

+ 23

# *Back To Basics*

## Testing

PHIL NASH



**Cppcon**  
The C++ Conference

20  
23



October 01 - 06



# **What is a test?**

# what is a test?

operation

# what is a test?

operation

+

input values

# what is a test?

input values

operation

+

expected  
output values

# what is a test?

operation

input values

expected  
output values

```
ASSERT_EQ( add(1, 2), 3 );
```

# what is a test?

operation

input values

expected  
output values

I run my program

I type in some input

I expect it to show specific output



# what is a test?

operation

input values

expected  
output values

Invoke the compiler

give it my code

It compiles successfully



# What is a test?



# what is a test?

Unit

Integration

System

**what is a  
test?**

**System Tests**

**Integration Tests**

**Unit Tests**

**what is a  
test?**

The  
testing  
pyramid

**System Tests**

**Integration Tests**

**Unit Tests**

what is a  
**Unit Test?**

# What is a Unit Test?

Michael Feathers

“A test is **not** a unit test if:

- ▶ It talks to the **database**
- ▶ It communicates across the **network**
- ▶ It touches the **file system**
- ▶ It **can't run at the same time** as any of your other unit tests
- ▶ You have to do special things to your **environment** (such as editing config files) to run it ”



*Michael Feathers - A Set of Unit Testing Rules*

# What is a Unit Test?

Michael Feathers

“Tests that do these things aren't bad. Often they are worth writing, and they can be written in a unit test harness.”



*Michael Feathers - A Set of Unit Testing Rules*

# What is a Unit Test?

Michael Feathers

“

Tests that do these things aren't bad. Often they are worth writing, and they can be written in a unit test harness.

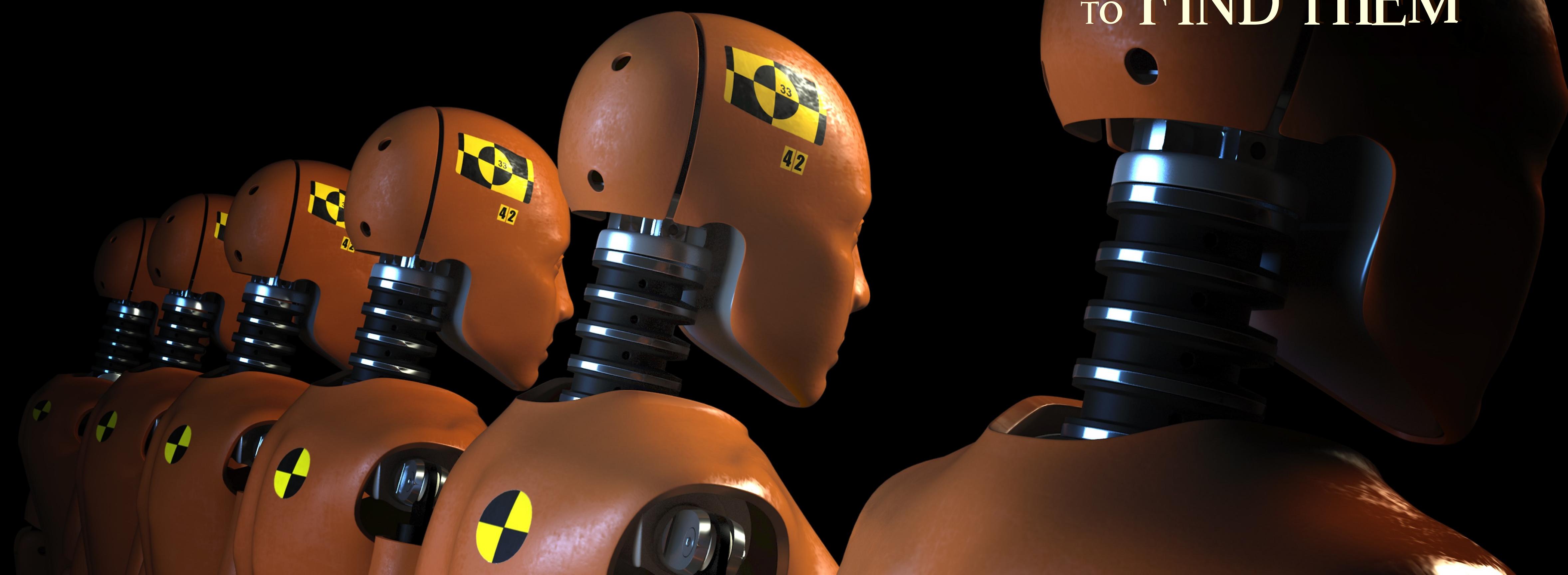
However, it is important to be able to separate them from true unit tests so that we can keep a set of tests that we can **run fast whenever we make our changes**”



*Michael Feathers - A Set of Unit Testing Rules*

# 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>

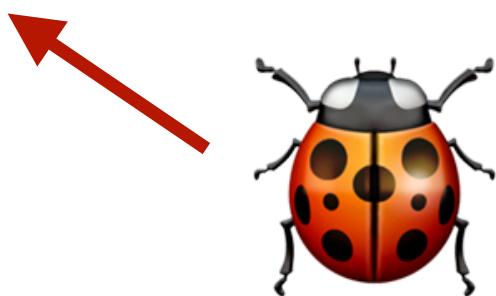
int add( int a, int b );

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



# WIKIPEDIA

The Free Encyclopedia

Main page  
Contents  
Featured content  
Current events  
Random article  
Donate to Wikipedia  
Wikipedia store

Interaction  
Help  
About Wikipedia  
Community portal  
Recent changes  
Contact page

Tools  
What links here  
Related changes  
Upload file  
Special pages  
Permanent link  
Page information  
Wikidata item  
Cite this page

## List of unit testing frameworks

From Wikipedia, the free encyclopedia



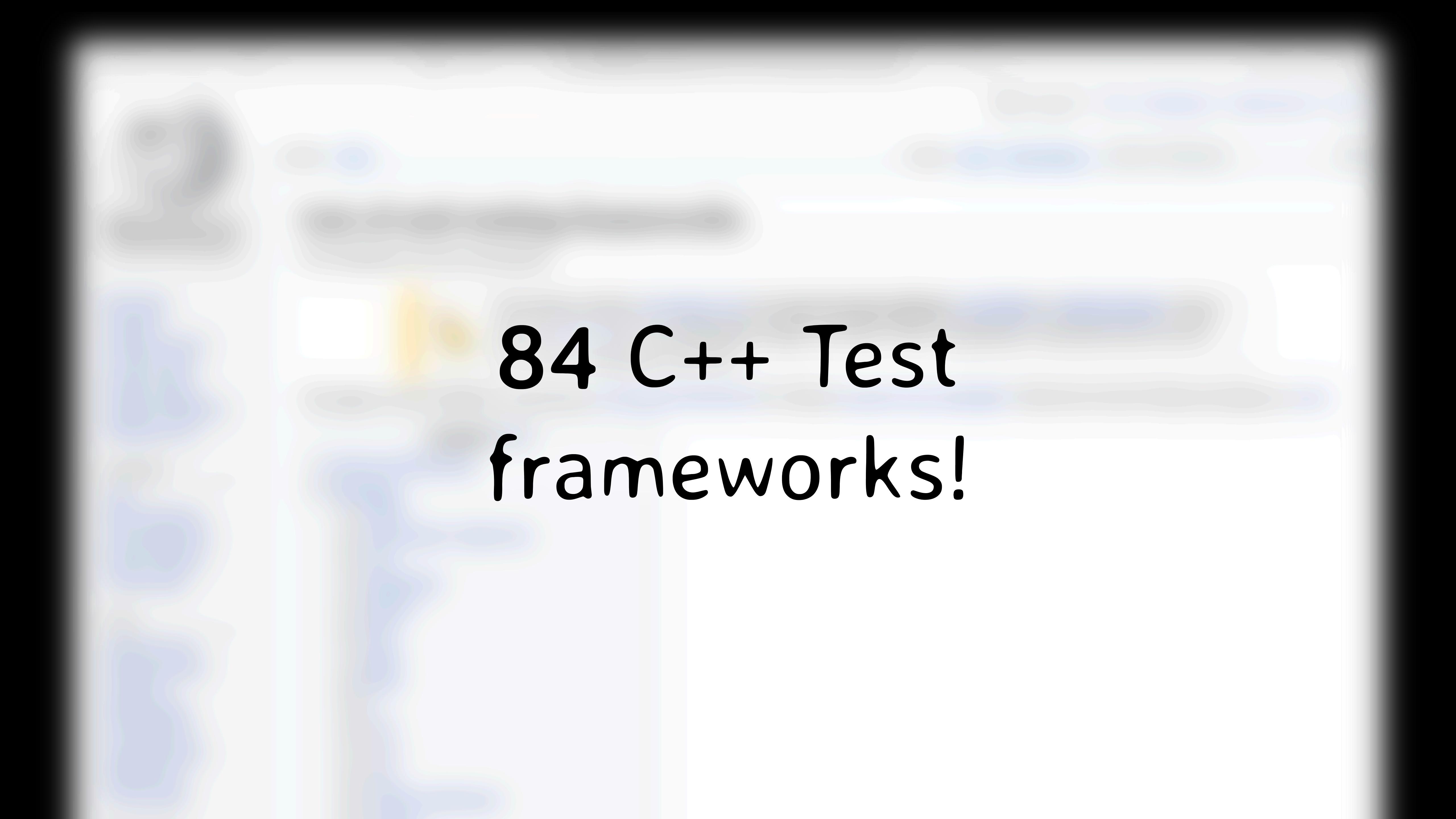
This article contains [embedded lists](#) that **may be poorly defined, unverified or indiscriminate**. Please help to [clean it up](#) to meet Wikipedia's quality standards. Where appropriate, incorporate items into the main body of the article. (April 2018)

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 \(ColdFusion\)](#)
  - 2.14 [CMake](#)
  - 2.15 [CMake](#)
  - 2.16 [CMake](#)
  - 2.17 [CMake](#)
  - 2.18 [CMake](#)
  - 2.19 [CMake](#)
  - 2.20 [CMake](#)
  - 2.21 [CMake](#)
  - 2.22 [CMake](#)
  - 2.23 [CMake](#)
  - 2.24 [CMake](#)
  - 2.25 [CMake](#)
  - 2.26 [CMake](#)
  - 2.27 [CMake](#)
  - 2.28 [CMake](#)
  - 2.29 [CMake](#)
  - 2.30 [CMake](#)
  - 2.31 [CMake](#)
  - 2.32 [CMake](#)
  - 2.33 [CMake](#)
  - 2.34 [CMake](#)
  - 2.35 [CMake](#)
  - 2.36 [CMake](#)
  - 2.37 [CMake](#)
  - 2.38 [CMake](#)
  - 2.39 [CMake](#)
  - 2.40 [CMake](#)
  - 2.41 [CMake](#)
  - 2.42 [CMake](#)
  - 2.43 [CMake](#)
  - 2.44 [CMake](#)
  - 2.45 [CMake](#)
  - 2.46 [CMake](#)
  - 2.47 [CMake](#)
  - 2.48 [CMake](#)
  - 2.49 [CMake](#)
  - 2.50 [CMake](#)
  - 2.51 [CMake](#)
  - 2.52 [CMake](#)
  - 2.53 [CMake](#)
  - 2.54 [CMake](#)
  - 2.55 [CMake](#)
  - 2.56 [CMake](#)
  - 2.57 [CMake](#)
  - 2.58 [CMake](#)
  - 2.59 [CMake](#)
  - 2.60 [CMake](#)
  - 2.61 [CMake](#)
  - 2.62 [CMake](#)
  - 2.63 [CMake](#)
  - 2.64 [CMake](#)
  - 2.65 [CMake](#)
  - 2.66 [CMake](#)
  - 2.67 [CMake](#)
  - 2.68 [CMake](#)
  - 2.69 [CMake](#)
  - 2.70 [CMake](#)
  - 2.71 [CMake](#)
  - 2.72 [CMake](#)
  - 2.73 [CMake](#)
  - 2.74 [CMake](#)
  - 2.75 [CMake](#)
  - 2.76 [CMake](#)
  - 2.77 [CMake](#)
  - 2.78 [CMake](#)
  - 2.79 [CMake](#)
  - 2.80 [CMake](#)
  - 2.81 [CMake](#)
  - 2.82 [CMake](#)
  - 2.83 [CMake](#)
  - 2.84 [CMake](#)
  - 2.85 [CMake](#)
  - 2.86 [CMake](#)
  - 2.87 [CMake](#)
  - 2.88 [CMake](#)
  - 2.89 [CMake](#)
  - 2.90 [CMake](#)
  - 2.91 [CMake](#)
  - 2.92 [CMake](#)
  - 2.93 [CMake](#)
  - 2.94 [CMake](#)
  - 2.95 [CMake](#)
  - 2.96 [CMake](#)
  - 2.97 [CMake](#)
  - 2.98 [CMake](#)
  - 2.99 [CMake](#)
  - 2.100 [CMake](#)
  - 2.101 [CMake](#)
  - 2.102 [CMake](#)
  - 2.103 [CMake](#)
  - 2.104 [CMake](#)
  - 2.105 [CMake](#)
  - 2.106 [CMake](#)
  - 2.107 [CMake](#)
  - 2.108 [CMake](#)
  - 2.109 [CMake](#)
  - 2.110 [CMake](#)
  - 2.111 [CMake](#)
  - 2.112 [CMake](#)
  - 2.113 [CMake](#)
  - 2.114 [CMake](#)
  - 2.115 [CMake](#)
  - 2.116 [CMake](#)
  - 2.117 [CMake](#)
  - 2.118 [CMake](#)
  - 2.119 [CMake](#)
  - 2.120 [CMake](#)
  - 2.121 [CMake](#)
  - 2.122 [CMake](#)
  - 2.123 [CMake](#)
  - 2.124 [CMake](#)
  - 2.125 [CMake](#)
  - 2.126 [CMake](#)
  - 2.127 [CMake](#)
  - 2.128 [CMake](#)
  - 2.129 [CMake](#)
  - 2.130 [CMake](#)
  - 2.131 [CMake](#)
  - 2.132 [CMake](#)
  - 2.133 [CMake](#)
  - 2.134 [CMake](#)
  - 2.135 [CMake](#)
  - 2.136 [CMake](#)
  - 2.137 [CMake](#)
  - 2.138 [CMake](#)
  - 2.139 [CMake](#)
  - 2.140 [CMake](#)
  - 2.141 [CMake](#)
  - 2.142 [CMake](#)
  - 2.143 [CMake](#)
  - 2.144 [CMake](#)
  - 2.145 [CMake](#)
  - 2.146 [CMake](#)
  - 2.147 [CMake](#)
  - 2.148 [CMake](#)
  - 2.149 [CMake](#)
  - 2.150 [CMake](#)
  - 2.151 [CMake](#)
  - 2.152 [CMake](#)
  - 2.153 [CMake](#)
  - 2.154 [CMake](#)
  - 2.155 [CMake](#)
  - 2.156 [CMake](#)
  - 2.157 [CMake](#)
  - 2.158 [CMake](#)
  - 2.159 [CMake](#)
  - 2.160 [CMake](#)
  - 2.161 [CMake](#)
  - 2.162 [CMake](#)
  - 2.163 [CMake](#)
  - 2.164 [CMake](#)
  - 2.165 [CMake](#)
  - 2.166 [CMake](#)
  - 2.167 [CMake](#)
  - 2.168 [CMake](#)
  - 2.169 [CMake](#)
  - 2.170 [CMake](#)
  - 2.171 [CMake](#)
  - 2.172 [CMake](#)
  - 2.173 [CMake](#)
  - 2.174 [CMake](#)
  - 2.175 [CMake](#)
  - 2.176 [CMake](#)
  - 2.177 [CMake](#)
  - 2.178 [CMake](#)
  - 2.179 [CMake](#)
  - 2.180 [CMake](#)
  - 2.181 [CMake](#)
  - 2.182 [CMake](#)
  - 2.183 [CMake](#)
  - 2.184 [CMake](#)
  - 2.185 [CMake](#)
  - 2.186 [CMake](#)
  - 2.187 [CMake](#)
  - 2.188 [CMake](#)
  - 2.189 [CMake](#)
  - 2.190 [CMake](#)
  - 2.191 [CMake](#)
  - 2.192 [CMake](#)
  - 2.193 [CMake](#)
  - 2.194 [CMake](#)
  - 2.195 [CMake](#)
  - 2.196 [CMake](#)
  - 2.197 [CMake](#)
  - 2.198 [CMake](#)
  - 2.199 [CMake](#)
  - 2.200 [CMake](#)
  - 2.201 [CMake](#)
  - 2.202 [CMake](#)
  - 2.203 [CMake](#)
  - 2.204 [CMake](#)
  - 2.205 [CMake](#)
  - 2.206 [CMake](#)
  - 2.207 [CMake](#)
  - 2.208 [CMake](#)
  - 2.209 [CMake](#)
  - 2.210 [CMake](#)
  - 2.211 [CMake](#)
  - 2.212 [CMake](#)
  - 2.213 [CMake](#)
  - 2.214 [CMake](#)
  - 2.215 [CMake](#)
  - 2.216 [CMake](#)
  - 2.217 [CMake](#)
  - 2.218 [CMake](#)
  - 2.219 [CMake](#)
  - 2.220 [CMake](#)
  - 2.221 [CMake](#)
  - 2.222 [CMake](#)
  - 2.223 [CMake](#)
  - 2.224 [CMake](#)
  - 2.225 [CMake](#)
  - 2.226 [CMake](#)
  - 2.227 [CMake](#)
  - 2.228 [CMake](#)
  - 2.229 [CMake](#)
  - 2.230 [CMake](#)
  - 2.231 [CMake](#)
  - 2.232 [CMake](#)
  - 2.233 [CMake](#)
  - 2.234 [CMake](#)
  - 2.235 [CMake](#)
  - 2.236 [CMake](#)
  - 2.237 [CMake](#)
  - 2.238 [CMake](#)
  - 2.239 [CMake](#)
  - 2.240 [CMake](#)
  - 2.241 [CMake](#)
  - 2.242 [CMake](#)
  - 2.243 [CMake](#)
  - 2.244 [CMake](#)
  - 2.245 [CMake](#)
  - 2.246 [CMake](#)
  - 2.247 [CMake](#)
  - 2.248 [CMake](#)
  - 2.249 [CMake](#)
  - 2.250 [CMake](#)
  - 2.251 [CMake](#)
  - 2.252 [CMake](#)
  - 2.253 [CMake](#)
  - 2.254 [CMake](#)
  - 2.255 [CMake](#)
  - 2.256 [CMake](#)
  - 2.257 [CMake](#)
  - 2.258 [CMake](#)
  - 2.259 [CMake](#)
  - 2.260 [CMake](#)
  - 2.261 [CMake](#)
  - 2.262 [CMake](#)
  - 2.263 [CMake](#)
  - 2.264 [CMake](#)
  - 2.265 [CMake](#)
  - 2.266 [CMake](#)
  - 2.267 [CMake](#)
  - 2.268 [CMake](#)
  - 2.269 [CMake](#)
  - 2.270 [CMake](#)
  - 2.271 [CMake](#)
  - 2.272 [CMake](#)
  - 2.273 [CMake](#)
  - 2.274 [CMake](#)
  - 2.275 [CMake](#)
  - 2.276 [CMake](#)
  - 2.277 [CMake](#)
  - 2.278 [CMake](#)
  - 2.279 [CMake](#)
  - 2.280 [CMake](#)
  - 2.281 [CMake](#)
  - 2.282 [CMake](#)
  - 2.283 [CMake](#)
  - 2.284 [CMake](#)
  - 2.285 [CMake](#)
  - 2.286 [CMake](#)
  - 2.287 [CMake](#)
  - 2.288 [CMake](#)
  - 2.289 [CMake](#)
  - 2.290 [CMake](#)
  - 2.291 [CMake](#)
  - 2.292 [CMake](#)
  - 2.293 [CMake](#)
  - 2.294 [CMake](#)
  - 2.295 [CMake](#)
  - 2.296 [CMake](#)
  - 2.297 [CMake](#)
  - 2.298 [CMake](#)
  - 2.299 [CMake](#)
  - 2.300 [CMake](#)
  - 2.301 [CMake](#)
  - 2.302 [CMake](#)
  - 2.303 [CMake](#)
  - 2.304 [CMake](#)
  - 2.305 [CMake](#)
  - 2.306 [CMake](#)
  - 2.307 [CMake](#)
  - 2.308 [CMake](#)
  - 2.309 [CMake](#)
  - 2.310 [CMake](#)
  - 2.311 [CMake](#)
  - 2.312 [CMake](#)
  - 2.313 [CMake](#)
  - 2.314 [CMake](#)
  - 2.315 [CMake](#)
  - 2.316 [CMake](#)
  - 2.317 [CMake](#)
  - 2.318 [CMake](#)
  - 2.319 [CMake](#)
  - 2.320 [CMake](#)
  - 2.321 [CMake](#)
  - 2.322 [CMake](#)
  - 2.323 [CMake](#)
  - 2.324 [CMake](#)
  - 2.325 [CMake](#)
  - 2.326 [CMake](#)
  - 2.327 [CMake](#)
  - 2.328 [CMake](#)
  - 2.329 [CMake](#)
  - 2.330 [CMake](#)
  - 2.331 [CMake](#)
  - 2.332 [CMake](#)
  - 2.333 [CMake](#)
  - 2.334 [CMake](#)
  - 2.335 [CMake](#)
  - 2.336 [CMake](#)
  - 2.337 [CMake](#)
  - 2.338 [CMake](#)
  - 2.339 [CMake](#)
  - 2.340 [CMake](#)
  - 2.341 [CMake](#)
  - 2.342 [CMake](#)
  - 2.343 [CMake](#)
  - 2.344 [CMake](#)
  - 2.345 [CMake](#)
  - 2.346 [CMake](#)
  - 2.347 [CMake](#)
  - 2.348 [CMake](#)
  - 2.349 [CMake](#)
  - 2.350 [CMake](#)
  - 2.351 [CMake](#)
  - 2.352 [CMake](#)
  - 2.353 [CMake](#)
  - 2.354 [CMake](#)
  - 2.355 [CMake](#)
  - 2.356 [CMake](#)
  - 2.357 [CMake](#)
  - 2.358 [CMake](#)
  - 2.359 [CMake](#)
  - 2.360 [CMake](#)
  - 2.361 [CMake](#)
  - 2.362 [CMake](#)
  - 2.363 [CMake](#)
  - 2.364 [CMake](#)
  - 2.365 [CMake](#)
  - 2.366 [CMake](#)
  - 2.367 [CMake](#)
  - 2.368 [CMake](#)
  - 2.369 [CMake](#)
  - 2.370 [CMake](#)
  - 2.371 [CMake](#)
  - 2.372 [CMake](#)
  - 2.373 [CMake](#)
  - 2.374 [CMake](#)
  - 2.375 [CMake](#)
  - 2.376 [CMake](#)
  - 2.377 [CMake](#)
  - 2.378 [CMake](#)
  - 2.379 [CMake](#)
  - 2.380 [CMake](#)
  - 2.381 [CMake](#)
  - 2.382 [CMake](#)
  - 2.383 [CMake](#)
  - 2.384 [CMake](#)
  - 2.385 [CMake](#)
  - 2.386 [CMake](#)
  - 2.387 [CMake](#)
  - 2.388 [C](#)

## List of unit testing frameworks



84 C++ Test  
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

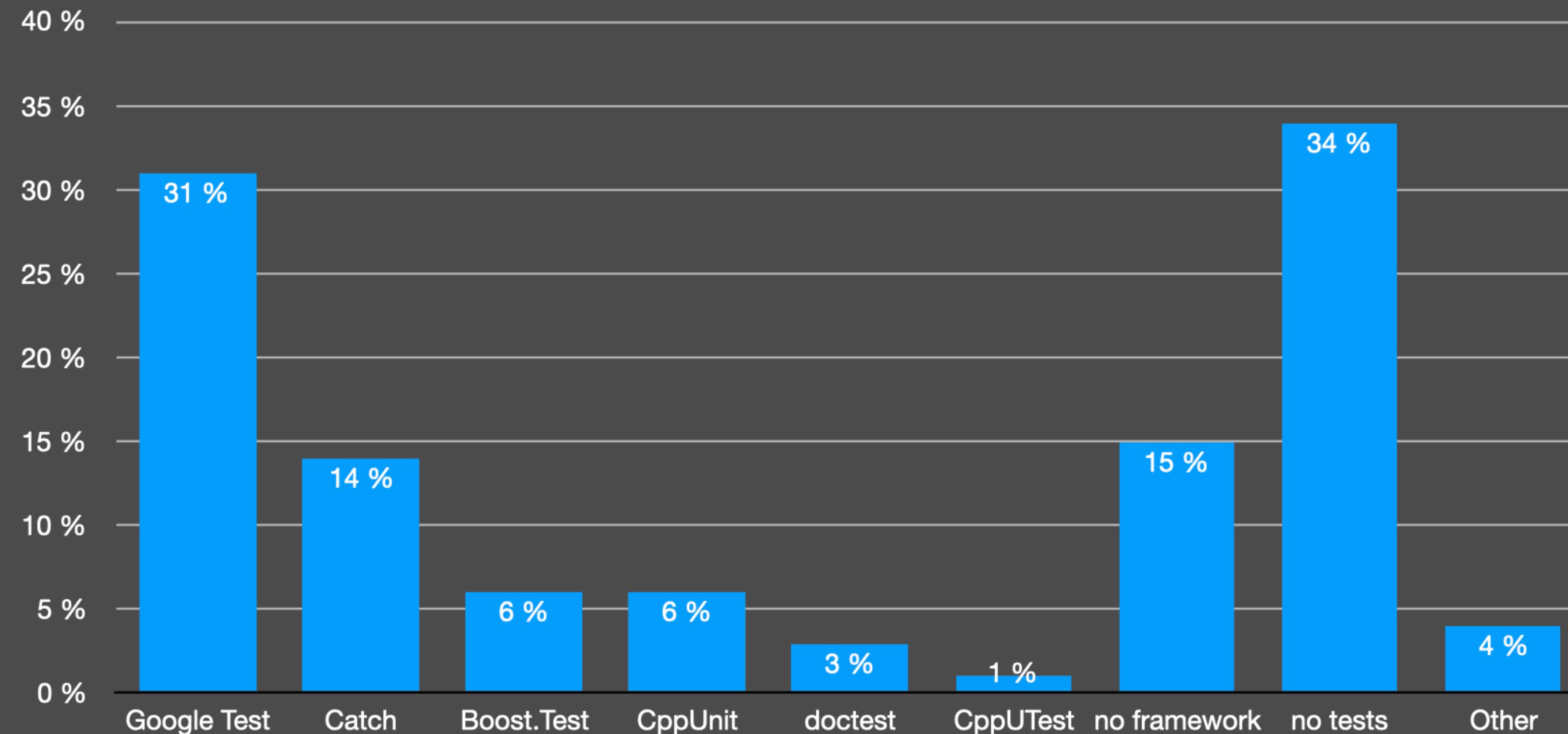
Catch/ Catch2

doctest

UT

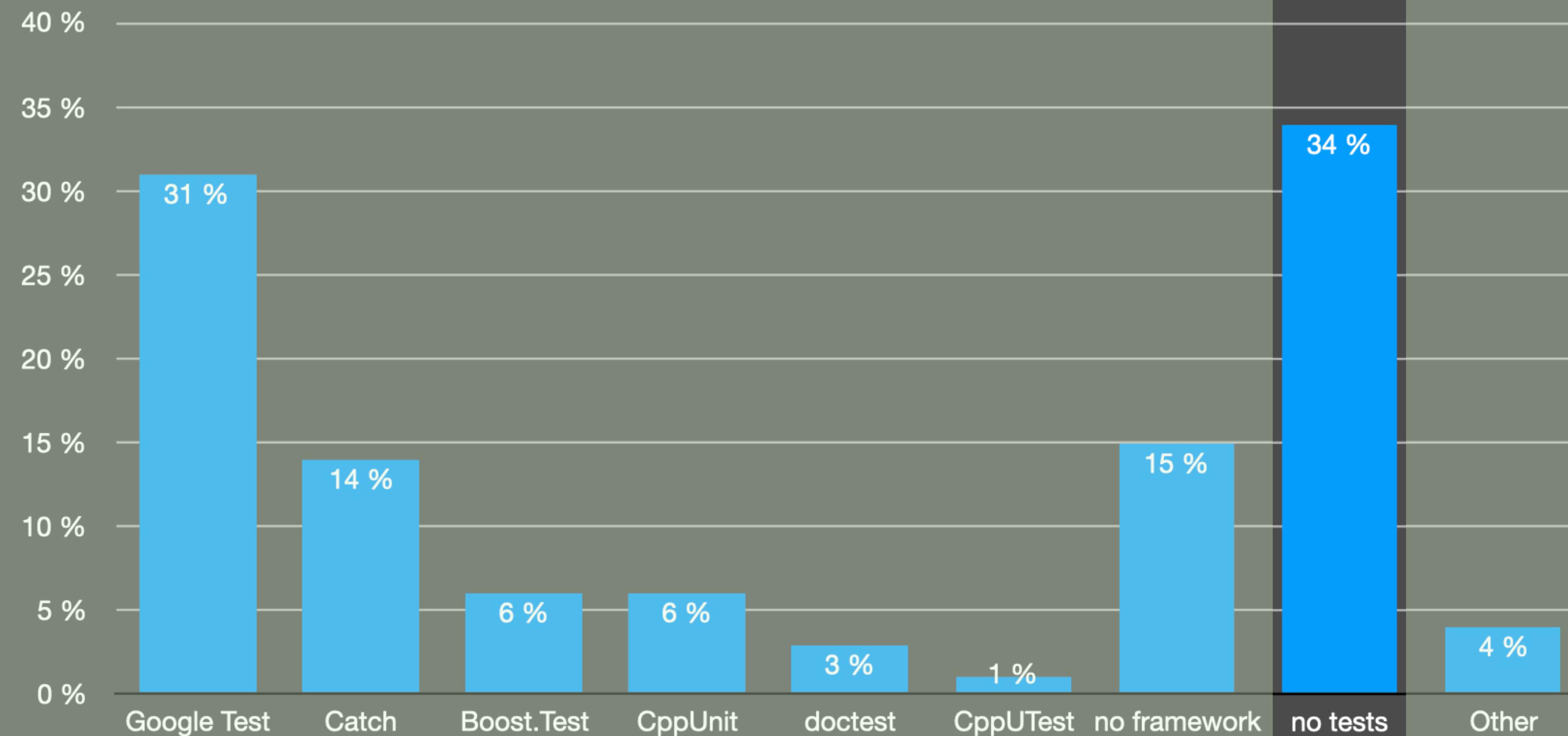
# JetBrains Developer Ecosystem report 2020

## Quality tools: unit testing

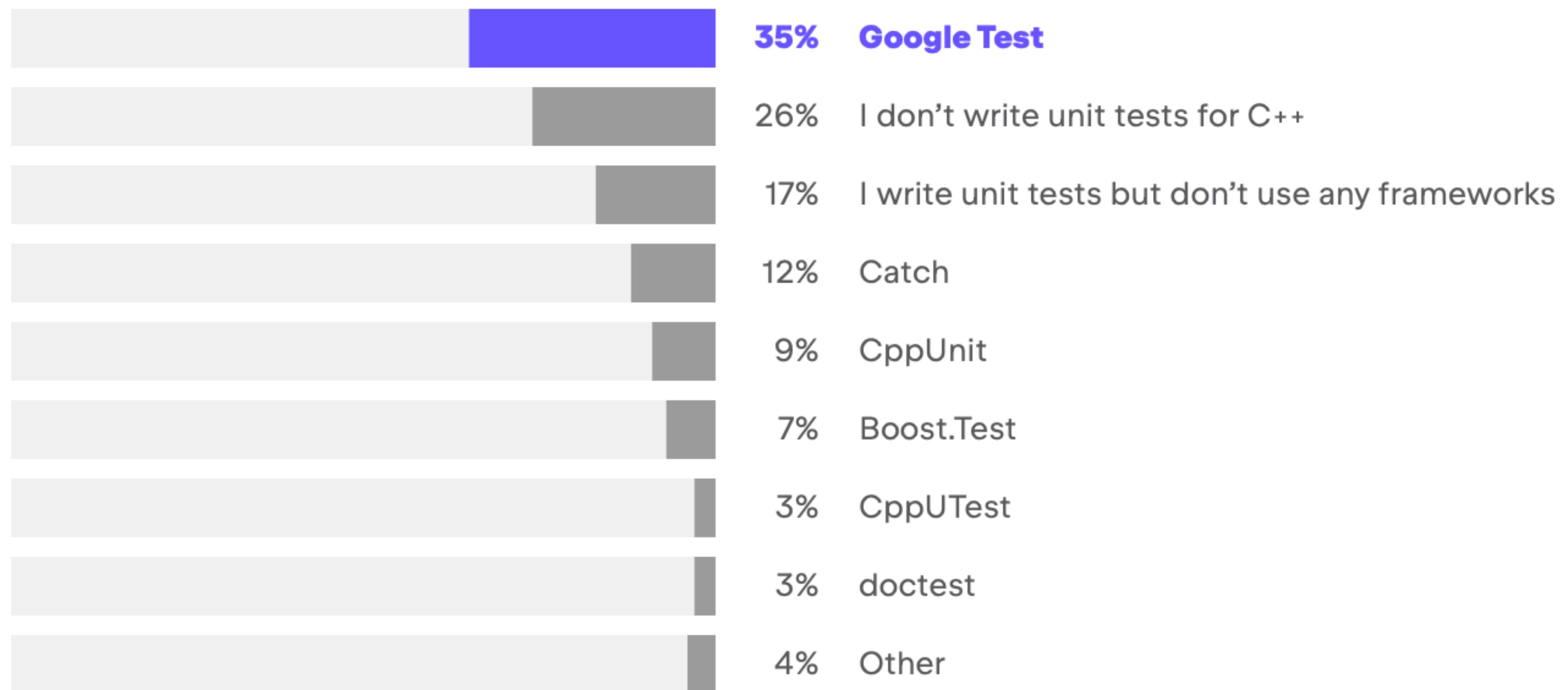


# JetBrains Developer Ecosystem report 2020

## Quality tools: unit testing



# JetBrains Developer Ecosystem report 2022



left-pad



```
std::string left_pad(std::string const& str, size_t min_len, char pad_char=' ');
```

```
std::string left_pad(std::string const& str, size_t min_len, char pad_char=' ');  
  
#include "catch.hpp"  
  
TEST_CASE( "left pad" ) {  
  
    /* .. */  
  
}
```

```
std::string left_pad(std::string const& str, size_t min_len, char pad_char=' ');
```

```
#include "catch.hpp"
```

```
TEST_CASE("left pad") {
```

```
    /* .. */
```

```
}
```

```
#include "gtest/gtest.h"
```

```
TEST(left_pad, empty) {
```

```
    /* .. */
```

```
}
```

```
std::string left_pad(std::string const& str, size_t min_len, char pad_char=' ');

#include "catch.hpp"

TEST_CASE("left_pad pads strings to a minimum length") {
    /* .. */
}

#include "gtest/gtest.h"

TEST(left_pad_pads_strings_to_a_minimum_length, empty) {
    /* .. */
}
```

```
std::string left_pad(std::string const& str, size_t min_len, char pad_char=' ');

#include "catch.hpp"

TEST_CASE("left_pad pads strings to a minimum length") {

    REQUIRE(left_pad("", 0) == "");

}

#include "gtest/gtest.h"

TEST(left_pad_pads_strings_to_a_minimum_length,
     an_empty_string_with_zero_padding) {

    ASSERT_EQ(left_pad("", 0), "");

}


```

```
std::string left_pad(std::string const& str, size_t min_len, char pad_char=' ');  
  
#include "catch.hpp"  
  
TEST_CASE("left_pad pads strings to a minimum length") {  
  
    REQUIRE(left_pad("", 0) == "");  
    REQUIRE(left_pad("abc", 0) == "abc");  
  
}  
  
}
```

```
#include "gtest/gtest.h"  
  
TEST(left_pad_pads_strings_to_a_minimum_length,  
      not_padded_when_min_len_is_less_than_input_string_len) {  
  
    ASSERT_EQ(left_pad("", 0), "");  
    ASSERT_EQ(left_pad("abc", 0), "abc");  
  
}
```

```
std::string left_pad(std::string const& str, size_t min_len, char pad_char=' ');

#include "catch.hpp"

TEST_CASE("left_pad pads strings to a minimum length") {
    SECTION("When padding is < string length, the existing string is returned") {
        REQUIRE(left_pad("", 0) == "");
        REQUIRE(left_pad("abc", 0) == "abc");
    }
}
```

```
#include "gtest/gtest.h"

TEST(left_pad_pads_strings_to_a_minimum_length,
     not_padded_when_min_len_is_less_than_input_string_len) {

    ASSERT_EQ(left_pad("", 0), "");
    ASSERT_EQ(left_pad("abc", 0), "abc");
}
```

```
std::string left_pad(std::string const& str, size_t min_len, char pad_char=' ');

#include "gtest/gtest.h"

TEST(left_pad_pads_strings_to_a_minimum_length,
     not_padded_when_min_len_is_less_than_input_string_len) {
    ASSERT_EQ(left_pad("", 0), "");
    ASSERT_EQ(left_pad("abc", 0), "abc");
}

TEST(left_pad_pads_strings_to_a_minimum_length,
     padded_when_min_len_equal_than_input_string_len) {
    ASSERT_EQ(left_pad("abc", 3), "abc");
}

TEST(left_pad_pads_strings_to_a_minimum_length,
     padded_when_min_len_greater_than_input_string_len) {
    ASSERT_EQ(left_pad("abc", 4), " abc");
}

TEST(left_pad_pads_strings_to_a_minimum_length,
     custom_pad_char_can_be_used) {
    ASSERT_EQ(left_pad("abc", 5, '*'), "**abc");
}
```

```
std::string left_pad(std::string const& str, size_t min_len, char pad_char=' ');

#include "catch.hpp"

TEST_CASE("left_pad pads strings to a minimum length") {
    SECTION("When padding is < string length, the existing string is returned") {
        REQUIRE(left_pad("", 0) == "");
        REQUIRE(left_pad("abc", 0) == "abc");
    }

    SECTION("When padding is == string length, the existing string is returned") {
        REQUIRE(left_pad("abc", 3) == "abc");
    }

    SECTION("When padding is > string length, the returned string is padded") {
        REQUIRE(left_pad("abc", 4) == " abc");
    }

    SECTION("String can be padded with a custom char") {
        REQUIRE(left_pad("abc", 5, '*') == "***abc");
    }
}
```

# Testing Best Practices

# Testing

## Best Practices

### Unit Tests should:

- ▶ Run fast
- ▶ Be deterministic & reproducible
- ▶ Be self-contained/ isolated
- ▶ Be consistent & reliable
- ▶ Be obvious & self-documenting
- ▶ Be focused on one thing ("single assert")
- ▶ Use public interfaces

# Testing

## Best Practices

### All tests should:

- ▶ Be well named
- ▶ Provide clear and useful feedback
- ▶ Have a clear reason
- ▶ Have low cyclomatic complexity

```
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( "An MRU list acts like a stack, "
           "but duplicate entries replace existing ones" ) {

    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( "An MRU list acts like a stack,  
          "but duplicate entries replace existing ones" ) {  
  
    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( "An MRU list acts like a stack,  
          "but duplicate entries replace existing ones" ) {  
  
    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( "An MRU list acts like a stack, "
           "but duplicate entries replace existing ones" ) {

    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( "An MRU list acts like a stack, "
           "but duplicate entries replace existing ones" ) {

    MRUList<std::string> list;                                "ARRANGE"

    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( "An MRU list acts like a stack, "
           "but duplicate entries replace existing ones" ) {

    MRUList<std::string> list;                                "ARRANGE"

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

## UNIT TESTS SHOULD BE REPRODUCIBLE AND DETERMINISTIC

```
TEST_CASE( "An MRU list acts like a stack, "
           "but duplicate entries replace existing ones" ) {

    MRUList<std::string> list;                                "ARRANGE"

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

TESTS SHOULD:

HAVE A SINGLE  
“LOGICAL” ASSERT

```
TEST_CASE( "An MRU list acts like a stack,  
"but duplicate entries replace existing ones" ) {  
  
    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 );  
    }  
}
```

TESTS SHOULD:

HAVE A SINGLE  
“LOGICAL” ASSERT

```
TEST_CASE( "An MRU list acts like a stack,  
"but duplicate entries replace existing ones" ) {
```

```
    MRUList<std::string> list;
```

```
    SECTION( "An empty list has no elements" ) {
```

```
        REQUIRE( list.empty() );
```

```
        REQUIRE( list.size() == 0 );
```

SINGLE ASSERT

```
}
```

```
    SECTION( "Adding to an empty list increases the size to 1" ) {
```

```
        list.add("item1");
```

```
        REQUIRE( list.empty() == false );
```

```
        REQUIRE( list.size() == 1 );
```

```
}
```

```
}
```

TESTS SHOULD:

HAVE A SINGLE  
“LOGICAL” ASSERT

```
TEST_CASE( "An MRU list acts like a stack,  
"but duplicate entries replace existing ones" ) {
```

```
    MRUList<std::string> list;
```

```
    REQUIRE( list.empty() );
    REQUIRE( list.size() == 0 );
```

```
    list.add("item1");
```

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

```
}
```

DEPENDENCY

MULTIPLE ASSERTS

# The Challenges & Pitfalls of Testing

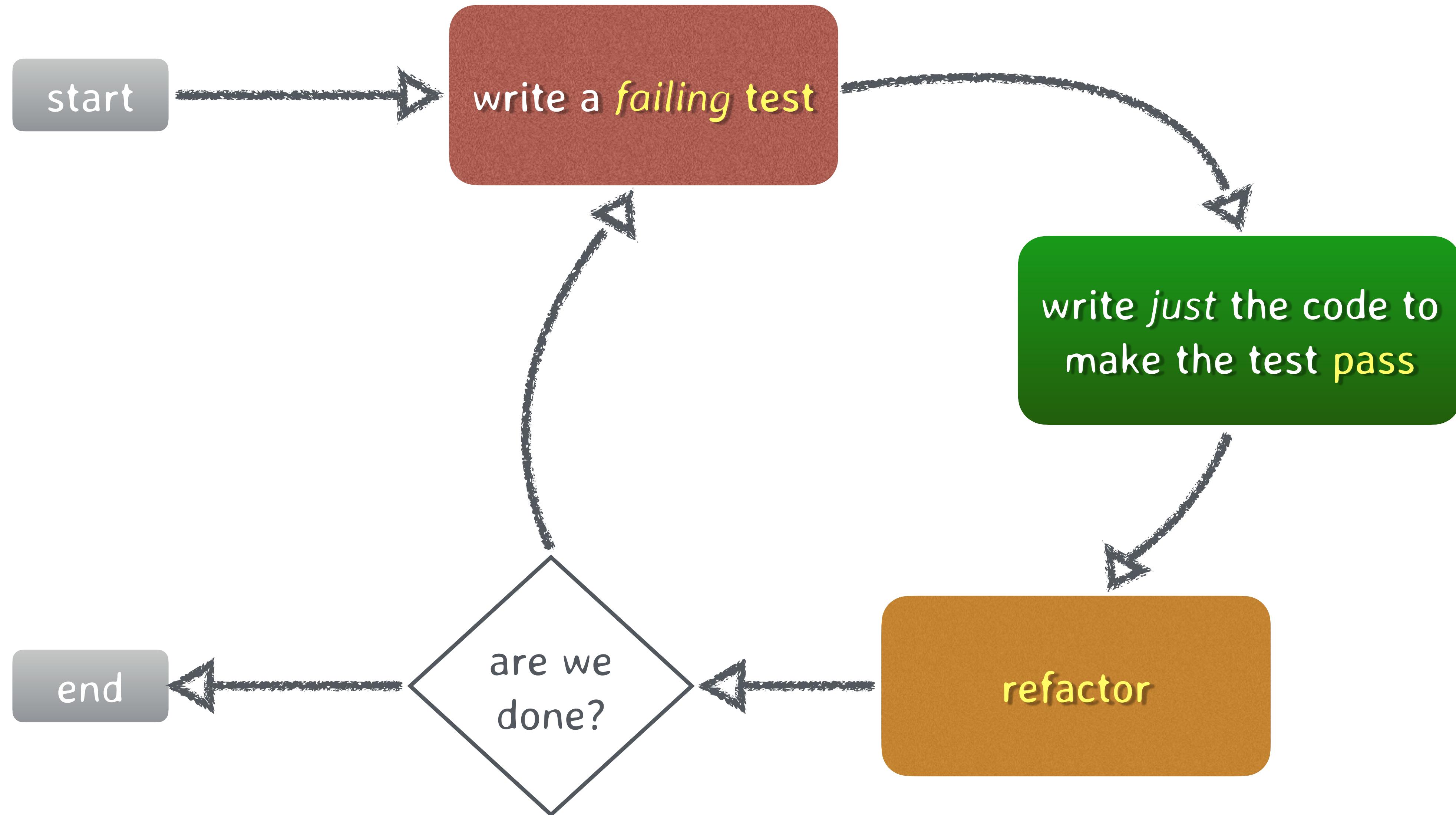
# The Challenges & Pitfalls of Testing

- ▶ Flaky tests
- ▶ Slow tests
- ▶ Brittle tests
- ▶ Hard to write

# what next?

# What next?

- ▶ Get comfortable with the best practices
- ▶ Become familiar with the design principles
- ▶ Pay attention to when testing is harder
  - ▶ **Then** consider trying TDD



The background of the image is a vibrant, abstract pattern of swirling ink. It features a dense concentration of fine, translucent lines in shades of green, teal, and blue on the left side, transitioning into darker purple and magenta on the right. Interspersed among these cooler tones are several larger, more opaque strokes in warm colors like yellow, orange, and red, creating a sense of depth and movement.

Design

# The SOLID principles

**SRP** Single Responsibility Principle

**OCP** Open-Closed Principle

**LSP** Liskov Substitution Principle

**ISP** Interface Segregation Principle

**DIP** Dependency Inversion Principle



**Cppcon**  
The C++ Conference



# Breaking Dependencies: The SOLID Principles

Klaus Iglberger

20  
20 |   
September 13-18

seeking

# Simplicity

NDC { London }

11-15 January 2016

Inspiring Developers  
since 2008



# *Back To Basics*

## Testing

[levelofindirection.com/refs/testing.html](http://levelofindirection.com/refs/testing.html)

PHIL NASH



@phil\_nash

@[mastodon@philnash.me](https://mastodon.social/@philnash)