ECE 3574: Applied Software Design

Integration Testing

Today we will take a look at integration testing and QtTest.

- Techniques for testing command-line applications
- GUI Testing using QtTest
- Examples
- Exercise

Recall our discussion of unit tests

- Unit tests exercise each module, generally a class and associated functions.
- Treat the public interface as a contract. Your test code checks the contract.

Integration Tests

Integration tests verify the function of assemblies of modules or an application overall.

Functional testing of non-interactive applications

- Non-interactive applications which read files and write files specified through arguments are easy to test.

You write another application to read the output and compare it to the expected output.

For example consider a non-interactive application that reads inputfile and writes an outputfile taken as command-line arguments. In CMake

```
add_executable(the_app the_app.cpp)
add_exectuable(compare_tool compare_tool.cpp)
add_test(runtest1 the_app inputfile outputfile)
add_test(comparetest1 compare_tool outputfile
        ${CMAKE_SOURCE_DIR}/output1.expected)
```

where the file `outputfile.expected` lives in the source directory.

Functional testing of interactive Text-Mode applications

For simple interactive applications you can pipe in standard input and pipe out standard output.

```
$ my_exe < stdin_file > stdout_file
```

For more complex interactive text-mode applications, e.g. a REPL, you can use a scripting language like Expect (Tcl) or Pexpect (Python).

Testing using QtTest

Tests are defined as the private slots of a class derived from QObject.

A simple example using a single cpp file: mytest.cpp

```
class MyTest: public QObject
{
  Q_OBJECT

private slots:

  // define as many tests as you like
  void test1() { QVERIFY(true); };
};

QTEST_MAIN(MyTest)
#include "mytest.moc"
```

This could be used for unit tests in the same way as Catch.

Assertions in QtTest are similar to those in other testing frameworks

```
QCOMPARE(actual, expected)
QVERIFY(condition)
QVERIFY2(condition, message)
QVERIFY_EXCEPTION_THROWN(expression, exceptiontype)
```

Testing a Qt GUI

QTTest can be used for general testing but it really shines for Qt GUI testing because it can plug into the object tree and signal-slot mechanism.

You can simulate Clicks and KeyPress events to get objects to handle events and emit signals as if they were triggered manually.

- QTest::keyClick()
- QTest::keyPress(), QTest::keyRelease()
- QTest::mouseClick(), etc.

See the Qt documentation for details.

To simulate events on a widget you need a pointer to it.

You can search for widgets using QObject (templated) find members

```
T findChild(const QString &name)
QList<T> findChildren(const QString &name)
QList<T> findChildren(const QRegularExpression &re)
```

where the argument is the (optional) name property of the widget being searched for or a Perl-compatible regular expression for matching names.

- T is the sub-type of QObject
- by default this is done recursively

Example: find a pointer to a widget by type alone

See `TestExampleWidget::testFindByType` in `test_example_widget.cpp`

Example: find a pointer to a widget by name alone

See `TestExampleWidget::testFindByName` in `test_example_widget.cpp`

Example: find a pointer to some widgets by regular expression

See `TestExampleWidget::testFindByTRegExp` in `test_example_widget.cpp`

Integration of CMake and QtTest

Similar to configuring any Qt app from CMake

```
set(CMAKE_AUTOMOC ON)
set(CMAKE_INCLUDE_CURRENT_DIR ON)
find_package(Qt5 COMPONENTS Test REQUIRED)

add_executable(mytest mytest.cpp)
target_link_libraries(mytest Qt5::Test)

enable_testing()
add_test(mytest mytest)
```

You run the tests manually or through cmake (output goes in the same place `Testing/Temporary/LastTest.log`).

Exercise

See website

Next Actions and Reminders

- Read about Design Patterns