Parameterized Unit Testing Tool for .Net Framework

Doaa Sami
Computer Science,
Misr University
Cairo, Egypt

Hala abdel-Galil,Ph.D
Computer Science,
Helwan University
Cairo, Egypt.

Prof.Mostafa Sami
Computer Science
Helwan University
Cairo, Egypt.

ABSTRACT
Unit testing has been widely recognized as an important and valuable means of improving software reliability, as it exposes bugs early in the software development life cycle. However, manual unit testing is often tedious and insufficient. Testing tools can be used to enable economical use of resources by reducing manual effort [11]. Recently the use of parameters in unit testing has emerged as a very promising and effective methodology to allow the separation of two testing concerns or tasks: the specification of external, black-box behavior (i.e., assertions or specifications) by developers and the generation and selection of internal, white-box test inputs (i.e., high-code-covering test inputs) by tools [4, 12]. The Unit Testing Tool produced in this research is based on a parameterized test method that takes parameters, calls the code under test, and states assertions.

Keywords

Testing, unit testing, parameterized unit testing

1. INTRODUCTION

The essence of software testing is the comparison of the actual execution of a piece of software against that piece of software's expected behaviour. As such, any attempt at automating the whole process of unit testing involves the mechanical generation of test cases that will exercise the software unit, the execution of these test cases, and an automated mechanism for determining whether the software behaved as expected [2]. However, to be of any practical use to the software development professional, he must also be able to measure the thoroughness of the automatically generated test suite [5, 12].

A unit test is simply a method without parameters that performs a sequence of method calls that exercise the code under test and asserts properties of the code expected behaviour. Unit tests are a key component of software engineering [8]. The Extreme Programming discipline, for instance, leverages them to permit easy code changes. Being of such importance, many companies now provide tools, frameworks, and services around unit tests and each tool dedicated to only programs written in a special programming language [3, 7, 11].

So this paper presenting a parameterized unit testing tool that has the Standard unit testing features such as test, fixture, setup, teardown, ignore, expected exception, etc, Easy to use graphical user interface, Recipes for combining several test assemblies into one test suite, Search capabilities across tests, output, and statistics, Statistics per test to create performance base line, Categories to group tests for execution, Works with any .NET language (C#, VB.NET, Managed C++, etc.)

2. THE PARAMETERIZED TESTS IN THE TOOL

Parameterized testing is sometimes also referred to as data-driven testing. The Unit Tool supports parameterization of tests in several ways:

- Simple Parameterization of a single test.
- Parameterization using a static method or property.
- Parameterization using an XML file .
- Parameterization using a database table.

Which option will be chosen depends on the circumstances. The order in which they are listed here starts with the simplest case and ends with the most powerful scenario. Parameterization can be used to test algorithms, APIs, and similar items. All types for parameterization are currently located in the parameterized Unit Testing Tool.

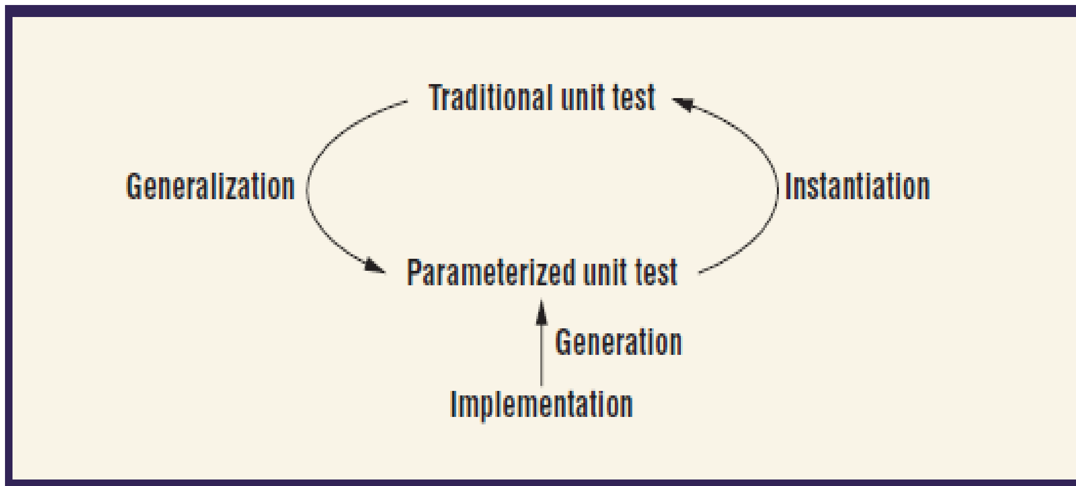

Fig .1: Connections between traditional and parameterized unit tests.

2.1 Simple Parameterization

Most tests can be written without the need of parameterizing them [9]. In some cases however, the user would like to be able to test an algorithm that takes a number of inputs and produces some number of outputs so there is a connection between the traditional test and parameterized test as fig. 1: As a very simple example, let's use the calculation of a discount as a percentage of the invoice amount:

This simple case is certainly not very thrilling, but gets the idea. Without parameterization the user would have to write four tests, one for each invoice amount.

Instead, it would be nice to refactor the obvious duplication within the tests and write a far simpler test. Parameterized tests allow the user to do exactly that.

With parameterization, the user adds parameters to a test and tells The Unit testing Tool where to read the parameter values from. To support this, by the attribute DataRow, this takes any number of parameters. The user can use the DataRowAttribute to decorate parameterized tests. Each attribute corresponds to a single execution of the test.

In essence the user has to give the test parameters and tell the parameterized Unit Testing Tool where to get the parameter values from. So introducing the attribute DataRow, this takes any number of parameters. The user can use the DataRowAttribute to decorate parameterized tests [1].

2.2 Specifying an ExpectedException

What if for some data rows the user would expect an exception to be thrown? Well, The Unit Testing Tool supports this as well, through a named property to the DataRow attribute, ExpectedException.

Note: more than one data row can have an expected exception. Also, the expected exception can be different for each of those data rows.

2.3 Parameterization with Static Method or Property

The DataRow approach is useful if the user only want to use a set of parameters once. However, in some cases the user may want to use the same set of data for more than one test. In this case the user can use the DataSourceAttribute and specify a type as a parameter for the attribute. That type is used as the data provider for the parameterization. It needs to implement a static method or a static property that returns an array of data rows [10].

This test requires a class with the name Fixture with Static DataProvider to be implemented elsewhere.

At runtime, the parameterized Unit Testing Tool will search the data provider class for a static method that returns an array of DataRow objects. It will invoke the first one it can find and use the returned array of DataRow objects as the parameter sets for the parameterized test method.

Note: the data provider class and the test fixture containing the test can be the same.

Again, if the user expects an exception to be thrown for one or more of the data rows, the user can assign the expected exception type to the named property ExpectedException of the DataRow attribute. This can be seen on the third data row above.

2.4 Parameterization with XML-File

Suppose that the user would like the parameters to be read from an XML file. The user can do this with the DataSource attribute as well [5].

And again, the user also needs to account for the possibility that one or more of the data rows expects an exception.

Note: as with the other ways of specifying an expected exception, the user can specify any type. This includes exceptions that the user has implemented.

2.5 Parameterization using a Database Table

The fourth option to provide sets of parameter values to a parameterized test is specifying a .NET data provider, a connection string, and a database table name.

This is just a standard connection string which the parameterized Unit Testing Tool passes on to the managed ADO.NET data provider. The first parameter is the Invariant Name of the .NET data provider. The factory for the data provider must be registered in the machine.config file. By default .NET has factories registered for SQL, Oracle, OLE DB, and ODBC.

During runtime the parameterized Unit Testing Tool executes the test once for each data row and reports separately on the outcome [1].

Note: accessing a database, if local, is an expensive operation. Some databases work in-memory thus at least avoiding the cross-process and/or cross-machine communication. Whether the user chooses a database table as a feed for his parameterized test requires careful considerations and trade-offs.

3. CATEGORIES

The Unit Testing Tool supports categorization of tests. Basically this means the user can assign categories to tests and test fixtures, and then use the categorization for instance for selecting tests.

If the user doesn't need this feature right now he can safely ignore it. This is one of the design principles The Tool try to follow wherever possible.

If the user assigns one or more category to a test or a test fixture, the user should be aware of the following rules:

1. If a test has no categories assigned then the default setup/teardown method will be executed. The default setup/teardown method is the one that has no categories assigned to it. If no such default setup/teardown method exists, no setup/teardown will get executed.
2. If a test has one or more categories assigned then the setup/teardown for that category or those categories will be executed. If a categorized setup/teardown method doesn't exist, the default method will be executed if it exists.
3. If more than one default setup/teardown method exists, or if more than one categorized setup/teardown method for the same category exists, this is considered to be an error and the test(s) will fail. This test is performed per test fixture. The latter means that in a hierarchy of test fixtures a base class can have a setup/teardown method and a derived class can have a setup/teardown method, either default or categorized.
4. If a test fixture is derived from a base class that is itself a test fixture, setup/teardown methods from the base class will not be considered for the derived test fixture. Also, even if a method in the base class is declared virtual and marked as SetUp/TearDown, it will not be considered by The Unit Testing Tool's runtime. If the user need to execute setup/teardown code in the base class, the user need to call the base class method from the code, e.g. base.MySpecialSetupMethod ().

Once the user has defined categories for the tests he can then select categories in the graphical user interface. When the user then save his settings as a recipe, the category selection will be save along with it. After that the user can supply the recipe to UnitCmd, e.g. for inclusion in his automated build. The category selector is part of The Unit Testing Tool's runtime environment regardless of the front end.

At the end comparing the unit testing tool with other famous open source testing frameworks as in table [1].

Table 1. Comparison of testing frameworks.

Feature	Unit Testing Tool	TestNG	Jtest
Test classes extend framework class or implement interface	yes	no	yes
Test method discovery	Reflection	Annotations , Javadoc	Reflection
Test setup methods	Naming conventions	Annotations , Javadoc	Naming conventio ns
Test case selection	Program code, XML,via GUI	XML and annotations, Javadoc	Via GUI
Test and configuratio n method parameters	Via GUI	XML and annotations, Javadoc	Via GUI
Automatic test case generation	yes	no	yes
Support for generating stub and mock objects	With MockObject s or EasyMock libraries	With MockObject s or EasyMock libraries	yes

4. CONCLUSIONS

This paper presenting the concept of parameterized unit tests, a generalization of established closed unit tests. Parameterization allows the separation of two concerns: The specification of the behaviour of the system, and the test cases to cover a particular implementation.

The tool introduced in this paper is a parameterized unit testing framework for the .NET Framework. It is designed to work with any .NET compliant language. It has specifically been tested with C#, Visual Basic .NET, Managed C++, and J#.

The Tool follows the concepts of other parameterized unit testing frameworks in the XUnit family [1, 6]. Along with the standard features, the tool offers abilities that are uncommon in other parameterized unit testing frameworks for .NET:

- Categories to group included, excluded tests
- ExpectedException working with concrete instances rather than type only
- A tab for simple performance base lining
- A very rich set of assertions, continuously expanded
- Rich set of attributes for implementing tests
- Parameterized testing, data-driven testing
- Search abilities, saving time when test suites have thousands of tests.

5. REFERENCES

[1] Tillmann, N.; de Halleux, J.; Tao Xie, Parameterized unit testing: theory and practice, Software Engineering, ACM/IEEE 32nd International Conference, Volume: 2, Page(s): 483 - 484, 2-8 May 2010.

[2] Narendra Kumar Rao, B.;Rama Mohan Reddy, A. ; Ravi, K. , Level dependencies of individual entities in random unit testing of structured code, Electronics Computer Technology (ICECT), 2011 3rd International Conference, Volume: 6, Page(s): 223 - 226 , 8-10 April 2011.

[3] Runeson, P., A survey of unit testing practices, IEEE Journals & Magazines, Volume: 23, Issue: 4, Page(s): 22 - 29, July-Aug. 2006.

[4] Williams, L.; Kudrjavets, G.; Nagappan, N., On the Effectiveness of Unit Test Automation at Microsoft, Software Reliability Engineering, ISSRE '09. 20th International Symposium, Digital Object Identifier: 10.1109/ISSRE.2009.32, Page(s): 81 – 89, 2009.

[5] Cheng-hui Huang, A semi-automatic generator for unit testing code files based on JUnit, Systems, Man and Cybernetics, 2005 IEEE International Conference, Page(s) 140 - 145 Vol., 10-12 Oct. 2005.

[6] Tao Xie; Taneja, K.; Kale, S.; Marinov, D., Towards a Framework for Differential Unit Testing of Object-Oriented Programs, Automation of Software Test, 2007. AST '07. Second International Workshop, 20-26 May 2007.

[7] Gupta, A., Jalote, P., Test Inspected Unit or Inspect Unit Tested Code? , Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium, Page(s): 51 – 60, 20-21 Sept. 2007.

[8] Bin Xu, Towards Efficient Collaborative Component-Based Software Unit Testing via Extend E-CARGO Model-Based Activity Dependence Identification, Intelligent Ubiquitous Computing and Education, International Symposium, Page(s): 172 – 175, 15-16 May 2009.

[9] Vegas, S.; Juristo, N.; Basili, V.R., Maturing Software Engineering Knowledge through Classifications: A Case Study on Unit Testing Techniques, Software Engineering, IEEE Transactions, Volume: 35, Issue: 4, Page(s): 551 – 565, Digital Object Identifier: 10.1109/TSE.2009.13,2009.

[10] Na Zhang;Xiaoan Bao; ZuohuaDing,Unit Testing: Static Analysis and Dynamic Analysis, Computer Sciences and Convergence Information Technology, 2009. ICCIT '09. Fourth International Conference**,** Page(s): 232 – 237, 24-26 Nov. 2009.

[11] Liangliang Kong; Zhaolin Yin, the Extension of the Unit Testing Tool Junit for Special Testings, Computer and Computational Sciences, IMSCCS '06. First International Multi-Symposiums, Volume: 2, Page(s): 410 – 415, 20-24 June 2006.

[12] Mouy, P.; Marre, B.; Williams, N.; Le Gall, P., Generation of All-Paths Unit Test with Function Calls , Software Testing, Verification, and Validation, 2008 1st International Conference, Digital Object Identifier: 10.1109/ICST.2008.35, Page(s): 32 – 41, 2008.