Introduction to unit testing in C++ with the googletest framework

C++ Community

Julius Füg

01.07.2022

Table of contents

Unit testing: „manual“ vs. framework

- What is a unit test?
  - Testing of the smallest possible unit of the system under test that can be tested alone (ISTQB Glossary)
- Examples: components, units, modules, code, functions, data structures, classes, etc.
- Goal:
  - Reduce risks of false behaviour
  - Verify functional and non-functional requirements
  - Prevent component bugs to affect higher testing levels
  - Increase trust in the quality of the component

Unit testing: „manual“ vs. framework

- Test Suite: A set of test scripts or test procedures to be executed in a specific test run (ISTQB Glossary)
- Test Case: A set of preconditions, inputs, actions, expected results, post conditions, which are developed based on test conditions (ISTQB Glossary)

Googletest framework

- Testing framework developed by Googles Testing Technology team
- Goals:
  - Tests should be *independent* and *repeatable*
  - Tests should be well *organized* and reflect the structure of the tested code
  - Tests should be *portable* and *reusable*
  - Failed tests should provide as much *information* about the problem as possible

Googletest framework

- TEST macros:
  - Allow to define test suites and test cases
  - Offer different testing styles (plain, parameterized, fixture, typed, etc.)
- ASSERT_* and EXPECT_* macros:
  - Statements to check on conditions
  - Allow testing of a unit by making assertions about its behavior
  - ASSERT_* macros generate fatal failures and abort the current function
  - EXPECT_* macros generate nonfatal failures and allow the current function to continue running

Macro TEST

```
TEST(TestSuiteName, TestName) {
... statements ...
}
```

- If one assertion or expect in a TEST fails, then the whole TEST is marked as failed
- Test information is printed after test run
- Preconditions need to be defined for each TEST

Macro TEST_P

TEST_P(TestFixtureName, TestName) {
... *statements* ...
}

- Inherit from ::testing::TestWithParam<*dataType*>
- Instantiate with INSTANTIATE_TEST_CASE_P(InstantiationName, TestSuiteName, param_generator)
- Parameterized testing

Macro TEST_F (test fixture)

- Idea: define preconditions only once and use them across multiple test cases. Do not use global variables for this.

```
TEST_F(TestFixtureName, TestName) {
... statements ...
}
```

- Inherit class from ::testing::Test
- Write a constructor or SetUp() if necessary
- Write a destructor or TearDown() if necessary

Most important assertions:

- Explicit Success and Failure
- Generalized Assertion
- Boolean conditions
- Binary comparisons
- String comparison
- Floating point comparisons
- Exception Assertions
- Predicate Assertions
- Death Test

Summary and Outlook

- Unit testing „manual“ vs. googletest framework
- Macros: TEST, TEST_F, TEST_P
- Assertions, expectations, matchers
- Outlook: Gmock

Sources:

- Google test documentation: https://google.github.io/googletest/
- GoogleTest github repository: https://github.com/google/googletest (at the IIS we have to use a specific repository)
- Google test assertions: https://google.github.io/googletest/reference/assertions.html
- Google test matchers: https://google.github.io/googletest/reference/matchers.html
- Slides from „ISTQB Certified Tester“ seminar by Method Park
- ISTQB Glossary:
  - https://glossary.istqb.org/en/search/testing
  - https://glossary.istqb.org/en/search/test%20suite
  - https://glossary.istqb.org/en/search/test%20case

Thank you for your attention!

Additional Slides

Macro TYPED_TEST_SUITE

- Testing for data types

TYPED_TEST_SUITE(TestFixtureName, Types)

TYPED_TEST(TestFixtureName, TestName) {
... *statements* ...
}

- Inherit class from ::testing::Test

Run tests with CMake and ctest

- Configure: Cmake –S . –B build
- Build: cmake --build build
- Run ctest: cd build && ctest

Advanced options:

- See all options: --help
- Filter tests in file: --gtest_list_tests
- Run only specific tests: --gtest_filter=
- Stop execution after first failure: --gtest_fail_fast
- Disable test: use prefix DISABLED_ (--gtest_also_run_disabled_tests)
- Repeat selected tests: --gtest_repeat
- Run tests in random order: --gtest_shuffle
- Set seed: --gtest_random_seed=SEED
- Run tests on different machines: GTEST_TOTAL_SHARDS, GTEST_SHARD_INDEX
- Define colors: --gtest_color
- Suppress passing tests and elapsed time: --gtest_brief=1, --gtest_print_time=0
- Create JSON report:
- Detecting Test Premature Exit:
- Turning Assertion Failures into break points:
- Sanitizer Integration:

Unmentioned gtest topics:

- Teaching googletest How to Print Your Values
- Using Assertions in Sub-routines
  - Adding Traces to Assertions (SCOPED_TRACES) -> should I add it in?
  - Propagating Fatal Failures
- Logging Additional Information
- Sharing Resources Between Tests in the Same Test Suite
- Global Set-Up and Tear-Down
- Parameterized Tests
  - Creating Value-Parameterized Abstract Tests
  - Specifying Names for Value-Parameterized Test Parameters
- Testing Private Code (FRIEND_TEST)
- “Catching” Failures (EXPECT_FATAL_FAILURE, EXPECT_NONFATAL_FAILURE)
- Registering tests programmatically
- Getting the Current Test’s Name
- Extending googletest by Handling Test Events
- Classes and Types: TestSuite, TestInfo, TestParamInfo, UnitTest, TestEventListener, TestPartResult, TestProperty, TestResult, TimeInMillis, Types and many many more options
- Functions

Supported Plattforms

- Operating systems: Linux, macOS, Windows
- Compilers: gcc 5.0+, clang 5.0+, MSVC 2015+
- CMake: for compatible operating system and at least C++ 11 compiler
  - https://google.github.io/googletest/quickstart-cmake.html
  - for internal projects we have to use the svn repository
- Writing own main function is possible
- #include <gtest/gtest.h> to use googletest framework

Short introduction to software testing

- What is testing?

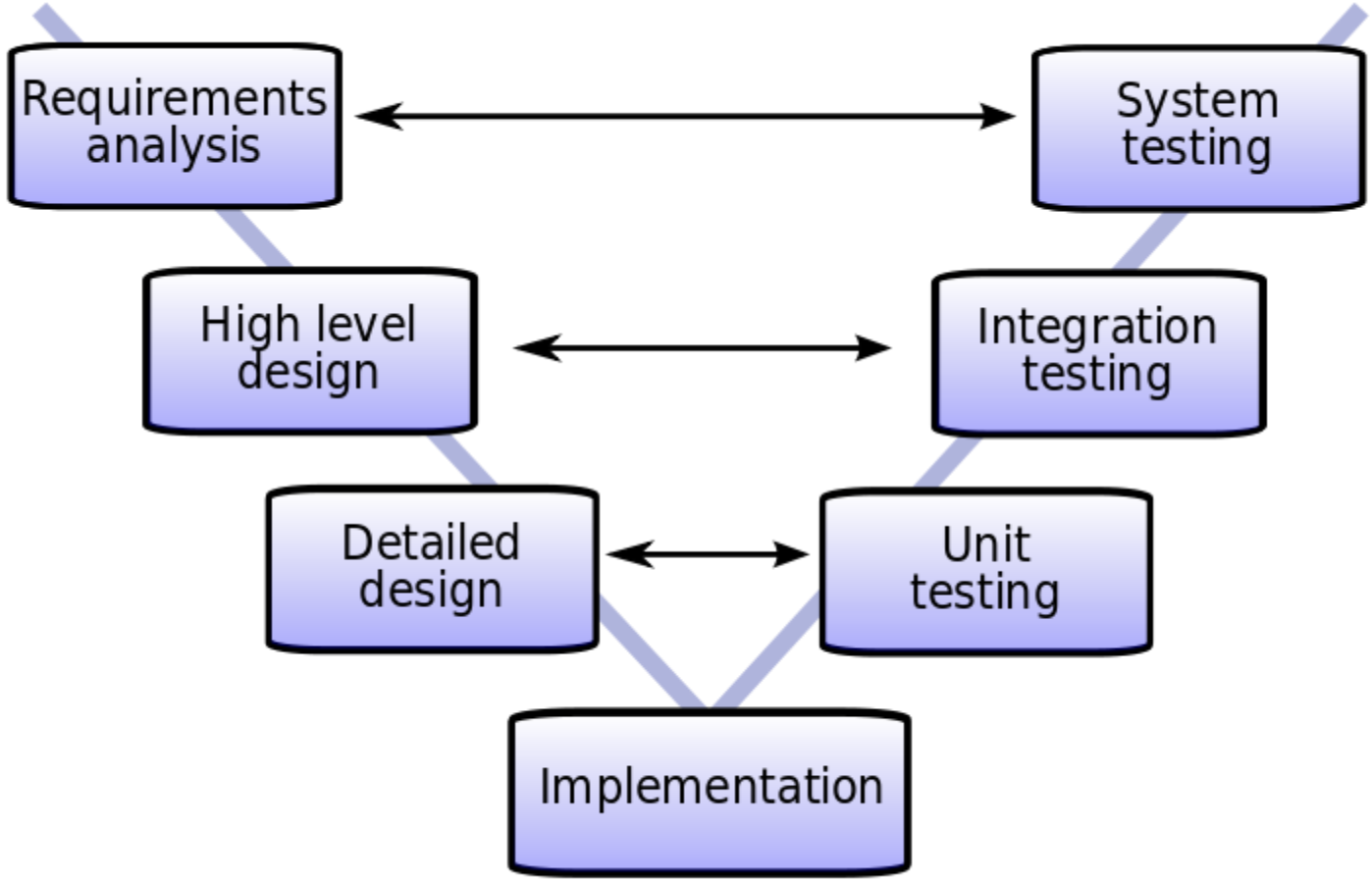
“The process consisting of all lifecycle activities, both static and dynamic, concerned with planning, preparation and evaluation of a component or system and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.“

- Definition from ISTQB (International Software Testing Qualification Board)

Short introduction to software testing

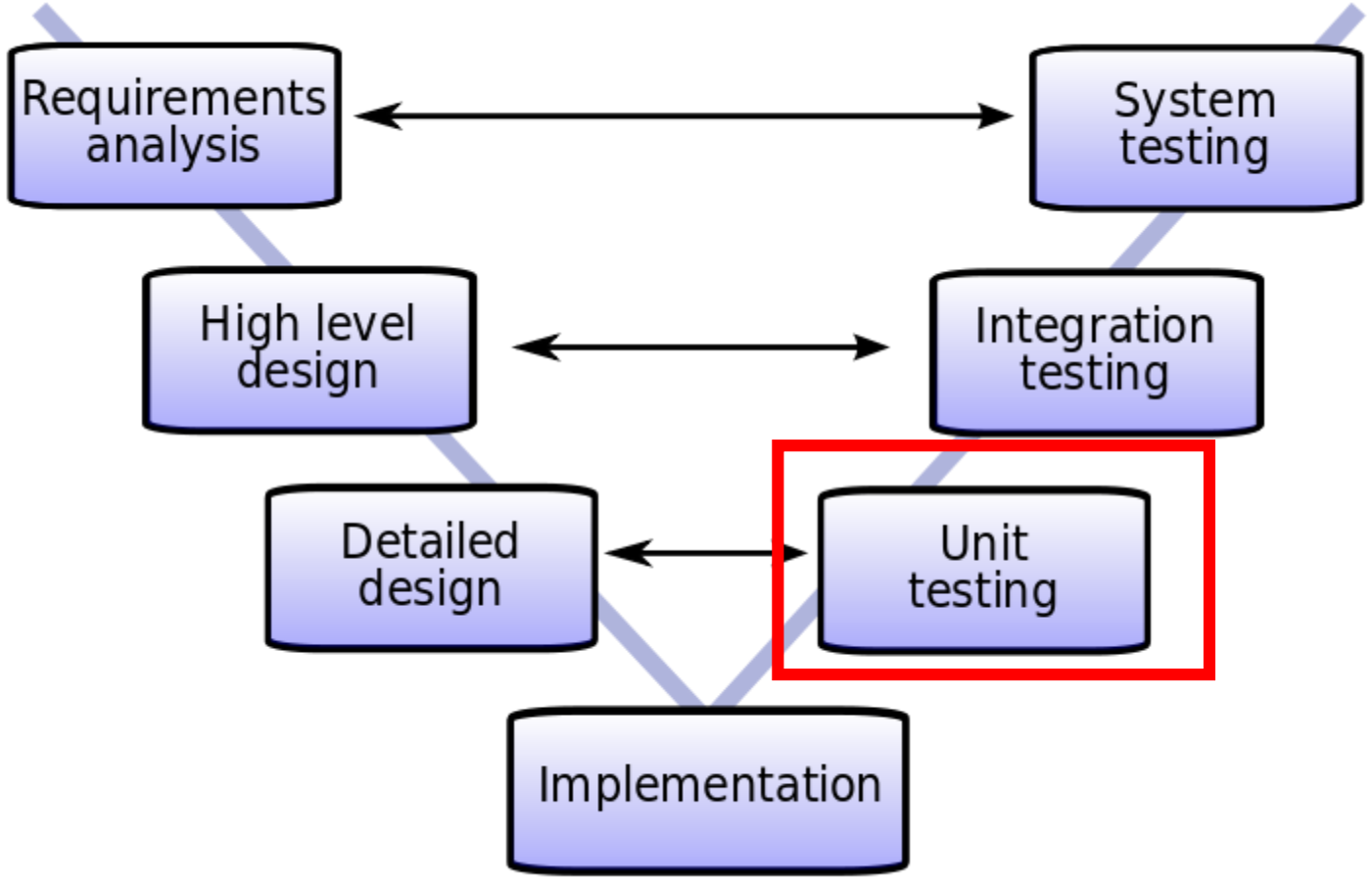
- Software testing is the activity of examining an object under test
- Objects under test: software systems, parts of software systems, products that contain software systems, documents for development, etc.
- Goal:
  - Check if the test object is complete
  - Validate functionality and identify false behaviour systematically
  - Reproduce false behaviour
  - Collect and document information about false behaviour
  - Rate test results in oder to identify bugs and errors
  - Gain trust in the quality of the test object

System development cycle: The V-Model



Source: https://www.pngegg.com/en/png-zkemm

System development cycle: The V-Model



Source: https://www.pngegg.com/en/png-zkemm