ACCU 2021
VIRTUAL EVENT

Bloomberg
Engineering

undo

mosaic
CONSULTANTS TO FINANCIAL SERVICES

Testing Your Tests With Code Coverage

Richard Wallman

Why bother testing?

- Testing improves quality
  - Better testing == higher quality
  - Fewer regressions == happier clients (== happier devs)

- Testing is time-consuming and boring
  - Dependant on the skill of the test writer
  - Fatigue can make you miss things
  - Writing tests generates no perceivable benefit
  - Tests themselves are not normally tested for “quality”

Example library/class

```
#include "TestLib.h"

// Header also defines a class attribute:
// private: int m_x;

TestLib::TestLib()
    : m_x(0)
{
    // No further initialisation required
}

TestLib::TestLib(int x)
    : m_x(x)
{
    // No further initialisation required
}

TestLib::~TestLib()
{
    // No further deinitialisation required
}
```

```
int
TestLib::f(int x)
{
    if (x > 100) {
        return 100;
    }
    else if (x > 50) {
        return 50;
    }
    else {
        return x;
    }
}

int
TestLib::g(int x)
{
    return x + m_x;
}

int
TestLib::h(int x)
{
    return x - m_x;
}
```

Example test suite

```
#include "TestLib.h"

int
main(int argc, char** argv)
{
    TestLib t1;

    // Call TestLib.f() with one value
    t1.f(105);

    // Call TestLib.f() with another value
    t1.f(4);

    // Everything is awesome
    return 0;
}
```

Building the code (GCC)

- Add GCC option to generate coverage notes when compiling:

  -ftest-coverage

- Add GCC option to generate coverage data when running:

  -fprofile-arcs

- Add GCC option to link coverage library into the test suite:

  -lgcov

- GCC has a convenience option that does everything: --coverage

```
wallmari@kaiju:~/ACCU$ make
g++ -Iinclude --coverage -c -o src/library.o src/library.cpp
g++ -Iinclude --coverage -c -o src/test.o src/test.cpp
g++ -o test_suite -Iinclude --coverage src/library.o src/test.o
```

New file - *.gcno

- One file per source file
- Generated alongside the object file
- Constructs the block graph from source code
- Maps source code line numbers to blocks

Generating coverage data

- Run the test suite as normal
  - Coverage reporting has been compiled in
- Multiple runs can be made
  - Useful for trying different input parameters
  - Allows mutually exclusive execution paths to be tested

New file - *.gcda

- Generated alongside object file
  - But can be configured to store elsewhere at compile-time
- Contains runtime data
  - Transition counts
  - Value profile counts
- Cumulative
  - Multiple runs increase counts rather than replace them

Generating the coverage report

- Command is run against each source file
- Immediately returns coverage percentage
- Generates an annotated source code file

```
wallmari@kaiju:~/ACCU$ gcov src/library.cpp
File 'src/library.cpp'
Lines executed:55.56% of 18
Creating 'library.cpp.gcov'
```

Understanding the coverage report

```
    -:    0:Source:src/library.cpp
    -:    0:Graph:src/library.gcno
    -:    0:Data:src/library.gcda
    -:    0:Runs:1
    -:    1:#include "TestLib.h"
    -:    2:
    1:    3:TestLib::TestLib()
    1:    4:        : m_x(0)
    -:    5:{
    -:    6:        // No further initialisation required
    1:    7:}
    -:    8:
#####:    9:TestLib::TestLib(int x)
#####:   10:        : m_x(x)
    -:   11:{
    -:   12:        // No further initialisation required
#####:   13:}
    -:   14:
    1:   15:TestLib::~TestLib()
    -:   16:{
    -:   17:        // No further deinitialisation required
    1:   18:}
    -:   19:
```

```
    -:   20:int
    2:   21:TestLib::f(int x)
    -:   22:{
    2:   23:        if (x > 100) {
    1:   24:                return 100;
    -:   25:        }
#####:   26:        else if (x > 50) {
#####:   27:                return 50;
    -:   28:        }
    -:   29:        else {
    1:   30:                return x;
    -:   31:        }
    -:   32:}
    -:   33:
    -:   34:int
#####:   35:TestLib::g(int x)
    -:   36:{
#####:   37:        return x + m_x;
    -:   38:}
    -:   39:
    -:   40:int
#####:   41:TestLib::h(int x)
    -:   42:{
#####:   43:        return x - m_x;
    -:   44:}
```

Benefits of checking code coverage

- Ensure completeness of test suite
  - Tests should cover as close to 100% of the code under test, even if that requires multiple runs
    - There can be extreme edge cases that prevent a perfect score
- Removal of redundant code
  - Logical conditions prevent the execution path
  - Old, dead code

TL;DR Guide

- Write your code and tests
- Build with coverage options enabled
- Run the test suite (as many times as required)
- Generate code coverage report
- Take action if there is not 100% coverage

Questions?