Clean testing

Good practices in general coding

Alessandro Sciarra

Z02 – Software Development Center

XX August 2023

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

AxelKrypton

CRC-TR 211
Strong-interaction matter under extreme conditions

Prelude

Leave me a feedback

Was the talk clear, useful, inspiring?

Any general comment?

Drop me an email! → ✉ Alessandro

Disclaimer

Slides are quite full of text for later reading.

Most example are oversimplified to fit on a slide.

Tests shown before BDD/TDD sections are probably bad.

Let's discuss!

Outline

1. A short recap about clean code
2. Why (automated) testing?
3. Different types of testing
4. Tests goals – properties – good principles
5. Test driven development as discipline
6. What to do, now?

A short recap about clean code

What we explored in the clean code talk

- Take advantage of your IDE
- Use meaningful names
- Limit comments and care about formatting
- Function and classes should have a single responsibility (SRP)
- Integration Operation Separation Principle (IOSP)
- Don't repeat yourself (DRY)
- Keep it simple, stupid (KISS)
- Beware of optimisation
- The boyscout rule
- Keep improving

Why (automated) testing?

Lots of software should rather work...

In our society you cannot spend 60 seconds without interacting with a software system

1. In modern car there is software that controls the steering wheel...

2. Think of medical devices, aeroplanes, lifts, etc.
3. Aren't you upset when an app on your phone stalls or crashes?!

Lots of software should rather work...

In our society you cannot spend 60 seconds without interacting with a software system

1. In modern car there is software that controls the steering wheel...

2. Think of medical devices, aeroplanes, lifts, etc.
3. Aren't you upset when an app on your phone stalls or crashes?!

An remarkable story about manual testing

I have been using an own manually tested "agenda-script" to track my working time **for months** and then one day it suddenly crashed. Where was the problem?

Lots of software should rather work...

In our society you cannot spend 60 seconds without interacting with a software system

1. In modern car there is software that controls the steering wheel...

2. Think of medical devices, aeroplanes, lifts, etc.
3. Aren't you upset when an app on your phone stalls or crashes?!

An remarkable story about manual testing

I have been using an own manually tested "agenda-script" to track my working time **for months** and then one day it suddenly crashed. Where was the problem? I assumed **every** day has 24 hours... 😇

The unfortunate tendency in many cases

- It gives reasonable results, it works! 😱
- Manual (and possibly not systematic) testing is unfortunately still common practice in our community!
- Even if done in a systematic way, you will never run your tests enough!

Theorem:

Prove that, with $n, x, y, z \in \mathbb{N} > 0$ and $n > 2$, the equation $x^n + y^n = z^n$ has no solutions.

Would you accept this as proof or even argument?

$2^3 + 3^3 \neq 4^3$ and $1^4 + 5^4 \neq 6^4$ and $4^5 + 2^5 \neq 5^5$

The unfortunate tendency in many cases

- It gives reasonable results, it works! 😱
- Manual (and possibly not systematic) testing is unfortunately still common practice in our community!
- Even if done in a systematic way, you will never run your tests enough!

Now, software testing is not like mathematics...

...but would you accept a physics fact **without rigorous evidence**?

Consider one of your programs. Does it work?

How do you prove that a software works?

Consider one of your programs. Does it work?

How do you prove that a software works?

You don't. You try proving it doesn't work... and fail!

Consider one of your programs. Does it work?

How do you prove that a software works?

You don't. You try proving it doesn't work... and fail!

The scientific method

1. Make an hypothesis C $\longrightarrow$ «My code is correct»
2. Tests attempt to show $!C$
3. Confidence in C tracks thoroughness of tests

Testing code is science (cf. Popper's falsification principle)

Uncle Bob's magic button

Uncle Bob's magic button

Fearless competence

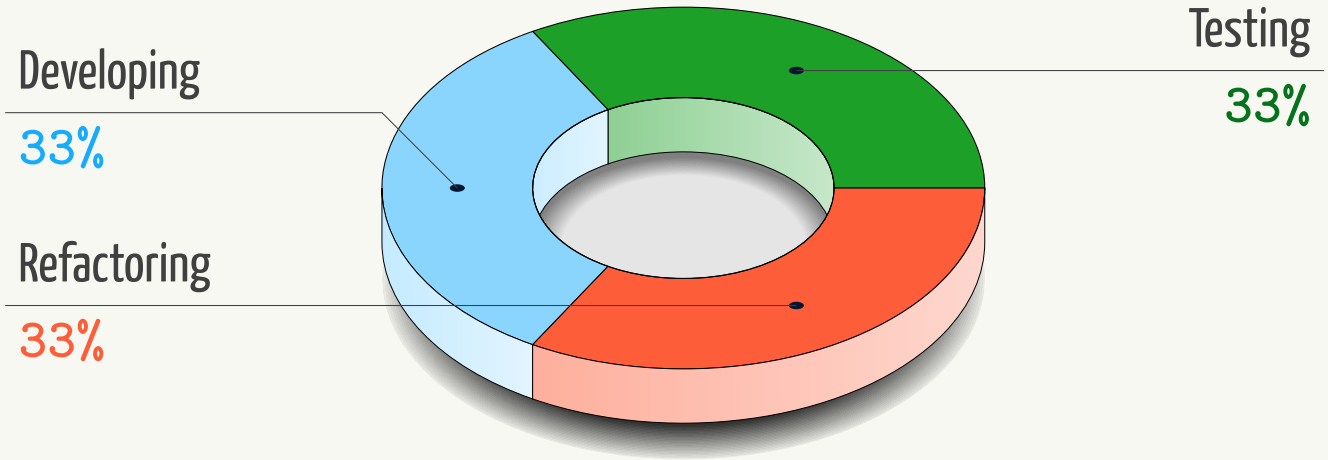
Are you afraid to touch the code? Are you afraid to improve the code? Do you have that little echo in your ear saying – Ah, if it ain't broken, don't fix it? I expect you to improve the code, all of the time, fearless competence. How do you get fearless competence?

Tests! What if I had a button, a little button I could push on my keyboard and some lights would blink and science-fiction sounds would come out of my laptop for a few minutes and then a little green light would light up and that green light told me that the system worked. And I believed it.

Robert C. Martin

The correct mental approach to testing

- Testing is part of the natural evolution of the code (not an overhead)
- Test code has to fulfil the same clean code rules as production code
- Tests are the key to be free to change (improve) the production code!
- Keep balance between adding new code, writing tests and refactoring

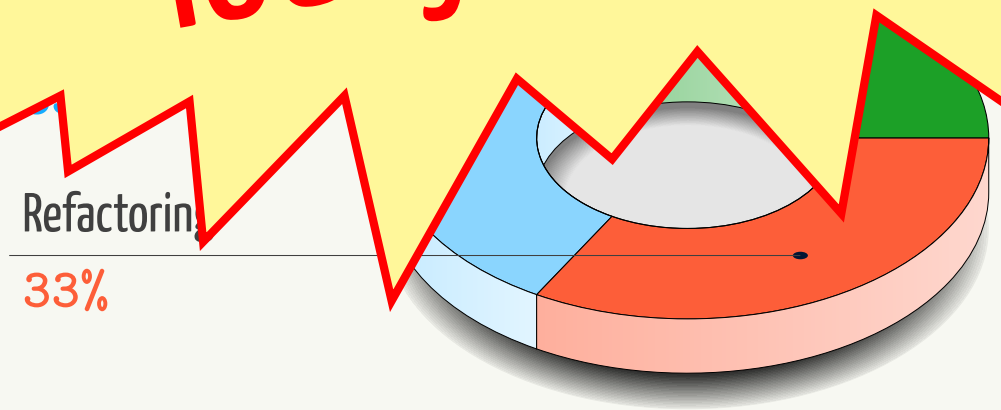


Slide taken from the Clean Code talk.

The correct mental approach to testing

- Testing is part of the natural evolution of the code (not an overhead)
- Test code has to fulfil the same clean c e rules as production ode
- Tests are th ey to be free t ro de!
- Keep b ance ng

Today much more!



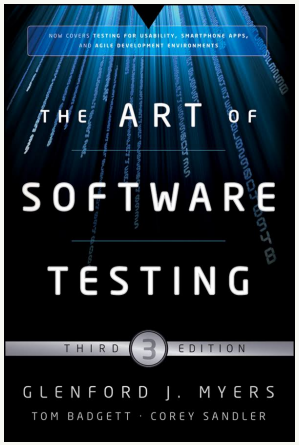
Slide taken from the Clean Code talk.

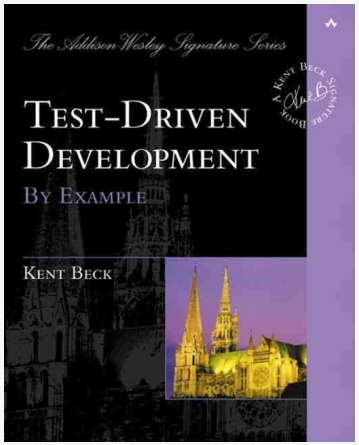
Different types of testing

Software testing

Test code is just as important as production code

- Testing levels
  - Unit testing
  - Integration testing
  - System testing
  - Acceptance testing





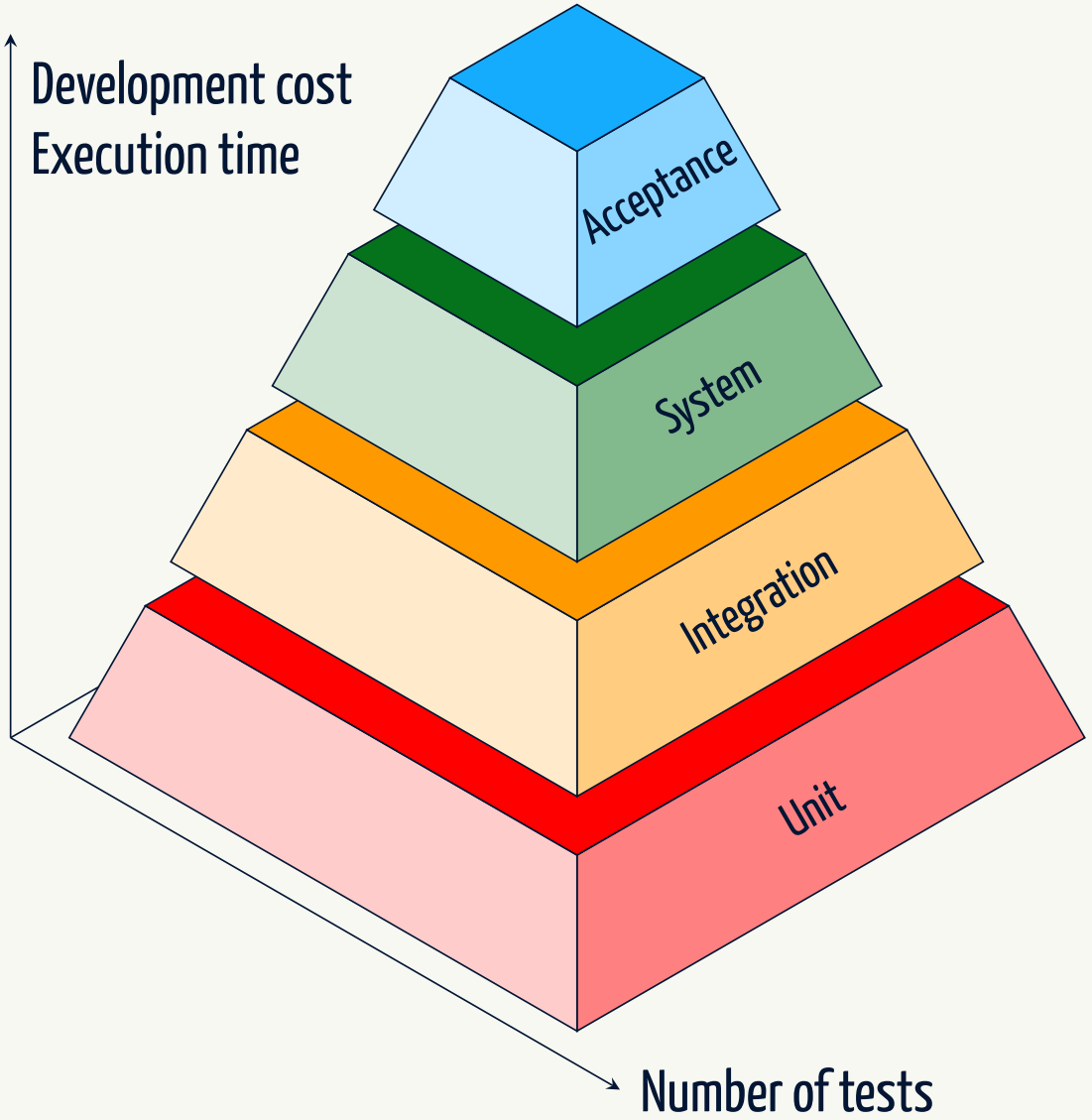
- Testing types and techniques
  - Manual testing
  - Automated testing
  - Continuous testing
  - Functional testing
  - Mutation testing
  - Fuzzy testing
  - …

- Plenty of videos/resources on the web

🔗 CppCon2015 – All your tests are terrible…
🔗 CppCon2020 – The science of unit tests
🔗 CppCon2020 – Back to Basics: Unit Tests
🔗 TDD, Where Did It All Go Wrong
🔗 TDD for those who don't need it
…

The videos linked here above (as well as the references therein) were mainly used to prepare this talk.

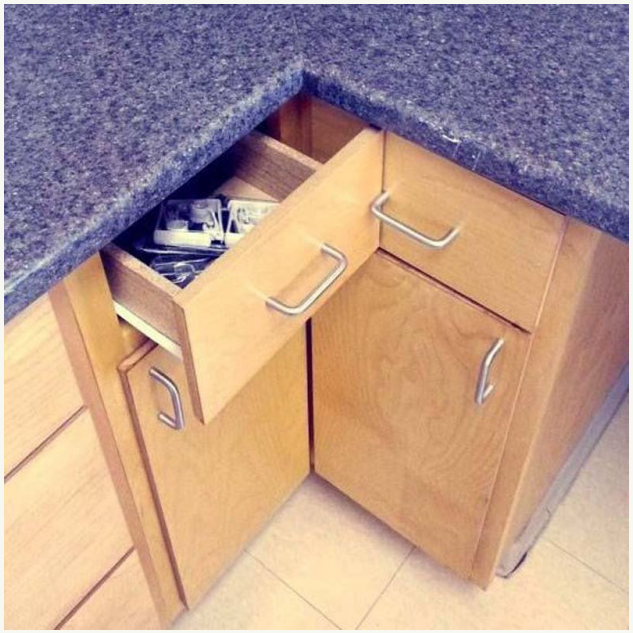
The tests pyramid



Unit tests are not enough!

Integration tests

Once you are sure that all components work correctly alone, be sure they work as expected together!

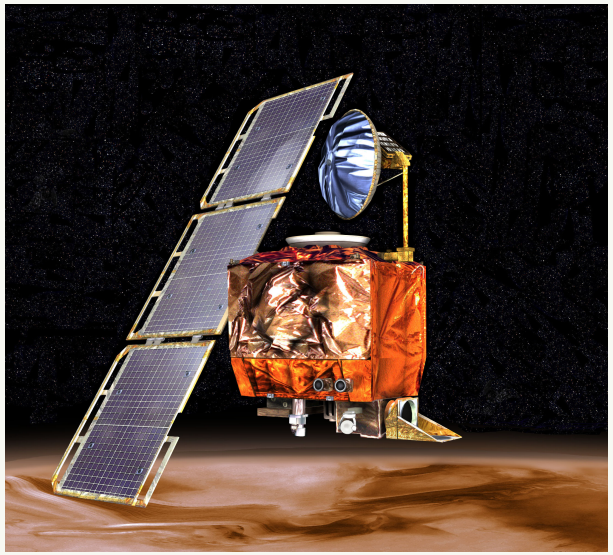


I already mentioned this in the Clean code talk.

Unit tests are not enough!

Integration tests

Once you are sure that all components work correctly alone, be sure they work as expected together!



Mars Climate Orbiter lost in 1999 – By NASA/JPL/Corby Waste – © Public Domain

I already mentioned this in the Clean code talk.

And once integration tests pass?

System testing

System testing tests a completely integrated system to verify that the system meets its requirements.

Wikipedia

Acceptance testing

Formal testing with respect to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether to accept the system.

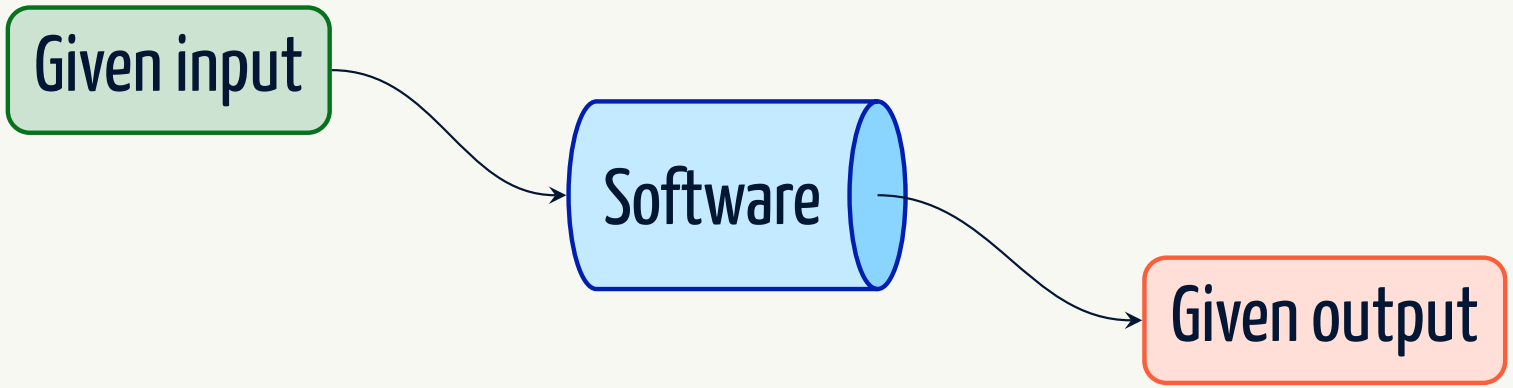
Standard Glossary of Terms used in Software Testing

In smaller projects

You might not need all categories, but, if you drop any, do it consciously!

A technique for system/acceptance tests

> **Functional tests**
>
> They tend to answer the question of "can the user do this" or "does this particular feature work".
>
> Wikipedia



- Consider to implement these tests in an external script!
- This is a good starting point to work with legacy code without tests!

Tests goals – properties – good principles

General idea and overview

There is slightly different advice around, pick your favourite:

Write F.I.R.S.T tests first

Fast:	To be run frequently
Independent:	To make failures meaningful
Repeatable:	To be run everywhere with same outcome
Self-validating:	To be automatised
Timely:	To make production code testable

General idea and overview

There is slightly different advice around, pick your favourite:

Google's perspective

Readable:	Tests are correct by inspection
Correct:	Do not rely on known bugs and test real scenarios
Complete:	Test most edge cases
Documenting:	Demonstration of how the API works
Resilient:	Repeatable, independent, hard to break, hermetic, etc.

General idea and overview

There is one not-negotiable **requirement of tests**, though:

General idea and overview

There is one not-negotiable **requirement of tests**, though:

A small interlude: Unit test frameworks for C++

Essential for self-validation and automation!

1 Boost Test Library

Documentation

```
#define BOOST_TEST_MODULE My Test
#include <boost/test/included/unit_test.hpp>

BOOST_AUTO_TEST_CASE(first_test)
{
    int i = 1;
    BOOST_TEST(i);
    BOOST_TEST(i == 2);
}
```

```
Running 1 test case...
test_file.cpp(8): error: in "first_test": check i == 2 has
    ↪ failed [1 != 2]

*** 1 failure is detected in the test module "My Test"
```

A small interlude: Unit test frameworks for C++

Essential for self-validation and automation!

1. Boost Test Library — Documentation
2. GoogleTest — User's guide

```
#include <gtest/gtest.h>
#include "src/factorial.h"

TEST(FactorialTest, HandlesZeroInput) {
    EXPECT_EQ(factorial(0), 1);
}

TEST(FactorialTest, HandlesPositiveInput) {
    EXPECT_EQ(factorial(1), 1);
}

int main(int argc, char **argv) {
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

A small interlude: Unit test frameworks for C++

Essential for self-validation and automation!

1. Boost Test Library — Documentation
2. GoogleTest — User's guide
3. Vir's Unit Test Framework — GitHub repository

```
#include <vir/test.h>

TEST(test_name) {
    int test = 3;
    COMPARE(test, 2) << "more " << "details";
    //COMPARE(test, 2).on_failure("more ", "details");
}
```

```
FAIL: ┌ at tests/testfile.cpp:5 (0x40451f):
FAIL: │ test (3) == 2 (2) -> false more details
FAIL: └ test_name

Testing done. 0 tests passed. 1 tests failed.
```

A small interlude: Unit test frameworks for C++

Essential for self-validation and automation!

1. Boost Test Library — Documentation
2. GoogleTest — User's guide
3. Vir's Unit Test Framework — GitHub repository

...and so many mores! — List on Wikipedia

Not all have the same weight and functionality

Pick your favourite depending on your needs and project:

1. Natural choice if you already depend on Boost
2. More complex but also more powerful
3. Very simple and lightweight

If you use a different language, you can find at the Wikipedia link lists for most programming languages.

Having tests is not enough, you must trust them!

Few words about tests correctness

- Never rely on known bugs

Having tests passing is simple...

Having tests is not enough, you must trust them!

Few words about tests correctness

- Never rely on known bugs

Having tests passing is simple...

```
int cube(int x) {
    //TODO: To be implemented
    return 0;
}

TEST(cube_test) {
    VERIFY(0 == cube(2));
    VERIFY(0 == cube(3));
    VERIFY(0 == cube(42));
}
```

Having tests is not enough, you must trust them!

Few words about tests correctness

- Never rely on known bugs

...better to have them failing!

```
int cube(int x) {
    //TODO: To be implemented
    return 0;
}

TEST(cube_test) {
    VERIFY(8 == cube(2));
    VERIFY(27 == cube(3));
    VERIFY(74088 == cube(42));
}
```

Having tests is not enough, you must trust them!

Few words about tests correctness

- Never rely on known bugs
- Never mock the code that you are testing

This is probably useless

```
class MockCoffeeMachine : public CoffeeMachine {
    //For simplicity we assume always in temperature
    double warmUp() override {
        return constants::brewingT;
    }
}

BOOST_AUTO_TEST_CASE(warmUp_test) {
    MockCoffeeMachine machine{};
    BOOST_REQUIRE_CLOSE(machine.warmUp(),
                        constants::brewingT, 0.1);
}
```

Having tests is not enough, you must trust them!

Few words about tests correctness

- Never rely on known bugs
- Never mock the code that you are testing

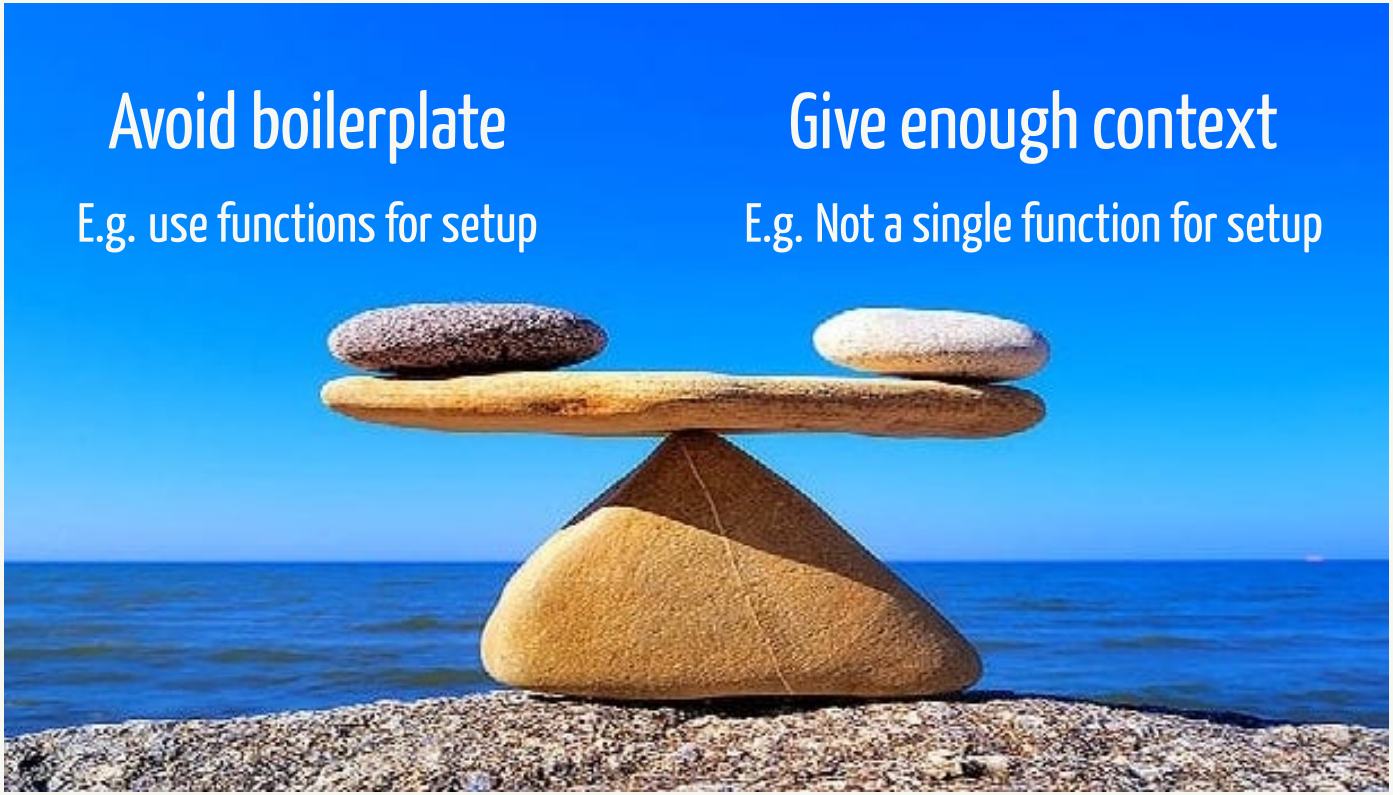
Think of a clear reaction procedure in your team

What should happen in your project when a test fails?
...and when a bug that was not caught by tests is found?

↓

Never miss any chance to improve your tests!

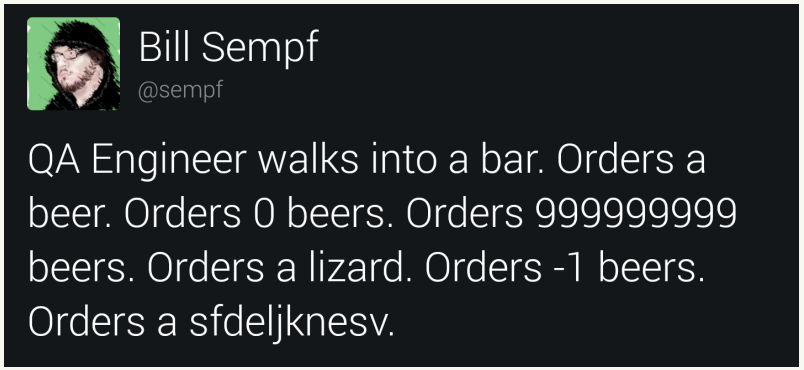
Tests readability



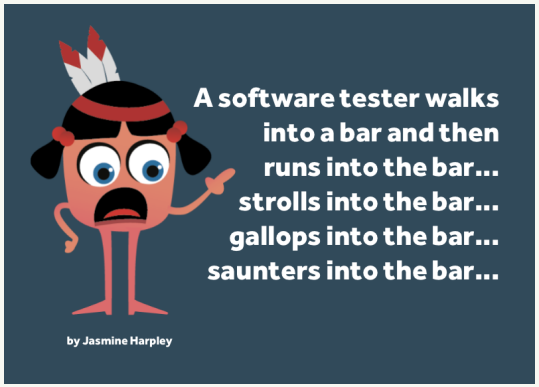
Remember: Tests are correct by inspection!

Tests completeness: What should I test and how?

- Values



- Actions



...and anything that makes sense!

I already mentioned this in the Clean code talk.

Tests completeness: What should I test and how?

A very nice example from Titus Winter

```
TEST(FactorialTest, BasicTests) {
    EXPECT_EQ(1, Factorial(1));
    EXPECT_EQ(120, Factorial(5));
}
```

Tests completeness: What should I test and how?

A very nice example from Titus Winter

```
int Factorial(int n) {
    if (n == 1) return 1;
    if (n == 5) return 120;
    return -1; // TODO(goofus): figure this out.
}

TEST(FactorialTest, BasicTests) {
    EXPECT_EQ(1, Factorial(1));
    EXPECT_EQ(120, Factorial(5));
}
```

Tests completeness: What should I test and how?

A very nice example from Titus Winter

```
int Factorial(int n) {
    if (n == 1) return 1;
    if (n == 5) return 120;
    return -1; // TODO(goofus): figure this out.
}

TEST(FactorialTest, BasicTests) {
    EXPECT_EQ(1, Factorial(1));
    EXPECT_EQ(120, Factorial(5));
    EXPECT_EQ(1, Factorial(0));
    EXPECT_EQ(479001600, Factorial(12));
    EXPECT_EQ(std::numeric_limits::max<int>() , Factorial(13));
    EXPECT_EQ(1, Factorial(0));
    EXPECT_EQ(std::numeric_limits::max<int>() , Factorial(-10));
}
```

This will pretty much naturally emerge when doing TDD

The hidden value of tests

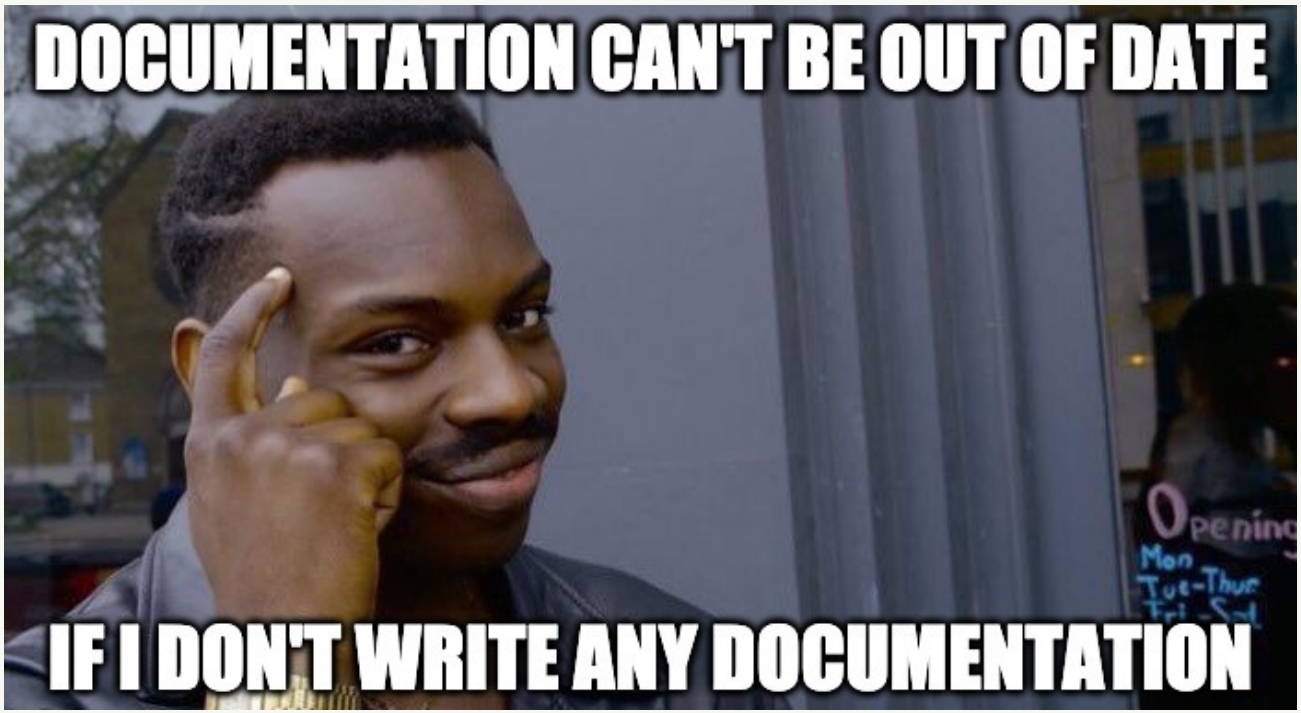
Few words about tests as API usage examples

Of course, a meme, do not take it seriously, but...

The hidden value of tests

Few words about tests as API usage examples

Of course, a meme, do not take it seriously, but...



...good tests should show how to use your code!

In this sense, it should be avoided to use implementation details in tests (cf. white box testing downsides).

Tests resilience: Google advice

- No flaky tests → Tests should always give the same outcome
- No brittle tests → One failure should not trigger many failures
- Tests reference values should not come from the system under test!
- Changing tests order should never change the outcome
- Tests should be as hermetic as possible → No I/O, no network, etc.
- No deep dependence → Avoid any implicit assumption

Let's see some examples!

This list and all GoogleTest examples are taken from Titus Winters' and Hyrum Wright's talk at CppCon2015.

Tests resilience: Google advice

A sneaky flaky test: What happens if the test-system is overloaded?

```
TEST(UpdaterTest, RunsFast) {
    Updater updater;
    updater.UpdateAsync();
    SleepFor(Seconds(.5)); // Half a second should be *plenty*.
    EXPECT_TRUE(updater.Updated());
}
```


This list and all GoogleTest examples are taken from Titus Winters' and Hyrum Wright's talk at CppCon2015.

Tests resilience: Google advice

A sneaky flaky test: What happens if the test-system is overloaded?

```
TEST(UpdaterTest, RunsFast) {
    Updater updater;
    updater.UpdateAsync();
    SleepFor(Seconds(.5)); // Half a second should be *plenty*.
    EXPECT_TRUE(updater.Updated());
}
```

An infamous brittle pattern

```
TEST(Logger, LogWasCalled) {
    StartLogCapture();
    EXPECT_TRUE(Frobber::Start());
    EXPECT_THAT(
      Logs(), Contains("file.cc:421: Opened file frobber.config" )
    );
}
```


This list and all GoogleTest examples are taken from Titus Winters' and Hyrum Wright's talk at CppCon2015.

Tests resilience: Google advice

Where are the reference values coming from?

```
BOOST_AUTO_TEST_CASE(Dirac_M)
{
    const hmc_float refs[4] =
        { 2610.3804893063798, 4356.332327032359,
          2614.2685771909237, 4364.1408252701831 };
    test_fermionmatrix<physics::fermionmatrix::Dslash>(refs);
}
```


This list and all GoogleTest examples are taken from Titus Winters' and Hyrum Wright's talk at CppCon2015.

Tests resilience: Google advice

Where are the reference values coming from?

```
BOOST_AUTO_TEST_CASE(Dirac_M)
{
    const hmc_float refs[4] =
        { 2610.3804893063798, 4356.332327032359,
          2614.2685771909237, 4364.1408252701831 };
    test_fermionmatrix<physics::fermionmatrix::Dslash>(refs);
}
```

A non-hermetic test: What if run twice in parallel?

```
TEST(Server, StorageTest) {
    StorageServer* server = GetStorageServerHandle();
    auto my_val = rand();
    server->Store("testkey", my_val);
    EXPECT_EQ(my_val, server->Load("testkey"));
}
```


This list and all GoogleTest examples are taken from Titus Winters' and Hyrum Wright's talk at CppCon2015.

Tests resilience: Google advice

Deep dependence example

```
class File {
  public:
    // ...
    virtual bool StatWithOptions(Stat_t* stat, StatOptions opts) {
        return this->Stat(stat); // In base class ignore options
    }
};
TEST(Filesystem, FSUsage) {
    // ... and call to StatWithOptions
    EXPECT_CALL(file, Stat(_)).Times(1);
}  // This relies on a call to base StatWithOptions!!
```

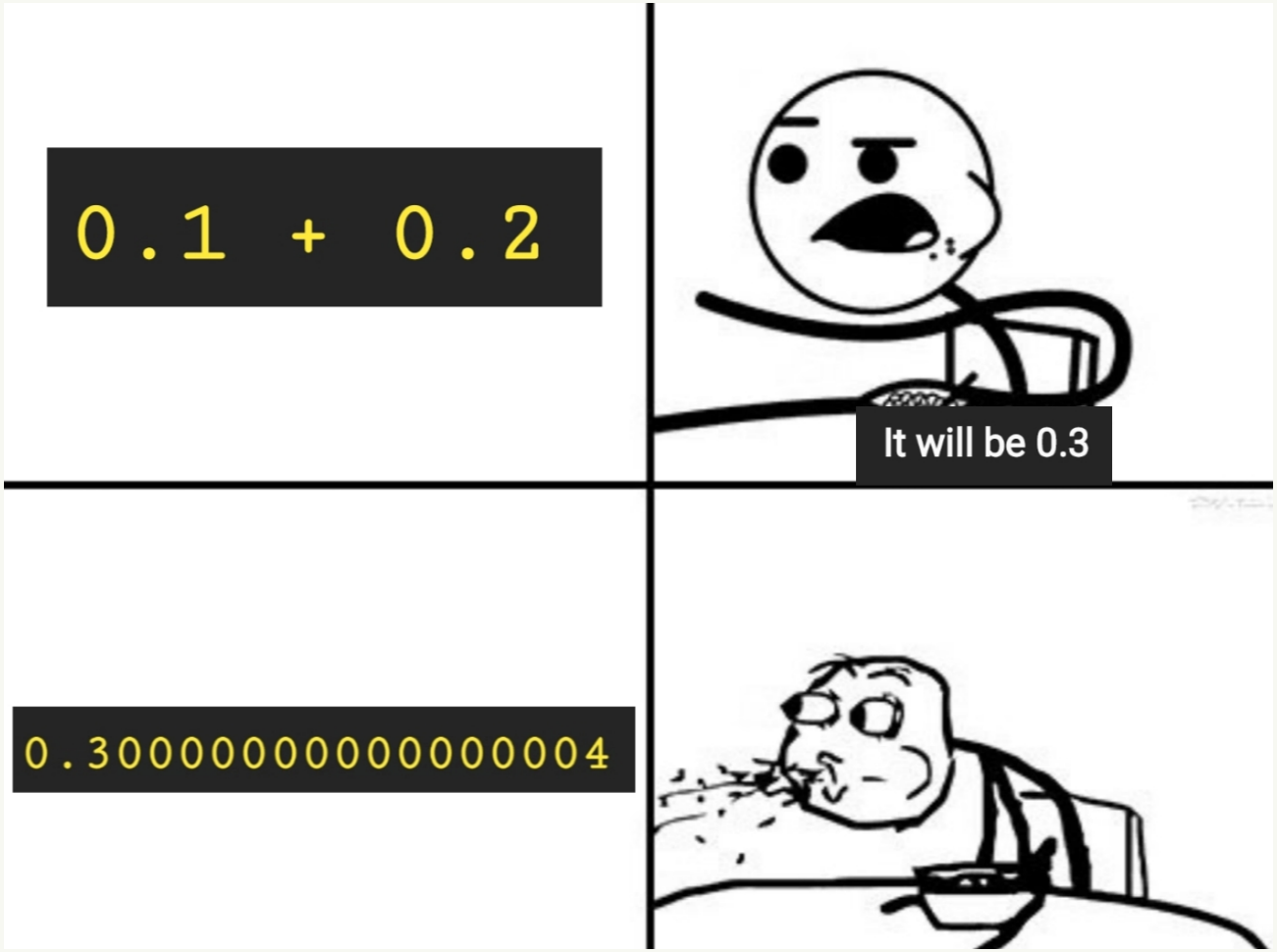
The law of implicit interfaces (AKA Hyrum's law)

Given enough users of your public interface, soon or later someone will start implicitly relying on your implementation details (speed, memory consumption, etc.).

Hyrum Wright

This list and all GoogleTest examples are taken from Titus Winters' and Hyrum Wright's talk at CppCon2015.

A slide about precision



A slide about precision

- We all know that machine precision is finite!
- Comparing floating point numbers has to be done carefully
- If you know a reference value exactly, put in the code more than the needed digits $\longrightarrow \pi = 3.14159265358979323846264 3$
- Be aware of possible compiler techniques, e.g. FMA { fused multuply-add }
  ↳ This could make your test fail if you require too high precision
- Requiring too high precision makes tests brittle
- Requiring too low precision makes tests useless
- After all it is a judgement call!

Few more words about reproducibility

Tests accuracy and reacting to failures

Test driven development as discipline

What to do, now?