C++ UT Framework Comparison

Sergio Arroutbi
Israel Pavón

EACS. Pruebas del Software

Google Test + Google Mock

- License: BSD
- xUnit
- Test Discovery
- Many Assertions and Matchers(Goggle Mock) + User defined
- Fatal & non Fatal assertions
- Value-parametrized test
- Type-parametrized test
- Multiple test running options
- XML reporting
- Mocking with Google Mock

Google Test + Google Mock

- PROS:
  - Mocks & Matchers
  - Custom Matchers
- CONS:
  - Higher Learning curve (lots of macros)
  - Installation & lib integration needed

Boost.Test + Boost.Turtle

- License: Boost (Permissive in the style of BSD, MIIT)
- Header only (optional)
- xUnit
- Fixtures & Group Fixtures
- Generators
- Mocks in .Turtle
- Exceptions assertions
- Macros if user requires it
- Template class testing
- Suites Grouping
- Data driven: random data capabilities.

Boost.Test + Boost.Turtle

- PROS:
  - Data Driven (Random data)
    - Some may think this makes the tests less understandable.
  - Header only
- CONS:
  - Mock library not out of the box

CATCH2

- License: Boost (Permissive in the style of BSD, MIIT)
- Ease of use: One include “catch.hpp”
- Section Based structure (no fixture oriented)
- Given / When / Then or traditional TCs
- One Comparison (easier to decompose fails)
- Free form strings for test naming
- XML and custom reports
- Junit xml output

CATCH2

- “main” override and CLI parser
- Fatal & non Fatal assertions
- Floating point tolerance
- Friendly macros
- Matchers
- Auto Registration / Test Discovery

CATCH2

- PROS:
  - Ease of Use / Ease to start
  - Header only
  - Given / When / Then built in
- CONS:
  - No Mocks
  - No customizable Matchers
  - No xUnit
  - Make compilation slower

MINUT (used in EPG for User Plane code)

“There exists many unit testing framework for many different programming languages, including C/C++.

Many/most of them are rather complex and with a rich set of functionality. This rich set of functionality can be intimidating to someone who wants to do unit testing in a more constrained environment, such as an embedded system written in C.

But the important thing about unit testing is the testing, not the framework. Therefore, it is better to have (and use) a minimal unit testing framework, than not doing any testing at all. That is when MINUT can be used.”

MINUT

- Ease of use: Just one .h file
- Very basic macros only for asserts:
  - MINUT_TEST_EQ
  - MINUT_TEST_STREQ
  - MINUT_ASSERT_PTREQ
  - MINUT_ASSERT_FLOATEQ
  - MINUT_RETURN_SUCCESS
  - MINUT_RETURN_FAILURE
- No automatic Suite/TC support

NS UNIT (Used in DPI)

- so library.
- Very basic macros:
- Suite/TC macros:
  - TEST_SUITE_DECLARE
  - TEST_CASE_ADD
  - Etc
- Basic assertions macros:
  - CHECK_TRUE
  - CHECK_FALSE
  - CHECK_EQUAL
- XML reports

	CATCH 2	Gtest / Gmock	Boost
Fixtures	✓	✓	✓
Sections	✓	✗	✗
BDD Style	✓	✗	✗
Non Fatal Assertions	✓	✓	✓
Exception Assertions	✓	✓	✓
Advanced Assertions	✗	✓	✓
Death Tests	✗	✓	✗
Parameterized Tests	✗	✓	✓

	CATCH 2	Gtest / Gmock	Boost
xUnit	✗	✓	✓
Enhanced Failure Messages	✗	✓	✓
Thread Safe	?	✓	?
Builtin Matchers	✓	✓	✓
Custom Matchers	✗	✓	✗
Mocks	✗	✓	✓

CATCH2 Example

```
// Let Catch provide main():
#define CATCH_CONFIG_MAIN

#include "catch.hpp"
#include "../src/Calculator.hpp"

TEST_CASE( "Adding 0 + 0 returns 0" ) {
  Calculator calc;
  REQUIRE(calc.Add(0, 0) == 0);
}

TEST_CASE( "Adding 0 + 1 returns 0" ) {
  Calculator calc;
  REQUIRE(calc.Add(0, -1) == -1);
}

TEST_CASE( "Divide by 0 returns exception" ) {
  Calculator calc;
  REQUIRE_THROWS_WITH(calc.Divide(1, 0), Catch::Matchers::Contains("received 0 in divisor"));
}
```

Boost.Test Example

```
1   #include "../src/Calculator.hpp"
2
3   #define BOOST_TEST_MODULE const_string CalcTest
4   #define BOOST_TEST_SHOW_PROGRESS yes
5   #include "boost/test/included/unit_test.hpp"
6
7   BOOST_AUTO_TEST_CASE( GivenCalculator_WhenAdd0_0_ThenResultIs_0 )
8   {
9     Calculator calc;
10    BOOST_CHECK_EQUAL(0, calc.Add(0,0));
11  }
12
13  BOOST_AUTO_TEST_CASE( GivenCalculator_WhenDivide1_0_ThenExceptionIsThrown )
14  {
15    Calculator calc;
16    BOOST_CHECK_THROW(calc.Divide(1,0), std::invalid_argument);
17
18  }
```

REFS

- https://github.com/google/googletest
- https://github.com/catchorg/Catch2
- http://www.boost.org/doc/libs/1_58_0/libs/test/doc/html/index.html