Universiteit Antwerpen
Faculteit Wetenschappen

Testing & design-by-contract

Waarom (unit) testen?

- **Tijd besparen** tijdens het **programmeren**
  - Bugs kunnen eenvoudig gevonden worden
  - Toont aan waar een bug gevonden is
- **Tijd besparen** tijdens het **testen**
  - Je programma testen kan met één druk op de knop

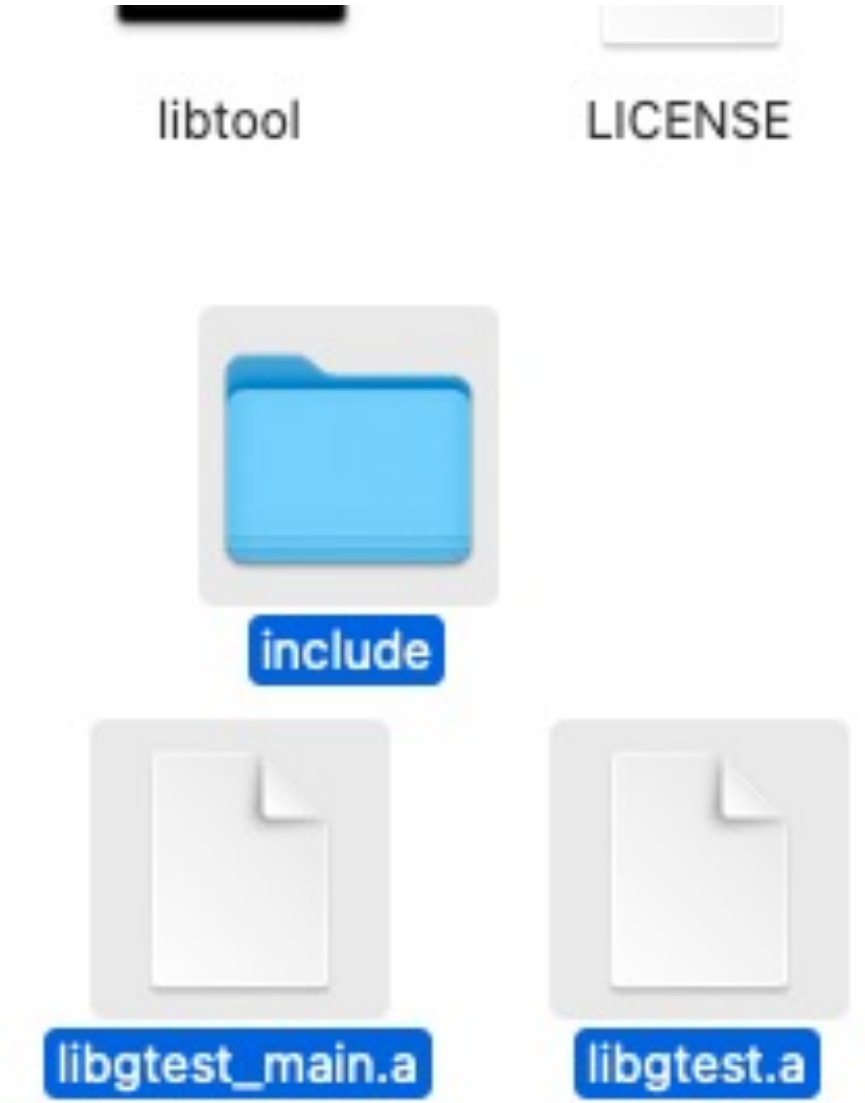
Waarom contracten?

- **Duidelijkheid**
  - Hoe roep ik deze functie op? (REQUIRE)
  - Onder welke omstandigheden moet ik garanderen dat mijn code werkt? (REQUIRE)
  - Welke functionaliteit/garanties voorziet mijn code? (ENSURE)
- **Errors expliciet maken**
  - Als je een programmeerfout maakt, zal er een error komen te staan
  - Maakt het eenvoudiger om bugs te vinden

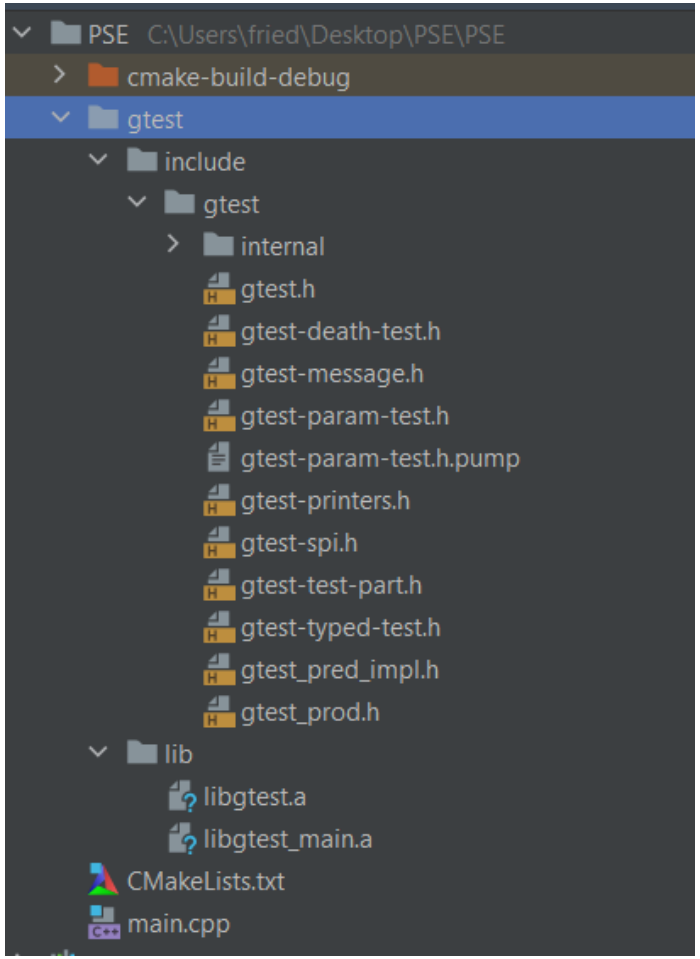
Gebruik van Google Test (gtest)

- Stappen:
  - Google Test compileren (verschillen tussen Windows en MacOS/Ubuntu!)
    - (Op Ansymore staan slides voor Windows, MacOS, en Ubuntu)
  - **“libgtest_main.a” en “libgtest.a” moeten vermeld staan in je .gitignore file!**
  - Google Test in het project duwen met CMakeLists.txt
  - Unittesten schrijven met Google Test
- “TicTacToe” voorbeeld: https://ansymore.uantwerpen.be/inleiding-software-engineering-1e-bac/studiemateriaal/se1bac-laad-tictactoe-clion

Belangrijke files



Gtest files toevoegen



Project structuur

- **Project_PSE**
  - gtest
    - include **(gekopieerd van gtest)**
    - lib
      - libgtest_main.a **(gekopieerd van gtest)**
      - libgtest.a **(gekopieerd van gtest)**
  - src
    - … src files …
  - CMakeLists.txt **(gekopieerd van Ansymore website)**
  - .gitignore **(gekopieerd van Ansymore website)**

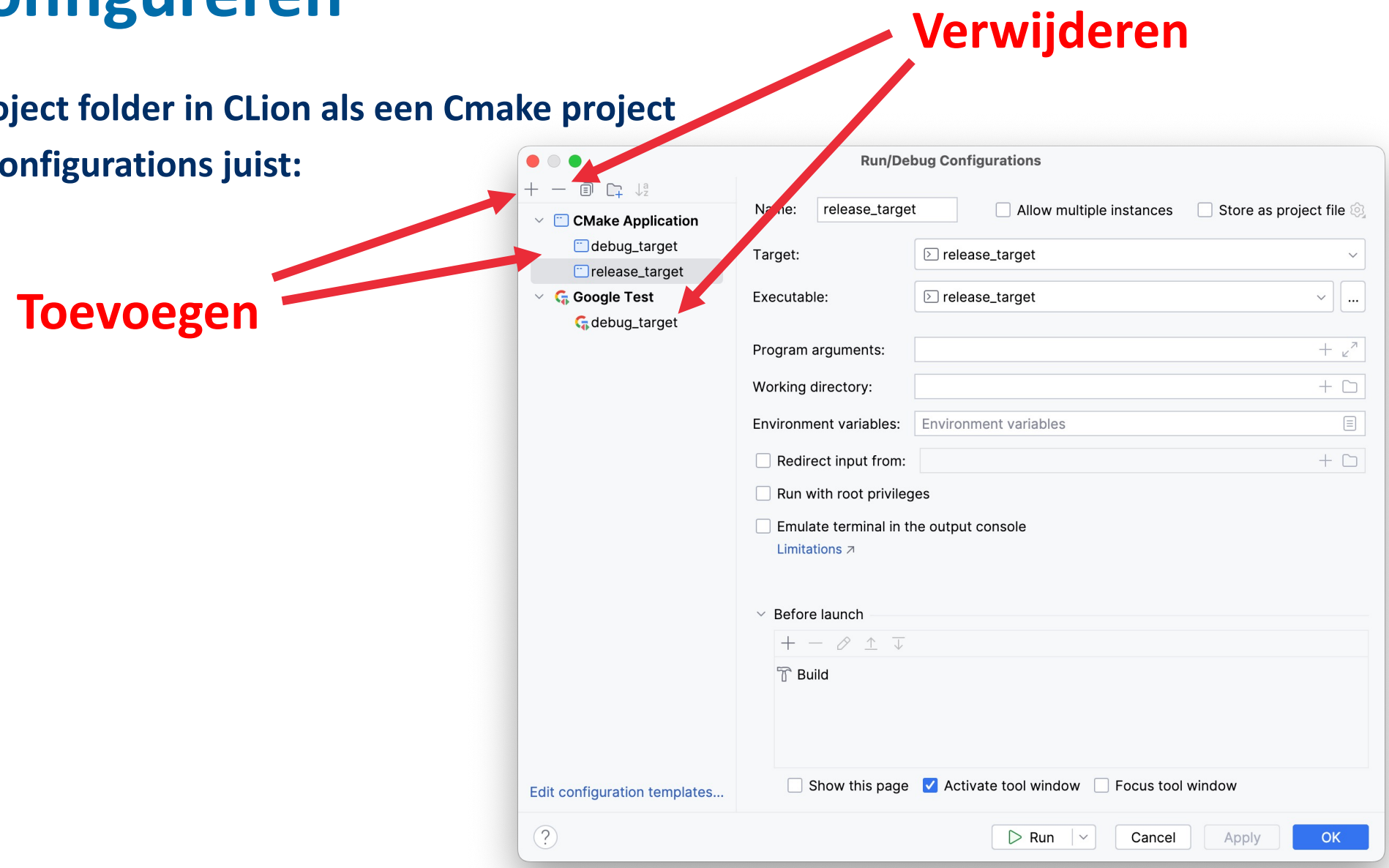
CMakeLists voorbeeld

- **Zelf aanpassen!**

```
cmake_minimum_required(VERSION 3.6)
project(ProjectTitle)

set(CMAKE_VERBOSE_MAKEFILE ON)

set(CMAKE_CXX_STANDARD 17)
# Remove the "-O2" flag if you have problems with the debugger.
set(CMAKE_CXX_FLAGS "-std=c++17 -Wall -Werror -O2")
set(CMAKE_EXE_LINKER_FLAGS -pthread)

# Set include dir
include_directories(./gtest/include)

# Set Library dir
link_directories(./gtest/lib)

# Set source files for RELEASE target
# Only add cpp files, no h files!
# Only add files with functionality here, no test files!
set(RELEASE_SOURCE_FILES EXAMPLE1.cpp EXAMPLE2.cpp main_release.cpp)

# Set source files for DEBUG target
# Only add cpp files, no h files!
set(DEBUG_SOURCE_FILES EXAMPLE1.cpp EXAMPLE2.cpp EXAMPLE_TESTS1.cpp EXAMPLE_TESTS1.cpp main_debug.cpp)

# Create RELEASE target
add_executable(release_target ${RELEASE_SOURCE_FILES})

# Create DEBUG target
add_executable(debug_target ${DEBUG_SOURCE_FILES})

# Link library
target_link_libraries(debug_target gtest)
```

CLion configureren

- Open de project folder in CLion als een Cmake project
- Zet de run configurations juist:



tests.cpp

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

Voorbeeld tests

```cpp
#include <gtest/gtest.h>

#include "Largest.h"
#include "Fighterplane.h"

class LargestTest: public ::testing::Test {
protected:
    // You should make the members protected s.t. they can be
    // accessed from sub-classes.

    // virtual void SetUp() will be called before each test is run.  You
    // should define it if you need to initialize the variables.
    // Otherwise, this can be skipped.
    virtual void SetUp() {
    }

    // virtual void TearDown() will be called after each test is run.
    // You should define it if there is cleanup work to do.  Otherwise,
    // you don't have to provide it.
    virtual void TearDown() {
    }
};
```

Voorbeeld tests

```
TEST_F(LargestTest, SimpleTest) {
    std::vector<int> some_numbers = std::vector<int>();
    some_numbers.push_back(3);
    some_numbers.push_back(5);
    some_numbers.push_back(-11);

    int max = largest2( list: &some_numbers);

    EXPECT_EQ(5, max);
}
```

Contracten

```
/**
 * Een voorbeeld van een functie met een contract.
 */
std::string simple_function_with_contract(int value_a, std::string value_b) {
    REQUIRE(value_a > 0, "Value a must be bigger than zero");
    REQUIRE(value_b.length() >= 4, "Value b must have more than 3 characters.");

    // works fine
    std::string return_value = value_b + " -> " + std::to_string( val: value_a);

    // if you do this, the 'ENSURE' will fail
//    std::string return_value = "";

    ENSURE(return_value != "", "Return value must never be the empty string.");

    return return_value;
}
```