NOKIA

C++ Unit Tests

Piotr Nycz (Nokia)

09-03-2020

Agenda

- Introduction
- Test types
- Test Good Practices
- Cyclomatic Complexity
- TDD/ATDD
- Google-test
- Google-mock

Links

- google-test/google-mock: https://github.com/google/googletest
- TDD: http://agiledata.org/essays/tdd.html
- FIRST: http://agileinaflash.blogspot.fi/2009/02/first.html
- GivenWhenThen: https://martinfowler.com/bliki/GivenWhenThen.html
- AAA (ArangeActAssert): http://wiki.c2.com/?ArrangeActAssert

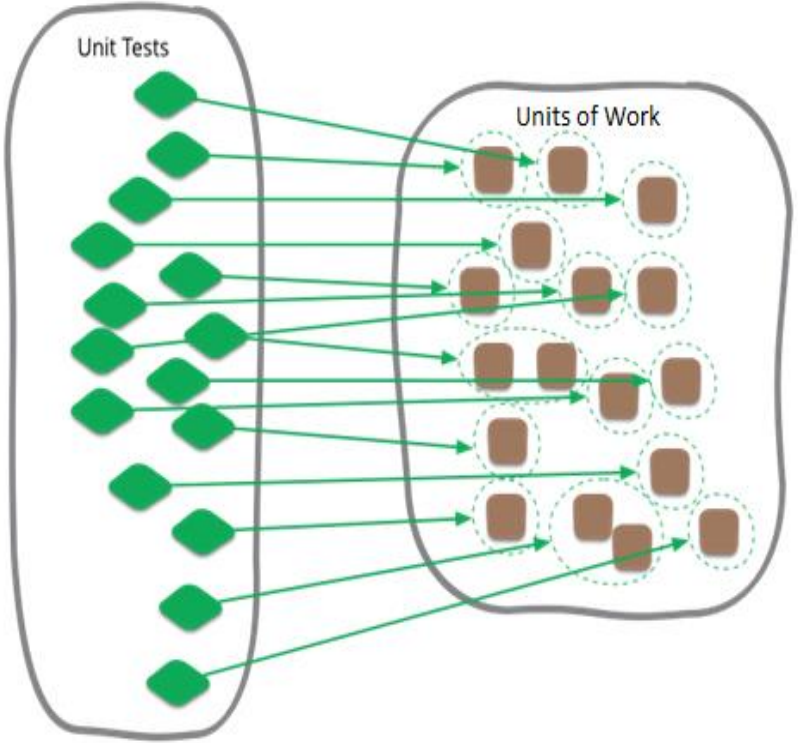
Unit Test

Unit Test (UT) is:

- **automated** piece of code
- that invokes a **unit of work** in the system
- checks the **single** assumption about the behavior of that unit of work
- **white box** testing

Unit of Work is:

- single logical functional use case in the system
- can be invoked by public interface
- single method, whole class or multiple classes working together to achieve **one single logical purpose** that can be verified.



NOKIA

Acceptance Tests

Acceptance test:

- verify that the **requirements (user stories)** are implemented
- **black box** testing – no assumption about implementation should be made.
- there could be a few levels of acceptance tests:
  - component tests (e.g. single level of application)
  - subsystem tests (e.g. group of collaborating applications)
  - system tests (e.g. full end-to-end scenario)

FIRST

fast – should be possible to run test as a part of build process

isolated – i.e. it should be only one reason to fail the test.
Without looking into test or report from its execution, it shall be obvious which behavior fails

repeatable – consecutive test execution results should not change.
When that can be broken? E.g. global variables, singletons, memorizing state from previous execution, might be the root cause

self-verifying – i.e. pass or fail
After test execution, it shall be no need to look into test execution logs, no need to debug, to know the test result with no doubts

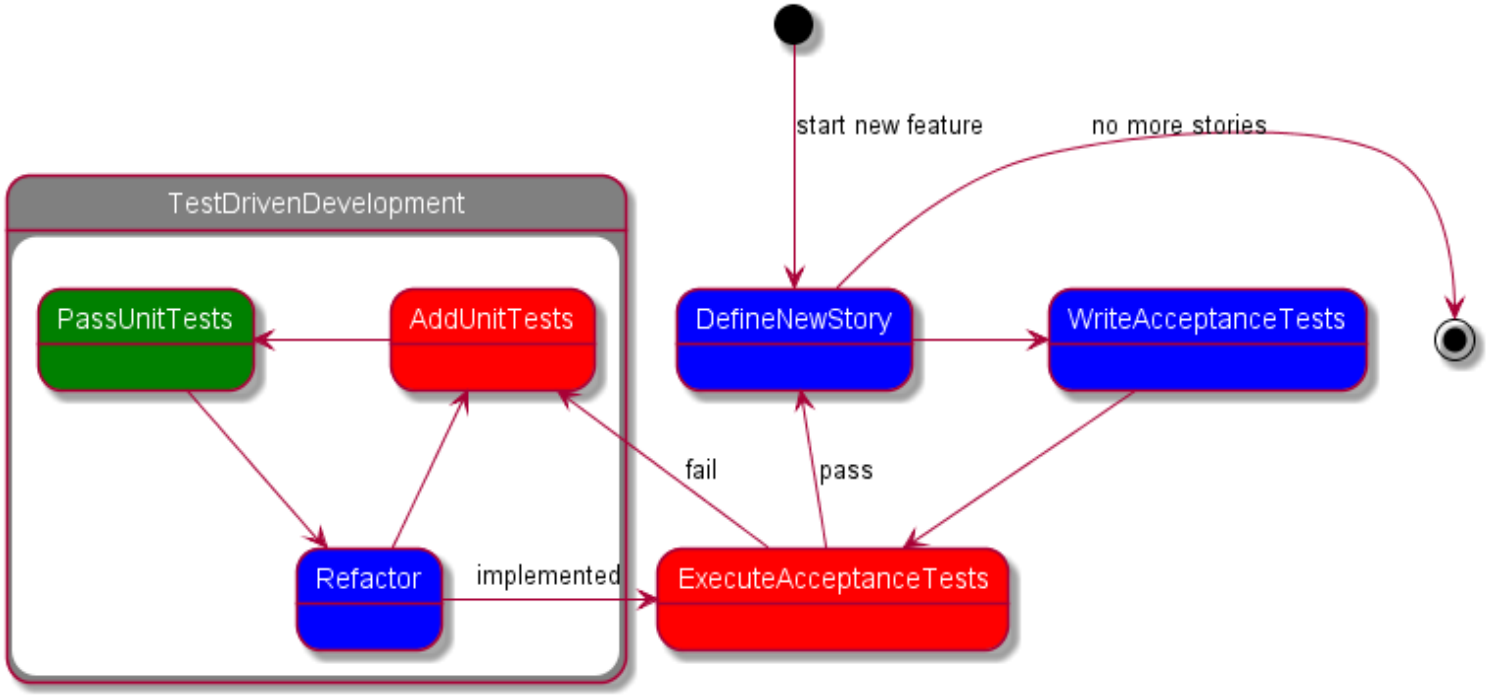
timely – test and code shall be defined as one task (see TDD)

Test-driven Development (TDD)

Principles of TDD:

- test-first programming concept
- short development cycles
- newly added test cases strictly cover new requirements
- the software shell be modified to **only pass the existing test cases** (new and legacy ones)

Test Driven Development (TDD) and Acceptance TDD



Unit Tests: Benefits vs. Costs

Benefits:

- inspires confidence
- easy to change and refactor code
- may improve the design of code
- form of documentation or requirements
- measure of completion
- many others

Unit Tests: Benefits vs. Costs

Drawbacks:

- takes more time for simple code
- does not show absence of errors
- hard to set up realistic, useful tests
- test code is likely to be at least as buggy as the code it is testing.
- value and accuracy of unit tests can be diminished if initial conditions are not set correctly.
- unit testing only helps with bugs you've anticipated or found

Cyclomatic Complexity

Cyclomatic Complexity (CC) is the number of **linearly independent paths** within source code

`auto a = b + c;` `std::cout << a;`	No control flow statements	CC==1

`auto a = b + c;`
`if (a > 2)`
`  std::cout << "Too big";`
	One IF-statement:
	path1: a>2
	path2: a<=2
		CC==2

`if (b > 2)`
`  if (c > 3)`
`  std::cout << "Too big";`
`auto a = b + c;`
	Two nested IF-s:
	path1: b>2,c>3
	path2: b<=2
	path3: b>2, c<=3
		CC==3

NOKIA

Cyclomatic Complexity vs. UT Branch Coverage

100% branch coverage in Unit Testing means:

each linearly independent path in the program will be tested **at least once**

CC is the **minimal number of test cases** to achieve 100% branch coverage

Google C++ Testing Framework

Google Test:

- widely used in industry
- cross-platform (Linux, Windows, Mac OS X, etc.)
- integrate **unit testing** and **mocking** functionality
- free
- supported by Google
- https://github.com/google/googletest

Basic test cases in google-test

- Macro **TEST()** defines a new test

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

using namespace ::testing;

TEST(FirstTestSuite, testThatEmptyTestcasePasses)
{
}
```

```
Running main() from gmock_main.cc
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from FirstTestSuite
[ RUN      ] FirstTestSuite.testThatEmptyTestcasePasses
[       OK ] FirstTestSuite.testThatEmptyTestcasePasses (0 ms)
[----------] 1 test from FirstTestSuite (4 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (12 ms total)
[  PASSED  ] 1 test.
```

Test Fixtures in google-test

```
using ::testing::Test;
class StdStackTestSuite : public Test
{
public:
    StdStackTestSuite() {}
    ~StdStackTestSuite() {}
    void SetUp() override {}
    void TearDown() override {}
protected:
    std::stack<int> objectUnderTest;
    const int ELEM1 = 1;
    const int ELEM2 = 2;
};
```

```
TEST_F(StdStackTestSuite, topShallReferToLastElementPushed)
{
    objectUnderTest.push(ELEM1);
    ASSERT_EQ(ELEM1, objectUnderTest.top());
    objectUnderTest.push(ELEM2);
    ASSERT_EQ(ELEM2, objectUnderTest.top());
}

TEST_F(StdStackTestSuite, topShallReferToLastButOneAfterPop)
{
    objectUnderTest.push(ELEM1);
    objectUnderTest.push(ELEM2);
    objectUnderTest.pop();
    ASSERT_EQ(ELEM1, objectUnderTest.top());
}
```

- **TEST_F()** defines **new** class **deriving** from TestSuite class
- New object of that class is created during each execution.
- Sequence:

  (1: Constructor) (2: SetUp) **(3: Test)** (4: TearDown) (5: Destructor)

NOKIA

Unit Testing – readability is important

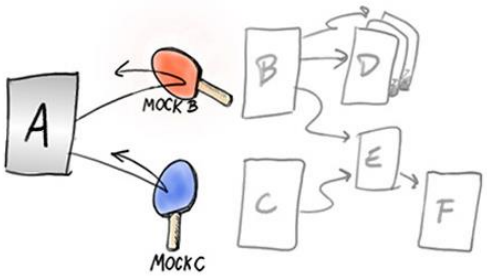
- Test name should answer these questions:
  - What is being tested?
  - In what condition, what is the context?
  - What behavior we expect?

TEST_F(CleanTestSuite, shallDoNothingWhenContainerEmpty)

Using mocked classes during UT

- In real software development most object depend on other objects
- Testing of **isolated** objects requires **substitution of dependent objects**

- Simulated objects:
  - **dummy** – no functionality, exist only to satisfy API or linker needs
  - **fake** – simplified implementation of real objects e.g. simple in-memory database instead OracleDB used in production environment
  - **stub** – predefined answers to function or method calls
  - **mock** – mimic the behavior of real object in controlled way
    - verify which functions are called
    - functions arguments, return value, side effects
    - order(sequence) of called functions



NOKIA

NOKIA

Copyright and confidentiality