РАЗРАБОТКА МЕТОДИЧЕСКИХ УКАЗАНИЙ ПО СОЗДАНИЮ UNIT-ТЕСТОВ НА ЯЗЫКЕ C++

Усольцев Н. Д.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Меженная М. М. – к.т.н., доцент

Представлен процесс создания unit-тестов на языке C++. Назначение работы – методическое обеспечение процесса обучения студентов практическому навыку unit-тестирования.

Цель работы – методическое обеспечение процесса обучения студентов практическому навыку unit-тестирования.

В зависимости от степени изолированности тестируемых компонентов выделяют следующие направления в тестировании: Unit/component (модульное) – тестируются отдельные части (модули) системы; Integration (интеграционное) – тестируется взаимодействие между отдельными модулями; System (системное) – тестируется работоспособность системы в целом.

С целью проверки корректной работы отдельных частей (модулей) исходного кода программы разрабатываются и выполняются модульные, или unit-тесты. В рамках unit-тестирования тесты пишутся и выполняются разработчиками для каждой функции или метода. Такой подход позволяет обнаружить ошибки и устранить их еще на этапе разработки.

Для языков программирования высокого уровня существуют специализированные инструменты модульного тестирования. В работе рассматривается Google C++ Testing Framework – это фреймворк от Google для юнит-тестирования кода на C++. Для среды разработки Microsoft Visual Studio данный инструмент встроен по умолчанию [1].

Существует единый подход к написанию тестов, который призван сделать их более читабельными и понятными. Называется он AAA (arrange, act, assert). Он состоит в следующем: тест разбивается на три части. В первой (arrange) происходит объявление переменных тестирования и объектов проверяемого класса. Во второй (act) – вызов проверяемых функций, а в третьей (assert) – сравнение ожидаемого результата функций с фактическим [2].

Создание юнит-теста происходит следующий образом:

```
TEST(название_тест-кейса , название_теста) {
        …
}
```

Далее в теле следуют сами функции-тесты. Это функции, проверяющие работоспособность различных частей программы.

Пример теста, генерируемого по умоолчанию:

```
TEST(TestCaseName, TestName) {
  EXPECT_EQ(1, 1);
  EXPECT_TRUE(true);
}
```

В данном автоматически сгенерированном тесте показаны два примера: EXPECT_EQ и ECPECT_TRUE. Первый (EXPECT_EQ) принимает два аргумента: ожидаемое значение и значение, которое мы проверяем. В случае если ожидаемое значение равно проверяемому, тест выполняется успешно. Второй пример (ECPECT_TRUE) принимает один bool-агрумент и, если он возвращает true, тест выполняется.

Сборка и запуск модульных тесов на выполнение происходит с помощью встроенного в Microsoft Visual Studio 2017 обозревателя тестов. Об успешном выполнении всех тестов сигнализирует зеленая полоса вверху обозревателя и зеленые пункты в дереве тестов (рисунок 1).

Создадим несколько собственных unit-тестов для проверки простейшего приложения Калькулятор. В шаблоне TEST пропишем имя тест-кейса, как SumTest вместо TestCaseName. Также TestName изменим на sum2And7Eq9. Данное имя является осмысленным и создает самодокументирующийся код. Далее напишем тело теста следующим образом (следует обратить внимание на то, что вместо EXPECT используется ASSERT, что позволяет автоматически завершить работу функции, если хоть один из тестов не проходит, в то время, как EXPECT продолжает работу; в остальном этим макросы идентичны [3]):

```
TEST(SumTest, sum2And7Eq9) {
  // arrange
  double a = 2;                // первое слагаемое
  double b = 7;                // второе слагаемое
Calculate calculate(a, b);   // объект тестируемого класса
  // act
  double sum = calculate.sum(); // записываем сумму a и b в sum
  // assert
  ASSERT_EQ(9, sum);
}
```

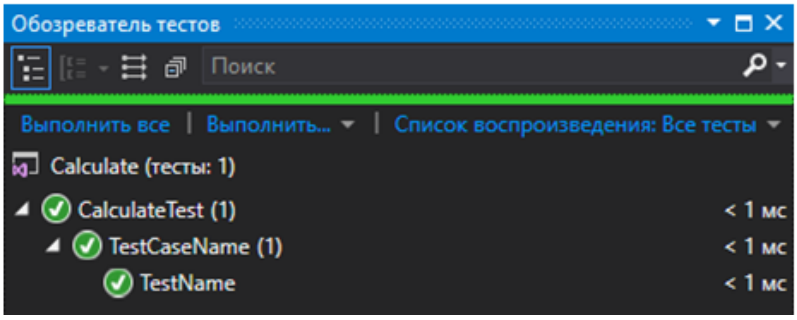

Рисунок 1 – Обозреватель тестов Microsoft Visual Studio 2017

Аналогично создадим еще одну функцию, которая будет проверять некоторые дробные значения. Теперь файл test.cpp будет выглядеть так:

```
TEST(SumTest, sum2And7Eq9) {
  // arrange
  double a = 2;                // первое слагаемое
  double b = 7;                // второе слагаемое
Calculate calculate(a, b);   // объект тестируемого класса
  // act
  double sum = calculate.sum(); // записываем сумму a и b в sum
  // assert
  ASSERT_EQ(9, sum);
}

TEST(SumTest, sum2p17And11p7Eq13p87) {
  // arrange
  double a = 2.17;                    // первое слагаемое
  double b = 11.7;                    // второе слагаемое
Calculate calculate(a, b);   // объект тестируемого класса
  // act
  double sum = calculate.sum(); // записываем сумму a и b в sum
  // assert
  ASSERT_EQ(13.87, sum);
}
```

Запустим все тесты на выполнение и убедимся, что все работает корректно. Стоит отметить, что идентичные по смысловой нагрузке тесты нужно объединять в тест-кейсы. Как видно из примера, два unit-теста были объединены в один тест-кейс SumTest. Для другого действия, например, произведения, следует задавать другой тест-кейс. Также в одном тесте можно вызывать несколько тест-функций.

Список использованных источников:

1. Википедия [Электронный ресурс] – Google C++ Testing Framework. – Режим доступа: https://ru.wikipedia.org/wiki/Google_C++_Testing_Framework, свободный.

2. Bitbucket [Электронный ресурс] – Введение в Google C++ Testing Framework. – Режим доступа: https://bitbucket.org/sonnayasomnambula/googletestprimerrussian/src, свободный.

3. GitHub [Электронный ресурс] – Документация по Google C++ Testing Framework. – Режим доступа: https://github.com/google/googletest/tree/master/googletest/docs, свободный.