no excuses

for no automatic testing on c++ projects

by Dmitry Ledentsov on 26.03.2015
presented at TU München
for the MUC++ Meetup

Why?

- humans make mistakes
- heard and made many excuses
- seen benefits of the empirical approach
- would like to hear more excuses
- by testing I don’t mean hours of manual clicking to see if something that you waited for half an hour to compile works
- tests: automated, preferably fast, useful, “just enough"

for today

- excuses
- libraries: a run-through, a tracer bullet
- code
- links (clickable)
- books
- discussion

a situation

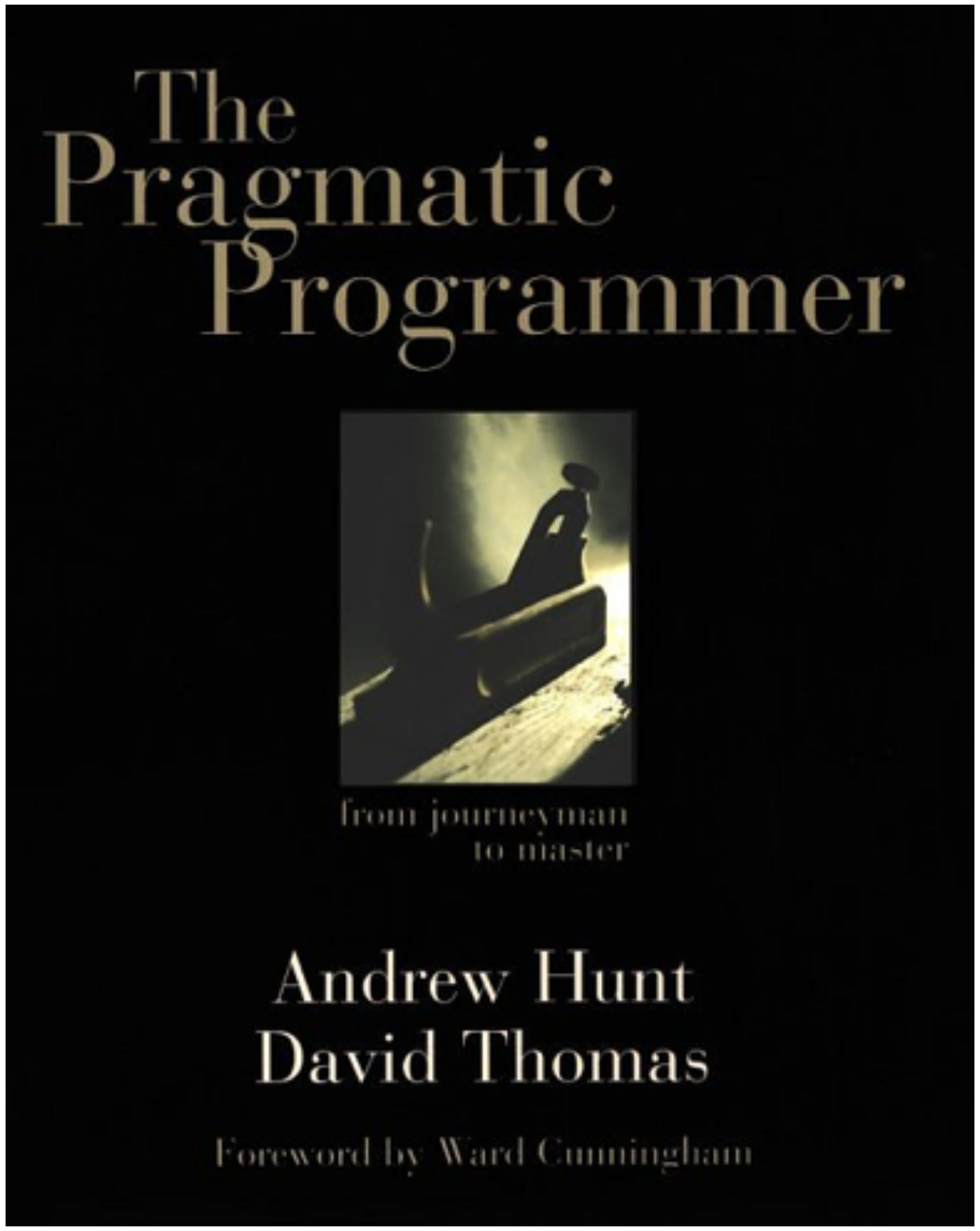
It’s Thursday. Your boss asks you to deliver a shaky feature for Monday.
You accept, but discover the task not as easy as thought.
There are risky dependencies that might break - it’s a big ball of mud
But! There’s an easy hack! Who needs testing, I’ll be ready by Monday!

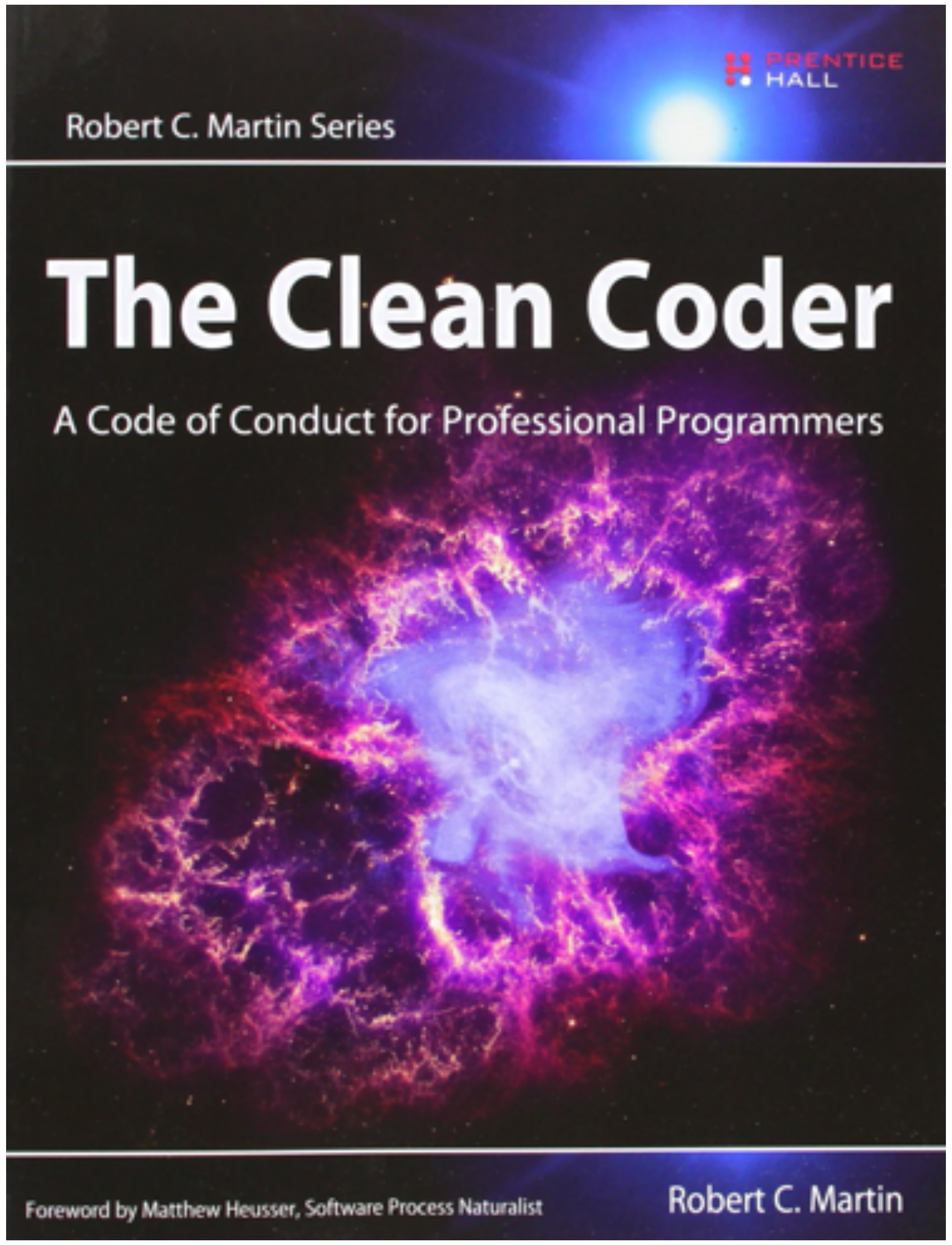
excuses not to test

- no time, gotta deliver
- boss forbids to
- no tools for c++
- too long or hard to set up testing environment
- dependency hell
- I know what I'm doing!
- nobody tests around here

name yours later!

get scared




PRENTICE HALL
Robert C. Martin Series
The Clean Coder
A Code of Conduct for Professional Programmers
Foreword by Matthew Heusser, Software Process Naturalist
Robert C. Martin

Source: Amazon

questions

- how do you know it works, then?
- how do you know it's the right thing?
- how does it affect the rest of the system?
- does it have the expected quality?
- how do others know what exactly you've done?

no time, gotta deliver

it'll bite you back!

TECHNICAL DEBT

boss forbids to

you'll be blamed for failure!

how does he know, you've done your job properly?

too complicated
to set up or learn

let's talk about that

some subjective priorities when choosing a tool

- *sufficient documentation/test/community (quality)*
- *platform-'independent'*
- single header
- header-only
- single header, single source
- builds ad hoc
- builds out of the box

no tools for c++
/
don’t know how

let's see what we've got

starting small

```
#define CATCH_CONFIG_MAIN
#include <catch.hpp>

TEST_CASE("failing") {
  FAIL("wrote a failing test!");
}
```

catch-lib.net

- single header
- very little to learn

a neat feedback

```
---------------------------------------------------------------
failing
---------------------------------------------------------------
src/no_excuses.cpp:4
...............................................................

src/no_excuses.cpp:5: FAILED:
explicitly with message:
  wrote a failing test!

===============================================================
test cases: 1 | 1 failed
assertions: 1 | 1 failed
```

fix it

```
TEST_CASE("that's easy") {
  CHECK( (2 + 2) == 4 );
}
```

```
===============================================================================
All tests passed (1 assertion in 1 test case)
```

expression template magic —> simple assertion macros

a simple build config

```
include 'premake'

make_solution 'no_excuses'

platforms 'native'

includedirs {
    'deps/Catch'
}

make_console_app('no_excuses', {
    './src/no_excuses.cpp'
})

run_target_after_build()
```

http://premake.bitbucket.org

https://github.com/d-led/premake-meta-cpp

before we continue

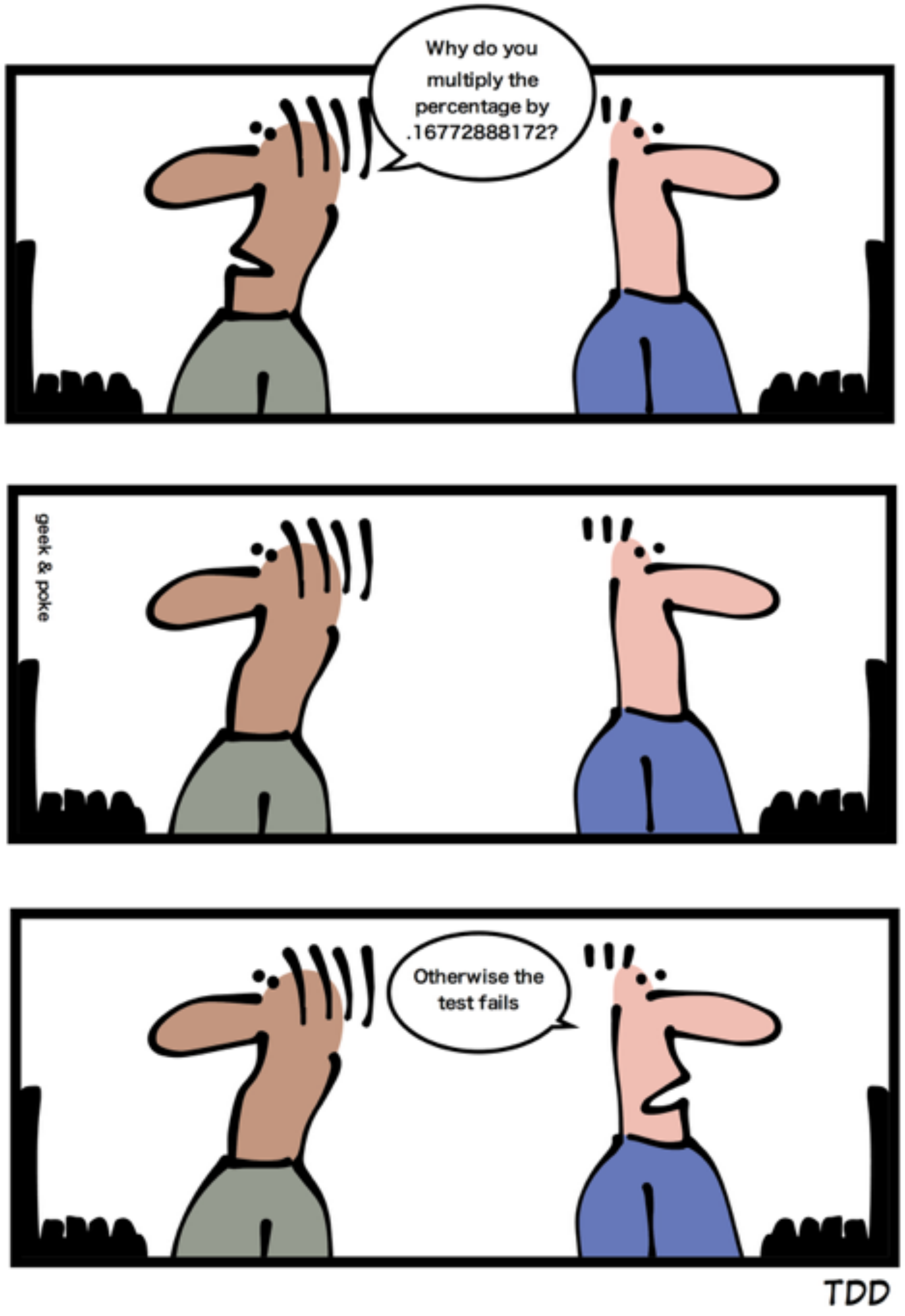
there are anti-patterns

```
TEST_CASE("blop_blup","[mip][x63h]") {
 auto i = blop_blup(77);
 i += 21;
 strt_wbsrvr_chk();
 CHECK( zog(i) == 42 );
}
```

- what's that about?
- what is it trying to convey?
- will you understand it tomorrow?

assertFalse? SHOULDN'T IT BE assertTrue?
geek & poke
WITH assertTrue THE TEST WENT RED
DEVELOPMENT DRIVEN TESTS

http://geekandpoke.typepad.com/geekandpoke/2011/10/ddt.h



http://geek-and-poke.com/geekandpoke/2013/7/28/tdd

before we continue

```
TEST_CASE("finding the ultimate answer","[answer][ultimate]") {
   REQUIRE( find_ultimate_answer() == 42 );
}
```

- wouldn’t it be nice if test read like English?
- you might be able to read and understand it tomorrow

structuring tests

```
TEST_CASE("my superb feature") {
  CHECK( feature() == ok );
}

TEST_CASE("another feature") {
  SECTION("preparing stuff") {
    auto stuff = prepare_stuff();
    REQUIRE( stuff.get() );

    SECTION("stuff should work") {
      CHECK( stuff->works() );
    }
  }
}
```

tests are code - it's still a good idea to structure them

mocks? that's for Java!

google
mock

an interface and its mock

```
struct id_source {
    virtual int new_id() = 0;
    virtual ~id_source() {}
};
```

```
class mock_id_source : public id_source {
public:
    MOCK_METHOD0(new_id, int());
};
```

yes, you can use them as template parameters too

mock configuration & test

```
auto ids = std::make_shared<mock_id_source>();
auto fac = factory(ids);

using ::testing::Return;
EXPECT_CALL(*ids, new_id())
    .Times(1)
    .WillRepeatedly(Return(42));


EXPECT_EQ("42", fac.new_element()->id());
```

behavior-driven?
that's for ruby

here's a cheap one

catch-lib.net

focus on behavior

```
SCENARIO("acquiring wisdom") {
   GIVEN("an oracle") {
      oracle gus;

      WHEN("I ask it to speak") {
         auto answer = gus.speak();

         THEN("wisdom is apparent") {
            CHECK( answer != "bla" );
         }
      }
   }
}
```

pretty failure reporting

```
-------------------------------------------------------------------------------
Scenario: acquiring wisdom
     Given: an oracle
      When: I ask it to speak
      Then: wisdom is apparent
-------------------------------------------------------------------------------
src/no_excuses.cpp:15
...............................................................................

src/no_excuses.cpp:23: FAILED:
  CHECK( answer != "bla" )
with expansion:
  "bla" != "bla"

===============================================================================
test cases: 2 | 1 passed | 1 failed
assertions: 1 | 1 failed
```

test quality

- who do you write the test for?
- what's its value?
- what's its maintenance cost?

communicating via specification

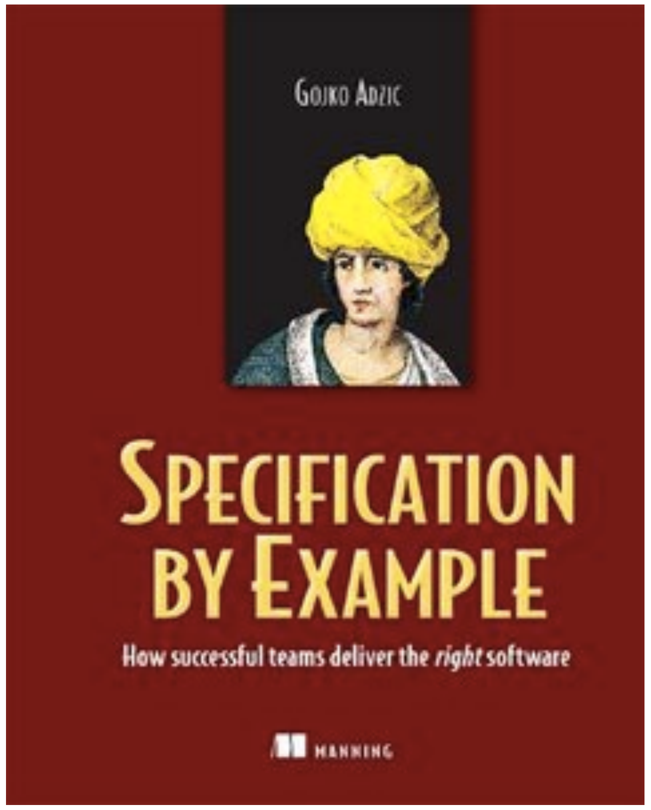
let's grow cucumbers

what are you doing, actually?

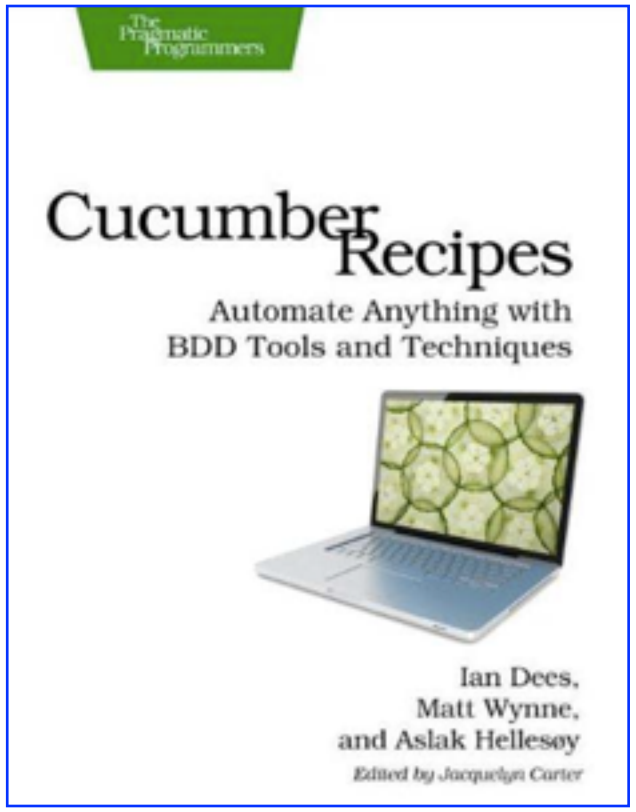
It’s Monday, and your boss wants to know what you’re coding.
The feature should have been implemented by morning!
Do you have an answer?

here’s the user story!

specification by example



https://cukes.info


The Pragmatic Programmers
Cucumber Recipes
Automate Anything with BDD Tools and Techniques
Ian Dees, Matt Wynne, and Aslak Hellesøy
Edited by Jacquelyn Carter

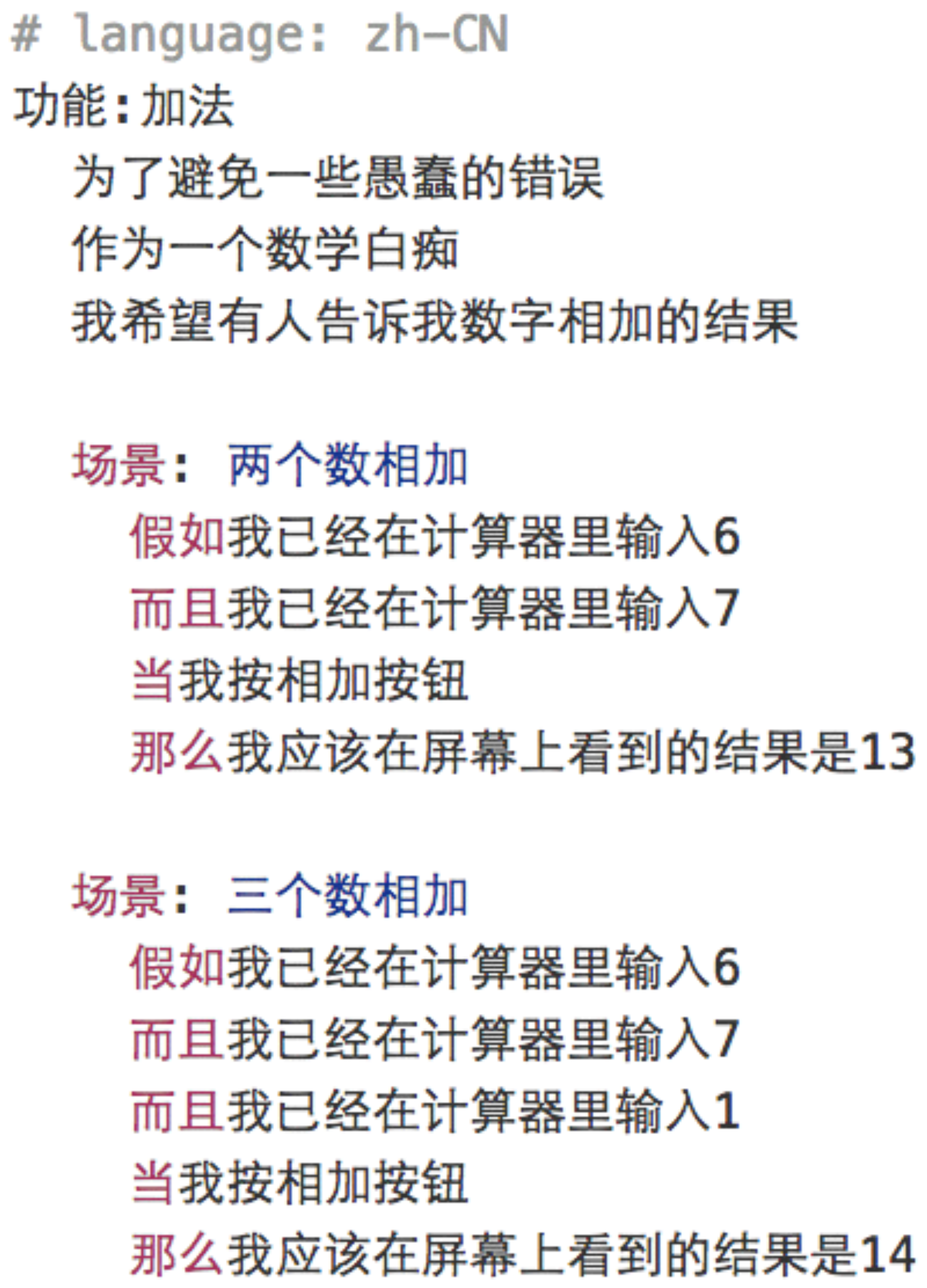
- you're not testing, but specifying behaviour of software
- communicating in a semi-formal, but readable language
- the specification is parsed to drive tests

readable specification: gherkin

```
# language: en
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario Outline: Add two numbers
    Given I have entered <input_1> into the calculator
    And I have entered <input_2> into the calculator
    When I press <button>
    Then the result should be <output> on the screen

  Examples:
    | input_1 | input_2 | button | output |
    | 20      | 30      | add    | 50     |
    | 2       | 5       | add    | 7      |
    | 0       | 40      | add    | 40     |
```

https://github.com/cucumber/cucumber/blob/master/examples/i18n/en/features/addition.feature

localizable



```
# language: zh-CN
功能:加法
  为了避免一些愚蠢的错误
  作为一个数学白痴
  我希望有人告诉我数字相加的结果

  场景: 两个数相加
    假如我已经在计算器里输入6
    而且我已经在计算器里输入7
    当我按相加按钮
    那么我应该在屏幕上看到的结果是13

  场景: 三个数相加
    假如我已经在计算器里输入6
    而且我已经在计算器里输入7
    而且我已经在计算器里输入1
    当我按相加按钮
    那么我应该在屏幕上看到的结果是14
```

https://github.com/cucumber/cucumber/blob/master/examples/i18n/zh-CN/features/addition.feature

github.com/.../cucumber-cpp

```
Feature: Elements with Ids
  In order to manage objects
  As a user
  I want objects to be identifiable

  Scenario: first element
    Given an element source
    When I request an element
    Then its Id is not 0
```

```
THEN("^its Id is not 0$") {
    ScenarioScope<Elements> context;

    specify(context->ids.size(), should.equal(1));
    specify(context->ids[0], not should.equal("0"));
}
```

failing

```
# language: en
Feature: Elements with Ids
  In order to manage objects
  As a user
  I want objects to be identifiable

  Scenario: first element      # features/elements.feature:8
    Given an element source    # cppspec_steps.cpp:26
    When I request an element # cppspec_steps.cpp:30
    Then its Id is not 0       # cppspec_steps.cpp:38
      expected 0, but was 0 (Cucumber::WireSupport::WireException)
      features/elements.feature:11:in `Then its Id is not 0'

Failing Scenarios:
cucumber features/elements.feature:8 # Scenario: first element

1 scenario (1 failed)
3 steps (1 failed, 2 passed)
0m0.010s
Done.
```

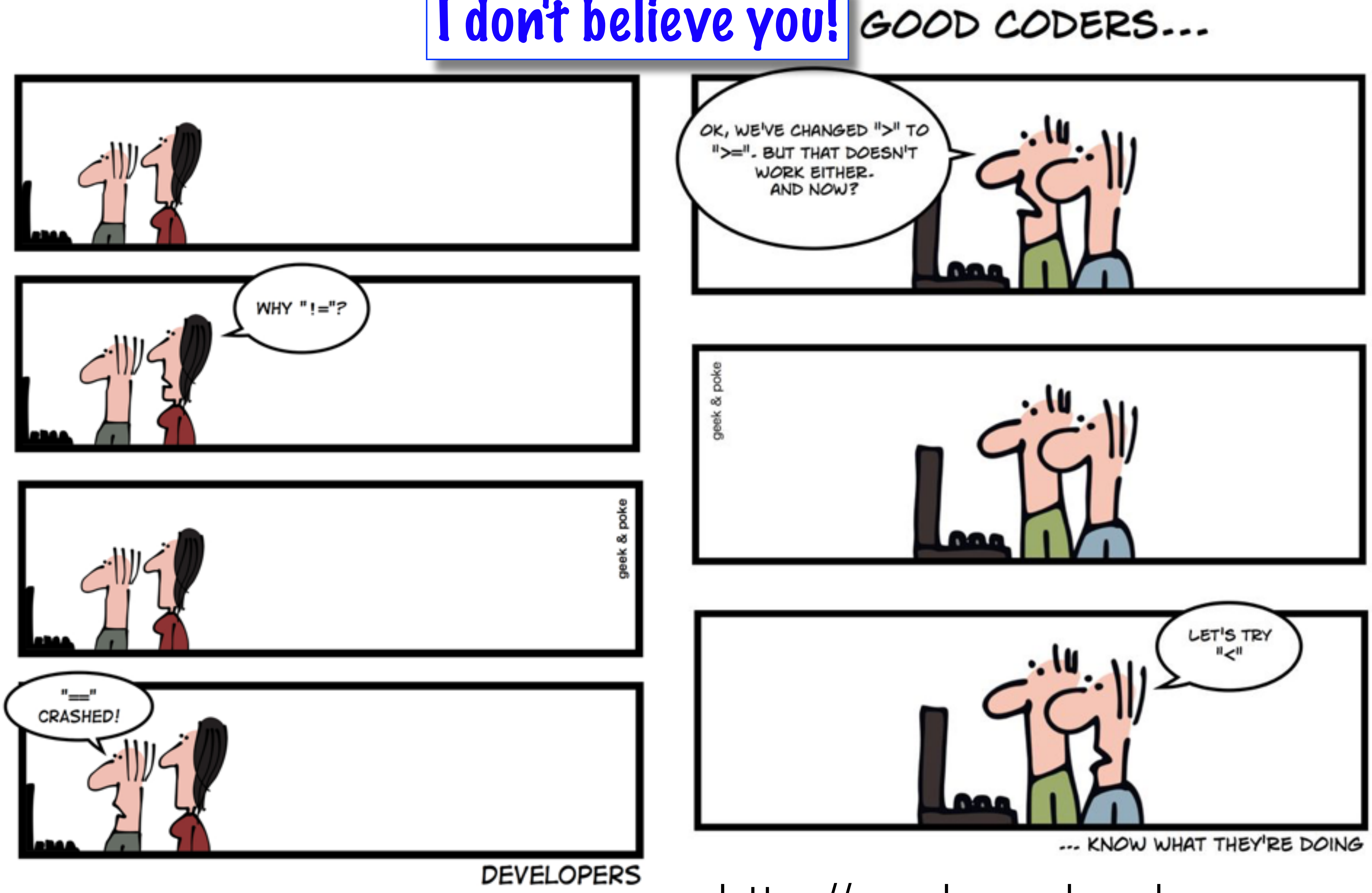
missing step definitions

```
Feature: Elements with Ids
  In order to manage objects
  As a user
  I want objects to be identifiable

...

  Scenario: consecutive elements                    # features/elements.feature:13
    Given an element source                         # cppspec_steps.cpp:26
    When I request an element                       # cppspec_steps.cpp:30
    And then I request another element              # cppspec_steps.cpp:30
    Then the names of the elements are different    # features/elements.feature:17

2 scenarios (1 undefined, 1 passed)
7 steps (1 undefined, 6 passed)
0m0.028s

You can implement step definitions for undefined steps with these snippets:

THEN("^the names of the elements are different$") {
    pending();
}
```

the stakeholders can change the specification and see the impact of the change

I know what I'm doing!

I don't believe you!


GOOD CODERS...
OK, WE'VE CHANGED ">" TO ">=". BUT THAT DOESN'T WORK EITHER. AND NOW?
LET'S TRY "<"
... KNOW WHAT THEY'RE DOING
WHY "!="?
"==" CRASHED!
DEVELOPERS
geek & poke
geek & poke

http://geek-and-poke.com

measure it!

```
#include <iostream>
#include <hayai.hpp>
#include <boost/lexical_cast.hpp>

BENCHMARK(Stringify, stringstream, 1000, 3000)
{
    std::stringstream ss;
    ss<<42;
    auto res = ss.str();
}

BENCHMARK(Stringify, lexical_cast, 1000, 3000)
{
    auto res = boost::lexical_cast<std::string>(42);
}
```

https://github.com/nickbruun/hayai

https://github.com/DigitalInBlue/Celero

measure and remeasure!

```
[==========] Running 2 benchmarks.
[ RUN      ] Stringify.stringstream (1000 runs, 3000 iterations per run)
[     DONE ] Stringify.stringstream (1917.194273 ms)
[   RUNS   ]        Average time: 1917.194 us
                         Fastest: 1567.196 us (-349.998 us / -18.256 %)
                         Slowest: 3757.601 us (+1840.407 us / +95.995 %)

             Average performance: 521.59555 runs/s
                Best performance: 638.08228 runs/s (+116.48673 runs/s / +22.33277 %)
               Worst performance: 266.12724 runs/s (-255.46830 runs/s / -48.97824 %)
[ITERATIONS]        Average time: 0.639 us
                         Fastest: 0.522 us (-0.117 us / -18.256 %)
                         Slowest: 1.253 us (+0.613 us / +95.995 %)

             Average performance: 1564786.64799 iterations/s
                Best performance: 1914246.84596 iterations/s (+349460.19797 iterations/s / +22.33277 %)
               Worst performance: 798381.73345 iterations/s (-766404.91454 iterations/s / -48.97824 %)
[ RUN      ] Stringify.lexical_cast (1000 runs, 3000 iterations per run)
[     DONE ] Stringify.lexical_cast (573.416800 ms)
[   RUNS   ]        Average time: 573.417 us
                         Fastest: 456.937 us (-116.480 us / -20.313 %)
                         Slowest: 1302.187 us (+728.770 us / +127.093 %)

             Average performance: 1743.93216 runs/s
                Best performance: 2188.48550 runs/s (+444.55334 runs/s / +25.49144 %)
               Worst performance: 767.93886 runs/s (-975.99330 runs/s / -55.96510 %)
[ITERATIONS]        Average time: 0.191 us
                         Fastest: 0.152 us (-0.039 us / -20.313 %)
                         Slowest: 0.434 us (+0.243 us / +127.093 %)

             Average performance: 5231796.48730 iterations/s
                Best performance: 6565456.50713 iterations/s (+1333660.01983 iterations/s / +25.49144 %)
               Worst performance: 2303816.57934 iterations/s (-2927979.90796 iterations/s / -55.96510 %)
[==========] Ran 2 benchmarks.
```

data-driven? script it!

```
#include <counter/counter.h>

#include <lua.hpp>
#include <LuaBridge.h>
#include <RefCountedPtr.h>

void register_bindings(lua_State* L) {
    luabridge::getGlobalNamespace(L)
        .beginNamespace("my")
            .beginClass<counter<>>("counter")
                .addConstructor<void(*)(), RefCountedPtr<counter<>>>()
                .addFunction("next", &counter<>::next)
            .endClass()
        .endNamespace()
    ;
}

#ifdef _MSC_VER
#define TEST_BINDINGS __declspec(dllexport)
#else
#define TEST_BINDINGS
#endif

extern "C" TEST_BINDINGS int luaopen_test_bindings(lua_State* L) {
    register_bindings(L);
    return 0;
}
```

bdd-style without recompilation of steps

```lua
assert(require 'test_bindings')

describe("a counter",function()
  local counter = my.counter()

  local starting_value = counter:next()

  it("should start with a zero",function()
    assert.are.equal(starting_value, 0)
  end)

  it("should continue with an increment of 1",function()
    assert.are.equal(counter:next(), 1)
    assert.are.equal(counter:next(), 2)
  end)
end)
```

http://olivinelabs.com/busted/

red—>green

```
cpp-testing-no-excuses$ busted
●■
1 success / 1 failure / 0 errors / 0 pending : 0.002864 seconds

Failure → ./spec/counter_spec.lua @ 14
a counter should continue with an increment of 1
./spec/counter_spec.lua:16: Expected objects to be equal.
Passed in:
(string) 'oops!'
Expected:
(number) 2


…


cpp-testing-no-excuses$ busted
●●
2 successes / 0 failures / 0 errors / 0 pending : 0.001519 seconds
```

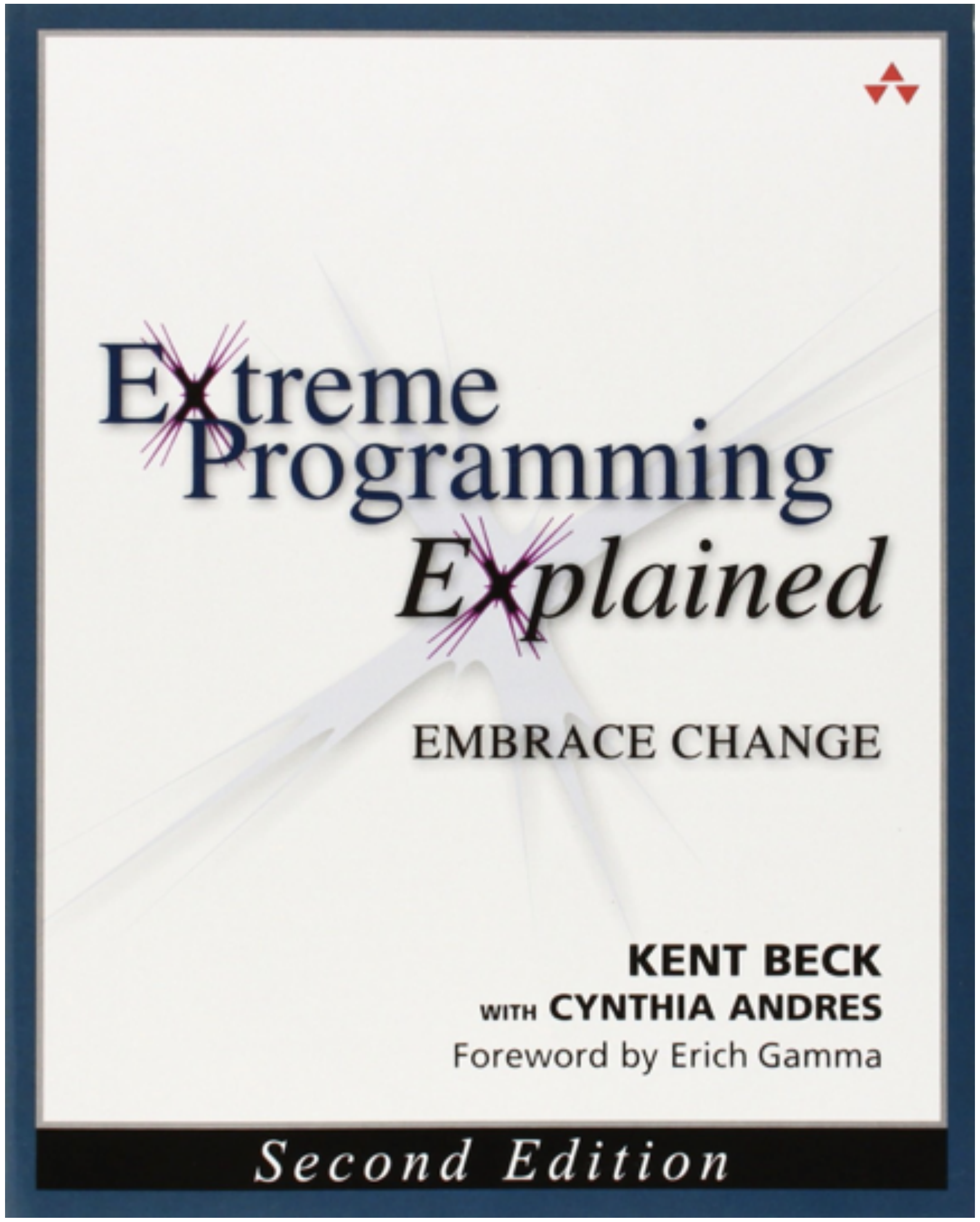
there's more to testing

- nonfunctional tests: code quality tools
- stress/load testing
- property-based testing (quickcheck++)
- eyeball testing
- name yours...

there's more to testing

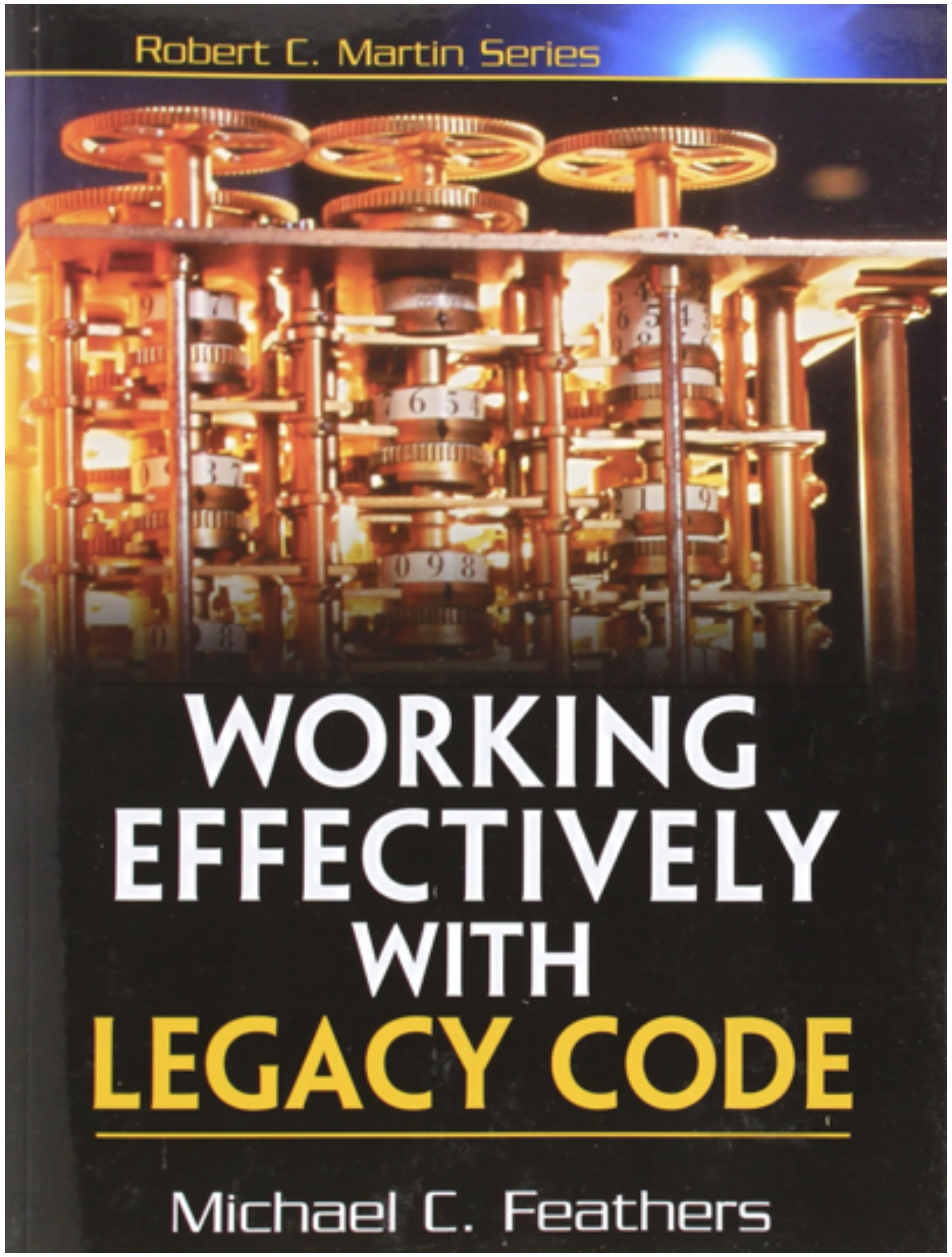
- Management
- Team
- Prioritization
- Maintenance
- Style
- Learning
- Living documentation

just enough, just in time



Source: Amazon

dependency hell
aka big ball of mud



Source: Amazon

don't forget
the big picture

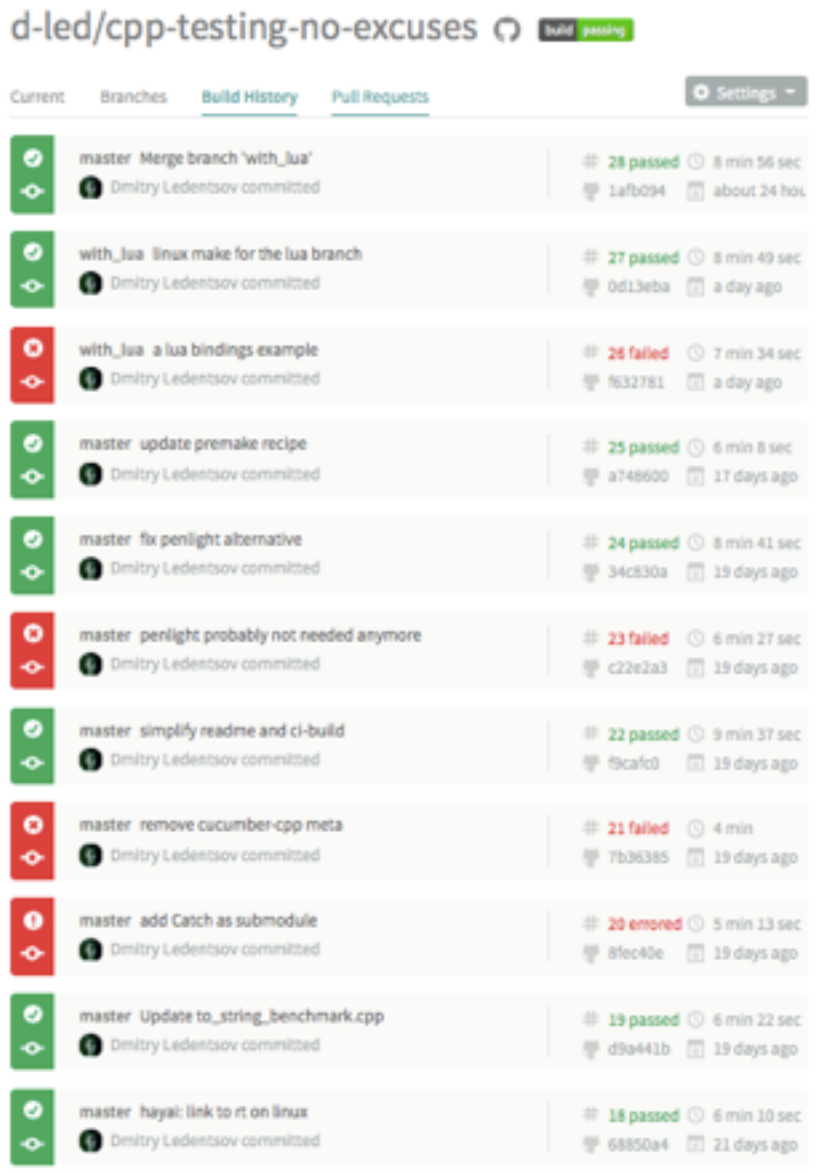
more reading

- Jeff Langr - Modern C++ Programming with Test-Driven Development: Code Better, Sleep Better
- Kent Beck - Test Driven Development: By Example
- Robert C. Martin - Clean Code: A Handbook of Agile Software Craftsmanship
- Jef Raskin - The Humane Interface: New Directions for Designing Interactive Systems
- Henrik Kniberg - * from the Trenches, Agile Product Ownership *
- Gerard Meszaros - http://xunitpatterns.com/
- Testable code via SOLID & FIRST principles
- Books by Tom DeMarco
- name yours…

code

build passing

https://github.com/d-led/cpp-testing-no-excuses



d-led/cpp-testing-no-excuses

master	Merge branch 'with_lua'	Dmitry Ledentsov committed	# 28 passed	8 min 56 sec
			1afb094	about 24 hou
with_lua	linux make for the lua branch	Dmitry Ledentsov committed	# 27 passed	8 min 49 sec
			0d13eba	a day ago
with_lua	a lua bindings example	Dmitry Ledentsov committed	# 26 failed	7 min 34 sec
			f632781	a day ago
master	update premake recipe	Dmitry Ledentsov committed	# 25 passed	6 min 8 sec
			a748600	17 days ago
master	fix penlight alternative	Dmitry Ledentsov committed	# 24 passed	8 min 41 sec
			34c830a	19 days ago
master	penlight probably not needed anymore	Dmitry Ledentsov committed	# 23 failed	6 min 27 sec
			c22e2a3	19 days ago
master	simplify readme and ci-build	Dmitry Ledentsov committed	# 22 passed	9 min 37 sec
			f9cafc0	19 days ago
master	remove cucumber-cpp meta	Dmitry Ledentsov committed	# 21 failed	4 min
			7b36385	19 days ago
master	add Catch as submodule	Dmitry Ledentsov committed	# 20 errored	5 min 13 sec
			8fec40e	19 days ago
master	Update to_string_benchmark.cpp	Dmitry Ledentsov committed	# 19 passed	6 min 22 sec
			d9a441b	19 days ago
master	hayai: link to rt on linux	Dmitry Ledentsov committed	# 18 passed	6 min 10 sec
			68850a4	21 days ago

thank you!

ledentsov.de
github.com/d-led

discussion welcome!

- What are you excuses? Why NOT test?
- What's your biggest pain?
- Would you like to present your topic?
- Your unique set-up?
- ...