Test Resources and their implementation in SUnit 3

Joseph Pelrine
MetaProg GmbH
jpelrine@acm.org

Abstract

The high performance aspect of extreme Programming derives in part from the rapid feedback cycles in unit testing. Certain resources required for testing, however, are very time-intensive in their instantiation, and can slow down the development process to the point where the advantages of test-driven programming are lost. Through the implementation of "persistent" test resources, though, this deficit can be compensated for. TestResources offer a flexible implementation of this in Smalltalk.

Contents

The Problem

One of the practices of extreme Programming that contributes greatly to its success is the idea of test-driven programming using extremely (sic) short feedback cycles. Press a button, run your test, get your answer. The longer this feedback cycle takes, the more difficult it becomes to program in an extreme style, and the more thinking has to be done up-front to avoid wasting precious time.

There are usually a few test objects that can be expensive to instantiate, though:

Database connections

Extremely complex objects

and we don't want to have to do this for each Test Case. Keeping a test object alive over multiple Test Cases breaks one of the primary rules of unit testing, by not starting with a clean slate before each Test Case is run, but is nevertheless desirable from the point of rapid feedback.

> *One final bit of philosophy. It is tempting to set up a bunch of test data, then run a bunch of tests, then clean up. In my experience, this always causes more problems than it is worth. Tests end up interacting with one another, and a failure in one test can prevent subsequent tests from running. The testing*

> *framework makes it easy to set up a common set of test data, but the data will be created and thrown away for each test. The potential performance problems with this approach shouldn't be a big deal because suites of tests can run unobserved.* [Beck95]

This is why we've developed Test Resources.

Test Resources

A Test Resource is an object which is needed by (as a rule) a number of Test Cases, and whose instantiation is so costly in terms of time or resources that it becomes advantageous to only initialize it once for a Test Suite run.

Example

Test Resources are currently implemented as an optional singleton:

Follows standard singleton #current protocol

But #new is not overridden to return an error

in order to give developers the option of using a sole instance of a resource, or multiple instances. Test Resources have a polymorphic syntax with TestCase, with #setUp and #tearDown messages.

Cookbook

How do I set up a Test Resource?

Per default, all required resources are initialized before a TestSuite runs. This occurs non-deterministically by collecting the resources defined as required by the Test Cases into a Set, and sending them the message #isAvailable. The default implementation of #isAvailable checks to see if the variable holding the singleton is nil, and performing a lazy initialization it if it is not. This allows subclasses of TestResource to override #isAvailable to perform a different checking routine, or not to initialize the resource if the test environment allows manual initialization of resources.

How can I use Test Resources?

Test Cases optionally/preferably define required resources by overriding the class method #resources to return a Collection of resource class names. By not defining a resource in this method, its initialization becomes responsibility of the TestCase itself. In this case, the resource will probably not be accessed via the singleton, but rather held in a variable of the Test Case object.

When do you release a TestResource? This is a difficult question to answer, and the decision was made not to answer it. In the default implementation, TestRunner sends #reset to the resource class when it is finished. This invokes #tearDown on the resource and nils it out. A number of developers working with the new Test Resource version of SUnit have had other needs, and have easily implemented timed-release TestResources and manual-release TestResource.

Some future directions for development include initialization order and conditional initialization (dependent upon a pre-run TestCase).

Testing Practices

Conclusion

The implementation of TestResources seems to be the next logical step in the development of SUnit, and has been happily accepted by all developers working with it. Compromises have been made, but as few restrictions as possible have been built into the tool. Try it, you'll like it! SUnit is available at

http://ansi-st-tests.sourceforge.net/SUnit.html

References

[Beck95] Kent Beck, 1995. *Simple Smalltalk Testing: with Patterns*. Available at http://www.xprogramming.com/testfram.htm

---

Joseph Pelrine
MetaProg GmbH
Bachlettenstrasse 41
CH-4054 Basel
Switzerland
Tel: +41 61/281 07 18
Fax +41 61/281 07 80
Email: jpelrine@acm.org