https://cmake.org/cmake/help/v3.0/module/FindGTest.html

https://github.com/google/googletest/blob/master/googletest/docs/Documentation.md

https://cmake.org/cmake/help/v3.0/module/FindGTest.html

Why use the Google C++ Testing Framework?

```
--gtest_repeat=1000 --gtest_break_on_failure.
Assertion
--gtest_output="xml:<file name>"
```

Installation

```
#install gtest on ubuntu
     #this package only install source files
          sudo apt-get install libgtest-dev

     #to create the necessary library files
          cd /usr/src/gtest
          sudo cmake CMakeLists.txt
          sudo make

     # copy or symlink libgtest.a and libgtest_main.a to your
     /usr/lib folder
          sudo cp *.a /usr/lib
```

Compilation

```
#g++ ../test.cpp -o test -lgtest -lpthread
```

Creating a basic test

```
#include <iostream>
#include "auto.hpp"
#include <gtest/gtest.h>

using namespace std;

TEST(BMW, CheckConstructor)
```

```
{
     BMW b = BMW("black",100);
     ASSERT_EQ(100, b.getSpeed());
     ASSERT_EQ("black", b.getColour());
}
```

Running the first test

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

```
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from BMW
[ RUN      ] BMW.CheckConstructor
[       OK ] BMW.CheckConstructor (0 ms)
[----------] 1 test from BMW (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (0 ms total)
[  PASSED  ] 1 test.
```

Options for the Google C++ Testing Framework

InitGoogleTest function accepts the **arguments** to the test infrastructure.

```
--gtest_output="xml:report.xml"
--gtest_repeat=2 --gtest_break_on_failure
--gtest_filter=<test string>
     --gtest_filter=*GoBMW
     --gtest_filter=-BMW.CheckGo
```

Temporarily disabling tests

```
DISABLED_ prefix
     YOU HAVE 1 DISABLED TEST
     --gtest_also_run_disabled_tests
```

It's all about assertions

There are two kinds of assertions

```
     ASSERT_
     EXPECT_
```

Floating point comparisons

```
     ASSERT_FLOAT_EQ (expected, actual)
     ASSERT_DOUBLE_EQ (expected, actual)
     ASSERT_NEAR (expected, actual, absolute_range)

     EXPECT_FLOAT_EQ (expected, actual)
     EXPECT_DOUBLE_EQ (expected, actual)
     EXPECT_NEAR (expected, actual, absolute_range)
```

it's smarter to use the macros specifically meant for floating point comparisons.

Typically, different central processing units (CPUs) and operating environments store floating points differently and simple comparisons between expected and actual values don't work.

Death tests

```
ASSERT_DEATH(statement, expected_message)
ASSERT_EXIT(statement, predicate, expected_message)
Statement - ::testing::ExitedWithCode(exit_code)

          std::cerr << "Error: Negative Input\n";
          exit(-1);

          ASSERT_EQ (0.0, X (0.0));
          ASSERT_EXIT (PoleProstokat(-22.0),
     ::testing::ExitedWithCode(-1), "Error: Negative Input");
```

Understanding test fixtures

initialization work **before** executing a unit test.

```
class myTestFixture1: public testing::Test {
public:
   myTestFixture1( ) {
       // initialization code here
   }

   void SetUp( ) {
       // code here will execute just before the test ensues
   }

   void TearDown( ) {
       // code here will be called just after the test completes
       // ok to through exceptions from here if need be
   }

   ~myTestFixture1( )  {
       // cleanup any pending stuff, but no exceptions allowed
   }

   // put in any custom data members that you need
};
```

::testing::test - declared in gtest.h

```
TEST_F (X, X) {
}
```

- initialization or allocation - the SetUp method
- deallocation in TearDown - throwing an exception from the destructor results in undefined behavior