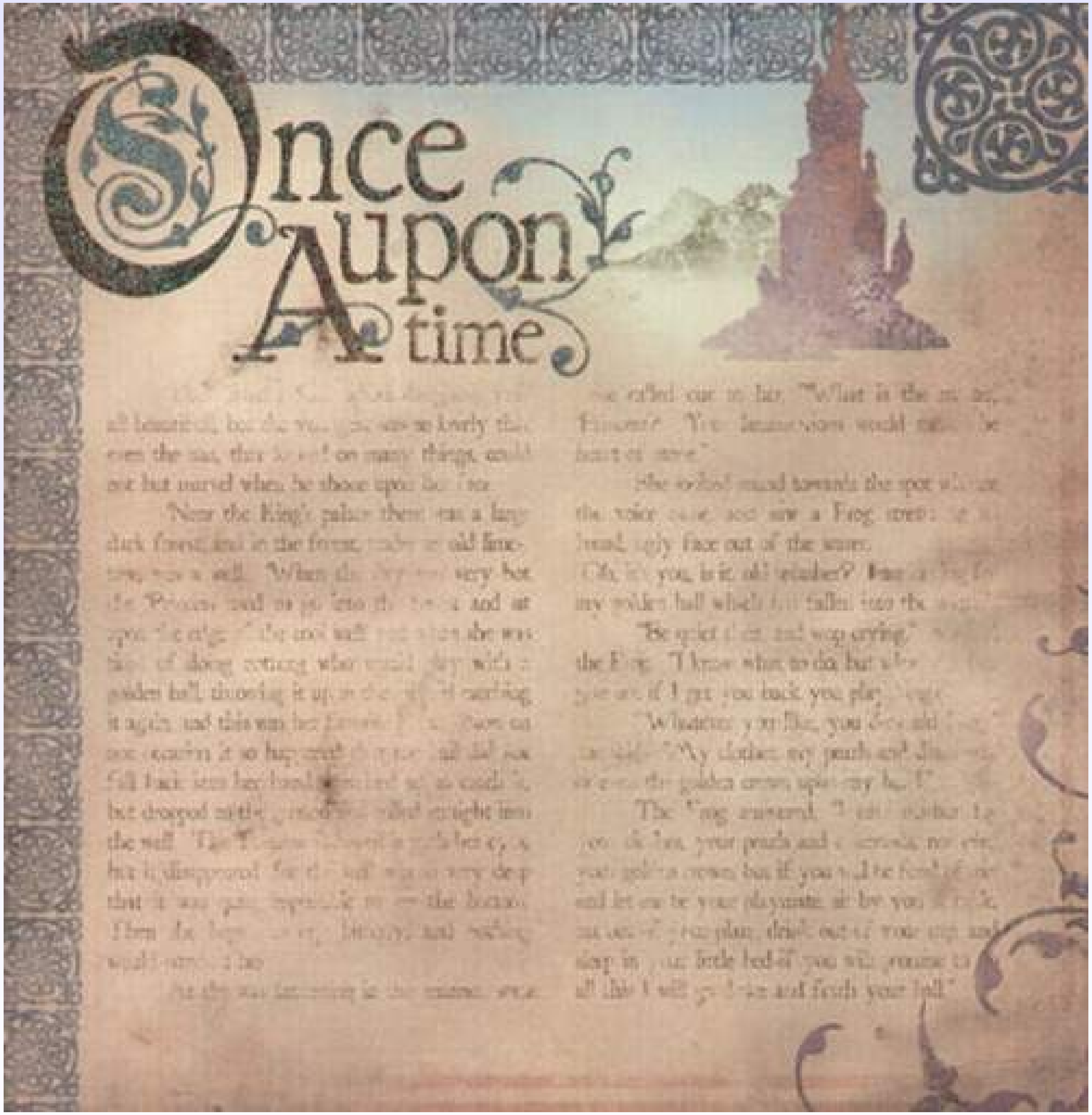
"How to write quality software using the magic of tests"

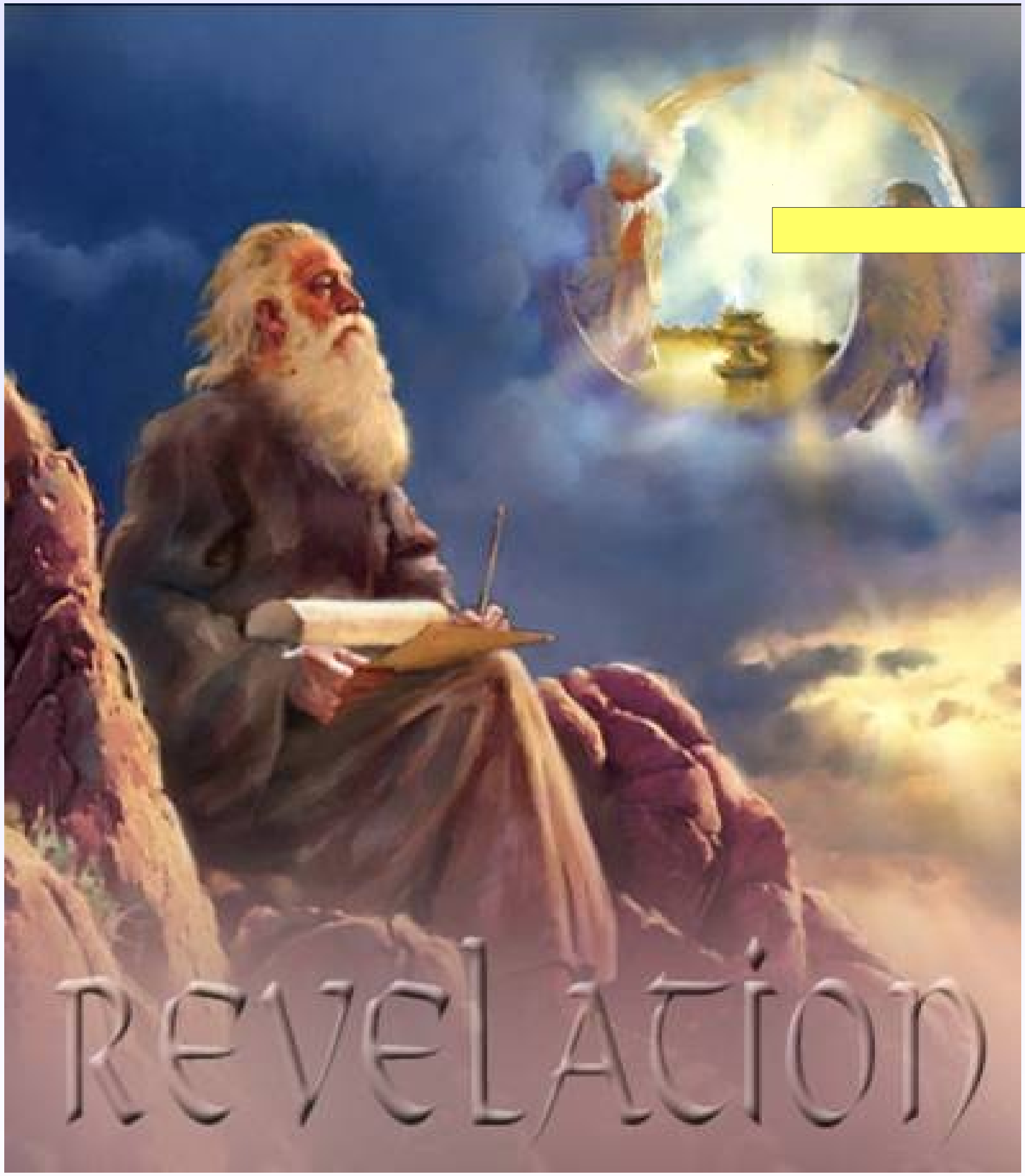
Igal Koshevoy, Pragmaticraft
Business-Technology Consultant
igal@pragmaticraft.com
@igalko on Twitter & Identi.ca

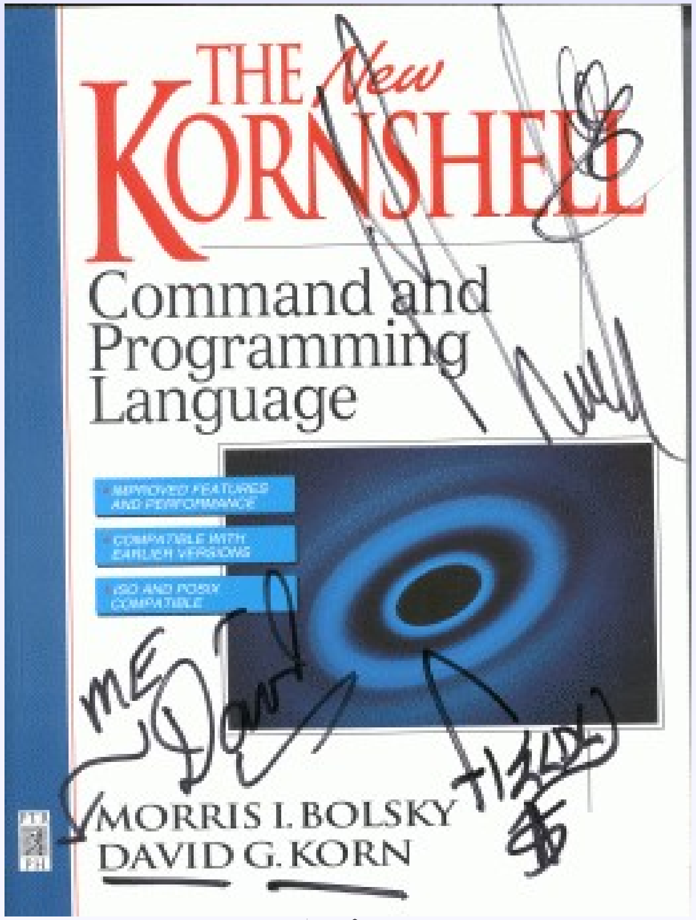

Once upon A time

...in a dark, terrible time...

…while doing data entry...

...there was a revelation...

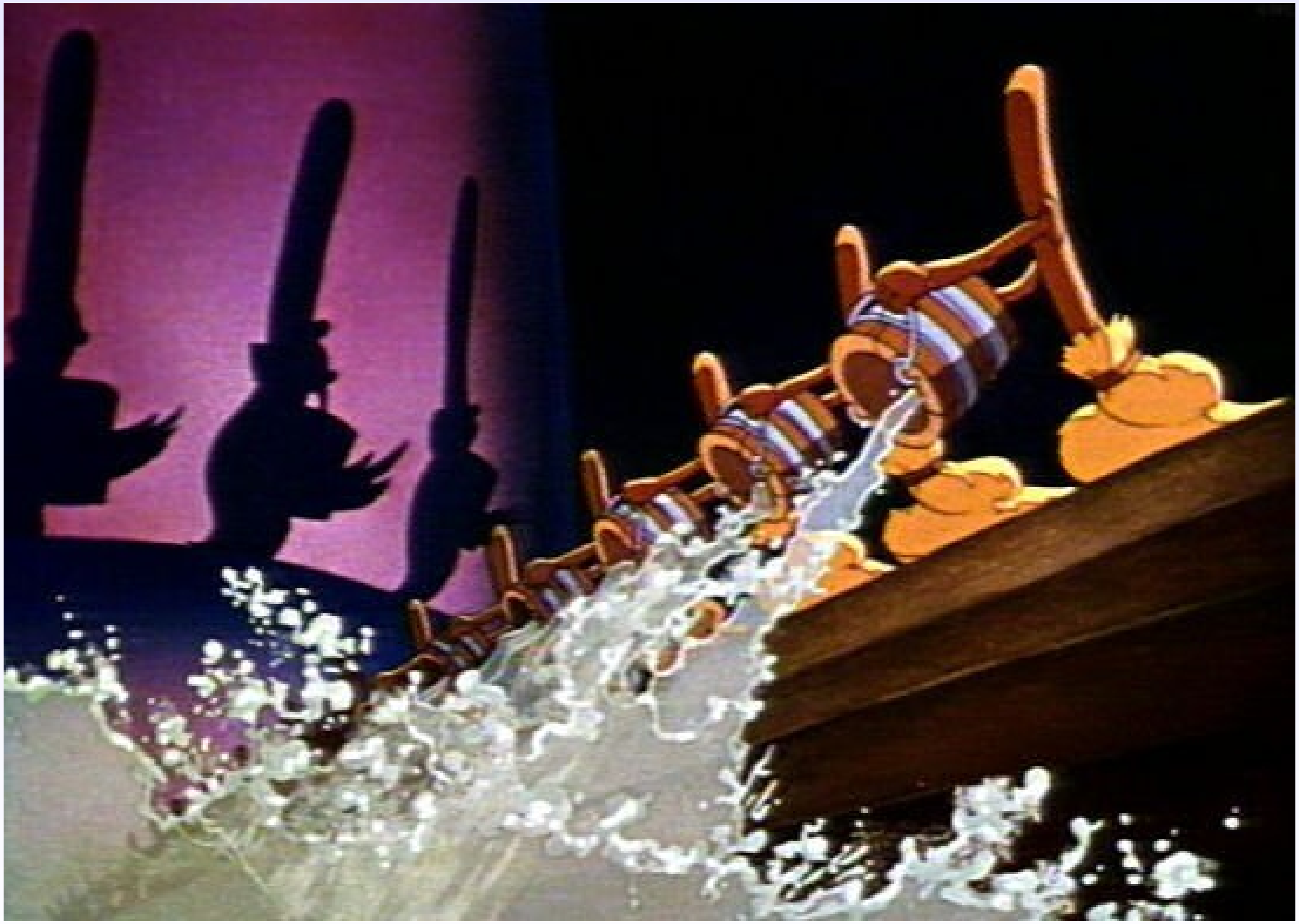



THE New KORNSHELL
Command and Programming Language
IMPROVED FEATURES AND PERFORMANCE
COMPATIBLE WITH EARLIER VERSIONS
ISO AND POSIX COMPATIBLE
ME
MORRIS I. BOLSKY
DAVID G. KORN

(Signed by band KoЯn)

...to magic the work away...

...seemed a good idea, until...



...so needed magic to control magic: tests!

At first there wasn't much need..

Code, test and runner

```
# Code in file "code1.rb"
def add_one_to(value)
  return(value + 1)
end
```

```
# Test in "test1.rb"
require 'code1'
add_one_to(41) == 42 or fail
```

```
# Test runner from command-line
$ ruby test1.rb
```

...but then it got complicated...

Conditionals and test frameworks

```
# Code in file 'code2.rb'
def add_one_when_even(value)
  if (value % 2) == 0
    return value + 1
  else
    return value
  end
end
```

```
# Test runner
ruby test2.rb
```

```
# Test in file 'test2.rb'
require 'test/unit'
require 'code2'

class TestCode2 < Test::Unit::TestCase
  def test_should_add_when_given_even_number
    assert_equal(3, add_one_when_even(2))
  end

  def test_should_not_add_when_given_odd_number
    assert_equal(1, add_one_when_even(1))
  end
end
```

...and required an understanding of history...

Test-wide setup

```
# Code in 'code3.rb'
class StatefulAdder
  attr_accessor :current_value

  def initialize(initial_value)
    self.current_value = initial_value
  end

  def increment_when_even(value)
    if (value % 2) == 0
      self.current_value += 1
      return(self.current_value)
    else
      return(self.current_value)
    end
  end
end
```

```
# Test in 'test3.rb'
require 'test/unit'
require 'code3'

class TestCode3 < Test::Unit::TestCase
  def setup
    @adder = StatefulAdder.new(1)
  end

  def test_increment_when_given_even_number
    @adder.increment_when_even(2)
    assert_equal(2, @adder.current_value)
  end

  def test_not_increment_when_given_odd_number
    @adder.increment_when_even(1)
    assert_equal(1, @adder.current_value)
  end
end
```

...and need a little help...

Utility methods

```
# Test in 'test3b.rb'
require 'test/unit'
require 'code3'

class TestCode3b < Test::Unit::TestCase
  def assert_adder(initial_value, added_value, expected_value)
    adder = StatefulAdder.new(initial_value)
    adder.increment_when_even(added_value)
    assert_equal(expected_value, adder.current_value)
  end

  def test_should_increment_when_given_even_number
    assert_adder 1, 8, 2
  end

  def test_should_not_increment_when_given_odd_number
    assert_adder 1, 9, 1
  end
end
```

...and complex preparation...

Factories

```
# Test in 'test3c.rb'
require 'test/unit'
require 'code3'

class TestCode3c < Test::Unit::TestCase
  def adder_factory(initial_value)
    return StatefulAdder.new(initial_value)
  end

  def test_should_increment_when_given_even_number
    adder = adder_factory(1)
    adder.increment_when_even(2)
    assert_equal(2, @adder.current_value)
  end

  def test_should_not_increment_when_given_odd_number
    adder = adder_factory(3)
    adder.increment_when_even(1)
    assert_equal(3, @adder.current_value)
  end
end
```

..and invasive...

Stubbing

```
# Code in 'code4.rb'
require 'uri'
require 'net/http'

def count_es_in_document_at(url_string)
  return get_document_at(url_string) ⇢
      .scan(/e/i).length
end

def get_document_at(url_string)
  # TODO Download and return body
end
```

```
# Test in 'test4.rb'
require 'test/unit'
require 'code4'
require 'rubygems'
require 'mocha'

class TestCode4 < Test::Unit::TestCase
  def test_count_es_in_document

    sample_document = ⇢
      "This is some text containing the letter 'E'."

    self.stubs(:get_document_at => sample_document)

    assert_equal(6, ⇢
      count_es_in_document_at("http://foo.bar/baz"))
  end
end
```

...and need to blend in...

Mocks

```
# Code in 'code5.rb'
class Employee
  attr_accessor :name
  attr_accessor :company

  def initialize(opts)
    self.name = opts[:name]
    self.company = opts[:company]
  end

  def label
    if self.company
      return self.name + ', ' ⤑
        + self.company.name
    else
      return self.name
    end
  end
end
```

```
require 'test/unit'
require 'code5'
require 'rubygems'
require 'mocha'

class TestCode5 < Test::Unit::TestCase
  def test_label_without_company
    employee = Employee.new(:name => 'Joe Smith')
    assert_equal('Joe Smith', employee.label)
  end

  def test_label_with_company
    company = mock('Company', :name => 'SmithCo.')
    employee = Employee.new(
      :name => 'Joe Smith', :company => company)
    assert_equal('Joe Smith, SmithCo.', employee.label)
  end
end
```

TDD: Test-Driven Development

Fundamentalist:

1. Write a test
2. Run the test
3. See test fail
4. Write code
5. Run the test
6. Keep at it till the test passes

Liberal:

Write your test and code in same session.

BDD: Behavior-Driven Development

Like TDD, but goal is to write tests as natural language specifications, e.g.:

Employee label without a company
- should include just the employee's name

Employee label with company
- should include the employee's name and company name

Test vs. spec

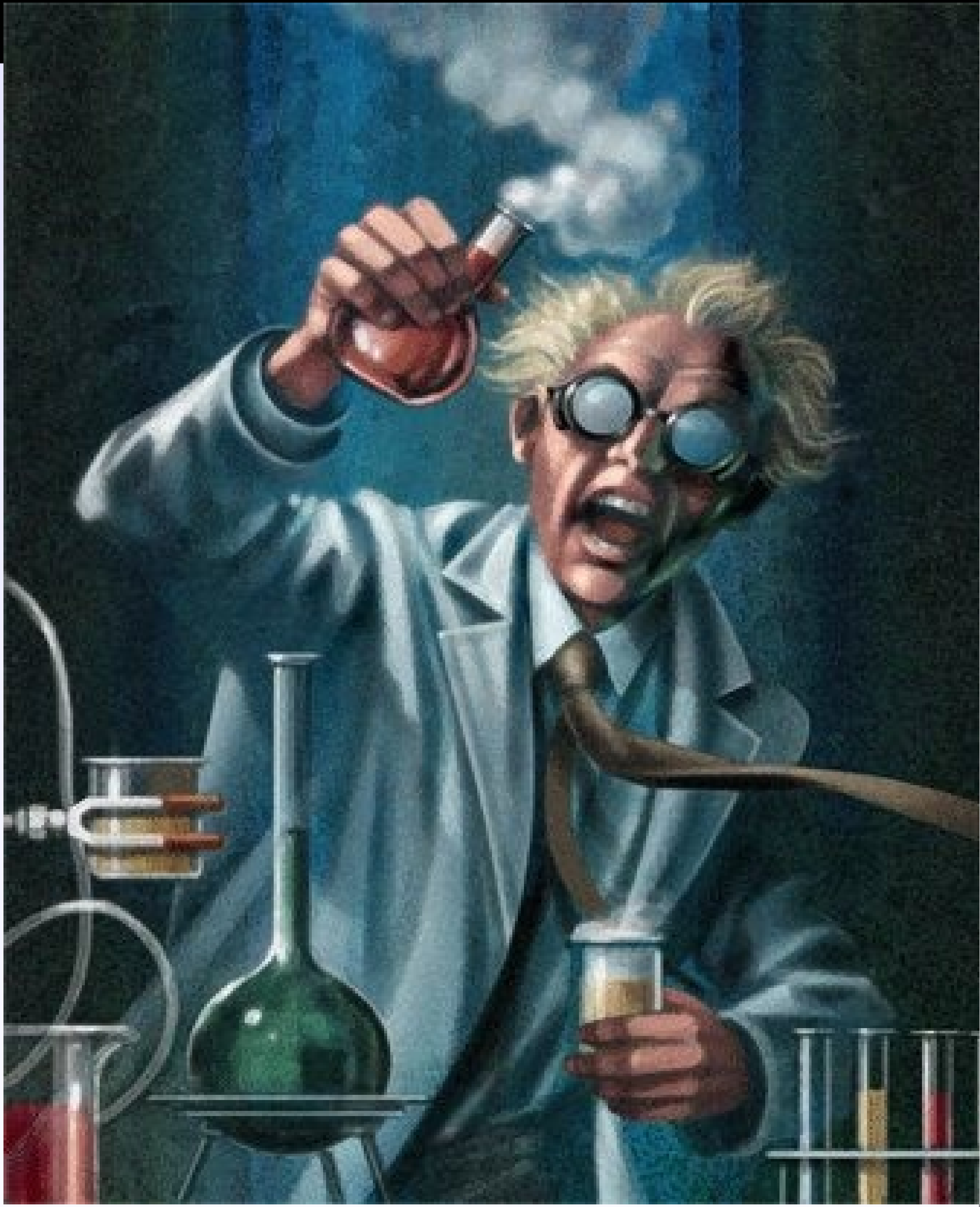
```
# Test
class TestCode5 < Test::Unit::TestCase
  def test_label_without_company...
  def test_label_with_company...
end

# Spec
describe Employee
  describe "label" do
    describe "without a company" do
      it "should include just the employee's name"
    end
    describe "with a company" do
      it "should include the employee's name and company name"
    end
  end
end
```

QUALITY!!1!

It depends.

Who decides? The geeks??!



The Executive
The Sponsor
The Client

The Stakeholders

Not everyone will be convinced.

Rescuing a troubled project.

Sanity check?

How much is too much?

Happy trails!

Igal Koshevoy
Biz-Tech Consultant
igal@pragmaticraft.com
@igalko