CANTATA

Feature Brief

Reusing xUnit Tests

Cantata provides a full and unique suite of intelligent testing capabilities for the efficient unit and integration testing of C and C++. In this feature brief we highlight the ways in which existing open source test scripts implemented in C/C++ (such as CppUnit and CxxTest), generically known as xUnit can be re-used and embedded inside Cantata.

Contents

Copyright Notice

1 Introduction

Cantata provides a full and unique suite of intelligent testing capabilities for the efficient unit and integration testing of C and C++. This feature brief highlights the ways in which existing open source test scripts implemented in C/C++ (such as CppUnit and CxxTest), generically known as xUnit can be re-used and embedded inside Cantata.

1.1 Open source tool limitations

Open source xUnit testing tools have four main limitations which lead organisations to consider upgrading to commercial tools:

- Total Cost of Ownership: organization must configure, maintain and support the tool, which is not its core business
- Efficiency of testing: Restricted style basic assertion testing requires significant manual scripting
- Lack of integrated code coverage: Additional manual review of code coverage is required
- Tool qualification: Open source tools do not meet regulated industries' requirement that tools are certified or qualifiable

1.2 Investment in existing xUnit test cases

Cantata provides an extensive range of intelligent testing capabilities, support and tool qualification evidence which address these open source limitations. However, if organisations have invested in a large body of unit tests implemented with open source xUnit tools, preserving that investment can be an important decision when considering upgrading to a commercial grade test tool.

Where code is unit tested with open source xUnit tools providing an acceptable degree of confidence, one option for companies is to replace the existing tests with an automatically generated set of Cantata baseline unit tests. For more information on this please see the Cantata Baseline Testing Feature Brief.

As Cantata tests are implemented in C/C++, an alternative option is to re-use open source tests by embedding them within Cantata test scripts. This has the advantage of retaining all the existing test cases, whilst supplementing them with the more powerful features of Cantata.

1.3 Cantata enhancements to existing xUnit tests

The following Cantata capabilities provide enhancements to existing test cases written using open source xUnit testing tools, when embedding them inside a Cantata test script:

- Call interface control – stubs and isolates to simulate or wrappers to intercept calls
- Integrated code coverage analysis and checks on achieving coverage target
- Checking data (including white-box checks on data declared static to the file or private)

Additional Cantata test cases can then also be added to the test script to supplement the existing

1.4 Steps for Embedding xUnit tests

Existing xUnit tests can be embedded into Cantata as one or more Cantata test cases. The following steps can be used to set up a new Cantata test script which will run existing CppUnit tests. The precise steps for other xUnit tools will vary.

1) Ensure the CppUnit tests build.
2) Enable Cantata on the project in Eclipse, and perform a build.
3) Select the relevant file(s) to create a Cantata test script, but create only a single generic test case (by selecting one of the radio buttons on the test script creation wizard).
4) Open the newly generated test script in the C/C++ editor, and locate the test case.
5) Where the comment "/* Call the SUT */" is found, paste in the following code:

```
CppUnit::Test *suite =
CppUnit::TestFactoryRegistry::getRegistry().makeTest();
            CppUnit::TextUi::TestRunner runner;
            runner.addTest(suite);
            bool wasSuccessful = runner.run();
```

6) A test case check similar to:

```
CHECK_NAMED("The existing CPP Unit tests", wasSuccessful, true);
```

can be added after the code in step 5 in order to check the CppUnit tests.

7) The following Header files from the CppUnit distribution will also be needed in the test script:

```
#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/ui/text/TestRunner.h>
```

8) Ensure that there is only 1 main function (provided by the Cantata test script), and perform a full rebuild.

1.5 Example CppUnit test in Cantata

A CppUnit example test_student.cpp embedded in a Cantata test script is available with this document. The test script code therein is reproduced below:

```
/*****************************************************************************/
/*                          Cantata Test Script                              */
/*****************************************************************************/
/*
 *    Filename: test_Student.cpp
 *    Author: chris.sharpe
 *    Generated on: 17-Dec-2008 16:35:04
 *    Generated from: Student.cpp
 */
/*****************************************************************************/
/* Environment Definition                                                    */
/*****************************************************************************/

#define TEST_SCRIPT_GENERATOR 1
#include <cantpp.h>  /* Cantata Directives */

/* Include files from software under test */
#include "Student.h"
#include "../TestStudent/TestStudent.h"
#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/ui/text/TestRunner.h>

/* pragma ipl Cantata testscript start */

/* Global Functions */
extern __cdecl void* operator new(size_t param_1);
/* None */

/* Global data */

/* Expected variables for global data */

/* This function initialises global data to default values. This function       */
/* is called by every test case so must not contain test case specific settings */
void initialise_global_data(){

    /* No initialisation for const data: MAXNUM */
}

/* This function copies the global data settings into expected variables for */
/* use in check_global_data(). It is called by every test case so must not   */
/* contain test case specific settings.                                      */
void initialise_expected_global_data(){

}

/* This function checks global data against the expected values. */
void check_global_data(){

}

/*****************************************************************************/
/* Test class declaration                                                    */
/*****************************************************************************/
class TEST_CLASS(Student)
{
public:
    void run_tests();
    void test_1(int);
    void test_Student_assignGrade(int);
};
/* Coverage Rule Set: 100% Entry Point + Statement + Call + Decision Coverage */
```

```
void rule_set(char* cppca_sut,
              char* cppca_context)
{
#ifdef CANTPP_SUBSET_DEFERRED_ANALYSIS
    TEST_SCRIPT_WARNING("Coverage Rule Set ignored in deferred analysis mode\n");
#else
    ANALYSIS_CHECK("100% Entry Point Coverage",
                   cppca_entrypoint_cov,
                   100.0);

    ANALYSIS_CHECK("100% Statement Coverage",
                   cppca_statement_cov,
                   100.0);

    ANALYSIS_CHECK("100% Call Return Coverage",
                   cppca_callreturn_cov,
                   100.0);

    ANALYSIS_CHECK("100% Decision Coverage",
                   cppca_decision_cov,
                   100.0);

    REPORT_COVERAGE(cppca_entrypoint_cov|
                    cppca_statement_cov|
                    cppca_callreturn_cov|
                    cppca_decision_cov,
                    cppca_sut,
                    cppca_all_details|cppca_include_catch,
                    cppca_context);
#endif
}
/*****************************************************************************/
/* Program Entry Point                                                       */
/*****************************************************************************/
int main()
{
    OPEN_LOG("test_Student.ctr", false);
    START_SCRIPT("Student", true);

    TEST_CLASS(Student) test_object;
    test_object.run_tests();

    return !END_SCRIPT(true);
}

/*****************************************************************************/
/* Test Control                                                              */
/*****************************************************************************/
/* run_tests() contains calls to the individual test cases, you can turn test*/
/* cases off by adding comments*/
void TEST_CLASS(Student)::run_tests()
{
    test_1(1);
    test_Student_assignGrade(1);

    rule_set("Student::*", "*");
    EXPORT_COVERAGE("test_Student.cov", cppca_export_replace);
}

/*****************************************************************************/
/* Test Cases                                                                */
/*****************************************************************************/
void TEST_CLASS(Student)::test_1(int doIt){
if (doIt) {
    START_TEST("CPPUnitTest",
               "<Insert test case description here>");

        /* Expected Call Sequence  */
        EXPECTED_CALLS("2*Course::getCourseGrade()#default");
        START_EXCEPTION
            /* Set global data */
            initialise_global_data();
```

```
            /* Set expected values for global data checks*/
            initialise_expected_global_data();

            /* Call SUT                                                  */
            /* There are two methods outlined below that can be used.    */
            /* Others may also exist but they are not shown here.        */
            /* 1) Use CPPUnit directives to create a CPPUnit test suite, */
            /* and then run it using the CPPUnit TestRunner method.      */

            /* Get the top level CPPUnit test suite from the registry.   */
            CppUnit::Test *suite =
            CppUnit::TestFactoryRegistry::getRegistry().makeTest();
            CppUnit::TextUi::TestRunner runner;
            runner.addTest(suite);
            /* execute the runner. */
            bool wasSuccessful = runner.run();

            /* 2) Create the CPPUnit test object directly                */
            /* (the StudentTest class in this case)                      */
            /* and then call each test method from the test object.      */
            /*   StudentTest* testObj = new StudentTest();               */
            /*   testObj->testConstructor();                             */
            /*   testObj->testAssignAndRetrieveGrades();                 */

            /* Test case checks */
            /* Since method 1 was used, we can check the result with a   */
            /* Cantata check.                                          */
            CHECK_NAMED("The existing CPP Unit tests", wasSuccessful, true);
            /* Checks on global data */
            check_global_data();
            NO_EXCEPTIONS
        END_EXCEPTION
        END_CALLS();
    END_TEST();
}}

void TEST_CLASS(Student)::test_Student_assignGrade(int doIt){
if (doIt) {
    START_TEST("test_Student_assignGrade",
               "<Insert test case description here>");

        /* Expected Call Sequence  */
        EXPECTED_CALLS("");
        START_EXCEPTION
            /* Test case data declarations */
            Student* testObj = new Student("Chris", "2");
            testObj->no_of_courses = MAXNUM;
            std::string co = "New Course";
            int gr = 14;
            /* Set global data */
            initialise_global_data();
            /* Set expected values for global data checks*/
            initialise_expected_global_data();

            /* Call SUT */
            testObj->assignGrade(co, gr);

            /* Test case checks */
            /* Checks on global data */
            check_global_data();
        NO_EXCEPTIONS
        END_EXCEPTION
        END_CALLS();
    END_TEST();
}}

/*****************************************************************************/
/* Stubs and Wrappers                                                        */
/*****************************************************************************/
#pragma ipl Cantata ignore on

/* BEFORE wrapper for Course::getCourseGrade*/
```

```
bool BEFORE_Course::getCourseGrade(Course*& cppsm_this_object){
    int instanceType = AFTER_WRAPPER;
    REGISTER_CALL("Course::getCourseGrade()");

    IF_INSTANCE("default") {
        return AFTER_WRAPPER;
    }

    LOG_SCRIPT_ERROR("Call instance not defined.");
    return instanceType;
}

/* AFTER wrapper for Course::getCourseGrade*/
int AFTER_Course::getCourseGrade(int cppsm_return_value,
                                 Course* const& cppsm_this_object){
    int returnValue;
    IF_INSTANCE("default") {
        return cppsm_return_value;
    }

    LOG_SCRIPT_ERROR("Call instance not defined.");
    return cppsm_return_value;
}

/* REPLACE wrapper for Course::getCourseGrade*/
int REPLACE_Course::getCourseGrade(Course*& cppsm_this_object){
    int returnValue;

    LOG_SCRIPT_ERROR("Call instance not defined.");
    return returnValue;
}

/* Stub for Course::getCourseName */
std::string Course::getCourseName(){
    std::string returnValue;
    REGISTER_CALL("Course::getCourseName()");

    IF_INSTANCE("default") {
        return NOT_SET;
    }

    LOG_SCRIPT_ERROR("Call instance not defined.");
    return returnValue;
}

#pragma ipl Cantata ignore off
/* pragma ipl Cantata testscript end */
/*****************************************************************************/
/* End of test script                                                        */
/*****************************************************************************/
```