Quickly Testing Qt Desktop Applications with Approval Tests

Clare Macrae (She/her)
clare@claremacrae.co.uk
16 September 2020
CppCon (Online)

Audience:
Developers testing Desktop GUIs, including Qt-based ones

Approval Tests: claremacrae.co.uk/conferences/presentations.html

Contents

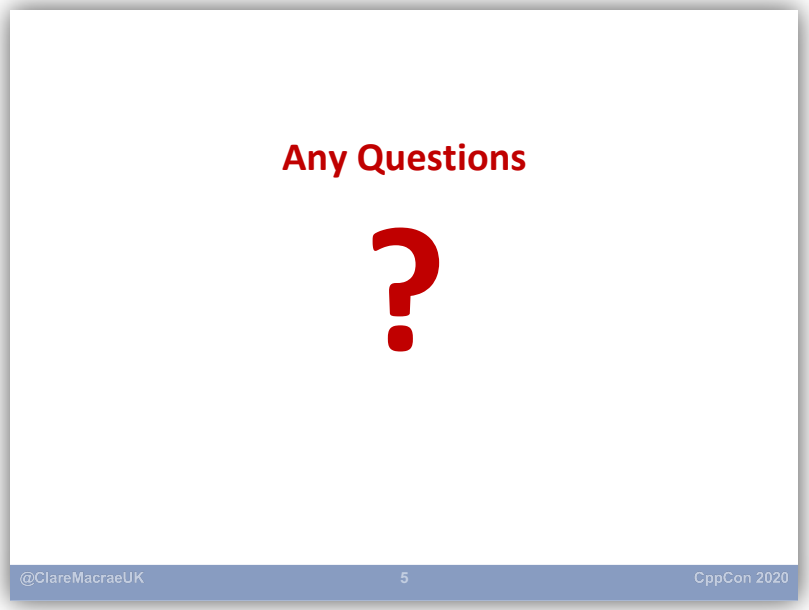
About Me

- 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…

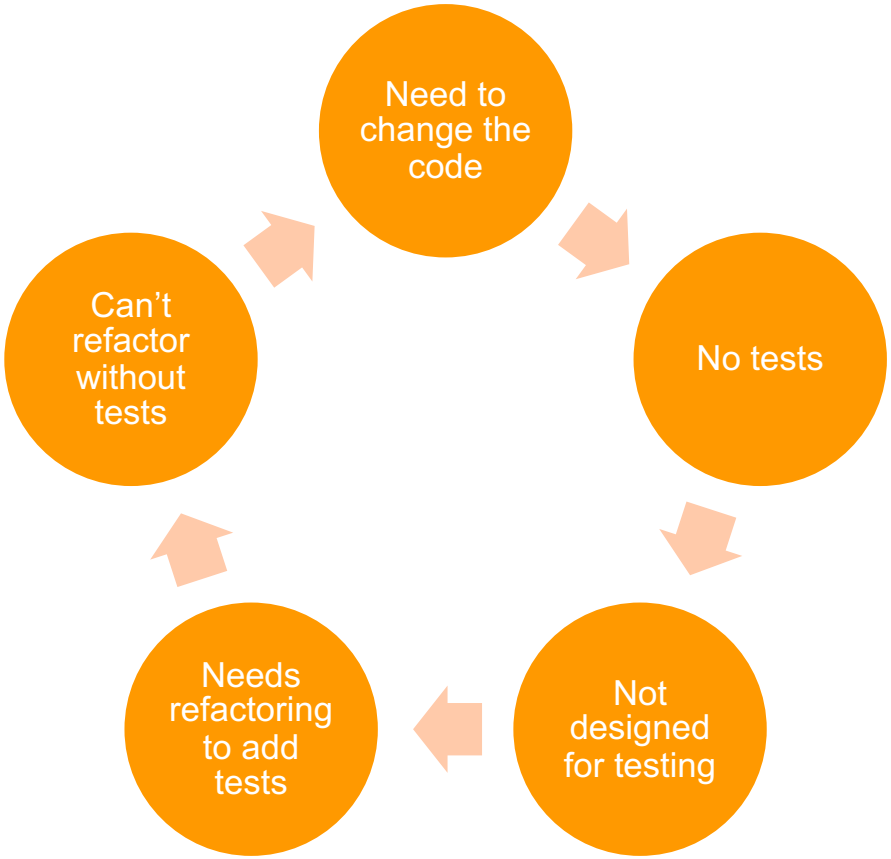

Building a
network for the
gigabit age


Any Questions
?
@ClareMacraeUK
5
CppCon 2020


Fuzzing/Testing Track
Test track at CppCon 2020
fuzzing()
1
E

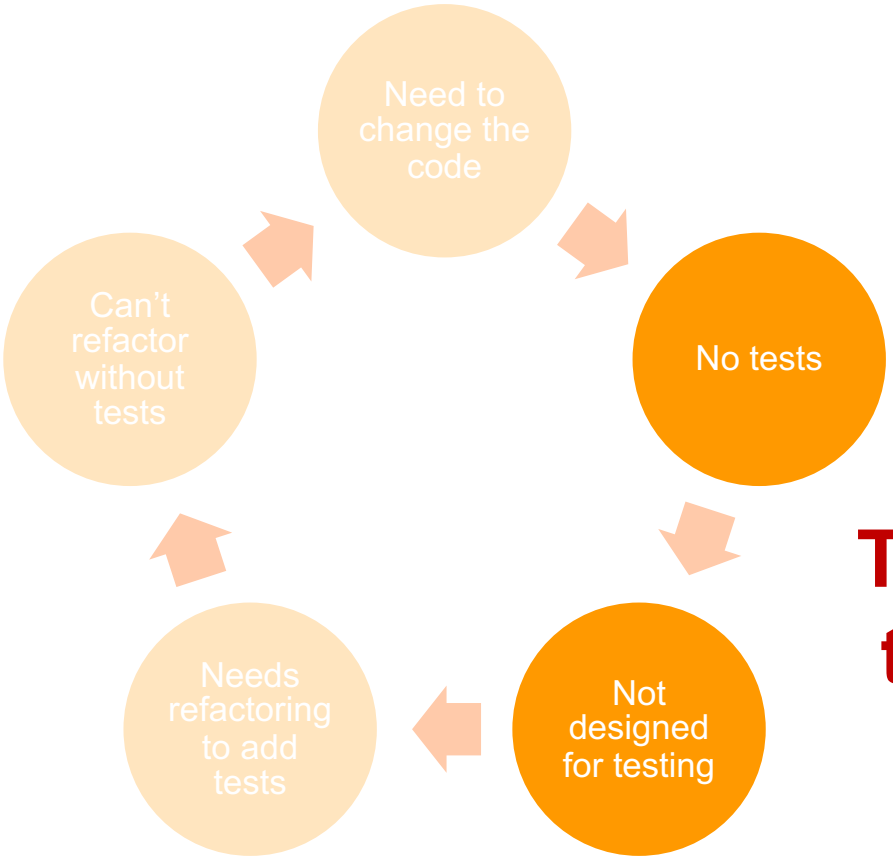
Typical Scenario

- 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

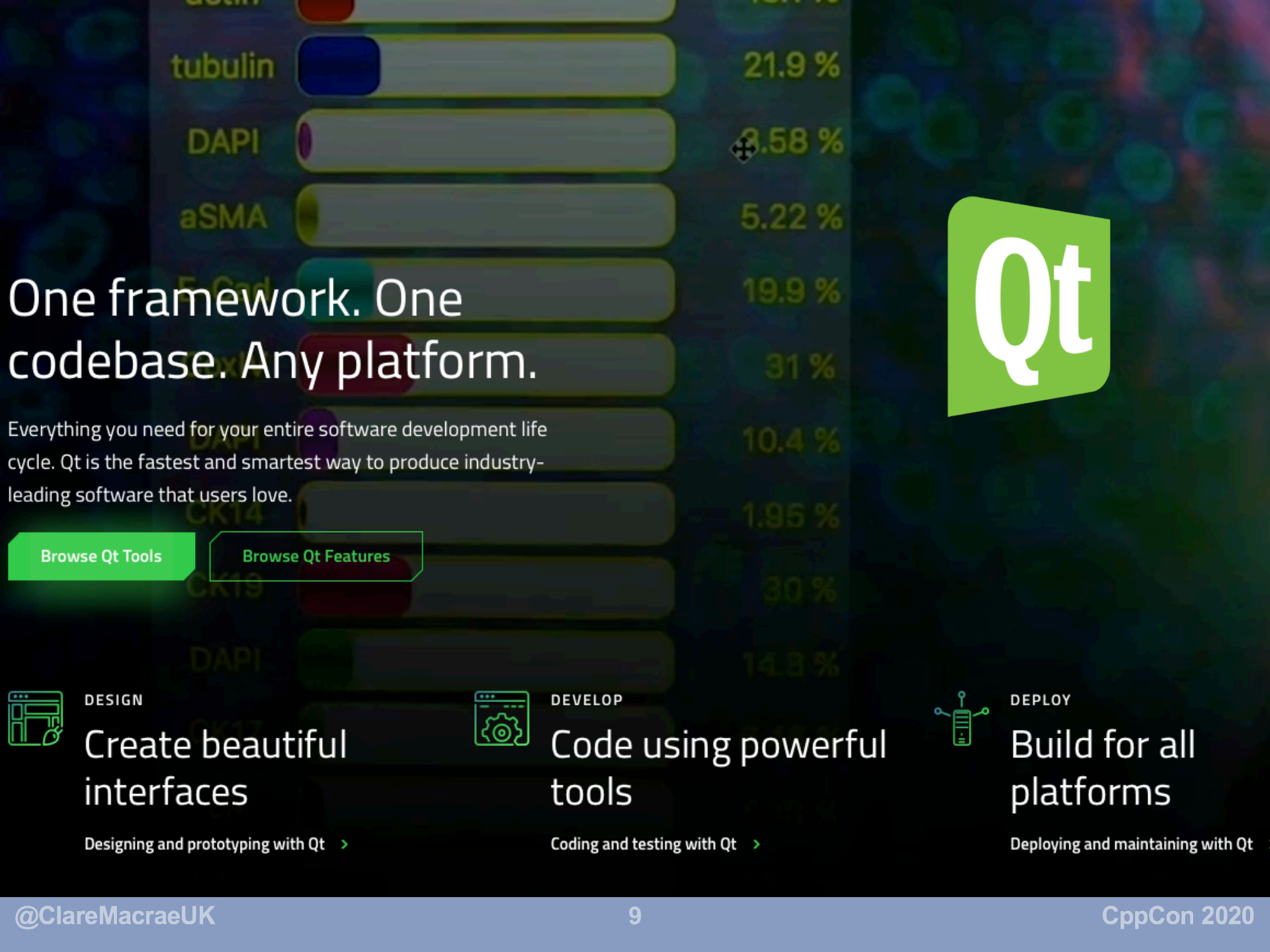
- I've inherited some qt GUI code
- It's valuable
- I need to add feature
- Or fix bug
- How can I ever break out of this loop?



Topics of this talk

Contents

- Introduction
- **Qt**
  - –Setting Up Testing
  - –Error-prone Things
  - –Approval Tests
- Extras
  - –Tools
  - –Summary


tubulin
21.9 %
DAPI
aSMA
5.22 %
19.9 %
31 %
10.4 %
1.95 %
30 %
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

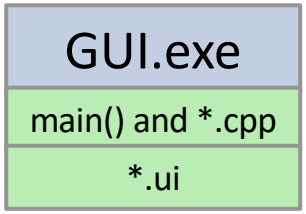
Give you **Confidence**
to **Start testing**
your **Qt application**
Effectively

Contents

Q: How do I add tests to existing app?

A: Ah – Good point – You don't

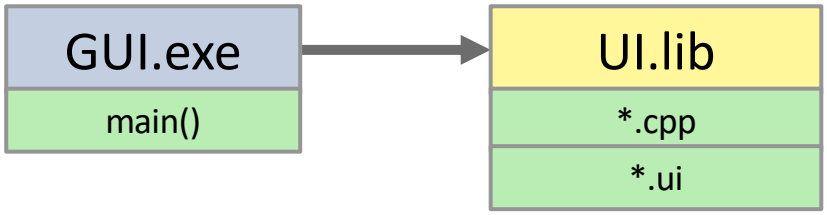
Introduce static lib for tests



Color Key

Executable

Introduce static lib for tests

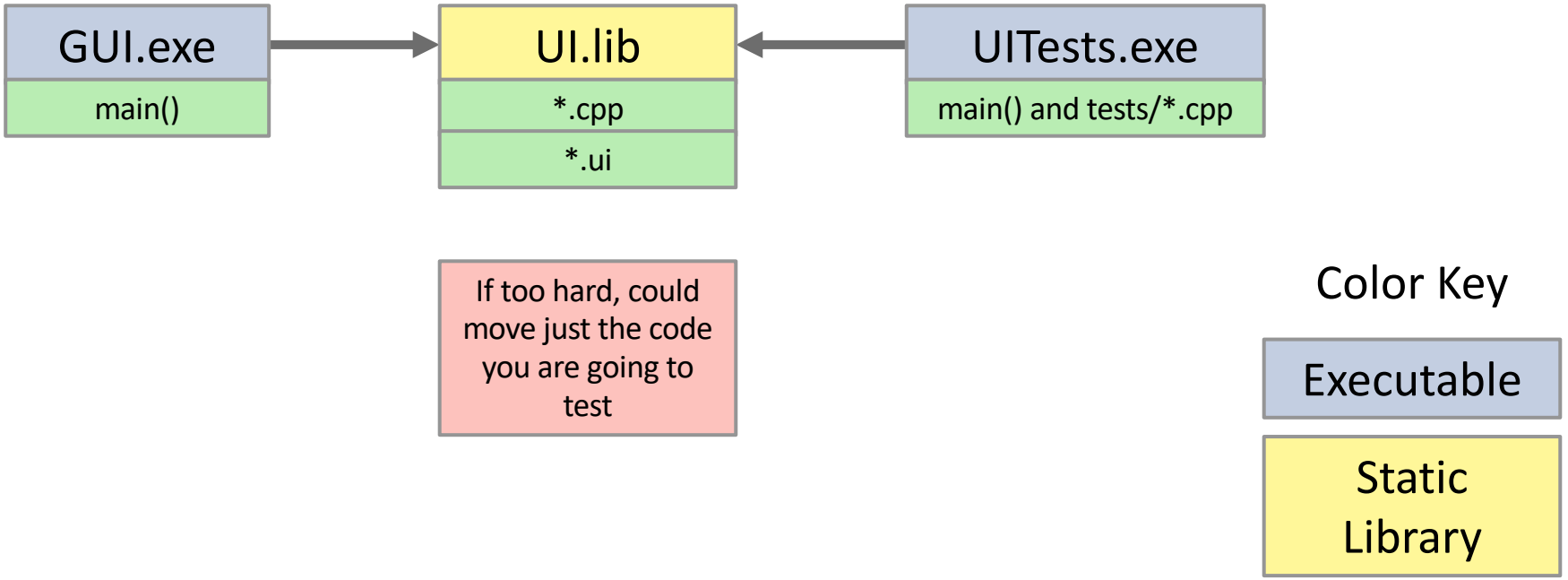


Color Key

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

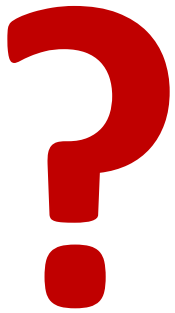
Release process	Add the dynamic libraries to your distributions	No Change

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!

Any Questions



Context: Type of testing

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

Creating the Test Executable

C++ Test Runners

- Lots to choose from:

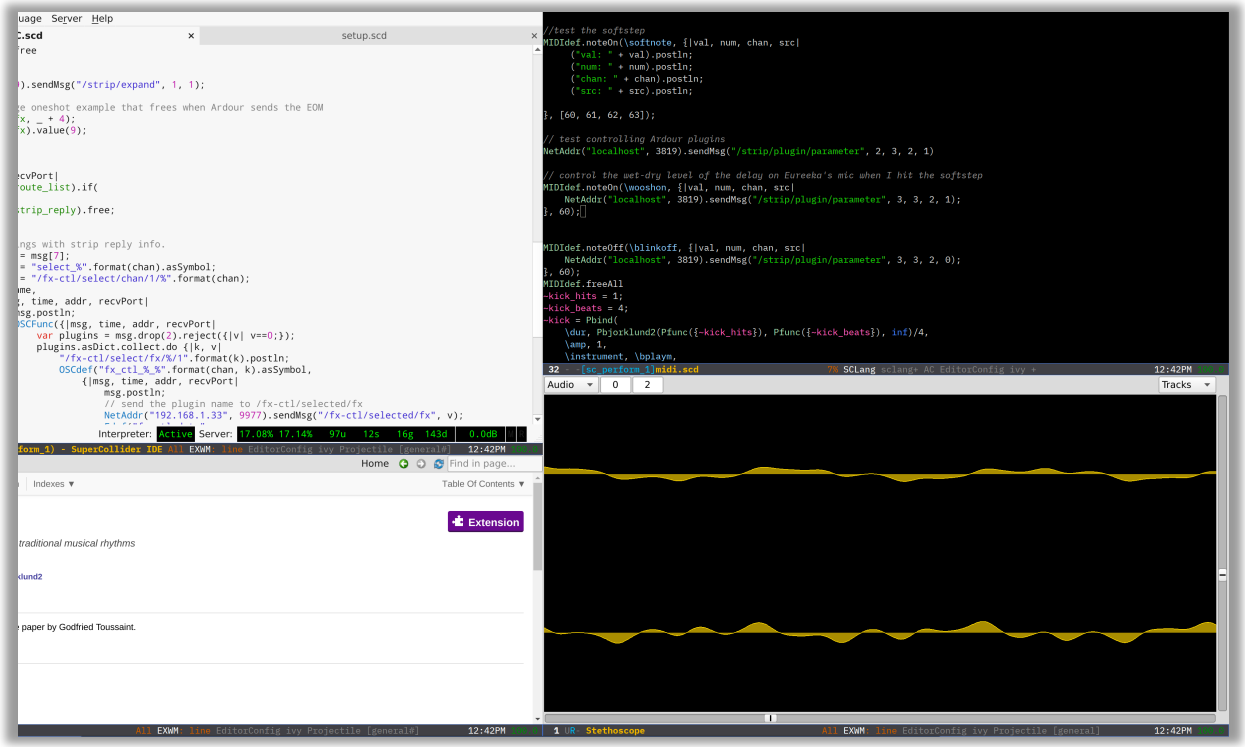
- Catch2, with a little bit of Qt Test
- Tricks needed to use Qt with Catch2
  - Useful if using different framework

Setting up testsuite main()

```
// 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

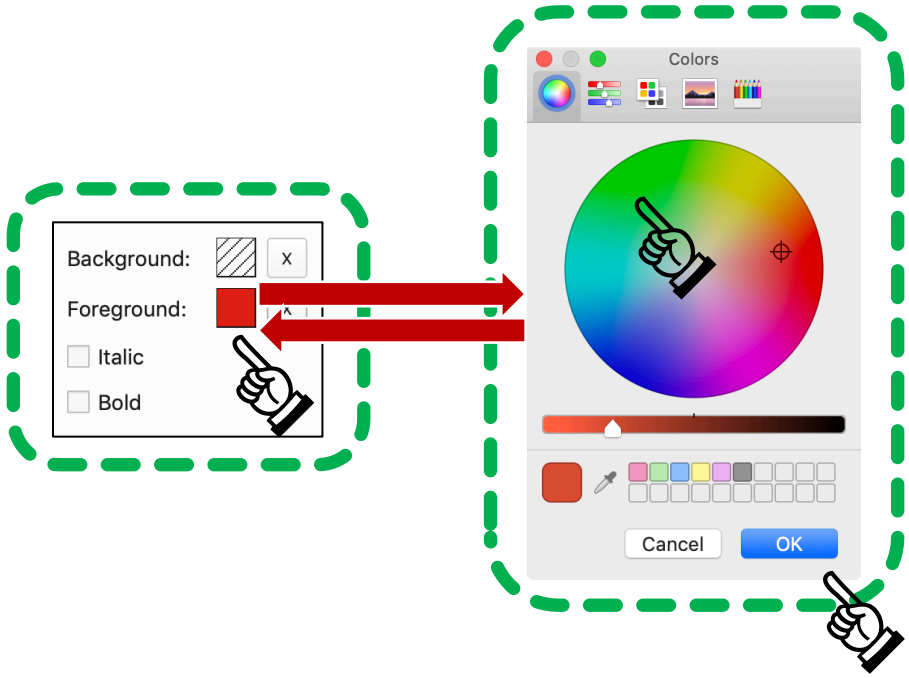
- supercollider.github.io
- platform for audio synthesis and algorithmic composition



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);
}
```

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 == "");
}
```

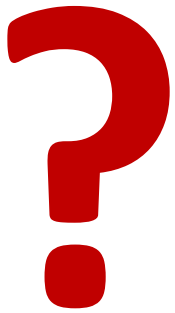
ColorWidget changing state

```
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!

Any Questions



Context: “Go To Line” Panel

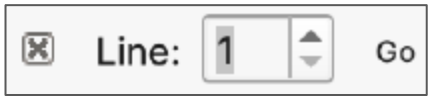
babbling brook.scd

```
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

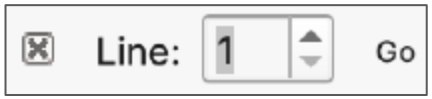
Goal: Demo the basic code

- **Warning**: Some detail coming up!
  - Later I'll show how to avoid it…
- Basic steps:
  - Arrange
  - Act
  - Assert



Arrange: Set up Widget

```
TEST_CASE("GoToLineTool emits signal when Go button clicked")
{
    // ------------------------------------------------------
    // Arrange
    GoToLineTool widget;
    widget.raise();
    widget.show();
    widget.setMaximum(27);

    // This is needed to make sure text in the spinner is selected, so that
    // as we type in characters, the initial text ("1") is erased
    widget.setFocus();
```



Arrange: Accessing private widgets – V1

```
// Allow us to interact with widgets inside GoToLineTool:
auto spinner = widget.findChild<QSpinBox*>();
REQUIRE(spinner != nullptr);

auto goButton = widget.findChild<QToolButton*>();
REQUIRE(goButton != nullptr);
```

✗ **Beware**: findChild() not recommended, especially in production code!

Test actions – Signals and Slots


```
emit activated(17);
```

```
15    var pitch_contour, drum_osc, drum_lpf, drum_env;
16    var beater_source, beater_hpf, beater_lpf, lpf_cutoff_contour, beater_env;
17    var kick_mix;
18    pitch_contour = Line.kr(freq*2, freq, 0.02);
```

```
QSignalSpy(&mGoToLineWidget,
&GoToLineTool::activated)
```

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

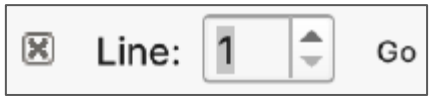
Arrange: Listen to activate() signal


```
// Enable tracking of one signal
// New-style code
QSignalSpy activatedSpy(&widget, &GoToLineTool::activated);
REQUIRE(activatedSpy.isValid());

// Qt4-style code
//QSignalSpy activatedSpy(&widget, SIGNAL(activated(int)));
//REQUIRE(activatedSpy.isValid());
```

Act: Generate user events

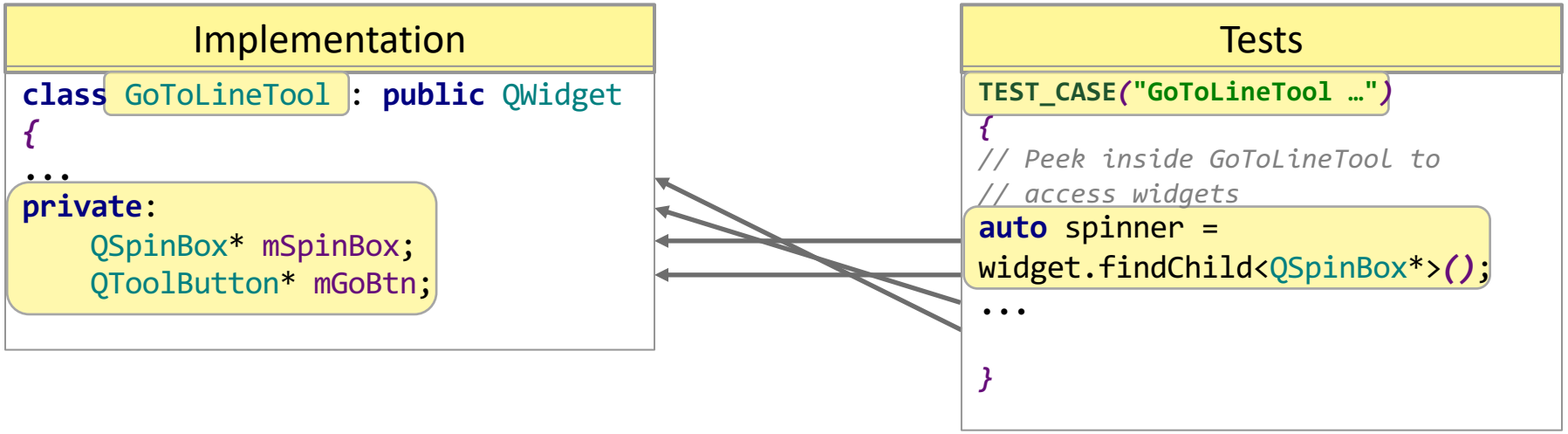

```
// ------------------------------------------------------
// Act
// Type a number, one digit at a time
QTest::keyClicks(spinner, "1");
QTest::keyClicks(spinner, "7");

// Clicking the Go button:
goButton->click();
```

Assert: Check the changes




```
    // ------------------------------------------------------
    // Assert

    // Check the activated() signal was emitted only once:
    REQUIRE(activatedSpy.count() == 1);

    // And check that the signal emitted the new value:
    QList<QVariant> arguments = activatedSpy.takeFirst();
    const QVariant& argument = arguments.at(0);
    CHECK(argument.type() == QVariant::Int);
    CHECK(argument.toInt() == 17);
}
```

High level view – version 1



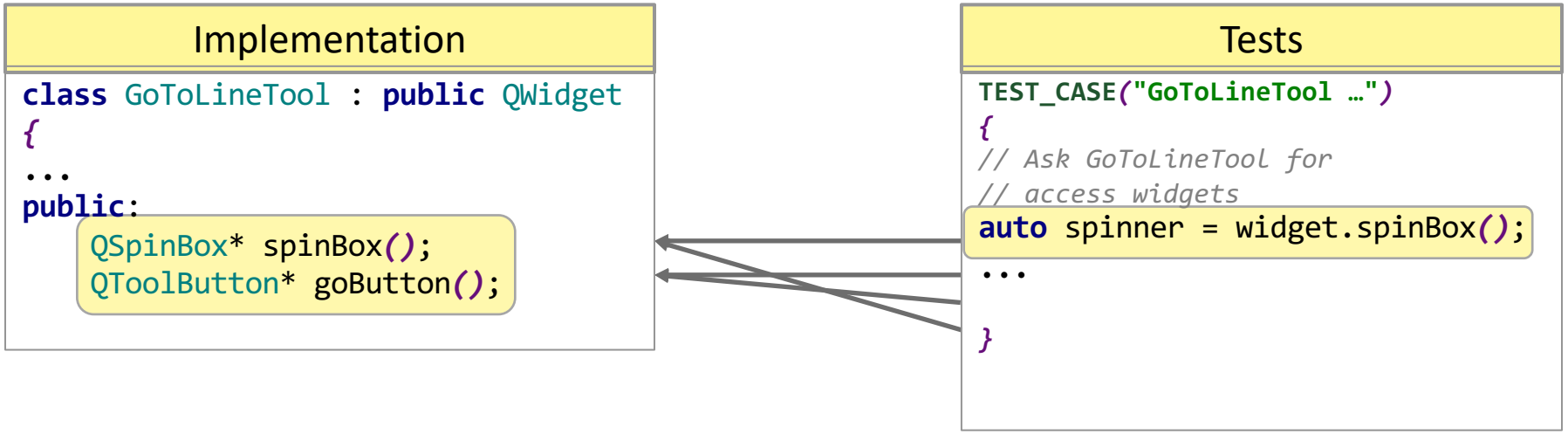
Goal: Make tests super easy to write

- findChild() is really fragile
  - prone to future breakage at run-time
  - not checked at compile-time
  - finding interface type: `QAbstractButton` better than `QToolButton`

Testing Custom Qt Widgets – part 1

Option	Advantages	Disadvantages
widget.findChild()/findChildren()	Don't need to change code being tested	No compile time feedback. Maintenance overhead
Add **public** accessor to widget being tested	Testing easier	Maybe encourages misuse of library if internals exposed

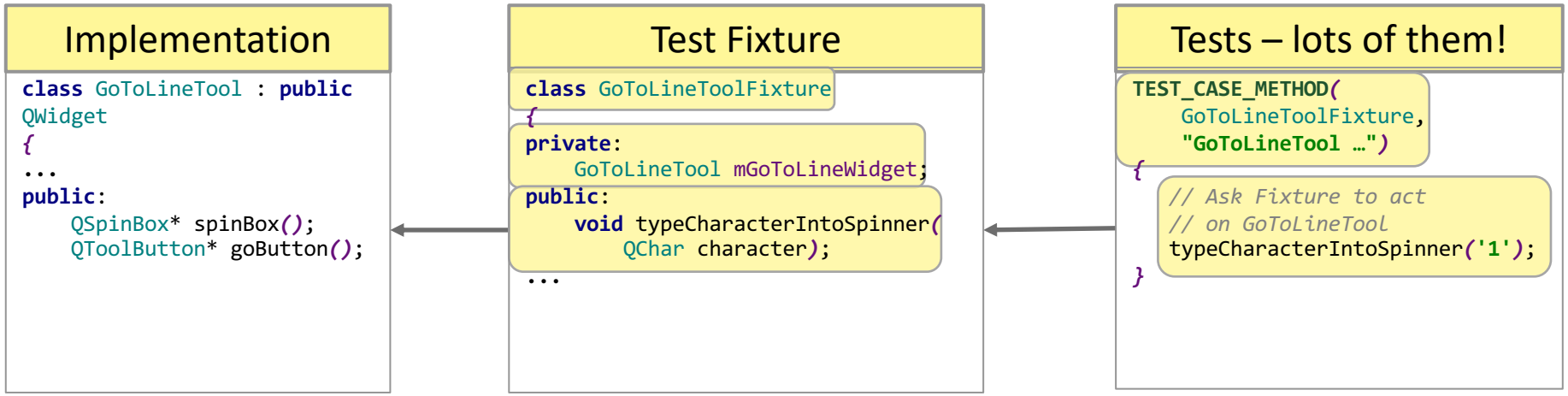
High level view – version 2



Goal: Hide implementation details

- Want to avoid tests knowing insides of widgets being tested
  - As much as possible
- Can we improve readability of test?
- Can we guard against changes to implementation?

High level view – version 3

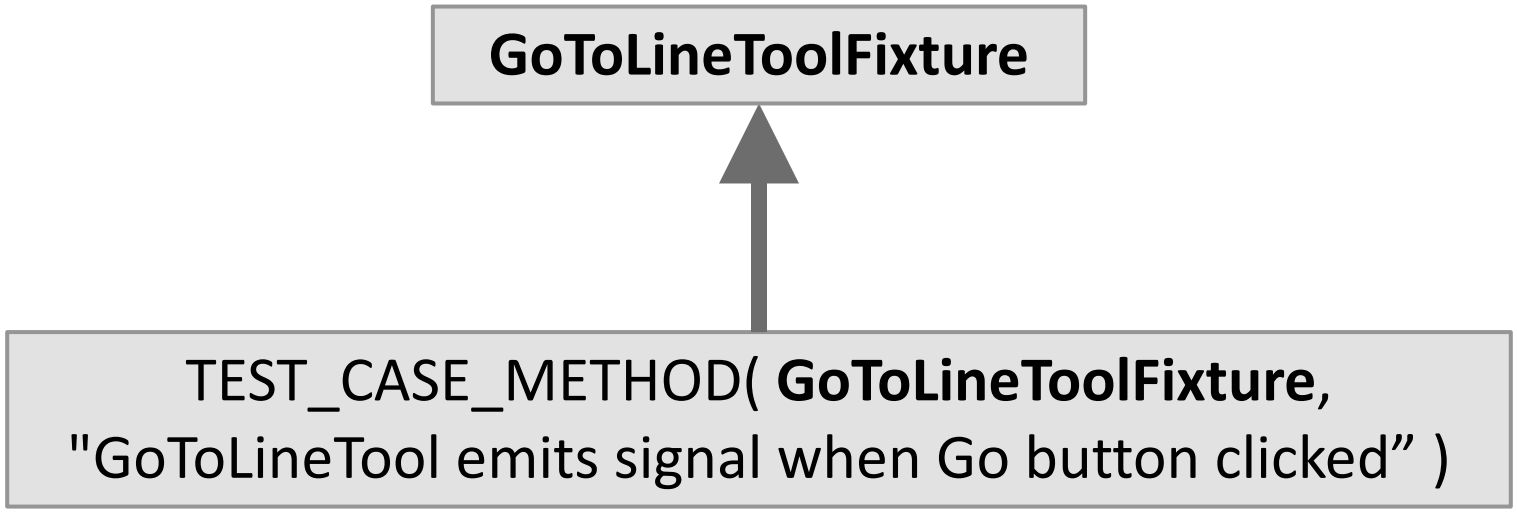


Introduce a Fixture

```
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);
}
```

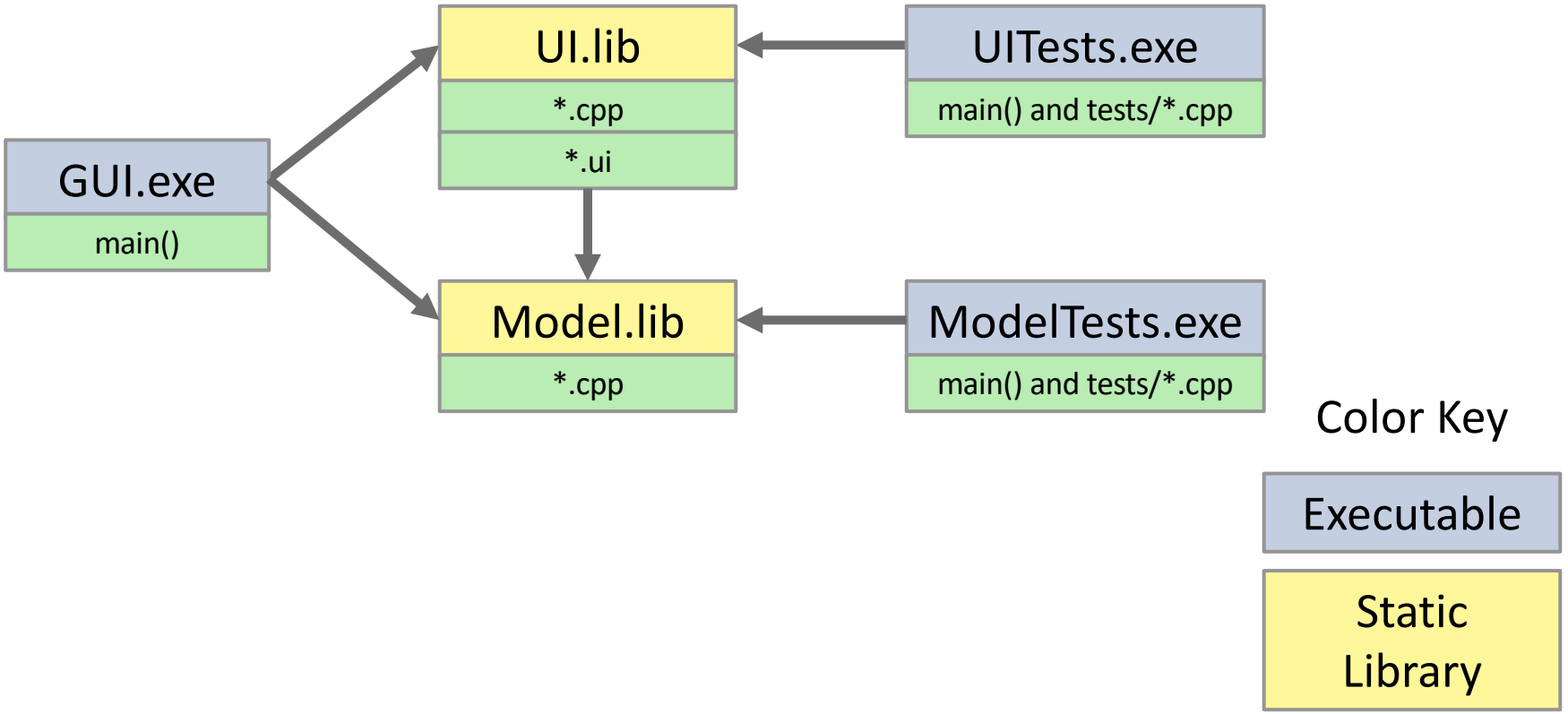
Testing Custom Qt Widgets – part 1

Option	Advantages	Disadvantages
widget.findChild()/findChildren()	Don't need to change code being tested	No compile time feedback. Maintenance overhead
Add **public** accessor to widget being tested	Testing easier	Maybe encourages misuse of library if internals exposed

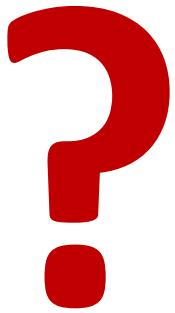
Testing Custom Qt Widgets – part 2

Option	Advantages	Disadvantages
widget.findChild()/findChildren()	Don't need to change code being tested	No compile time feedback. Maintenance overhead
Add **public** accessor to widget being tested	Testing easier	Maybe encourages misuse of library if internals exposed
Add **protected** accessor … Make the test fixture **inherit** the widget	Mostly hides the implementation	Maybe encourages inheritance to customize behavior. Test fixture must inherit class being tested – maybe confusing
Add **private** accessor … Make the test fixture a **friend** of the widget	Hides the implementation more	Friendship not inherited by test implementation – so there's more work to create the test fixture

Bonus Points: Separate logic from UI code



Any Questions



Contents

- Introduction
- Qt
  - Setting Up Testing
  - **Error-prone Things**
  - Approval Tests
- Extras
  - Tools
  - Summary

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()

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()`

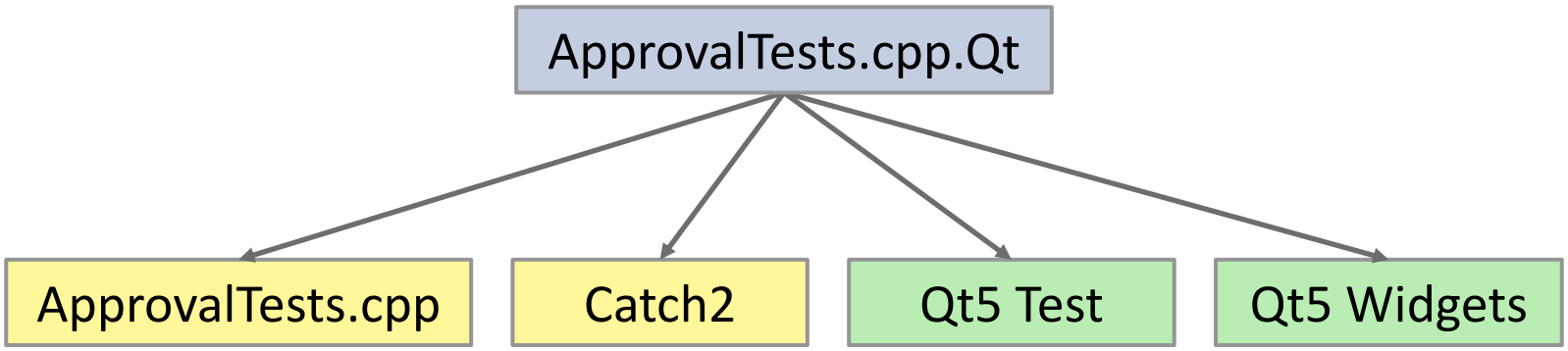
Contents

- Introduction
- Qt
  - Setting Up Testing
  - Error-prone Things
  - **Approval Tests**
- Extras
  - Tools
  - Summary

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

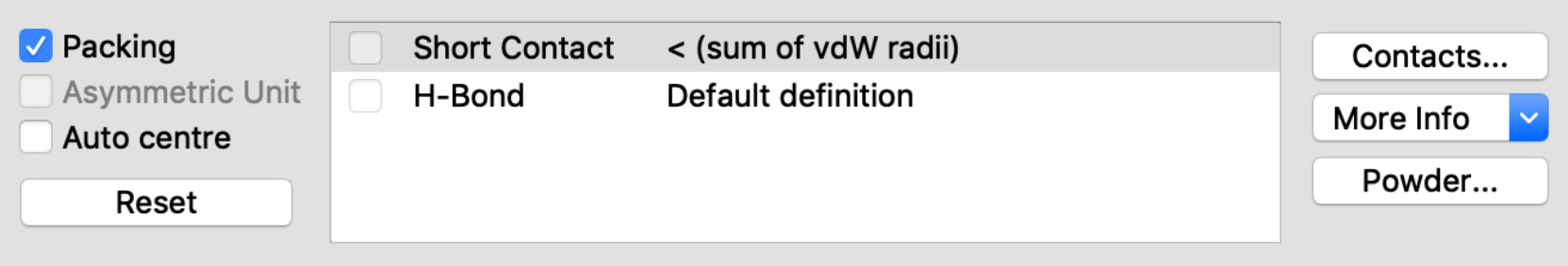
- Inherited complex code to set up a table
- Want to add at least a first test – of the **text in the cells**

	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)	

[Demo]

Checking Screenshots? No!

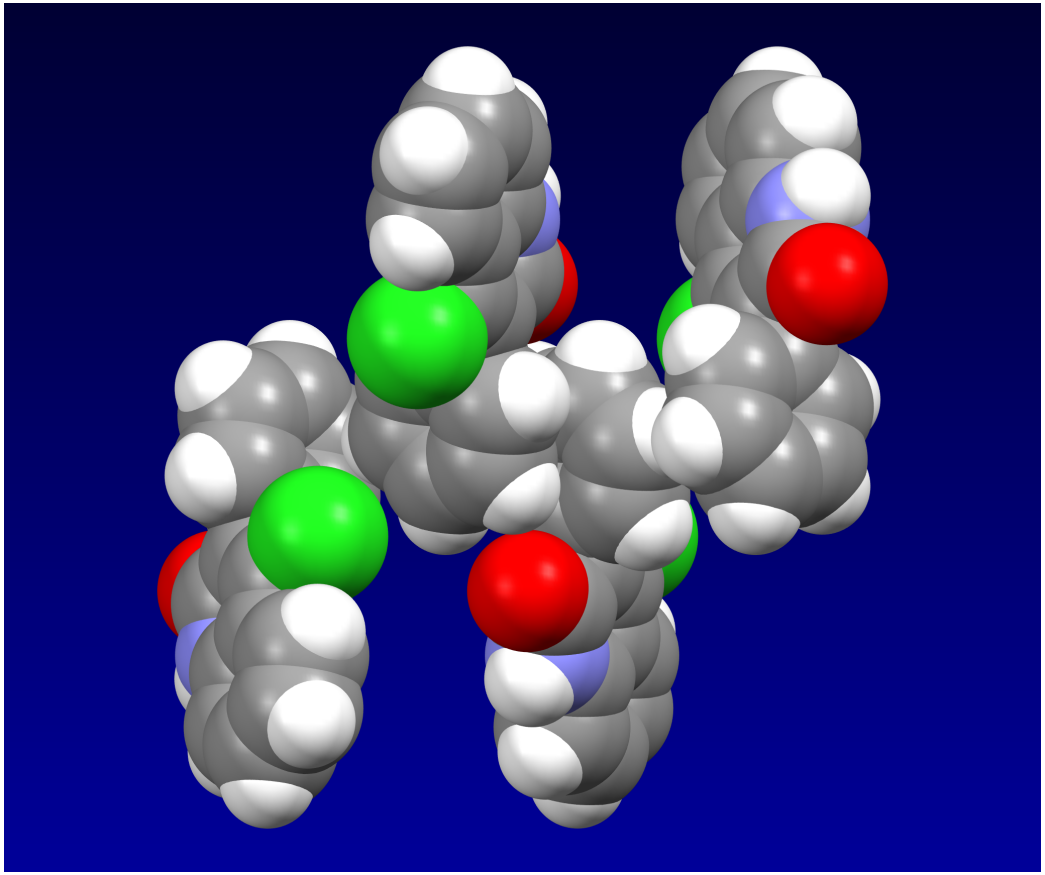
- Not recommended, in general
- Styles differ between Operating Systems
- Better to **test behaviour**, not appearance

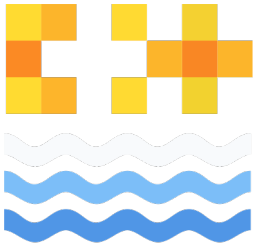


[Demo]

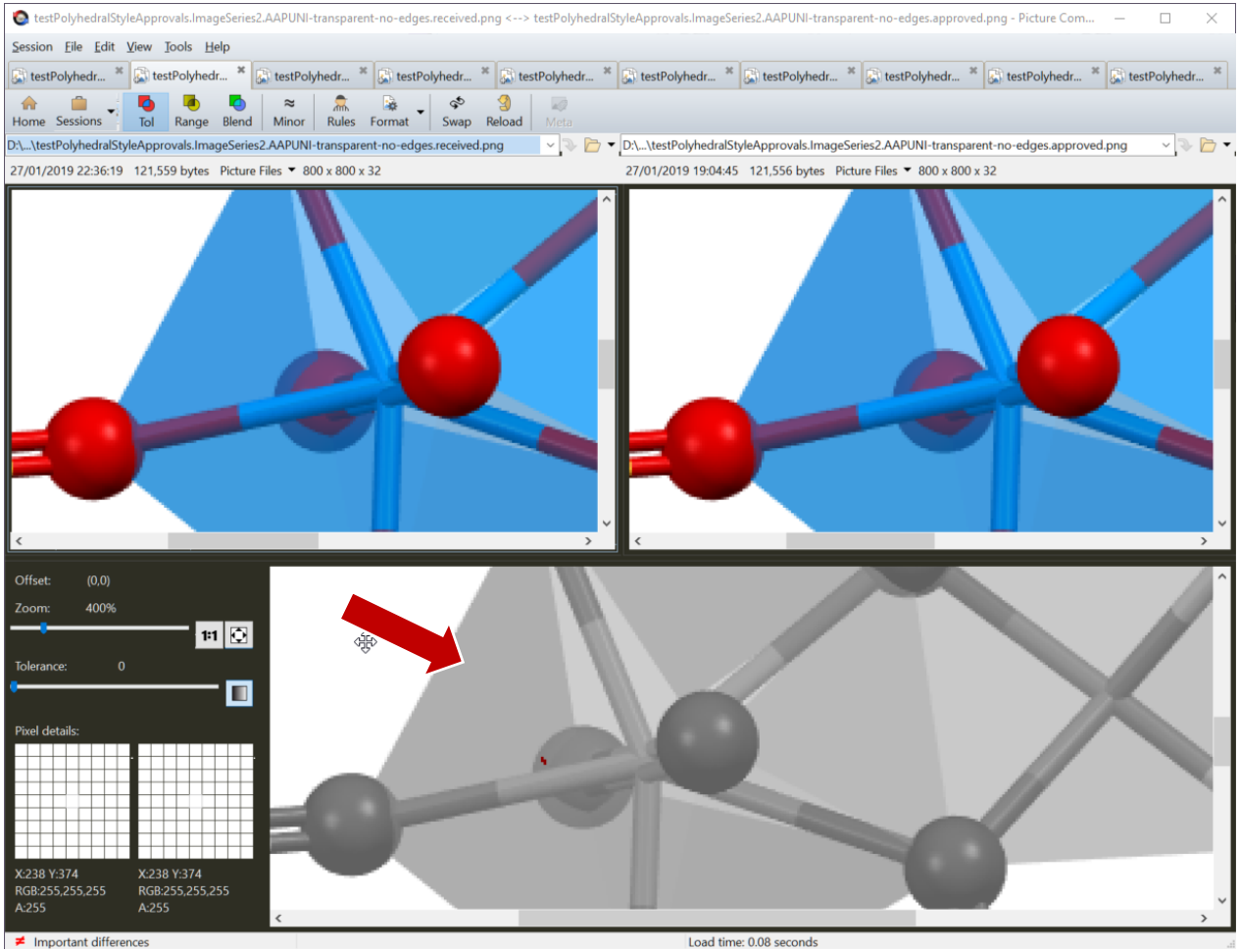
Checking rendering

- But if you have to do it…





2019


testPolyhedralStyleApprovals.ImageSeries2.AAPUNI-transparent-no-edges.received.png <--> testPolyhedralStyleApprovals.ImageSeries2.AAPUNI-transparent-no-edges.approved.png - Picture Com...
Session File Edit View Tools Help
Home Sessions Tol Range Blend Minor Rules Format Swap Reload Meta
D:\...\testPolyhedralStyleApprovals.ImageSeries2.AAPUNI-transparent-no-edges.received.png
D:\...\testPolyhedralStyleApprovals.ImageSeries2.AAPUNI-transparent-no-edges.approved.png
27/01/2019 22:36:19 121,559 bytes Picture Files 800 x 800 x 32
27/01/2019 19:04:45 121,556 bytes Picture Files 800 x 800 x 32
Offset: (0,0)
Zoom: 400%
1:1
Tolerance: 0
Pixel details:
X:238 Y:374
RGB:255,255,255
A:255
X:238 Y:374
RGB:255,255,255
A:255
Important differences
Load time: 0.08 seconds

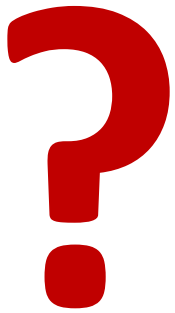
Caution!

- **Avoid Qt Test** macros in this project
  - **QCOMPARE, QVERIFY, QTRY_COMPARE** and so on
  - Any test failures will be silently swallowed
  - Tests will spuriously pass.
  - [November 2019: Fabian Kosmale at Qt has been really helpful on this – there is hope!]

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

Any Questions

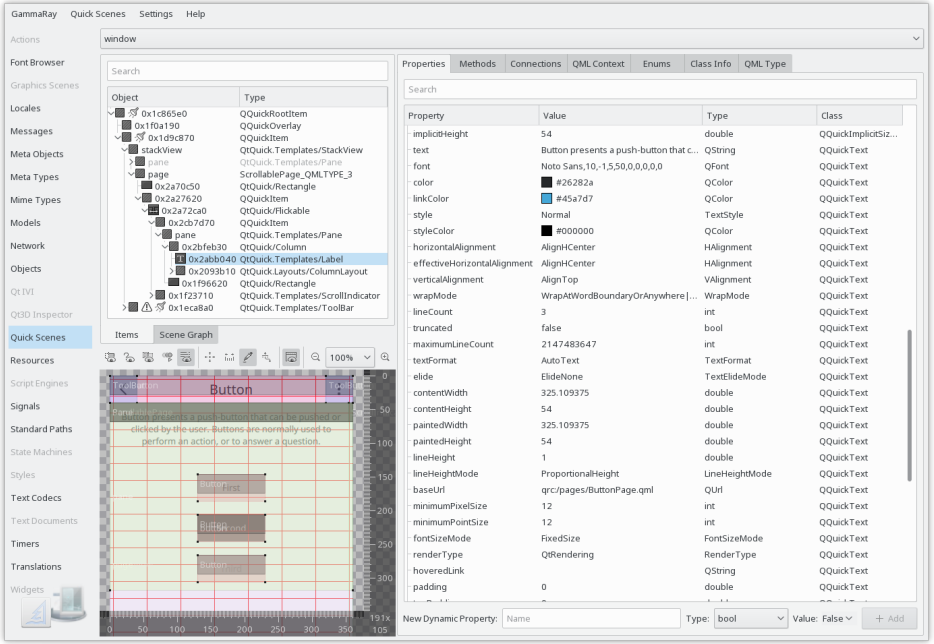


Contents

- Introduction
- Qt
  - Setting Up Testing
  - Error-prone Things
  - Approval Tests
- Extras
  - **Tools**
  - Summary

Navigation around your GUI

- Where is that widget in the source code?
  - Find strings in UI and search
  - Use KDAB GammaRay
    kdab.com/development-resources/qt-tools/gammaray



clazy

- Qt related static analysis and fixits
- Lots of useful warnings
- github.com/KDE/clazy
- Level 2 check: **old-style-connect**
  - Read the warnings on this check, though

Squish

froglogic

- 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

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 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