EE5191 Software Testing & Security Checking

Project 1: White-box unit testing with CPP and coverage instrumentation

Deadline: Oct. 18, 2023 (23:59)

Requirement:

1. a) Software under test (SUT) is a GitHub open-source CPP module.
   b) Software under test (SUT) has been written by yourself.
   c) Software under test (SUT) has been modified from the example given by the TA.
2. Must write assertions to evaluate the pass or fail of the test scripts. For more details, you can refer CppUnit - The Unit Testing Library (sourceforge.net)
3. You should submit the following files 1) CPPUnit test modules, 2) Test reports, 3) SUT source code. In your test reports, you should explain the SUT, your test requirements, your test plan, and your test result with coverage analysis.

Item	Detail	Example
Explanation of the SUT	Briefly tell me what the software has done.	Student.cpp Record student's name and student id. Course.cpp Record course names and grades
Specification (mainly for 1b) & c) ONLY)	Restriction of your program	Grade should be an integer from 0 to 100.
Test Requirements	Description of what need to be tested in a software system.	The program should allow assigning grades to a student for specific courses.
Test Plan	Tell me the details of what you have done and tell me the test input. Related **TestCase** should be identified in the Test Plan.	Create a student object, assign grades for multiple courses, and verify that the assigned grades (A+/100/…) are correct.
Test Result and Bugs	Result (Pass/Fail) and Line Coverage should be provided. Tell me what bugs were detected.	**1. Screenshot (Pass/Fail, Coverage)** **2. Details of bugs detected:** Letter grade can be typed, it is not allowed…
Bonus for New Test Result and Fixed Bigs	After debug, the same test case will be pass/fail. Result and Line Coverage should be provided.	**1. Screenshot (Pass/Fail, Coverage)** **2. Old code vs new code & Explanation:** Warning message will be given / Letter grade cannot be typed…
Bonus for no. of stars in GitHub	N/A	**Screenshot**

4. Your score is: (line coverage) x $\log_2$(#lines) (maximum: 7.5) +

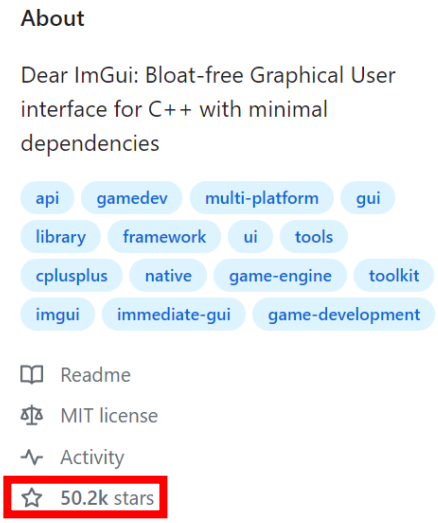
1.5 x (bugs detected) (maximum: 7.5) +

1.5 x (bugs fixed) + no. of stars in GitHub + completeness of your report (0/1) (maximum: 7.5)

Maximum score: 15

no. of stars in GitHub

>20k	>10k	>5k	>1k
3	2	1	0.5



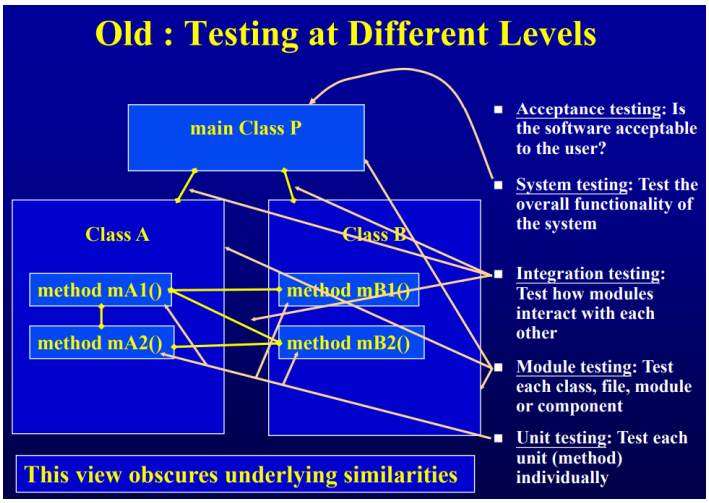
5. Late submission

	Oct. 19	Oct. 20	Oct. 21	Oct. 22	>= Oct. 23
Final Score	80%	60%	40%	20%	0

Before you start, you should know:

1. White-box testing: Deriving tests from the source code internals of the software, specifically including branches, individual conditions, and statements. (Chapter 1 P.56)

2. Unit testing: Test each unit (method) (Chapter 1 P.57)



3. CppUnit: C++ port of the JUnit testing framework. The main purpose of CppUnit is to support developers in doing their unit testing of C++ programs. For more details, you can refer [CppUnit - The Unit Testing Library (sourceforge.net)]

   In CppUnit, you should know the meaning of 1) TestCase, 2) TestFixture, 3) TestSuite, 4) TestRunner and 5) TestFactoryRegistry

   - TestCase represents a single test scenario and can be associated with a TestFixture for setup and teardown.
   - TestSuite groups multiple TestCase together, providing a way to organize and execute them collectively.
   - TestRunner orchestrates the execution of TestSuite, TestCase, and their associated TestFixture, collects test results, and generates reports.
   - TestFactoryRegistry is used to create and manage TestCase or TestSuite at runtime.

   Sometimes, TestCaller will be involved in the testing.

```h
// ComplexNumberTest.h

#ifndef COMPLEX_NUMBER_TEST_H
#define COMPLEX_NUMBER_TEST_H

#include <cppunit/extensions/HelperMacros.h>
#include "Complex.h" // Include the Complex class header.

class ComplexNumberTest : public CPPUNIT_NS::TestFixture {
    CPPUNIT_TEST_SUITE(ComplexNumberTest);
    CPPUNIT_TEST(testEquality);
    CPPUNIT_TEST(testAddition);
    CPPUNIT_TEST_SUITE_END();

public:
    void setUp();
    void tearDown();

    void testEquality();
    void testAddition();

private:
    Complex* m_10_1;
    Complex* m_1_1;
    Complex* m_11_2;
};

#endif // COMPLEX_NUMBER_TEST_H
```

TestFixture & TestSuite

TestCase

```cpp
// complexnumbertest.cpp

#include "ComplexNumberTest.h"
#include "Complex.h" // Include the Complex class header.

// Registers the fixture into the 'registry'
CPPUNIT_TEST_SUITE_REGISTRATION( ComplexNumberTest ); // M

void ComplexNumberTest::setUp() {
    m_10_1 = new Complex(10, 1);
    m_1_1 = new Complex(1, 1);
    m_11_2 = new Complex(11, 2);
}

void ComplexNumberTest::tearDown() {
    delete m_10_1;
    delete m_1_1;
    delete m_11_2;
}

void ComplexNumberTest::testEquality() {
    CPPUNIT_ASSERT(*m_10_1 == *m_10_1);
    CPPUNIT_ASSERT(!(*m_10_1 == *m_11_2));
}

void ComplexNumberTest::testAddition() {
    CPPUNIT_ASSERT(*m_10_1 + *m_1_1 == *m_11_2);
}
```

TestFactoryRegistry

TestFixture

TestCase

Please identify test case in test plan in here

e.g., Compare 10+1i = 10+1i

```cpp
// main.cpp

#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/ui/text/TestRunner.h>

int main(int argc, char* argv[])
{
    // Get the top level suite from the registry
    CPPUNIT_NS::Test *suite = CPPUNIT_NS::TestFactoryRegistry::getRegistry().makeTest();
    CppUnit::TextUi::TestRunner runner;

    runner.addTest(suite);
    bool wasSucessful = runner.run();

    getchar();

    return wasSucessful ? 0 : 1;
}
```

TestSuite & TestFactoryRegistry

TestRunner

4. Gcovr: A utility for managing the use of the GNU gcov utility and generating summarized code coverage results. This command is inspired by the Python coverage.py package, which provides a similar utility for Python. For more details, you can refer [gcovr — gcovr 6.0 documentation]

5. Another example: [Crash Course in using CppUnit]

6. Environment:
   Ubuntu 18.04/20.04/22.04

   For CppUnit:
   $ sudo apt-get update
   $ sudo apt-get install libcppunit-dev

   For Gcovr:
   $ sudo apt-get install gcovr

7. Compile & run the test:
   $ apt install g++
   $ g++ --coverage -g -O0 -o output_filename [cppfiles] -lcppunit
   - coverage file(.gcno) is generated after compiling.

   e.g., $ g++ --coverage -g -O0 -o test TestStudentMain.cpp Course.cpp Student.cpp TestStudent.cpp -lcppunit

   $ ./output_filename
   - run the executable file then the coverage data(.gcda) is generated.

   e.g., $ ./test

```
root@LAPTOP-OVHJ6FMG:/mnt/c/users/angus/downloads/studentunittest/c++# g++ --coverage -g -O0 -o test TestStudentMain.cpp
 Course.cpp Student.cpp TestStudent.cpp -lcppunit
root@LAPTOP-OVHJ6FMG:/mnt/c/users/angus/downloads/studentunittest/c++# ./test
..


OK (2 tests)
```

8. Check line coverage:
   $ gcovr
   - Analyze coverage data generated when compiling and running.

```
root@LAPTOP-OVHJ6FMG:/mnt/c/users/angus/downloads/studentunittest/c++# gcovr
------------------------------------------------------------------------------
                           GCC Code Coverage Report
Directory: .
------------------------------------------------------------------------------
File                                       Lines    Exec  Cover   Missing
------------------------------------------------------------------------------
Course.cpp                                     9       9   100%
Student.cpp                                   19      17    89%   21-22
TestStudent.cpp                               22      22   100%
TestStudent.h                                  4       4   100%
TestStudentMain.cpp                            8       8   100%
------------------------------------------------------------------------------
TOTAL                                         62      60    96%
------------------------------------------------------------------------------
```

9. If you are not familiar with C++, I also provide the sample code in Python version.

pip install coverage (Gcovr in Python)

For more details, you can refer https://coverage.readthedocs.io/en/7.3.0/

pip install pytest (another unit test library in Python)

For more details, you can refer

pytest: helps you write better programs — pytest documentation

unittest — Unit testing framework — Python 3.11.5 documentation

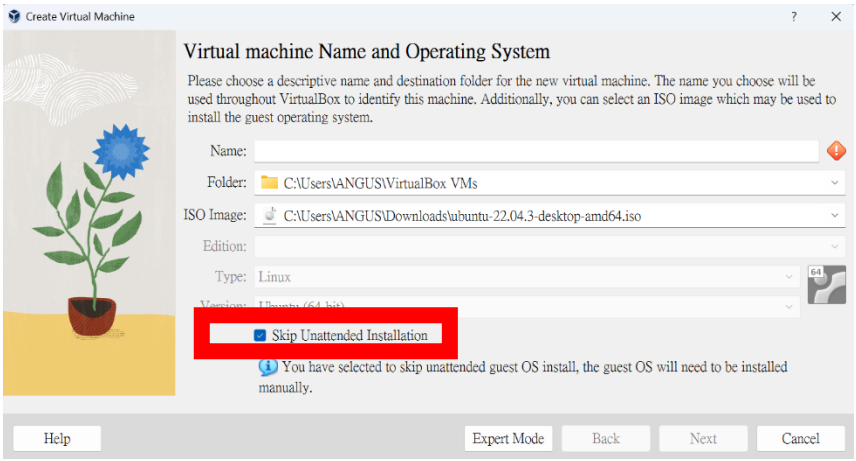
```
C:\Users\ANGUS\Downloads\StudentUnitTest\python1>coverage run -m unittest discover
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK

C:\Users\ANGUS\Downloads\StudentUnitTest\python1>coverage report -m
Name                Stmts   Miss  Cover   Missing
-------------------------------------------------
Course_Student.py      31      2    94%   30-31
Test_Student.py        21      0   100%
-------------------------------------------------
TOTAL                  52      2    96%
```

```
C:\Users\ANGUS\Downloads\StudentUnitTest\python2>coverage run -m pytest Test_Student.py
============================================= test session starts =============================================
platform win32 -- Python 3.7.9, pytest-7.4.1, pluggy-1.2.0
rootdir: C:\Users\ANGUS\Downloads\StudentUnitTest\python2
plugins: anyio-3.7.0
collected 2 items

Test_Student.py ..                                                                                    [100%]

============================================== 2 passed in 0.02s ==============================================

C:\Users\ANGUS\Downloads\StudentUnitTest\python2>coverage report -m
Name             Stmts   Miss  Cover   Missing
------------------------------------------------
Course.py            8      0   100%
Student.py          24      2    92%   23-24
Test_Student.py     21      0   100%
------------------------------------------------
TOTAL               53      2    96%
```

Remark:

1. If you install Ubuntu from Microsoft Store, you may need to install WSL first.
   安裝 WSL | Microsoft Learn

2. If you install Ubuntu in VirtualBox, you must select “Skip Unattended Installation”.



3. For file transfers between a Windows OS and Ubuntu in VirtiualBox, you may need to refer 如何與虛擬機內的 ubuntu 共享資料夾 - 大大通(繁體站) (wpgdadatong.com)