UNIVERSITÄT
BERN

TestView Plugin

A Nautilus Plugin to facilitate Test-driven Developement

Bachelor Thesis

Dominic Sina
from
Zollikofen BE, Switzerland

Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

05. Februar 2015

Prof. Dr. Oscar Nierstrasz
Boris Spasojevi

Software Composition Group
Institut für Informatik und angewandte Mathematik
University of Bern, Switzerland

Abstract

The purpose of this bachelor project is to improve the Pharo enviroment by making it more unit test friendly. Instead of writing a new system browser we chose to realise this in the form of a Nautilus plugin since this makes it easy to set up and builds on an established part of the Pharo enviroment. Our plugin's functions includes making it easier to creating new tests for a given method. Specifically being able to see a method and the test that is currently being written side by side and a fast way to create new tests with if necessary new test classes and packages. The second provided functionality is a search for existing tests corresponding to the currently selected method. The search takes in to account if the methods are similarly named, if the supposed test method is a subclass of TestCase and if it contains the selector of the original method. The user can then add and remove elements from the resulting collection of tests to ensure that only acctual tests are shown. These features not only facilitate test driven development but can also be used to help understanding methods by looking at their tests.

Contents

1 Introduction

1.1 Unit Testing

1.2 Test-driven Development

1.3 The Nautilus Plugin Framework

2
Related Work

In which we learn what have other done to address similar problems. For example, the work of Star [**?**]

3
The Problem

In which we understand what the problem is in detail.

4
The TestView Plugin

In which you describe your solution.

4.1 Installation and activation

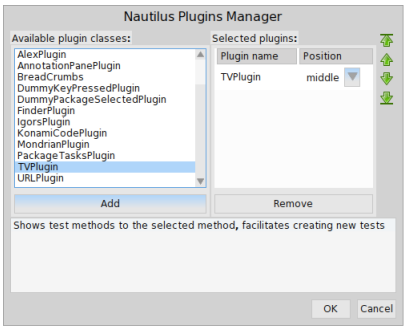
To install and active the TVPlugin follow the steps listed bellow:

1. To download the necessary packages simply simply execute the following lines in a Pharo workspace

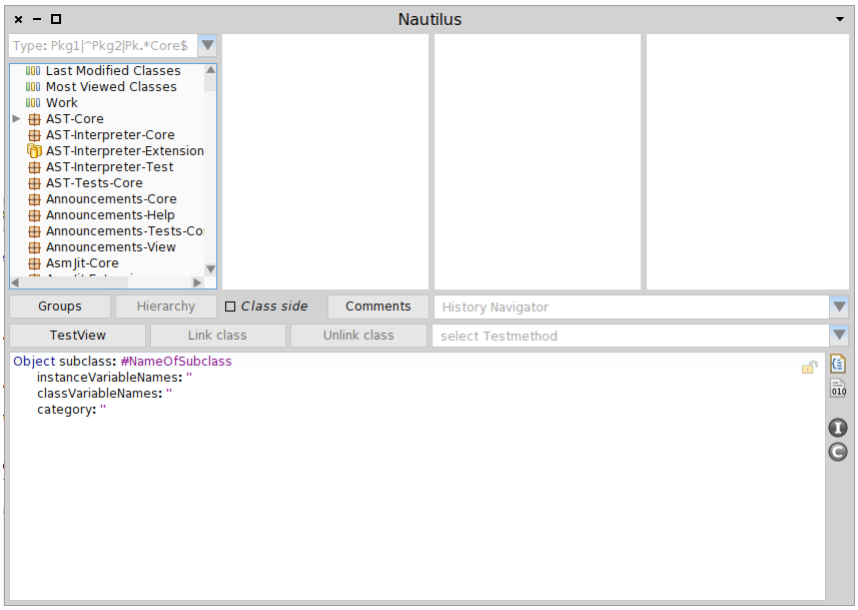
```
url: 'http://smalltalkhub.com/mc/DominicSina/TestView/main';
package: 'ConfigurationOfTestView';
load.
(Smalltalk at: #ConfigurationOfTestView) loadDevelopment.
NautilusPluginManager new openInWorld
```

   Once this is finished the Nautilus Plugins Manager will open.

2. Here you click on ”TVPlugin” under ”Available plugin classes” and then press on the ”Add” button. In the ”Selected plugins” collumn you can specify where most visual elements will be shown in your Nautilus windows. Click ”Ok” to confirm.



3. When you open a Nautilus window from now on the plugin will be started until you remove it again using the Nautilus Plugins Manager. To verify if the plugin is activated check if this row is displayed in the position you selected.



4.2 Basic Components

Here a quick overview of all the visiual elements can be found. The TVPlugin adds 3 buttons and one droplist to your Nautilus windows. The ”TestView” button toggles the search for test methods to the currently selected method. If it is toggled on a second source code editor will appear at the bottom where the test methods will be displayed. This list will change whenever a new method is selected in the Nautilus window. The initial list shown in the droplist is composed of tests found through the search algorithm described under 4.3 The search Algorithm. The additional buttons ”Link Class” and ”Unlink Class” allow to customise this list if the search for any reason did not find some tests.

4.3 The search Algorithm

In this section I will provide a detailed explanation how corresponding tests to a certain method are found. It is a hierarchical search with two stages.

In the first stage all test classes are determined. For this all classes in the your enviroment are taken into consideration in the beginning. By the end of the first stage only those classes that inherit from ”TestCase” as well as pass a substring search in the class name are then passed over to the next step. The substring search requires the name of the class in question to contain both ”test” and the name of the selected class. If those two conditions are met then the class in question is a test class of the selected class. The substring search is not case sensitive and the matches for ”test” as well as the selected method can’t overlap, meaning one letter can only be used to match partially either ”test” or the selected class name.

Original class name	Possible class names	Not a test class name
String	StringTest, stringtest, TestString	String, Test, StrinTgEST
Protest	ProtestTest	Protest

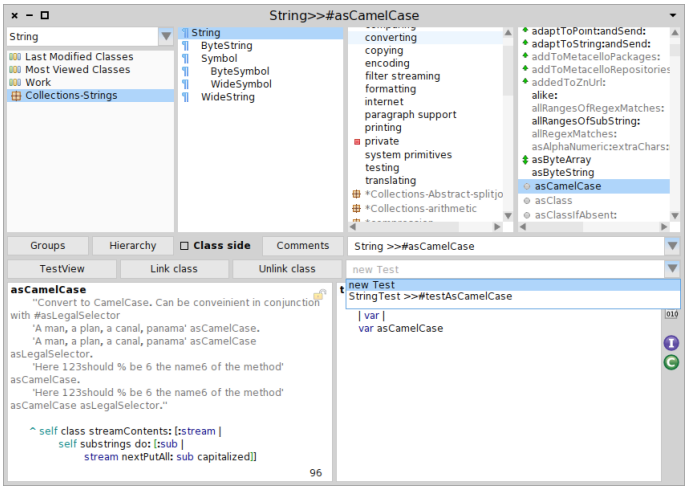
In the second stage all test methods corresponding to the selected method out of all methods of the test classes are determined. Similarly as before the test method name needs to contain the name of the selected method in addition to ”test”. The second criterion is if the method in question uses the selector of the selected method. Unlike stage one this stage is fairly inclusive in that if only one of those criteria is satisfied it still qualifies as a test to the selected method.

The two criteria of the second stage also serve to order the found test methods. The ones displayed on top satisfy both criteria. The ones that only satify the naming requirement are shown below these and the last ones are those that only contain the selector of the selected method.

4.4 Creating new Tests

The other main functionality of the plugin besides finding tests is to facilitate creating them. When you have a method open in the first code editor to which you would like to add a test select the ”new Test” option from the droplist. This option is an exception to the order discribed at the end of 4.3 The search Algorithm and is always shown on top of the list. This option also can be found there regardless wheter there were any tests found for the selected method. A template will appear in the second editor where the user can write the new test. When the test is accepted the user will be asked to name the class in which this test will be added. If the specified class doesn’t exist it will be created and the user will be asked for a package to place it in.

A faster way to create a new test is to select an existing test from the droplist and alter it. By selecting a new name the old test will not be overwritten and the new test will be added to the class of the old test. Any test that written using the second editor will result in its class being linked to the class of the selected method. This makes sure that the class in which the test was added is later recognised by the search algorithm.



5
The Validation

In which you show how well the solution works.

6
Conclusion and Future Work

In which we step back, have a critical look at the entire work, then conclude, and learn what lays beyond this thesis.

7 Anleitung zu wissenschaftlichen Arbeiten

This consists of additional documentation, e.g. a tutorial, user guide etc. Required by the Informatik regulation.