C++ Code Coverage Analysis

Qt Developer Days 2012

by Harri Porten

About me

- Name: Harri Porten
- Company: froglogic GmbH, vendor of Squish for Qt and Squish Coco
- Position: co-founder and CEO
- Worked as Software Engineer at Trolltech and contributor to the KDE project

Overview

froglogic

- Theory
  - What is Code Coverage Analysis?
  - Why Code Coverage Analysis?
  - Coverage Levels
  - Instrumentation Types
  - Types of Testing
  - Challenges with Qt
  - Available Tools
  - Integrations
- Demos
  - Qt 5 Code Coverage Analysis
  - Demo with Squish Coco

What is Code Coverage Analysis?

- A way to measure the quality of software testing
- Percentage of code being tested
- White box testing (or glass or structural testing)
- Most commonly used metric
- An indirect indicator of code quality
- Not a quality guarantee

Why Code Coverage?

froglogic

- Discover yet untested code
  - Add new tests
  - Modify existing tests (data-driven)
- Discover unreachable code
- Focus testing effort

Coverage Analysis (Example)

C++ code coverage analysis when the addressbook application is passed the `-h` (help) command line option.

```
1  int main(int argc, char **argv) {
2    QApplication app(argc, argv);
3    if (argc == 2) {
4      QString arg(argv[1]);
5      if (arg == "-h" ||
6          arg == "--help") {
7        QTextStream out(stderr);
8        return 1;
9        out << "usage: "<<  argv[0] << "\n";
10     }
11   }
12   MainWindow mainWindow;
```

```
1    Executed code
0-1  argc == 2: was never false
1    Executed code
0-1  arg=="-h": was never false
0    arg=="--help": was never true or false
1    Executed code
1    Executed code
X    Unreachable code
-
-
0    Never executed
```

Result Summary

Lines executed:	4
Lines not executed:	2
Lines of dead code:	1
Expressions not covered:	3
Decision branches not executed:	2

Coverage Levels – Source File

First, rough overview:

File	Coverage
src/corelib/thread/qmutex.cpp	100%
...	...
src/corelib/tools/qstringlist.cpp	98.4%
...	...
src/gui/painting/qcolor.cpp	66.1%
...	...
src/widgets/widgets/qtoolbar.cpp	44.2%

Coverage Levels – Function

Drilling down:

File	Coverage
QThread::currentThread()	100%
...	...
QtPrivate::QStringList_removeDuplicates()	84.6%
...	...
QPrinter::setPageSize()	36.4%
...	...
QSqlError::text()	0.0%

Coverage Levels – Line/Statement

```
716     int QtPrivate::QStringList_removeDuplicates(QStringList *that)
717     {
718       int n = that->size();
719       int j = 0;
720       QSet<QString> seen;
721       seen.reserve(n);
722       for (int i = 0; i < n; ++i) {
723           const QString &s = that->at(i);
724           if (seen.contains(s))
725               continue;
726           seen.insert(s);
727           if (j != i)
728               (*that)[j] = s;
729           ++j;
730       }
731       if (n != j)
732           that->erase(that->begin() + j, that->end());
733       return n - j;
734     }
```

Coverage Levels – Decision

```
716     int QtPrivate::QStringList_removeDuplicates(QStringList *that)
717     {
718       int n = that->size();
719       int j = 0;
720       QSet<QString> seen;
721       seen.reserve(n);
722       for (int i = 0; i < n; ++i) {
723           const QString &s = that->at(i);
724           if (seen.contains(s))
725               continue;
726           seen.insert(s);
727           if (j != i)
728               (*that)[j] = s;
729           ++j;
730       }
731       if (n != j)
732           that->erase(that->begin() + j, that->end());
733       return n - j;
734     }
```

Decision	TRUE	FALSE
if (j != i)	0	3

Coverage Improvement

```
--- a/tests/auto/corelib/tools/qstringlist/tst_qstringlist.cpp
+++ b/tests/auto/corelib/tools/qstringlist/tst_qstringlist.cpp
@@ -292,6 +292,7 @@ void tst_QStringList::removeDuplicates_data()

     QTest::newRow("empty-1") << "Hello,Hello" << "Hello" << 1;
     QTest::newRow("empty-2") << "Hello,World" << "Hello,World" << 0;
+    QTest::newRow("shift") << "Hello,Hello,World" << "Hello,World" << 1;
 }
```

Coverage Levels – Condition

Line/statement coverage:

```
360     inline char qToLower(char ch)
361     {
362         if (ch >= 'A' && ch <= 'Z')
363             return ch - 'A' + 'a';
364         else
365             return ch;
366     }
```

Condition coverage:

```
360     inline char qToLower(char ch)
361     {
362         if (ch >= 'A' && ch <= 'Z')
363             return ch - 'A' + 'a';
364         else
365             return ch;
366     }
```

Condition	TRUE	FALSE
ch >= 'A'	848517	0
ch <= 'Z'	4992	843525

Coverage Levels – Decision/Condition

- Hybrid metric
- Union of decision and condition coverage
- Measure branch coverage
- Cover complex boolean logic
- Avoids drawbacks of line/statement coverage
- Still simple

Other Coverage Levels

- Path Coverage
- Modified condition/decision coverage
- Multiple condition coverage
- Loop coverage
- Entry/exit coverage
- ....

Instrumentation Types

- Source code insertion at compile time
  - Pro: highest possible coverage type
  - Drawback: increased binary size
- Binary instrumentation at runtime
  - Pro: No recompilation necessary
  - Con: Limited coverage type
  - Con: Less portable
  - Con: Requires debug information
  - Con: wrong results with optimized builds

Types of Testing (1/2)

froglogic

- Unit tests
  - Most often found use case
  - Leaving out GUI
- Automatic GUI tests
  - Becoming more vital
  - Still new approach
- Manual tests
  - Less popular
  - Small coverage but good to bridge gaps
  - Expose source code?

Types of Testing (2/2)

froglogic

Usage in QTestLib:

- Test build, execution and coverage as single step
- Associate test run and result with respective coverage

```
...
void TestCoverageObject::cleanup()
{
  cleanupTest();
  QString test_name="unittest/";
  test_name += metaObject()->className();
  test_name += "/";
  test_name += QTest::currentTestFunction();
  __coveragescanner_testname(test_name.toLatin1());
  if (QTest::currentTestFailed())
    __coveragescanner_teststate("FAILED");
  else
    __coveragescanner_teststate("PASSED") ;
  __coveragescanner_save();
}
```

Challenges with Qt (1/3)

Challenges with analysis of Qt-based applications:

- Macros
- Generated code

Q_FOREACH 1/2

What the programmer wrote:

```
QStringList l;
Q_FOREACH(QString s, l) { // or foreach()
    doSomething(s);
}
```

Q_FOREACH 2/2

froglogic

Definition for gcc compiler:

```
#define Q_FOREACH(variable, container)                                \
for (QForeachContainer<__typeof__(container)> _container_(container); \
     !_container_.brk && _container_.i != _container_.e;              \
     __extension__  ({ ++_container_.brk; ++_container_.i; }))        \
    for (variable = *_container_.i;; __extension__ ({--_container_.brk; break;}))
```

Definition for MSVC:

```
#  define Q_FOREACH(variable, container) \
    for (const QForeachContainerBase &_container_ = qForeachContainerNew(container); \
         qForeachContainer(&_container_, true ? 0 : qForeachPointer(container))->condition();       \
         ++qForeachContainer(&_container_, true ? 0 : qForeachPointer(container))->i)               \
        for (variable = *qForeachContainer(&_container_, true ? 0 : qForeachPointer(container))->i; \
             qForeachContainer(&_container_, true ? 0 : qForeachPointer(container))->brk;           \
             --qForeachContainer(&_container_, true ? 0 : qForeachPointer(container))->brk)
```

Q_OBJECT 1/2

What the programmer writes:

```
class MyWidget : public QWidget
{
    Q_OBJECT
public:
    MyWidget(QWidget *parent);
    ....
```

Q_OBJECT 2/2

Pre-processor will first expand:

```
#define Q_OBJECT \
public: \
    Q_OBJECT_CHECK \
    static const QMetaObject staticMetaObject; \
    Q_OBJECT_GETSTATICMETAOBJECT \
    virtual const QMetaObject *metaObject() const; \
    virtual void *qt_metacast(const char *); \
    QT_TR_FUNCTIONS \
    virtual int qt_metacall(QMetaObject::Call, int, void **); \
private: \
    Q_DECL_HIDDEN static const QMetaObjectExtraData staticMetaObjectExtraData; \
    Q_DECL_HIDDEN static void qt_static_metacall(QObject *, QMetaObject::Call, int, void **);
```

Generated code

froglogic

- Qt tools generating C++ code:
  - MOC → moc_*.cpp and *.moc
  - UIC → ui_*.h
  - RCC → qrc_*.cpp
- Coverage of little interest
- Approaches: ignore or (better) filter

Available Tools for C++

- PureCoverage
- gcov
- Squish Coco
- BullseyeCoverage
- Testwell CTC++
- VectorCAST/Cover
- …?

Available Tools vs. Qt

Watch out for:

- Supported platforms
- Supported compilers
- Ways to deal with Qt-specifics
- Integration with build system (e.g. QMake, CMake)
- Integration with unit test framework (e.g. QTestLib)

Build System Integration

Main approaches:

1. Replace C++ compiler transparently
2. Modify build to call instrumentation tool

Wanted for e.g. QMake users:

```
qmake CONFIG+=CodeCoverage
```

Addition to .pro file:

```
CodeCoverage {
    QMAKE_CXX=cs$$QMAKE_CXX
    ...
}
```

Test Framework Integration

froglogic

- Continuous Integration (CI):
  - Jenkins
  - Bamboo
  - Hooks in revision control system

- Result reporting:
  - HTML exports
  - EMMA XML (W3C recommendation)
  - Interactive GUIs

Qt 5 Code Coverage Analysis

froglogic

- Daily run of main Qt unit tests
- Found in: qtbase.git/tests/auto/
- Current coverage: 54.372% (134820/247959)
- http://download.froglogic.com/public/qt5-squishcoco-report
- Plan: analysis of each patch (sanity check in gerrit)
- Helps to find and understand bugs
- Contribute tests!

Squish Coco Demo

- Instrumenting an example
- Running some tests
- Viewing, interpreting and discussing coverage results
- Optimizing test order based on coverage results
- Comparing executions
- Generating HTML report