Exam

Use Case Specification: "Unit Test Framework"

Version 0.1

Revision History

Date	Version	Description	Author
2007–03–08	1.0	Initial document	Petr Ovtchenkov

Contents

1 Unit Test Framework **2**

1.1 Brief Description . . . 2

2 Flow of Events **2**

2.1 Basic Flow . . . 2

2.2 Alternative Flows . . . 2

2.2.1 Test suite for interactive program . . . 2

2.2.2 Multi-threaded test suite . . . 3

2.2.3 Test suite with fork . . . 3

2.2.4 Test suite with process intercommunication . . . 3

3 Special Requirements **5**

3.1 Suitable for debugging . . . 5

3.2 Output from few control flows . . . 5

3.3 Integrable . . . 6

3.4 Tests dependency . . . 6

3.5 Tests grouping . . . 6

3.6 Report format . . . 6

3.7 Test timeout . . . 6

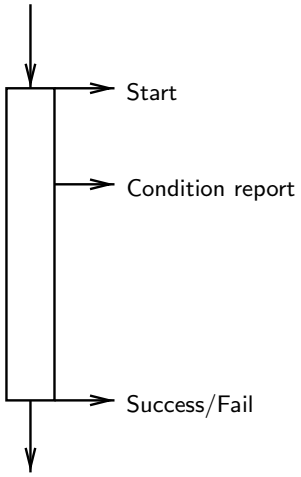

Figure 1: The test case with single control flow.

4 Extension Points **7**

4.1 Remote testing 7

4.2 Performance Mesure 7

1 Unit Test Framework

1.1 Brief Description

Unit test framework should provide a matched set of components for writing test programs, grouping tests into test cases and test suites, monitoring and controlling their runtime execution. Unit test framework should be convenient for testing programs and set of programs that implement services and client–server technologies (the set of intercommunicating programs, many control flows).

2 Flow of Events

2.1 Basic Flow

The test suite with single control flow. It notify about start of test suite, run tests within this test suite (in the order following from tests dependency graph), report about checks of conditions (verbosity depends upon log level), and print the resulting summary of the test suite (see fig.1).

2.2 Alternative Flows

2.2.1 Test suite for interactive program

The test suite with single control flow. It notify about start of test suite, run tests within this test suite (in the order, described by tests dependency graph), print conditions report (verbosity depends upon log

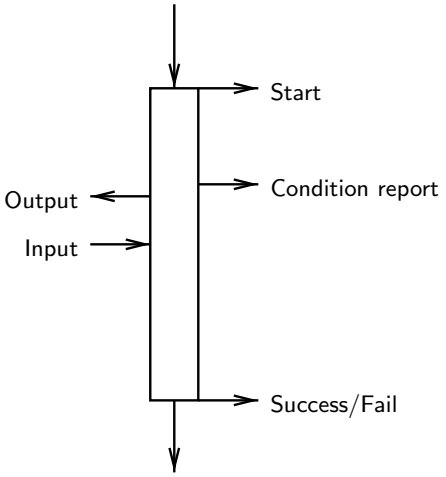

Figure 2: The test case with single control flow; test suite is interactive, it print some message to terminal and expect input from terminal.

level), and print the resulting summary of the test suite (see fig.2). During execution of the test suite, it require interaction with: it print messages onto terminal and expect input from terminal.

This use-case reflect test suite for interactive part of program.

2.2.2 Multi-threaded test suite

The test suite with more then one control flow. It register start of test suite, run tests within this test suite (in the order, described by tests dependency graph), print conditions report (verbosity depends upon log level), and print the resulting summary of the test suite (see fig.3). Test control flow is splitted into few threads.

This use-case reflect test suite for multi-threaded program, such as server, that provide some kind of service.

2.2.3 Test suite with fork

The test suite with more then one control flow. It register start of test suite, run tests within this test suite (in the order, described by tests dependency graph), print conditions report (verbosity depends upon log level), and print the resulting summary of the test suite (see fig.4). Some tests of the test suite forked into several processes.

This use-case reflect test suite for daemons and client-server interaction.

2.2.4 Test suite with process intercommunication

The test suite with more then one control flow. It register start of test suite, run tests within this test suite (in the order, described by tests dependency graph), print conditions report (verbosity depends upon log level), and print the resulting summary of the test suite (see fig.5). Some tests of the test suite forked into several processes.

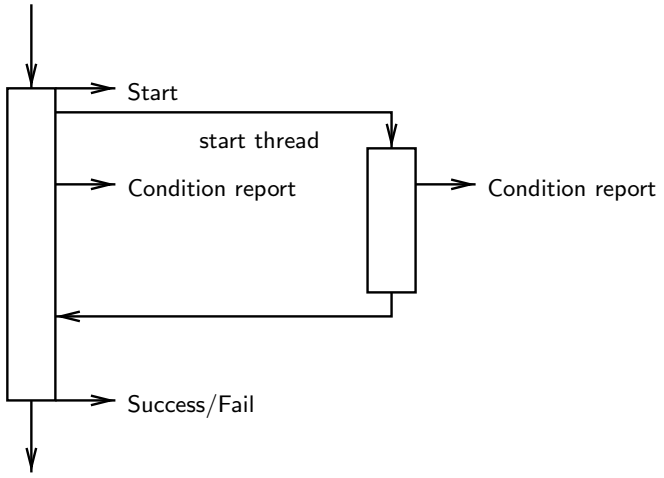

Figure 3: The test case with few control flows, single process.

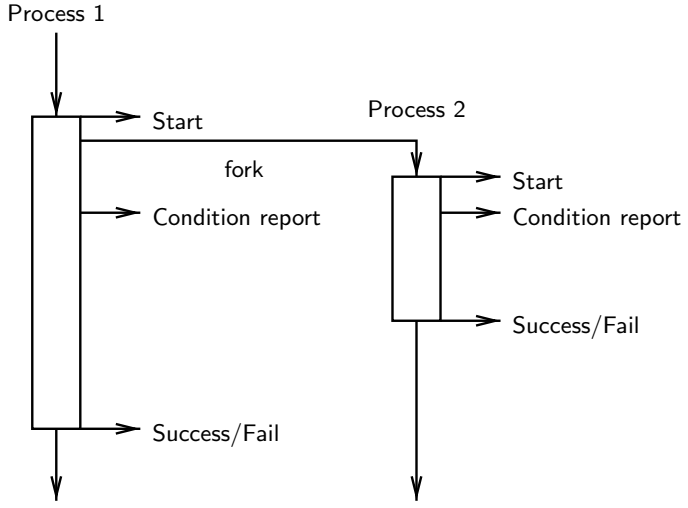

Figure 4: The test case with few control flows, process forked.

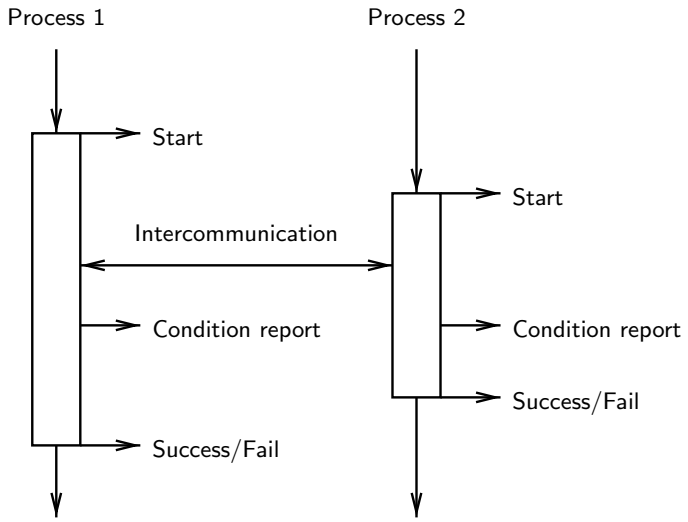

Figure 5: The test case with few control flows, intercommunicated processes.

This use-case reflect test suite for daemons and client-server interaction.

3 Special Requirements

3.1 Suitable for debugging

Unit tests not only instrument of QA (see fig. 6). Unit tests written in parallel with functionality implementation,[1] provide a good practice ground for debugging. Implementation of special test cases for debugging after detection problem in unit test isn't a good idea by financial (time and other resources) reasons. Even more: in non-trivial cases not evident, what was wrong: either incorrect functionality imlementation, or test, or even specification. Developer should has a chance to debug program keeping in mind as main functionality, as unit test logic.

Due to this unit test framework intended for testing services, and services often has own logic of signal processing, the unit test framework should not (at least by default) intervenes into signal catching. This also concern fatal errors with dumping core: dumping core "as is" is more preferable way (useful for post-mortal debugging) then nice report "you test died" with unuseful stack.

3.2 Output from few control flows

Unit test suite should provide reasonable output as for tests with single control flow, as for multi-threaded and multi-process tests.

Unit test suite should provide reasonable output for test with console output and input, i.e. split prints from interaction with user from prints about failed conditions.

[1] as assume XP technique

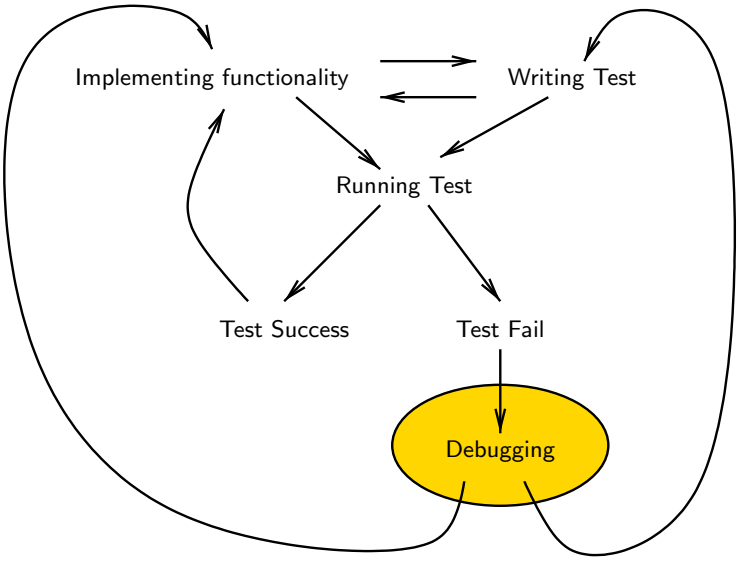

Figure 6: Development application: functionality implementation cycle.

3.3 Integrable

To be light-weight, easy integrable into other tools and scripts.

3.4 Tests dependency

Tests run order should be determined from tests dependency graph.

3.5 Tests grouping

The ability of grouping tests is desirable:

- single function tests
- class (group of tests) with initialization and finalization.

3.6 Report format

Unit test framework should provide few levels of output verbosity, and few formats of reports (plain text, xml/html).

3.7 Test timeout

Due to deadlock (or other conditions, leads to stalled test) may occur, it would be nice if monitoring program abort such test after specified timeout, and continue other tests (taken into account tests dependencies, section 3.4).

4 Extension Points

4.1 Remote testing

Test suite monitoring may be performed from remote host. This also useful if test suite allow monitoring a few (intercommunicating) processes, that run on different hosts.

4.2 Performance Mesure

Unit tests may be used for performance measure of key technology solutions.