Міністерство освіти і науки України
Національний університет “Запорізька політехніка”

кафедра програмних засобів

Реферат

з дисципліни «Якість програмного забезпечення та тестування»

на тему:

«Google Test»

Виконала:
ст. групи КНТ-116 Н. В. Мироненко
Прийняла:
проф. к.т.н. Г. В. Табунщик

2019

ЗМІСТ

1 ВСТУП

Метою даної роботи є розгляд і вивчення можливостей бібліотеки GoogleTest для модульного тестування програм на мові C++.

Тестування програмного забезпечення (Software Testing) – це перевірка відповідності між реальною та очікуваною поведінкою програми, яка здійснюється на кінцевому наборі тестів, обраному певним чином [1].

Тестування є дуже важливим етапом у розробці програмного забезпечення, адже код пишуть люди, а людям властиво помилятися. По статистиці: на кожну тисячу рядків коду припадає від 5 до 15 помилок, пошук і виправлення яких займає багато часу розробки та налагодження ПЗ. Окрім того наслідки «дефектного» програмного забезпечення можуть бути найрізноманітнішими – від незначних до катастрофічних, і навіть смертельних. Тому тестування є одним із найважливіших та найвідповідальніших етапів.

Одним з найпоширеніших видів тестування є модульне тестування.

Модульне тестування (Unit testing) — це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. У процедурному програмуванні модулем вважають окрему функцію або процедуру. В об'єктно-орієнтованому програмуванні — метод [2].

Модульні тести, або unit-тести, розробляють в процесі розробки програмісти та, іноді, тестувальники білої скриньки (white-box testers).

Зазвичай unit-тести застосовують для того, щоб упевнитися, що код відповідає вимогам архітектури та має очікувану поведінку.

Для проведення модульного тестування існує багато спеціалізованих бібліотек. Одна з них має назву Google Test. Саме вона буде розглянута в основній частині реферату.

2 ОСНОВНА ЧАСТИНА

2.1 Порівняння існуючих бібліотек

Для модульного тестування на мові C++ існує багато бібліотек. Наприклад, GoogleTest, Boost Test, CppUnit, VSCppUnit та ін. Порівняння даних бібліотек наведено у таблиці 1.

Таблиця 1 – Порівняльна характеристика бібліотек

Характеристика	**GoogleTest**	**Boost Test**	**CppUnit**	**VSCppUnit**
Професійна розробка	+	+	+	+
Підтримка розробниками	+	+	-	+
XML вивід	+	+	+	+
Тести «на смерть»	+	-	-	-
Тести фіксацій	+	+	+	+
Автоматична реєстрація тестів	+	+	+	+
Відключення тесту	+	-	-	+
Багатий набір вбудованих тверджень	+	-	-	-
Твердження користувача	+	+	+	+
Нефатальні твердження	+	+	-	-
Обробка збоїв	+	-	-	+

Провівши аналіз таблиці 1, можна зробити висновок, що одною з найкращих бібліотек модульного тестування на мові C++ є GoogleTest.

2.2 Короткий огляд бібліотеки Google Test

2.2.1 Визначення поняття Google Test

Google C ++ Testing Framework (Google Test) – бібліотека для модульного тестування на мові C ++. Вихідні тексти відкриті з середини 2008 року під ліцензією BSD. Документація частково переведена на російську мову [3].

Google Test побудована на методології тестування xUnit, тобто коли окремі частини програми (класи, функції, модулі) перевіряються окремо один від одного, в ізоляції. Бібліотека сама по собі розроблена з активним застосуванням тестування, коли при додаванні будь-яких частин в офіційну версію, крім коду самих змін необхідно написати набір тестів, що підтверджують їх коректність.

2.2.2 Основні особливості

Основні особливості використання бібліотеки Google Test:

– мінімальною одиницею тестування є одиночний тест. Тести не потрібно окремо реєструвати для запуску. Кожен оголошений в програмі тест автоматично буде запущений.

– тести об'єднуються в групи (набори). Повне ім'я тесту формується з імені групи і власного імені тесту.

– тести можуть використовувати тестові класи, що дозволяє створювати і повторно використовувати одну і ту ж конфігурацію об'єктів для кількох різних тестів.

– бібліотека є безпечною для багатопотокового використання.

– до складу бібліотеки входить спеціальний скрипт, який упаковує її вихідні тексти всього в два файли: gtest-all.cc і gtest.h. Ці файли можуть бути включені до складу проекту без будь-яких додаткових зусиль по попередній збірці бібліотеки.

2.2.3 Платформи для використання бібліотеки

Google Test може використовуватися на наступних платформах:

– Linux;

– Mac OS X;

– Windows;

– Cygwin;
– MinGW;
– Windows Mobile;
– Symbian;
– PlatformIO.

2.2.4 Вимоги для використання бібліотеки

Google Test розроблений з урахуванням мінімальних вимог до створення і використання з проектами, але є деякі:

– Bazel або CMake. Bazel - це система збирання, яку googletest використовує для своїх цілей і перевіряє.
– компілятор, сумісний зі стандартом C ++ 11

2.2.5 Інструменти пов'язані з Google Test

Google Test UI – це тестовий прогін, який запускає тестовий двійковий файл, дозволяє відстежувати його прогрес за допомогою індикатора виконання і відображає список невдалих тестів. Інтерфейс Google Test написаний на C #. Крім того, є повнофункціональним розширення Visual Studio з Google Test Adapter.

Популярні інструменти, використовувані в Chrome в поєднанні з Google Test:

– інструменти розробника дозволяє подивитися на елементи сторінки, ресурси і сценарії і включити відстеження ресурсів;
– консоль JavaScript дозволяє визначити, чи правильно працює JavaScript в консолі;
– перегляд вихідного коду дозволяє визначити, чи легко читати за допомогою колірного кодування, і чи легко потрапити у відповідний розділ;
– диспетчер завдань дозволяє визначити, чи правильно відображаються процеси і скільки ресурсів споживає веб-сторінка.

2.3 Основні поняття та концепції використання бібліотеки

2.3.1 Ключові поняття

Ключовим поняттям у тестовій системі Google є поняття ***твердження*** (assert). Твердження представляє собою вираз, результат виконання якого може бути успішним (success), некритичноє відмовою (nonfatal failure) та критичною відмовою (fatal failure). Критична відмова генерує завершення виконання тесту, в інших випадках тест продовжується.

Сам ***тест*** представляє собою набір тверджень. Крім того, тести можуть бути згруповані в ***набори*** (test case).

Якщо складна група об'єктів повинна бути використана в різних тестах, можна використовувати ***фіксації*** (fixture).

Об’єднані набори тестів називаються ***тестовою програмою*** (test program).

2.3.2 Твердження (assert)

Твердження googletest – це макроси, які нагадують виклики функцій. Тестувальник перевіряє клас або функцію, роблячи твердження про її поведінку. Якщо твердження не виконується, googletest роздруковує вихідний файл підтвердження і розташування номера рядка разом з повідомленням про помилку [4].

Твердження приходять парами, які перевіряють одне і те ж, але по-різному впливають на поточну функцію. ASSERT_ * версії генерують фатальні збої і скасовують поточну функцію. EXPECT_ * версії генерують нефатальні збої, які не скасовують поточну функцію. Зазвичай EXPECT_ * краще, бо вони дозволяють повідомляти про декілька невдач в тесті. ASSERT_ * потрібно використовувати, якщо немає сенсу продовжувати.

Основні твердження наведено у таблиці 2.

Таблиця 2 – Основні твердження

Група тверджень	**Фатальне твердження**	**Нефатальне твердження**	**Перевірка**
Логічні	ASSERT_TRUE(condition);	EXPECT_TRUE(condition);	condition правда
	ASSERT_FALSE(condition);	EXPECT_FALSE(condition);	condition помилка
Двійкове порівняння	ASSERT_EQ(val1, val2);	EXPECT_EQ(val1, val2);	val1 == val2
	ASSERT_NE(val1, val2);	EXPECT_NE(val1, val2);	val1 != val2
	ASSERT_LT(val1, val2);	EXPECT_LT(val1, val2);	val1 < val2
	ASSERT_LE(val1, val2);	EXPECT_LE(val1, val2);	val1 <= val2
	ASSERT_GT(val1, val2);	EXPECT_GT(val1, val2);	val1 > val2
	ASSERT_GE(val1, val2);	EXPECT_GE(val1, val2);	val1 >= val2
Порівняння рядків	ASSERT_STREQ(str1,str2);	EXPECT_STREQ(str1,str2);	два рядки мають однаковий зміст
	ASSERT_STRNE(str1,str2);	EXPECT_STRNE(str1,str2);	два рядки мають різний зміст
	ASSERT_STRCASEEQ(str1,str2);	EXPECT_STRCASEEQ(str1,str2);	два рядки мають однаковий зміст (без перевірки регістру)
	ASSERT_STRCASENE(str1,str2);	EXPECT_STRCASENE(str1,str2);	два рядки мають різний зміст (без перевірки регістру)

Логічні твердження виконують базове тестування на правильну / помилкову умову. Коли вони зазнають невдачі, ASSERT_ * призводить до фатального збою і повертає поточну функцію, в той час як EXPECT_ * до фатального збою, що дозволяє функції продовжувати роботу.

Двійкове порівняння описує твердження, які порівнюють два значення. Типи аргументів повинні буди такими, які можна порівняти між собою. Ці твердження

можуть працювати з користувацькими типами даних, але тільки якщо визначено відповідний оператор порівняння (наприклад, == або <).

Твердження у групі порівняння рядків порівнюють два С-рядки. Варто звернути увагу, що «CASE» в імені твердження означає, що регістр ігнорується. NULL покажчик і порожній рядок вважаються різними.

2.3.3 Тести (tests)

Тести використовують твердження для перевірки поведінки тестованого коду. Якщо тест має помилкове твердження, то він зазнає невдачі; в іншому випадку – успіху.

Для визначення тесту використовується макрос TEST. Він визначає void функцію, в якій можна використовувати твердження.

TEST приймає 2 параметри, які унікально ідентифікують тест, – назва тестового набору і назва тесту. В рамках одного і того ж тестового набору назви тестів не повинні збігатися. Якщо назва починається з DISABLED_, це означає, що тест (набір тестів) позначено як тимчасово не використовуваний.

Можна використовувати твердження не тільки в складі тесту, але і викликати їх з будь-якої функції. Приклад створення тесту:

```
TEST(test_case_name, test_name)
{
     ASSERT_EQ(1, 0) << "1 is not equal 0";
}
```

2.3.4 Фіксації (fixtures)

Трапляється, що об'єкти, які беруть участь в тестуванні, складно налаштовуються для кожного тесту. Можна задати процес налаштування один раз і

виконувати його для кожного тесту автоматично. У таких ситуаціях використовуються фіксації.

Фіксація – це клас, успадкований від :: testing :: Test, всередині якого оголошені всі необхідні для тестування об'єкти, при цьому в конструкторі або функції SetUp () виконуються їх налаштування, а в функції TearDown () звільнення ресурсів. Самі тести, в яких використовуються фіксації, повинні бути оголошені за допомогою макроса TEST_F, в якості першого параметра якого має бути зазначена не назва набору тестів, а назва фіксації [5].

Для кожного тесту буде створена нова фіксація, налаштована за допомогою SetUp (), запущений тест, звільнені ресурси за допомогою TearDown () і віддалений об'єкт фіксації. Таким чином кожен тест буде мати свою копію фіксації «не зіпсована» попереднім тестом.

2.4 Запуск тестів

TEST () і TEST_F () неявно реєструють свої тести в googletest. Таким чином, на відміну від багатьох інших C ++ каркасів тестування, тествальнику не потрібно повторно перераховувати всі тести, щоб запустити їх.

Визначивши тести, можна запустити їх за допомогою команди RUN_ALL_TESTS (), яка повертає результат 0, якщо всі тести пройдені успішно, або 1 у іншому випадку.

Коли викликано команду RUN_ALL_TESTS () макрос:

- зберігає стан всіх прапорів googletest;
- створює тестовий об'єкт для першого тесту;
- ініціалізується через SetUp ();
- запускає тест на об'єкті;
- очищає об'єкт через TearDown ();
- видаляє об'єкт;
- підновлює стан всіх прапорів googletest;

– повторює вищевказані кроки для наступного тесту, поки всі тести не будуть запущені. У разі фатального збою наступні кроки будуть пропущені.

2.5 Приклади тестування

2.5.1 Приклад тестування функції

Наприклад, є функція, яка повертає значення цілого типу

```
int Factorial(int n); // Повернути факторіал n
```

Тест для цієї функції може бути таким:

```
// Перевірити факторіал від 0.
 TEST(FactorialTest, HandlesZeroInput) {
   EXPECT_EQ(1, Factorial(0));
 }

 // Перевірити факторіал деяких додатних значень.
 TEST(FactorialTest, HandlesPositiveInput) {
   EXPECT_EQ(1, Factorial(1));
   EXPECT_EQ(2, Factorial(2));
   EXPECT_EQ(6, Factorial(3));
   EXPECT_EQ(40320, Factorial(8));
 }
```

2.5.2 Приклад тестування класу

Тест для черги типу FIFO з іменем Queue і з наступним інтерфейсом:

```
template <typename E> // E - тип елемента.
 class Queue {
  public:
   Queue();
   void Enqueue(const E& element);
   E* Dequeue(); // Повертає NULL, якщо черга порожня.
   size_t size() const;
   ...
 };
```

Спочатку потрібно визначити тестовий клас.

```
class QueueTest : public ::testing::Test {
  protected:
   virtual void SetUp() {
     q0_.Enqueue(1);
     q1_.Enqueue(2);
     q2_.Enqueue(3);
   }

   Queue<int> q0_;
   Queue<int> q1_;
   Queue<int> q2_;
 };
```

Тепер сам тест (макрос TEST_F() використовується замість TEST(), бо функція тестування повинна мати доступ до полів і методів класу):

```
// Перевірка ініціалізації черги.
 TEST_F(QueueTest, IsEmptyInitially) {
   EXPECT_EQ(0, q0_.size());
 }

 // Перевірка отримання елементів з черги.
 TEST_F(QueueTest, DequeueWorks) {
   int* n = q0_.Dequeue();
   EXPECT_EQ(NULL, n);

   n = q1_.Dequeue();
   ASSERT_TRUE(n != NULL);
   EXPECT_EQ(1, *n);
   EXPECT_EQ(0, q1_.size());
   delete n;

   n = q2_.Dequeue();
   ASSERT_TRUE(n != NULL);
   EXPECT_EQ(2, *n);
   EXPECT_EQ(1, q2_.size());
   delete n;
 }
```

3 ВИСНОВОК

У даній роботі було розглянуто можливості бібліотеки GoogleTest для модульного тестування програм на мові C++.

Тестування програмного забезпечення (Software Testing) – це перевірка відповідності між реальною та очікуваною поведінкою програми, яка здійснюється на кінцевому наборі тестів, обраному певним чином [1].

Модульне тестування (Unit testing) — це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми [2].

GoogleTest є однією з найкращих бібліотек для модульного тестування на мові програмування C++.

Google Test побудована на методології тестування xUnit, тобто коли окремі частини програми (класи, функції, модулі) перевіряються окремо один від одного, в ізоляції.

У порівнянні з існуючими аналогами, GoogleTest має значно багатший функціонал (таблиця 1), що робить її більш привабливою для користувача.

У розділі 2.2 розглянуто:

- короткий опис бібліотеки;
- основні особливості її використання;
- платформи, на яких вона може застосовуватися;
- вимоги до використання;
- інструменти, пов’язані з GoogleTest.

Для роботи з бібліотекою потрібно знати і вміти використовувати наступні поняття:

- твердження представляє собою вираз, результат виконання якого може бути успішним (success), некритичноє відмовою (nonfatal failure) та критичною відмовою (fatal failure);
- тест представляє собою набір тверджень;

– набір (test case) – група тестів;

– фіксація – це клас, успадкований від :: testing :: Test, всередині якого оголошені всі необхідні для тестування об'єкти, при цьому в конструкторі або функції SetUp () виконуються їх налаштування, а в функції TearDown () звільнення ресурсів;

– тестова програма (test program) – об’єднані набори тестів називаються.

Запуск тестів можна виконати за допомогою команди RUN_ALL_TESTS (), яка повертає результат 0, якщо всі тести пройдені успішно, або 1 у іншому випадку.

Приклад написання тестів для функції і класу наведено у підрозділі 2.5.

4 ЛІТЕРАТУРА

1. Лекційні матеріали навчального курсу "Software testing for Universities" – Київ: Провідна українська компанія з тестування програмного забезпечення QATestLab, 2019. – 353 с.

2. Модульне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Модульне_тестування

3. Google Test [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Google_Test.

4. Introduction: Why Google C++ Testing Framework? [Електронний ресурс] – Режим доступу до ресурсу: https://chromium.googlesource.com/external/github.com/google/googletest/+/refs/tags/release-1.8.0/googletest/docs/Primer.md.

5. Googletest Primer [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/google/googletest/blob/master/googletest/docs/primer.md.