Quickly and Effectively Testing Qt Desktop Applications

Clare Macrae (She/her)
clare@claremacrae.co.uk
3 December 2020
Meeting C++ Online

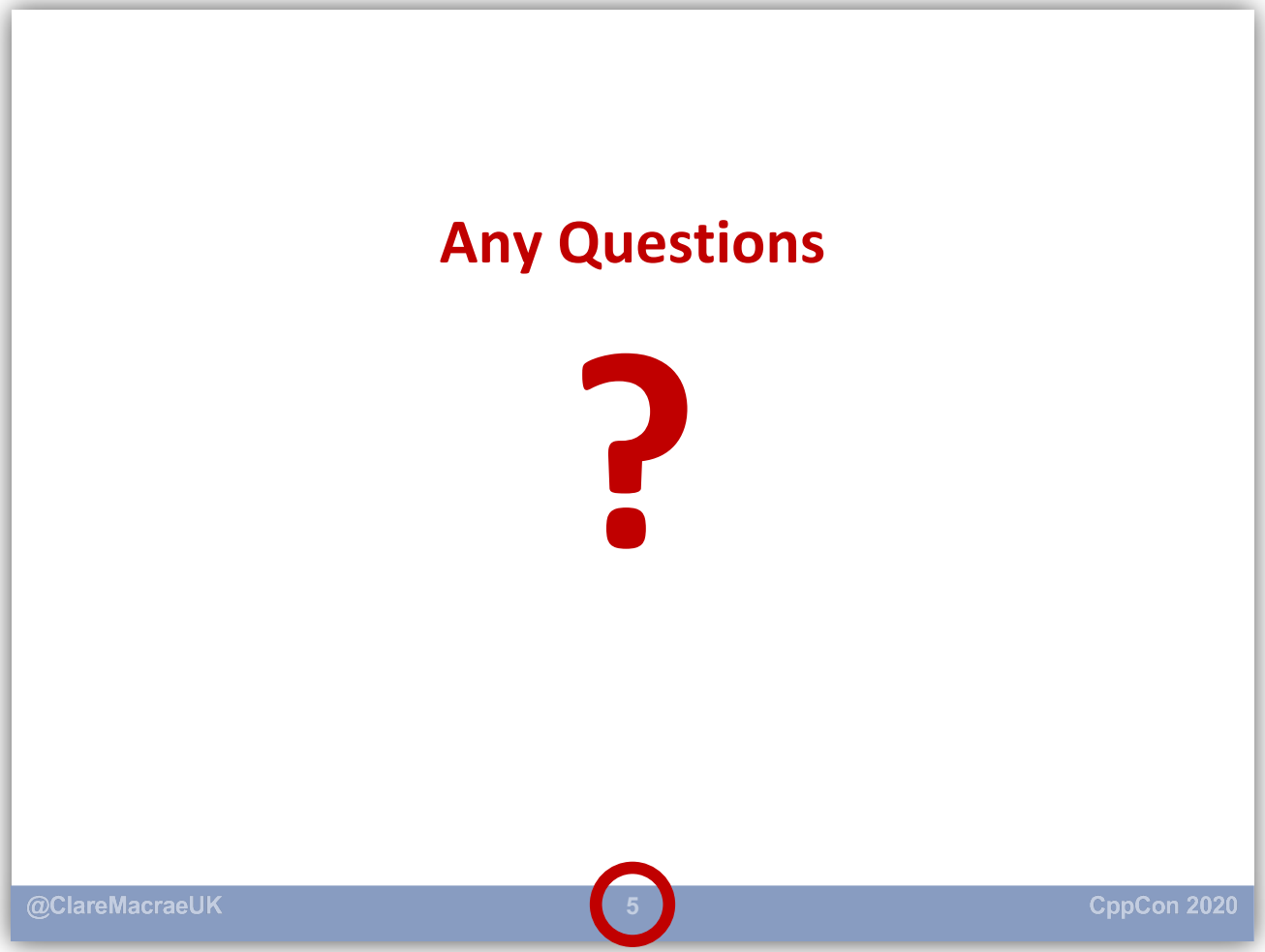
Contents

About Me

- Scientific **C++** and **Qt** developer since 1999
- **My mission: Sustainable and efficient testing and refactoring of legacy code**
  - Co-author of **"Approval Tests for C++"**
- **Consulting & training** via "Clare Macrae Consulting Ltd"
  - claremacrae.co.uk

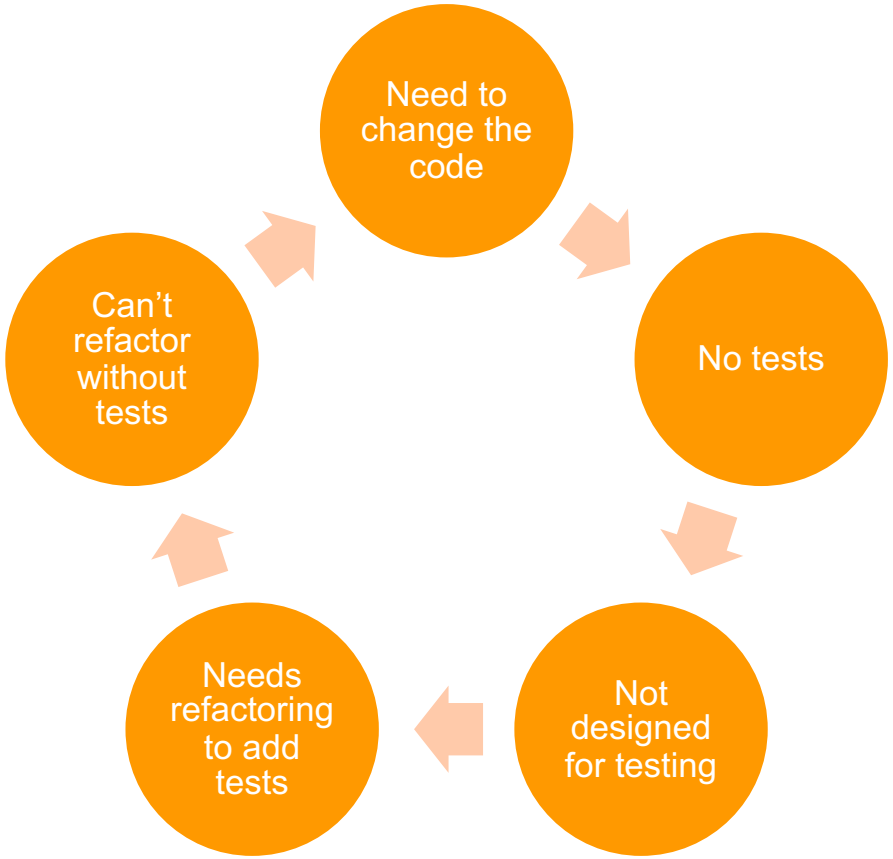
- **All links from this talk via:**
  - bit.ly/TestingQt
  - github.com/claremacrae/talks

About…



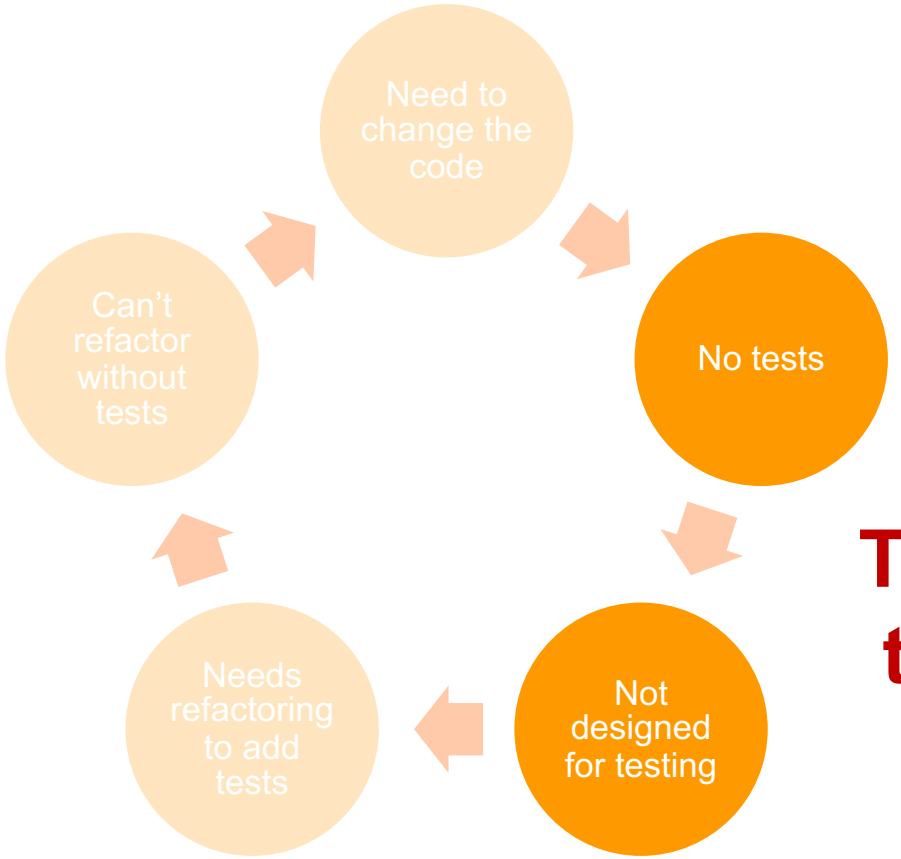
Typical Scenario

- I've inherited some Qt GUI code
- It's valuable
- I need to add feature
- Or fix bug
- How can I ever break out of this loop?



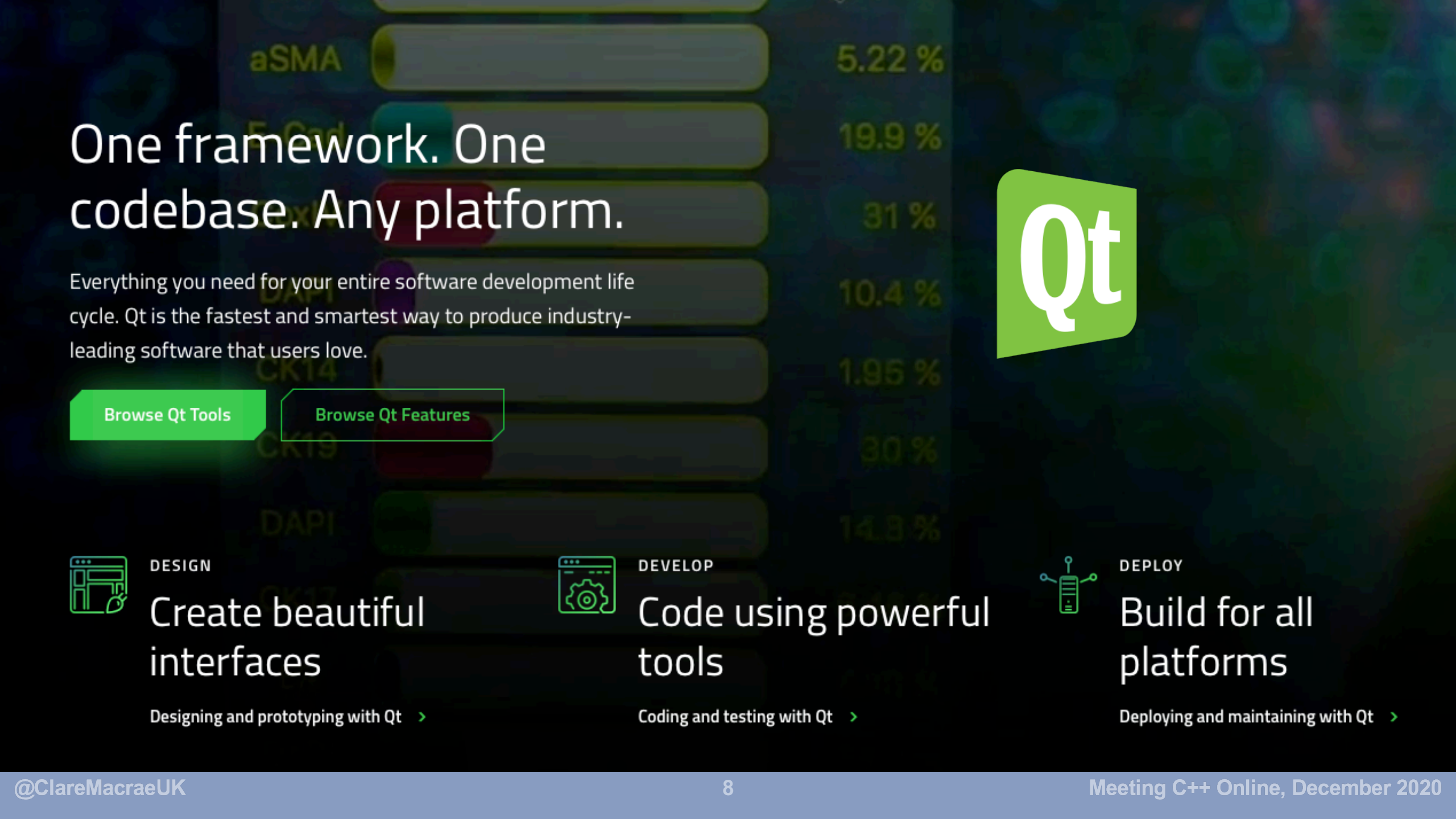
Typical Scenario

- I've inherited some qt GUI code
- It's valuable
- I need to add feature
- Or fix bug
- How can I ever break out of this loop?



Topics of this talk

Contents


aSMA
5.22 %
19.9 %
31 %
10.4 %
1.95 %
30 %
14.8 %
One framework. One codebase. Any platform.
Everything you need for your entire software development life cycle. Qt is the fastest and smartest way to produce industry-leading software that users love.
Browse Qt Tools
Browse Qt Features
Qt
DESIGN
Create beautiful interfaces
Designing and prototyping with Qt ›
DEVELOP
Code using powerful tools
Coding and testing with Qt ›
DEPLOY
Build for all platforms
Deploying and maintaining with Qt ›

Qt's GUI powers make Automated Testing Harder

Give you **Confidence**
to **Start testing**
your **Qt application**
Effectively

Contents

Squish

- Automated GUI Testing – of your whole application
  - Semi-opaque-box testing
- froglogic.com/squish – Commercial tool
- Basics
  - Record actions in your application
  - Refactor in to reusable tests, e.g. in Python
- Opinion
  - Not a replacement for unit tests
  - Requires commitment to training and maintenance
  - If used well, can replace lots of manual testing


Qt DESKTOP|DAYS
KDAB
Frerich Raabe
Jesper Pedersen
Qt DESKTOP|DAYS
September 7th - 11th
Behaviour-Driven GUI Test Automation of Qt Applications
froglogic
Frerich Raabe, raabe@froglogic.com
September 9th, 2020
0:09 / 51:58

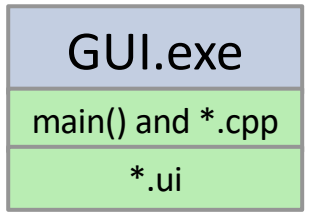
Context: Type of testing

- **Glass-box or transparent-box testing**
- Have access to the code
- And can potentially even make changes to it

Q: How do I add tests to existing app?

A: Ah – Good point – You don't

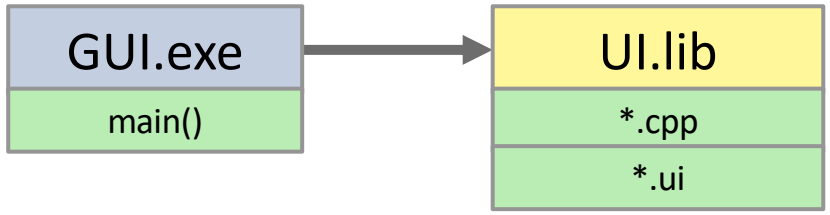
Introduce static lib for tests



Color Key

Executable

Introduce static lib for tests

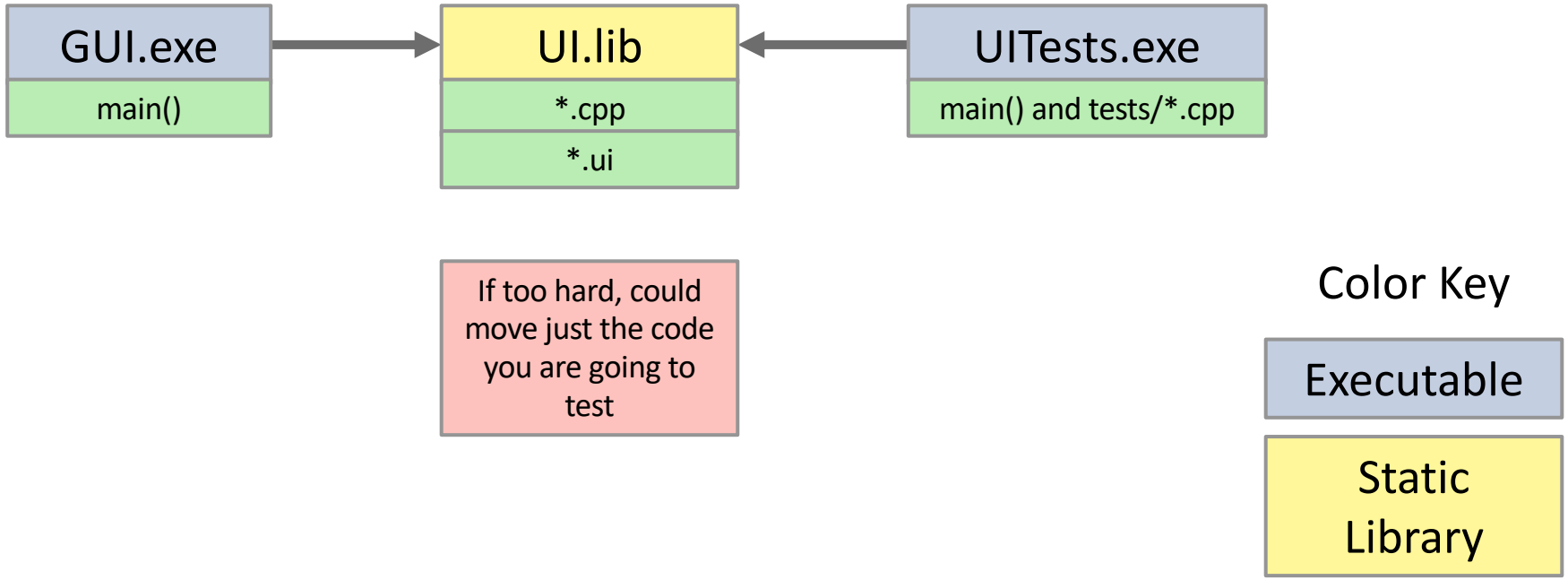


Color Key

Executable

Static Library

Introduce static lib for tests



Why Static Library?

Impact	Dynamic Library	Static Library

Source code	Windows builds need code changes
	`__declspec( dllimport )`
	`__declspec( dllexport )`
		No Change

Release process	Add the dynamic libraries to your distributions	No Change

Static is Smaller Change!

Doing the Separation

- For CMake, see Arne Mertz's excellent tutorial:
- arne-mertz.de/2018/05/hello-cmake/
- Remember to make the library static!

Creating the Test Executable

C++ Test Runners

- Lots to choose from:

- Catch2, with a little bit of Qt Test
- Tricks needed to use Qt with Catch2
  - Useful if using different framework

QtTest vs Catch2: Testing Strings

```cpp
// Based on https://doc.qt.io/qt-5/qttestlib-tutorial1-example.html

#include <QtTest>

class TestQString : public QObject
{
    Q_OBJECT
private slots:
    void toUpper();
};

void TestQString::toUpper()
{
    QString str = "Hello";

    QVERIFY(str.toUpper() == "HELLO");
    QCOMPARE(str.toUpper(), QString("HELLO"));
}

QTEST_MAIN(TestQString)
#include "TestQString.moc"
```

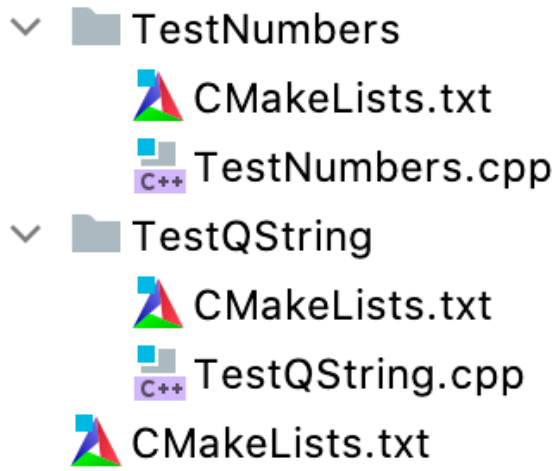
```cpp
#include <catch2/catch.hpp>

#include <QString>

TEST_CASE("String::toUpper")
{
    QString str = "Hello";
    CHECK(str.toUpper() == "HELLO");
}
```

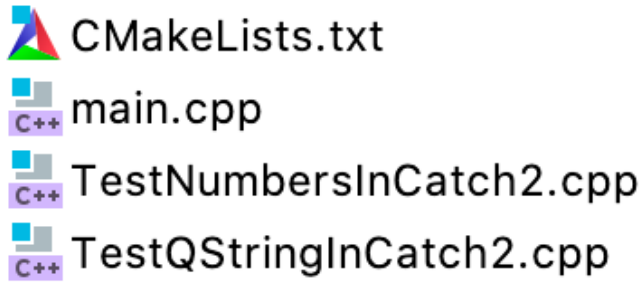
QtTest vs Catch2: Testing Numbers

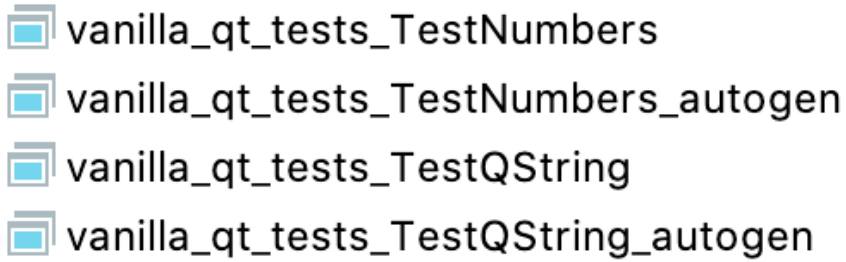
```cpp
#include <QtTest>

class TestNumbers : public QObject
{
    Q_OBJECT
private slots:
    void lessThan();
};

void TestNumbers::lessThan()
{
    QVERIFY(12 < 42);
    /* Failure would look like:
FAIL!  : TestNumbers::lessThan() '52 < 42' returned FALSE. ()
Loc: [.../ApprovalTests.cpp.Qt/examples/vanilla_qt_tests/TestNumbers/TestNumbers.cpp(12)]
    */
}

QTEST_MAIN(TestNumbers)
#include "TestNumbers.moc"
```

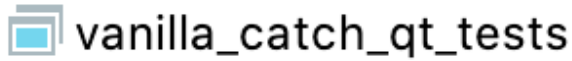
```cpp
#include <catch2/catch.hpp>

#include <QString>

TEST_CASE("LessThan")
{
    CHECK(12 < 42);
    /* Failure would look like:
.../ApprovalTests.cpp.Qt/examples/vanilla_catch_qt_tests/TestNumbersInCatch2.cpp:7: FAILED:
CHECK( 52 < 42 )
    */
}
```

QtTest vs Catch2: Project Structure







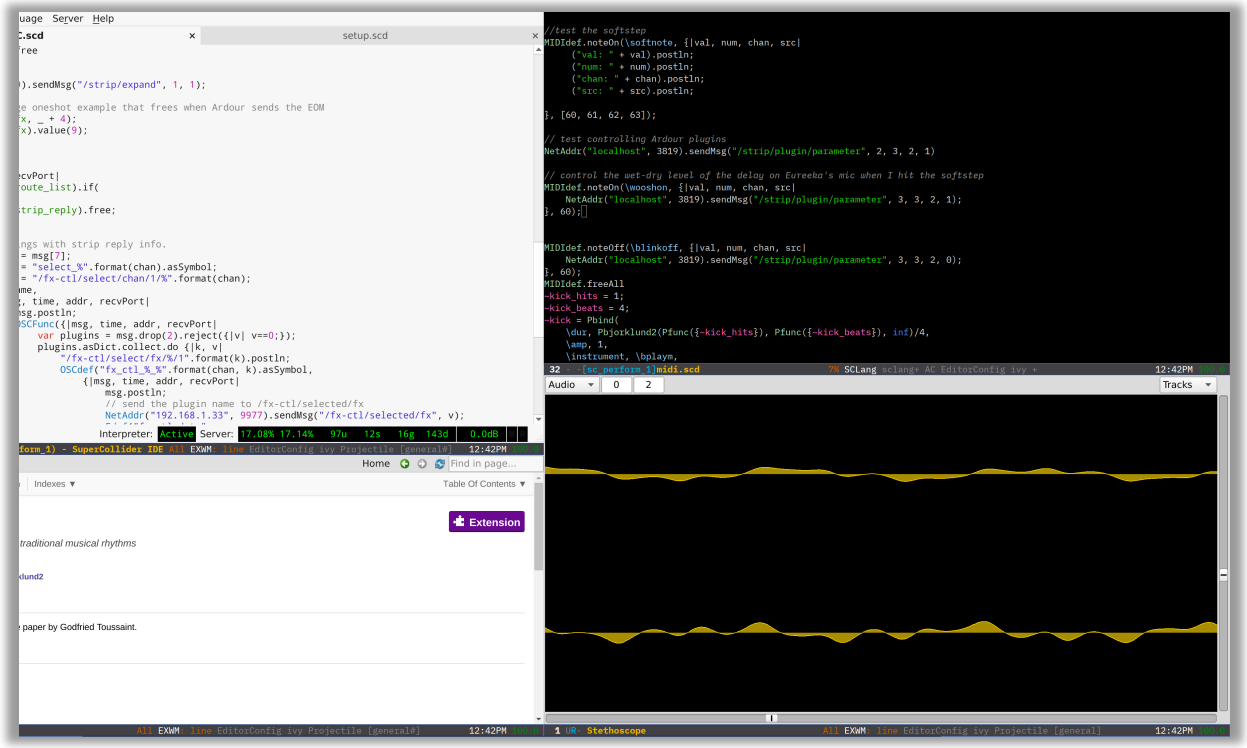


Catch2: Setting up Catch2 v2 test main() with Qt

```
// Demonstrate how to create a Catch main() for testing Qt GUI code

#define CATCH_CONFIG_RUNNER
#include <Catch.hpp>

#include <QApplication>

int main(int argc, char* argv[])
{
    QApplication app(argc, argv);                    // -platform offscreen
    int result = Catch::Session().run(argc, argv); // --break
    return result;
}
```

Context: SuperCollider

- supercollider.github.io
- platform for audio synthesis and algorithmic composition

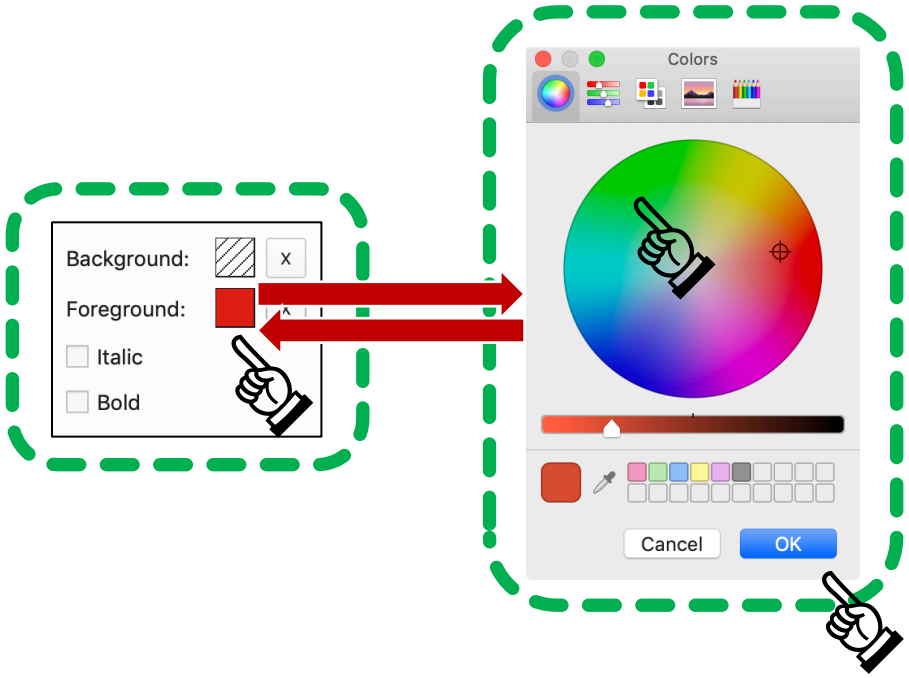


Picture credit: Chad Cassady, @beatboxchad

Context: ColorWidget color-picker

This set of widgets is "our" code:

- we want to test it



This set of widgets is "Qt" code:

- not our job to test it
- hard to test anyway
- **only test our stuff!**

ColorWidget setup – QColor object

```
#include "catch.hpp"
#include "color_widget.hpp"
#include "catch_qt_string_makers.hpp"

using namespace ScIDE;

TEST_CASE("ColorWidget initial state") {
    ColorWidget widget;
    QColor expected_color(0, 0, 0, 255);
    CHECK(widget.color() == expected_color);
}
```

```
FAILED:
  CHECK( widget.color() == expected_color )
with expansion:
  {?} == {?}
```

ColorWidget setup – approval testing

```
#include "catch.hpp"
#include "color_widget.hpp"
#include "catch_qt_string_makers.hpp"

using namespace ScIDE;

TEST_CASE("ColorWidget initial state - approval testing") {
    ColorWidget widget;
    auto color = Catch::StringMaker<QColor>::convert(widget.color());
    CHECK(color == "(0, 0, 0), alpha = 1");
    // was initially:
    // CHECK(color == "");
}
```

ColorWidget changing state

```
TEST_CASE("ColorWidget changing color updates correctly") {
    // Arrange
    ColorWidget widget;
    QColor red("red");

    // Act
    widget.setColor(red);

    // Assert
    CHECK(widget.color() == red);
}
```

“Quickly Testing?”

- From **0 to 1** tests always **hardest** – that’s normal
- From **1 to 2** … **2 to 3** always **much** easier
- **Step 1**: Make first test **easy as possible** – it’s going to be hard!
- **Step 2**: Write at least 3 tests – before deciding if it’s a good idea!

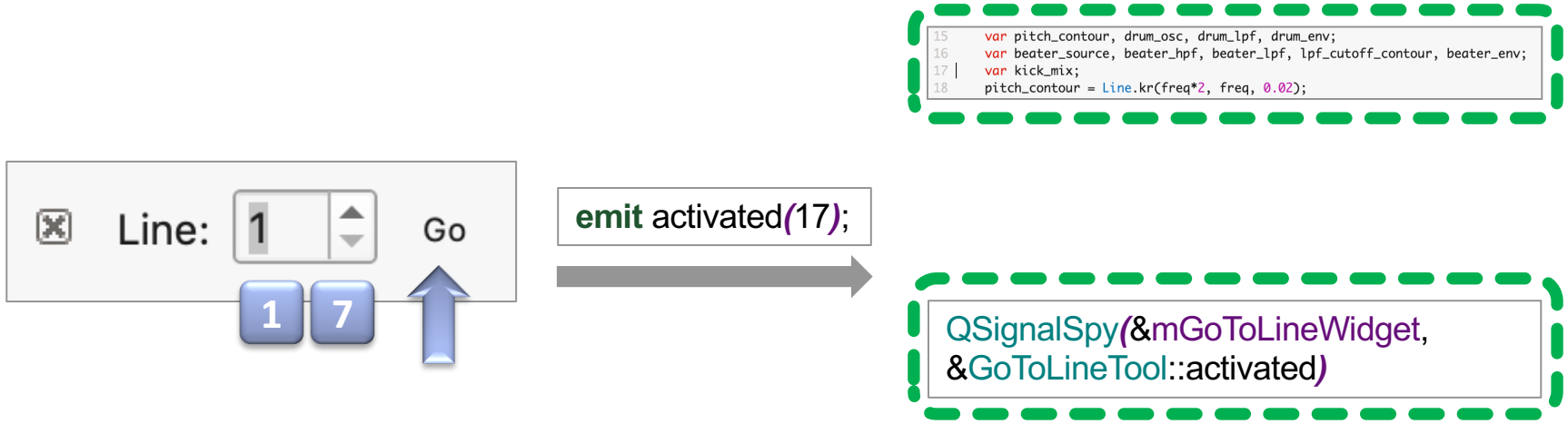
Context: “Go To Line” Panel

babbling brook.scd

```
1
2
3  /*
4  A babbling brook example, by James McCartney 2007. See
5  http://www.create.ucsb.edu/pipermail/sc-users/2007-April/033231.html
6  */
7
8  (
9  {
10     ({RHPF.ar(OnePole.ar(BrownNoise.ar, 0.99), LPF.ar(BrownNoise.ar, 14)
11         * 400 + 500, 0.03, 0.003)}!2)
12         + ({RHPF.ar(OnePole.ar(BrownNoise.ar, 0.99), LPF.ar(BrownNoise.ar, 20)
13         * 800 + 1000, 0.03, 0.005)}!2)
14         * 4
15 }.play
16 )
17
```

Line: 1 Go

Test actions – Signals and Slots

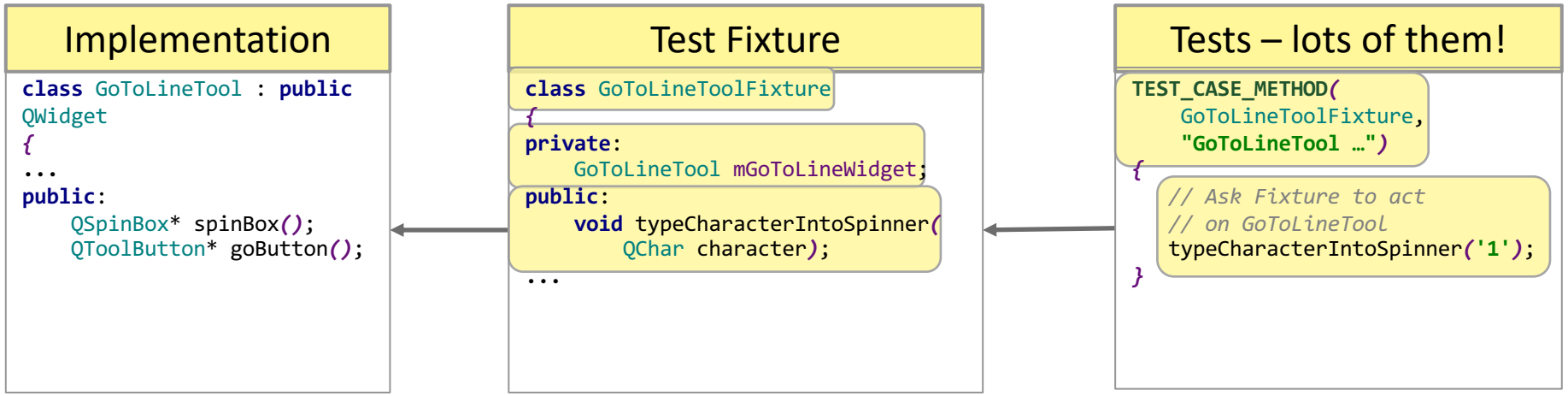


Qt **Signal** = “special method for announcing changes”
Qt **Slot** = “special method for responding to changes”

Goal: Hide implementation details

- Want to avoid tests knowing insides of widgets being tested
  - As much as possible
- Can we have really readable tests?
- Can we guard against changes to implementation?

High level view

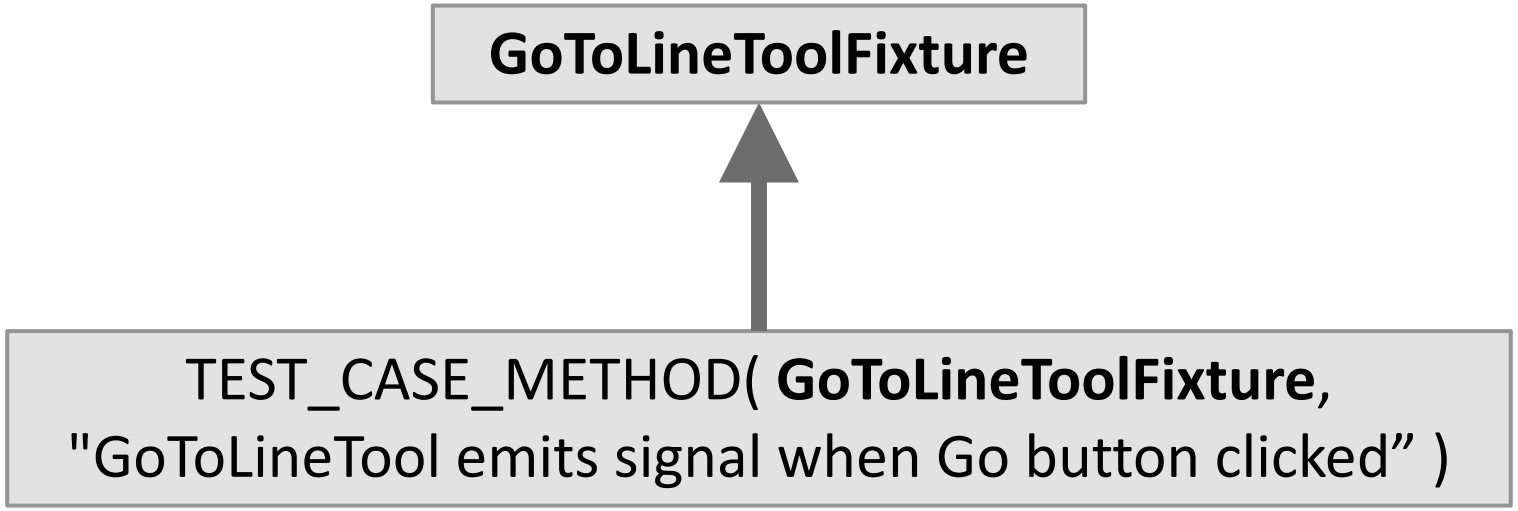


Introduce a Fixture

```
class GoToLineToolFixture
{
private:
    GoToLineTool mGoToLineWidget;
    QSpinBox* mSpinner = nullptr;
    QToolButton* mGoButton = nullptr;
    std::unique_ptr<QSignalSpy> mActivatedSpy;

protected:
    GoToLineToolFixture()
    {
        mGoToLineWidget.raise();
        mGoToLineWidget.show();
        // ... and so on ...
```

Fixtures in Catch2

- Fixtures share code between tests
- Also improve test design

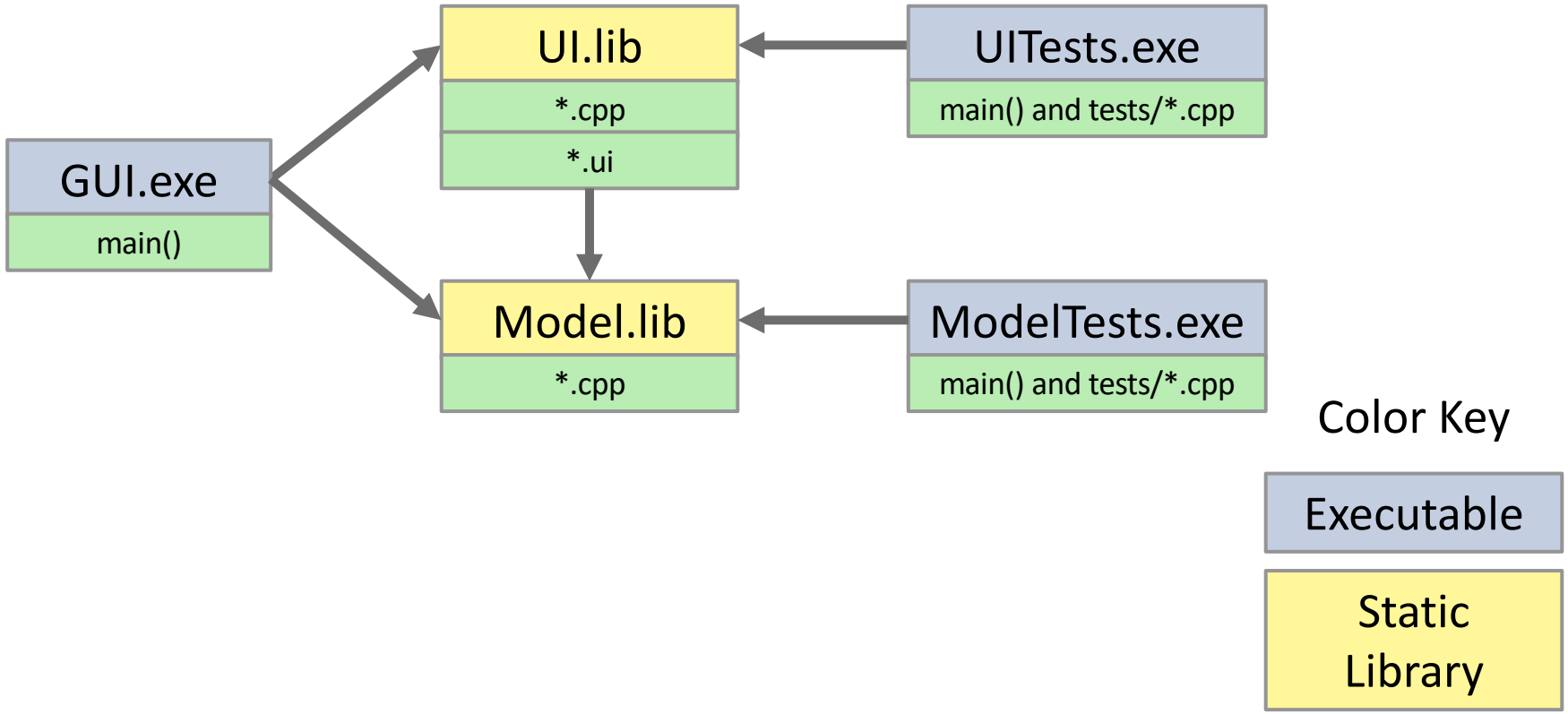


GoToLineTool – testing events, expressively

```
TEST_CASE_METHOD(GoToLineToolFixture,
                 "GoToLineTool emits signal when Go button clicked")
{
    // Arbitrary upper limit in number of lines.
    // When used in the application, this would be obtained from the open
    // document
    setMaximumLineCount(27);

    // Type a number, one digit at a time
    typeCharacterIntoSpinner('1');
    typeCharacterIntoSpinner('7');
    clickGoButton();

    checkActivatedSignalCount(1);
    checkActivatedSignalValue(17);
}
```

More Thorough Fixture Explanation

Bonus Points: Separate logic from UI code



Contents

Pitfall: Qt4-style Signal/Slot connect()

- Run-time error checking:

```
connect(
    ui->quitButton, SIGNAL(clicked()),
    this, SLOT(quit()));
```

```
QObject::connect: No such slot SampleMainWindow::quit()
QObject::connect:  (sender name:   'quitButton')
QObject::connect:  (receiver name: 'SampleMainWindow')
```

Safer: Qt5-style Signal/Slot connect()

- Use Qt5 pointer-to-member-function connections

```
connect(
    ui->quitButton, SIGNAL(clicked()),
    this, SLOT(quit()));

// Solution: Qt5-style connections get checked at compile-time:
connect(
    ui->quitButton, &QPushButton::clicked,
    this, &SampleMainWindow::quit
);
```

Pitfall: Event processing in tests

- Normally, Qt's event loop processes events automatically
  - **QCoreApplication::processEvents**()
- Can't rely on that in test programs
- Know your options:
  - bool **QTest::qWaitForWindowActive**(*window/widget*, *timeout*)
  - bool **QTest::qWaitForWindowExposed**(*window/widget*, *timeout*)
  - bool **QSignalSpy::wait**(*timeout*)
- If your test runner is Qt Test:
  - **QTRY_COMPARE**, **QTRY_COMPARE_WITH_TIMEOUT**
  - **QTRY_VERIFY**, **QTRY_VERIFY_WITH_TIMEOUT**
  - These macros include a return statement on failure
  - Don't use these if supplying your own main(), e.g. with Catch2

Reference: SafeQTestMacros.h

```
9   namespace qft
10  {
11      void sender(...);
12  }
13
14  // A custom QCOMPARE implementation, for use with tests that are driven
15  // by Catch2, or any other non-QtTest main(), that want access to QtTest
16  // in order to use QSignalSpy - but must not use QCOMPARE and similar,
17  // as any failures in these macros are silently ignored when called
18  // from outside a QObject slot.
19  //
20  // Credit: Fabian Kosmale at Qt, November 2019
21
22  #undef QCOMPARE
23  #define QCOMPARE(actual, expected)                                          \
24      do                                                                      \
25      {                                                                       \
26          using namespace qft;                                                \
27          static_assert(std::is_same<decltype(sender()), QObject*>::value,    \
28                        "Cannot user QCOMPARE outside of a QObject slot");    \
29          if (!QTest::qCompare(actual, expected, #actual, #expected, __FILE__, __LINE__))  \
30              return;                                                         \
31      } while (false)
```

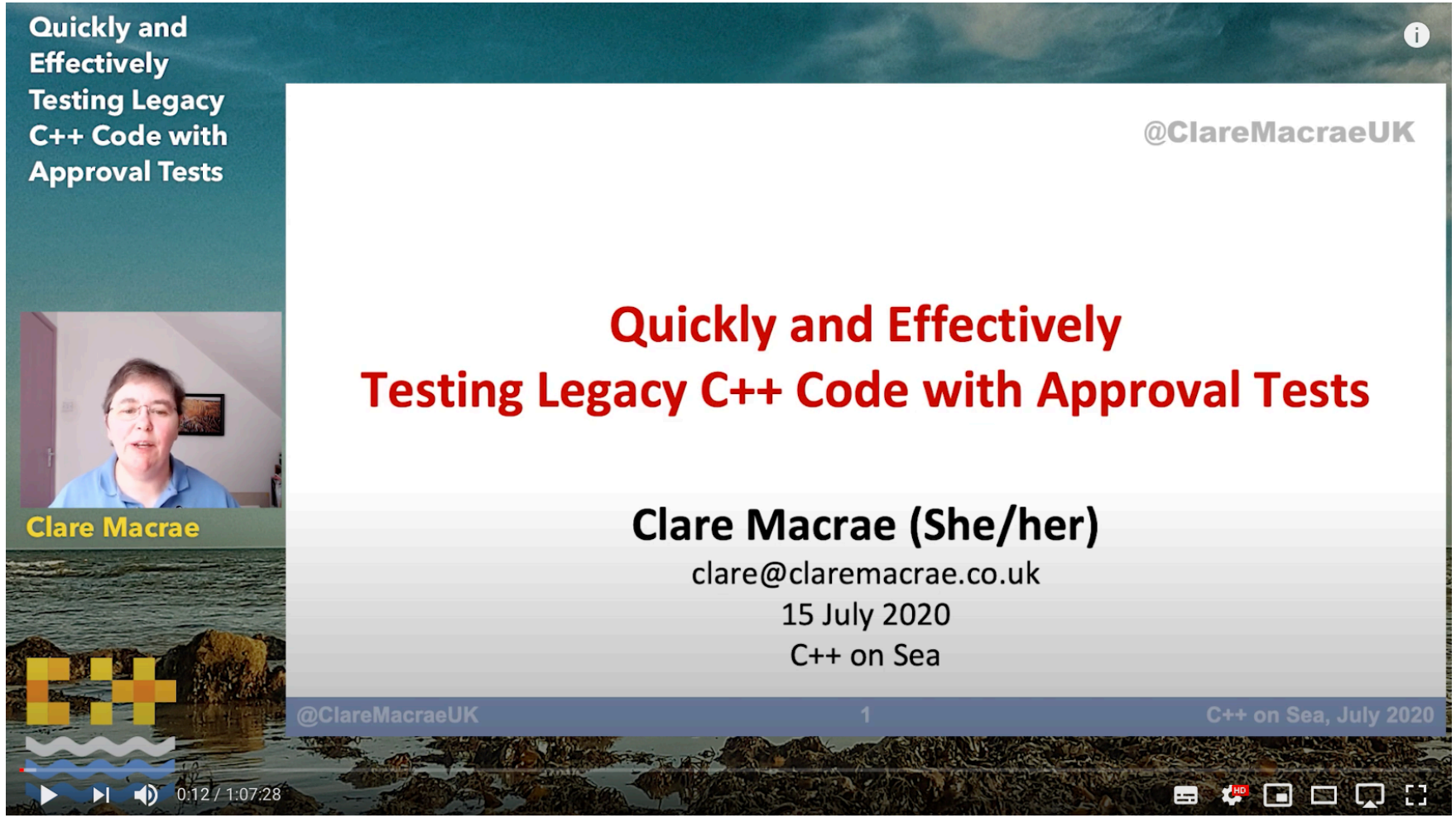
Running Qt tests in CI system

- Windows:
  - Will probably just work
- Linux:
  - Typically use virtual display, e.g. xvfb
  - Could investigate `-platform offscreen`
- Mac:
  - Found needed to keep a user logged in
- Make tests that need a GUI behave gracefully if no display found:
  - `IF_NO_GUI_PRINT_AND_RETURN()`

Contents

Approval Tests?

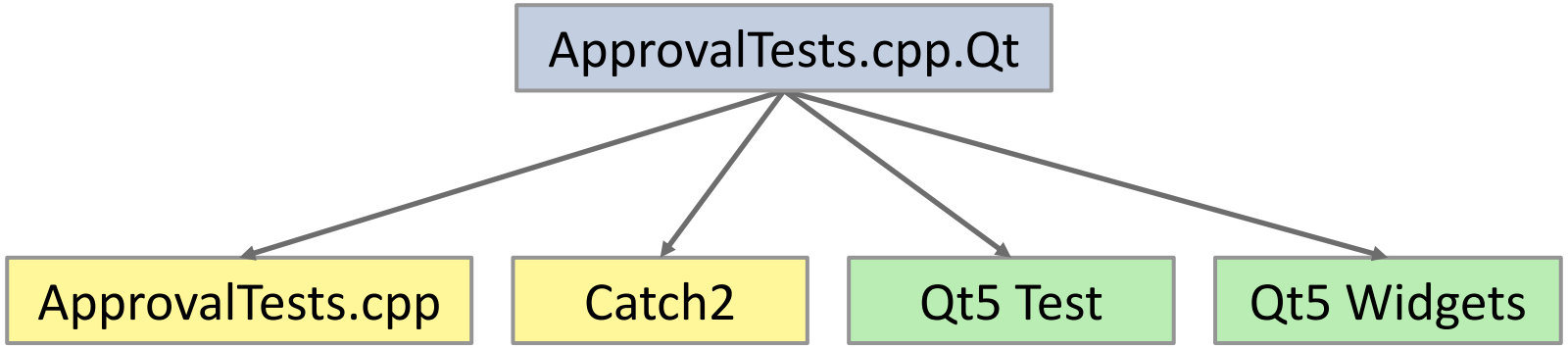
- Verification/**snapshot**/golden copy testing
- "Expected" stored **separate** from source code
- Extremely **powerful**, convenient, quick tactic for testing **legacy code**



Introducing ApprovalTests.cpp.Qt

Introducing ApprovalTests.cpp.Qt

- github.com/approvals/ApprovalTests.cpp.Qt



Introducing ApprovalTests.cpp.Qt

- **Goals**

- Rapidly start testing Qt code
  - Useful even if you don't use Approval Tests!
- Approval Tests support for Qt types
  - Easy saving of state in Golden Output files

- github.com/approvals/ApprovalTests.cpp.Qt
- github.com/approvals/ApprovalTests.cpp.Qt.StarterProject
- v.0.0.2

Setting up testsuite main()

```
// main.cpp:
#define APPROVALS_CATCH_QT
#include "ApprovalTestsQt.hpp"
```

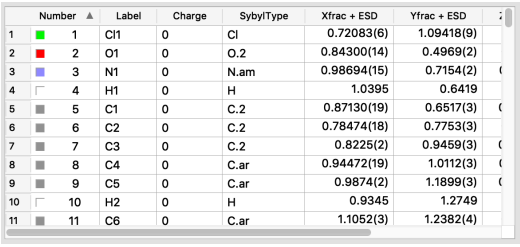
```
// Original Code
#define CATCH_CONFIG_RUNNER
#include <Catch.hpp>

#include <QApplication>

int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    int result = Catch::Session().run(argc, argv);
    return result;
}
```

Checking Table Contents

- Inherited complex code to set up a table
- Want to add at least a first test – of the **text in the cells**

	Number ▲	Label	Charge	SybylType	Xfrac + ESD	Yfrac + ESD	Z
1	1	Cl1	0	Cl	0.72083(6)	1.09418(9)	
2	2	O1	0	O.2	0.84300(14)	0.4969(2)	
3	3	N1	0	N.am	0.98694(15)	0.7154(2)	(
4	4	H1	0	H	1.0395	0.6419	
5	5	C1	0	C.2	0.87130(19)	0.6517(3)	(
6	6	C2	0	C.2	0.78474(18)	0.7753(3)	
7	7	C3	0	C.2	0.8225(2)	0.9459(3)	(
8	8	C4	0	C.ar	0.94472(19)	1.0112(3)	(
9	9	C5	0	C.ar	0.9874(2)	1.1899(3)	(
10	10	H2	0	H	0.9345	1.2749	
11	11	C6	0	C.ar	1.1052(3)	1.2382(4)	

Verifying a QTableWidget



	Number	Label	Charge	SybylType	Xfrac + ESD	Yfrac + ESD
1	1	Cl1	0	Cl	0.72083(6)	1.09418(9)
2	2	O1	0	O.2	0.84300(14)	0.4969(2)
3	3	N1	0	N.am	0.98694(15)	0.7154(2)
4	4	H1	0	H	1.0395	0.6419
5	5	C1	0	C.2	0.87130(19)	0.6517(3)
6	6	C2	0	C.2	0.78474(18)	0.7753(3)
7	7	C3	0	C.2	0.8225(2)	0.9459(3)
8	8	C4	0	C.ar	0.94472(19)	1.0112(3)
9	9	C5	0	C.ar	0.9874(2)	1.1899(3)
10	10	H2	0	H	0.9345	1.2749
11	11	C6	0	C.ar	1.1052(3)	1.2382(4)

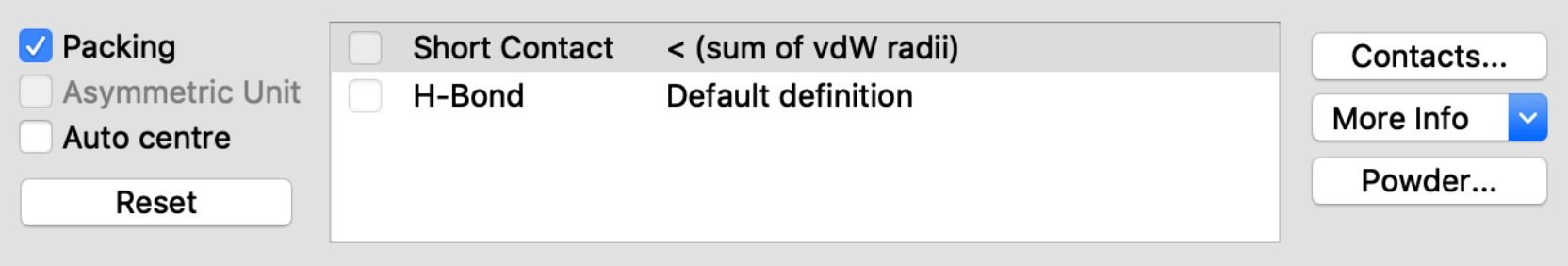
```
TEST_CASE("It approves a QTableWidget")
{
    // A note on naming: QTableWidget is a concrete class that implements
    // the more general QTableView. Here we create a QTableWidget,
    // for convenience.

    QTableWidget tableWidget;
    populateTable(tableWidget);
    ApprovalTestsQt::verifyQTableView(tableWidget);
}
```

Approval file: .tsv

Number	Label	Charge	SybylType	Xfrac + ESD	Yfrac + ESD	Zfrac + ESD	Symm. op.
1	Cl1	0	Cl	0.72083(6)	1.09418(9)	0.68032(5)	x,y,z
2	O1	0	O.2	0.84300(14)	0.4969(2)	0.52577(12)	x,y,z
3	N1	0	N.am	0.98694(15)	0.7154(2)	0.56335(12)	x,y,z
4	H1	0	H	1.0395 0.6419	0.5420	x,y,z	
5	C1	0	C.2	0.87130(19)	0.6517(3)	0.56084(15)	x,y,z
6	C2	0	C.2	0.78474(18)	0.7753(3)	0.60122(14)	x,y,z
7	C3	0	C.2	0.8225(2)	0.9459(3)	0.63380(14)	x,y,z
8	C4	0	C.ar	0.94472(19)	1.0112(3)	0.63352(14)	x,y,z
9	C5	0	C.ar	0.9874(2)	1.1899(3)	0.66585(16)	x,y,z
10	H2	0	H	0.9345 1.2749	0.6899	x,y,z	
11	C6	0	C.ar	1.1052(3)	1.2382(4)	0.66196(17)	x,y,z
12	H3	0	H	1.1320 1.3565	0.6831	x,y,z	
13	C7	0	C.ar	1.1857(2)	1.1136(4)	0.62701(17)	x,y,z
14	H4	0	H	1.2661 1.1481	0.6254	x,y,z	
15	C8	0	C.ar	1.1473(2)	0.9387(3)	0.59469(16)	x,y,z
16	H5	0	H	1.2014 0.8549	0.5712	x,y,z	
17	C9	0	C.ar	1.02672(19)	0.8885(3)	0.59747(14)	x,y,z
18	C10	0	C.ar	0.65855(19)	0.7054(3)	0.60428(15)	x,y,z
19	C11	0	C.ar	0.6305(2)	0.5733(3)	0.66680(17)	x,y,z
20	H6	0	H	0.6908 0.5265	0.7058	x,y,z	
21	C12	0	C.ar	0.5135(2)	0.5104(4)	0.6718(2)	x,y,z
22	H7	0	H	0.4953 0.4228	0.7145	x,y,z	

Checking Screenshots? No!

- Not recommended, in general
- Styles differ between Operating Systems
- Better to **test behaviour**, not appearance



What next for ApprovalTests.cpp.Qt?

- Fix Qt Test macros problem
- Seek feedback
- Could…
  - Support more test frameworks
  - Support approving more Qt types
  - Add real-world examples to the docs

Contents

Summary

- It’s never too late to start testing!
- What to test
  - Try to test smaller units of code, such as individual widgets
  - Test behaviors, not appearance (mainly)
  - Keep application logic separate from GUI
- Make tests …
  - Easy to write
  - Hide details of custom widgets behind helper functions, in tests
  - Expressive and readable with fixtures
- ApprovalTests.Cpp.Qt feedback welcome!

Quickly and Effectively Test Qt Desktop Applications

- **All links from this talk, and more, via:**
  - bit.ly/TestingQt
  - github.com/claremacrae/talks

- **Sustainable and efficient testing and refactoring of legacy code**
- **Consulting & training** via “Clare Macrae Consulting Ltd”
  - claremacrae.co.uk
  - clare@claremacrae.co.uk