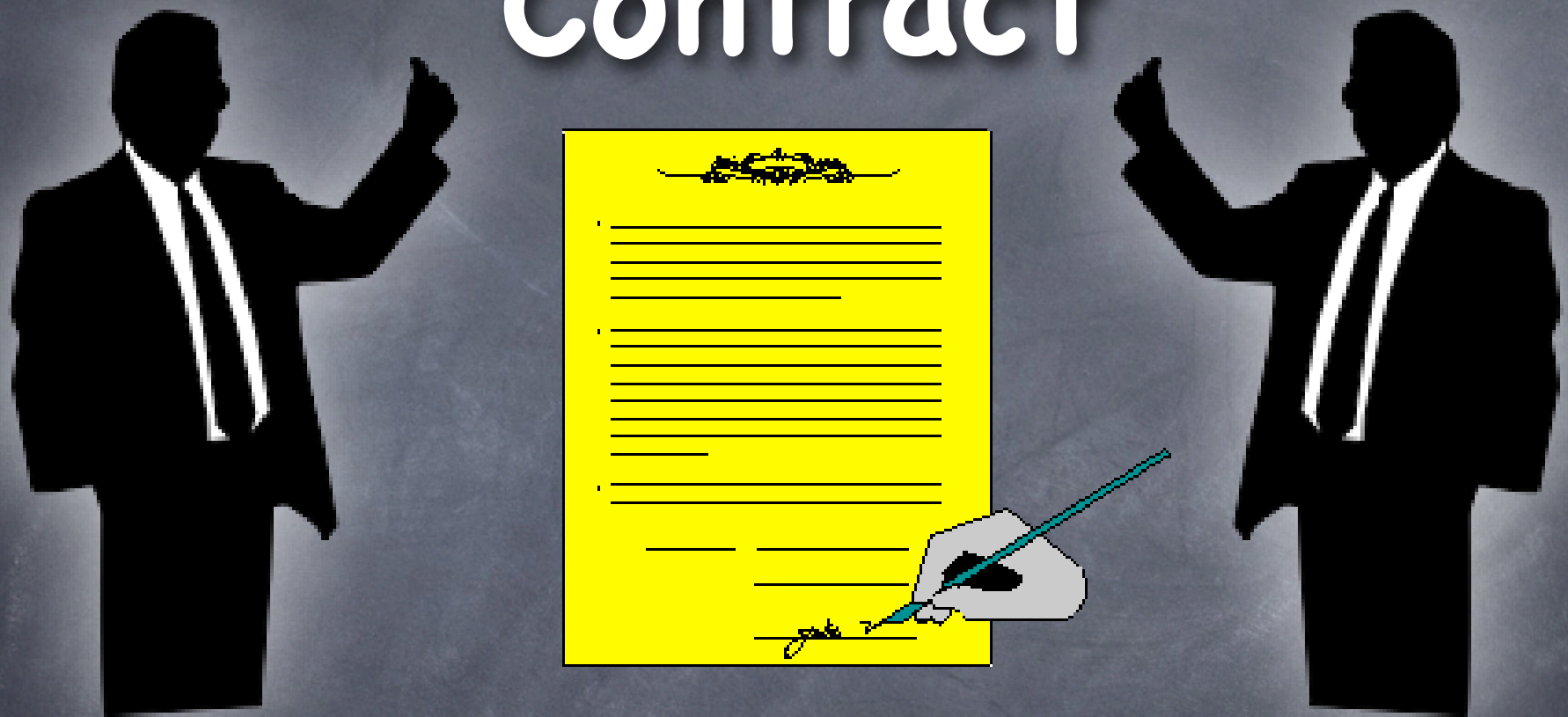
Wondrous Numbers

(n.even?) ? n/2 : 3*n+1

Specification

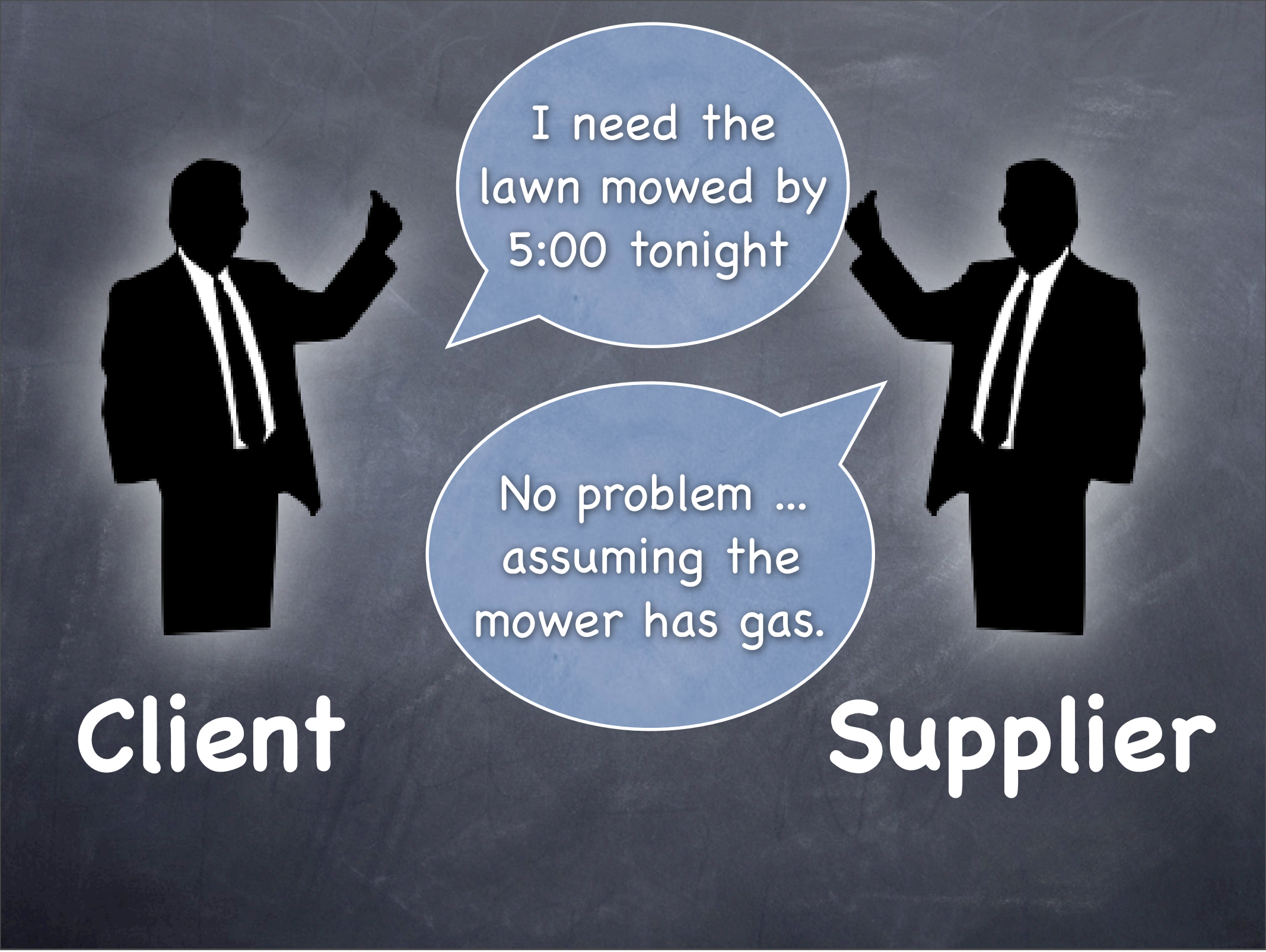
by Contract

Contract



Client

Supplier


I need the lawn mowed by 5:00 tonight
No problem ... assuming the mower has gas.
Client
Supplier

Post Condition

I need the lawn mowed by 5:00 tonight

No problem ... assuming the mower has gas.

Pre Condition

Post Condition

I need the lawn mowed by 5:00 tonight

Responsible

Supplier

No problem ... assuming the mower has gas.

Client

Responsible

Pre Condition

Post Condition

I need the lawn mowed by 5:00 tonight

Benefits

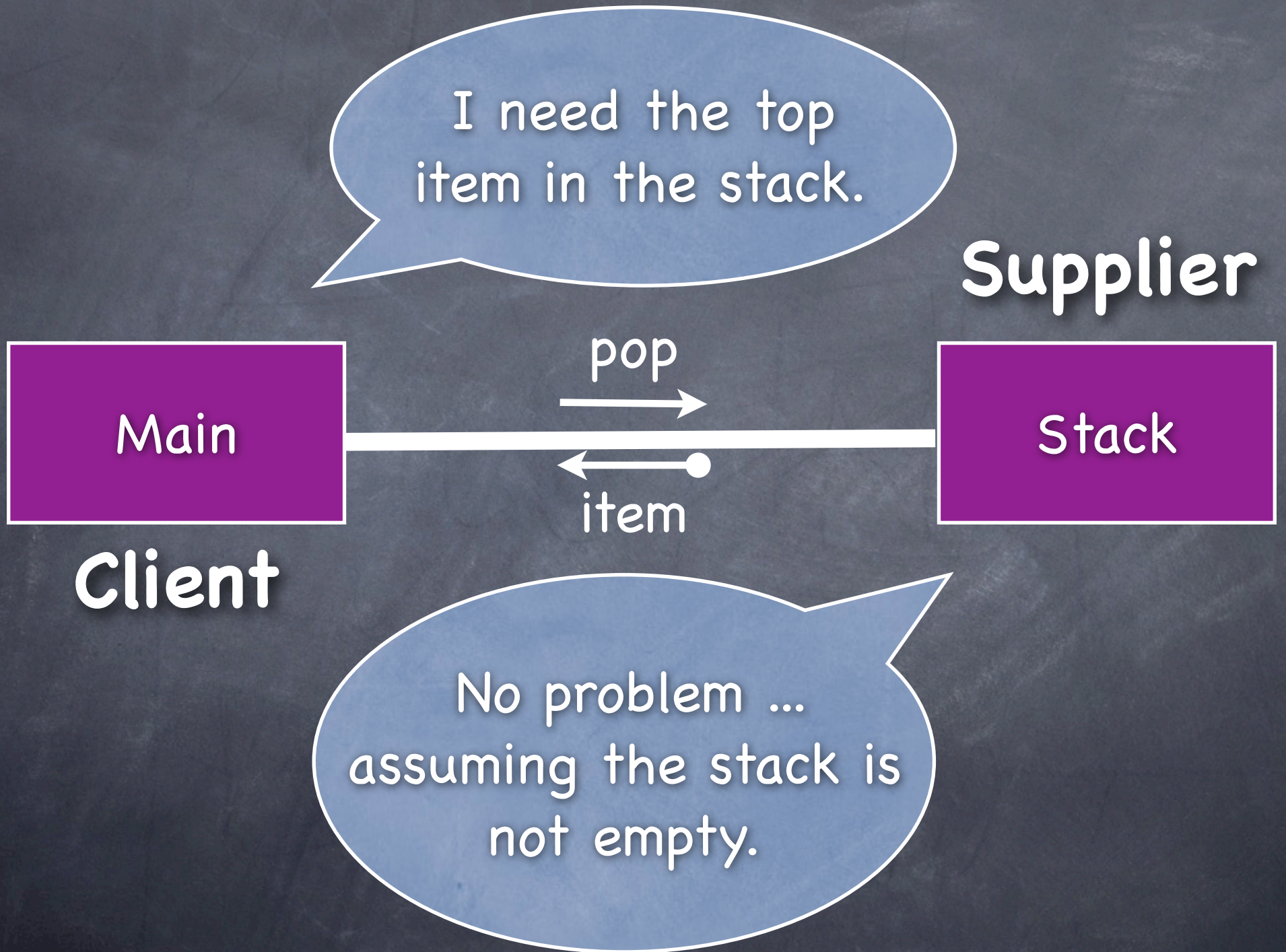
Client

Benefits

Supplier

No problem ... assuming the mower has gas.

Pre Condition


I need the top item in the stack.
Supplier
pop
Main
Stack
item
Client
No problem ... assuming the stack is not empty.

Example

```
class Stack
  def initialize ... end
  def depth ... end
  def empty? ... end
  def top ... end
  def pop ... end
  def push ... end
end
```

```
class Stack
  def initialize ... end
  def depth ... end
  def empty? ... end
  def top ... end
  def pop ... end
  def push ... end
end
```

Queries

```
class Stack
  def initialize ... end
  def depth ... end
  def empty? ... end
  def top ... end
  def pop ... end
  def push ... end
end
```

Commands

Demo

```
require 'dbc'

DBC.specification "Stack" do
end

class MyStack
end

MyStack.obeys_contracts_for "Stack"
```

```
require 'dbc'

DBC.specification "Stack" do
end

class MyStack
end

MyStack.obeys_contracts_for "Stack"
```

```
require 'dbc'

DBC.specification "Stack" do
end

class MyStack
end

MyStack.obeys_contracts_for "Stack"
```

```
require 'dbc'

DBC.specification "Stack" do
end

class MyStack
end

MyStack.obeys_contracts_for "Stack"
```

```
$ ruby demo.rb
Validating MyStack against Stack
```

```
DBC.specification "Stack" do
  contract :empty? do
    post("empty if depth is zero"){
      |result|
      result == (depth == 0)
    }
  end
end
```

```
$ ruby demo.rb
Validating MyStack against Stack
```

```
DBC.specification "Stack" do
  contract :empty? do
    post("empty if depth is zero"){
      |result|
      result == (depth == 0)
    }
  end
  example "Initial Condition" do
    |stack|
    stack.empty?
  end
end
```

```
$ ruby demo.rb
Validating MyStack against Stack
Running Example Initial Condition
./dbc.rb:192:in `__send__':
undefined method `empty?' for
#<MyStack:0xdd68
@dbc_remembered_values={}>
(NoMethodError)
  from ./dbc.rb:192:in `run'
```

```
DBC.specification "Stack" do
  contract :empty? do
    post("empty if depth is zero"){
      |result|
      result == (depth == 0)
    }
  end
  example "Initial Condition" do
    |stack|
    stack.empty?
  end
end
```

```
DBC.specification "Stack" do
   contract :empty? do
      query
   end

   invariant {
      empty? == (depth == 0)
   }

   example "Initial Condition" do
   end
end
```

```
$ ruby demo.rb
Validating MyStack against Stack
Running Example Initial Condition
./dbc.rb:192:in `__send__':
undefined method `empty?' for
#<MyStack:0xdd68
@dbc_remembered_values={}>
(NoMethodError)
  from ./dbc.rb:192:in `run'
```

```
class MyStack
  def empty?
    true
  end
end
```

```
$ ruby demo.rb
Validating MyStack against Stack
Running Example Initial Condition
demo.rb:8: undefined local
variable or method `depth' for
#<MyStack:0xdaac
@dbc_remembered_values={}>
(NameError)
```

```
class MyStack
  def depth
    0
  end
  def empty?
    true
  end
end
```

```
$ ruby demo.rb
Validating MyStack against Stack
Running Example Initial Condition
```

```
DBC.specification "Stack" do
  contract :empty? do
    query
  end

  invariant {
    empty? == (depth == 0)
  }

  example "Initial Condition" do
  end
end
```

```
DBC.specification "Stack" do
  ...
  contract :push do
    remember :depth
    post("Stack size increases"){
      depth == dbc_old(:depth) + 1
    }
  end
  ...
end
```

```
DBC.specification "Stack" do
  ...
  example "Pushing Items" do |stack|
    stack.push(1)
  end
  ...
end
```

```
$ ruby demo.rb
Validating MyStack against Stack
Running Example Empty Stack
Running Example Pushing Items
./dbc.rb:192:in `__send__':
undefined method `push' for
#<MyStack:0xbd88
@dbc_remembered_values={}>
(NoMethodError)
```

```
class MyStack
  def depth
    0
  end
  def empty?
    true
  end
  def push(item)
  end
end
```

```
$ ruby demo.rb
Validating MyStack against Stack
Running Example Empty Stack
Running Example Pushing Items
./dbc.rb:212:in
`check_postconditions':
Postcondition failed on method
push in MyStack: Stack size
increases
(DBC::PostconditionError)
```

```
class MyStack
  attr_reader :depth
  def initialize
    @depth = 0
  end
  def empty?
    true
  end
  def push(item)
    @depth += 1
  end
end
```

```
$ ruby demo.rb
Validating MyStack against Stack
Running Example Empty Stack
Running Example Pushing Items
./dbc.rb:109:in
`check_invariants': Invariant
failed after Method push in
MyStack:  (DBC::InvariantError)
```

```
DBC.specification "Stack" do
   ...
   invariant {
      empty? == (depth == 0)
   }
   ...
end
```

```
class MyStack
  attr_reader :depth
  def initialize
    @depth = 0
  end
  def empty?
    true
  end
  def push(item)
    @depth += 1
  end
end
```

```
class MyStack
  attr_reader :depth
  def initialize
    @depth = 0
  end
  def empty?
    depth == 0
  end
  def push(item)
    @depth += 1
  end
end
```

```
$ ruby demo.rb
Validating MyStack against Stack
Running Example Empty Stack
Running Example Pushing Items
```

```
DBC.specification "Stack" do
  ...
  contract :push do
    remember :depth
    post("Stack size increases"){
      depth == dbc_old(:depth) + 1
    }
    post("Top is pushed item") {
      |result, item|
      top == item
    }
  end
  ...
end
```

```
$ ruby demo.rb
Validating MyStack against Stack
Running Example Empty Stack
Running Example Pushing Items
demo.rb:13:in
`__instance_exec_101280_21000':
undefined local variable or
method `top' for #<MyStack:0xa410
@depth=1, @dbc_remembered_values=
{:depth=>0}> (NameError)
```

```
class MyStack
  attr_reader :depth
  def initialize
    @depth = 0
  end
  def empty?
    depth == 0
  end
  def top
  end
  def push(item)
    @depth += 1
  end
end
```

```
$ ruby demo.rb
Running Example Empty Stack
Running Example Pushing Items
./dbc.rb:212:in
`check_postconditions':
Postcondition failed on method
push in MyStack: Top is pushed
item (DBC::PostconditionError)
```

```
class MyStack
   def initialize() @items = [] end
   def depth
      @items.size
   end
   def empty?
      depth == 0
   end
   def top
      @items.last
   end
   def push(item)
      @items << item
   end
end
```

```
$ ruby demo.rb
Validating MyStack against Stack
Running Example Empty Stack
Running Example Pushing Items
```

```
DBC.specification "Stack" do
  ...
  contract :pop do
    remember :depth
    pre("Stack is not empty"){
      ! empty?
    }
    post("Stack size is decreased") {
      depth == dbc_old(:depth) - 1
    }
  end
  ...
end
```

```
DBC.specification "Stack" do
  ...
  contract :pop do
    remember :depth
    pre("Stack is not empty"){
      ! empty?
    }
    post("Stack size is decreased") {
      depth == dbc_old(:depth) - 1
    }
  end
  ...
end
```

```
DBC.specification "Stack" do
  ...
  contract :pop do
    remember :depth
    pre("Stack is not empty"){
      ! empty?
    }
    post("Stack size is decreased") {
      depth == dbc_old(:depth) - 1
    }
  end
  ...
end
```

```
$ ruby demo.rb
Validating MyStack against Stack
Running Example Empty Stack
Running Example Pushing Items
```

What is missing from pop's contract?

```
DBC.specification "Stack" do
  ...
  contract :pop do
    remember :depth
    pre("Stack is not empty"){
      ! empty?
    }
    post("Stack size is decreased") {
      depth == dbc_old(:depth) - 1
    }
  end
  ...
end
```

Summary

Summary

DbC is a great way to think about behavior

(It plays well with Duck Typing)

Summary

DbC can be
Red/Green/Refactor

Summary

Mixing Contracts and ad hoc asserts.

THE END

Thank You!