Accelerated TDD

@phil_nash

Things

Please keep your **camera *on*** as much as possible

Keep **mics *off*** when you are not talking

Breaks every 60-90 minutes

Will do exercises - including group work

Who are we?

https://miro.com/app/board/o9J_km-wEi0=/

Accelerated TDD

@phil_nash

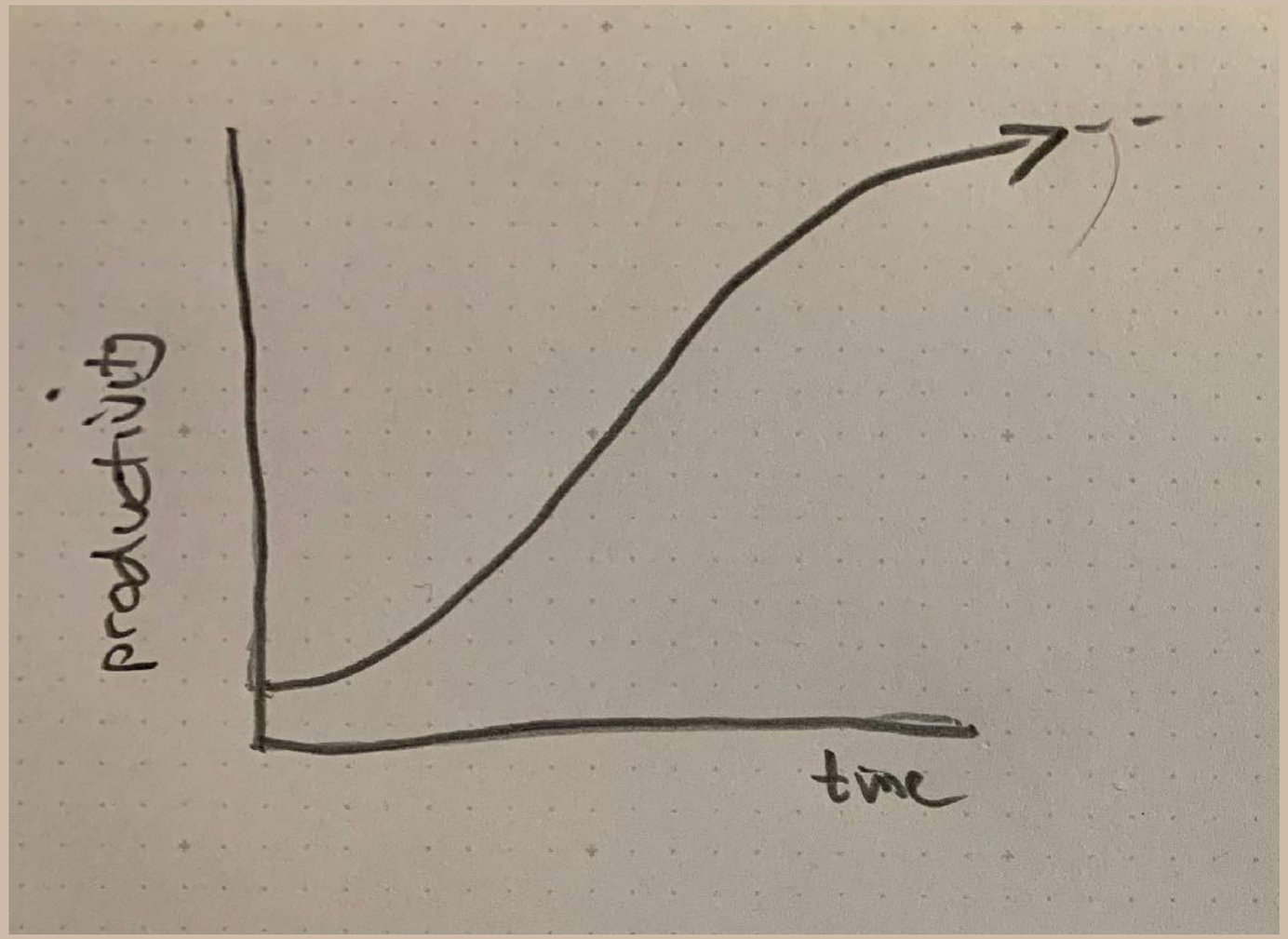
Why Accelerated?

Why Accelerated?

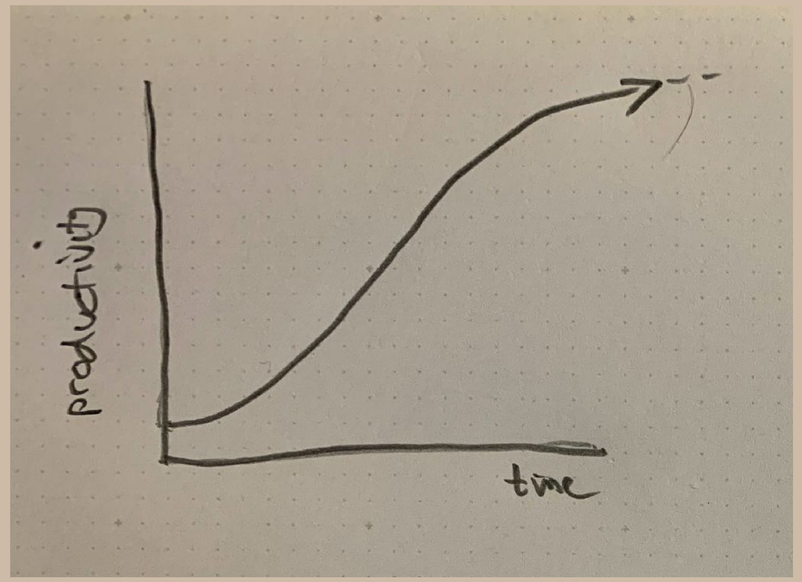
Many dead-ends - learn from hard experience

Can take while to become productive

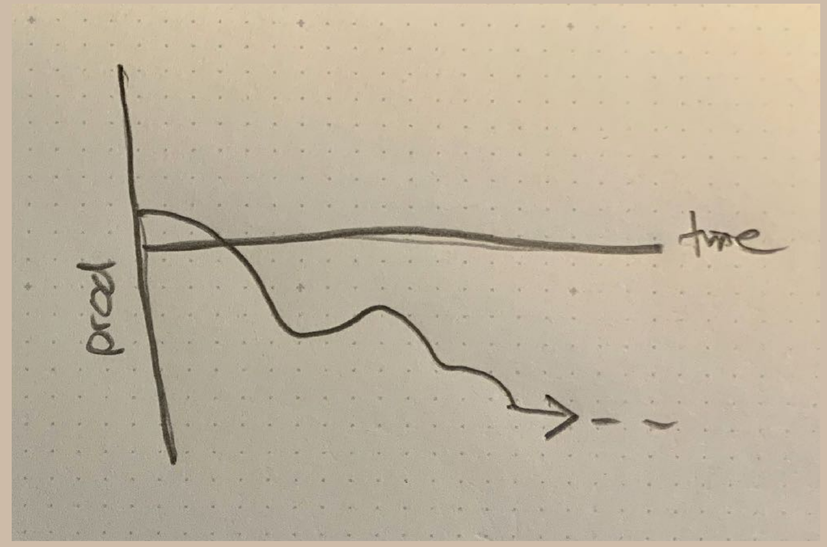
Many give up on TDD too soon



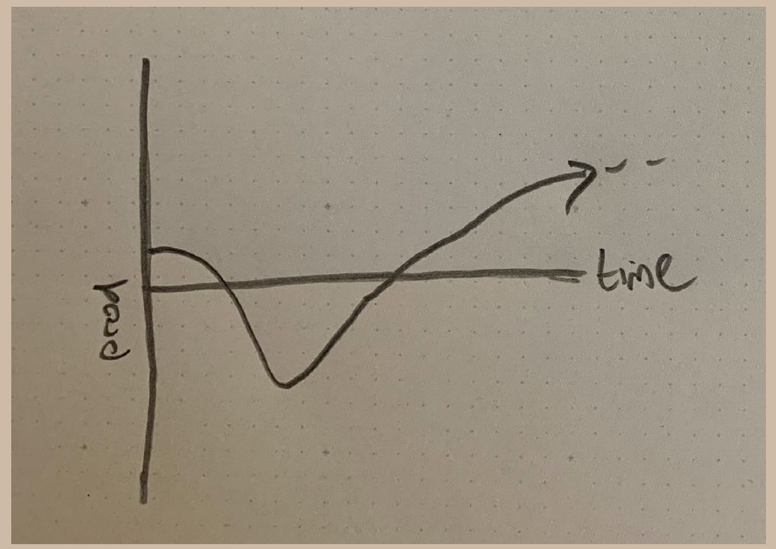
How TDD is sold



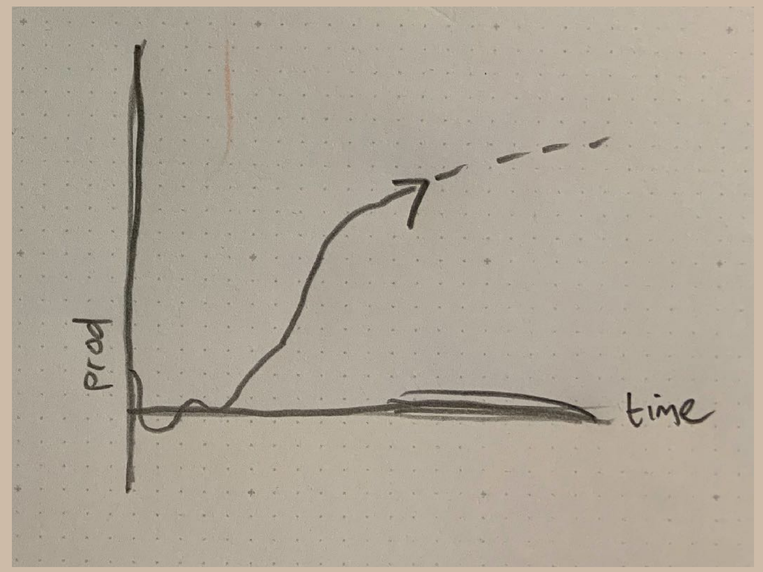
How TDD is sold



What most experience



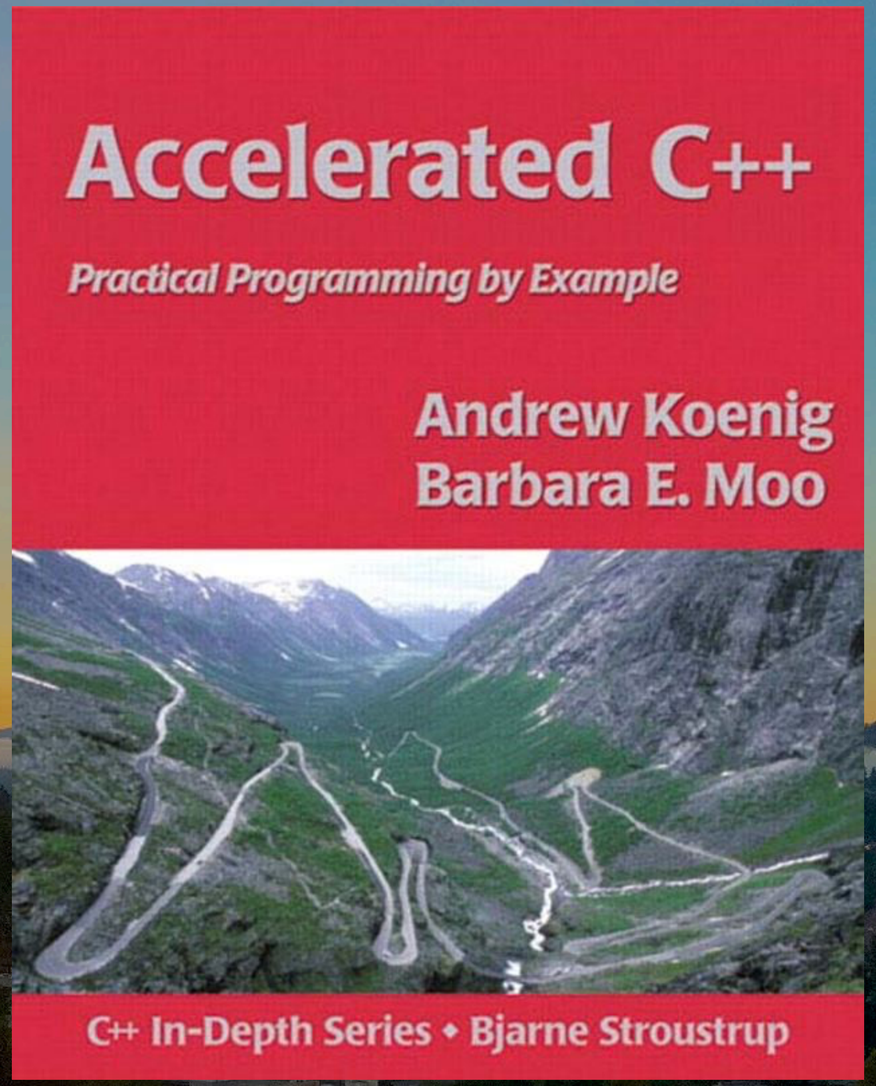
… if you persist



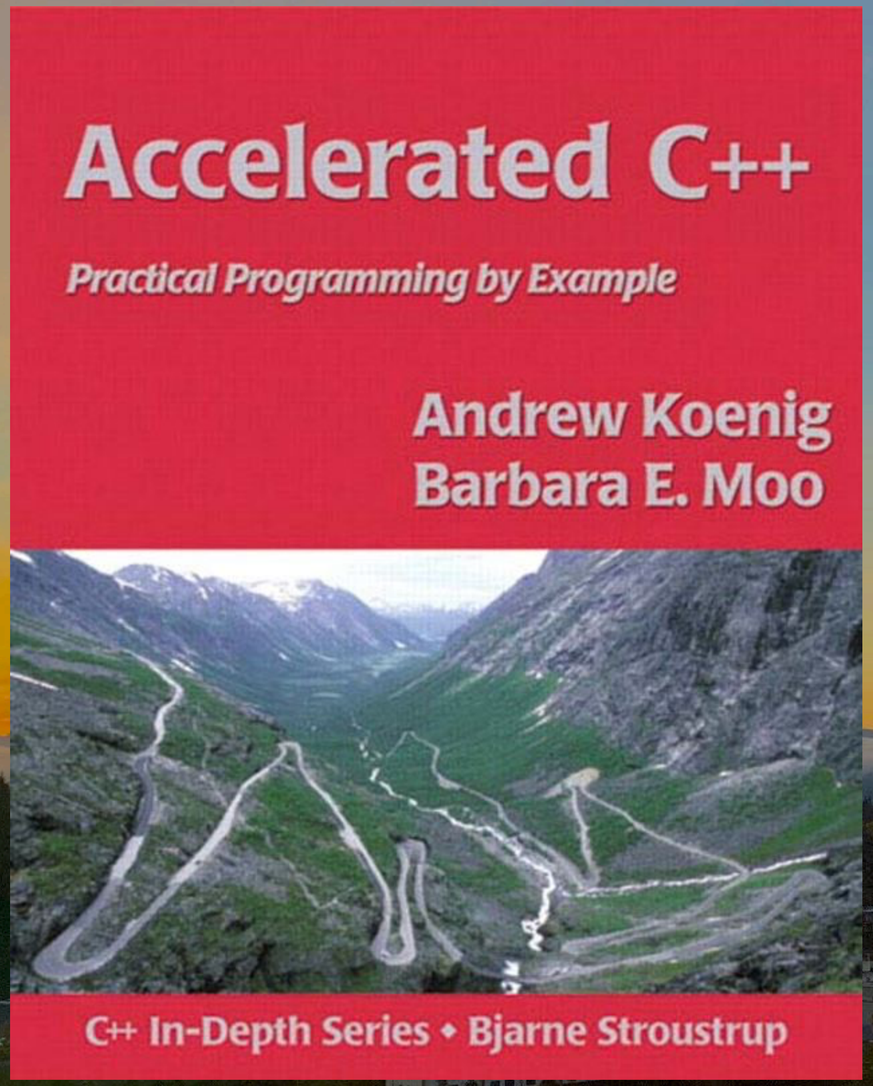
What we hope to do

How Accelerated?

How Accelerated?



How Accelerated?



Stop teaching ~~C++~~ Unit Testing

Practical Examples, early

Start high level

Good information

What is TDD?

Test Driven Development

Test Driven Design

X → Driven

↗ Design

↘ Development

X → Driven → Design
↘ Development

Test

Type

Domain

Feature

Behaviour

Data

Driven

dictionary.com

adjective

2 being under compulsion, as to succeed or excel:
a driven young man who was fiercely competitive.

3 controlled or propelled by something specified (used in combination):
*a **market-driven** approach to retaining talent; **data-driven** business strategies; a water pump that is solar driven.*

X → Driven → Design
↘ Development

Test

Type

Domain

Feature

Behaviour

Data

Questions?

The CO$T of bugs

The CO$T of *bugs*

The CO$T of *bugs*

2002 NIST study

The CO$T of *bugs*

2002 NIST study

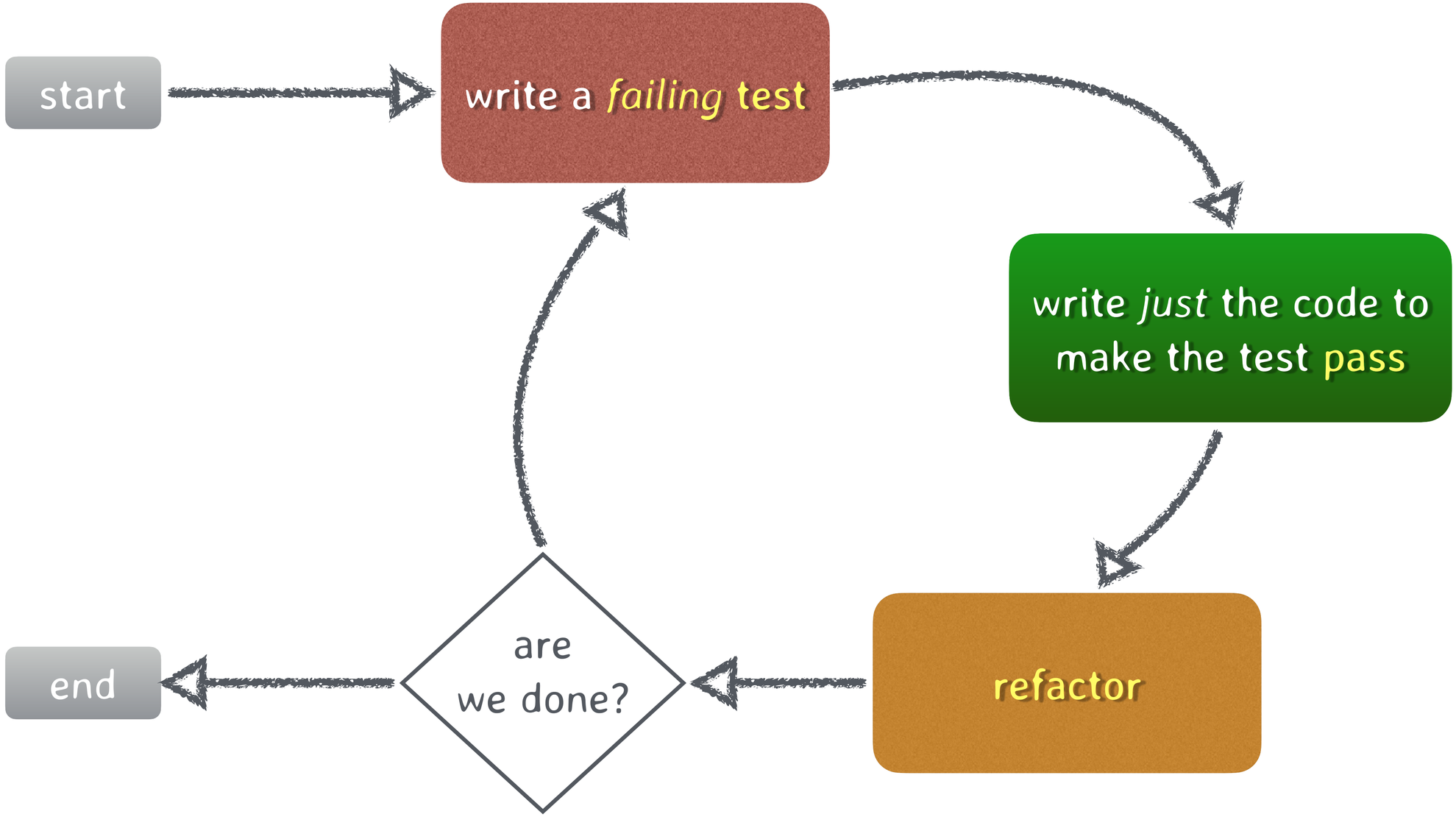
"Software bugs are costing the U.S. economy an estimated $59.5 billion each year"

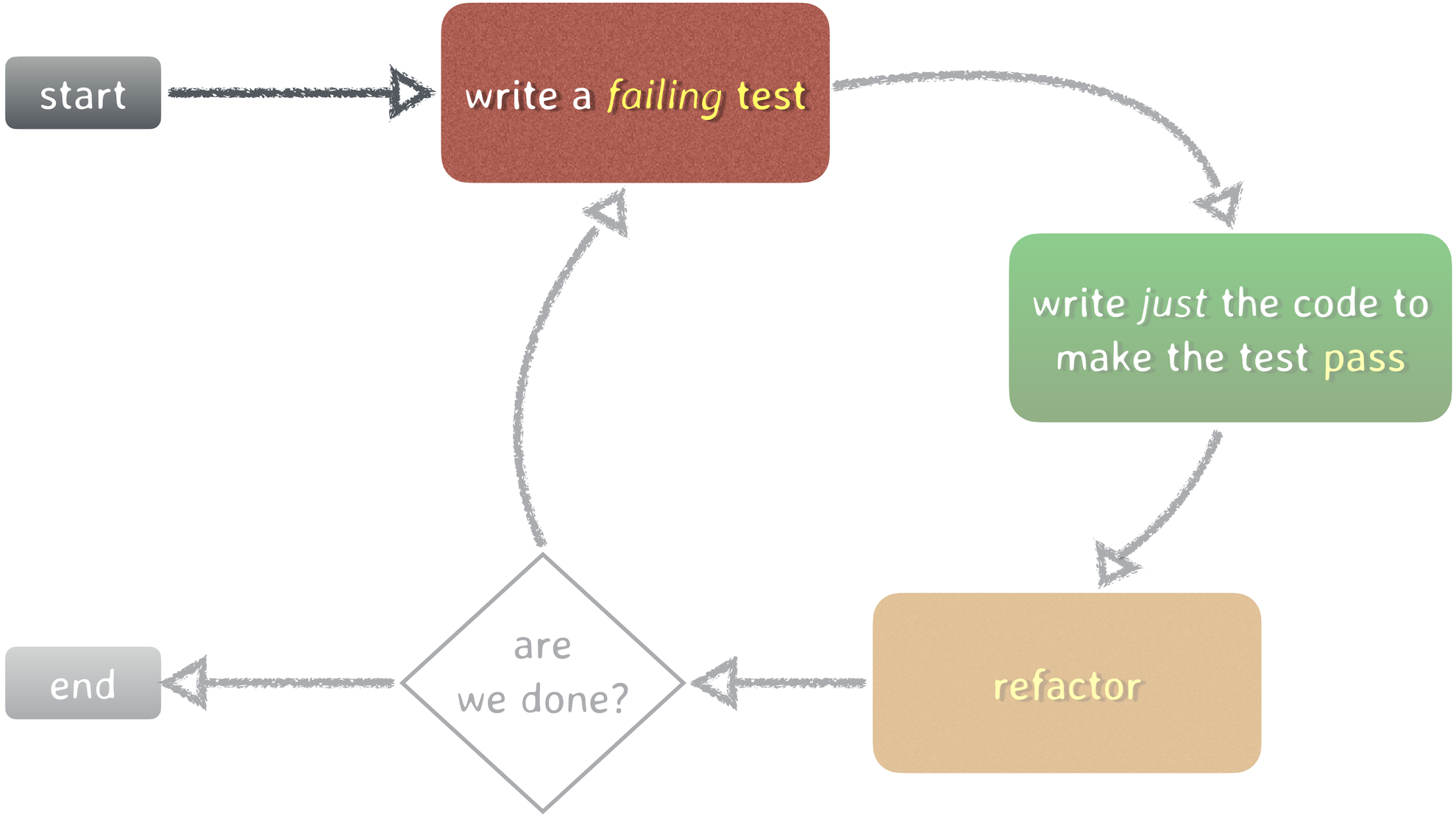
"Improvements in testing could reduce this cost by about a third, or $22.5 billion

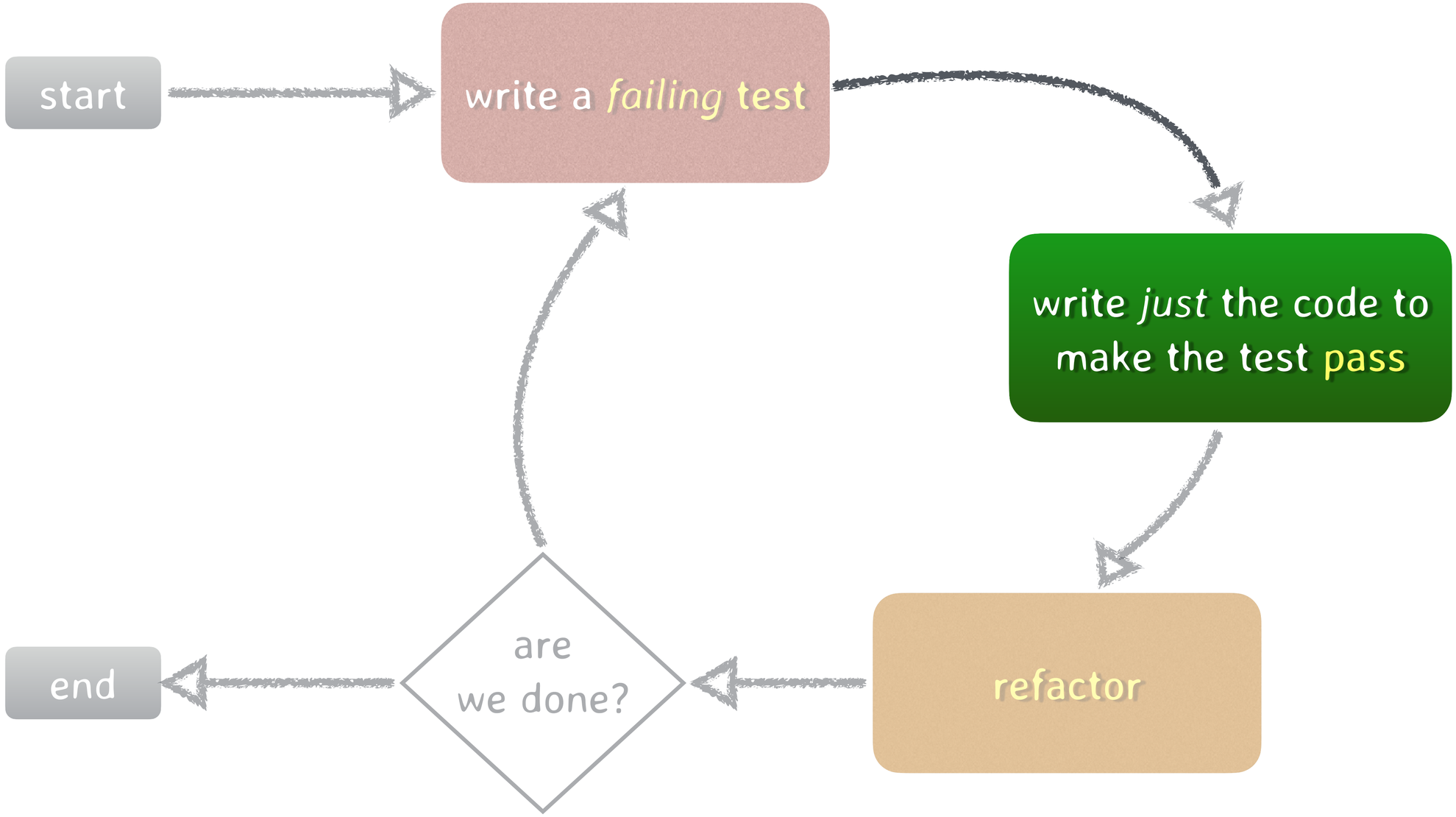
"The general consensus seems to be [that] the *current state of the art with respect to testing* is poor and can be sufficiently improved"

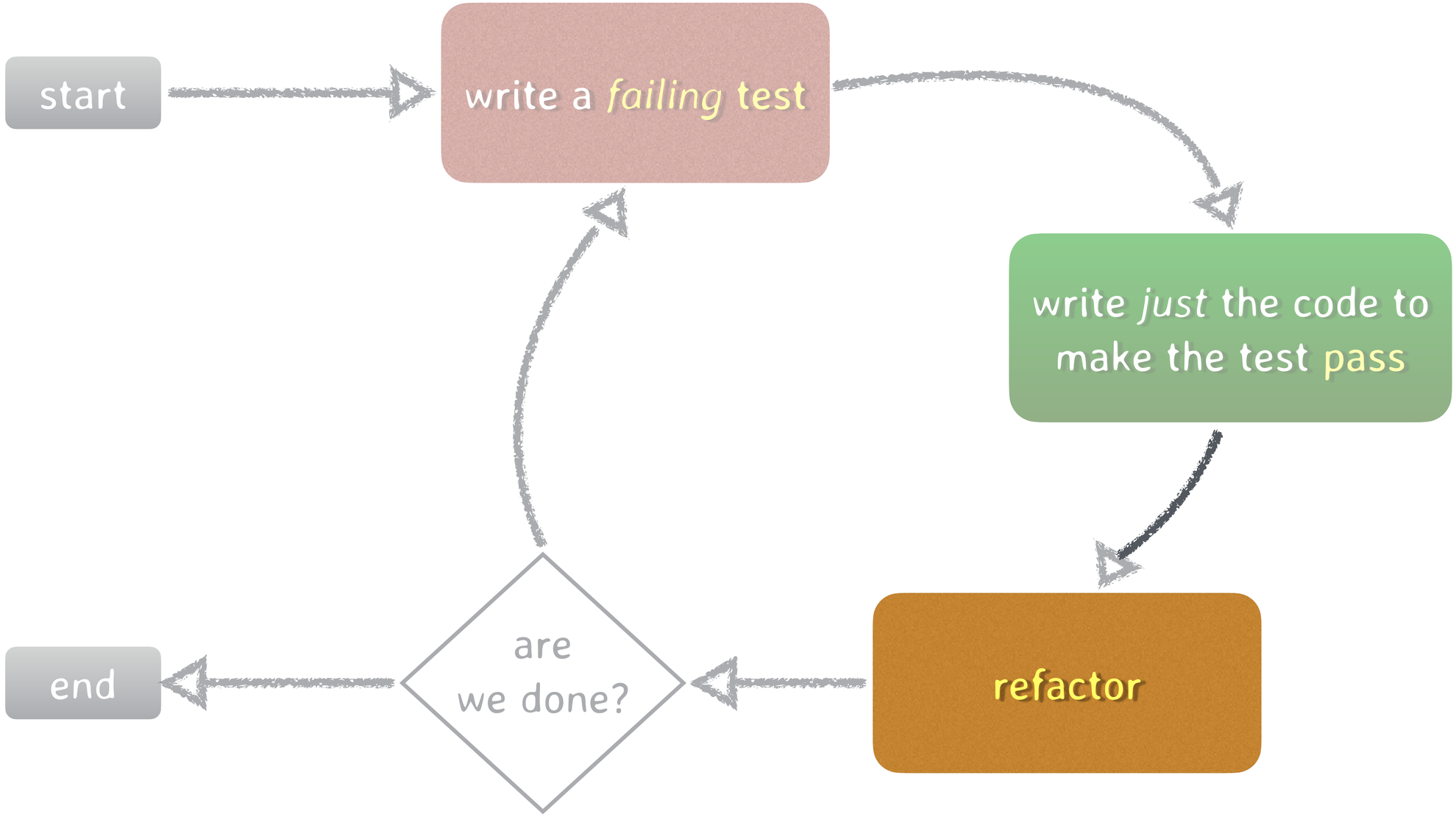
TDD IS HARD
LET'S GO SHOPPING (AND WRACK UP TECHNICAL DEBT)
imgflip.com

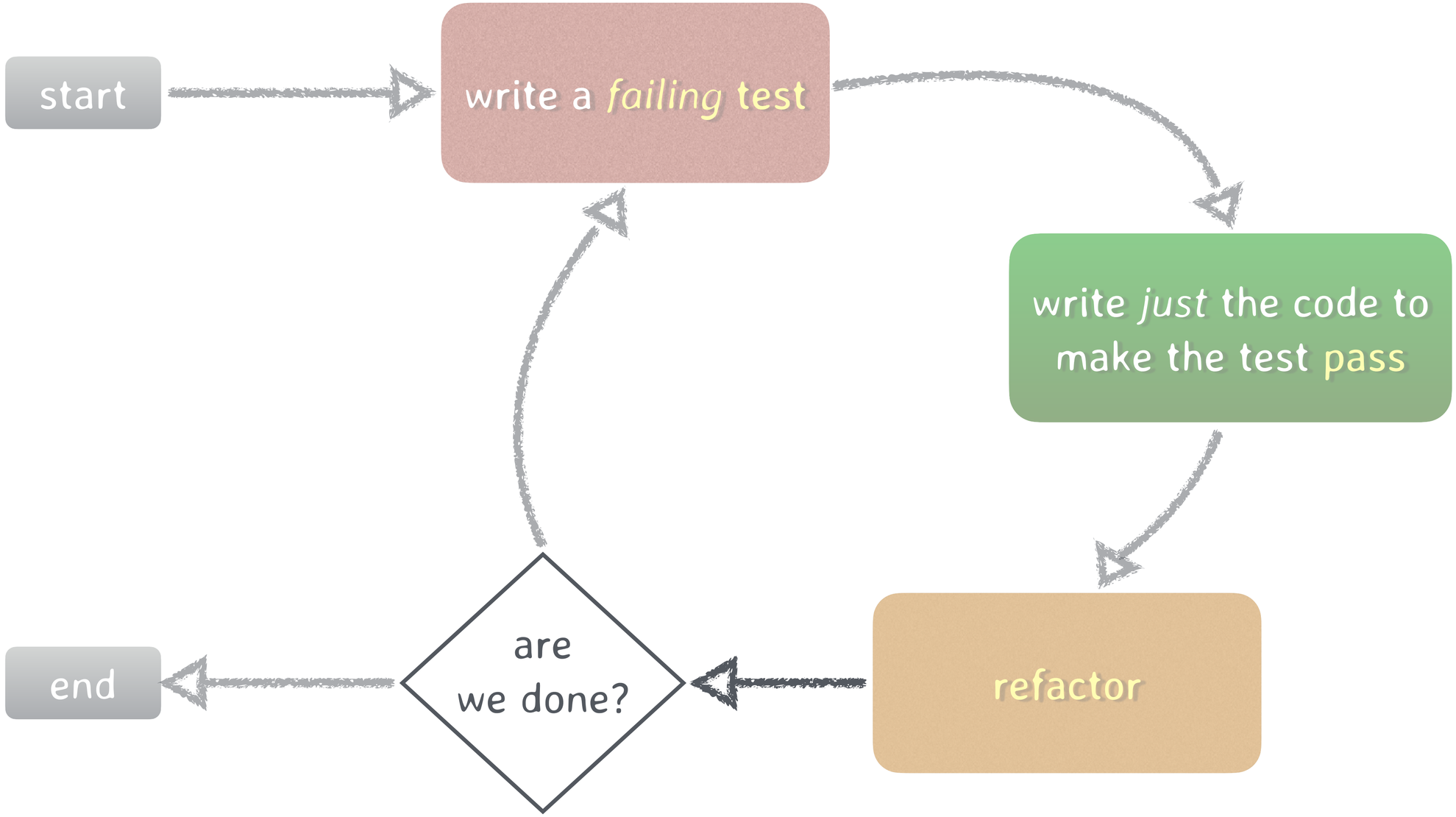
The TDD Cycle

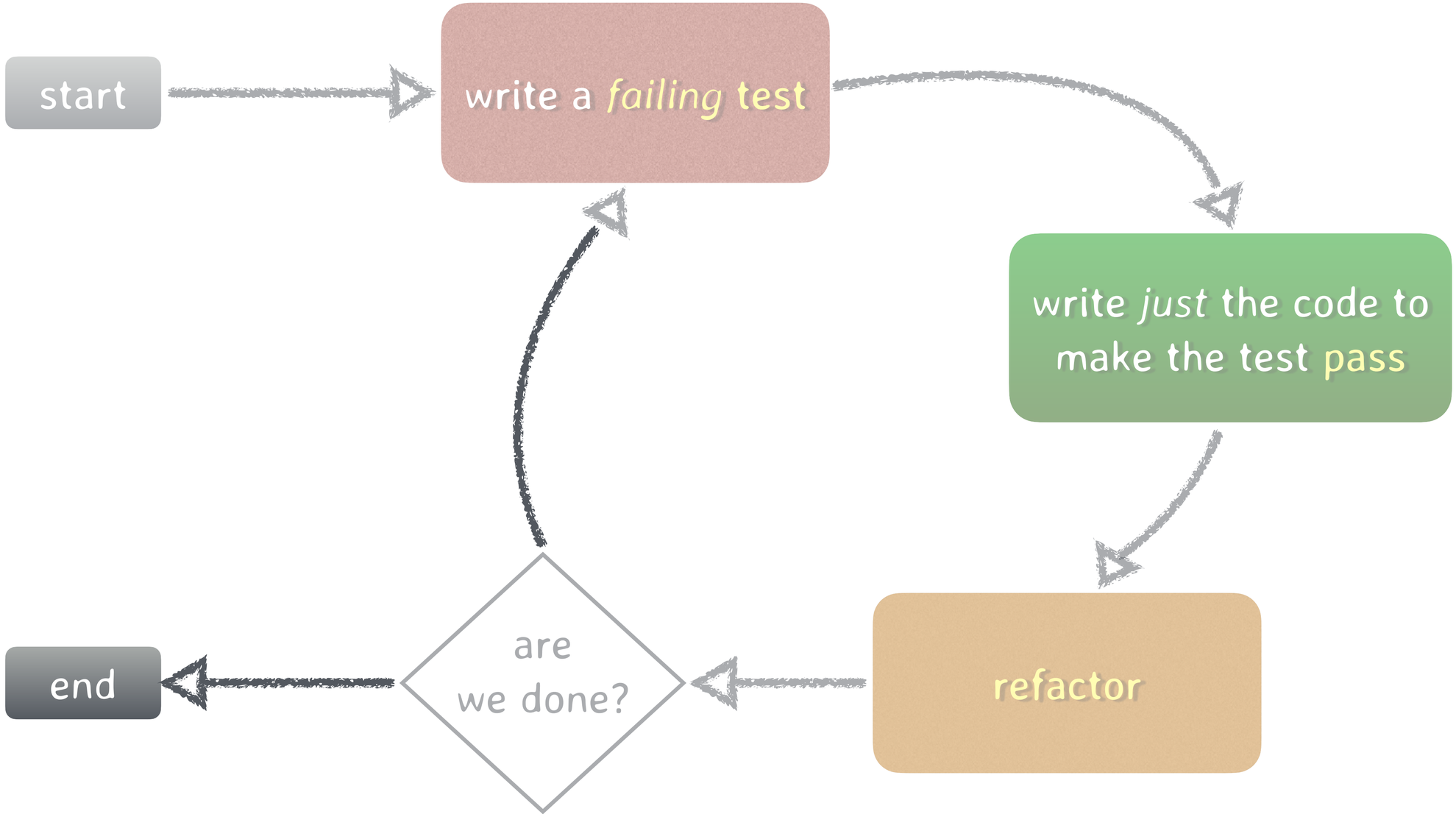

start
write a failing test
write just the code to make the test pass
refactor
are we done?
end

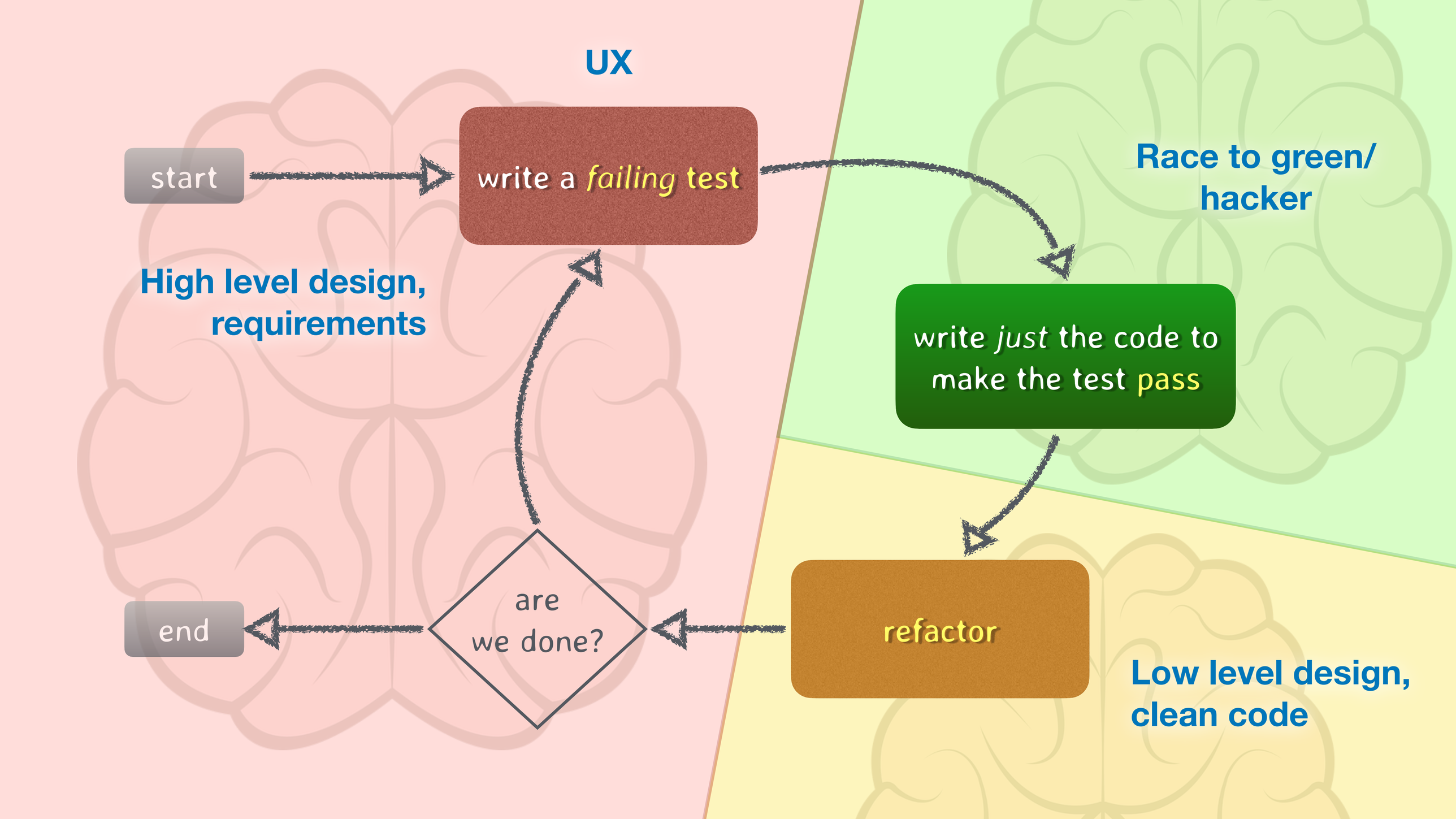

start
write a failing test
write just the code to make the test pass
refactor
are we done?
end


start
write a failing test
write just the code to make the test pass
refactor
are we done?
end


start
write a failing test
write just the code to make the test pass
refactor
are we done?
end


start
write a failing test
write just the code to make the test pass
refactor
are we done?
end


start
write a failing test
write just the code to make the test pass
refactor
are we done?
end


UX
start
write a failing test
Race to green/
hacker
High level design,
requirements
write just the code to
make the test pass
are
we done?
end
refactor
Low level design,
clean code

QUESTIONS?

Why is
TDD hard?

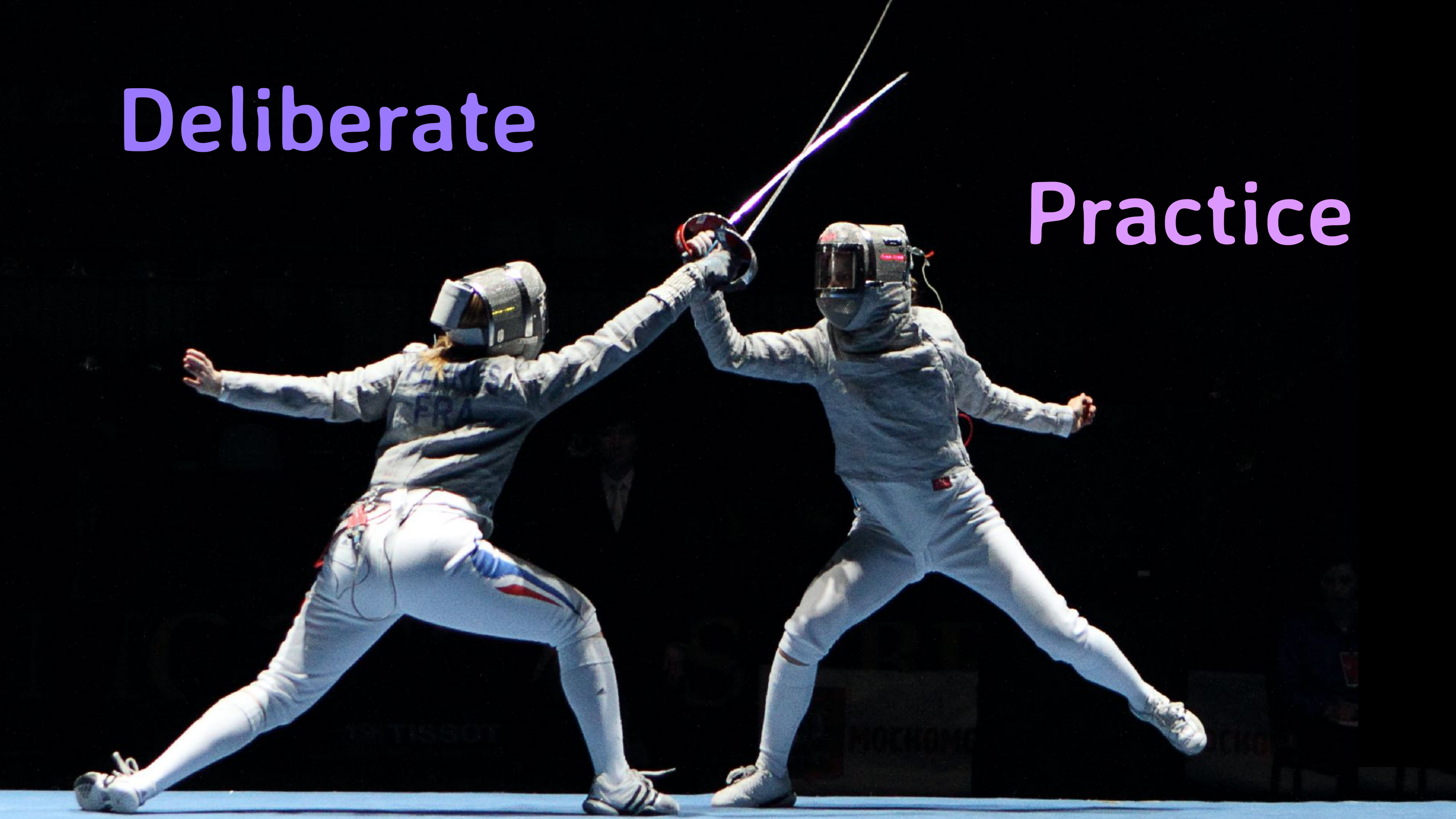
The trouble with TDD

The trouble with TDD

Hard to learn the *art*
- requires experience

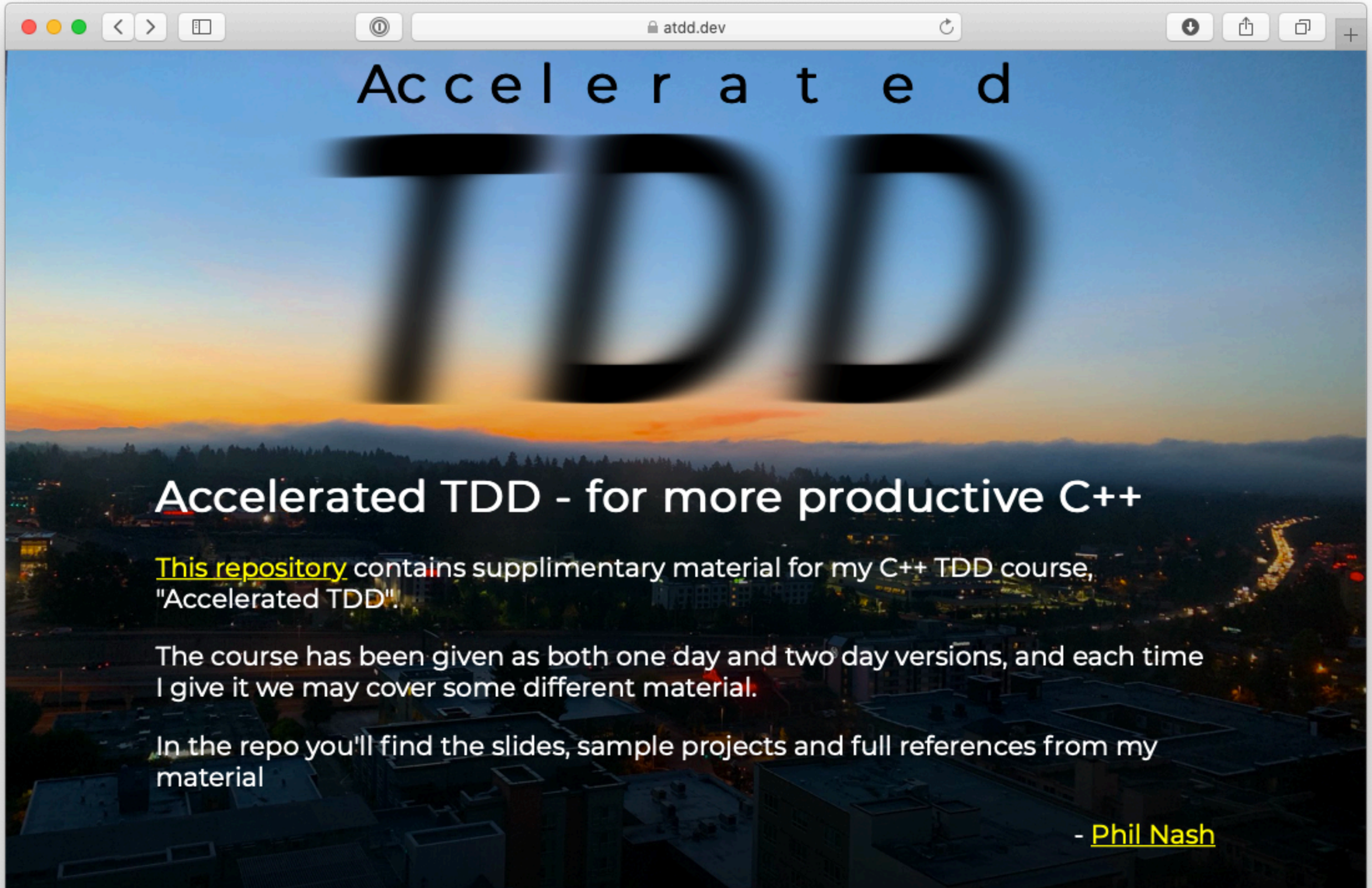
Productivity goes down before it goes up

Practicing on the job

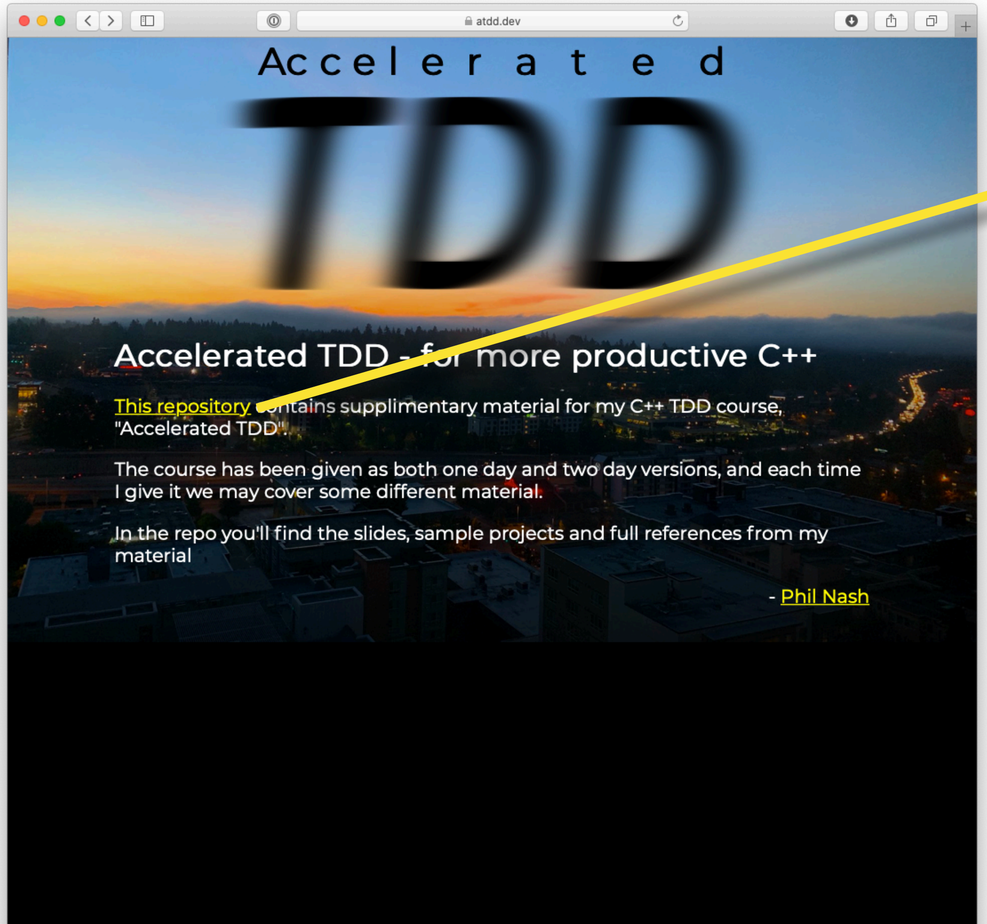

Deliberate
Practice

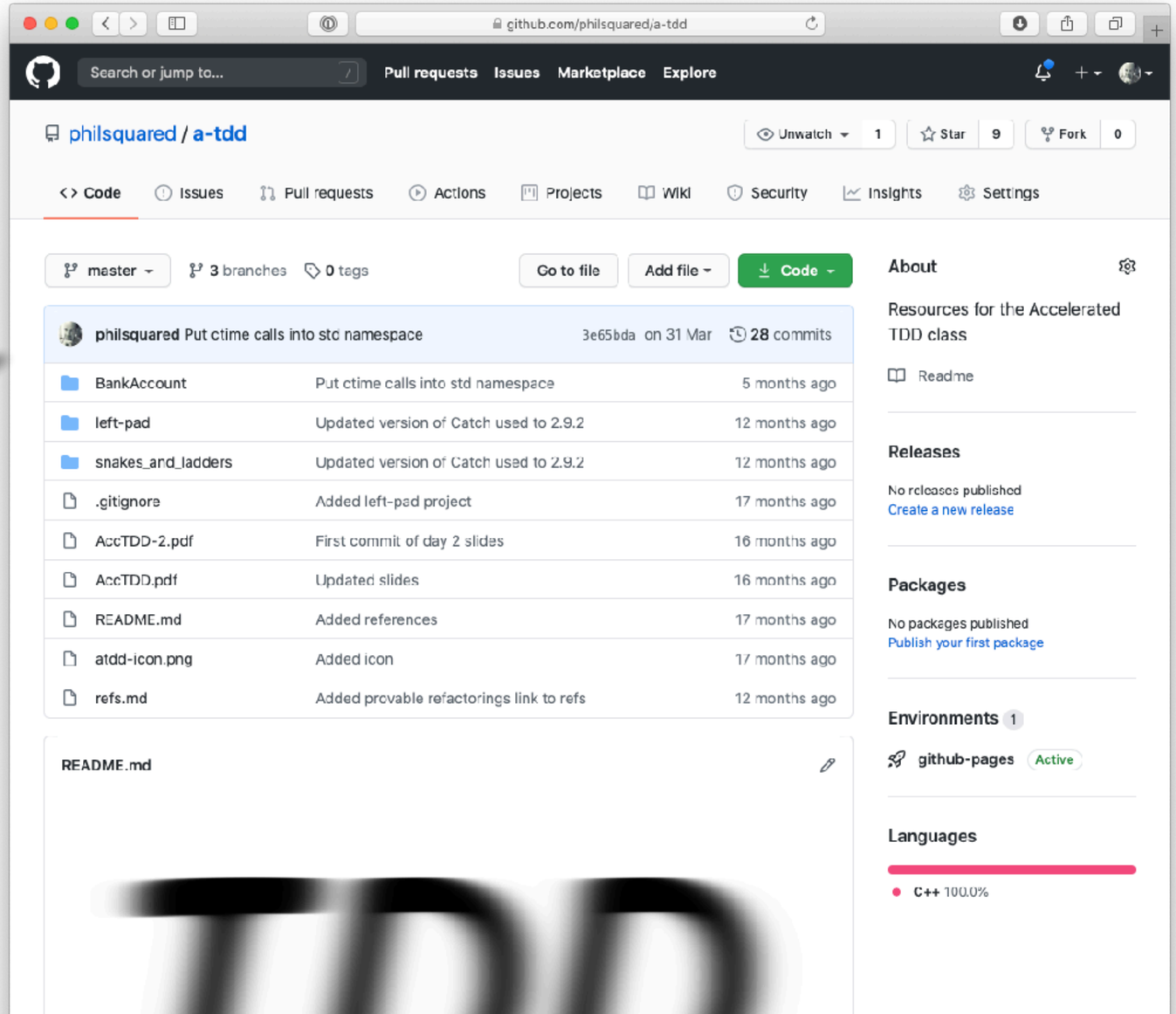
atdd.dev

atdd.dev

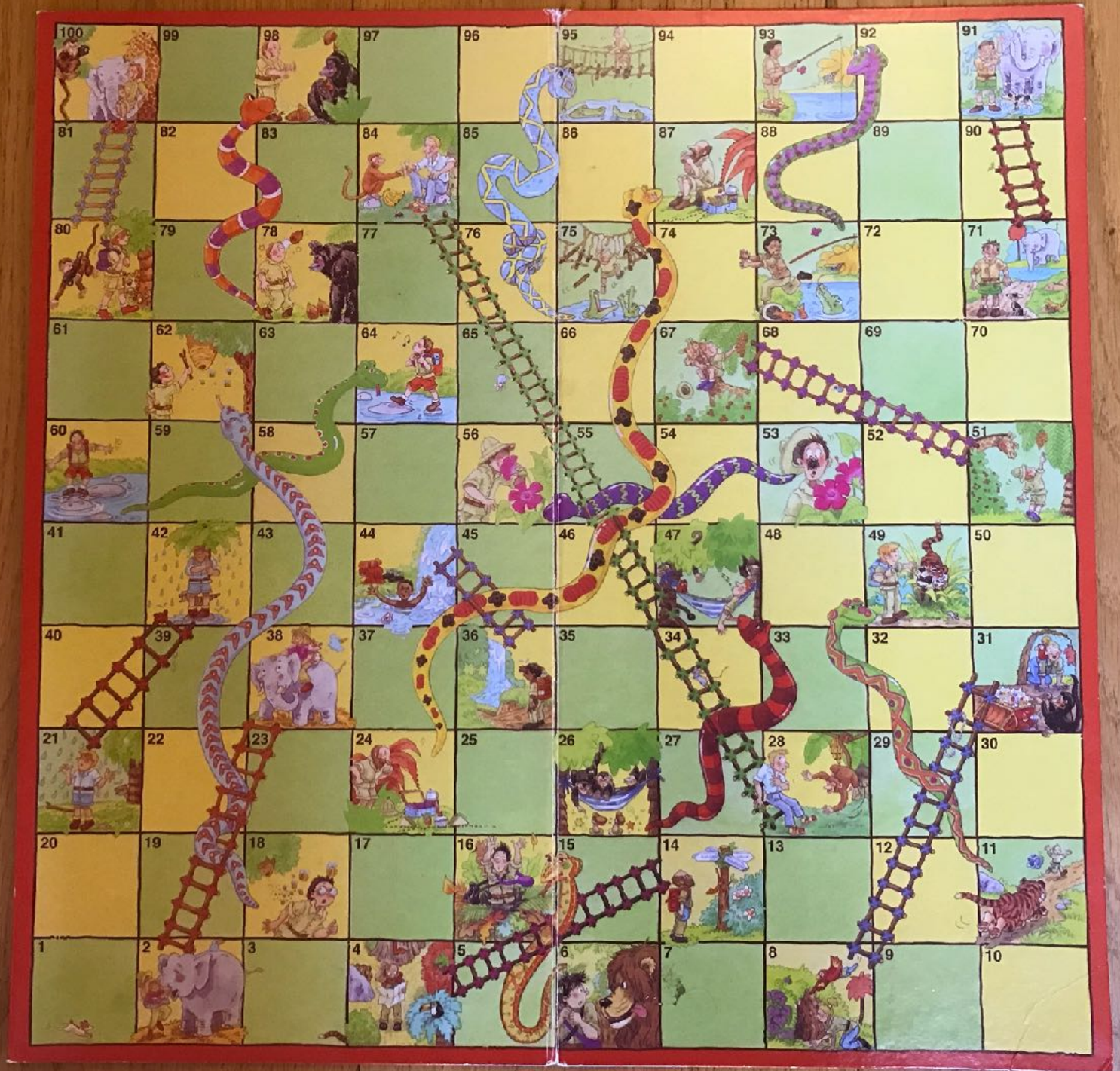

atdd.dev
Accelerated
TDD
Accelerated TDD - for more productive C++
This repository contains supplimentary material for my C++ TDD course, "Accelerated TDD".
The course has been given as both one day and two day versions, and each time I give it we may cover some different material.
In the repo you'll find the slides, sample projects and full references from my material
- Phil Nash

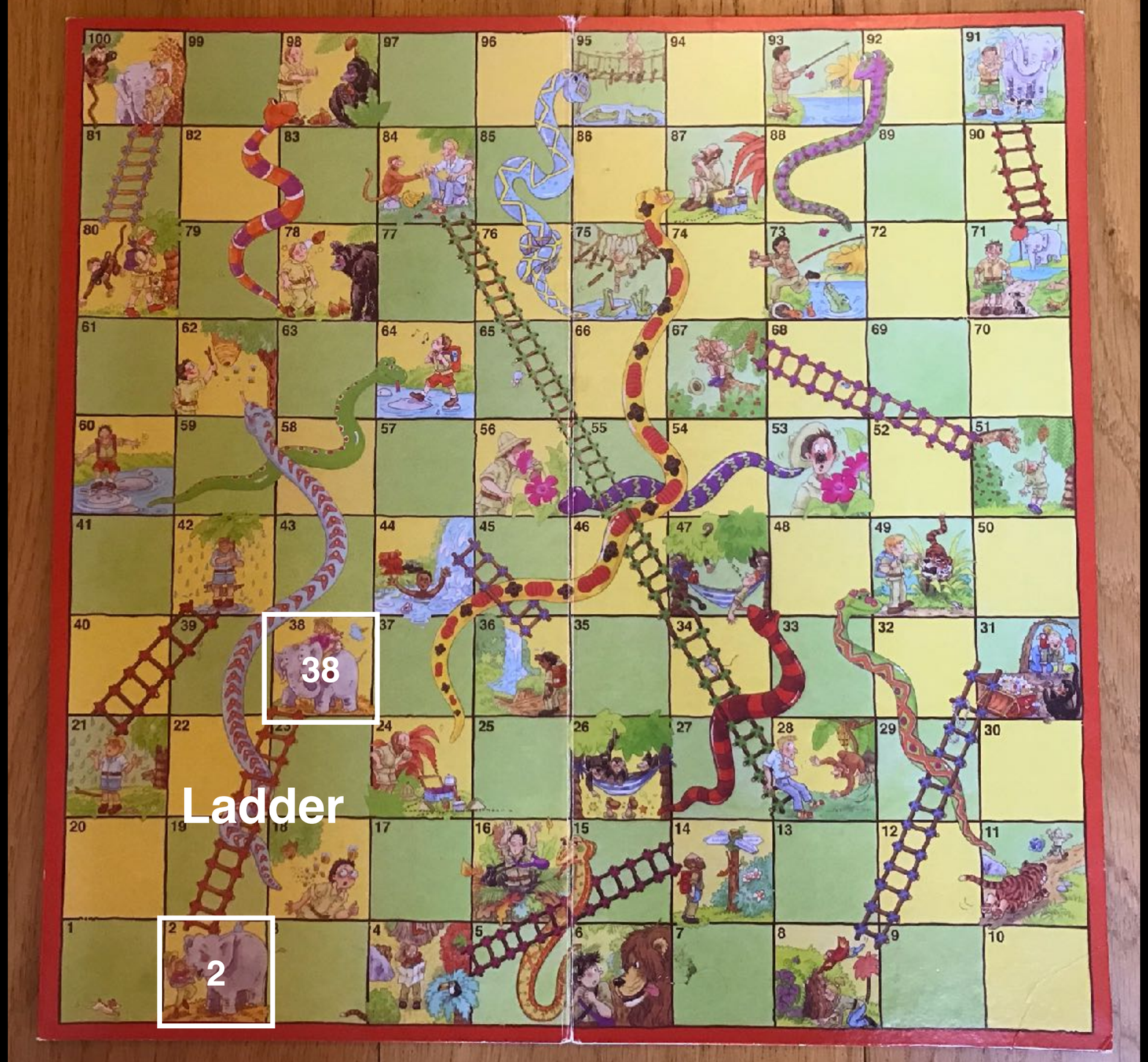
atdd.dev


atdd.dev
Accelerated
TDD
Accelerated TDD - for more productive C++
This repository contains supplimentary material for my C++ TDD course, "Accelerated TDD".
The course has been given as both one day and two day versions, and each time I give it we may cover some different material.
In the repo you'll find the slides, sample projects and full references from my material
- Phil Nash


github.com/philsquared/a-tdd
Search or jump to...
Pull requests
Issues
Marketplace
Explore
philsquared / a-tdd
Unwatch 1
Star 9
Fork 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
master
3 branches
0 tags
Go to file
Add file
Code
philsquared Put ctime calls into std namespace
3e65bda on 31 Mar
28 commits
BankAccount Put ctime calls into std namespace 5 months ago
left-pad Updated version of Catch used to 2.9.2 12 months ago
snakes_and_ladders Updated version of Catch used to 2.9.2 12 months ago
.gitignore Added left-pad project 17 months ago
AccTDD-2.pdf First commit of day 2 slides 16 months ago
AccTDD.pdf Updated slides 16 months ago
README.md Added references 17 months ago
atdd-icon.png Added icon 17 months ago
refs.md Added provable refactorings link to refs 12 months ago
README.md
About
Resources for the Accelerated TDD class
Readme
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Environments 1
github-pages Active
Languages
C++ 100.0%

snakes & ladders


100 99 98 97 96 95 94 93 92 91
81 82 83 84 85 86 87 88 89 90
80 79 78 77 76 75 74 73 72 71
61 62 63 64 65 66 67 68 69 70
60 59 58 57 56 55 54 53 52 51
41 42 43 44 45 46 47 48 49 50
40 39 38 37 36 35 34 33 32 31
21 22 23 24 25 26 27 28 29 30
20 19 18 17 16 15 14 13 12 11
1 2 3 4 5 6 7 8 9 10


38
Ladder
2

2 Ladder 38

2 Ladder 38

2	Ladder	38
4	Ladder	14

2	Ladder	38
4	Ladder	14
8	Ladder	31
21	Ladder	42
28	Ladder	84
36	Ladder	44
47	Snake	26
49	Snake	11
51	Ladder	67
56	Snake	53
62	Snake	18
64	Snake	60
71	Ladder	91
80	Ladder	100
87	Snake	24
93	Snake	73
95	Snake	75
98	Snake	78

```
{ 2, { Portal::Type::Ladder, 38 } },
{ 4, { Portal::Type::Ladder, 14 } },
{ 8, { Portal::Type::Ladder, 31 } },
{ 21, { Portal::Type::Ladder, 42 } },
{ 28, { Portal::Type::Ladder, 84 } },
{ 36, { Portal::Type::Ladder, 44 } },
{ 47, { Portal::Type::Snake, 26 } },
{ 49, { Portal::Type::Snake, 11 } },
{ 51, { Portal::Type::Ladder, 67 } },
{ 56, { Portal::Type::Snake, 53 } },
{ 62, { Portal::Type::Snake, 18 } },
{ 64, { Portal::Type::Snake, 60 } },
{ 71, { Portal::Type::Ladder, 91 } },
{ 80, { Portal::Type::Ladder, 100 } },
{ 87, { Portal::Type::Snake, 24 } },
{ 93, { Portal::Type::Snake, 73 } },
{ 95, { Portal::Type::Snake, 75 } },
{ 98, { Portal::Type::Snake, 78 } }
```

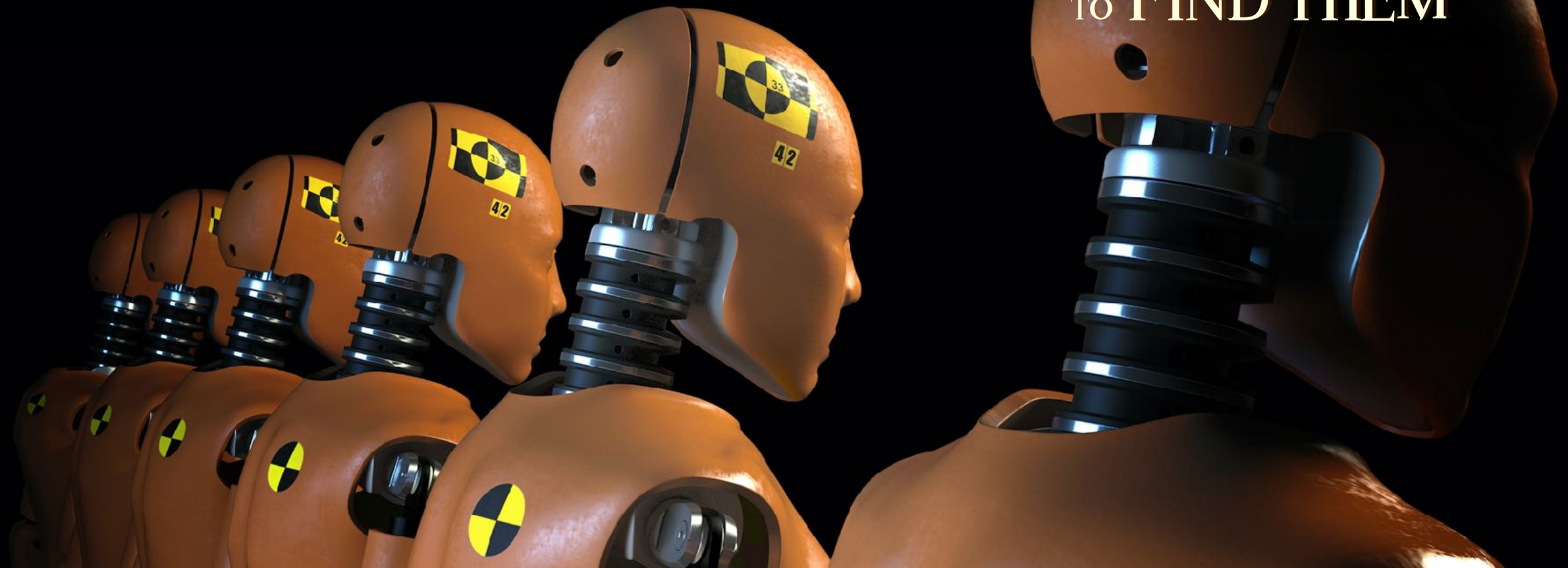
```
std::map<uint32_t, Portal> portals =
    {
        { 2, { Portal::Type::Ladder, 38 } },
        { 4, { Portal::Type::Ladder, 14 } },
        { 8, { Portal::Type::Ladder, 31 } },
        { 21, { Portal::Type::Ladder, 42 } },
        { 28, { Portal::Type::Ladder, 84 } },
        { 36, { Portal::Type::Ladder, 44 } },
        { 47, { Portal::Type::Snake, 26 } },
        { 49, { Portal::Type::Snake, 11 } },
        { 51, { Portal::Type::Ladder, 67 } },
        { 56, { Portal::Type::Snake, 53 } },
        { 62, { Portal::Type::Snake, 18 } },
        { 64, { Portal::Type::Snake, 60 } },
        { 71, { Portal::Type::Ladder, 91 } },
        { 80, { Portal::Type::Ladder, 100 } },
        { 87, { Portal::Type::Snake, 24 } },
        { 93, { Portal::Type::Snake, 73 } },
        { 95, { Portal::Type::Snake, 75 } },
        { 98, { Portal::Type::Snake, 78 } }
    };
```

```
struct Portal {
    enum class Type { Snake, Ladder };
    Type type;
    uint32_t target;
};

std::map<uint32_t, Portal> portals =
    {
        { 2, { Portal::Type::Ladder, 38 } },
        { 4, { Portal::Type::Ladder, 14 } },
        { 8, { Portal::Type::Ladder, 31 } },
        { 21, { Portal::Type::Ladder, 42 } },
        { 28, { Portal::Type::Ladder, 84 } },
        { 36, { Portal::Type::Ladder, 44 } },
        { 47, { Portal::Type::Snake, 26 } },
        { 49, { Portal::Type::Snake, 11 } },
        { 51, { Portal::Type::Ladder, 67 } },
        { 56, { Portal::Type::Snake, 53 } },
        { 62, { Portal::Type::Snake, 18 } },
        { 64, { Portal::Type::Snake, 60 } },
        { 71, { Portal::Type::Ladder, 91 } },
        { 80, { Portal::Type::Ladder, 100 } },
```

Walk through

FANTASTIC TEST FRAMEWORKS AND WHERE TO FIND THEM



WHY USE A TEST FRAMEWORK

```
int add( int a, int b );
```

```
#include <cassert>

int add( int a, int b );

int main() {
   assert( add( 1, 2 ) == 3 );
}
```

```
#include <cassert>

int add( int a, int b );

int main() {
   assert( add( 1, 2 ) == 3 );
}
```

```
#include <cassert>

int add( int a, int b ){
    return a*b;
}

int main() {
   assert( add( 1, 2 ) == 3 );
}
```

```
#include <cassert>

int add( int a, int b ){
    return a*b;
}

int main() {
   assert( add( 1, 2 ) == 3 );
}
```

```
Assertion failed: (add( 1, 2 ) == 3), function main,
  file /Users/phil/Dev/Scratch/AddTest/main.cpp, line 8.

Process finished with exit code 6
```

```
#include <cassert>

int add( int a, int b ){
    return a*b;
}

int main() {
   assert( add( 1, 2 ) == 3 );
}
```

```
Assertion failed: (add( 1, 2 ) == 3), function main,
  file /Users/phil/Dev/Scratch/AddTest/main.cpp, line 8.

Process finished with exit code 6
```

```
#include <cassert>

int add( int a, int b ){
    return a*b;
}

int main() {
   assert( add( 1, 2 ) == 3 );
}
```

```
Assertion failed: (add( 1, 2 ) == 3), function main,
  file /Users/phil/Dev/Scratch/AddTest/main.cpp, line 8.

Process finished with exit code 6
```

```
#include <cassert>

int add( int a, int b ){
    return a*b;
}

int main() {
   assert( add( 1, 2 ) == 3 );
}
```

```
Assertion failed: (add( 1, 2 ) == 3), function main,
  file /Users/phil/Dev/Scratch/AddTest/main.cpp, line 8.

Process finished with exit code 6
```

<cassert> pros:

- no dependencies
- some useful output (file, line, expr...)
- natural, familiar, syntax and usage

<cassert> pros:

- no dependencies
- some useful output (file, line, expr...)
- natural, familiar, syntax and usage

<cassert> cons:

<cassert> pros:

- no dependencies
- some useful output (file, line, expr...)
- natural, familiar, syntax and usage

<cassert> cons:

- not enough feedback (breakdown of expr?)

<cassert> pros:

- no dependencies
- some useful output (file, line, expr...)
- natural, familiar, syntax and usage

<cassert> cons:

- not enough feedback (breakdown of expr?)
- only console output

<cassert> pros:

- no dependencies
- some useful output (file, line, expr...)
- natural, familiar, syntax and usage

<cassert> cons:

- not enough feedback (breakdown of expr?)
- only console output
- no test names, groups/ suites

<cassert> pros:

- no dependencies
- some useful output (file, line, expr...)
- natural, familiar, syntax and usage

<cassert> cons:

- not enough feedback (breakdown of expr?)
- only console output
- no test names, groups/ suites
- aborts on first failure

List of unit testing frameworks

From Wikipedia, the free encyclopedia

This article contains embedded lists that **may be poorly defined, unverified or indiscriminate**. Please help to clean it up to meet Wikipedia's quality standards. Where appropriate, incorporate items into the main body of the article. *(April 2018)*

This page is a list of tables of code-driven unit testing frameworks for various programming languages. Some but not all of these are based on xUnit.

Contents [hide]

1 Columns (classification)
2 Languages
2.1 ABAP
2.2 ActionScript / Adobe Flex
2.3 Ada
2.4 AppleScript
2.5 ASCET
2.6 ASP
2.7 Bash
2.8 BPEL
2.9 C
2.10 C#
2.11 C++
2.12 Cg
2.13 CFML (ColdFusion)

List of unit testing frameworks

2.11 C++

79 frameworks!

CppUnit

CppUnit

CppUnitLite

CppUnit

CppUnitLite

Boost.Test

CppUnit

CppUnitLite

Boost.Test

Google Test

CppUnit

CppUnitLite

Boost.Test

Google Test

Catch/ Catch2

CppUnit

CppUnitLite

Boost.Test

Google Test

Catch/ Catch2

doctest

CppUnit

CppUnitLite

Boost.Test

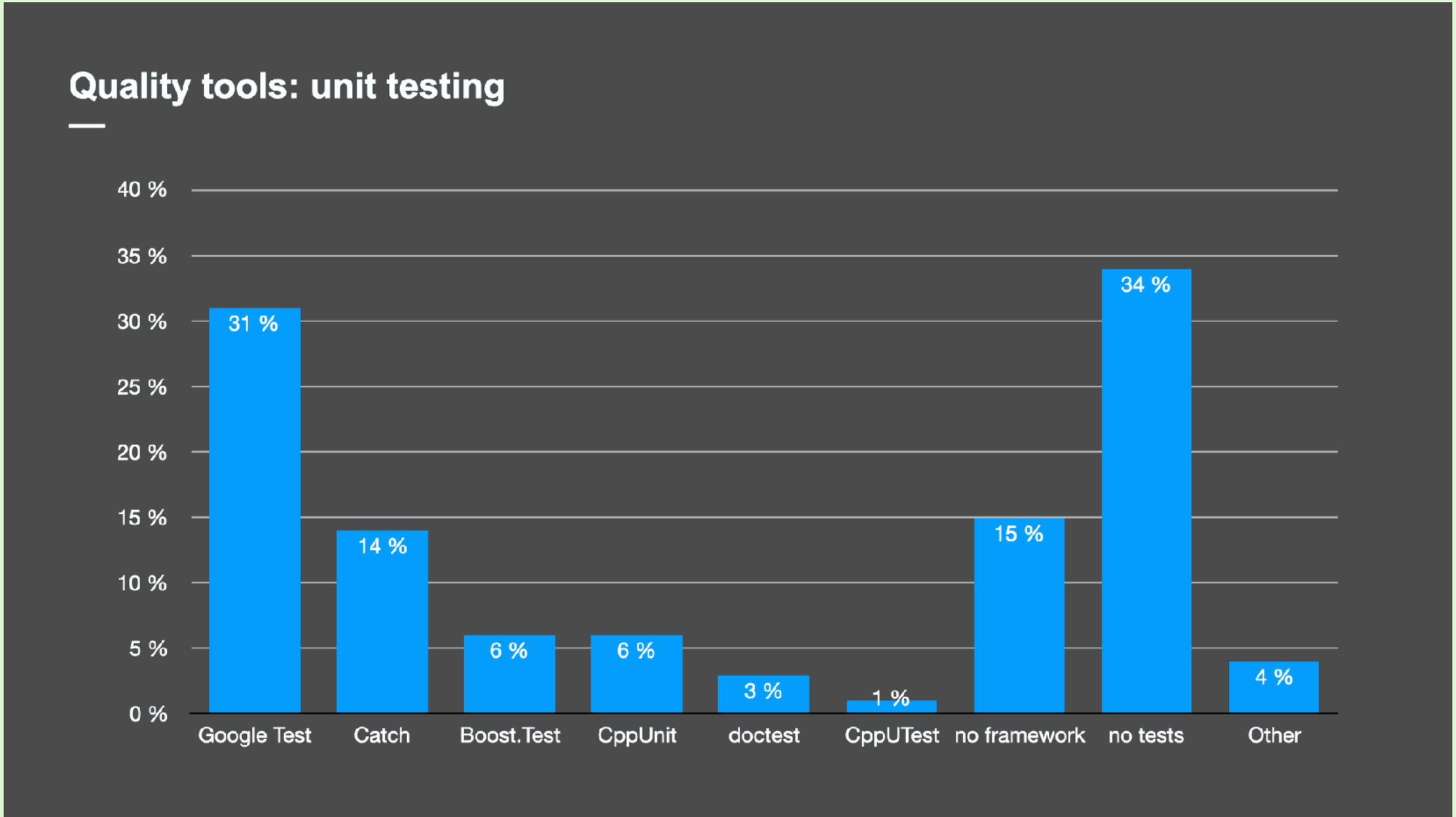
Google Test

Catch/ Catch2

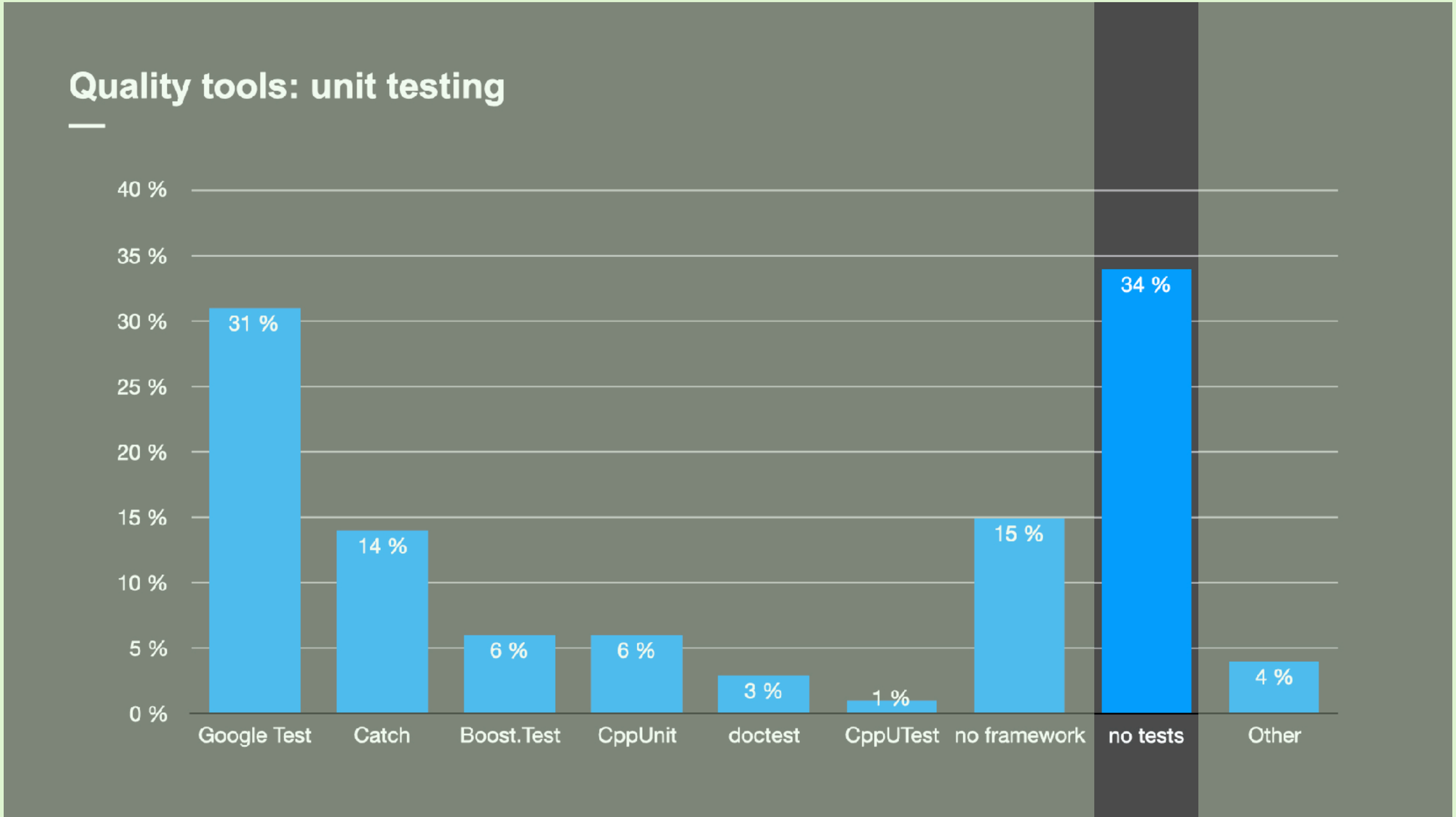
doctest

Boost.Experimental.UT

JetBrains Developer Ecosystem report 2020



JetBrains Developer Ecosystem report 2020



introducing
Catch²

```
TEST_CASE( "vectors can be sized and resized", "[vector]" ) {

    std::vector<int> v( 5 );

    REQUIRE( v.size() == 5 );
    REQUIRE( v.capacity() >= 5 );

    SECTION( "resizing bigger changes size and capacity" ) {
        v.resize( 10 );

        REQUIRE( v.size() == 10 );
        REQUIRE( v.capacity() >= 10 );
    }
    SECTION( "resizing smaller changes size but not capacity" ) {
        v.resize( 0 );

        REQUIRE( v.size() == 0 );
        REQUIRE( v.capacity() >= 5 );

        SECTION( "We can use the 'swap trick' to reset the capacity" ) {
            std::vector<int> empty;
            empty.swap( v );

            REQUIRE( v.capacity() == 0 );
        }
    }
    SECTION( "reserving bigger changes capacity but not size" ) {
        v.reserve( 10 );
```

```
TEST_CASE( "vectors can be sized and resized", "[vector]" ) {

    std::vector<int> v( 5 );

    REQUIRE( v.size() == 5 );
    REQUIRE( v.capacity() >= 5 );

    SECTION( "resizing bigger changes size and capacity" ) {
        v.resize( 10 );

        REQUIRE( v.size() == 10 );
        REQUIRE( v.capacity() >= 10 );
    }
    SECTION( "resizing smaller changes size but not capacity" ) {
        v.resize( 0 );

        REQUIRE( v.size() == 0 );
        REQUIRE( v.capacity() >= 5 );

        SECTION( "We can use the 'swap trick' to reset the capacity" ) {
            std::vector<int> empty;
            empty.swap( v );

            REQUIRE( v.capacity() == 0 );
        }
    }
    SECTION( "reserving bigger changes capacity but not size" ) {
        v.reserve( 10 );
```

```
TEST_CASE( "vectors can be sized and resized", "[vector]" ) {

    std::vector<int> v( 5 );

    REQUIRE( v.size() == 5 );
    REQUIRE( v.capacity() >= 5 );

    SECTION( "resizing bigger changes size and capacity" ) {
        v.resize( 10 );

        REQUIRE( v.size() == 10 );
        REQUIRE( v.capacity() >= 10 );
    }
    SECTION( "resizing smaller changes size but not capacity" ) {
        v.resize( 0 );

        REQUIRE( v.size() == 0 );
        REQUIRE( v.capacity() >= 5 );

        SECTION( "We can use the 'swap trick' to reset the capacity" ) {
            std::vector<int> empty;
            empty.swap( v );

            REQUIRE( v.capacity() == 0 );
        }
    }
    SECTION( "reserving bigger changes capacity but not size" ) {
        v.reserve( 10 );
```

```
std::vector<int> v( 5 );

REQUIRE( v.size() == 5 );
REQUIRE( v.capacity() >= 5 );

SECTION( "resizing bigger changes size and capacity" ) {
    v.resize( 10 );

    REQUIRE( v.size() == 10 );
    REQUIRE( v.capacity() >= 10 );
}
SECTION( "resizing smaller changes size but not capacity" ) {
    v.resize( 0 );

    REQUIRE( v.size() == 0 );
    REQUIRE( v.capacity() >= 5 );

    SECTION( "We can use the 'swap trick' to reset the capacity" ) {
        std::vector<int> empty;
        empty.swap( v );

        REQUIRE( v.capacity() == 0 );
    }
}
SECTION( "reserving bigger changes capacity but not size" ) {
    v.reserve( 10 );

    REQUIRE( v.size() == 5 );
    REQUIRE( v.capacity() >= 10 );
}
```

```
SCENARIO( "Vector resizing affects size and capacity", "[vector][bdd][size][capacity]" ) {
    GIVEN( "an empty vector" ) {
        std::vector<int> v;
        REQUIRE( v.size() == 0 );

        WHEN( "it is made larger" ) {
            v.resize( 10 );
            THEN( "the size and capacity go up" ) {
                REQUIRE( v.size() == 10 );
                REQUIRE( v.capacity() >= 10 );

                AND_WHEN( "it is made smaller again" ) {
                    v.resize( 5 );
                    THEN( "the size goes down but the capacity stays the same" ) {
                        REQUIRE( v.size() == 5 );
                        REQUIRE( v.capacity() >= 10 );
                    }
                }
            }
        }

        WHEN( "we reserve more space" ) {
            v.reserve( 10 );
            THEN( "The capacity is increased but the size remains the same" ) {
                REQUIRE( v.capacity() >= 10 );
                REQUIRE( v.size() == 0 );
            }
        }
    }
}
```

```
        v.resize( 10 );
        THEN( "the size and capacity go up" ) {
            REQUIRE( v.size() == 10 );
            REQUIRE( v.capacity() >= 10 );

            AND_WHEN( "it is made smaller again" ) {
                v.resize( 5 );
                THEN( "the size goes down but the capacity stays the same" ) {
                    REQUIRE( v.size() == 5 );
                    REQUIRE( v.capacity() >= 10 );
                }
            }
        }
    }

    WHEN( "we reserve more space" ) {
        v.reserve( 10 );
        THEN( "The capacity is increased but the size remains the same" ) {
            REQUIRE( v.capacity() >= 10 );
            REQUIRE( v.size() == 0 );
        }
    }
}
}
```

```
-------------------------------------------------------------------------------
Scenario: Vector resizing affects size and capacity
     Given: an empty vector
      When: we reserve more space
      Then: The capacity is increased but the size remains the same
-------------------------------------------------------------------------------
```

Anatomy of a good test

```
TEST_CASE("add() returns the sum of its arguments") {
   REQUIRE( add( 1, 2 ) == 3 );
}
```

```
TEST_CASE("add() returns the sum of its arguments") {
   REQUIRE( add( 1, 2 ) == 3 );
}
```

Tests should have a good name

```
TEST_CASE("add() returns the sum of its arguments") {
   REQUIRE( add( 1, 2 ) == 3 );
}
```

```
TEST_CASE( "Most recently used list" ) {

    MRUList<std::string> list;

    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }
    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

```
TEST_CASE( "Most recently used list" ) {

    MRUList<std::string> list;

    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }
    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

```
TEST_CASE( "Most recently used list" ) {

    MRUList<std::string> list;

    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }
    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

STATE EXPECTATIONS

Unit Tests should be reproducible and deterministic

```
TEST_CASE( "Most recently used list" ) {

    MRUList<std::string> list;

    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }
    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

Unit Tests should be reproducible and deterministic

```
TEST_CASE( "Most recently used list" ) {

    MRUList<std::string> list;

    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }
    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

"Arrange"

Unit Tests should be reproducible and deterministic

```
TEST_CASE( "Most recently used list" ) {

    MRUList<std::string> list;

    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }
    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

"Arrange"

"Act"

Unit Tests should be reproducible and deterministic

```
TEST_CASE( "Most recently used list" ) {

    MRUList<std::string> list;

    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }
    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

"Arrange"

"Act"

"Assert"

Tests should:

Have a good name

State expectations

Unit Tests should be reproducible and deterministic

"Arrange" → "Act" → "Assert"

Tests should:

Have a good name

State expectations

Have a single "Logical" Assert

Unit Tests should be reproducible and deterministic

"Arrange" → "Act" → "Assert"

Tests should:

Have a single "Logical" Assert

```
TEST_CASE( "Most recently used list" ) {

    MRUList<std::string> list;

    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }
    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

Single assert

Tests should:

Have a single "Logical" Assert

```
TEST_CASE( "Most recently used list" ) {

    MRUList<std::string> list;

    SECTION( "…?" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );

        list.add("item1");

        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

Dependency

Multiple asserts

Tests should:

Have a good name

State expectations

Have a single "Logical" Assert

Unit Tests should be reproducible and deterministic

"Arrange" → "Act" → "Assert"

Tests should:

Have a good name

State expectations

Have a single "Logical" Assert

Use Public Interface

Unit Tests should be reproducible and deterministic

"Arrange" → "Act" → "Assert"

Tests should: Use Public Interface

Q. How do you test private methods?

Tests should: Use Public Interface

Q. How do you test private methods?

A. You don’t

Tests should: Use Public Interface

Q. How do you test private methods?

A. You don’t

Q. But what if you really need to?

Tests should: Use Public Interface

```
friend class TestAccess;
```

Tests should: Use Public Interface

```
friend class TestAccess;
```

To test “non-functional” requirements

(ref counts, caches etc)

Tests should: Use Public Interface

```
friend class TestAccess;

extern name_not_in_header;
```

Tests should: Use Public Interface

```
friend class TestAccess;

extern name_not_in_header;

#define private public;
```

Tests should:

Have a good name

State expectations

Have a single "Logical" Assert

Use Public Interface

Unit Tests should be reproducible and deterministic

"Arrange" → "Act" → "Assert"

Tests should:

Have a good name

State expectations

Have a single "Logical" Assert

Use Public Interface

Have low Cyclomatic Complexity

Unit Tests should be reproducible and deterministic

"Arrange" → "Act" → "Assert"

TESTS SHOULD: HAVE LOW CYCLOMATIC COMPLEXITY

Cyclomatic complexity is a software metric used to indicate the complexity of a program. It is a quantitative measure of the number of *linearly independent paths* through a program's source code.

- Wikipedia

Tests should: have low Cyclomatic Complexity

`if` `switch` `for` `while` `? :`

Tests should:

have low Cyclomatic Complexity

`if`

`for`

TESTS SHOULD: HAVE LOW CYCLOMATIC COMPLEXITY

`if` `for`

Usually means a test should have aborted

Tests should: **have low Cyclomatic Complexity**

`if` `for`

Usually means a test should have aborted

```
REQUIRE( some_condition() );
CHECK( would_crash_if_condition_not_met() );
```

Tests should: **have low Cyclomatic Complexity**

`if`

Usually means a test should have aborted

`for`

Usually testing **properties** for a range of values

Tests should:

Have a good name

State expectations

Have a single "Logical" Assert

Use Public Interface

Have low Cyclomatic Complexity

Unit Tests should be reproducible and deterministic

"Arrange" → "Act" → "Assert"

Tests should:

Have a good name

State expectations

Have a single "Logical" Assert

Use Public Interface

Have low Cyclomatic Complexity

Unit Tests should be reproducible and deterministic

"Arrange" → "Act" → "Assert"

Always have A Reason

Tests should: Always have A Reason

Tests should: Always have A Reason

"This feels like we should have more tests"

Tests should: Always have A Reason

"This feels like we should have more tests"

"I'm writing this test because my boss told me to"

Tests should: Always have A Reason

"This feels like we should have more tests"

"I'm writing this test because my boss told me to"

"I went to a TDD class and they said I should write more tests"

Tests should: Always have A Reason

"This feels like we should have more tests"

"I'm writing this test because my boss told me to"

"I went to a TDD class and they said I should write more tests"

"I'm increasing my total lines of code!"

Tests should: Always have A Reason

~~"This feels like we should have more tests"~~

~~"I'm writing this test because my boss told me to"~~

~~"I went to a TDD class and they said I should write more tests"~~

~~"I'm increasing my total lines of code!"~~

Why write tests?

Google Jam board

Link in the chat...

Tests should:

Have a good name

State expectations

Have a single "Logical" Assert

Use Public Interface

Have low Cyclomatic Complexity

Unit Tests should be reproducible and deterministic

"Arrange" → "Act" → "Assert"

Always have A Reason

Left-pad

github.com/philsquared/a-tdd

Search or jump to... / Pull requests Issues Marketplace Explore

philsquared / a-tdd

Unwatch 1 Star 9 Fork 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 3 branches 0 tags Go to file Add file Code

philsquared Updated Catch to 2.13.0 4ff1f25 2 hours ago 30 commits

BankAccount	Updated Catch to 2.13.0	2 hours ago
left-pad	Updated Catch to 2.13.0	2 hours ago
snakes_and_ladders	Updated Catch to 2.13.0	2 hours ago
.gitignore	Added left-pad project	17 months ago
AccTDD-2.pdf	First commit of day 2 slides	16 months ago
AccTDD.pdf	Updated slides	16 months ago
README.md	Added references	17 months ago
atdd-icon.png	Added icon	17 months ago
refs.md	Added provable refactorings link to refs	12 months ago

README.md

About

Resources for the Accelerated TDD class

Readme

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Environments 1

github-pages Active

github.com/philsquared/a-tdd/tree/master/left-pad

Search or jump to... / Pull requests Issues Marketplace Explore

philsquared / a-tdd

Unwatch 1 Star 9 Fork 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master a-tdd / left-pad / Go to file Add file

philsquared Updated Catch to 2.13.0 4ff1f25 2 hours ago History

..

CATCH_CHEATSHEET.md	Added Catch cheatsheet doc	2 hours ago
CMakeLists.txt	Added left-pad project	17 months ago
README.md	Added left-pad project	17 months ago
catch.hpp	Updated Catch to 2.13.0	2 hours ago
main.cpp	Added left-pad project	17 months ago
tests.cpp	Added left-pad project	17 months ago

README.md

Left Pad

We've all heard of the notorious left-pad javascript library that was pulled from NPM a few years ago. We scoffed at how having a dependency on such a trivial library disrupted so much of the internet. But how trivial was it really? Let's implement it in C++!

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

master | a-tdd / left-pad / | Go to file | Add file

philsquared Updated Catch to 2.13.0 4ff1f25 2 hours ago History

..

CATCH_CHEATSHEET.md	Added Catch cheatsheet doc	2 hours ago
CMakeLists.txt	Added left-pad project	17 months ago
README.md	Added left-pad project	17 months ago
catch.hpp	Updated Catch to 2.13.0	2 hours ago
main.cpp	Added left-pad project	17 months ago
tests.cpp	Added left-pad project	17 months ago

README.md

Left Pad

We've all heard of the notorious left-pad javascript library that was pulled from NPM a few years ago. We scoffed at how having a dependency on such a trivial library disrupted so much of the internet. But how trivial was it really? Let's implement it in C++!

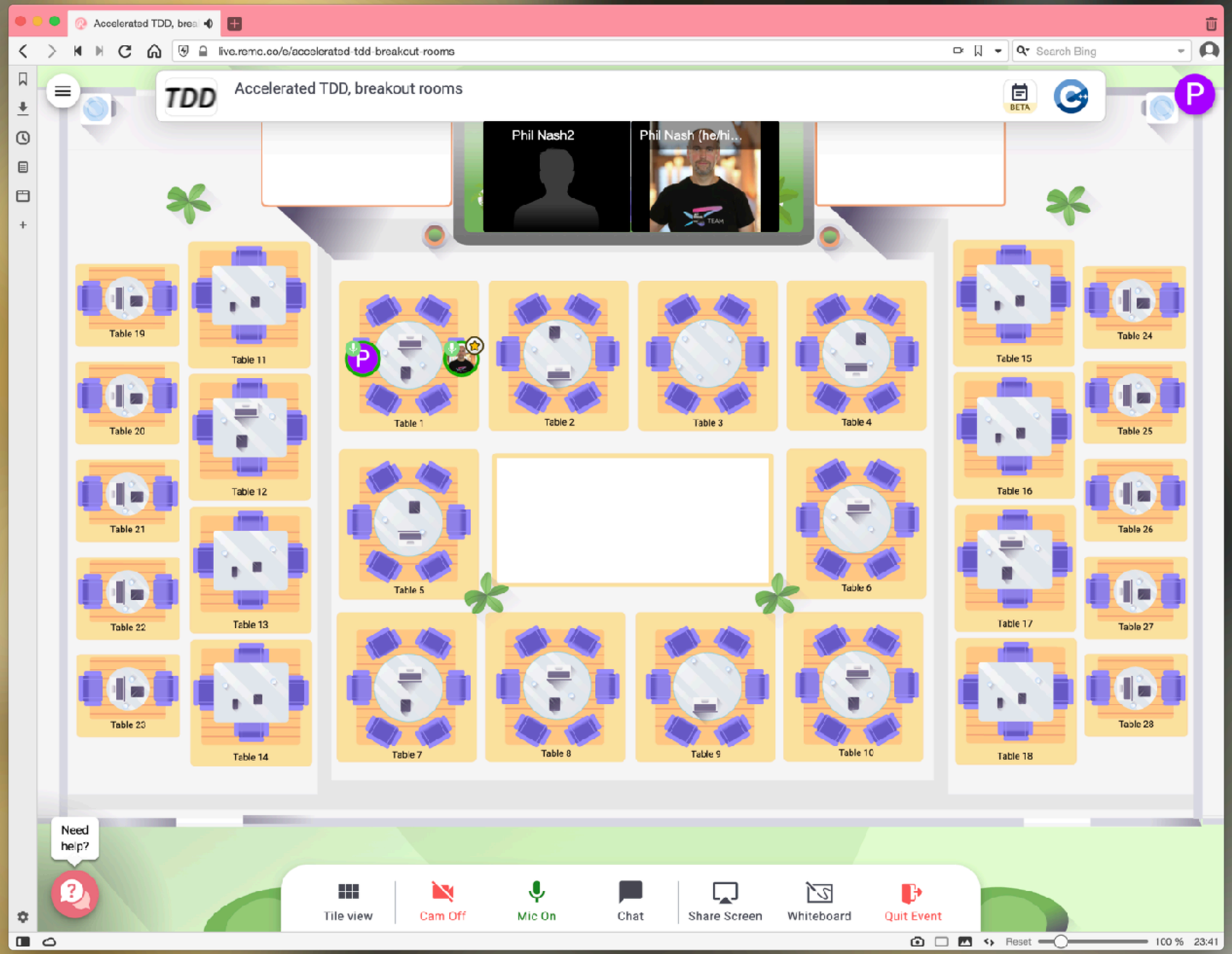
The requirements are:

- Write a function that takes a string and a number and returns a string
- The number represents the "padded" size.
- The returned string should be the input string, padded with spaces on the left if shorter than the size argument.
- The pad character should be optionally specifiable via an extra, defaulted argument

Remo

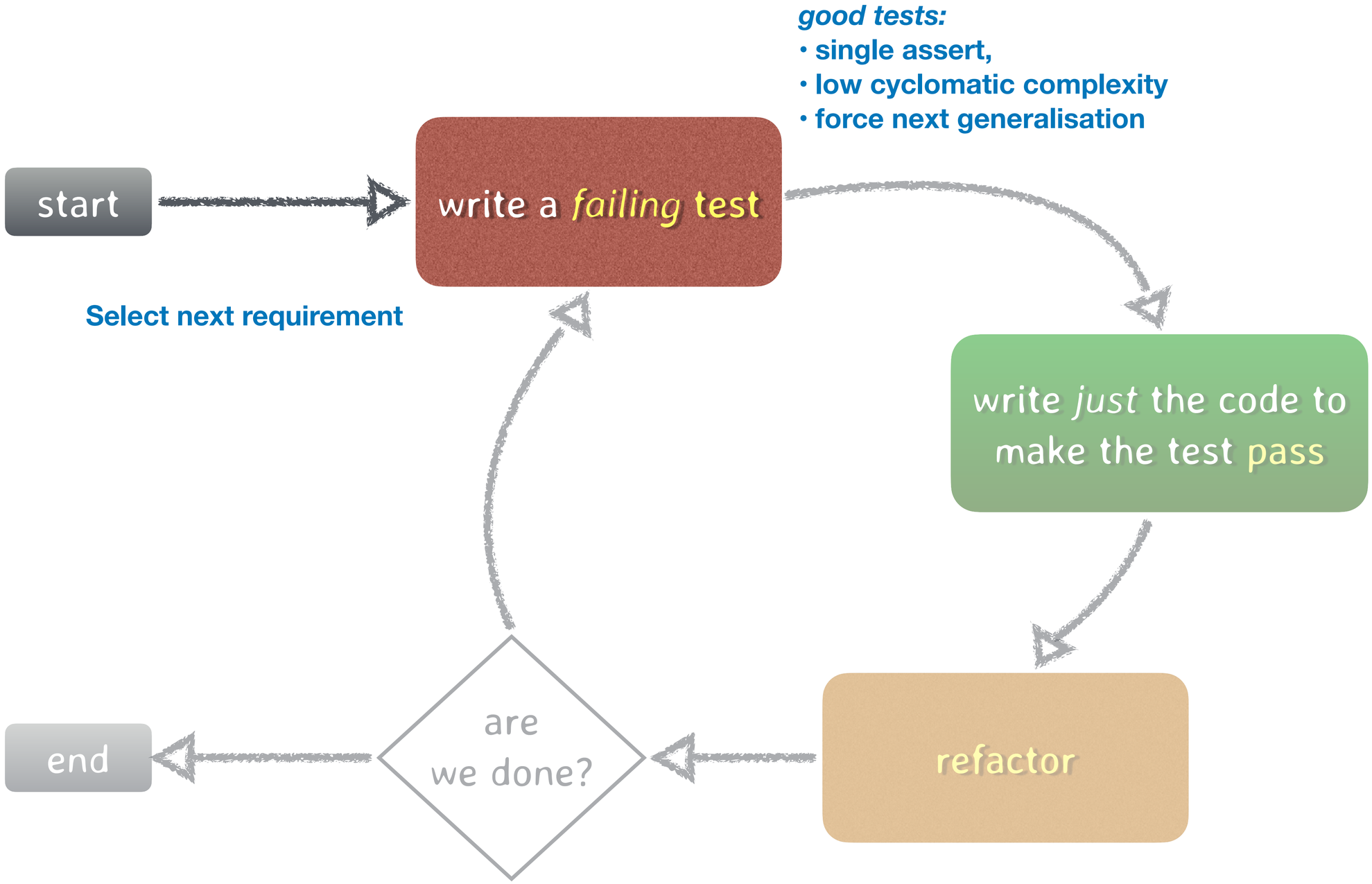
In 3s

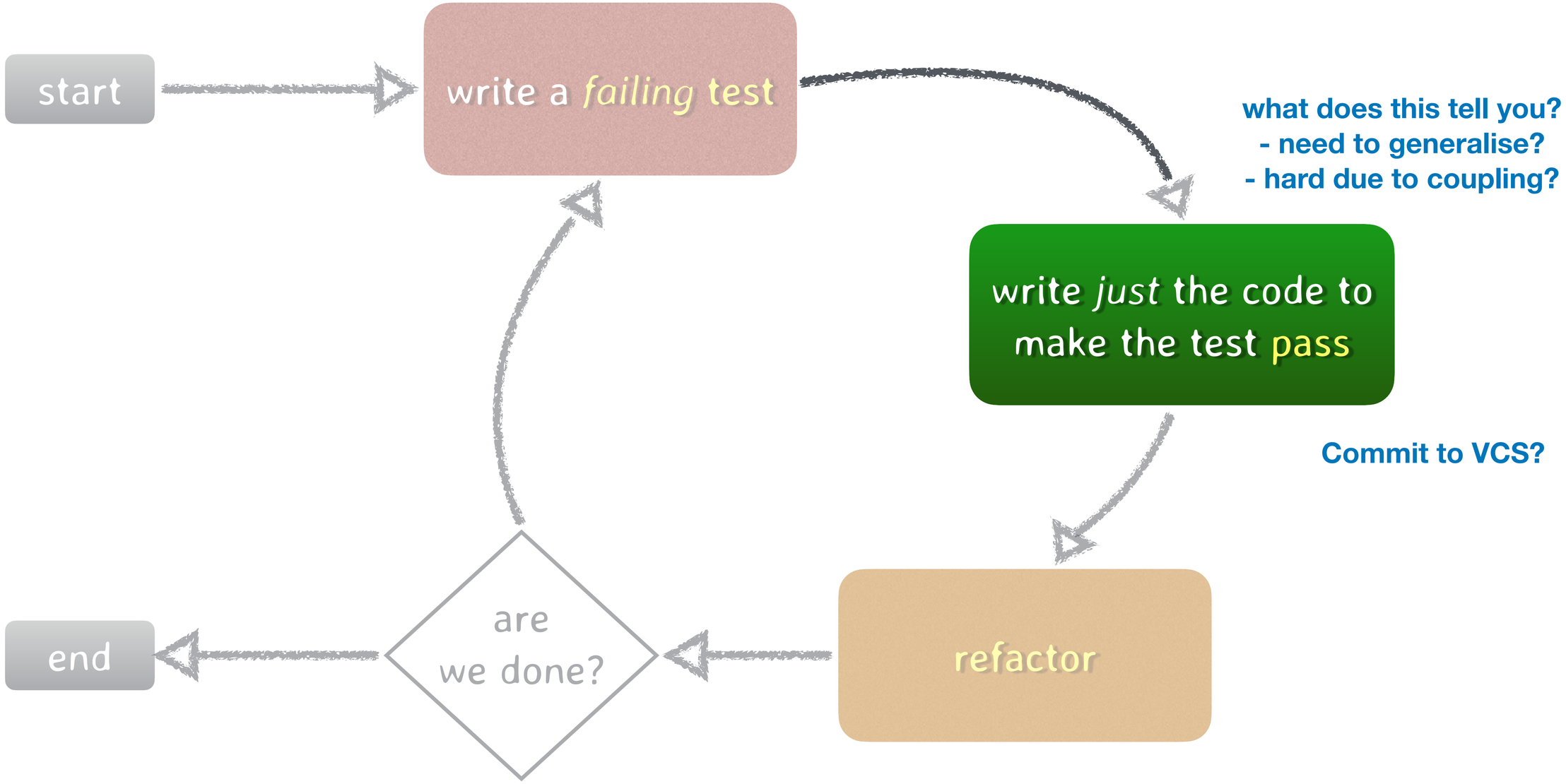
30 mins

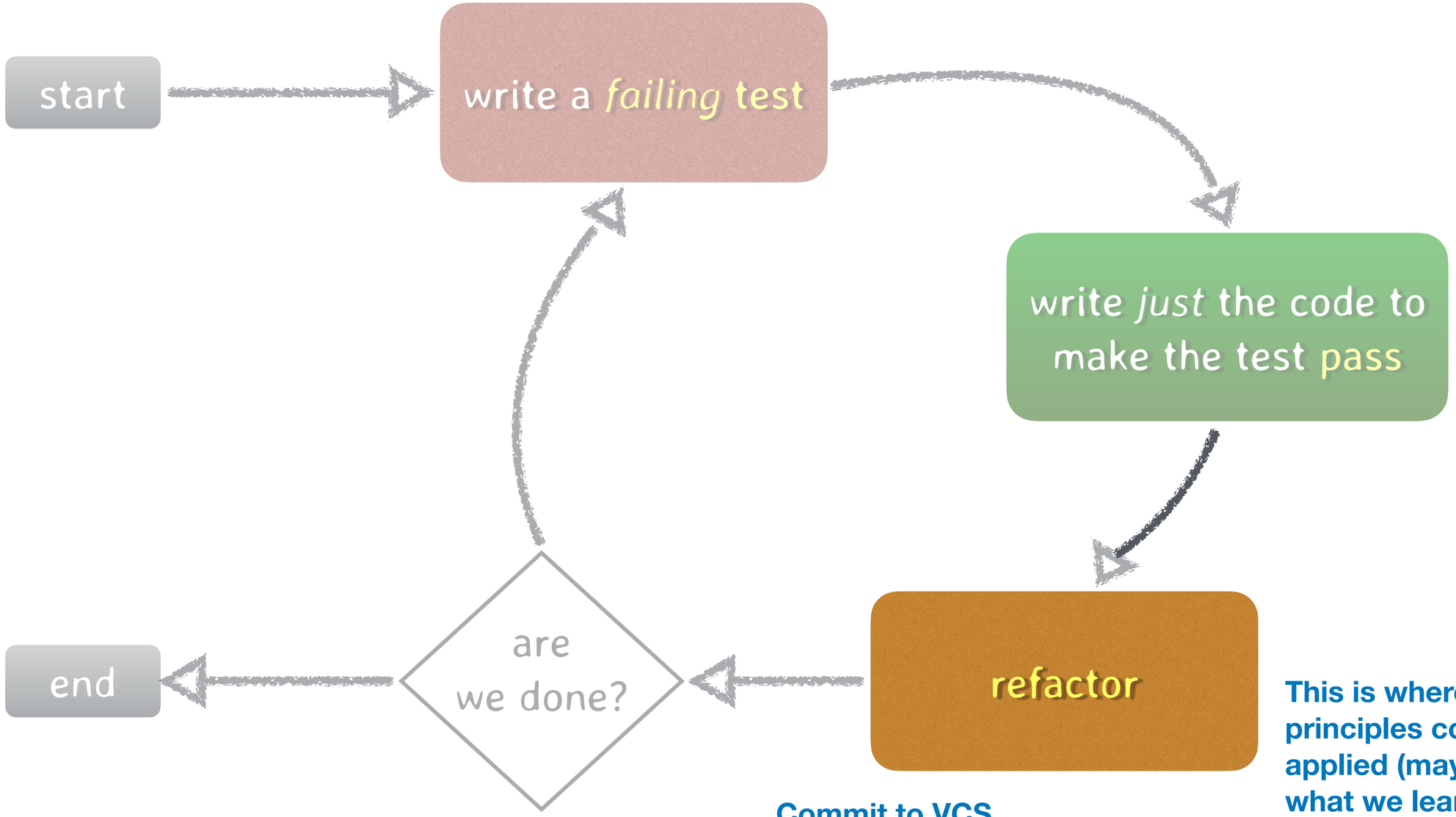


The TDD Cycle

pt II

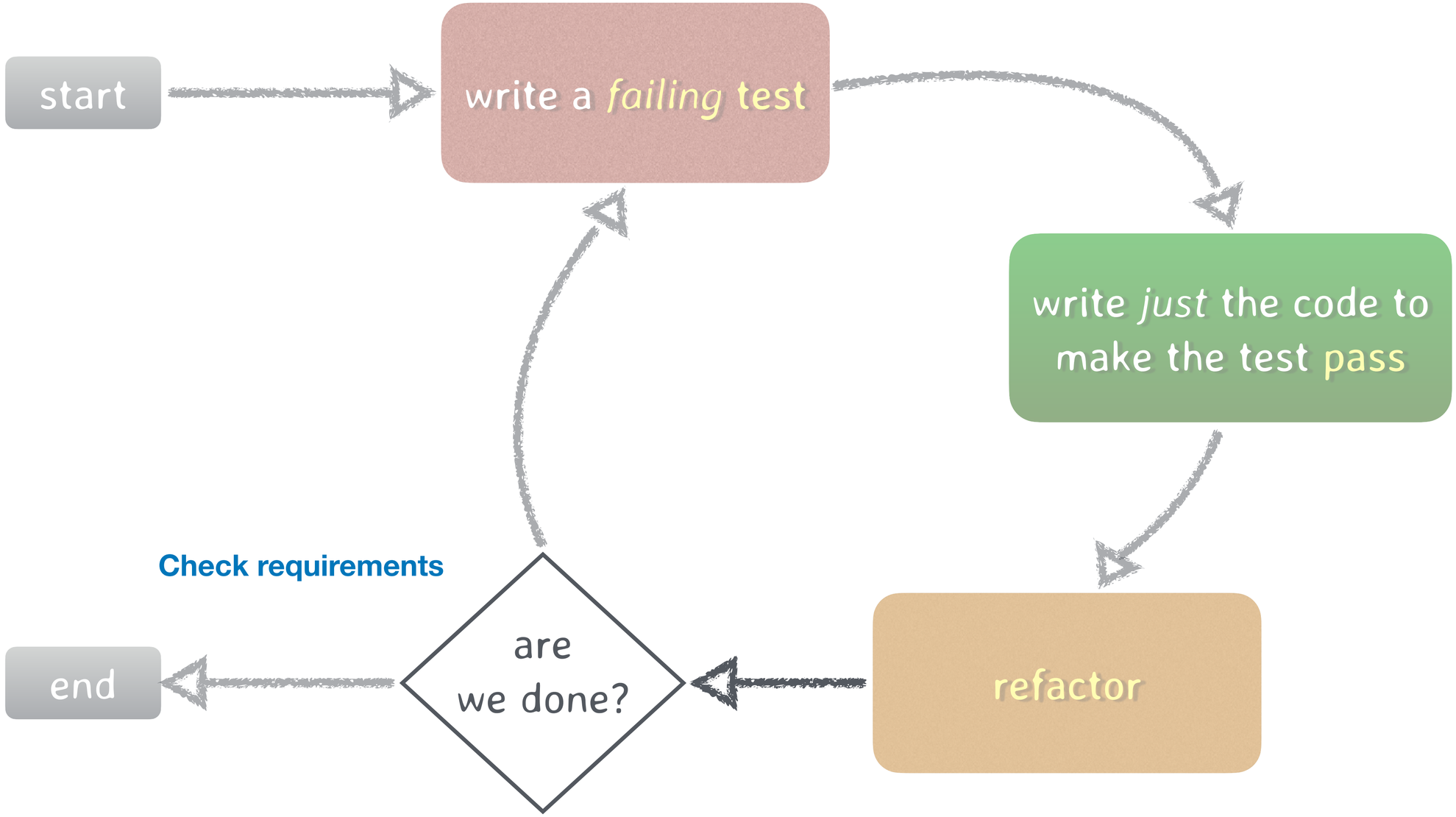

good tests:
• single assert,
• low cyclomatic complexity
• force next generalisation
start
write a failing test
Select next requirement
write just the code to make the test pass
refactor
are we done?
end

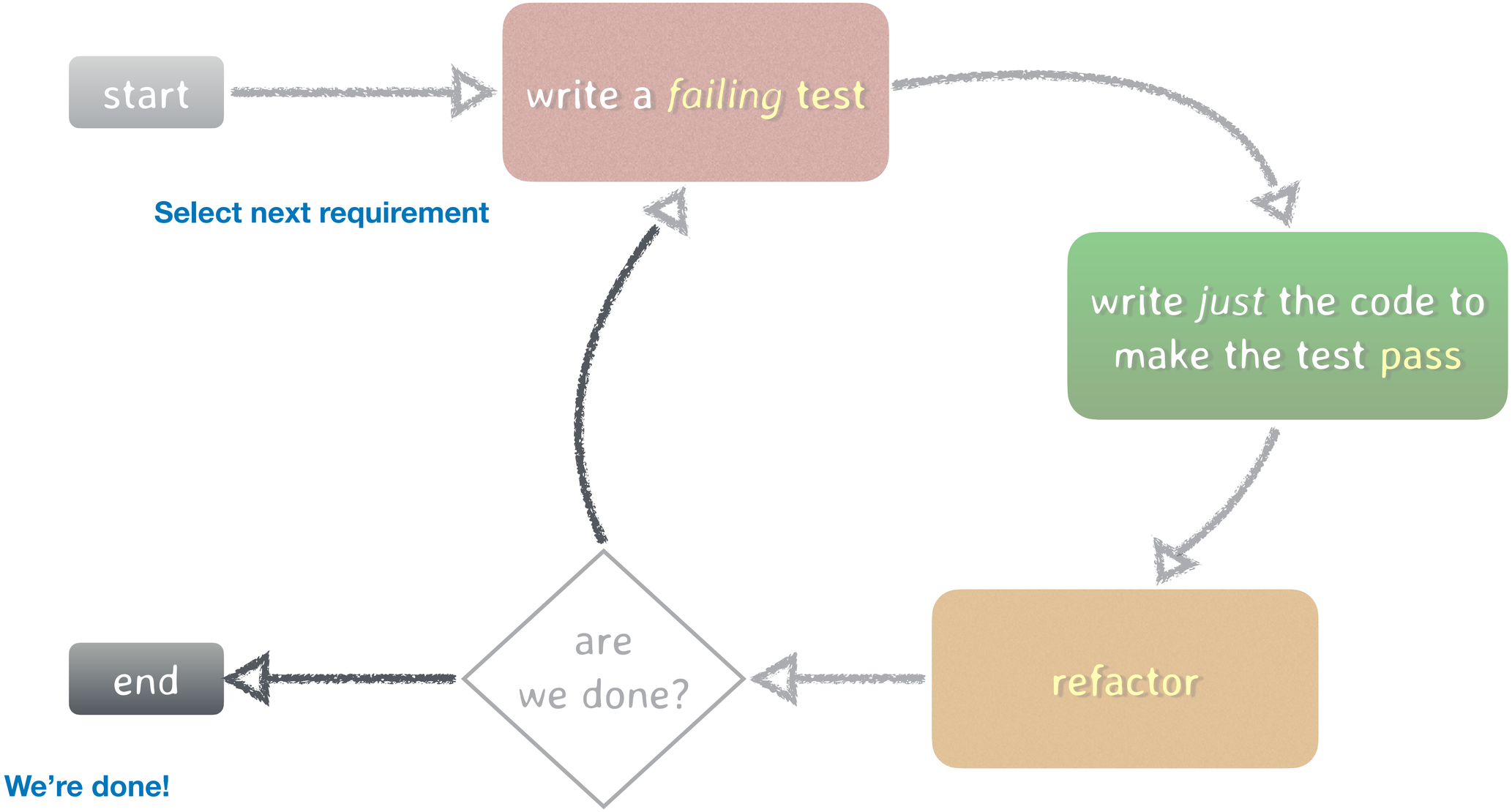

start
write a failing test
what does this tell you?
- need to generalise?
- hard due to coupling?
write just the code to make the test pass
Commit to VCS?
refactor
are we done?
end



Commit to VCS

This is where the design principles come in and are applied (maybe based on what we learnt at green)


start
write a failing test
write just the code to make the test pass
refactor
are we done?
Check requirements
end


start
write a failing test
Select next requirement
write just the code to make the test pass
refactor
are we done?
end
We're done!

S

O

L

D

I

S
O
L
I
D

S O L I D

The SOLID principles

Single responsibility
Open/ Closed
Liskov Substitution
Interface Segregation
Dependency Inversion

Single responsibility

Dependency Inversion

Seperation of Concerns

Dependency Inversion

Seperation of Concerns

Dependency Inversion

Seperation of Concerns

Program to an Interface

Share a Coke with
Kevlin
FROME VALLEY
HEREFORDSHIRE
HENNEY'S
DRY CIDER
MADE FROM 100% FRESH PRESSED JUICE
TRADE MARK
SINGLETON
Malt Scotch
of Duff
DUFFTOWN
STD 1896

Functional

Loose

Unit Testable

Introspective

‘**D**empotent

Seperation of Concerns

Dependency Inversion

The Cat Flap Principle

Who invented the cat flap?

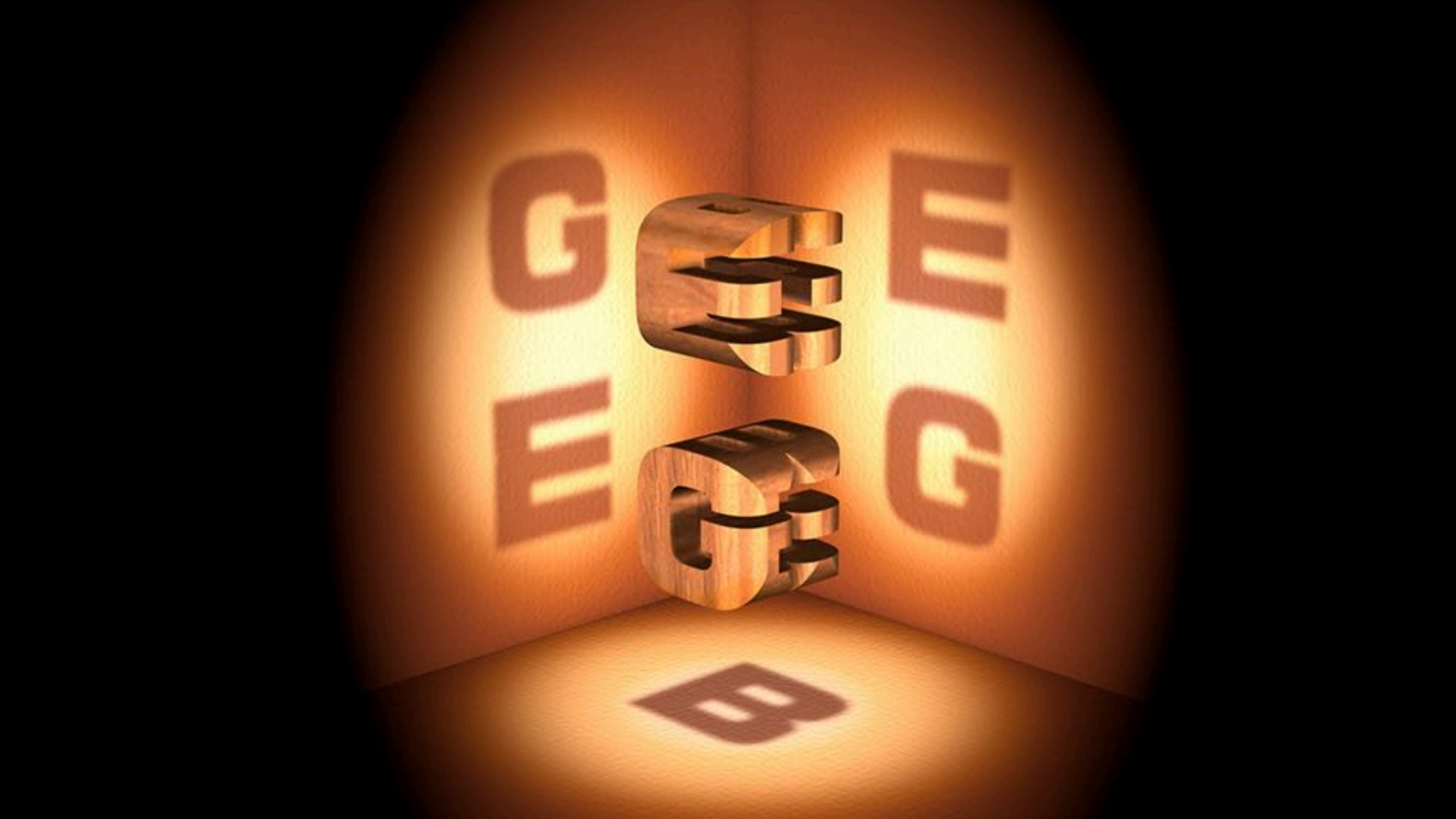
It is a rare mind indeed that can render the hitherto nonexistent blindingly obvious

- Dirk Gently

Everything should
be made as simple
as possible,
but no simpler

Einstein's Razor

You're Not as Smart as You Think You Are


G
E
E
G
B

It always takes longer than you expect, even when you take into account Hofstadter's Law

- Hofstadter's Law

You're not as smart as you think you are, even when you take into account Phil's Law

- Phil's Law

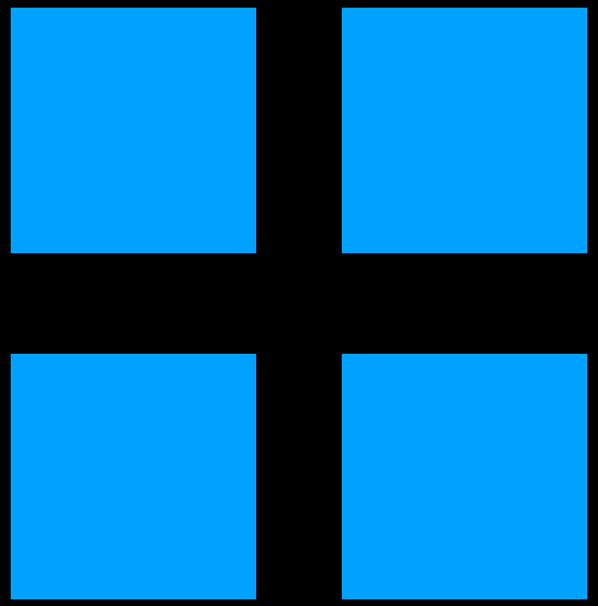
So how bad is it?

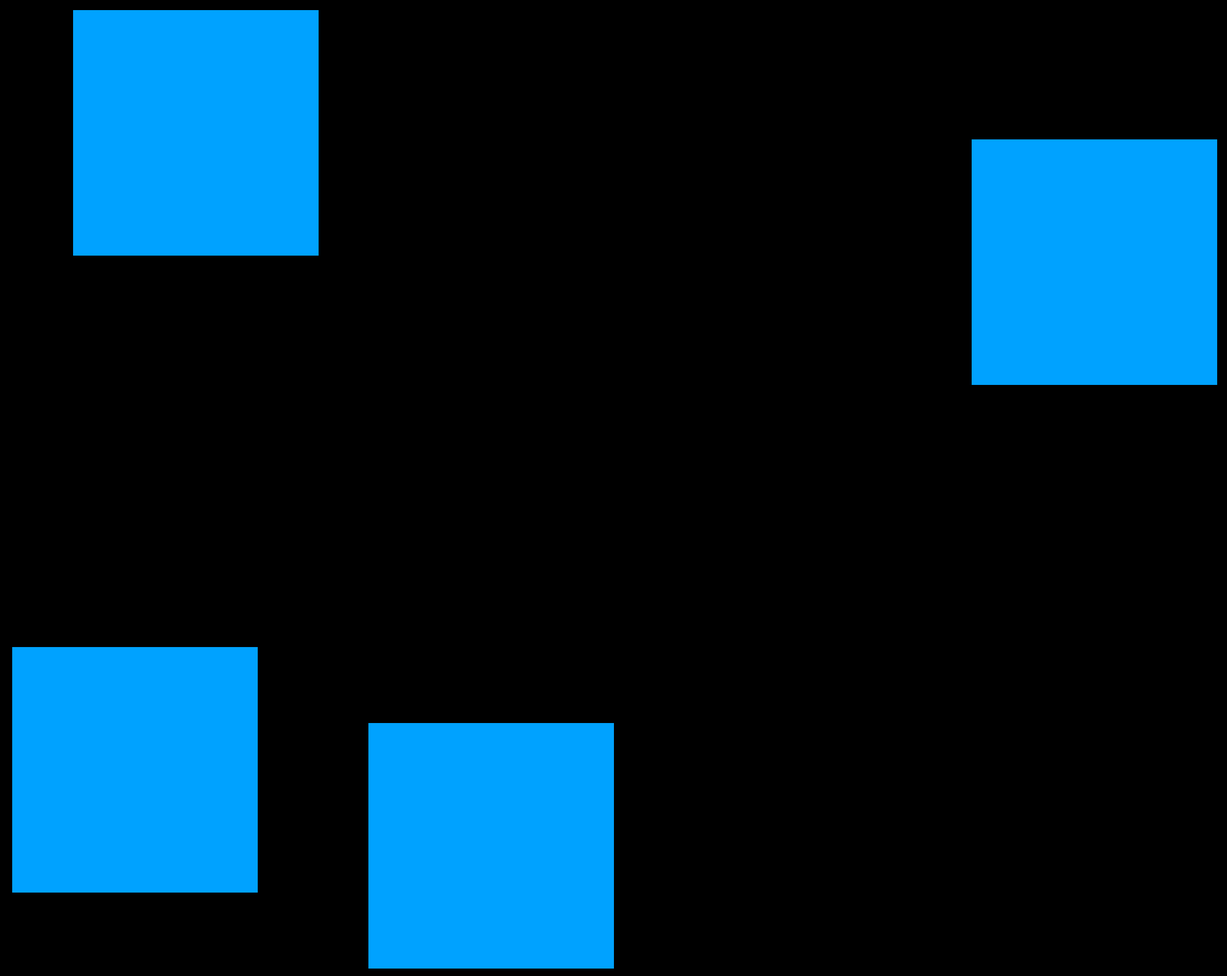
Magic numbers

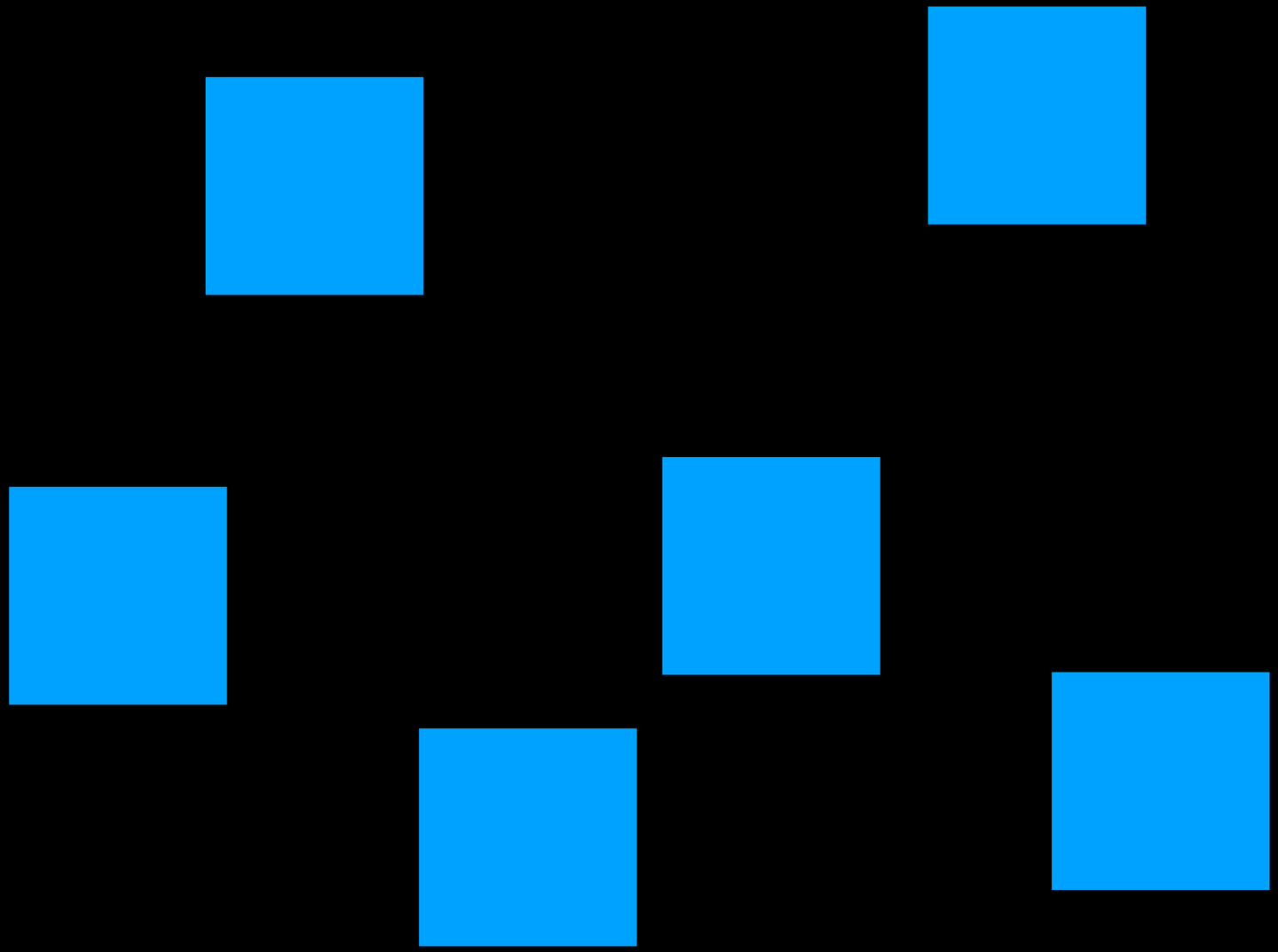
Magic number 7 (+/- 2)

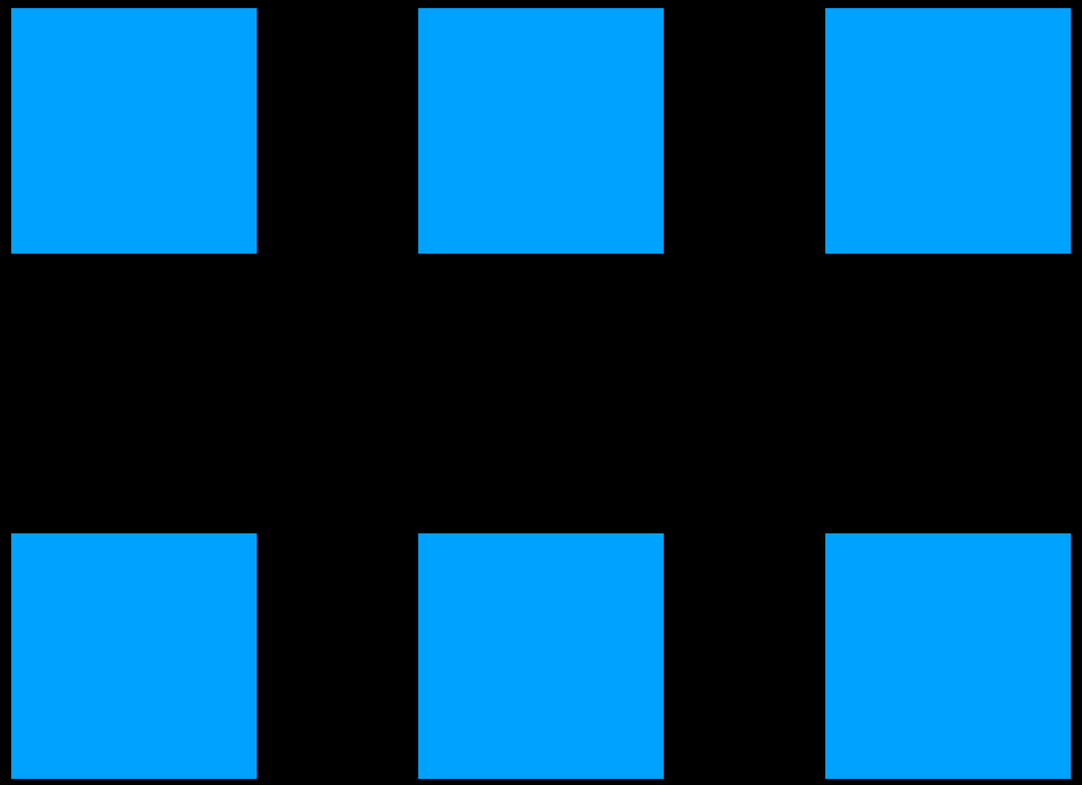
Magic number 7

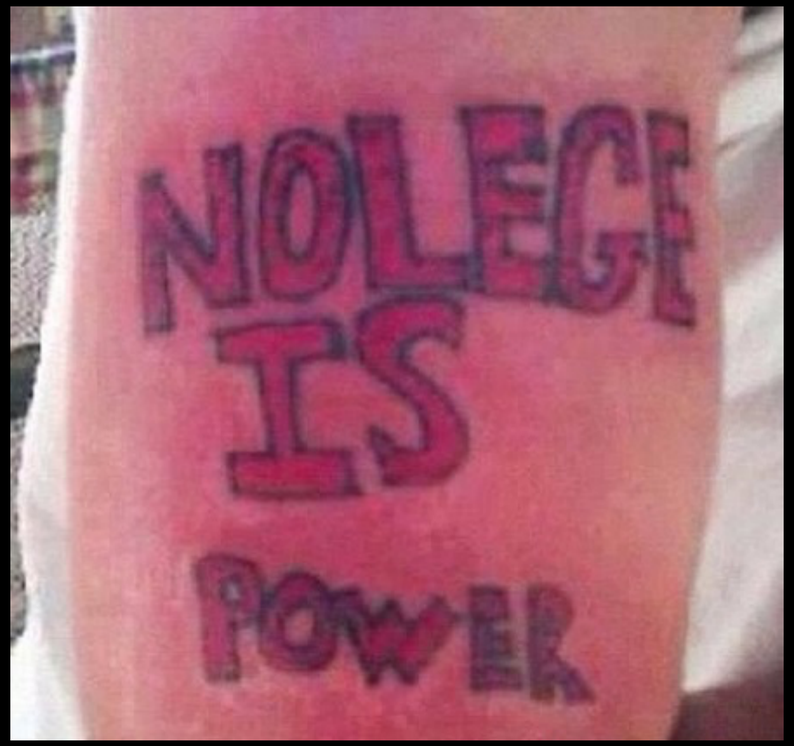
(+/- 2)










NOLEGE
IS
POWER

NO REGRETS

me
marry
you
will
Brittany

plex

complex

complex

many folds/
braids

simplex

one fold/ braid

complex

many folds/
braids

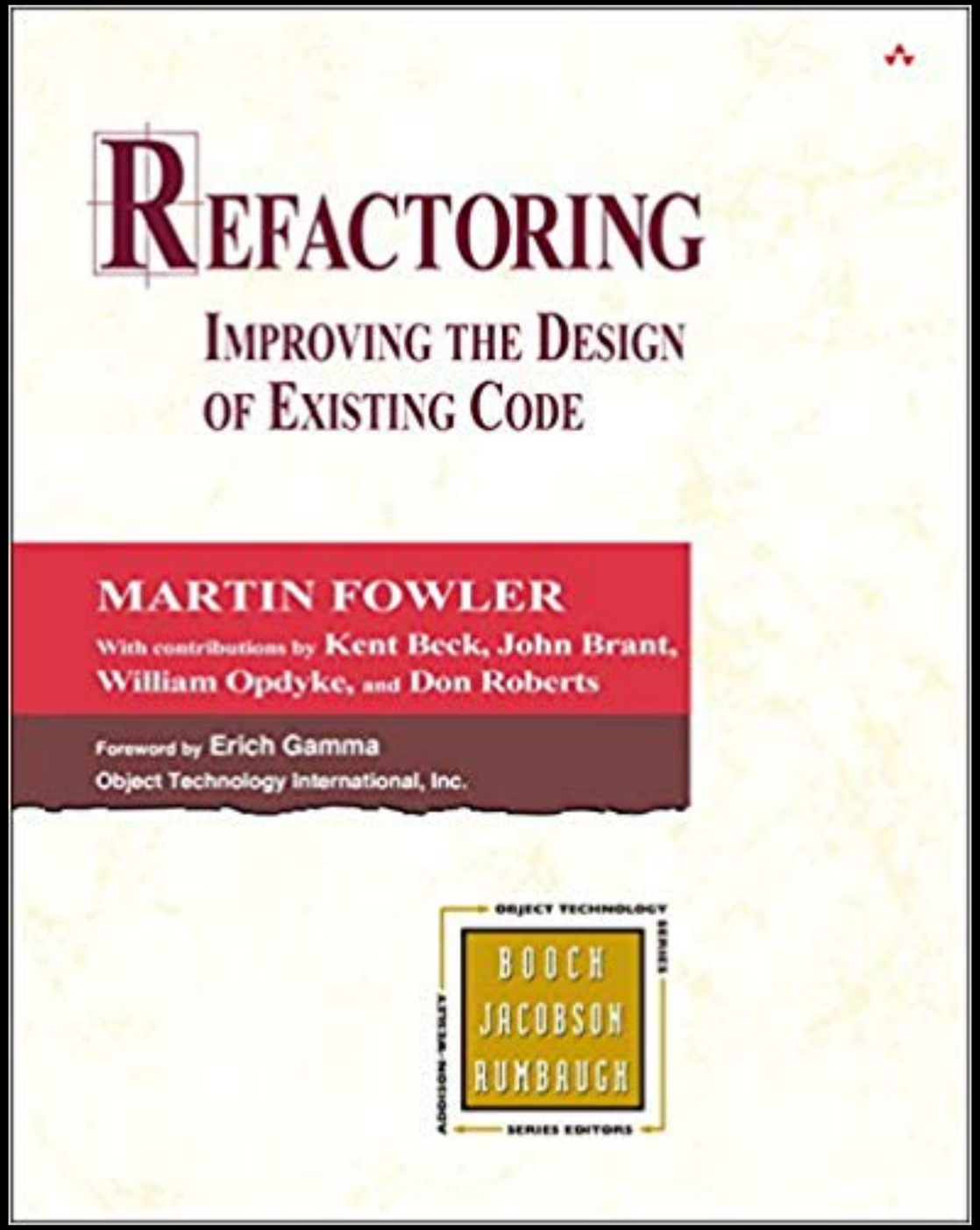
simple

one fold/ braid

complex

many folds/
braids

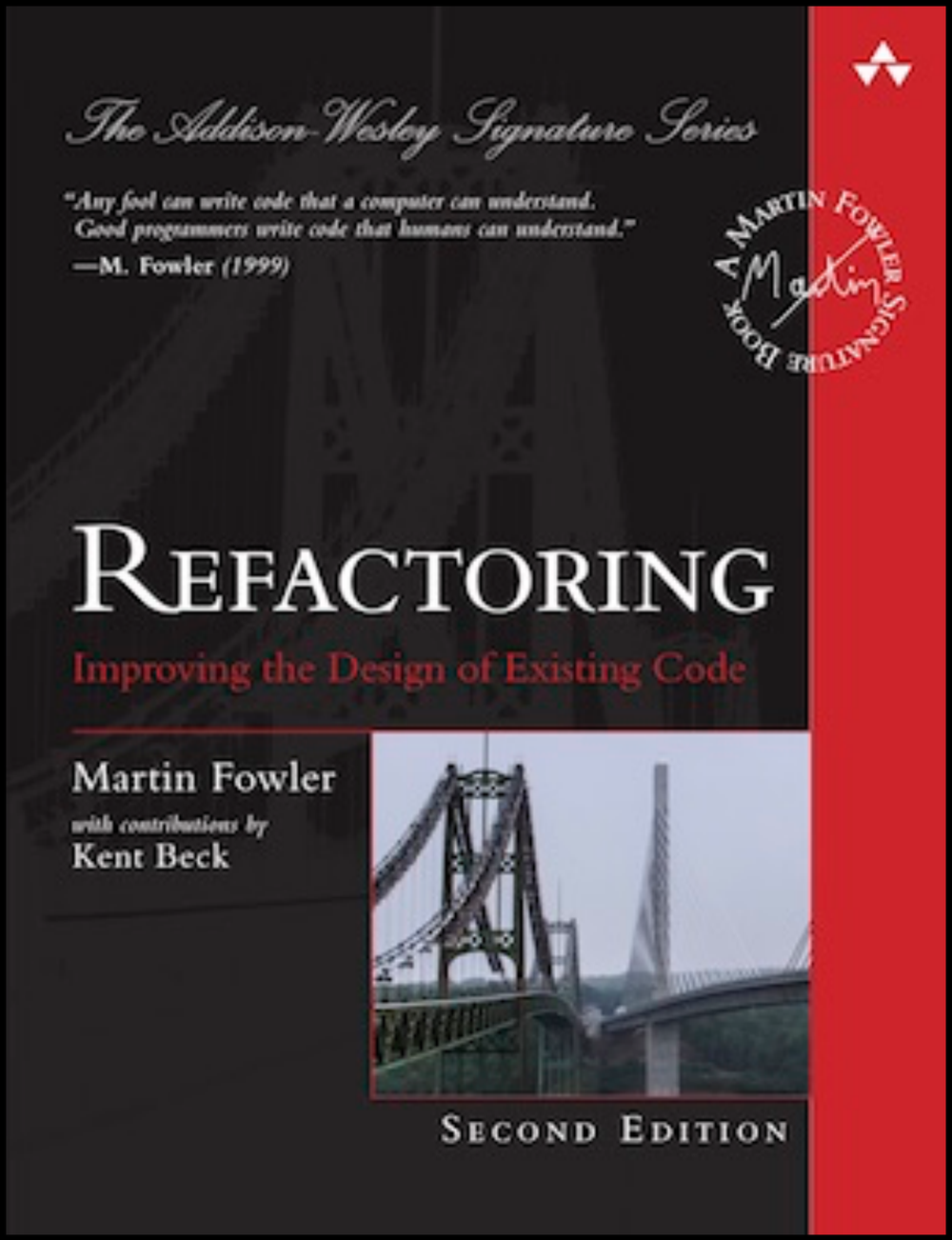
Fractal TDD


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International, Inc.
OBJECT TECHNOLOGY SERIES
BOOCH
JACOBSON
RUMBAUGH
ADDISON-WESLEY
SERIES EDITORS

Chris Oldwood @chrisoldwood · Apr 12

"I was in a bookshop recently and had a flick through the 2nd edition of Martin Fowler's Refactoring. I have to say I was slightly disappointed when I discovered it had exactly the same content but the chapters were just in a different order."

1 2


The Addison-Wesley Signature Series
"Any fool can write code that a computer can understand.
Good programmers write code that humans can understand."
—M. Fowler (1999)
A Martin Fowler Signature Book
Martin
Refactoring
Improving the Design of Existing Code
Martin Fowler
with contributions by
Kent Beck
Second Edition

> That's nearly twenty years ago, and the technique is now more widely known, although often not executed as well as it should be. The book has also held up pretty well, and I think you can take this old book and still learn how to refactor pretty much as well as you could all those years ago. But the book shows its age, with wrinkles like the use of java.util.Vector.

Martin Fowler

“

At the time I wrote it, it was becoming mainstream to consider classes as the dominant structuring the mechanism for code. These days, however, we see other structures playing a greater role. Classes still are valuable, in my view, but our refactoring needs to be less centered around them, realizing that they can come and go as code is trained into new shapes

Martin Fowler