ICS

Qt Test-Driven Development Using Google Test and Google Mock

Justin Noel, Senior Consulting Engineer

Webinar Contents

- Light introduction to Google Test / Google Mock
  - See bibliography for more information
- Light introduction to TDD
  - See bibliography for more information
- Designing applications for testability
  - Isolation of units
- How to test with classes that use Qt
  - Using Google Test with Qt Creator
  - QObject, Qt Data types, Signals, Events

Types of Testing

- Unit Testing
  - Test one discrete area of the software
    - Usually on a class basis. Think “Test Piston Rod”
  - Tests can be written and performed as code is developed
    - White box tests. Often written by developers
  - “Does the software perform as the developer intended?”
- Functional Testing
  - Tests subsystems of the whole software system
    - Usually a group of classes. Think “Test Engine”
  - Tests can be preformed once subsystem is wired together
    - Tests interaction between specific groups of units
  - “Does the software work together?”

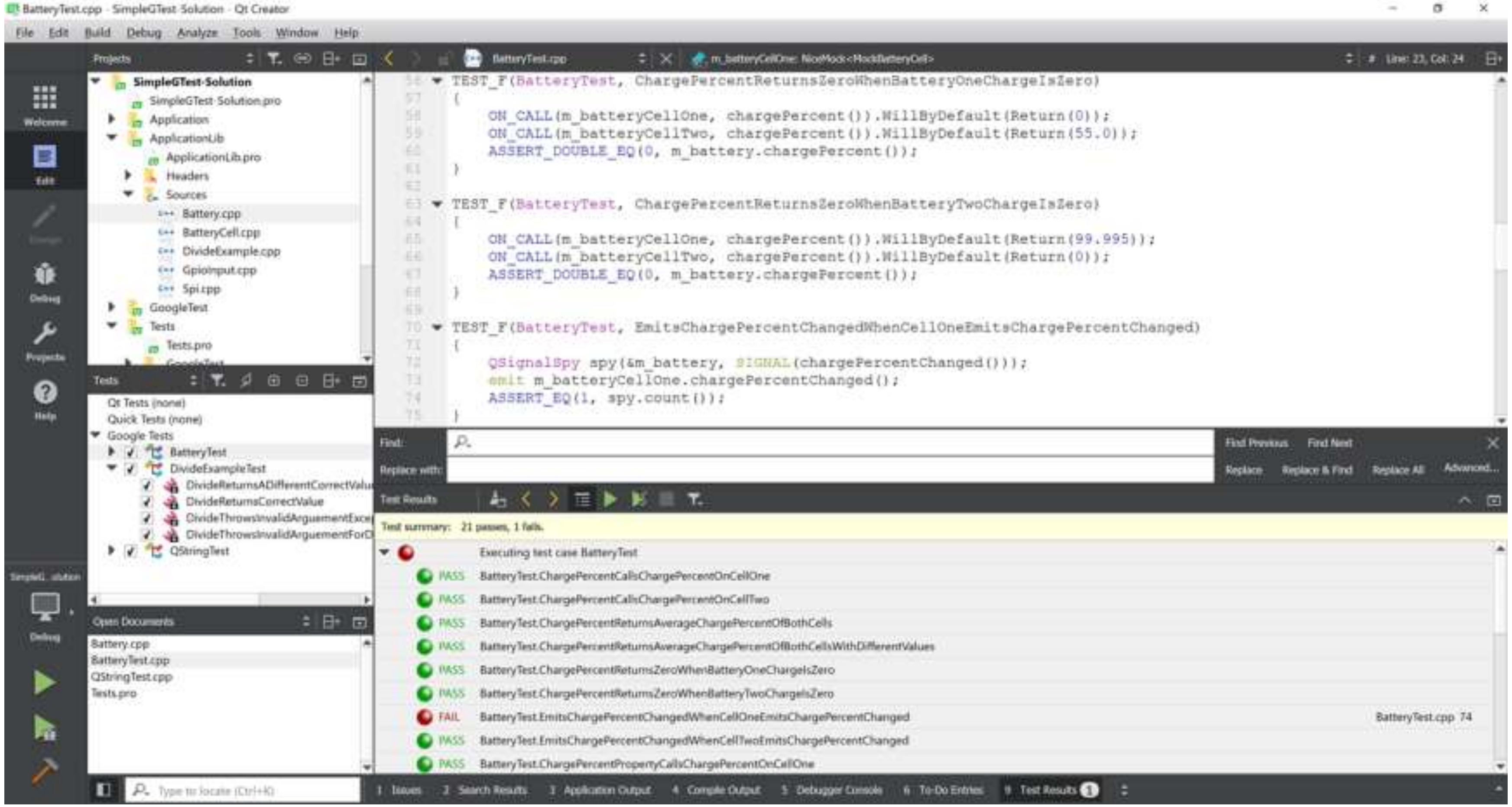
Types of Testing

- System Testing
  - End to end testing of the whole software system
    - Sometimes includes actual hardware. Think “Test Car”
  - Tests are written late in project
    - Because of the end to end nature
    - Black box tests. Often written by separate SQA team
  - “Does the software do the right thing?”

Google Test Creator Integration

- Unified AutoTest Plugin (Qt Creator 4.0+)
  - QTest
  - QTest-QML
  - GoogleTest
- Provides new Test Results output pane
  - Tree of Test Fixtures and Test Functions
  - Color codes success vs failure
  - Contains failure messages from Google Test
- Provide Tests project view
  - Select which tests to run

Google Test Creator Plugin



```
TEST_F(BatteryTest, ChargePercentReturnsZeroWhenBatteryOneChargeIsZero)
{
    ON_CALL(m_batteryCellOne, chargePercent()).WillByDefault(Return(0));
    ON_CALL(m_batteryCellTwo, chargePercent()).WillByDefault(Return(55.0));
    ASSERT_DOUBLE_EQ(0, m_battery.chargePercent());
}

TEST_F(BatteryTest, ChargePercentReturnsZeroWhenBatteryTwoChargeIsZero)
{
    ON_CALL(m_batteryCellOne, chargePercent()).WillByDefault(Return(99.995));
    ON_CALL(m_batteryCellTwo, chargePercent()).WillByDefault(Return(0));
    ASSERT_DOUBLE_EQ(0, m_battery.chargePercent());
}

TEST_F(BatteryTest, EmitsChargePercentChangedWhenCellOneEmitsChargePercentChanged)
{
    QSignalSpy spy(&m_battery, SIGNAL(chargePercentChanged()));
    emit m_batteryCellOne.chargePercentChanged();
    ASSERT_EQ(1, spy.count());
}
```

```
Test summary:  21 passes, 1 fails.
        Executing test case BatteryTest
  PASS  BatteryTest.ChargePercentCallsChargePercentOnCellOne
  PASS  BatteryTest.ChargePercentCallsChargePercentOnCellTwo
  PASS  BatteryTest.ChargePercentReturnsAverageChargePercentOfBothCells
  PASS  BatteryTest.ChargePercentReturnsAverageChargePercentOfBothCellsWithDifferentValues
  PASS  BatteryTest.ChargePercentReturnsZeroWhenBatteryOneChargeIsZero
  PASS  BatteryTest.ChargePercentReturnsZeroWhenBatteryTwoChargeIsZero
  FAIL  BatteryTest.EmitsChargePercentChangedWhenCellOneEmitsChargePercentChanged        BatteryTest.cpp  74
  PASS  BatteryTest.EmitsChargePercentChangedWhenCellTwoEmitsChargePercentChanged
  PASS  BatteryTest.ChargePercentPropertyCallsChargePercentOnCellOne
```

Enabling Auto Test Plugin

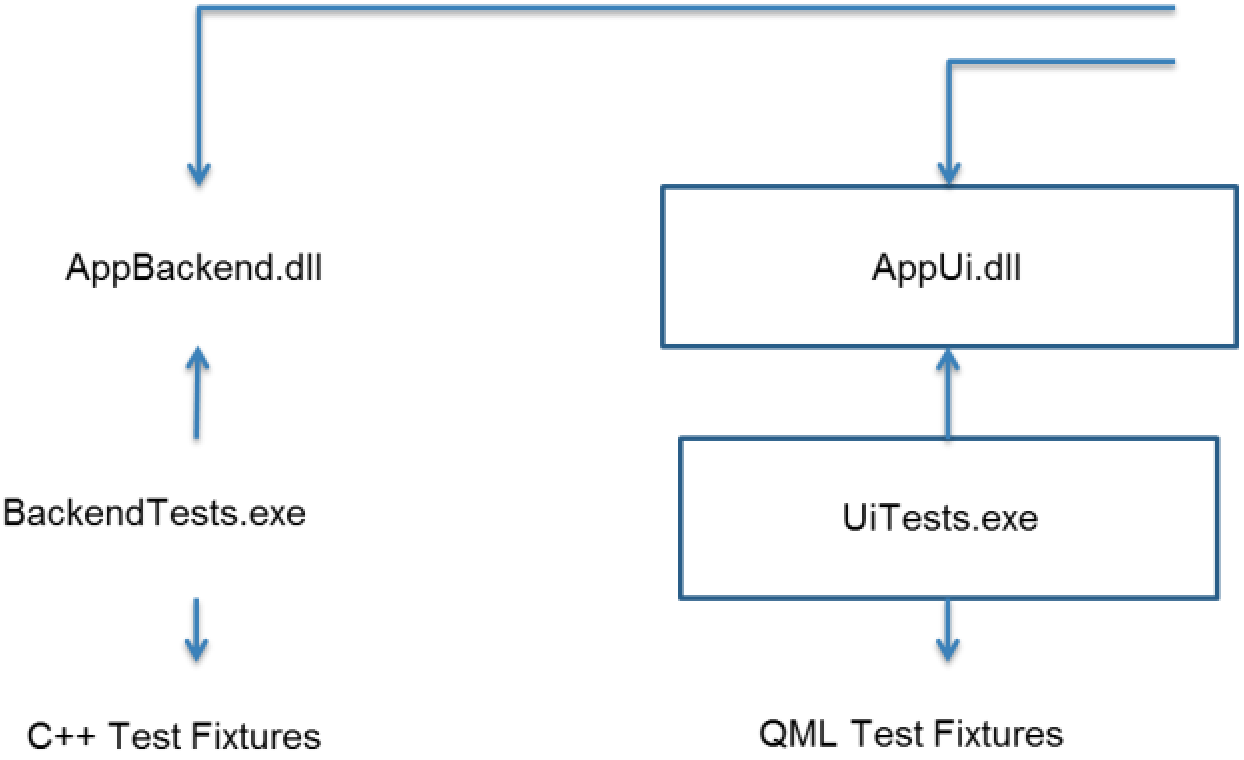
- Start Qt Creator and enable the Auto Test plugin
  - Help -> About Plugins
    - Check Load for AutoTest under Utilities
- Restart Qt Creator
- You now have the following available
  - Test Results Output Pane
  - Tests Project View
  - Test Settings Pane under Tools -> Options
    - Show/Hide debug/warning console output
    - GoogleTest specific settings are here

Design for Testability

- Your code must be testable!
  - There are design techniques to help make your code more testable
  - Try not to add functions to production code just for tests
    - Sub classing for tests is acceptable (access designer UI members)
- Isolation of units is your primary goal
  - Can I just test this class?
    - Without re-testing lower level classes?
- Also think about error conditions
  - Some may be hard to produce in production environments
    - Can I stimulate a bad MD5 error?
    - Socket disconnects?
    - Default switch cases?

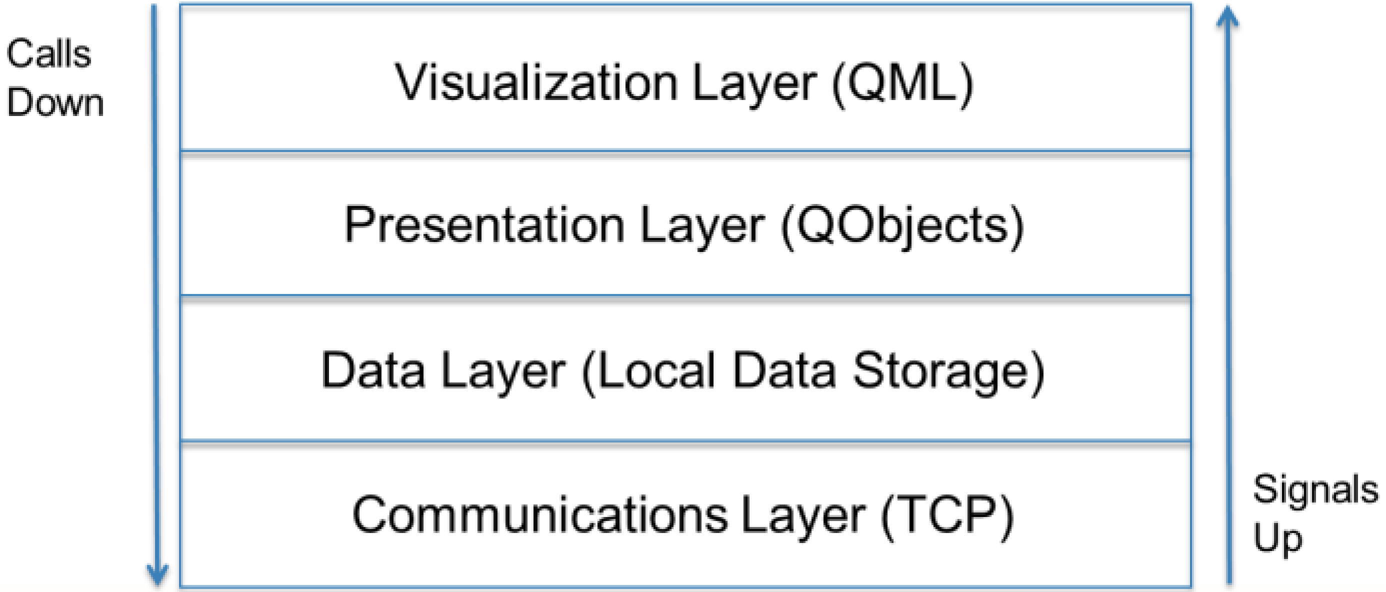
Build Application as Libraries

Building the bulk of your application as a set of libraries increases testability



Layered Design

- A layered design is more testable
  - Easy to isolate lower layers for testing
    - Test Communications without Data, Presentation or Visualization
  - No upwards dependencies
  - Hard downward dependencies



Calls Down

Visualization Layer (QML)

Presentation Layer (QObjects)

Data Layer (Local Data Storage)

Communications Layer (TCP)

Signals Up

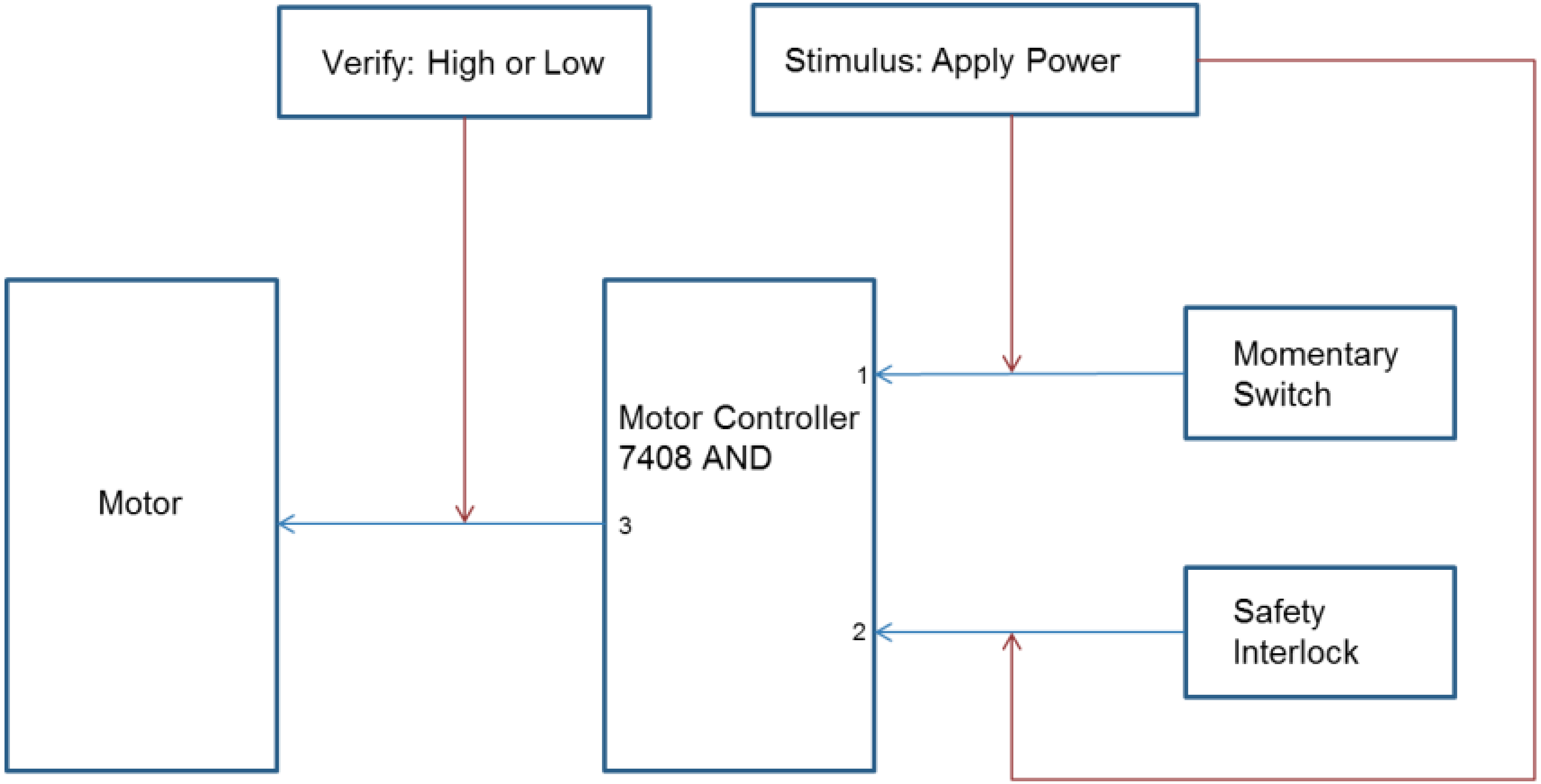
Break Downward Dependencies

- Dependency injection works well
  - An Inversion of Control Pattern
  - Classes take their dependencies as constructor arguments
    - Classes do not construct any members themselves
- Loose coupling with signals and slots also works well
  - Classes interface to each other via signals and slots only
    - 3rd class wires say the UI classes to the Backend classes
- Lets you substitute a test fixture object for a dependency object
  - Instead of an actual production object

Why is isolation important?

- Avoids testing the same code over and over
  - You have already tested the data model
    - Why do the UI tests need to also test the model?
  - This will make your tests run very quickly
  - If there is a failure in a low level class it won’t trigger errors in higher level tests
- Avoid testing side effects
  - Tests can be implementation agnostic
    - DB, Files, Cloud. Shouldn’t matter for UI tests.
  - Tests should only depend on interfaces, not behavior

EE Analogy: Motor Controller Test



Google Test/Mock Libraries

- Google Test and Google Mock are C++ libraries
  - New version (1.8) combines gtest and gmock in one project
    - Called “googletest”
  - Link to them as you would any other C++ library
    - Only your test executables should need gtest and gmock libs
      - g[test|mock]_main and libraries offer a prebuilt main function
- Build googletest using Cmake. make && make install
  - Use from a known location like Qt
- Build GoogleTest inside your project using short .pro/.pri
  - I prefer this to avoid compiler flag incompatibilities
    - Windows is famous for mix/match incompatibilities
  - Qt’s GoogleTest project template does this method

Google Test/Mock In Project

Build Google Test and Google Mock with this short .pro

```
TEMPLATE = lib
CONFIG += static exceptions
CONFIG -= debug_and_release

TARGET = GoogleTest

INCLUDEPATH += \
    googletest/googletest/include \
    googletest/googlemock/include \
    googletest/googletest \
    googletest/googlemock

SOURCES = \
    googletest/googletest/src/gtest-all.cc \
    googletest/googlemock/src/gmock-all.cc
```

GTest is a unit testing framework

- Prepare
  - Create dependencies
  - Create object under test
  - Setup expectations
    - Mock Expectations, QSignalSpy, etc
- Stimulate
  - Call a 1 function or emit a signal
- Verify
  - Check that expectations occurred
  - And/or check return value

GTest Verifications

- ASSERT_EQ(a, b) – Equality Operator
  - Works for QString, std::string, etc
    - QString needs a “Pretty Printer” to be truly integrated
- ASSERT_TRUE(exp) – Boolean Expression
- ASSERT_DOUBLE_EQ(a, b) – Fuzzy Compare
- ASSERT_FLOAT_EQ(a, b) – Fuzzy Compare
- ASSERT_STREQ(a, b) – char* comparison

Expection, Assert, Abort, Exit

- EXPECT_THROW(foo(), execptionType)
- EXPECT_NO_THROW(foo(), exceptionType)
- ASSERT_DEATH(foo())
  - Fails if program did not exit() or abort()
    - assert(), Q_ASSERT(), etc will abort()
  - Very useful for testing “Does CTRL-C exit my program”
- These types of tests can pass on unhandled exceptions and exiting. Tests will continue on as usual.
  - This is something that is hard to do in QtTest.

Qt Specific Things

- Not supplied by Google Test
  - QtTest does supply these. Link to QtTest and use them.
- QSignalSpy
  - Test for signal emits
- QTest::mouseClick, QTest::keyClicks, etc
  - Stimulate Qt events

Print Qt Data Types in Output

- Google Test would like to print text for objects using ASSERT_EQ
  - std::ostream << operators work
  - Or implement a PrintTo function
  - Put this in a header file and include it in your test files

```
QT_BEGIN_NAMESPACE
inline void PrintTo(const QString &qString,
::std::ostream *os)
{
    *os << qPrintable(qString);
}
QT_END_NAMESPACE
```

Creating a Test Fixture

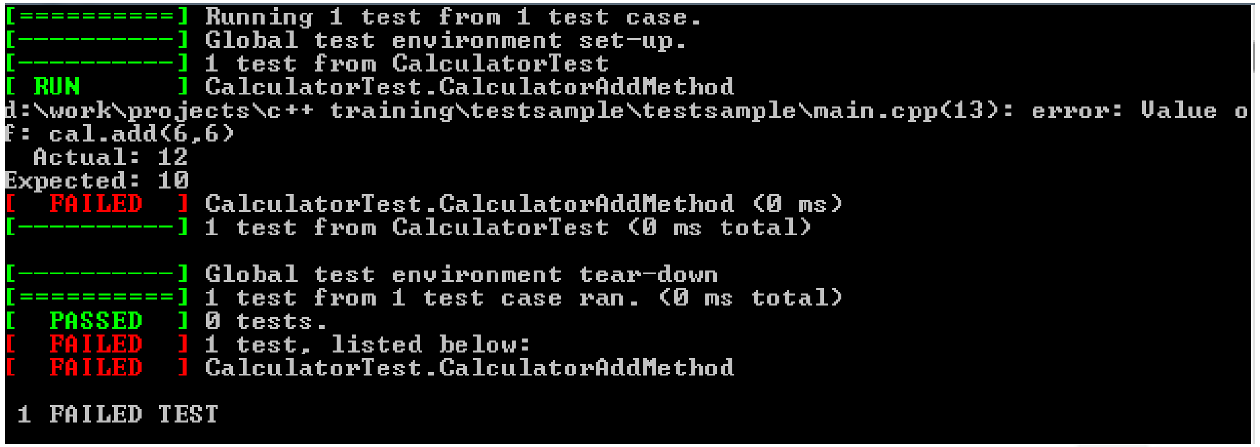
```
#include <gtest/gtest.h>
using namespace ::testing;

// Fixture
class ExampleTest : public Test
{
protected:
    Example m_example; // Object Under Test
};

// Test Function
TEST_F(ExampleTest, ThisShouldFail)
{
    ASSERT_TRUE(false);
}
```

New Objects For Every Test Function

- A new Fixture is created for every test function
  - All new objects are created for each test
  - Helps ensure that tests do no depend on each other
    - Or pollute the environment for future tests
  - Can use constructor and destructor to setup tests
- Unless there may be exceptions thrown
  - Fixtures have Setup() and TearDown() functions to handle that case
    - Some projects have policy to only use Setup() and TearDown() with empty constructor and destructor

Test Fixture Project

```
QT += test # QtTest includes QSignalSpy, event stimulators, etc
CONFIG += console # Create an stdout window on Windows
CONFIG += testcase   # Creates make check target
INCLUDEPATH += ../GoogleTest/include
LIBS += -L../GoogleTest/lib -lGoogleTest # Lib name(s) depends on build
SOURCES += TestQString.cpp

qmake
make
LD_LIBRARY_PATH=<paths> make check
```



```
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from CalculatorTest
[ RUN      ] CalculatorTest.CalculatorAddMethod
d:\work\projects\c++ training\testsample\testsample\main.cpp(13): error: Value o
f: cal.add(6,6)
  Actual: 12
Expected: 10
[  FAILED  ] CalculatorTest.CalculatorAddMethod (0 ms)
[----------] 1 test from CalculatorTest (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (0 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] CalculatorTest.CalculatorAddMethod

 1 FAILED TEST
```

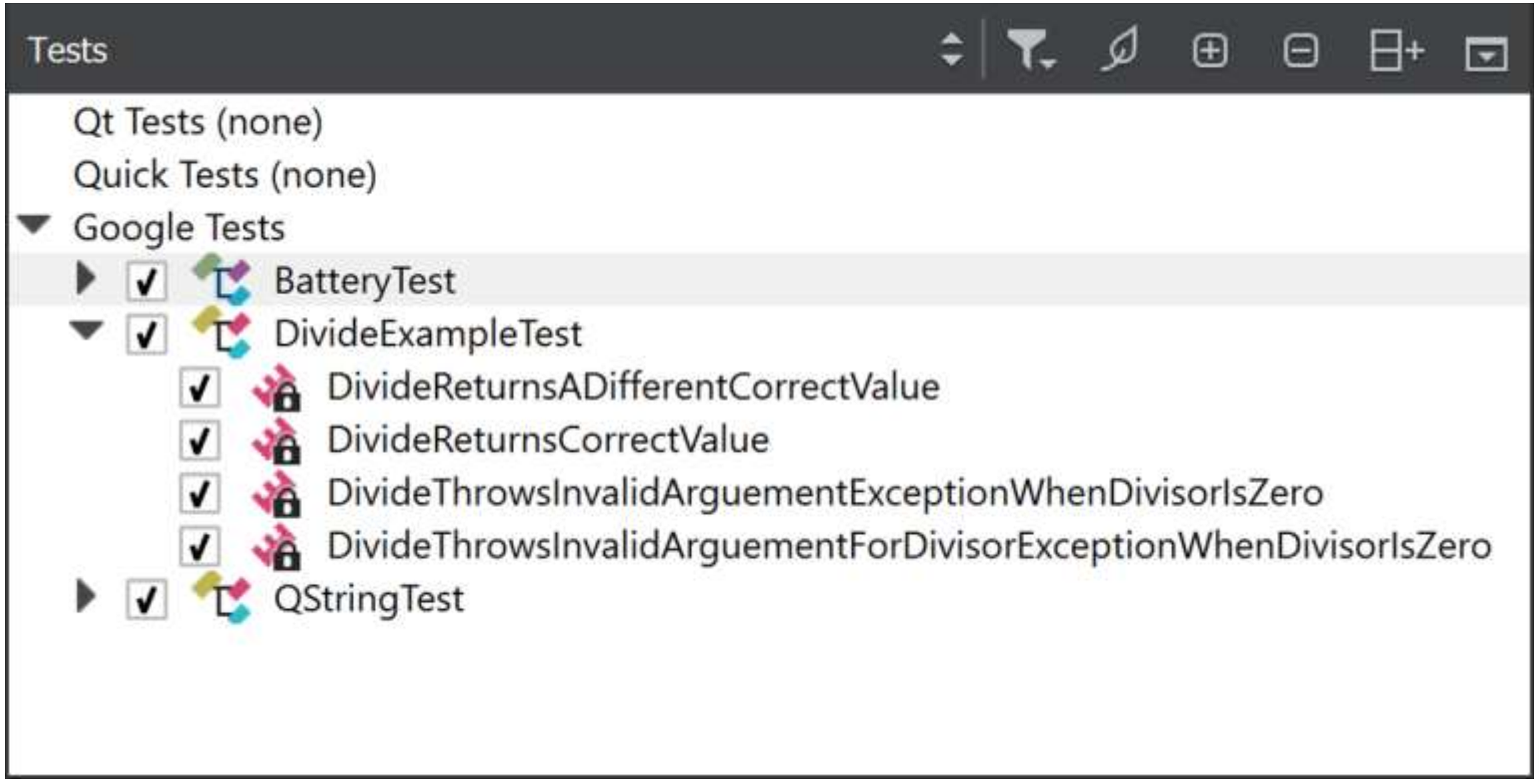
Running Selected Test Fixtures

- Google test supplies a very intelligent main helpers
  - Defaults to running all the tests
  - Command line options can run any subset of tests
    - By fixture name or test name

```
#include <gmock/gmock.h>

int main(int argc, char *argv[])
{
    testing::InitGoogleTest(&argc, argv);
    testing::InitGoogleMock(&argc, argv);
    return RUN_ALL_TESTS();
```

Running Selected Test Fixtures

- Test Project View can select tests

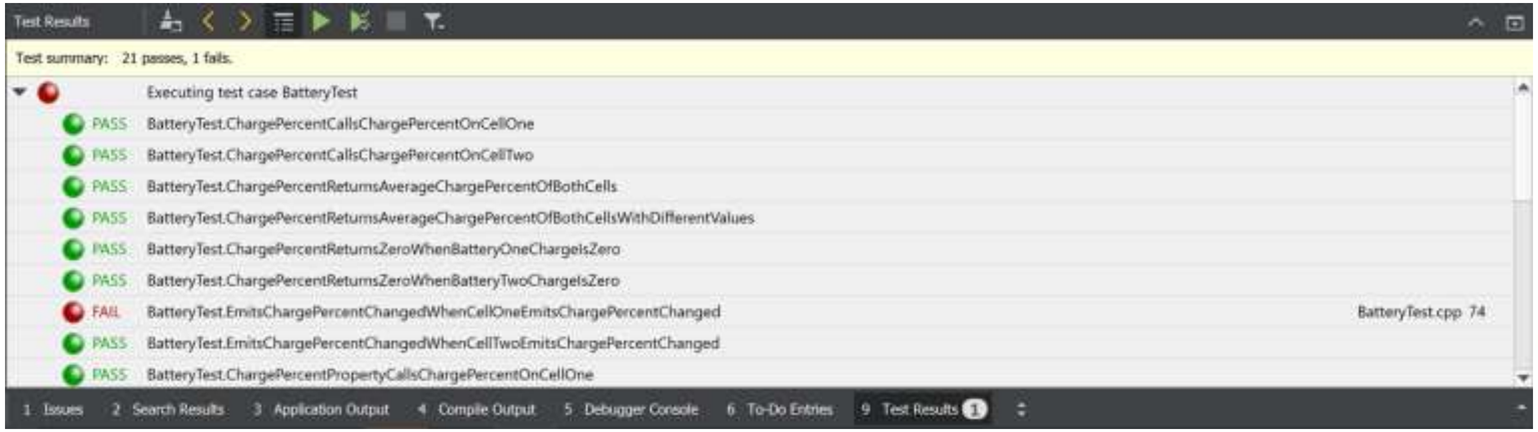


Tests

Qt Tests (none)
Quick Tests (none)
▼ Google Tests
  ▶ ☑ BatteryTest
  ▼ ☑ DivideExampleTest
    ☑ DivideReturnsADifferentCorrectValue
    ☑ DivideReturnsCorrectValue
    ☑ DivideThrowsInvalidArguementExceptionWhenDivisorIsZero
    ☑ DivideThrowsInvalidArguementForDivisorExceptionWhenDivisorIsZero
  ▶ ☑ QStringTest

Running Tests Within Creator

- Tests must be executed from the Test Results Pane
  - Run All
  - Run Selected
  - Filter Results
    - Show failures only
  - Double clicking a failure navigates to the test code



```
Test summary:   21 passes, 1 fails.
        Executing test case BatteryTest
  PASS  BatteryTest.ChargePercentCallsChargePercentOnCellOne
  PASS  BatteryTest.ChargePercentCallsChargePercentOnCellTwo
  PASS  BatteryTest.ChargePercentReturnsAverageChargePercentOfBothCells
  PASS  BatteryTest.ChargePercentReturnsAverageChargePercentOfBothCellsWithDifferentValues
  PASS  BatteryTest.ChargePercentReturnsZeroWhenBatteryOneChargeIsZero
  PASS  BatteryTest.ChargePercentReturnsZeroWhenBatteryTwoChargeIsZero
  FAIL  BatteryTest.EmitsChargePercentChangedWhenCellOneEmitsChargePercentChanged     BatteryTest.cpp 74
  PASS  BatteryTest.EmitsChargePercentChangedWhenCellTwoEmitsChargePercentChanged
  PASS  BatteryTest.ChargePercentPropertyCallsChargePercentOnCellOne
```

Writing Good Tests

- Test one thing at a time
  - It’s tempting to get as much done in one function as you can.
    - This makes one verification depend on another verification
    - Tests will be brittle and changes to code could break lots of verifications
- Do not have your tests depend on each other
  - Order should not matter!
  - Not depending on other tests will help you when you remove functionality.
    - You do not want to have to fix a cascading waterfall of tests!
  - Google Test has a shuffle mode command line parameter
    - Keeps you from depending on previous tests

Test Driven Development (TDD)

- An entirely new way of writing code
  - Mind bending at first. Hard to re-train your brain
- 1 Test is written before ~1 code is written
  - Test must fail before the line of code is written
    - Write the minimum code necessary to pass the test
      - Often the WRONG code to pass the first couple tests
    - Write another test

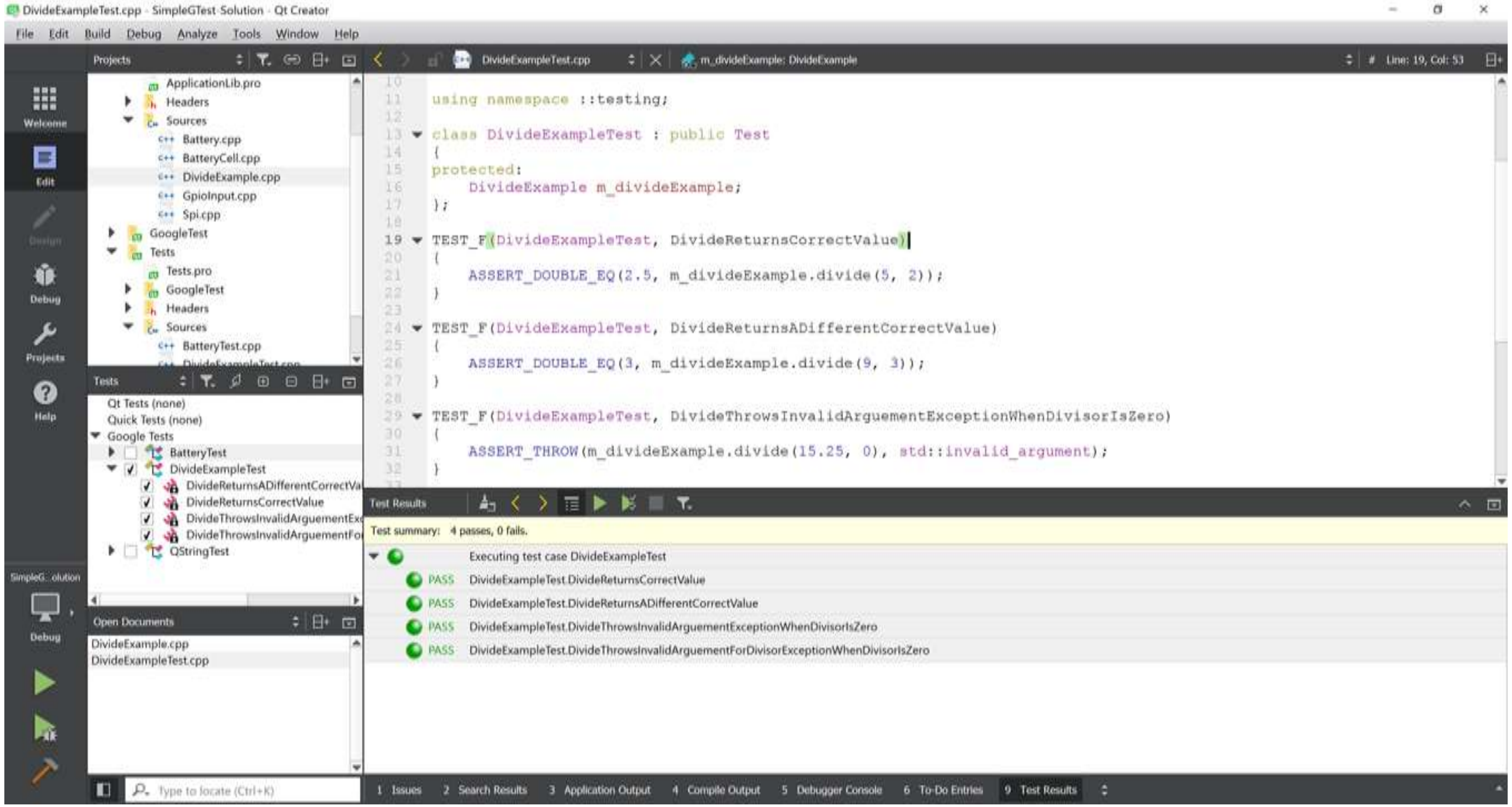
Benefits of TDD

- Every line of code is well tested
  - Tests are written as code is written
  - High code coverage
  - Obscure conditions are tested along with normal paths
- No unused code paths
  - Paths are used as test are written
  - Awkward APIs are found by class developers. Not class users!
- Tests are known to fail when expected code is not correct
  - Because the test failed without that line of code present
  - Tests did not pass by “coincidence”

Drawbacks of TDD

- Larger developed code size. More code more time.
  - Possibly > 10 SLOC of Tests for ~1 SLOC of Code
  - Re-working behavior can be difficult without re-writing many tests
    - Re-factor is easier as code is usually more modular
      - For example: Move tests with code to another class.
- Hard part is still hard.
  - Design is still the hardest part to coding.
    - Increased testing of the implementation will not fix design issues
- Code that isn’t written still isn’t tested
  - i.e. A function can be passed bad data.
    - If there is no test there is no guard for bounds... Boom!
      - Code still needs to be reviewed and corner cases caught.

TDD a Division Function Demo



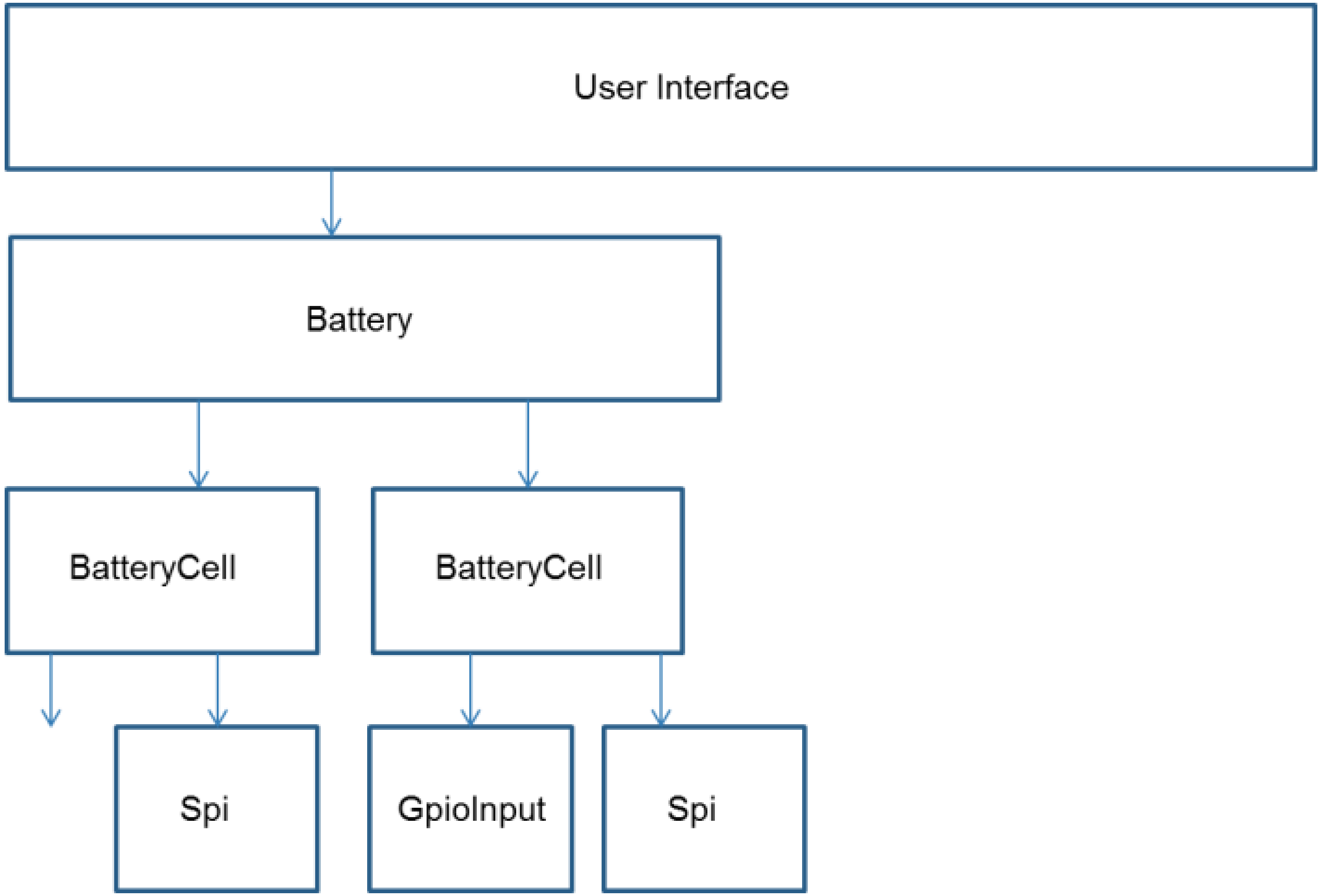
```cpp

using namespace ::testing;

class DivideExampleTest : public Test
{
protected:
    DivideExample m_divideExample;
};

TEST_F(DivideExampleTest, DivideReturnsCorrectValue)
{
    ASSERT_DOUBLE_EQ(2.5, m_divideExample.divide(5, 2));
}

TEST_F(DivideExampleTest, DivideReturnsADifferentCorrectValue)
{
    ASSERT_DOUBLE_EQ(3, m_divideExample.divide(9, 3));
}

TEST_F(DivideExampleTest, DivideThrowsInvalidArguementExceptionWhenDivisorIsZero)
{
    ASSERT_THROW(m_divideExample.divide(15.25, 0), std::invalid_argument);
}
```

```
Test summary:   4 passes, 0 fails.
        Executing test case DivideExampleTest
   PASS DivideExampleTest.DivideReturnsCorrectValue
   PASS DivideExampleTest.DivideReturnsADifferentCorrectValue
   PASS DivideExampleTest.DivideThrowsInvalidArguementExceptionWhenDivisorIsZero
   PASS DivideExampleTest.DivideThrowsInvalidArguementForDivisorExceptionWhenDivisorIsZero
```

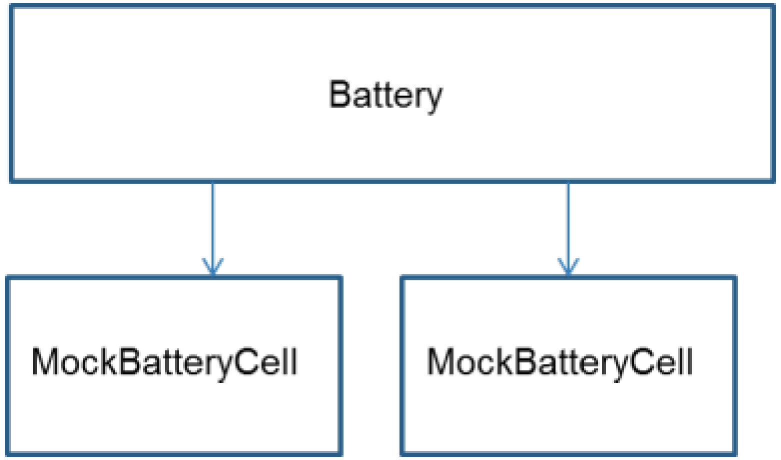
Google Mock is a Dependency Replacement Framework

- Mock allows you to easily replace production class dependencies with testing alternatives
  - Provides the ability to hi-jack return variables and error conditions
  - Provides ability to “expect calls” to be made
    - Check the parameters of those calls
- Mock allows us to test against the interface of the dependency
  - Not the behavior or implementation
    - No reason to actually write to a database or files
      - Simply verify that object calls Logger log() function satisfies test

Production Implementations



Battery Test Implementation



Battery has been isolated from the rest of the system

Dependency Injection

- Constructors to classes take references to dependencies
  - Rather than having self created members
  - Sometimes setters are used rather than constructors
  - Either way makes it easy to swap out dependencies
- Dependency Injection can help isolate units
  - Each class has a base class interface with pure virtuals
    - IBatteryCell
      - Inherited by BatteryCell and MockBatteryCell
  - Interface has to inherit QObject to use signals and Q_PROPERTY
    - Properties can be declared with pure virtual READ and WRITE

IBatteryCell

```
#include "ApplicationLibDecl.h"

#include <QObject>

class APPLICATIONLIB_EXPORT IBatteryCell : public QObject
{
    Q_OBJECT
    Q_PROPERTY(double chargePercent READ chargePercent
                                    WRITE chargePercentChanged)
public:
    virtual double chargePercent() const = 0;

signals:
    void chargePercentChanged();
};
```

MockBatteryCell

```
#include "IBatteryCell.h"

#include <gmock/gmock.h>

class MockBatteryCell : public IBatteryCell
{
    Q_OBJECT
public:
    MOCK_CONST_METHOD0(chargePercent, double());
};
```

Using Mock in a Fixture

```
class BatteryTest : public Test
{
public:
    BatteryTest() :
        m_battery(m_batteryCellOne, m_batteryCellTwo)
    {
    }

protected:
    //Dependencies
    NiceMock<MockBatteryCell> m_batteryCellOne;
    NiceMock<MockBatteryCell> m_batteryCellTwo;

    //Class Under Test
    Battery m_battery;
};
```

Mock Expectations

- EXPECT_CALL
  - Verifies that a function was called
    - How many times? With what parameters?

```
TEST_F(SomeTest, SetValueCalledWithCorrectParameter)
{
    EXPECT_CALL(m_mockDep, setValue(1)).Times(AtLeast(1));
    m_underTest.setAllValues();
}
```

Mock Return Value

- ON_CALL or EXPECT_CALL can make the mocked function return arbitrary values
  - Even return different values based on parameters
  - Or return different values based on how many times it was called

```
TEST_F(SomeTest, CalculateReturnsDepFooPlusBar)
{
    ON_CALL(m_mockDep, getValue(“Foo”)).WillByDefault(Return(5.0));
    ON_CALL(m_mockDep, getValue(“Bar”)).WillByDefault(Return(55.0));
    ASSERT_DOUBLE_EQ(60.0, m_underTest.calculate());
}
```

Turning Mock Implicit Failure

- Regular MockType instance
  - Warns on uninteresting calls
    - Functions called without explicit EXPECT_CALL or ON_CALL
    - Test passes with warnings printed to console
- StrictMock<MockType>
  - Fails on any uninteresting call
- NiceMock<MockType>
  - Fails only when explicit expectations are not met
  - Suppresses uninteresting warnings

Mock Has Many Features

- Far too many to list or describe in a 1 hour webinar
  - There are entire books dedicated to Google Mock
- Tune Expectations
  - Return(ByRef(val)), WillOnce, WillByDefault, Times(), GreaterThan()
  - Custom Matchers
    - Verify parameters by arbitrary means
      - Compare based on members
        - Id() function on a pointer type
- List goes on and on

Testing a Signal

- QSignalSpy from QTest can test signals without a slot
  - QTest also has functions for stimulating user events

```
TEST_F(SomeTest, MouseClickEmitsClickedSingal)
{
    QSignalSpy spy(&m_button, SIGNAL(clicked()));
    QTest::mouseClick(m_button);
    ASSERT_EQ(1, spy.count());
}
```

Stimulating a Signal

- signals is a #define public
  - Emit dependent object signals directly!
- emit is a #define to nothing

```
TEST_F(SomeTest, DepUpdateSignalChangesValue)
{
    m_underTest.setValue(5);
    emit m_mockDep.update(10); // emit is actually optional
    ASSERT_EQ(10, m_underTest.value());
}
```

Q_PROPERTY Macro Testing

- Two options to test Q_PROPERTY via TDD
  - 1) Use Private/Protected READ and WRITE methods
    - Test the property like a public function using
      - obj.property(“name”).to[Type]()
      - obj.setProperty(“name”, variantValue)
    - Only useful if class is only used via properties. Say for QML.
  - 2) Test public getter and setter as usual
    - Then test the Q_PROPERTY using dummy READ / WRITE methods
      - You end up with the exact same tests as get/set methods
    - Finally refactor the Q_PROPERTY to use regular get/set
      - All tests should still pass
        - However, you end up with 2x tests. Bulletproof tests.

Q_PROPERTY NOTIFY / CONSTANT

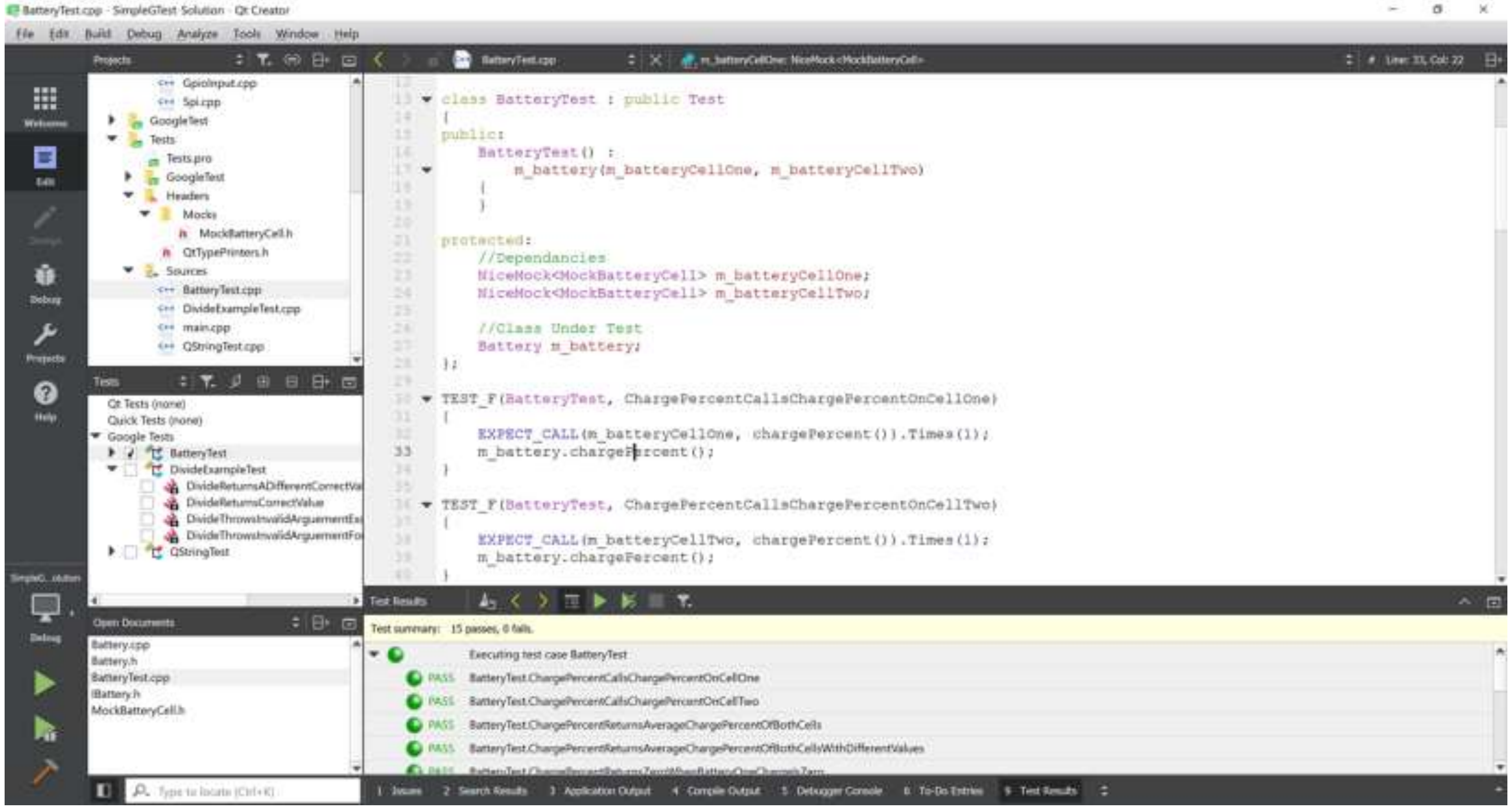
- Check the name of the property's NOTIFY signal
  - Use QMetaObject / QMetaProperty / QMetaMethod

```
int propertyIndex = m_battery.metaObject()->indexOfProperty("chargePercent");
QMetaProperty property = m_battery.metaObject()->property(propertyIndex);
ASSERT_STREQ("chargePercentChanged", property.notifySignal().name().data());
```

- Check for property’s CONSTANT flag
  - Use QMetaObject / QMetaProperty

```
int propertyIndex = m_battery.metaObject()->indexOfProperty("chargePercent");
QMetaProperty property = m_battery.metaObject()->property(propertyIndex);

ASSERT_TRUE(property.isConstant());
```

Battery Test Example



```cpp
class BatteryTest : public Test
{
public:
    BatteryTest() :
        m_battery(m_batteryCellOne, m_batteryCellTwo)
    {
    }

protected:
    //Dependancies
    NiceMock<MockBatteryCell> m_batteryCellOne;
    NiceMock<MockBatteryCell> m_batteryCellTwo;

    //Class Under Test
    Battery m_battery;
};

TEST_F(BatteryTest, ChargePercentCallsChargePercentOnCellOne)
{
    EXPECT_CALL(m_batteryCellOne, chargePercent()).Times(1);
    m_battery.chargePercent();
}

TEST_F(BatteryTest, ChargePercentCallsChargePercentOnCellTwo)
{
    EXPECT_CALL(m_batteryCellTwo, chargePercent()).Times(1);
    m_battery.chargePercent();
}
```

```
Test summary:  15 passes, 0 fails.
        Executing test case BatteryTest
  PASS  BatteryTest.ChargePercentCallsChargePercentOnCellOne
  PASS  BatteryTest.ChargePercentCallsChargePercentOnCellTwo
  PASS  BatteryTest.ChargePercentReturnsAverageChargePercentOfBothCells
  PASS  BatteryTest.ChargePercentReturnsAverageChargePercentOfBothCellsWithDifferentValues
  PASS  BatteryTest.ChargePercentReturnsZeroWhenBatteryOneChargeIsZero
```

Bibliography

Google Test

- Primer
  - https://github.com/google/googletest/blob/master/googletest/docs/Primer.md
- Cook Book
  - https://github.com/google/googletest/blob/master/googlemock/docs/CookBook.md

Bibliography

Google Mock

- Primer
  - https://github.com/google/googletest/blob/master/googlemock/docs/ForDummies.md
- Cook Book
  - https://github.com/google/googletest/blob/master/googlemock/docs/CookBook.md
- Cheat Sheet
  - https://github.com/google/googletest/blob/master/googlemock/docs/CheatSheet.md

Bibliography

TDD

- http://www.agiledata.org/essays/tdd.html
- http://www.jamesshore.com/Agile-Book/test_driven_development.html
- http://www.amazon.com/Modern-Programming-Test-Driven-Development-Better/dp/1937785483/ref=sr_1_1?ie=UTF8&qid=1455140098&sr=8-1&keywords=TDD+C%2B%2B
- http://www.amazon.com/Test-Driven-Development-Kent-Beck/dp/0321146530/ref=sr_1_fkmr0_1?ie=UTF8&qid=1455140098&sr=8-1-fkmr0&keywords=TDD+C%2B%2B

Bibliography

- SOLID Object Oriented Design Principles
  - https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
  - http://www.codemag.com/article/1001061
  - https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

Thank You!