МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Кафедра програмних засобів

РЕФЕРАТ

з предмета «Якість програмного забезпечення та тестування»
на тему: **«Фреймворки тестування для Qt»**

Виконав:
студент групи КНТ–147сп Захряпа Р.А.
Перевірив: доцент Табунщик Г.В.

2019

ЗМІСТ

ВВЕДЕННЯ

У наш час створюється безліч різних програмних продуктів для різних сфер – від мобільних ігор до комерційних настільних додатків.

Проекти бувають дуже різні, від простих до позбавлених простоти, що при тестуванні створює великі проблеми та ускладнює як розробку їх самих, так і налагодження, знаходження багів. Існує як автоматизоване тестування так і ручне тестування. Автоматичне тестування виконується за допомогою додаткового програмного забезпечення. Ручне тестування виконується розробником програми або тестувальником, не використовуючи додаткового програмного забезпечення. Ручне тестування не є еталонним тестуванням, але використовується якщо у вас немає знань, або ж ресурсів (часу на вивчення, грошових коштів, співробітників). Один з фундаментальних принципів тестування говорить: 100% автоматизація неможлива. Тому, ручне тестування – це необхідність.

Автоматизоване тестування передбачає використання спеціального програмного забезпечення (крім тестованого) для контролю виконання тестів і порівняння очікуваного з фактичним результатом роботи програми. Цей тип тестування допомагає автоматизувати часто повторювані, але необхідні для максимізації тестового покриття завдання.

Мануальне тестування може бути повторюваним і нудним. У той же час, автоматизація може допомогти цього уникнути – за вас все зробить комп'ютер.

Тестування ділиться на тестування користувальницького інтерфейсу і тестування функцій програми.

Основні види тестування для програмних продуктів та існуючі модулі тестування, призначені для фреймворка Qt

Unit – обов'язковий рівень тестування. Юніт-тестування призначене для того, щоб перевірити, що весь базовий функціонал компонентів, сервісів працює справно і виконує своє завдання.

UI – це User Interface, в перекладі з англійської «призначений для користувача інтерфейс». Полягає в тому, що ми імітуємо дії користувача – кліки, переходи по посиланнях, і інші дії подібного плану. Сенс його – в перевірці взаємодії компонентів один з одним. [1]

Існують як безкоштовні, так і платні модулі для виконання тестування.

Безкоштовні модулі тестування:

– QTestlib;

– Catch

Платні модулі тестування:

– Squish;

Модуль для тестування QTestlib та його особливості

QTestlib – безкоштовний модуль, призначений для юніт-тестування та надання повного базового функціоналу для тестування графічного інтерфейсу користувача. Даний модуль надає набір макросів для тестування функцій програми. Має функціонал симуляції повідомлень від миші і клавіатури. Користувацькі типи можуть бути легко додані до тестових даних та їх результатів.

Є кілька методів проведення тестів:

– завести тестовий проект в дочірньої директорії вашого проекту і тестувати в ньому;

– тестувати макросом qExec () в основному проекті.

Якщо ви використовуєте qmake як інструмент для створення, просто додайте в файл проекту наступне: «QT += testlib».

Для запуску тестів модуль має наступний синтаксис:

testname [options] [testfunctions[:testdata]]

У нашому випадку необхідно буде ввести наступну команду до командного рядка: Test_Smart max, де Test_Smart – це клас, який описує наш тест, а max – це функція, що тестується.

Аргументи командного рядка, які можна використовувати при запуску тестів: [2]

Команда	Опис
–help	Виводить можливі варіанти команд та їх описання
–functions	Виводить усі тестові функції, які доступні в тесті
–datatags	Виводить усі теги даних, доступні в тесті. Глобальному тегу даних "__global__"
–o filename	Записує вихідні дані тесту у вказаний файл, замість стандартного виводу
–silent	Безшумне виконання: показує лише попередження або збої при виконанні тестів
–v1	Виводить детальну інформацію при вході та виході з тестових функцій
–v2	Розширений детальний звіт, що спрацьовує на кожній функції QCOMPARE() та QVERIFY()
–vs	Виводить кожен сигнал, що був отриманий
–xml	Виводить відформатовані результати у вигляді XML розмітки замість простого тексту

Команда	Опис
–lightxml	Виводить результати у вигляді потоку тегів XML, без форматування тегів
–eventdelay ms	Якщо для моделювання клавіатури або миші не вказано затримки (QTest::keyClick(), QTest::mouseClick()), значення цього параметра (у мілісекундах) буде замінено за допомогою цього параметру

Для створення тесту перевизначите QObject і додайте один або кілька закритих слотів в ньому. Кожен закритий слот є функцією вашого тесту. QTest::qExec() та використовується для виконання всіх функцій тестів в об'єкті тесту.

Функції QTestlib які викликаються автоматично:

– initTestCase() виконується перед запуском першої функції тесту;

– cleanupTestCase() виконується після виконання останньої функції тесту;

– init() виконується перед запуском кожної функції тесту;

– cleanup() виконується після кожної функції тесту.

Якщо initTestCase() повернув помилку, значить немає функцій тестів, які можуть бути запущені. Якщо init() повернув помилку, це свідчить про те, що тест не був запущений, після чого тест перейде до наступної функції.

Для тестування функцій програми будемо використовувати наступні методи:

– QCOMPARE(actual, expected) – де actual – наш результат, а expected – це те, що ми очікуємо отримати, у разі відхилення – тест помічається як той , що не пройшов контроль;

– QVERIFY(condition) – макрос перевіряє правдива умова чи ні. Якщо умова правдива, то виконання продовжується, якщо ні -зберігається як помилка;

– QVERIFY2(condition, message) – працює аналогічно QVERIFY(condition), за винятком того, що воно виводить повідомлення, яке ви запишите у другому параметрі;

– QFAIL(message) – для того, щоб відобразити помилку, яку ви могли очікувати;

– QSKIP(description) – макрос зупиняє виконання тесту, не додаючи помилку до журналу тестів;

– QWARN(message) – додає повідомлення як попередження до журнала тестів. [3]

Приклад для юніт-тестування:

Створимо клас, який має назву Smart, для знаходження максимального значення серед цілих чисел – метод (max(int, int)). Для перевірки правильності роботи класу Smart створемо новий клас та назвемо його Smart_Test, що має метод max() для того, щоб протестувати функцію max() з декількома варіантами на вході.

Для запуску тесту необхідно ввести до командного рядка наступне: Test_Smart max

Клас який ми будемо перевіряти:

smart.h

```
#ifndef SMART_H
#define SMART_H

#include <QObject>
#include <QStringList>

class Smart : public QObject
{
    Q_OBJECT
public:
    explicit Smart(QObject *parent, const QStringList& list);

public slots:
    int max(int a, int b);
    int min(int a, int b);

};

#endif // SMART_H
```

smart.cpp

```
#include "smart.h"

Smart::Smart(QObject *parent, const QStringList& list) :
    QObject(parent)
{
}

int Smart::max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}

int Smart::min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
```

Клас який виконує перевірку має наступний вигляд:

test_smart.h

```
#ifndef TEST_SMART_H
#define TEST_SMART_H

#include <QObject>

class Test_Smart : public QObject
{
    Q_OBJECT
public:
    explicit Test_Smart(QObject *parent = 0);

private slots: // должны быть приватными
    void max(); // int max(int, int)

};

#endif // TEST_SMART_H
```

test_smart.cpp

```
#include <QTest>
#include "test_smart.h"
#include "smart.h"

Test_Smart::Test_Smart(QObject *parent) :
    QObject(parent)
{
}

void Test_Smart::max()
{
    Smart a;
    QCOMPARE(a.max(1,   0), 1);
    QCOMPARE(a.max(-1,  1), 1);
    QCOMPARE(a.max(4,   8), 8);
    QCOMPARE(a.max(0,   0), 0);
    QCOMPARE(a.max(1,   1), 1);
    QCOMPARE(a.max(-10,-5), -5);
}
```

Головний файл програми та результати тестування

main.cpp

```cpp
#include <QApplication>
#include <QTest>
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include "test_smart.h"

using namespace std;

int main(int argc, char *argv[])
{
    freopen("testing.log", "w", stdout);
    QApplication a(argc, argv);
    QTest::qExec(new Test_Smart, argc, argv);
    return 0;
}
```

testing.log

```
********* Start testing of Test_Smart *********
Config: Using QTest library 4.8.1, Qt 4.8.1
PASS   : Test_Smart::initTestCase()
PASS   : Test_Smart::max()
PASS   : Test_Smart::cleanupTestCase()
Totals: 3 passed, 0 failed, 0 skipped
********* Finished testing of Test_Smart *********
```

Дуже часто доводиться тестувати графічний інтерфейс. У QTestLib це теж реалізовано. Давайте протестуємо QLineEdit (це компонент для введення тексту). Для цього додамо до нашої форми QLineEdit компонент та напишемо для нього клас для перевірки його значення.

Приклад GUI тестування:

Код для **test_qlineedit.h**

```cpp
#ifndef TEST_QLINEEDIT_H
#define TEST_QLINEEDIT_H

#include <QObject>

class Test_QLineEdit : public QObject
{
    Q_OBJECT
private slots: // должны быть приватными
    void edit();

};

#endif // TEST_QLINEEDIT_H
```

Код для **test_qlineedit.cpp**

```cpp
#include <QtTest>
#include <QtGui>
#include "test_qlineedit.h"

void Test_QLineEdit::edit()
{
    QLineEdit a;
    QTest::keyClicks(&a, "abCDEf123-");

    QCOMPARE(a.text(), QString("abCDEf123-"));
    QVERIFY(a.isModified());
}
```

Код, що запускає наші тести	Результати виконання нашого коду

```
#include <QApplication>
#include <QTest>
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include "test_smart.h"
#include "test_qlineedit.h"

using namespace std;

int main(int argc, char *argv[])
{
    freopen("testing.log", "w", stdout);
    QApplication a(argc, argv);
    QTest::qExec(new Test_Smart, argc, argv);
    cout << endl;
    QTest::qExec(new Test_QLineEdit, argc, argv);
    return 0;
}
```

```
********* Start testing of Test_Smart *********
Config: Using QTest library 4.8.1, Qt 4.8.1
PASS   : Test_Smart::initTestCase()
PASS   : Test_Smart::max()
PASS   : Test_Smart::min()
PASS   : Test_Smart::cleanupTestCase()
Totals: 4 passed, 0 failed, 0 skipped
********* Finished testing of Test_Smart *********

********* Start testing of Test_QLineEdit *********
Config: Using QTest library 4.8.1, Qt 4.8.1
PASS   : Test_QLineEdit::initTestCase()
PASS   : Test_QLineEdit::edit()
PASS   : Test_QLineEdit::cleanupTestCase()
Totals: 3 passed, 0 failed, 0 skipped
********* Finished testing of Test_QLineEdit *********
```

Тест показав, що QLineEdit працює як заплановано.

Програмний продукт для тестування Squish та його особливості

Squish – кращий інструментрій для функціональних, регресійних, системних тестів та тестів користувацького інтерфейсу (GUI) і людино–машинних інтерфейсів (HMI). Squish – це кросплатформний інструмент тестування. Ліценція коштує близько 2650 евро на рік.

Переваги програмного продукту Squish:

– всебічна підтримка всіх основних технологій GUI;

– повна підтримка всіх настільних, мобільних, веб і вбудованих платформ;

– запис тестового скрипта та його прегляд;

– потужна і надійна ідентифікація об'єктів та їх перевірка;

– немає залежності від екранної форми додатка або зовнішнього вигляду;

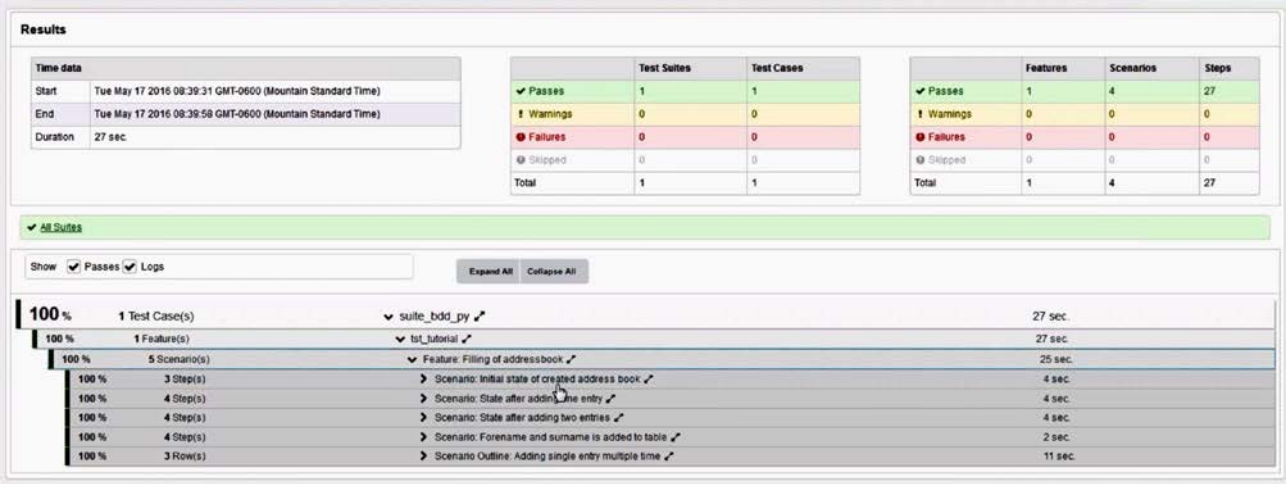
– потужна інтегрована середовище розробки (IDE);

– широкий вибір популярних мов сценаріїв для сценаріїв тестування;

– повна підтримка розробка через тестування поведінки (BDD);

– повний контроль за допомогою інструментів командного рядка. [4]

Приклад звіту програмного продукту Squish:



Фреймворк для тестування Catch та його особливості

Catch – простий модуль, що є безкоштовним та призначений для юніт-тестування.

Переваги фреймворка Catch:

– нуль залежностей;

– цілий фреймворк у одному заголовку;

– підтримка TDD і BDD;

– дуже простий і лаконічний;

– доступна документація.

Функції модуля Catch:

REQUIRE(expression) та CHECK(expression) – для перевірки вірності виразу, у разі якщо вираз не вірний, то тест буде зарахований як не вірний. [5]

Для роботи необхідно додати до головного файлу (main.cpp) вашого проекту наступні рядки:

#define CATCH_CONFIG_MAIN

#include "catch/catch.hpp"

Приклад тесту модуля Catch:

Файл з тестом

```cpp
// some_tests.cpp
#include "catch/catch.hpp"
#include <QString>

TEST_CASE("Some.Tests", "")
{
    SECTION("Something", "")
    {
        CHECK(QString() == QString());
        const QString someString(QLatin1String("foo"));
        CHECK(someString == QLatin1String("bar"));
    }
}
```

Результати тесту

```
catch_qt is a Catch v1.12.1 host application.
Run with -? for options

-------------------------------------------------------------------------------
Some.Tests
  Something
-------------------------------------------------------------------------------
some_tests.cpp:8
.....................................................................

some_tests.cpp:13: FAILED:
  CHECK( someString == QLatin1String("bar") )
with expansion:
  {?} == {?}

=============================================================================
test cases: 1 | 1 failed
assertions: 3 | 2 passed | 1 failed
```

Для детальнішої інформації про тест необхідно додати наступний код

```cpp
inline std::ostream &operator<<(std::ostream &os, const QByteArray &value)
{
    return os << '"' << (value.isEmpty() ? "" : value.constData()) << '"';
}

inline std::ostream &operator<<(std::ostream &os, const QLatin1String &value)
{
    return os << '"' << value.latin1() << '"';
}

inline std::ostream &operator<<(std::ostream &os, const QString &value)
{
    return os << value.toLocal8Bit();
}
```

Результат після виконання

```
some_tests.cpp:13: FAILED:
  CHECK( someString == QLatin1String("bar") )
with expansion:
  "foo" == "bar"
```

Для запуска тесту необхідно ввести наступний рядок:
TestName –c SectionName –c SectionName
Приклад:
Some.Tests –c Something

Команди для командного рядка:

Команда	Опис
–h, –?, —help	Виводить додаткову інформацію для всіх команд
–l, —list–tests	Виводить список тестів
–t, —list–tags	Виводить список тестів із тегом (тегами)
–s, —success	Виводить список тестів, що пройшли перевірку
–b, —break	Відображає стан тесту у разі збою проекта
–r, —reporter	Надає варіанти виведення результатів тесту у наступних форматах: console, compact, xml, junit

Команда	Опис
–w, —warn	Вмикає повідомлення про підозрілі тестові стани. У разі відсутності тесту або у разі відсутності результата
–d, —durations	Якщо встановлено значення True, Catch повідомляє про тривалість кожного тестового випадку
–c, —section	При запуску теста можно вказати конкретні секції, які ви хотіли би запустити

Створення проекту з тестами модуля Catch:

1. створюєте основний проект;
2. у папці основного проекту створюєте проект tests, там з'явиться папка tests з самим тестовим проектом
3. чи працюєте в тестовому проекті.
4. коли все готово, додаєте в основний .pro–файл рядок: subdirs + = tests.
5. тепер при складанні проекту, будуть збиратися тести.

Переваги модуля Squish:

– тісна дружба Squish с класами Qt;
– кросплатформеність;
– підтримка скриптових мов (JavaScript, Python);
– автоматизована генерація текста тесту;
– зручна система запуску тестів з консолі.

ВИСНОВКИ

Таким чином можна зробити висновки, що використання додаткового програмного забезпечення (модулів, фреймворків), такого як QTestlib, Squish, Catch, допомагає зберігти час, полегшує та спрощує тестування різних програмних продуктів.

Можна надати перевагу модулям тестування QTestlib та Catch у разі якщо у вас немає додаткових коштів на платні аналоги, такі як Squish.

Модуль QTestlib зручний для швидкого старту та для проектів, що не є дуже великими за розміром та кількістю модулів.

Фреймворк Catch має дуже багато можливостей для створення звітів та виведенням додаткової інформації.

Для комерційного використання програмний продукт Squish буде кращим варіантом, оскільки він має більш детальну інформацію до кожного тесту і має зручний відформатований звіт з усією інформацією. Squish має дуже зручний інтерфейс для перегляду створених тестів.

Автоматизація зберігає час, сили і гроші. Автоматизований тест можна запускати знову і знову, докладаючи мінімум зусиль.

ПЕРЕЛІК ПОСИЛАНЬ

1 Описание UI и Unit-тестирования [Электрон. ресурс]. – Режим доступа: https://habr.com/ru/post/335776/

2 Options to run your tests from command line [Электрон. ресурс]. – Режим доступа: https://doc.qt.io/archives/qt-4.8/qtestlib-manual.html

3 QTestlib functions and their purposes [Электрон. ресурс]. – Режим доступа: https://doc.qt.io/archives/qt–4.8/qtest.html

4 Squish Automated GUI Testing [Электрон. ресурс]. – Режим доступа: https://www.froglogic.com/squish/

5 Catch and its functions [Электрон. ресурс]. – Режим доступа: https://github.com/catchorg/Catch2/blob/master/docs/assertions.md#top