Quickly Testing Legacy C++ Code with Approval Tests

Clare Macrae (She/her)

clare@claremacrae.co.uk

16 September 2019

CppCon 2019, Aurora, USA

Contents

@ClareMacraeUK

Clare Macrae Consulting Ltd

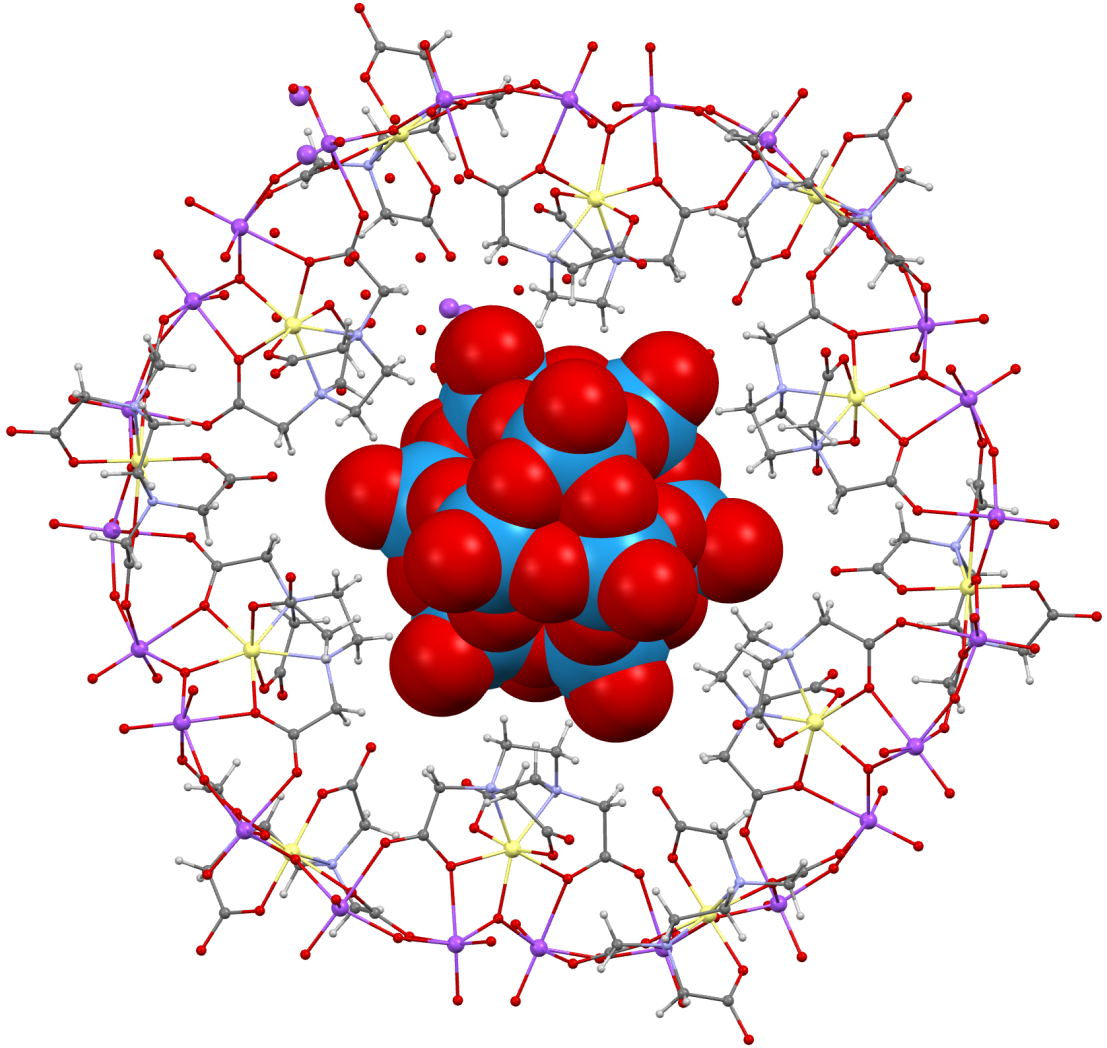
The Cambridge Structural Database (CSD)

The Cambridge Structural Database (CSD) is a highly curated and comprehensive resource.

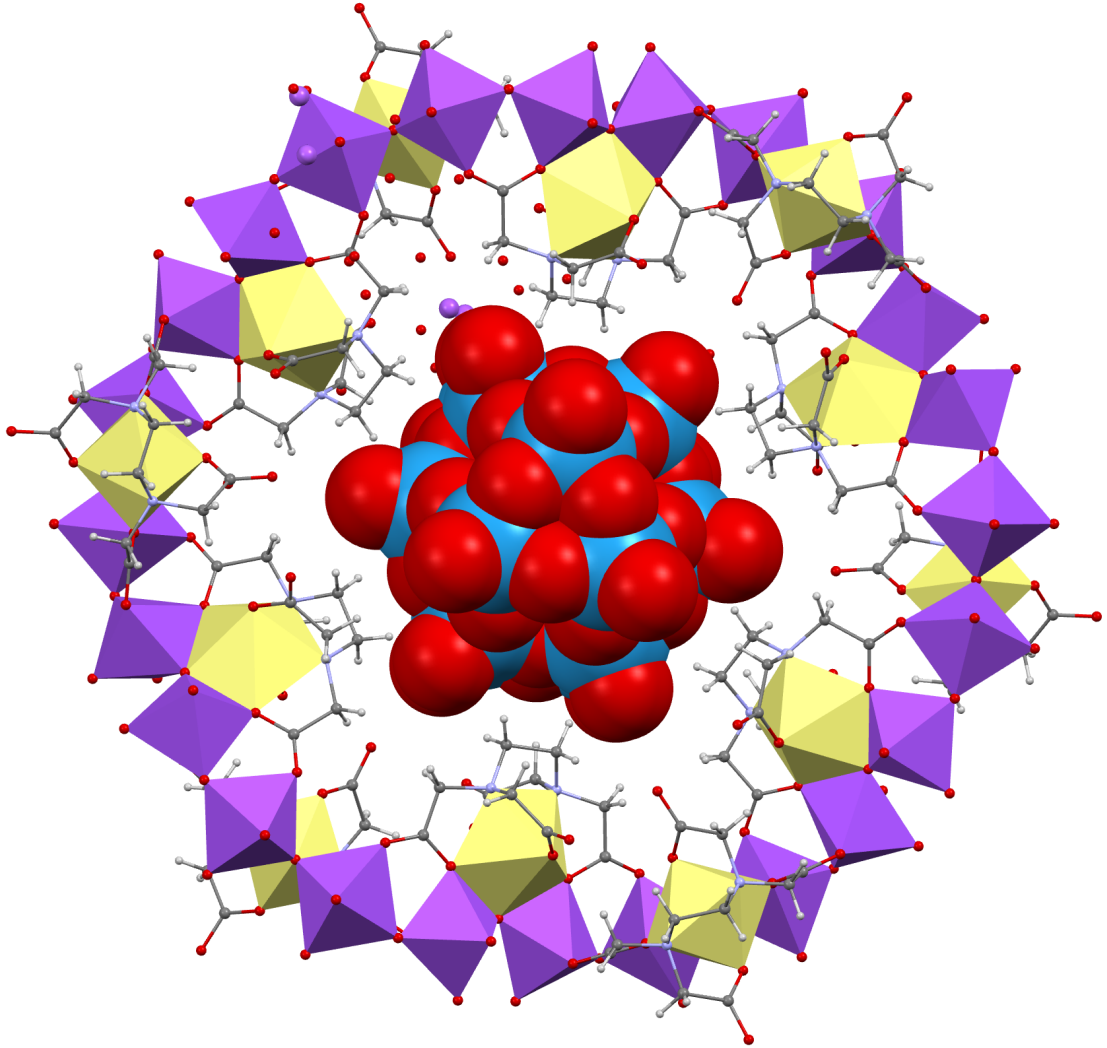
Established in 1965, the CSD is the world's repository for small-molecule organic and metal-organic crystal structures. Containing over one million structures from x-ray and neutron diffraction analyses, this unique database of accurate 3D structures has become an essential resource to scientists around the world.

Requirement

What I could do:

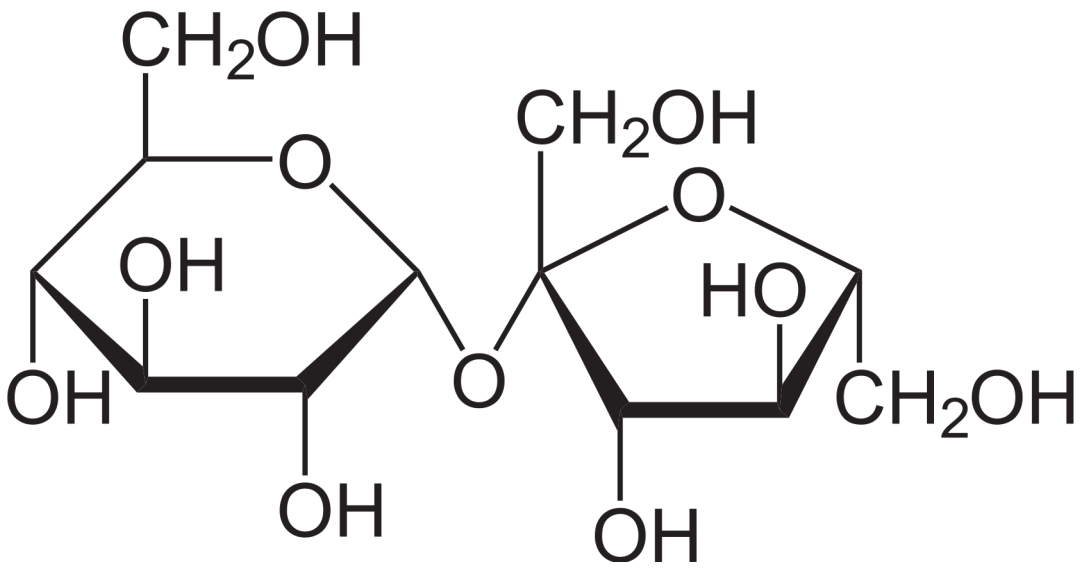


What I needed to do:

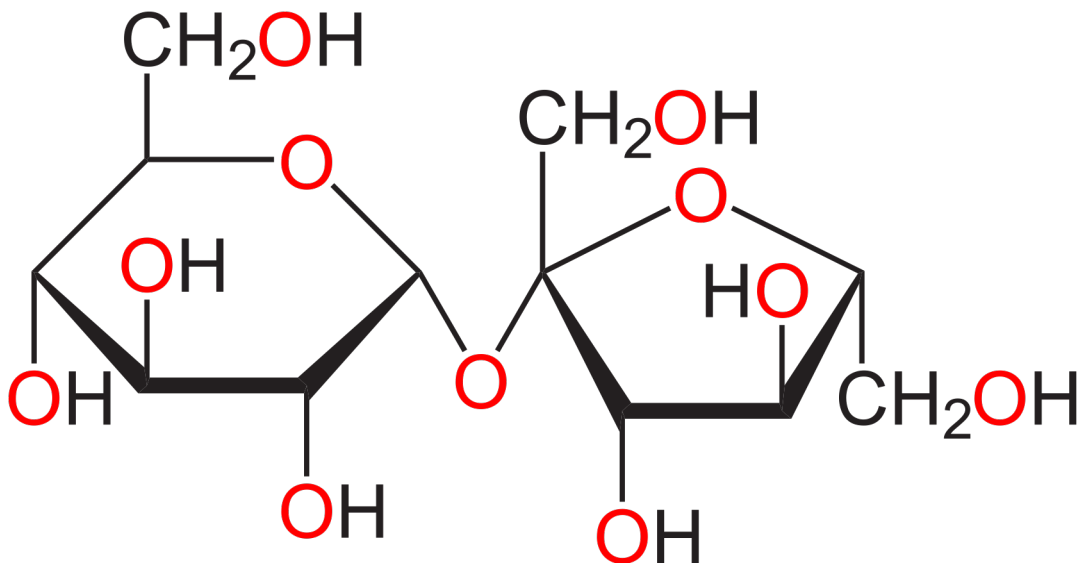


Sugar - sucrose

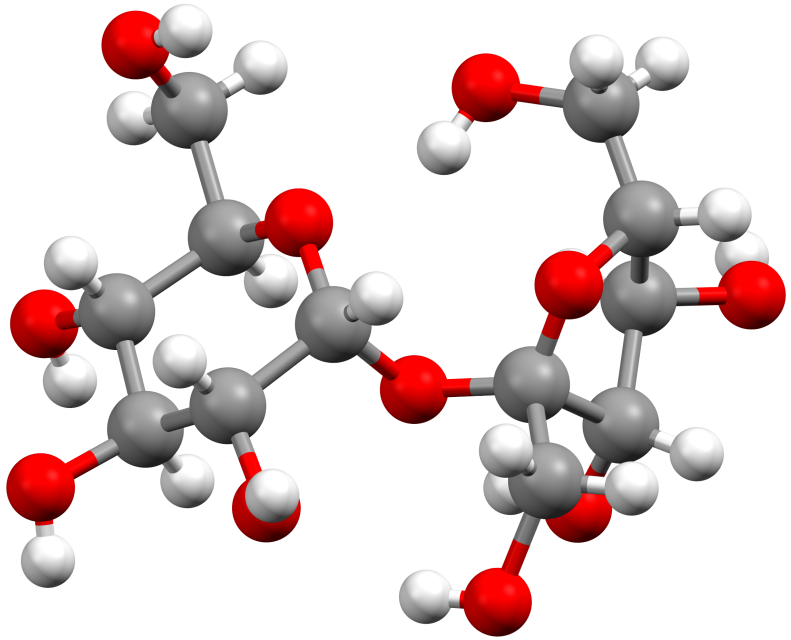
Sugar - sucrose



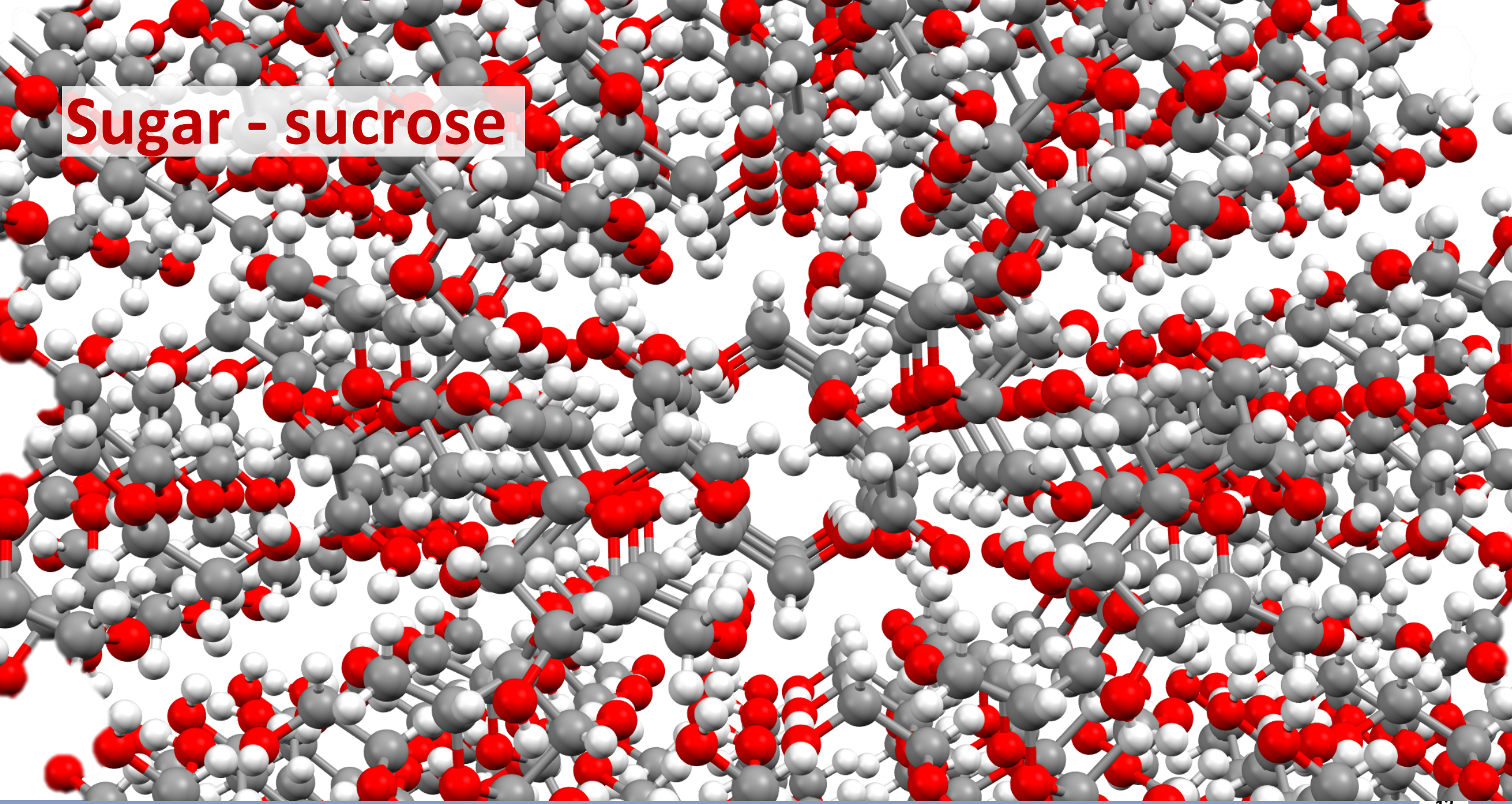
Sugar - sucrose



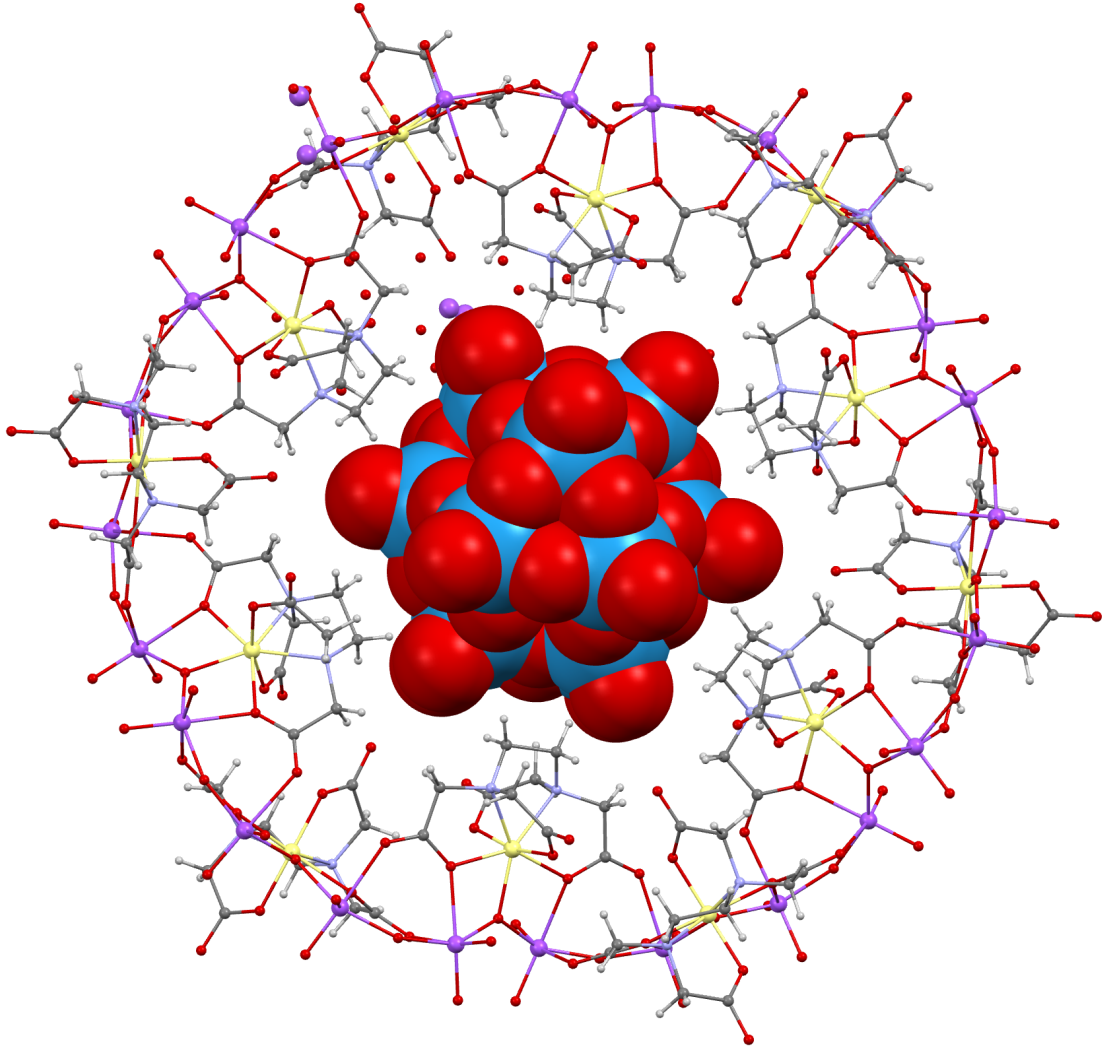
Sugar - sucrose



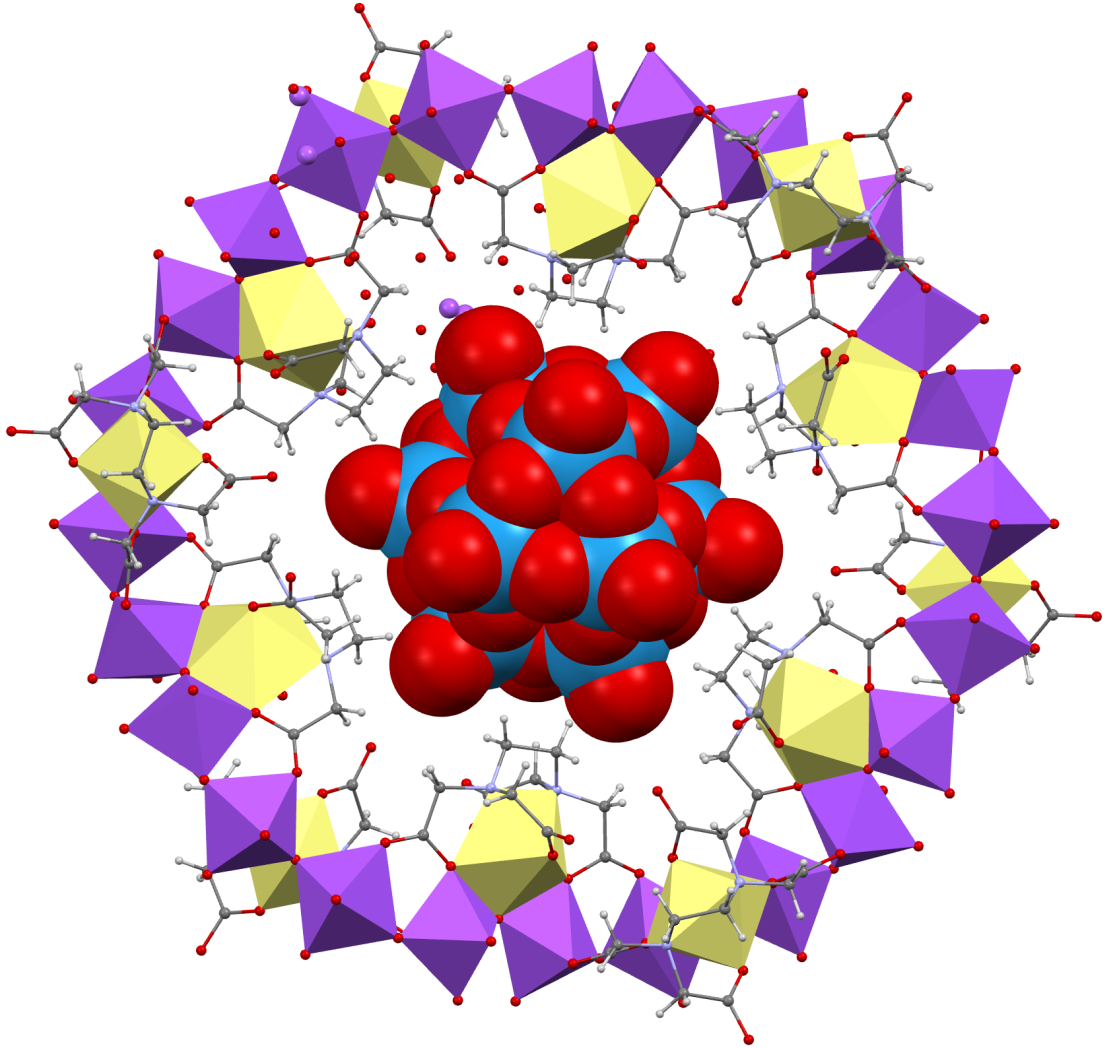
Sugar - sucrose



What I could do:



What I needed to do:



How could I implement this safely?

Llewellyn Falco
@LlewellynFalco

As part of expanding my c++ I was thinking of improving #ApprovalTests and making it work with GoogleTest. Anyone interested in pairing on that?

12:52 PM - 26 Nov 2017

Llewellyn Falco
@LlewellynFalco

As part of expanding my c++ I was thinking of improving #ApprovalTests and making it work with GoogleTest. Anyone interested in pairing on that?

12:52 PM - 26 Nov 2017

Clare Macrae
@ClareMacraeUK

Replying to @LlewellynFalco

Would be interested in hearing more.

2:46 PM - 26 Nov 2017

~2 Years of remote pairing later…

Goal:
Share techniques for easier testing in challenging scenarios

Contents

- Introduction
- **Legacy Code** 
- Golden Master
- Approval Tests
- Example
- Resources
- Summary

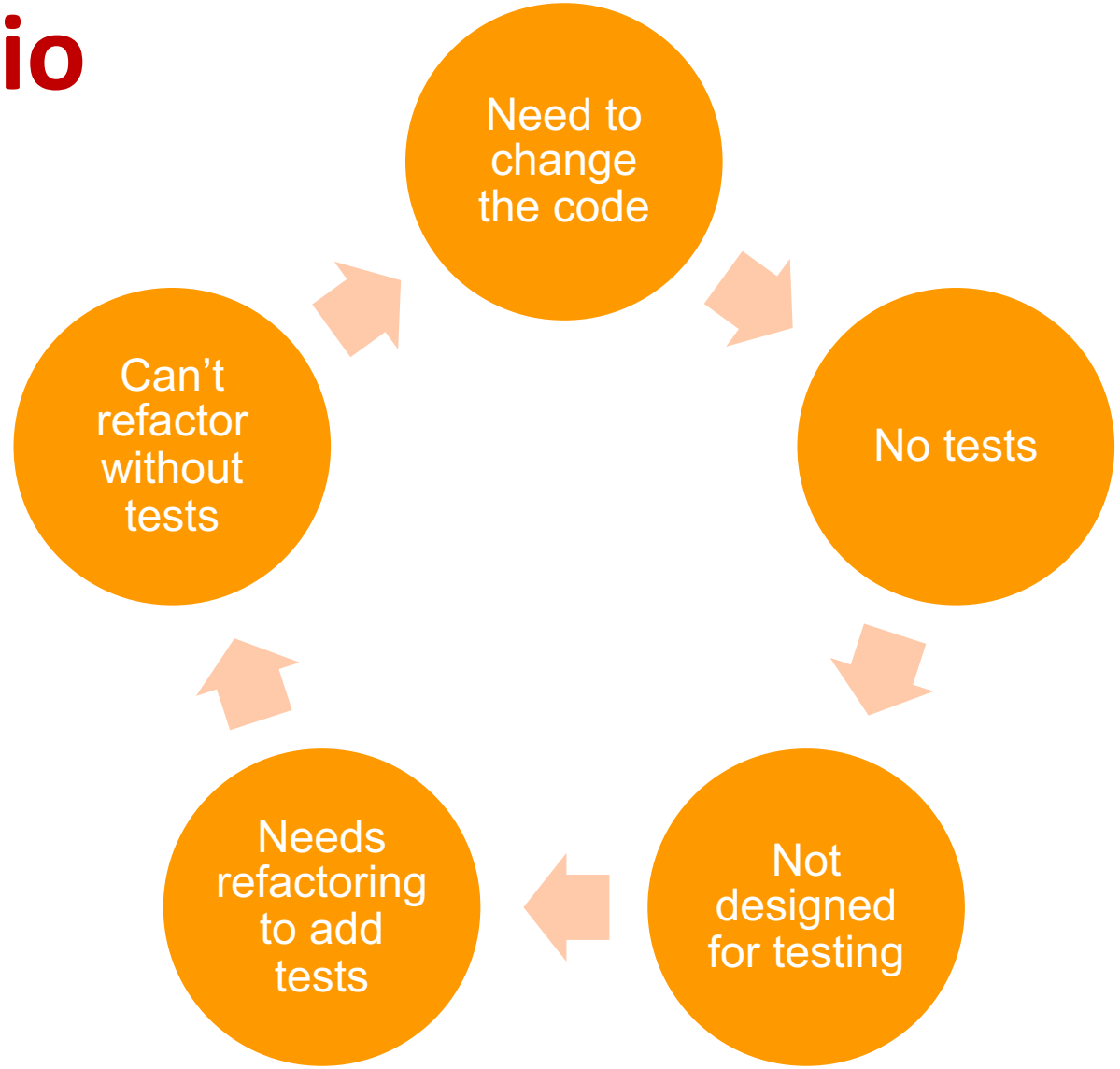
Quickly Testing **Legacy Code**

What does Legacy Code really mean?

- **Dictionary**
  - "**money or property that you receive from** someone after they die "
- **Michael Feathers**
  - "code **without unit tests**"
- **J.B. Rainsberger**
  - "**profitable code** that we feel afraid to change."
- **Kate Gregory**
  - "any code that you **want to change, but are afraid to**"

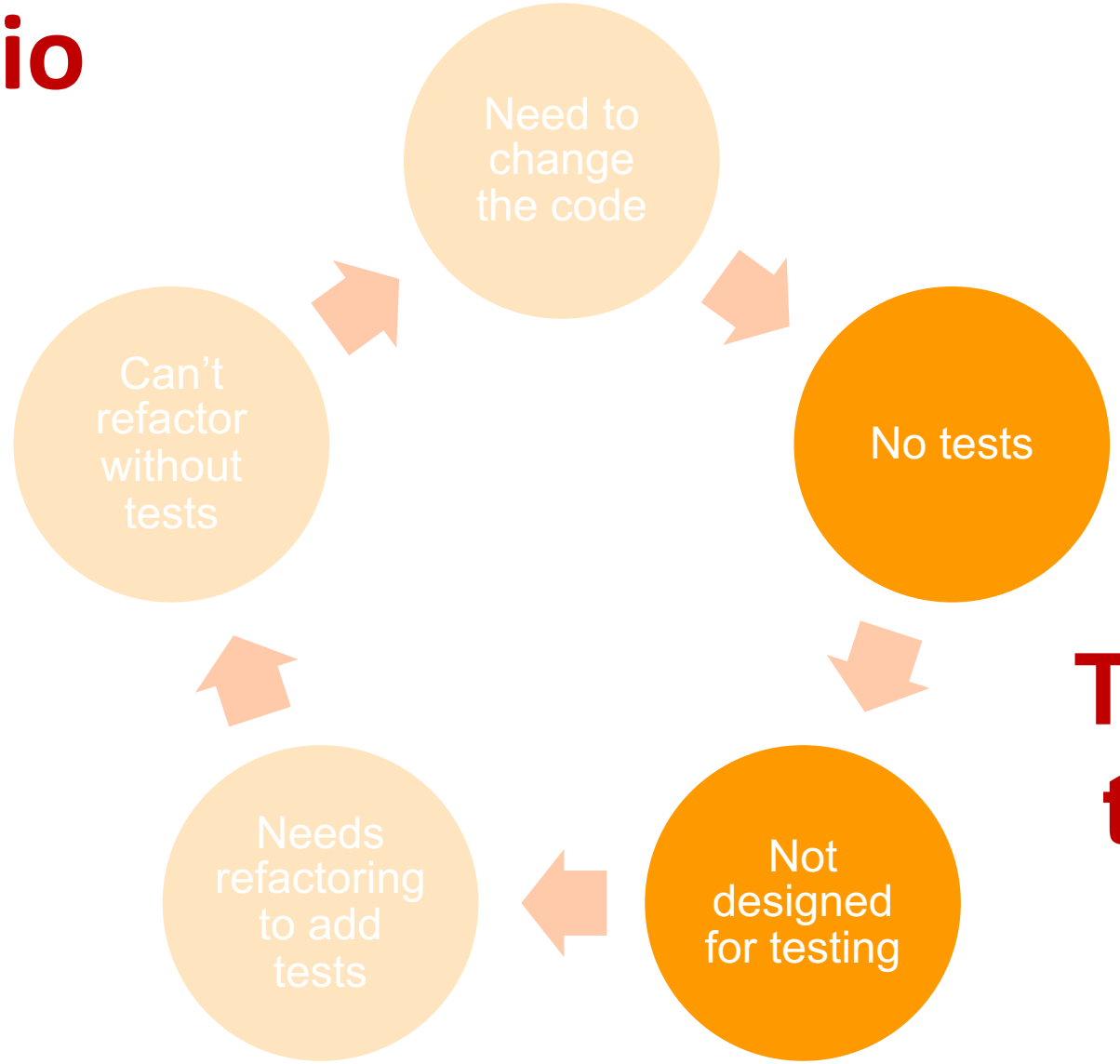
Typical Scenario

- I've inherited some legacy 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 legacy 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

Assumptions

- Value of automated testing
- Evaluate existing tests
  - Code coverage tools
  - Mutation testing
- No worries about types of tests
  - (unit, integration, regression)

Contents

- Introduction
- Legacy Code
- **Golden Master** 
- Approval Tests
- Example
- Resources
- Summary

Quickly **Testing** Legacy Code

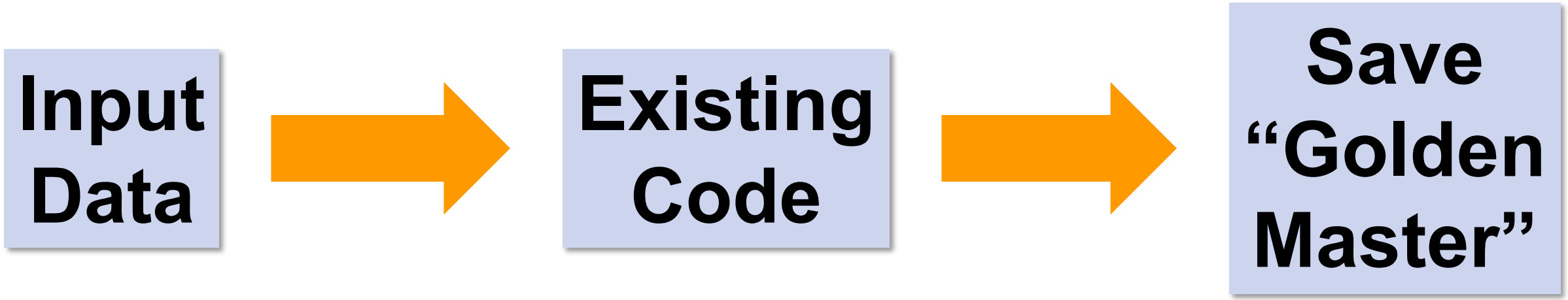
Key Phrase :
“Locking Down Current Behaviour”

Writing Unit Tests for Legacy Code

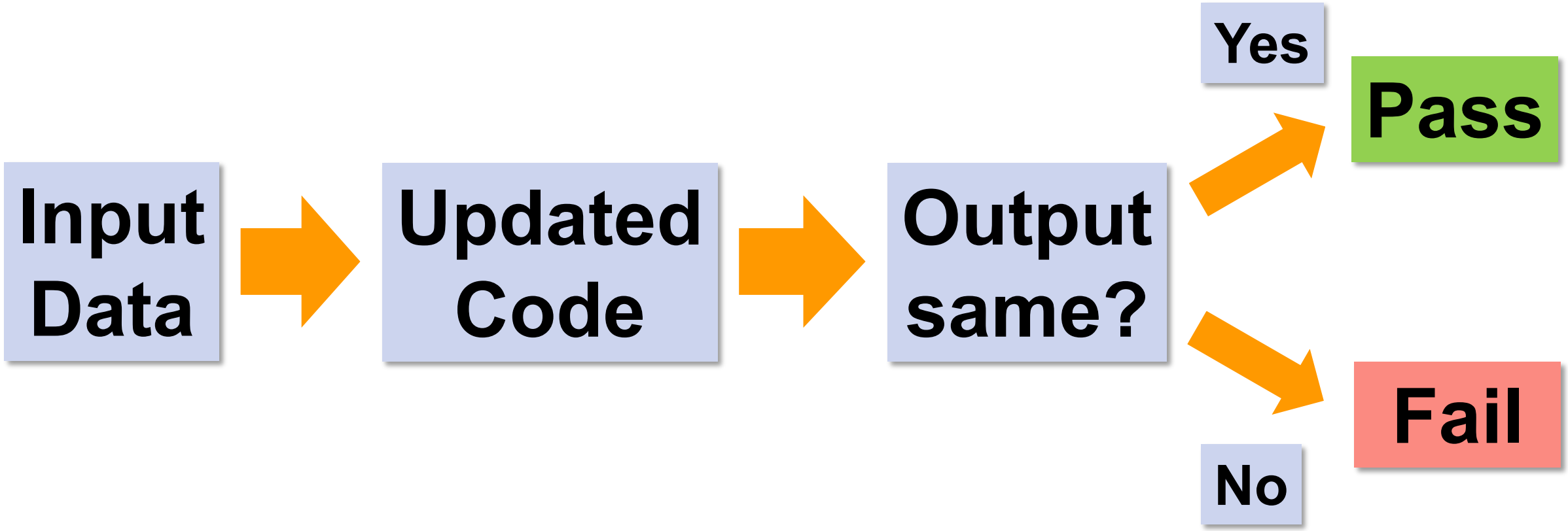
- Time-consuming
- What's intended behaviour?
- Is there another way?

Alternative: Golden Master Testing

Golden Master Test Setup



Golden Master Tests In Use



Thoughts on Golden Master Tests

- Good to start testing legacy systems
- Good when goal is to keep behaviour unchanged
- Depends on ease of:
  - Capturing output
  - Getting stable output
  - Reviewing any differences
  - Avoiding overwriting Golden Master by mistake!

Contents

- Introduction
- Legacy Code
- Golden Master
- **Approval Tests** 
- Example
- Resources
- Summary

Quickly Testing Legacy Code

One approach: “ApprovalTests”

- Llewellyn Falco’s convenient, powerful, flexible Golden Master implementation
- Supports many language

GO	.Net.ASP	Perl
Java	NodeJS	Python
Lua	Objective-C	Swift
.NET	PHP	

And now C++!

ApprovalTests.cpp – Approval Tests for C++

- New C++ library for applying Llewellyn Falco's "Approval Tests" approach
- For testing cross-platform C++ code (Windows, Linux, Mac)
- For legacy and green-field systems

ApprovalTests.cpp

- It’s on github
- Header-only
- Open source - Apache 2.0 licence

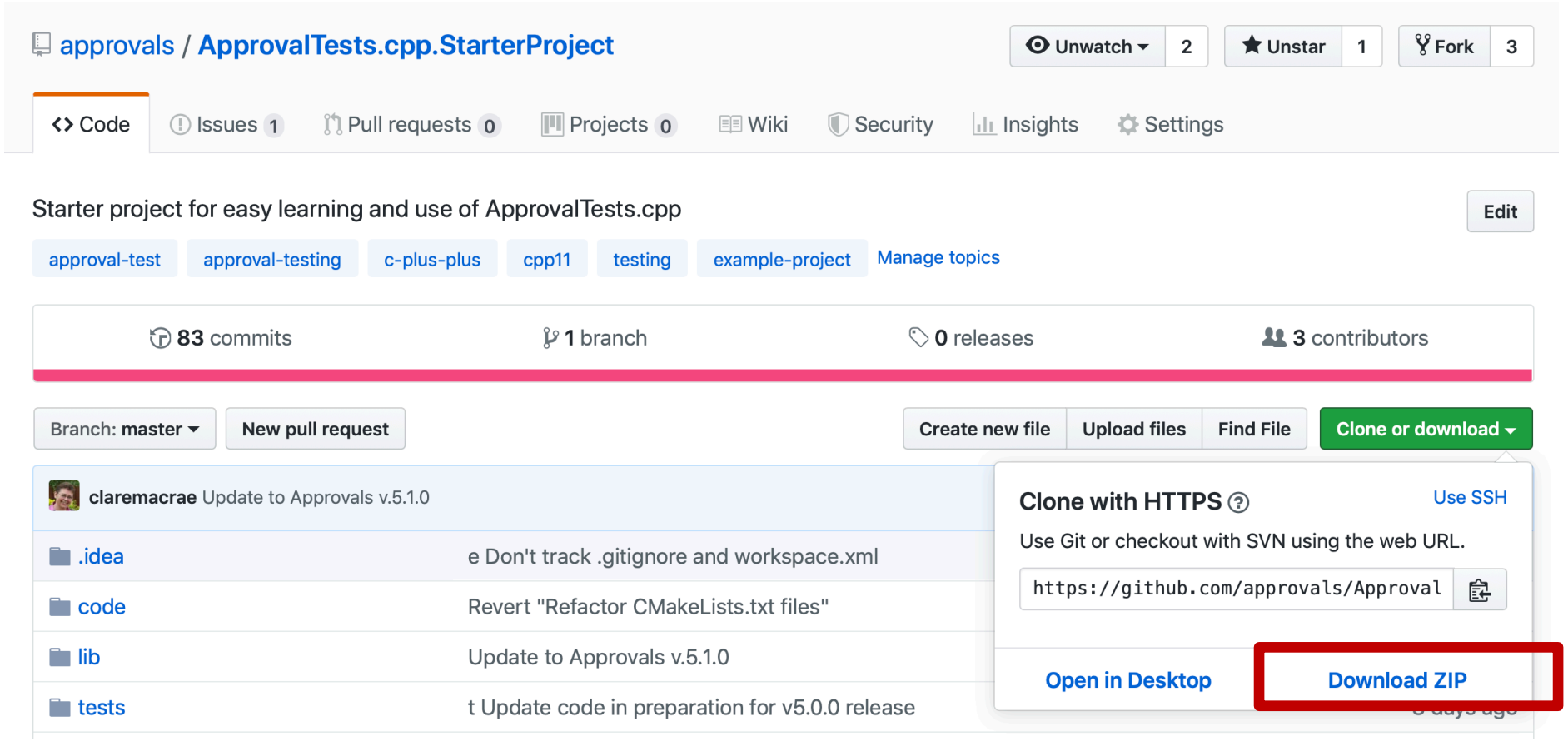
ApprovalTests.cpp

- C++11 and above
- Works with a range of testing frameworks
- Currently supports Google Test, Catch1, Catch2 and doctest
  - Easy to add more

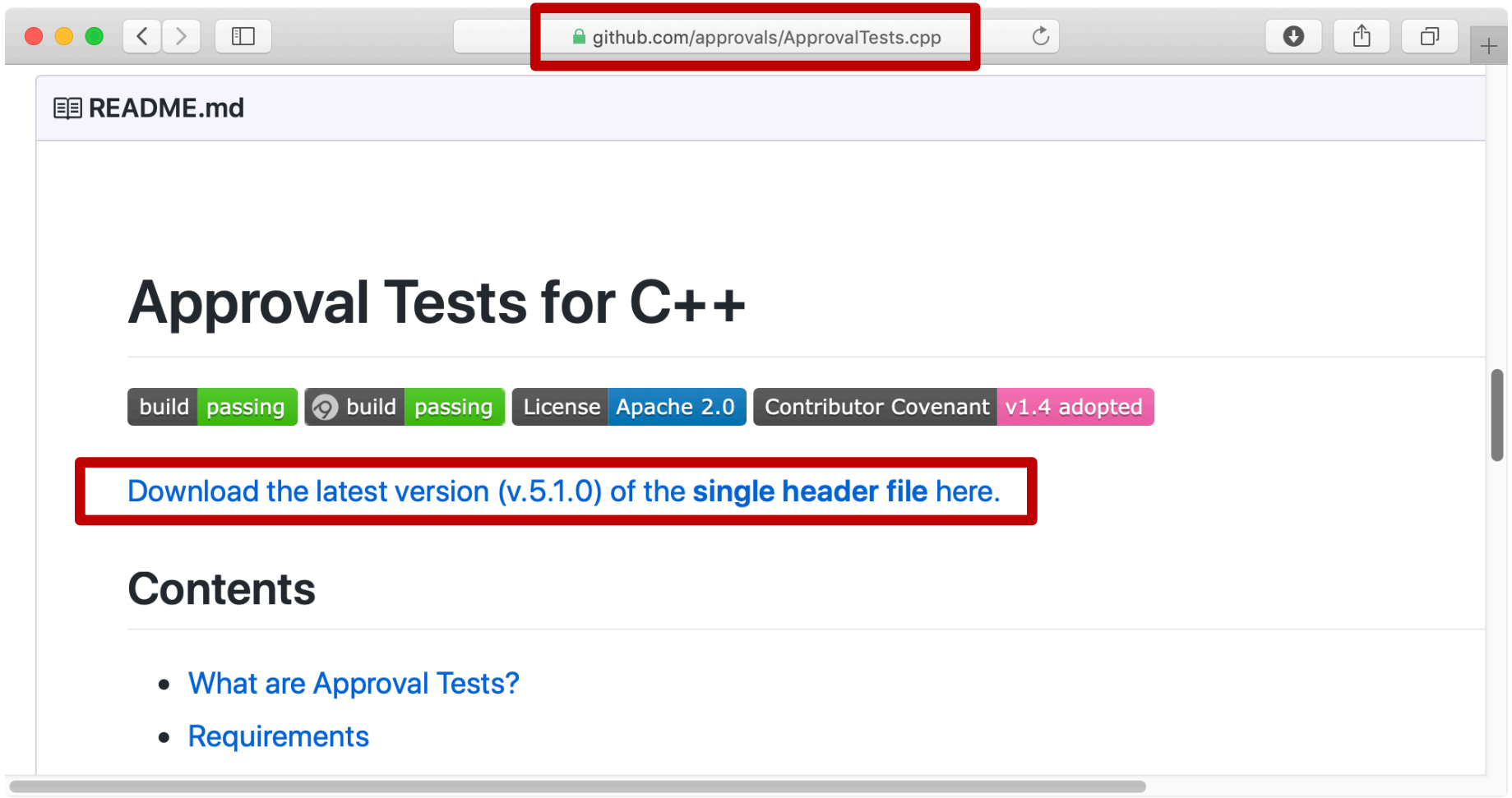
Getting Started with Approvals in C++

StarterProject with Catch2

- https://github.com/approvals/ApprovalTests.cpp.StarterProject
- Download ZIP



Download it Yourself…



Naming Options

- About that version number in the download (ApprovalTests.v.5.1.0.hpp)…

- Could just call it by your preferred convention:

```
ApprovalTests.h
```

```
ApprovalTests.hpp
```

- Or use a wrapper and keep the version number, e.g.:

```
// In ApprovalTests.hpp
#include "ApprovalTests.v.5.1.0.hpp"
```

Getting Started with Approvals in C++

Test Frameworks

- Using Approval Tests With Catch
- Using Approval Tests With Google Tests
- Using Approval Tests With Doctest
- Supporting a new test framework

How to use it: Catch2 Boilerplate

- Your main.cpp

```
// main.cpp:
#define APPROVALS_CATCH
#include "ApprovalTests.hpp"
```

How to use it: Catch2 Boilerplate

- Your main.cpp

```
// main.cpp:
#define APPROVALS_CATCH
#include "ApprovalTests.hpp"
```

How to use it: Catch2 Boilerplate

- Your main.cpp

```
// main.cpp:
#define APPROVALS_CATCH
#include "ApprovalTests.hpp"
```

Pure Catch2 Test

- A test file

```
#include "Catch.hpp"

// Catch-only test
TEST_CASE( "Sums are calculated" )
{
    REQUIRE( 1 + 1 == 2 );
    REQUIRE( 1 + 2 == 3 );
}
```

Pure Catch2 Test

- A test file

```
#include "Catch.hpp"

// Catch-only test
TEST_CASE( "Sums are calculated" )
{
    REQUIRE( 1 + 1 == 2 );
    REQUIRE( 1 + 2 == 3 );
}
```

Pure Catch2 Test

- A test file

```
#include "Catch.hpp"

// Catch-only test
TEST_CASE( "Sums are calculated" )
{
    REQUIRE( 1 + 1 == 2 );
    REQUIRE( 1 + 2 == 3 );
}
```

Approvals Catch2 Test

- A test file (Test02.cpp)

```
#include "ApprovalTests.hpp"
#include "Catch.hpp"

// Approvals test - test static value, for demo purposes
TEST_CASE( "TestFixedInput" )
{
    Approvals::verify("Some\nMulti-line\noutput");
}
```

Approvals Catch2 Test

- A test file (Test02.cpp)

```
#include "ApprovalTests.hpp"
#include "Catch.hpp"

// Approvals test - test static value, for demo purposes
TEST_CASE( "TestFixedInput" )
{
    Approvals::verify("Some\nMulti-line\noutput");
}
```

Approvals Catch2 Test

- A test file (Test02.cpp)

```
#include "ApprovalTests.hpp"
#include "Catch.hpp"

// Approvals test - test static value, for demo purposes
TEST_CASE( "TestFixedInput" )
{
    Approvals::verify("Some\nMulti-line\noutput");
}
```

Approvals Catch2 Test

- A test file (Test02.cpp)

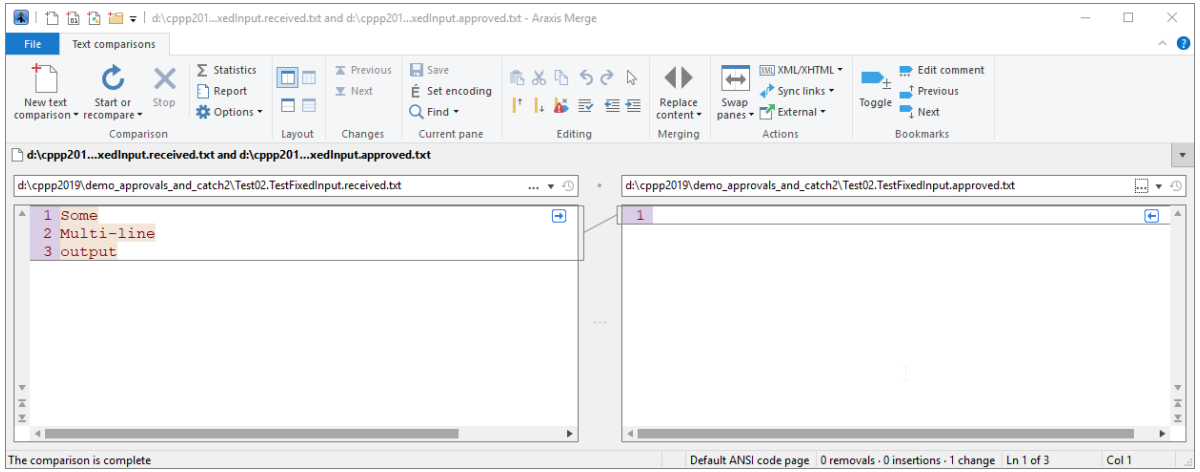
```
#include "ApprovalTests.hpp"
#include "Catch.hpp"

// Approvals test - test static value, for demo purposes
TEST_CASE( "TestFixedInput" )
{
    Approvals::verify("Some\nMulti-line\noutput");
}
```

First run

- 1. TestFixedInput failed

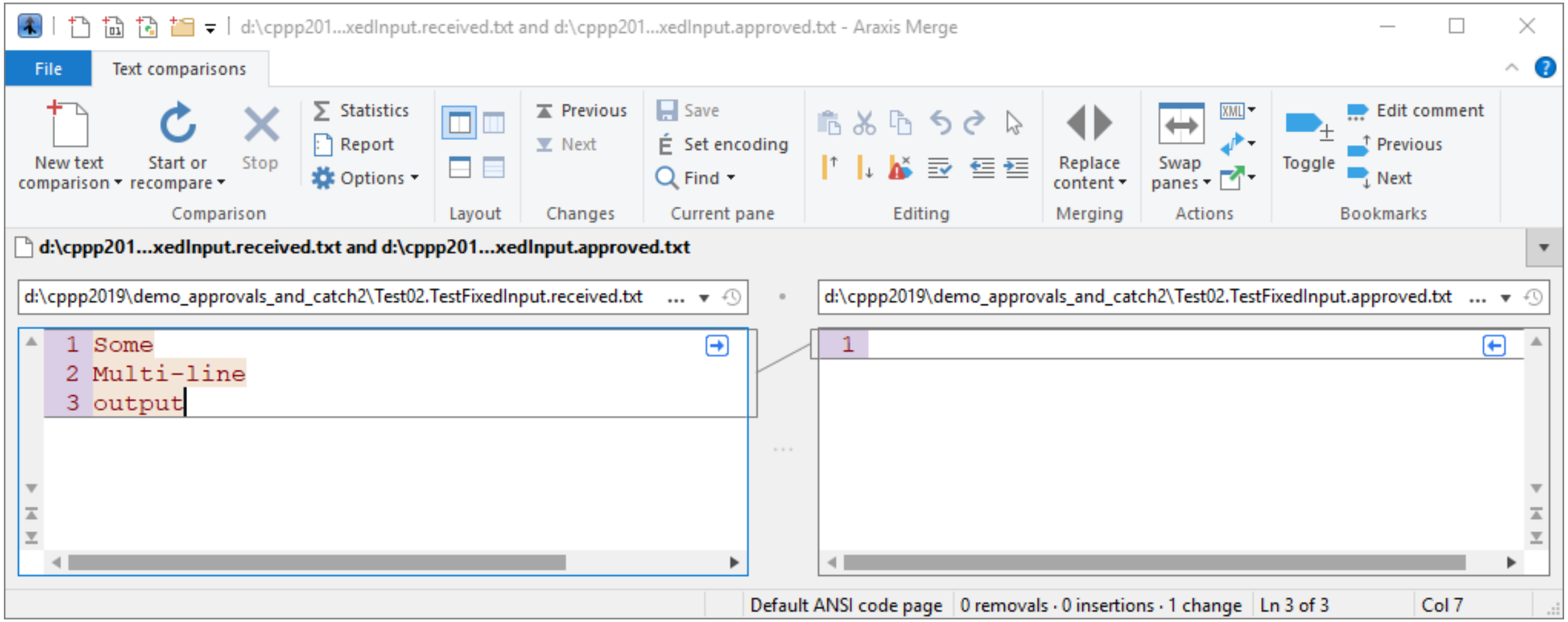
```
d:\cppp2019\demo_approvals_and_catch2\test02.cpp(5): exception: Failed Approval:
Approval File Not Found
File: "d:\cppp2019\demo_approvals_and_catch2\Test02.TestFixedInput.approved.txt"
```

- 2. Differencing tool pops up (Araxis Merge, in this case)



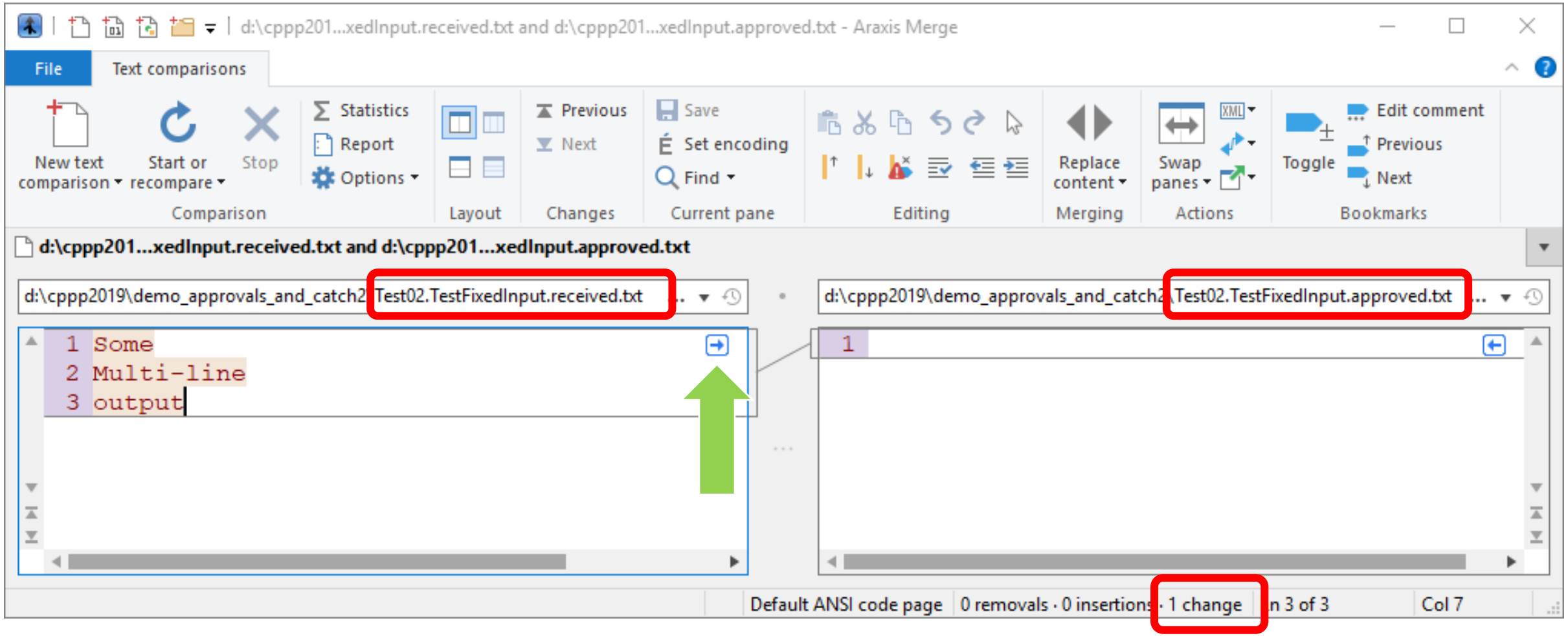
Actual/Received

Expected/Approved



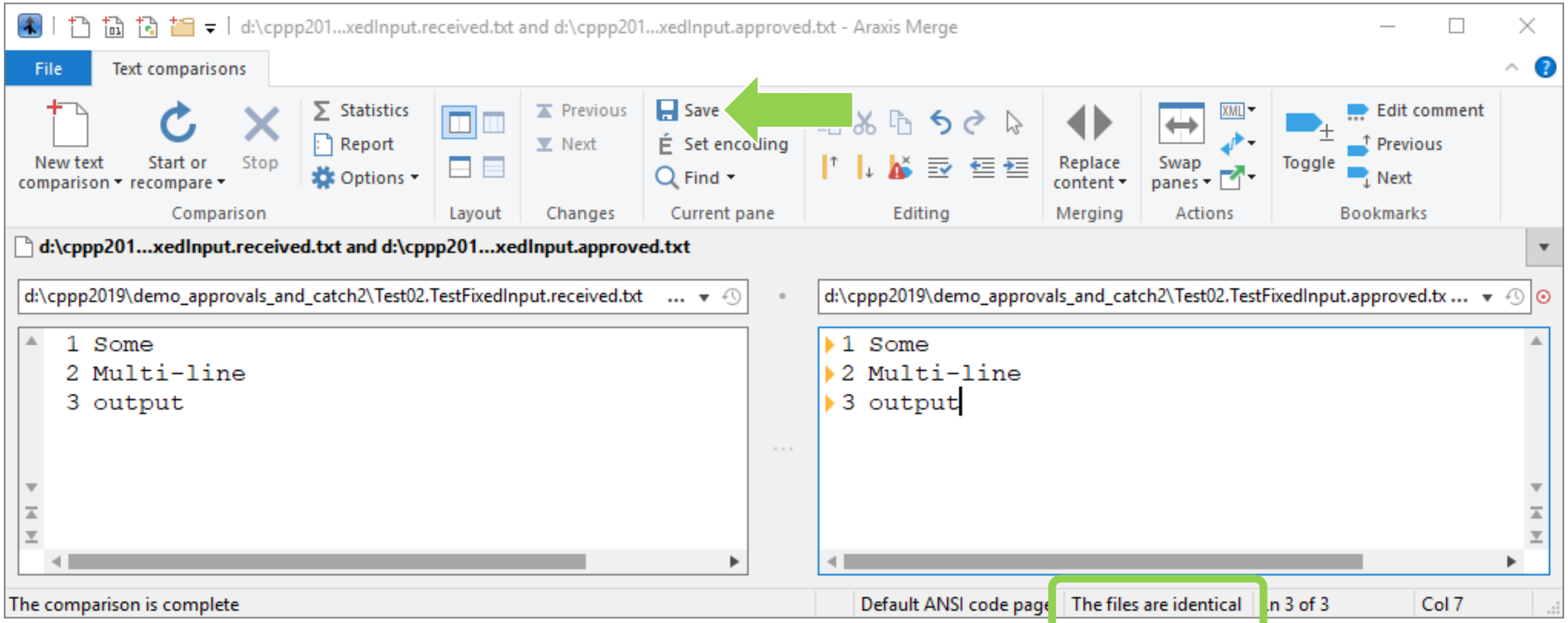
Actual/Received

Expected/Approved



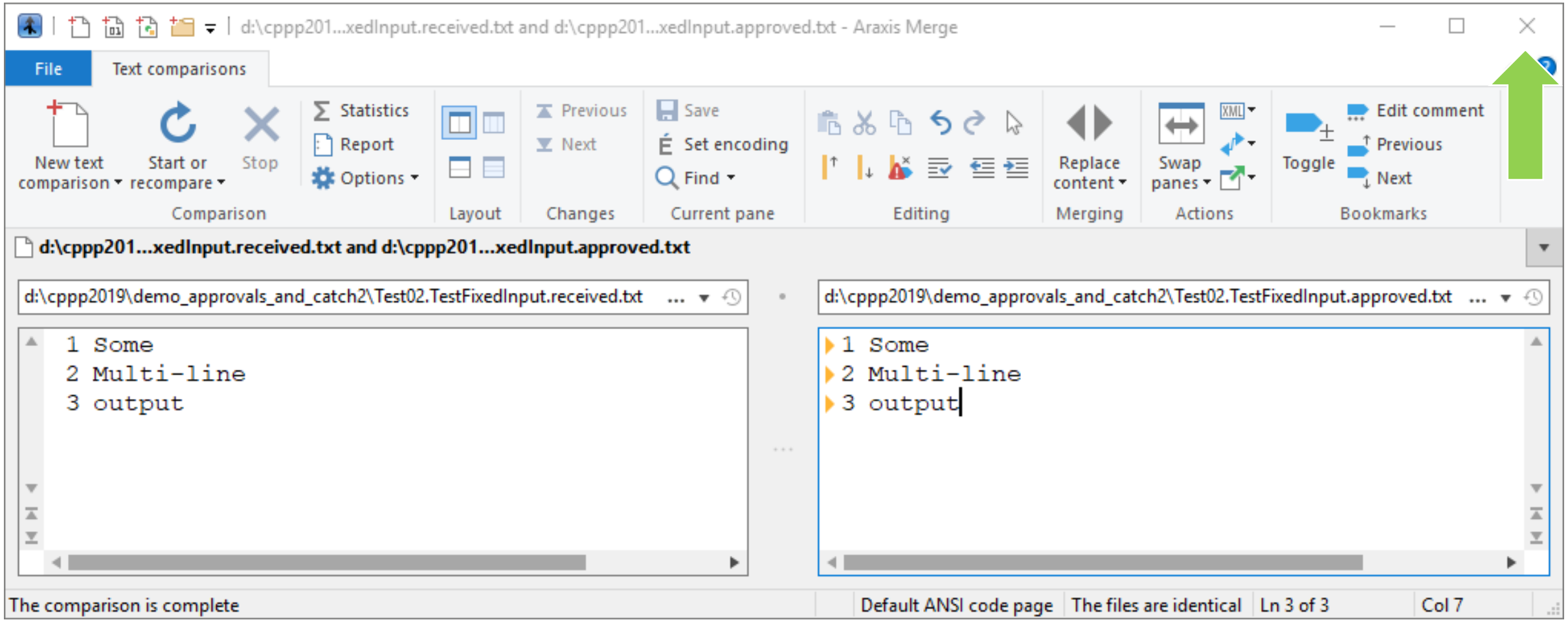
Actual/Received

Expected/Approved



Actual/Received

Expected/Approved



Second run

- I approved the output

```
TestFixedInput passed
```

- Then we commit the test, and the approved file(s) to version control
- The diffing tool only shows up if:
  - there is not (yet) an Approved file or
  - the Received differs from the Approved

What's going on here?

- It's a very convenient form of Golden Master testing
- Objects being tested are stringified
- Captures snapshot of current behaviour

Approval Tests *versus* Test Framework?

- Think of Approval Tests as an addition to your test framework
- … easy way to test larger, more complex things
- Useful for locking down behaviour of existing code
  - Combines with your testing framework
  - Not intended to replace Unit Tests

Reference: Variations on a Theme

Catch	doctest	Google Test
`#define APPROVALS_CATCH` `#include "ApprovalTests.hpp"`	`#define APPROVALS_DOCTEST` `#include "ApprovalTests.hpp"`	`#define APPROVALS_GOOGLETEST` `#include "ApprovalTests.hpp"`

`TEST_CASE( "Sums are calculated" )`
`{`
`  REQUIRE( 1 + 1 == 2 );`
`  REQUIRE( 1 + 2 == 3 );`
`}`
		`TEST( Test01, SumsAreCalculated )`
		`{`
		`  EXPECT_EQ( 1 + 1, 2 );`
		`  EXPECT_EQ( 1 + 2, 3 );`
		`}`

`TEST_CASE("TestFixedInput")`
`{`
`  Approvals::verify("Some\nMulti-line\noutput");`
`}`
		`TEST( Test02, TestFixedInput )`
		`{`
		`  Approvals::verify("Some\nMulti-line\noutput");`
		`}`

Test02.TestFixedInput.approved.txt		Test02.TestFixedInput.approved.txt

Example test directory

```
Test02.cpp
Test02.TestFixedInput.approved.txt
Test03CustomReporters.cpp
Test03CustomReporters.UseCustomReporter.approved.txt
Test03CustomReporters.UseQuietReporter.approved.txt
Test03CustomReporters.UseSpecificReporter.approved.txt
Test04ConsoleReporter.cpp
Test04ConsoleReporter.UseConsoleReporter.approved.txt
Test04ConsoleReporter.UseConsoleReporter.received.txt
```

How to use it: Catch2 Boilerplate

- Your main.cpp

```
#define APPROVALS_CATCH
#include "ApprovalTests.hpp"

// Put all approved and received files in "approval_tests/"
auto dir =
    Approvals::useApprovalsSubdirectory("approval_tests");
```

Example test directory

```
Test02.cpp
Test02.TestFixedInput.approved.txt
Test03CustomReporters.cpp
Test03CustomReporters.UseCustomReporter.approved.txt
Test03CustomReporters.UseQuietReporter.approved.txt
Test03CustomReporters.UseSpecificReporter.approved.txt
Test04ConsoleReporter.cpp
Test04ConsoleReporter.UseConsoleReporter.approved.txt
Test04ConsoleReporter.UseConsoleReporter.received.txt
```

Example test directory

```
Test02.cpp
Test03CustomReporters.cpp
Test04ConsoleReporter.cpp
```

```
approval_tests/
    Test02.TestFixedInput.approved.txt
    Test03CustomReporters.UseCustomReporter.approved.txt
    Test03CustomReporters.UseQuietReporter.approved.txt
    Test03CustomReporters.UseSpecificReporter.approved.txt
    Test04ConsoleReporter.UseConsoleReporter.approved.txt
    Test04ConsoleReporter.UseConsoleReporter.received.txt
```

Delving Deeper

To get the most out of ApprovalTests, start with the Tutorial. Once you're up and running, consider the following reference material.

Contents

https://github.com/approvals/ApprovalTests.cpp/blob/master/doc/README.md#top

Documentation

Writing the Test

Let's add our first test:

```
TEST_CASE("HelloApprovals")
{
    ApprovalTests::Approvals::verify("Hello Approvals");
}
```

snippet source / anchor

Source code

```
using namespace ApprovalTests;

16
17  // Maintainance note: keep the 'ApprovalTests::' - for the docs
18  // begin-snippet: hello_approvals
19  TEST_CASE("HelloApprovals")
20  {
21      ApprovalTests::Approvals::verify("Hello Approvals");
22  }
23  // end-snippet
24
```

How does this help with legacy code?

Applying this to legacy code

```
TEST_CASE("New test of legacy feature")
{
    // Standard pattern:
    // Wrap your legacy feature to test in a function call
    // that returns some state that can be written to a text file
    // for verification:
    const LegacyThing result = doLegacyOperation();
    Approvals::verify(result);
}
```

Implementation

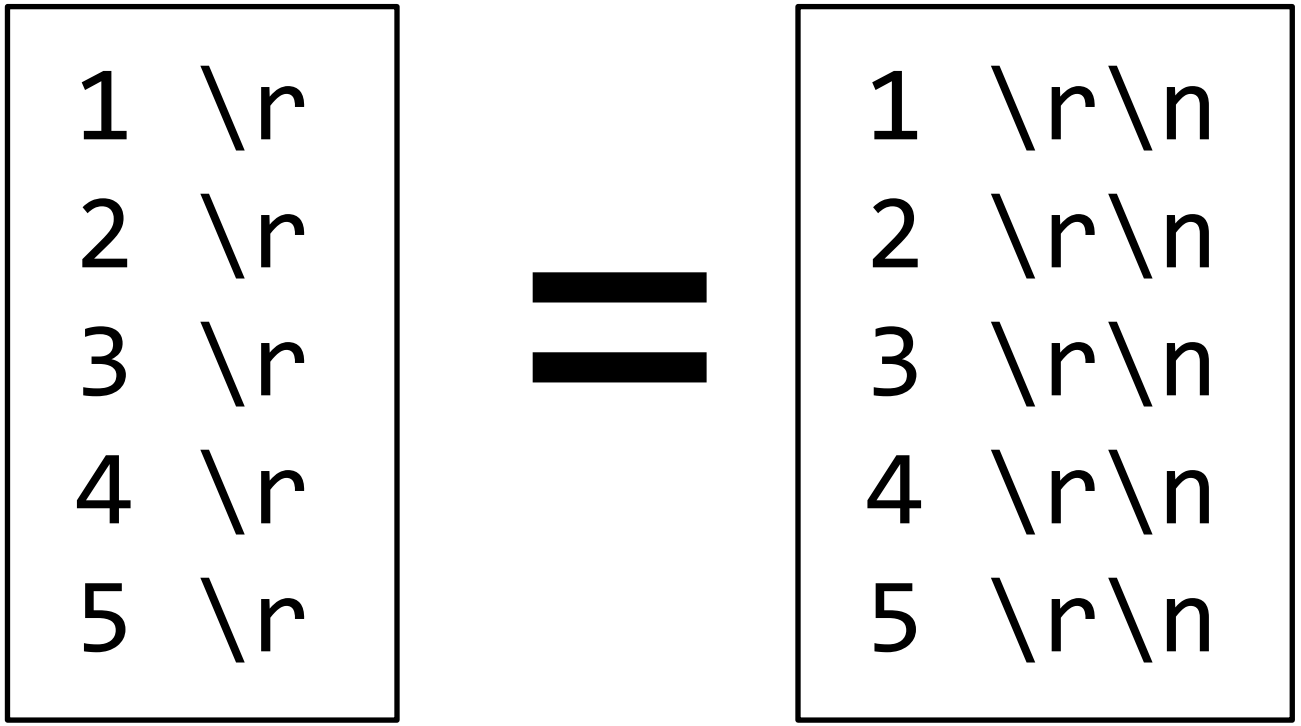
```
class LegacyThing;

std::ostream &operator<<(std::ostream &os, const LegacyThing &result) {
    // write interesting info from result here:
    os << result…;
    return os;
}
LegacyThing doLegacyOperation() {
    // your implementation here..
    return LegacyThing();
}
```

Feature: Consistency over machines

- Naming of output files
- Approved files version controlled
- **Caution**! Renaming **test files or test names** requires renaming **.approved files**

Feature: Consistency over Operating Systems

- Line-endings



Feature: Consistency over Languages

- Consistent concepts and nomenclature in all the implementations

.NET:

```
Approvals.Verify("should be approved");
```

C++:

```
Approvals::verify("should be approved");
```

Python:

```
verify("should be approved")
```

Feature: Quick to write tests

```
TEST_CASE("verifyAllWithHeaderBeginEndAndLambda")
{
  std::list<int> numbers{ 0, 1, 2, 3};
  // Multiple convenience overloads of Approvals::verifyAll()
  Approvals::verifyAll(
    "Test Squares", numbers.begin(), numbers.end(),
    [](int v, std::ostream& s) { s << v << " => " << v*v << '\n' ; });
}
```

Feature: Quick to write tests

```
TEST_CASE("verifyAllWithHeaderBeginEndAndLambda")
{
  std::list<int> numbers{ 0, 1, 2, 3};
  // Multiple convenience overloads of Approvals::verifyAll()
  Approvals::verifyAll(
    "Test Squares", numbers.begin(), numbers.end(),
    [](int v, std::ostream& s) { s << v << " => " << v*v << '\n' ; });
}
```

Feature: Quick to write tests

```
TEST_CASE("verifyAllWithHeaderBeginEndAndLambda")
{
  std::list<int> numbers{ 0, 1, 2, 3};
  // Multiple convenience overloads of Approvals::verifyAll()
  Approvals::verifyAll(
    "Test Squares", numbers.begin(), numbers.end(),
    [](int v, std::ostream& s) { s << v << " => " << v*v << '\n' ; });
}
```

All values in single output file:

Test Squares

0 => 0
1 => 1
2 => 4
3 => 9

Feature: Quick to get good coverage

```
TEST_CASE("verifyAllCombinationsWithLambda")
{
    std::vector<std::string> strings{"hello", "world"};
    std::vector<int> numbers{1, 2, 3};
    CombinationApprovals::verifyAllCombinations(
            // Lambda that acts on one combination of inputs, and returns the result to be approved:
            [](std::string s, int i) { return s + " " + std::to_string(i); },
            strings,                    // The first input container
            numbers);                   // The second input container
}
```

Feature: Quick to get good coverage

```
TEST_CASE("verifyAllCombinationsWithLambda")
{
    std::vector<std::string> strings{"hello", "world"};
    std::vector<int> numbers{1, 2, 3};
    CombinationApprovals::verifyAllCombinations(
            // Lambda that acts on one combination of inputs, and returns the result to be approved:
            [](std::string s, int i) { return s + " " + std::to_string(i); },
            strings,                    // The first input container
            numbers);                   // The second input container
}
```

Feature: Quick to get good coverage

```
TEST_CASE("verifyAllCombinationsWithLambda")
{
    std::vector<std::string> strings{"hello", "world"};
    std::vector<int> numbers{1, 2, 3};
    CombinationApprovals::verifyAllCombinations(
            // Lambda that acts on one combination of inputs, and returns the result to be approved:
            [](std::string s, int i) { return s + " " + std::to_string(i); },
            strings,                    // The first input container
            numbers);                   // The second input container
}
```

Feature: Quick to get good coverage

```
TEST_CASE("verifyAllCombinationsWithLambda")
{
    std::vector<std::string> strings{"hello", "world"};
    std::vector<int> numbers{1, 2, 3};
    CombinationApprovals::verifyAllCombinations(
            // Lambda that acts on one combination of inputs, and returns the result to be approved:
            [](std::string s, int i) { return s + " " + std::to_string(i); },
            strings,                    // The first input container
            numbers);                   // The second input container
}
```

All values in single output file:

```
(hello, 1) => hello 1
(hello, 2) => hello 2
(hello, 3) => hello 3
(world, 1) => world 1
(world, 2) => world 2
(world, 3) => world 3
```

Tip: To String docs

- https://github.com/approvals/ApprovalTests.cpp/blob/master/doc/ToString.md#top

Notice how this:

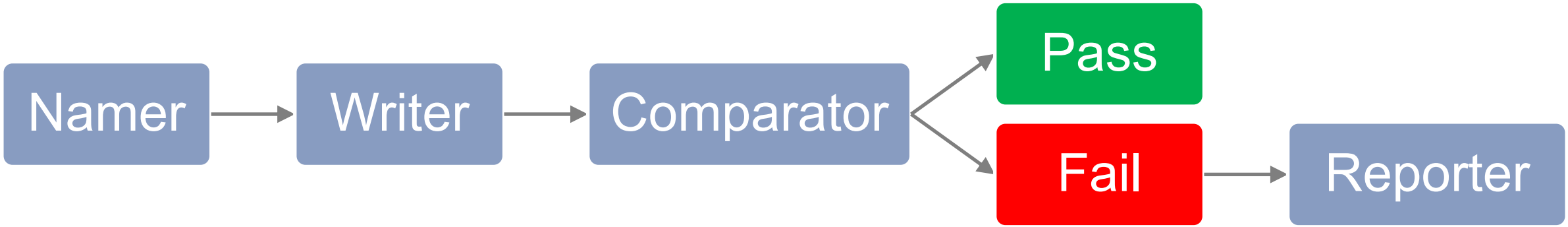
```
rectangles


[0] = [x: 4 y: 50 width: 100 height: 61]
[1] = [x: 50 y: 5200 width: 400 height: 62]
[2] = [x: 60 y: 3 width: 7 height: 63]
```

compares to this:

```
rectangles


(x,y,width,height) = (4,50,100,61)
(x,y,width,height) = (50,5200,400,62)
(x,y,width,height) = (60,3,7,63)
```

Customisability

Stages



Customisability: Reporters

Customisability: Reporters

- Reporters act on test failure
- Very simple but powerful abstraction
- Gives complete user control over how to inspect and act on a failure
  - If you adopt Approvals, do review the supplied reporters for ideas

“Disposable Objects” (RAII)

```
TEST_CASE("DisposableExplanation")
{
    {
        auto disposer = Approvals::useAsDefaultReporter(
                std::make_shared<Mac::BeyondCompareReporter>());
        // Your tests here will use Mac::BeyondCompareReporter...


    } // as soon as the code passes this }, the disposer is destroyed
      // and Approvals reinstates the previous default reporter
}
```

“Disposable Objects” (RAII)

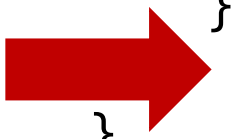
```
TEST_CASE("DisposableExplanation")
{
    {
        auto disposer = Approvals::useAsDefaultReporter(
                std::make_shared<Mac::BeyondCompareReporter>());
        // Your tests here will use Mac::BeyondCompareReporter...


    } // as soon as the code passes this }, the disposer is destroyed
      // and Approvals reinstates the previous default reporter
}
```

“Disposable Objects” (RAII)

```
TEST_CASE("DisposableExplanation")
{
    {
        auto disposer = Approvals::useAsDefaultReporter(
                std::make_shared<Mac::BeyondCompareReporter>());
        // Your tests here will use Mac::BeyondCompareReporter...


    } // as soon as the code passes this }, the disposer is destroyed
      // and Approvals reinstates the previous default reporter
}
```

“Disposable Objects” (RAII)

```
TEST_CASE("DisposableExplanation")
{
    {
        auto disposer = Approvals::useAsDefaultReporter(
                std::make_shared<Mac::BeyondCompareReporter>());
        // Your tests here will use Mac::BeyondCompareReporter...


    } // as soon as the code passes this }, the disposer is destroyed
      // and Approvals reinstates the previous default reporter
}
```



“Disposable Objects”

- Hold on to the object returned by a customization ([[nodiscard]])
- Most customizations are reversible
- Gives good, fine-grained control

Feature: Change the default Reporter

```
// main.cpp - or in individual tests:
#include <memory>

auto disposer = Approvals::useAsDefaultReporter(
    std::make_shared<Windows::AraxisMergeReporter>() );

auto disposer = Approvals::useAsDefaultReporter(
    std::make_shared<GenericDefaultReporter>(
        "E:/Program Files/Araxis/Araxis Merge/Compare.exe") );
```

Feature: Change the default Reporter

```
// main.cpp - or in individual tests:
#include <memory>

auto disposer = Approvals::useAsDefaultReporter(
    std::make_shared<Windows::AraxisMergeReporter>() );

auto disposer = Approvals::useAsDefaultReporter(
    std::make_shared<GenericDefaultReporter>(
        "E:/Program Files/Araxis/Araxis Merge/Compare.exe") );
```

Feature: Change the default Reporter

```
// main.cpp - or in individual tests:
#include <memory>

auto disposer = Approvals::useAsDefaultReporter(
    std::make_shared<Windows::AraxisMergeReporter>() );

auto disposer = Approvals::useAsDefaultReporter(
    std::make_shared<GenericDefaultReporter>(
        "E:/Program Files/Araxis/Araxis Merge/Compare.exe") );
```

Feature: Change the Reporter in one test

```
TEST_CASE("Demo custom reporter")
{
    std::vector<std::string> v{"hello", "world"};
    Approvals::verifyAll(v, MyCustomReporter{});
}
```

Feature: Doesn’t run GUIs on build servers

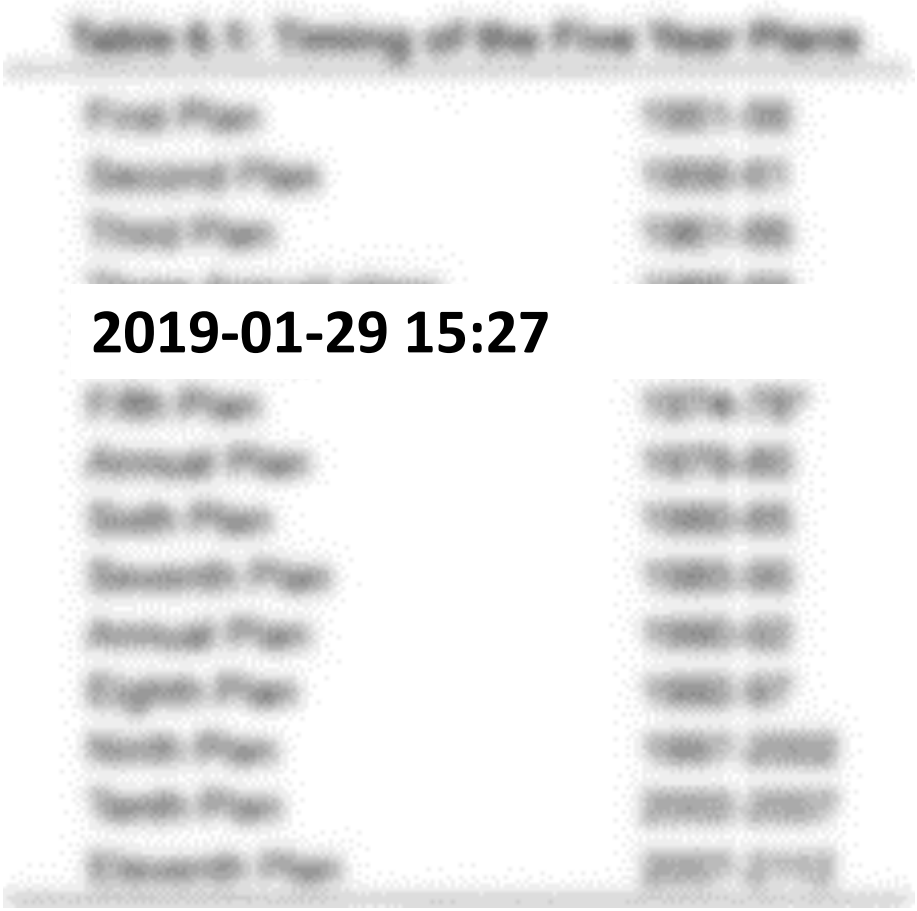
- Since v5.1.0
- Any failing tests will still fail
- No diff tool pops up
- Concept: “Front loaded reporters”

Feature: Convention over Configuration

- Fewer decisions for developers
- Users only specify unusual behaviours

Challenge: Golden Master is a log file

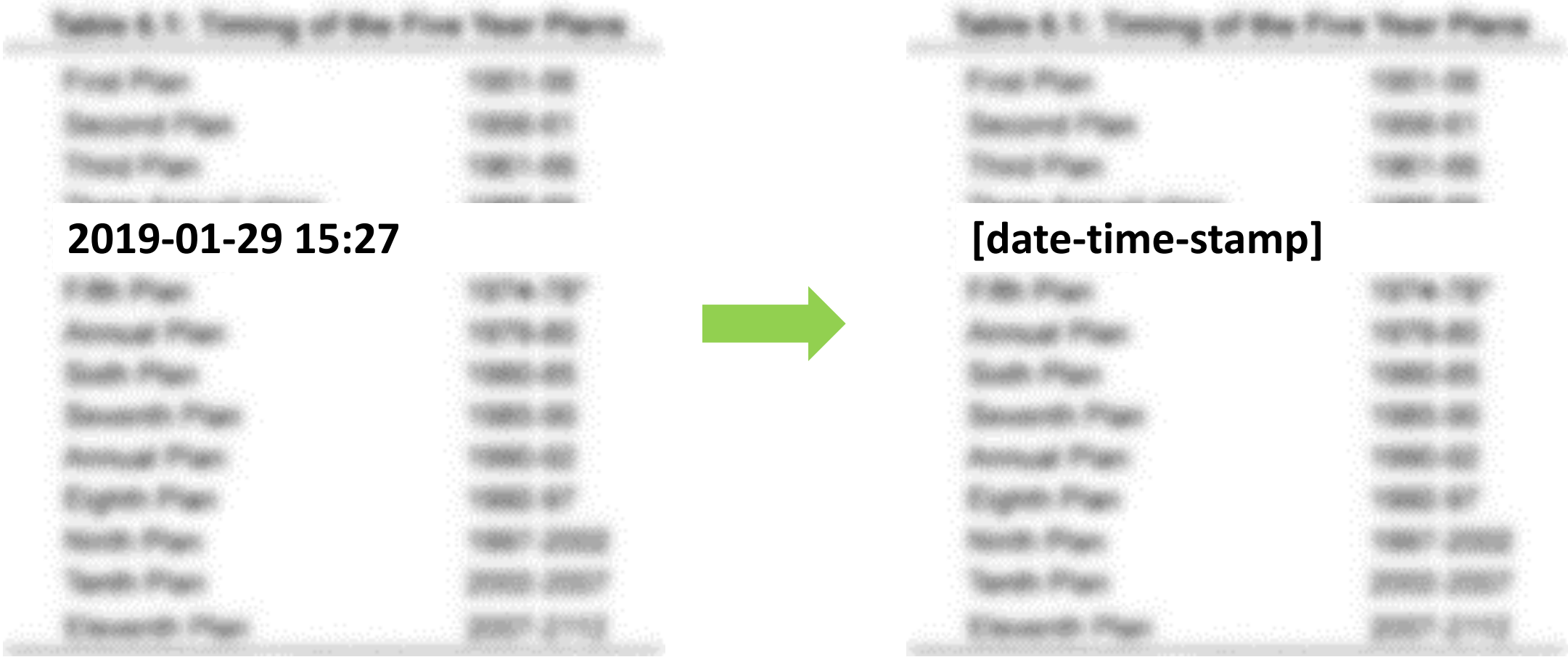
- Dates and times?
- Object addresses?



Options for unstable output

- Introduce date-time abstraction?
- Customised comparison function?
- Or: strip dates from the log file

Tip: Rewrite output file



Customisability: ApprovalWriter interface

```
class ApprovalWriter
{
public:
    virtual std::string getFileExtensionWithDot() = 0;
    virtual void write(std::string path) = 0;
    virtual void cleanUpReceived(std::string receivedPath) = 0;
};
```

Challenge: Multiple output files per test

- Naming convention generates one file name per test
- Multiple verify calls in one test case just overwrite same file
- Multiple options provided for this:

- https://github.com/approvals/ApprovalTests.cpp/blob/master/doc/MultipleOutputFilesPerTest.md

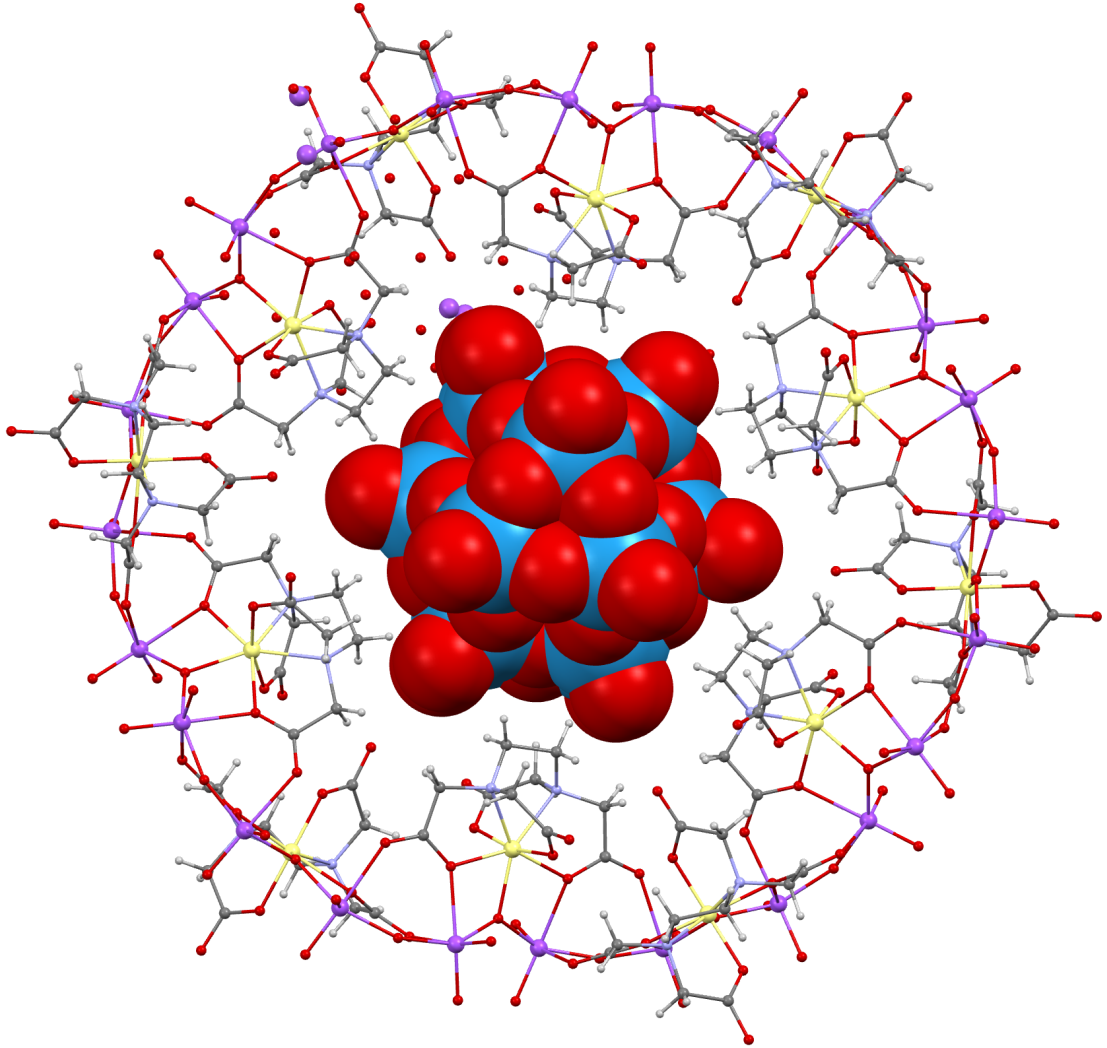
Flexibility gives non-obvious power

- Approval Tests approach is deceptively simple
- Reporter could:
  - Convert numbers to picture for comparison
  - Or Excel file
- ApprovalWriter could:
  - Write out a hash value of contents of very large file

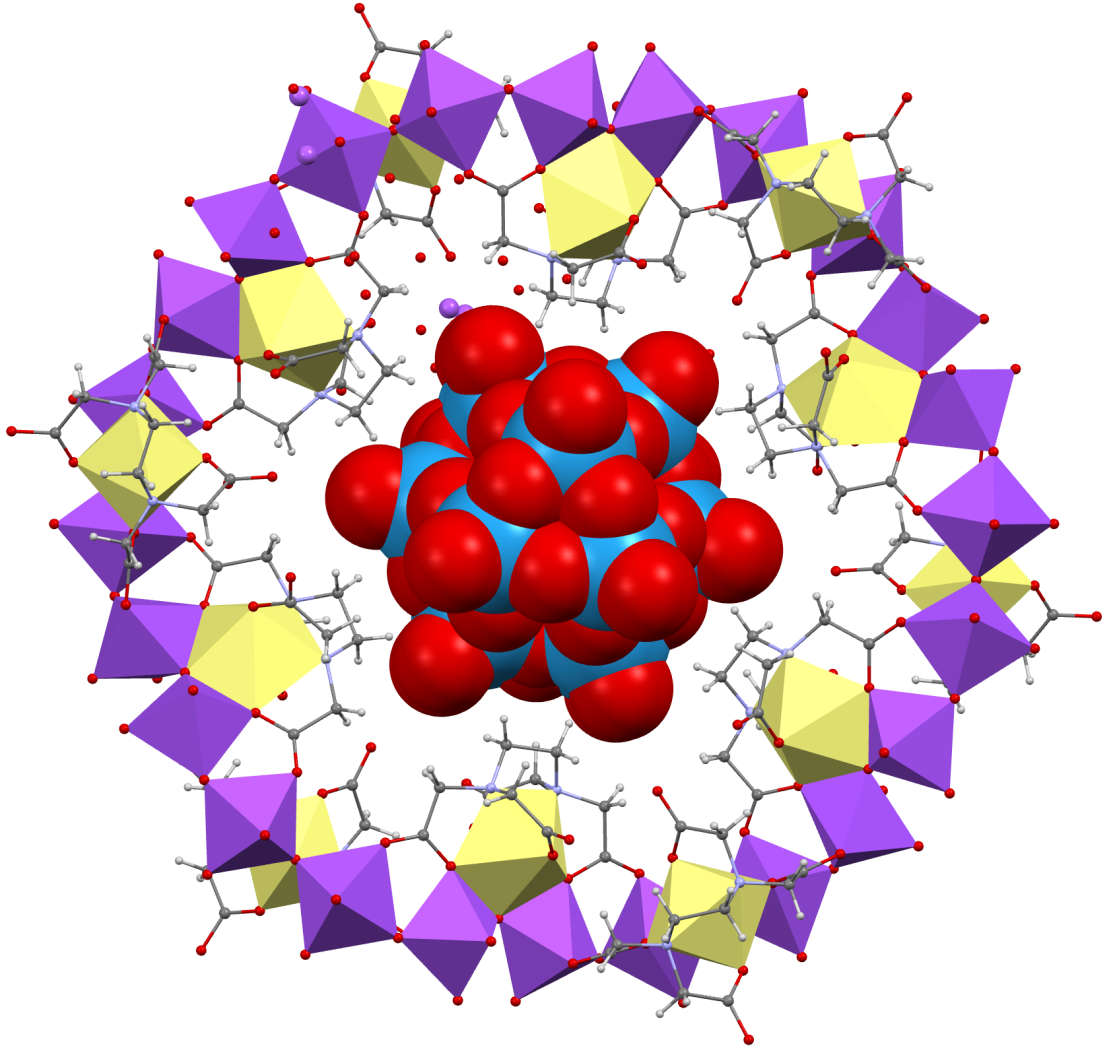
Contents

- Introduction
- Legacy Code
- Golden Master
- Approval Tests
- **Example**
- Resources
- Summary

What I could do:



What I needed to do:



Approving Images

What I'm aiming for

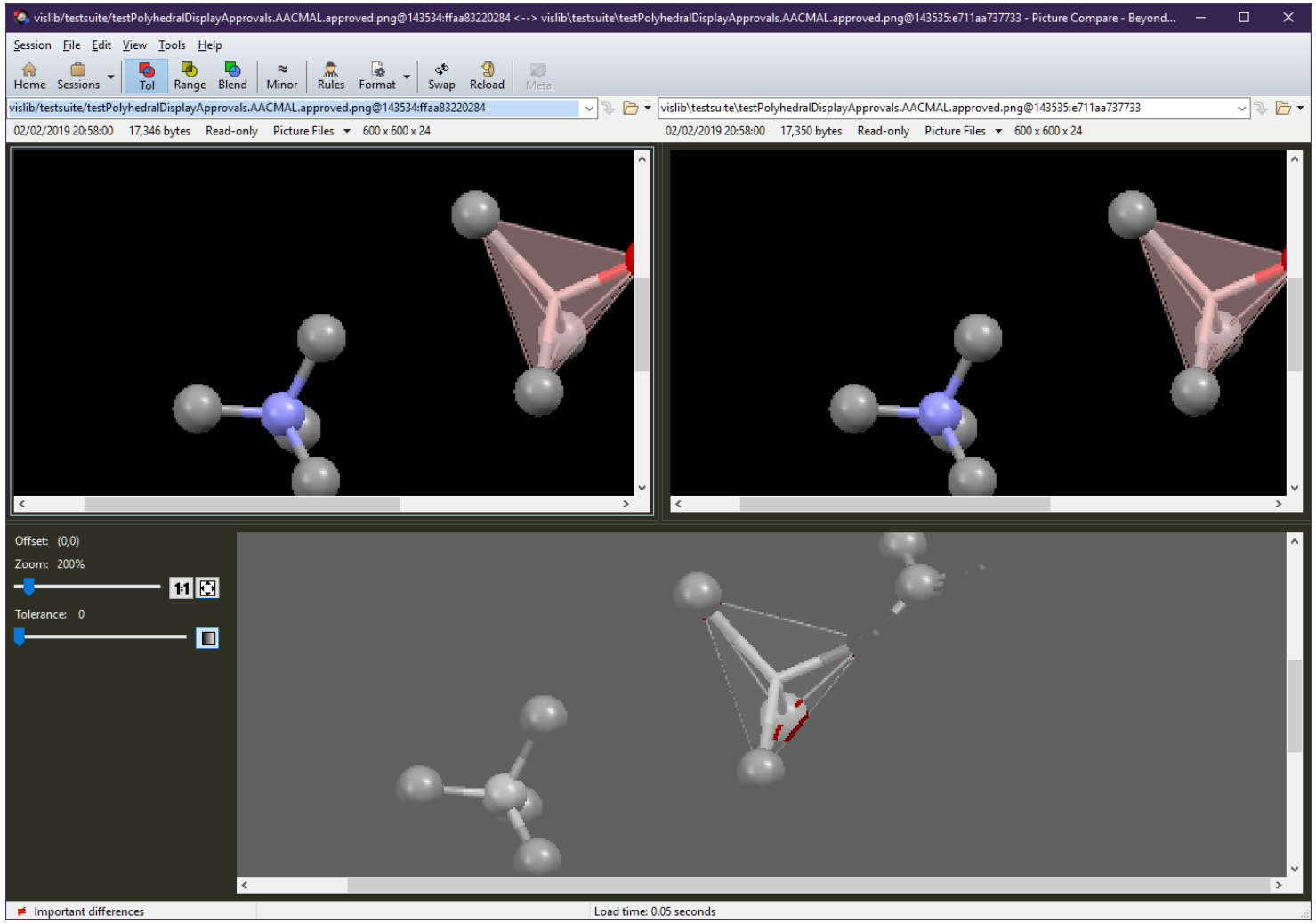
```
TEST(PolyhedralGraphicsStyleApprovals, CUVXAT)
{
    // CUVXAT identifies a crystal structure in the database
    const QImage crystal_picture = loadEntry("CUVXAT");
    verifyQImage(crystal_picture);
}
```

Idiomatic verification of Qt image objects

```
void verifyQImage(
    QImage image, const Reporter& reporter = DefaultReporter())
{
    QImageWriter writer(image); // implements ApprovalWriter
    Approvals::verify(imageWriter, reporter);
}
```

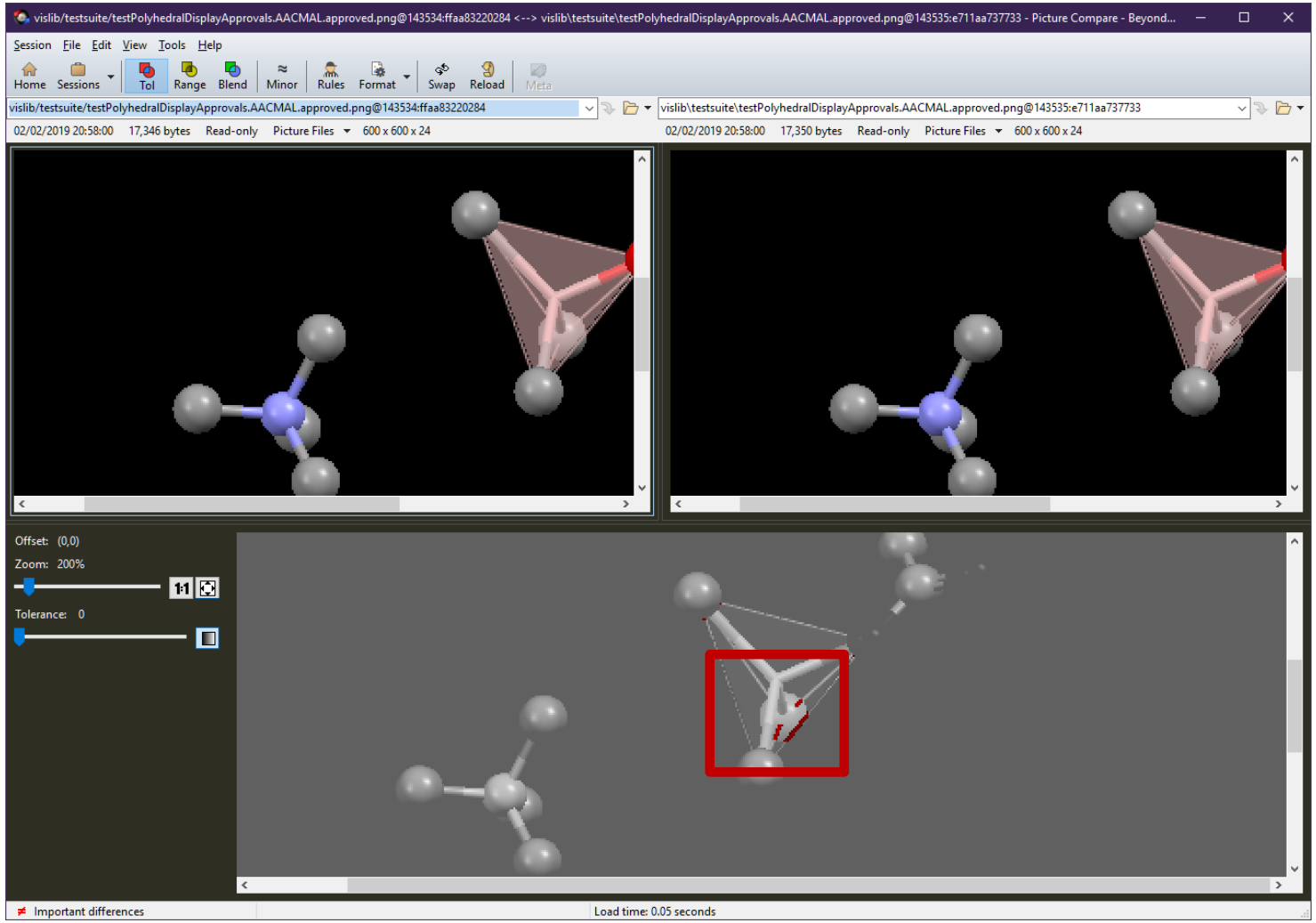
Useful feedback

BeyondCompare 4



Useful feedback

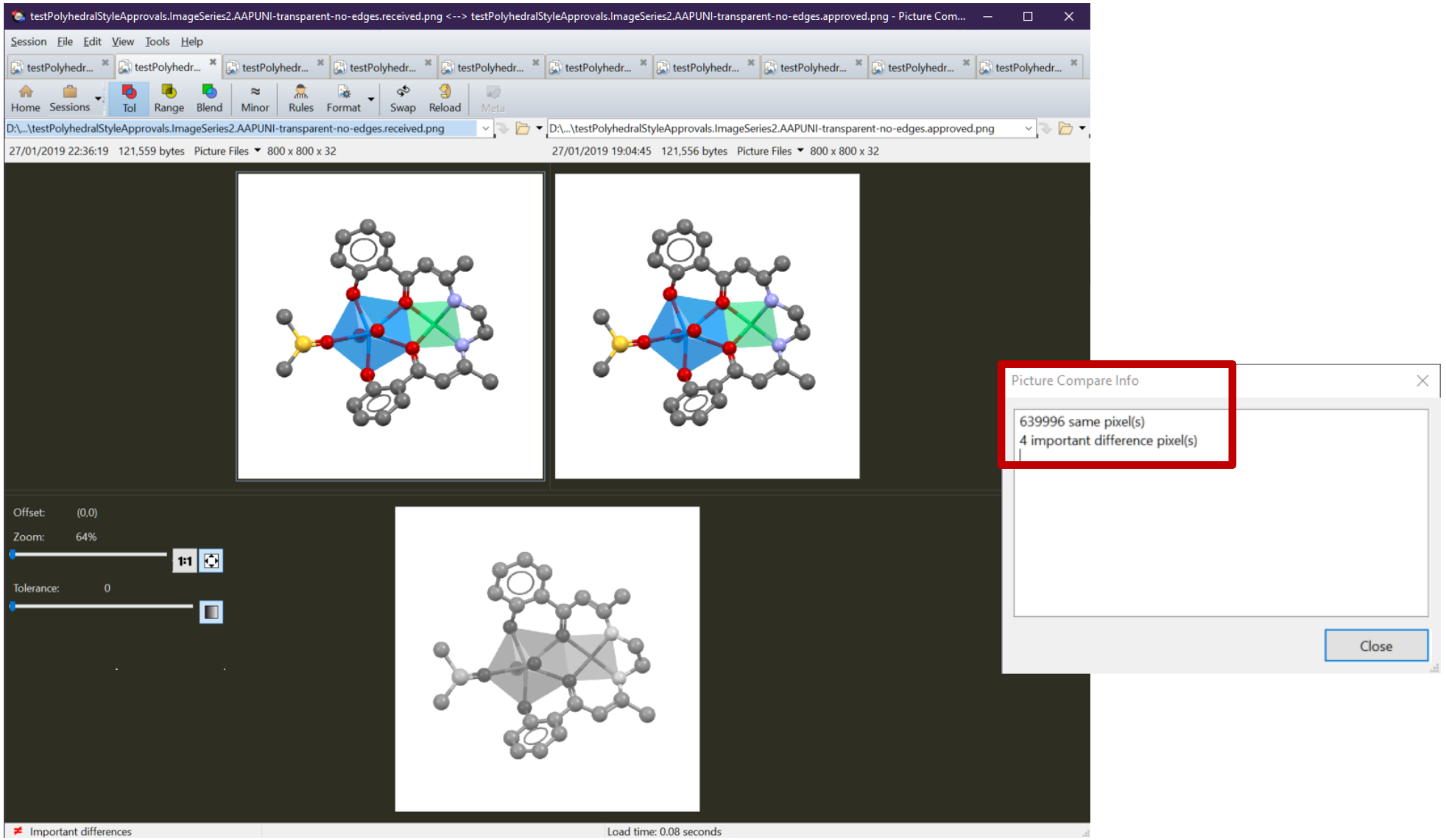
BeyondCompare 4

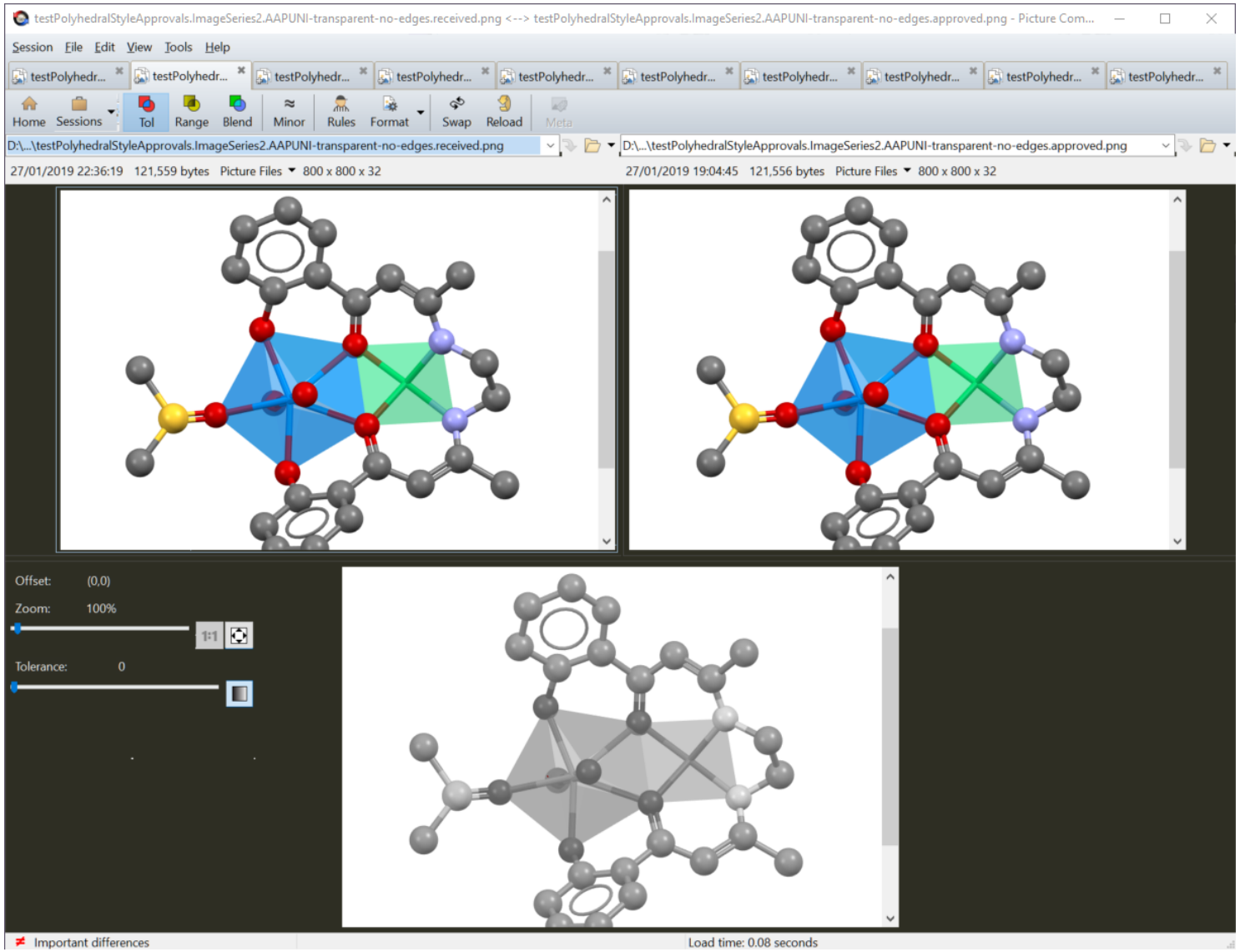


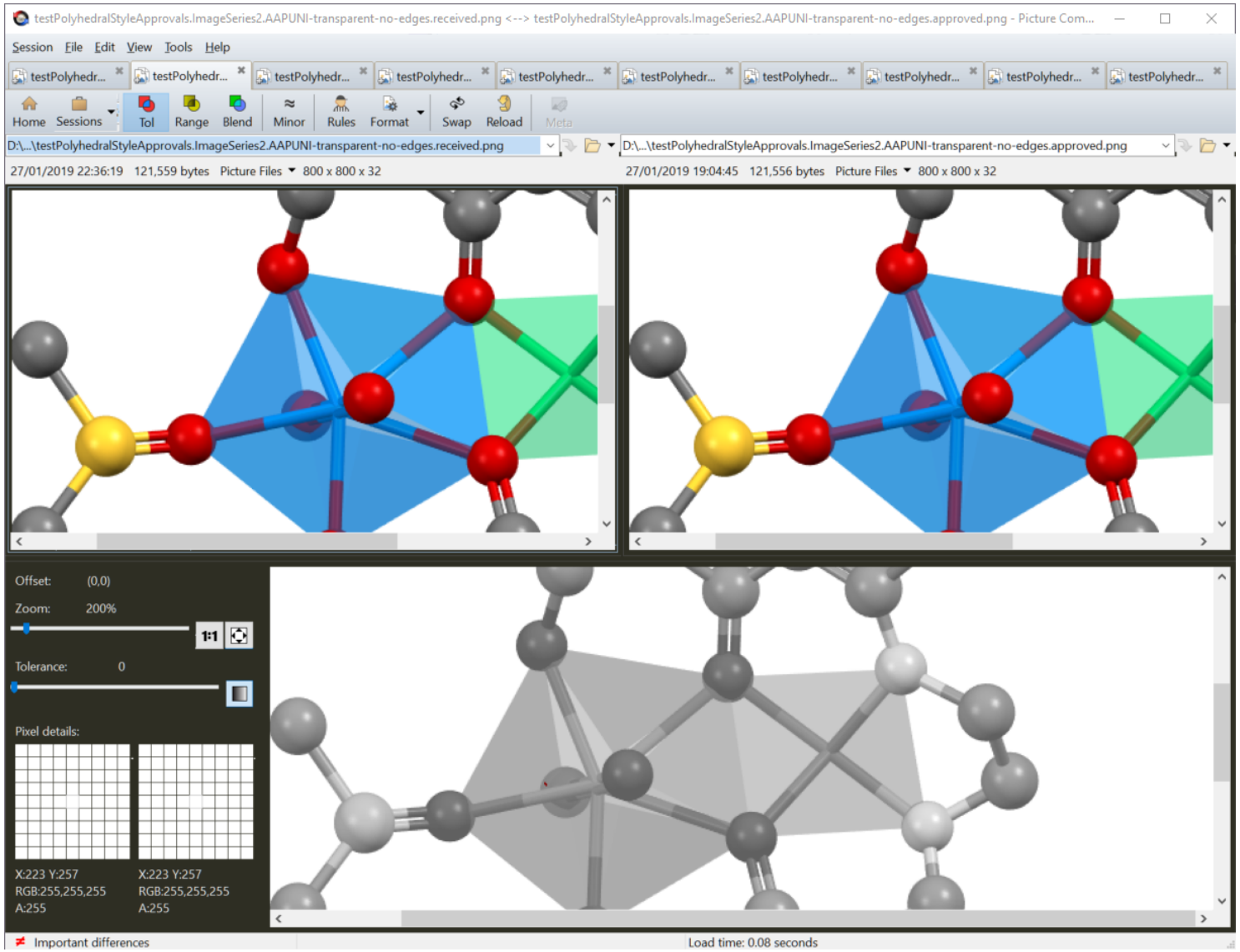
Back In work…

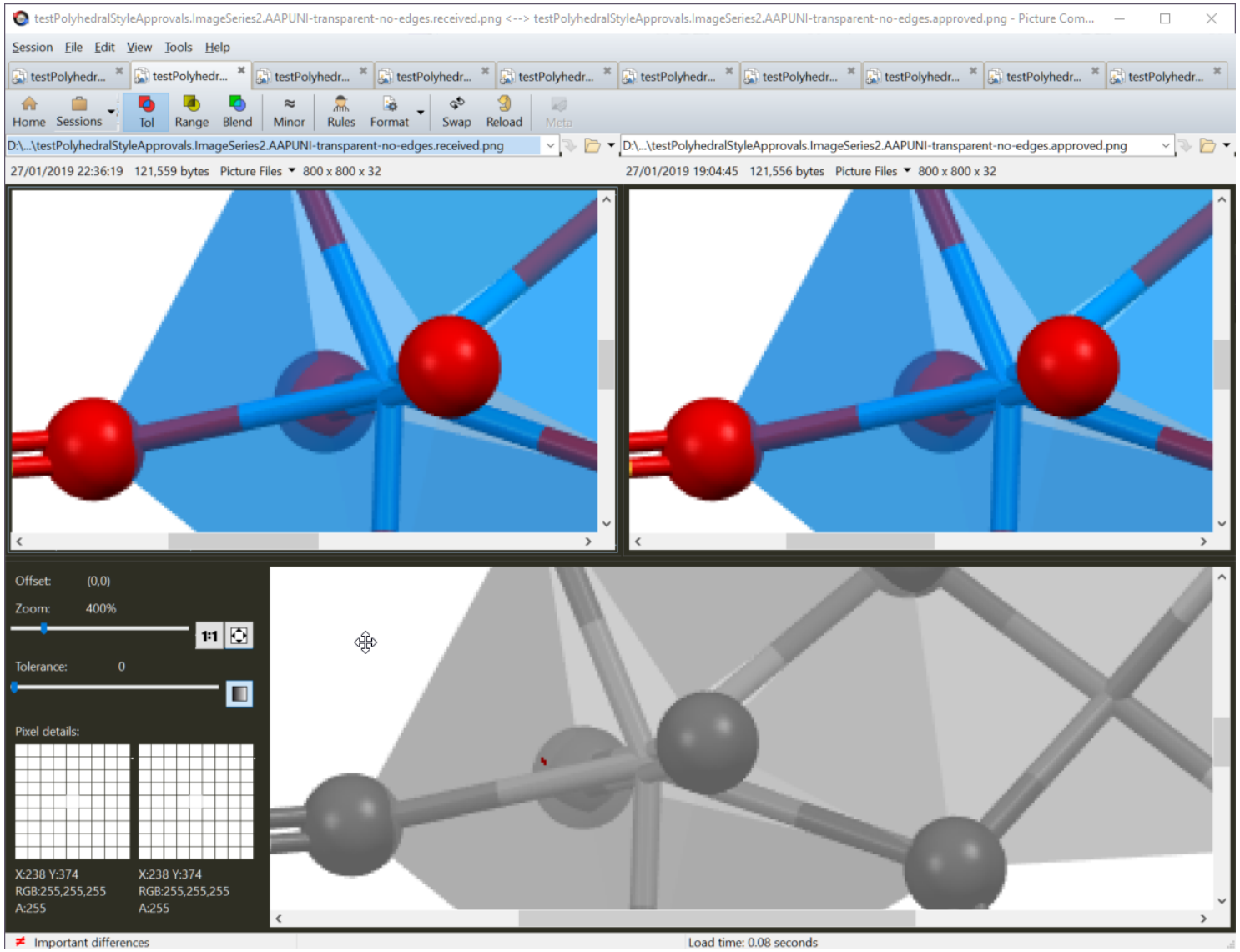
- Previously, working from home via Remote Desktop
- The tests started failing randomly when I ran them in work

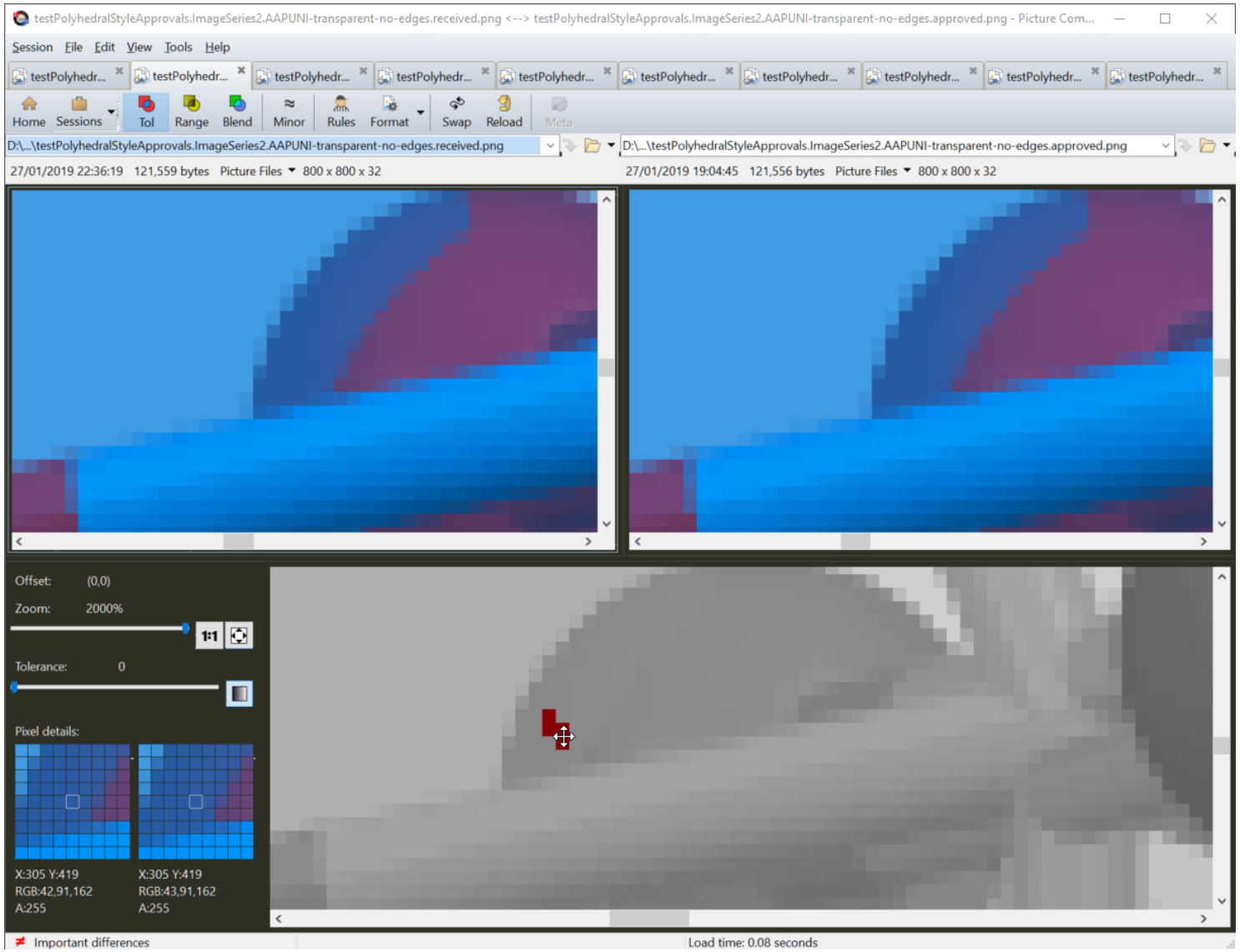
BeyondCompare 4

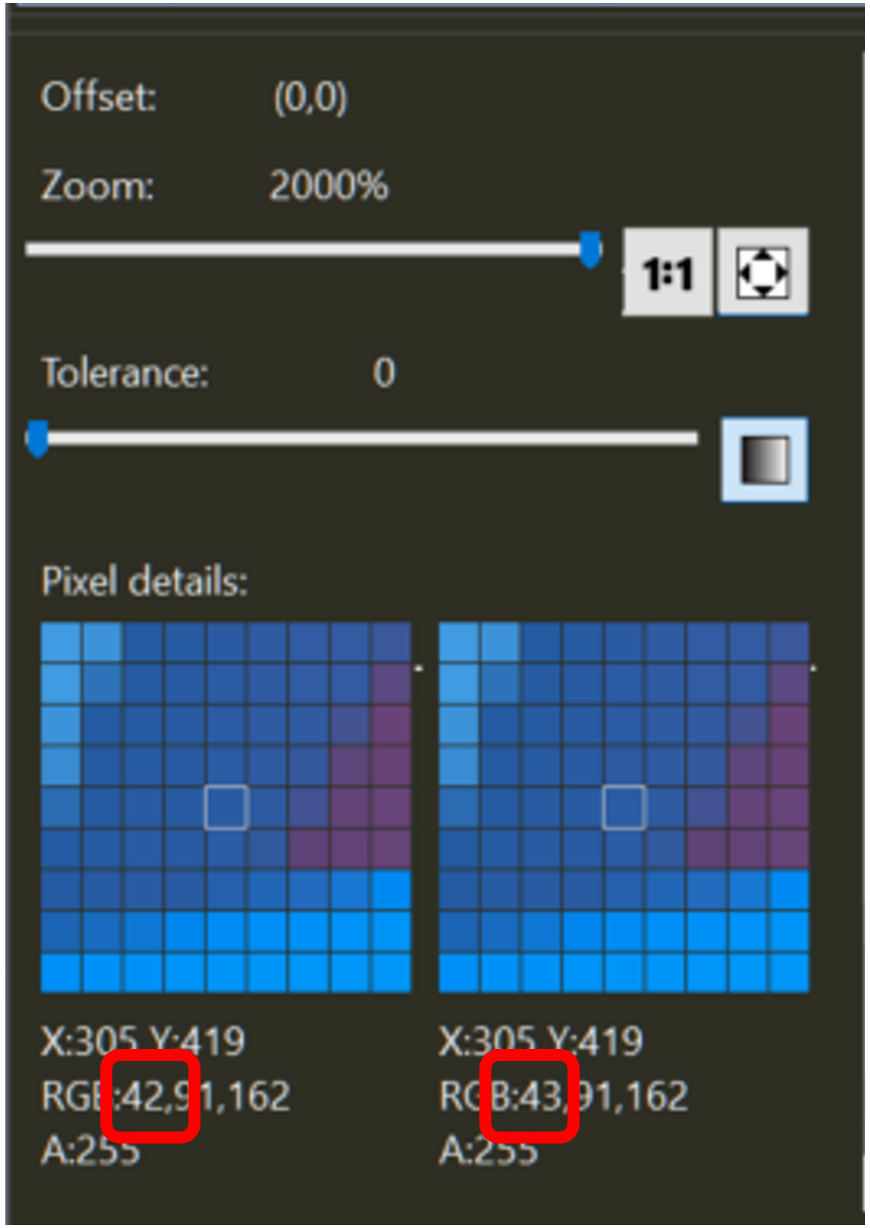



testPolyhedralStyleApprovals.ImageSeries2.AAPUNI-transparent-no-edges.received.png <--> testPolyhedralStyleApprovals.ImageSeries2.AAPUNI-transparent-no-edges.approved.png - Picture Com...
Session File Edit View Tools Help
Home Sessions Tol Range Blend Minor Rules Format Swap Reload Meta
D:\...\testPolyhedralStyleApprovals.ImageSeries2.AAPUNI-transparent-no-edges.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: 100%
1:1
Tolerance: 0
Important differences
Load time: 0.08 seconds


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: 200%
1:1
Tolerance: 0
Pixel details:
X:223 Y:257
RGB:255,255,255
A:255
X:223 Y:257
RGB:255,255,255
A:255
Important differences
Load time: 0.08 seconds


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


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: 2000%
1:1
Tolerance: 0
Pixel details:
X:305 Y:419
RGB:42,91,162
A:255
X:305 Y:419
RGB:43,91,162
A:255
Important differences
Load time: 0.08 seconds


Offset: (0,0)
Zoom: 2000%
1:1
Tolerance: 0
Pixel details:
X:305 Y:419
RGB:42,91,162
A:255
X:305 Y:419
RGB:43,91,162
A:255

Create custom QImage comparison class

- Solution: Allow differences in up to 1/255 of RGB values at each pixel
- Not visible to the human eye.

- https://github.com/approvals/ApprovalTests.cpp/blob/master/doc/CustomComparators.md

Does the difference matter?

- Legacy code is often brittle
- Testing makes changes visible
- Then decide if change matters
- Fast feedback cycle for efficient development

Looking back

- User perspective
- Learned about own code
- Enabled unit tests for graphics problems
- Approvals useful even for temporary tests

Contents

References: Tools Used

- Diffing
  - Araxis Merge: https://www.araxis.com/merge/
  - Beyond Compare: https://www.scootersoftware.com/

Contents

- Introduction
- Legacy Code
- Golden Master
- Approval Tests
- Example
- Resources
- **Summary** 

Adopting legacy code

- **You can do it!**
- ApprovalTests.cpp makes it really easy
  - Even for non-text types

Quickly Test Legacy C++ Code with Approval Tests

Want to know more?

- **Monday**: Beverages with Backtrace
- **Tuesday**: Tool Time – try it out!

Thank You

- **Clare Macrae Consulting Ltd**
  - **https://claremacrae.co.uk/**
  - clare@claremacrae.co.uk

Please connect on LinkedIn

- **Slides, Example Code**
  - https://claremacrae.co.uk/conferences/presentations.html
- **ApprovalTests.cpp**
  - https://github.com/approvals/ApprovalTests.cpp
  - https://github.com/approvals/ApprovalTests.cpp.StarterProject