Dakar Testing - An innovative and new way of writing tests

Karsten Kusche

karsten@heeg.de

Abstract: Even though SUnit, the first unit-testing framework, was developed in the early nineties, the way of writing unit-tests did not change much. Although this framework has also been ported from Smalltalk to a lot of other programming languages, and many development environments currently support running the tests, the test-driven development itself does not integrate into these development environments very well. This article will discuss the problems of the current method of writing tests and how this method has been improved by Dakar Testing.

1 The Current Problems

Unit tests are usually nothing more than normal classes and methods that can be run by a test-runner. There is nothing wrong with this, as it allows for a high degree of flexibility. Current development environments typically provide an easy method to run the tests and see if problems exist.

However, there is a 'implicit link' between the test methods and the normal methods. Tests call methods and by doing this, they test them. Some tests do not only test a simple method, but rather test something bigger, like some kind of working process. This 'implicit link' is only stored in the developers memory and thus can not be shared easily with other developers.

Figure 1: The model of tests linked to methods. Any number of tests cover any number of methods.

The creation of a test can be a tedious task. The developer always has to navigate between the test and the normal source code. This is annoying not only when tests and methods are implemented, but also when a developer wants to understand other developers source code. Tests help understanding the usage of the methods involved, but the need to switch between the source code and the tests can be very distracting.

It is not easy to begin writing tests and to think how components should be designed to be easily tested. Additionally the development environments typically treat test methods

as every other method. Some of them can create empty test methods, like Visual Studio 2005 Team System [Mic] or Eclipse [Ecl]. Those also annotate the test methods to help the compiler identifying them as tests, so that they can be run automatically, but these tools do not help managing the tests.

2 Dakar Testing

2.1 Explicit Links

To solve the problem of implicit links, the model of a test and a method is extended to support explicit linking (figure 2). This link is stored by a link manager. Every time the developer creates a new test, it is automatically linked to the selected method.

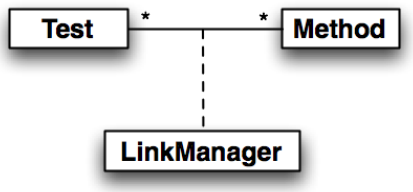

Figure 2: The model of tests that are explicitly linked to methods. The links are handled by the LinkManager.

The advantage of this link is the ability to see all tests that are testing the current method and when this method is changed, these tests are the ones that should at least be run. The link can also be used in various ways to improve the feedback to the developer. By defining states for a test like *passed*, *failed* or *pending* the methods can adopt this status and display the state of its assigned tests. With this information it is easier for the developer to find the tests that need to be run, or methods that need tests at all. This also illustrates the test-coverage of a project in a very simple way.

2.2 The User Interface

So, how is all this integrated into the development environment? Dakar Testing is an extension for VisualWorks, a Smalltalk development environment by Cincom [Vis]. As seen in figure 3, the lower part of the window is divided into three parts; two text editors and one list. While the left editor is the normal source code editor for the methods, the right editor is used for the modifying or creating test methods. The central list is showing all tests that are assigned to the current method. If no method is selected then it lists all linked tests from all methods of the selected class.

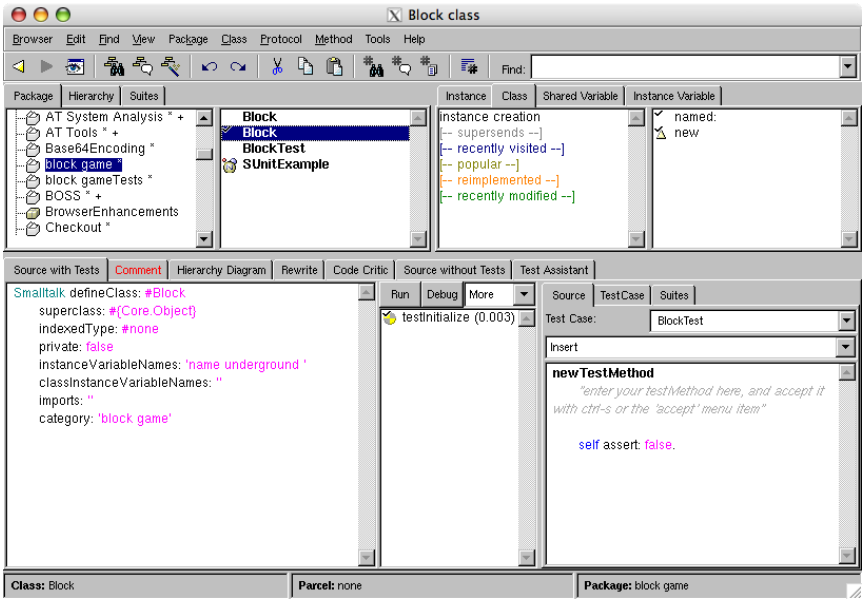

Figure 3: Dakar Testing adds the test-list and the test-editor to the lower part of the window, integrating the test-driven development directly into the development process.

This layout has several big advantages: the developer can develop the tests while he sees the method he wants to test. He can easily see and edit the tests that are linked to the current method without switching windows, or when trying to understand a method he can also see the tests that cover this method and may understand the usage and intention of this method faster.

In addition, the creation of tests has been simplified in Dakar Testing. The developer does not need to care about the storage of a test anymore. The tests are automatically compiled into test-case classes that are named by following a simple naming convention. There is also a popup menu that lists all the available testing-methods like *#assert:* or *#should:*. By selecting such a method the appropriate method-call is inserted at the cursor position.

Adding methods to the selected test-case is possible when the TestCase-tab is selected. Such methods can be methods like *#setUp* or *#tearDown* or any other method that may be required in certain test-methods of that test-case.

In Dakar Testing, the test-suite usage has been simplified a lot. Any tests can be combined into a new suite. This can be done using the Suite-Tab. This tool allows not only the creation of new suites, but also the assignment of existing tests to the current method. Suites can be assigned to methods just like any other test.

Running the tests is done with the existing unit-testing frameworks SUnit or SUnitToo, depending on which of them is installed. The status of each test updates according to the result of each test. If a test is included in a suite, then the suite's status is also updated, so that all suites are always showing the current status of all included tests. These states are

also shown in the icon of the method or the class to which the test is assigned, helping the developer to identify failing tests and methods much faster.

2.3 Storage

All the information about the links and suites is stored in classes and methods. They are created in the same package as the test-cases which allows the developer to easily find this information if he needs to. This method of storage also allows for the use of source control tools like Store [Sma] for VisualWorks. By using such management tools the links can be shared with other developers, or the links of several developers can be combined. The information is stored in a very readable way as can be seen in figure 4 in an example-method. Storing the links in such a simple way helps developers to merge their links without requiring special tools.

```
associationForBlockOnunderground
    <dakarCreateAssociatedSuiteWithClass: 'Block' onSelector: #underground>
    ^(AssociatedSuite withClass: 'Block' onSelector: #underground)
        cleanSuite;
        addTest: (TestMethod withClass: 'BlockTest' selector: #testInitialize);
        yourself
```

Figure 4: This example method shows how links are stored in Dakar Testing. The method #underground of the class Block is linked to the test #testInitialize of the test-case BlockTest. If more tests are linked to this method, they are all listed there and added with #addTest:. There is only one of these storage-methods per methods that have linked tests. If no tests are linked, then this storage-method is also removed.

2.4 Using Existing Tests

Dakar Testing can also be used for projects, where lots of tests have already been written. Obviously, these tests have to be assigned to methods without too much work. The problem is that this cannot be done automatically. For this purpose the tool 'Test Assistant' was developed. It enables the developer to easily identify the methods that were tested by a given test method and link them.

The usage of Test Assistant is fairly straightforward. The developer chooses the class to which he wants to assign tests. Then he selects a test-case and a test-method and runs that test. During that test the selected class is prepared for a code-coverage analysis. After the test-run all called methods are collected and all methods that are directly called by the selected test-method are listed. The developer can then decide which of them is tested and can link them. For tests that do not call the tested methods explicitly, there is also an option to show all called methods of the selected class or of its super-classes.

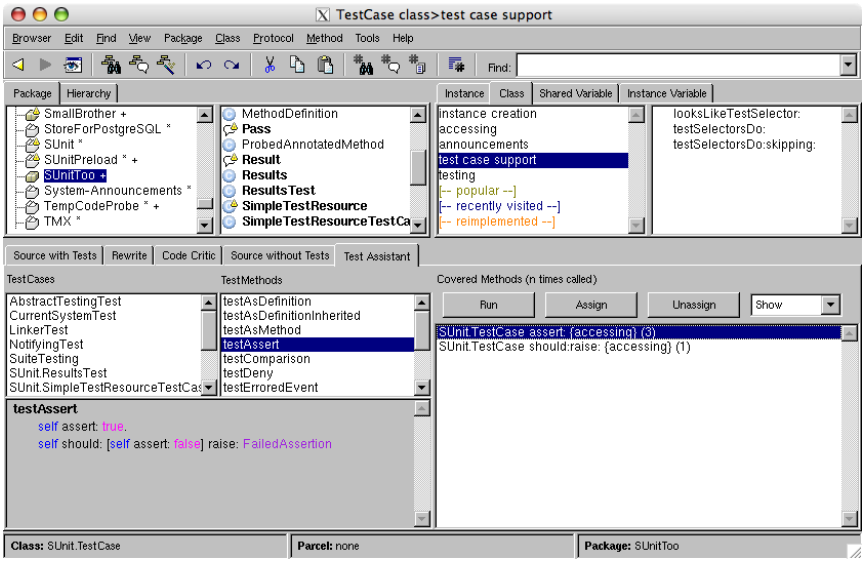

Figure 5: The Test Assistant in the lower part of the window allows easy identification of tested methods to link them to the selected test-method.

References

[Ecl] Eclipse Foundation Inc, http://www.eclipse.org. *Eclipse IDE*.

[Mic] Microsoft, http://msdn2.microsoft.com/en-us/teamsystem/. *Visual Studio Team System*.

[Sma] Cincom Smalltalk. *VisualWorks Source Code Management Guide*.

[Vis] *Cincom Smalltalk: VisualWorks*. http://www.cincomsmalltalk.com/userblogs/cincom/blogView?content=vwfactsheet.