Міністерство освіти і науки України

Національний університет «Запорізька політехніка»

Кафедра програмних засобів

РЕФЕРАТ

з дисципліни «Якість програмного забезпечення та тестування»

на тему: «Фреймворки для модульного тестування в Qt»

Виконав:

ст. групи КНТ-147сп Д.С. Куделя

Прийняв:

к.т.н., професор Г.В. Табунщик

2019

Зміст

[Вступ 3](#_Toc344364828)

[1 Огляд існуючих фреймворків модульного тестування для qt 4](#_Toc344364829)

[2 Детальний опис фреймворків 5](#_Toc344364829)

[2.1 Qt Test 5](#_Toc344364829)

[2.1.1 Характеристика Qt Test 5](#_Toc344364830)

[2.1.2 Підключення і використання Qt Test 6](#_Toc344364830)

[2.2 Google Test 9](#_Toc344364829)

[2.1.1 Характеристика Google Test 9](#_Toc344364830)

[2.1.2 Підключення і використання Google Test 10](#_Toc344364830)

[2.1 Boost.Test 12](#_Toc344364829)

[2.1.1 Характеристика Boost.Test 12](#_Toc344364830)

[2.1.2 Підключення і використання Boost.Test 13](#_Toc344364830)

[2.1 Catch & Trompeloil 14](#_Toc344364829)

[2.1.1 Характеристика Catch & Trompeloil 14](#_Toc344364830)

[2.1.2 Підключення і використання Catch & Trompeloil 15](#_Toc344364830)

[Висновки 17](#_Toc344364829)

[Список літератури 18](#_Toc344364829)

Вступ

Модульне тестування − це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми[1]. Модулем називають найменшу частину програми, яка може бути протестованою. У процедурному програмуванні модулем вважають окрему функцію або процедуру. В об'єктно-орієнтованому програмуванні − метод.

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.

Метою модульного тестування є ізоляція кожної частини програми та впевненість у тому, що кожна окрема частина є коректною. Модульний тест забезпечує жорсткий «контракт», за яким має працювати тестований код. Як результат, це надає деякі переваги. Модульне тестування допомагає знайти помилки раніше в циклі розробки ПЗ, що робить розробку дешевшою та швидшою.

Фреймворки для модульного тестування зазвичай не розповсюджуються в комплекті з компіляторами − це програмне забезпечення пишуть окремо. Вони розроблені для багатьох мов програмування й допомагають спростити процес тестування.

Qt − крос-платформовий інструментарій розробки програмного забезпечення (ПЗ) мовою програмування C++. Дозволяє запускати написане за його допомогою ПЗ на більшості сучасних операційних систем (ОС), просто компілюючи текст програми для кожної операційної системи без зміни вихідного коду[2].

1 Огляд існуючих фреймворків модульного тестування для Qt

Qt має власну бібліотеку компонентів, одним з яких є «Qt Test» − модуль для роботи з UNIT тестами.

Qt Test забезпечує класи для тестування одиниць програм Qt та бібліотек. Усі загальнодоступні методи знаходяться в просторі імен QTest. Крім того, клас QSignalSpy забезпечує просту інтроспекцію сигналів і слотів Qt, а QAbpositeItemModelTester дозволяє проводити тестування неруйнівних моделей елементів.

Зокрема існує також декілька сторонніх рішень, а саме:

* Google Test − це бібліотека модульного тестування для мови програмування C++, заснована на архітектурі xUnit;
* Boost.Test − бібліотека C++ для модульного тестування;
* Catch & Trompeloil, де Catch − це сучасна система модульного тестування на основі C++11 і вище, а Trompeloil − це макетний фреймворк, який використовує сучасні функції C ++ 14, і добре поєднується з Catch.

2 Детальний опис фреймворків

2.1 Qt Test

2.1.1 Характеристика Qt Test

Qt Test − це основа для тестування одиниць додатків і бібліотек на основі Qt. Qt Test забезпечує всю функціональність, зазвичай зустрічається в одиничних тестових рамках, а також розширення для тестування графічних інтерфейсів користувача. Побудований на ультра крос-платформі Qt Library. Дозволяє тестувати на Windows, Mac OS X, Linux, BSD, Symbian та будь-якій іншій платформі, на якій Qt може працювати.

Перелік характеристик:

* поширюється по ліцензії − GNU LGPL;
* підтримує xUnit;
* підтримує «Fixtures» (test fixture − це тестові класи, що використовуються для послідовного тестування певного предмета, пристрою чи частини програмного забезпечення);
* підтримує вийнятки;
* не підтримує макроси, макети, генератори і «Group fixtures».

Перелік особливостей:

* легкий − Qt Test складається з близько 6000 рядків коду та 60 експортованих символів;
* самодостатній − для автономного тестування Qt потрібно лише кілька символів з модуля Qt Core для тестування без gui;
* швидке тестування − Qt Test не потребує спеціальних запусників тестів, немає спеціальної реєстрації на тести;
* тестування на основі даних − тест може бути виконаний кілька разів з різними даними тесту;
* базове тестування графічного інтерфейсу − Qt Test пропонує функціональність для моделювання миші та клавіатури;
* бенчмаркінг − Qt Test підтримує бенчмаркінг тестування та забезпечує декілька результатів вимірювань;
* дружнє IDE − Qt Test виводить повідомлення, які можна інтерпретувати за допомогою Qt Creator, Visual Studio та KDevelop;
* потокобезпечний − повідомлення про помилки є безпечними для потоків та атомарними;
* типовобезпечний − широке використання шаблонів запобігає виникненню помилок, що вводяться під неявного приведення типів;
* легко розширюваний − користувацькі типи можуть бути легко додані до тестових даних та тестових результатів[3].

2.1.2 Підключення і використання Qt Test

Щоб створити тест Qt Test:

1. виберіть "Файл"> "Новий файл або проект"> "Інший проект"> "Автоматичний тестовий проект"> Виберіть створення проекту з кодовою табличкою для тесту Qt або швидкого тесту Qt;
2. у діалоговому вікні Інформація про проект та тест вкажіть параметри проекту та тесту:
3. у полі Test Framework виберіть Qt Test або Qt Quick Test;
4. для тесту Qt встановіть прапорець GUI Application, щоб створити додаток Qt;
5. у полі Ім'я тестового випадку введіть назву тестового випадку;
6. для тесту Qt встановіть прапорець Потрібно QApplication, щоб додати оператор include для QApplication у файл main.cpp проекту;
7. установіть прапорець Створити код ініціалізації та очищення, щоб додати функції до вашого тесту, які виконуються тестовою рамкою для ініціалізації та очищення тесту;
8. у полі Система побудови виберіть систему побудови, яка буде використана для побудови проекту: qmake, CMake або Qbs;

Щоб створити тест, створіть підклас від QObject і додайте до нього один або більше приватних слотів. Кожен приватний слот є тестовою функцією у вашому тесті. QTest::qExec() може використовуватися для виконання всіх тестових функцій у тестовому об'єкті.

Крім того, ви можете визначити такі приватні слоти, які не розглядаються як тестові функції. За наявності, вони будуть виконані тестовою рамкою і можуть бути використані для ініціалізації та очищення або всього тесту, або поточної функції тестування.

initTestCase() буде викликано перед виконанням першої функції тесту.

initTestCase\_data() буде викликано для створення глобальної таблиці тестових даних.

cleanupTestCase() буде викликано після виконання останньої функції тесту.

init() буде викликатися перед виконанням кожної тестової функції.

cleanup() буде викликано після кожної тестової функції.

Використовуйте initTestCase() для підготовки тесту. Кожен тест повинен залишити систему в придатному стані, щоб її можна було запустити повторно. Операції очищення слід обробляти в cleanupTestCase(), щоб вони запустилися, навіть якщо тест не вдався.

Використовуйте init() для підготовки тестової функції. Кожна тестова функція повинна залишати систему в придатному для використання режимі, тому її можна запускати повторно. Операції очищення слід обробляти під час cleanup(), щоб вони запускалися, навіть якщо тестова функція не працює та закінчується рано.

Крім того, ви можете використовувати RAII (resource acquisition is initialization), з операціями очищення, викликаними в деструкторах, щоб переконатися, що вони відбудуться, коли тестова функція повернеться і об'єкт вийде з поля його дії.

Якщо initTestCase() не вдається, жодна тестова функція не буде виконана. Якщо init() не вдається, наступна тестова функція не буде виконана, тест перейде до наступної тестової функції.

Приклад тесту зображено на рисунку 2.1.



Рисунок 2.1 – Приклад написання тесту з використанням Qt Test

2.2 Google Test

2.2.1 Характеристика Google Test

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

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[4].

Офіційно Google Test підтримує Linux, Windows і Mac. Для цих платформ бібліотека надає всі необхідні скрипти для збірки. Однак, Google Test також працює на AIX, HP-UX, Solaris, Tru64, zSeries і безлічі інших систем. Для платформ, які офіційно не піфдтримуються, розробник повинен самостійно скомпілювати Google Test.

Перелік особливостей:

* 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 (Test fixture), що дозволяє створювати і повторно використовувати одну і ту ж конфігурацію об'єктів для кількох різних тестів;
* розробники стверджують, що бібліотека є безпечною для багатопотокового використання, помічаючи при цьому, що для використання тверджень в різних потоках одночасно необхідно самостійно розробити примітиви синхронізації;
* 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.

2.2.2 Підключення і використання Google Test

Щоб створити тест Google:

1. виберіть "Файл"> "Новий файл або проект"> "Інший проект"> "Автоматичний тестовий проект"> Виберіть для створення проекту проект з кодовою табличкою для тесту Google;
2. у діалоговому вікні Інформація про проект та тест вкажіть параметри проекту та тесту:
3. у полі Test Framework виберіть Google Test;
4. у полі Ім'я тестового набору введіть ім'я для тестового набору;
5. у полі Ім'я тестового випадку введіть назву тестового випадку;
6. установіть прапорець Увімкнути C ++ 11, щоб підтримувати функції C ++ 11 у тесті;
7. у полі тестового сховища Google виберіть каталог, який містить клон репозиторію googletest;
8. у полі Система побудови виберіть систему побудови, яка буде використана для побудови проекту: qmake, CMake або Qbs.

Приклад тестування функції зображено на рисунку 2.2, приклад тестування класу зображено на рисунку 2.3.

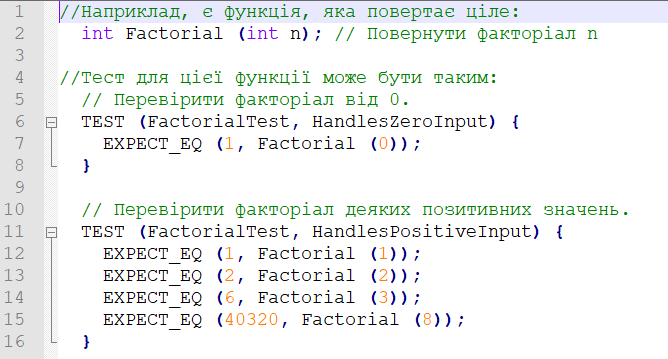


Рисунок 2.2 – Приклад тестування функції з використанням GoogleTest

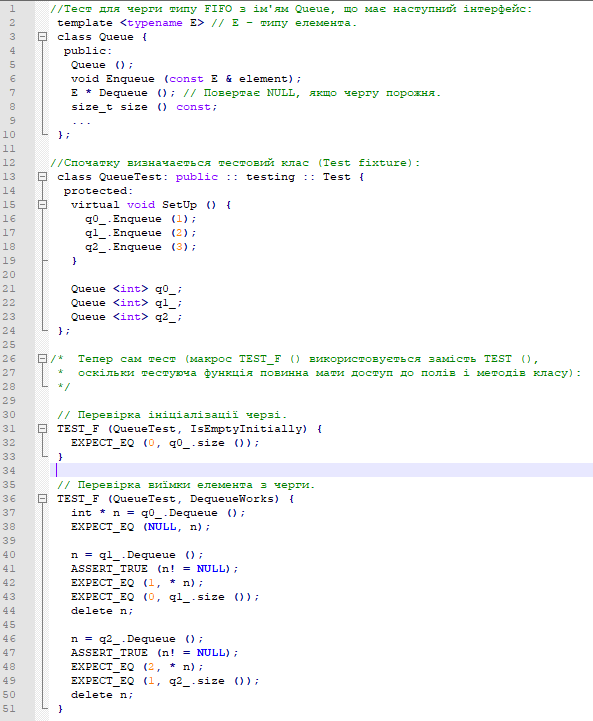


Рисунок 2.3 – Приклад тестування класу з використанням GoogleTest

2.3 Boost.Test

2.3.1 Характеристика Boost.Test

Boost.Test, частина бібліотеки Boost, потужна концепція набору даних для генерації тестових випадків. Різні рівні тестових класів (test fixtures) (глобальний, один на тестовий набір, один на кожен тестовий випадок в наборі). Потужне порівняння з плаваючою комою.

Бібліотека Boost.Test пропонує як простий у використанні, так і гнучкий набір інтерфейсів для написання тестових програм, організацію тестів у прості тестові кейси та тестові набори, а також контроль їх виконання. Деякі інтерфейси Boost.Test також корисні у виробничих (нетестових) середовищах[5].

Перелік характеристик:

* поширюється по ліцензії − Boost;
* підтримує xUnit;
* підтримує «Fixtures» (test fixture − це тестові класи, що використовуються для послідовного тестування певного предмета, пристрою чи частини програмного забезпечення);
* підтримує «Group fixtures»;
* підтримує генератори;
* підтримує макети тільки при використанні з бібліотекою «Turtle»;
* підтримує вийнятки;
* підтримує макроси;
* підтримує шаблони
* підтриує групування за тестовими наборами та мітками[6].

2.3.2 Підключення і використання Boost.Test

Щоб створити тест Boost:

1. виберіть "Файл"> "Новий файл або проект"> "Інший проект"> "Автоматичний тестовий проект"> Виберіть для створення проекту проект з кодовою табличкою для тесту Boost;
2. у діалоговому вікні Інформація про проект та тест вкажіть параметри проекту та тесту:
3. у полі Test Framework виберіть Boost Test;
4. у полі Ім'я тестового набору введіть ім'я для тестового набору;
5. у полі Ім'я тестового випадку введіть назву тестового випадку;
6. у поле Boost include dir (необов'язково) введіть шлях до каталогу, який містить файли, необхідні Boost.Test, такі як version.hpp та підпапка під назвою test, що містить файли тестових заголовків;
7. у полі Система побудови виберіть систему побудови, яка буде використана для побудови проекту: qmake, CMake або Qbs.

Приклад використання зображено на рисунку 2.4.

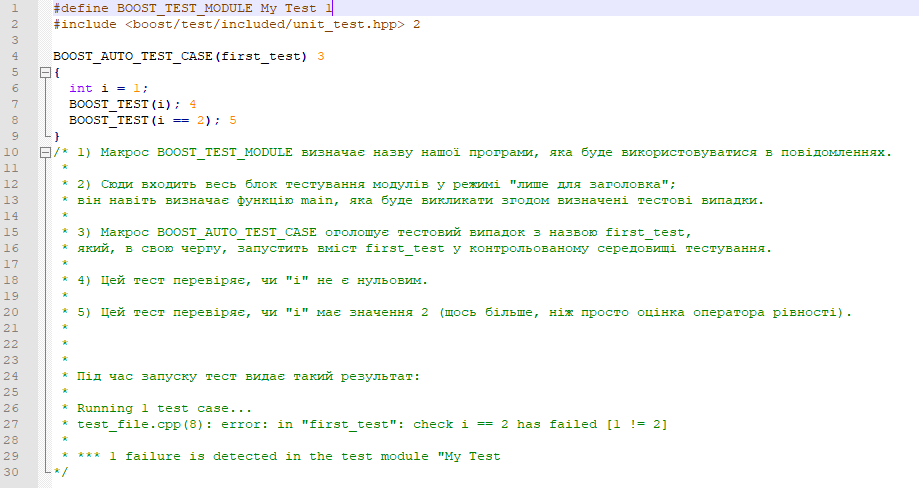


Рисунок 2.4 – Приклад використання Boost.Test

2.4 Catch & Trompeloil

2.4.1 Характеристика Catch & Trompeloil

Catch − це сучасна система тестування одиниць на основі C ++ 11 і вище. Багато його функцій на сьогоднішній день перевершили Qt Test. Найбільш важливим є те, що тести Catch будуються таким чином, щоб зменшити надмірність між тестовими кейсами.

Більше того, однією з ключових особливостей Catch є тестування розробкою, що керується поведінкою (BDD − Behavior-driven development), тобто тести структуровані таким чином, як ви, природно, описали б особливість продукту.

На жаль, Catch не підтримує «Gherkin language», у відмінності від схеми тестування Python BDD. Тим не менше, функція тестування BDD − це чудова і, безумовно, найкраща реалізація для C++[7]. Catch настільки простий у використанні, що підштовхує користувача писати тести частіше.

Оскільки написання тестових пар вручну − громіздке завдання, рекомендується використовувати макетну рамку.

Trompeloeil − безпечний для потоків макетний фреймворк для C++11/14, що використовує ліцензію Boost Software License 1.0. Trompeloeil виділяється з натовпу, оскільки він використовує сучасні функції C++14 і добре поєднується з Catch. Ідея Trompeloeil полягає в тому, щоб зробити тестування приладів якомога простішим, щоб люди не цуралися цього робити.

2.4.2 Підключення і використання Catch & Trompeloil

Для підключення:

1. створіть у підкаталозі «3rdparty/catch» файл, який називається «catch.pri», та файл під назвою «trompeloeil.pri» у каталозі «3rdparty/trompeloeil» із наступним вмістом: «INCLUDEPATH + = $$ PWD»;
2. далі нам потрібно змінити наш файл «test.pro» тесту підпроекту:
3. включіть наші «.pri» файли;
4. додайте підтримку C++14 для trompeloeil;
5. включіть наш каталог додатків;
6. додайте «main.cpp» для Catch, його зміст зображено на рисунку 2.5;
7. вигляд файлу після модифікацій зображено на рисунку 2.6.
8. змініть вміст автоматично створеного файлу «tst\_apptests.cpp» згідно з рисунком 2.7.

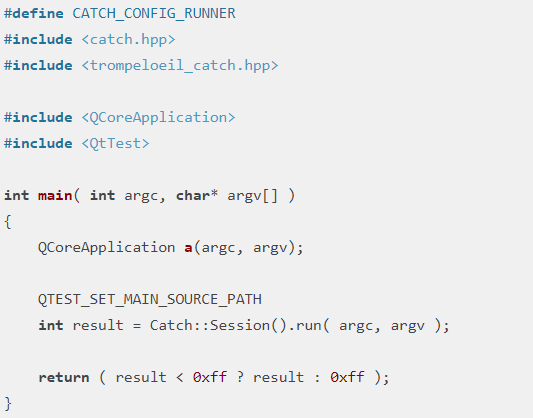


Рисунок 2.5 – Вміст файлу «main.cpp»

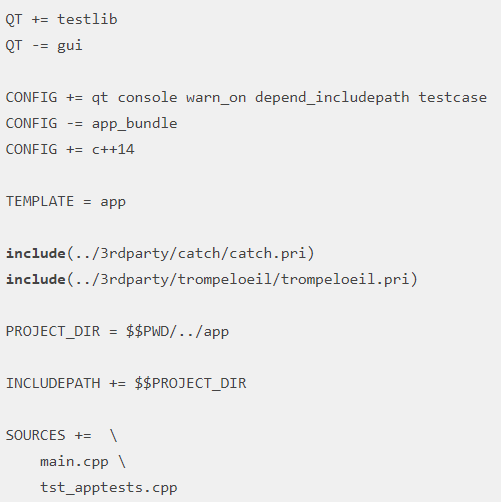


Рисунок 2.6 – Вміст файлу «test.pro»



Рисунок 2.7 – Вміст файлу «tst\_apptests.cpp»

Після запуску додатку в Qt Creator, ми отримаємо повідомлення від Catch, що зображено на рисунку 2.8

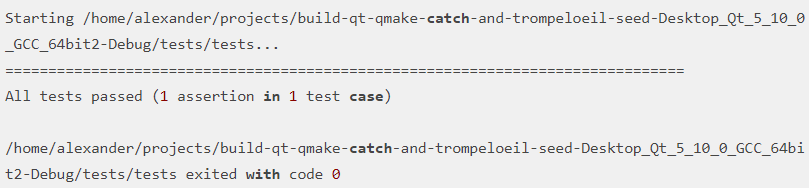


Рисунок 2.8 – Результат запуску тестового додатку з використанням Catch & Trompeloil

Висновки

В рефераті було розглянуто поняття модульного тестування, поняття фреймворків тестування, та Qt.

Наразі існує дуже багато різних фреймворків для тестування, тому були розглянуті лише актуальні фреймворки для модульного тестування для Qt, а саме:

* Qt Test;
* Google Test;
* Boost.Tes;
* Catch & Trompeloil.

Також було надано описано характеристики, надано кроки, які необхідно виконати для успішного підключення до Qt, та приклади використання розглянутих фреймворків.

СПИСОК ЛІТЕРАТУРИ

1. Wikipedia: Модульне тестування [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Модульне\_тестування;
2. Wikipedia: Qt [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Qt;
3. Qt Test Overview [Електронний ресурс]. – Режим доступу: https://doc.qt.io/qt-5/qtest-overview.html;
4. Google C++ Testing Framework [Електронний ресурс]. – Режим доступу: ru.wikipedia.org/wiki/Google\_C%2B%2B\_Testing\_Framework;
5. Boost library [Електронний ресурс]. – Режим доступу: https://www.boost.org/doc/libs/1\_70\_0/libs/test/doc/html/boost\_test/intro.html;
6. List of unit testing frameworks [Електронний ресурс]. – Режим доступу: h https://en.wikipedia.org/wiki/List\_of\_unit\_testing\_frameworks#C++;
7. Qt Unit Testing with Catch and Trompeloeil [Електронний ресурс]. – Режим доступу: https://machinekoder.com/qt-unit-testing-catch-trompeloeil/.