FRAMEWORK TO SUPPORT THE ASSESSMENT OF PROGRAMMING LANGUAGES STUDENTS' PRACTICAL WORK

Zenón Hernández-Figueroa[1], Margarita Díaz-Roca[2], Juan Carlos Rodríguez-Del-Pino[3], Francisco Javier Carreras-Riudavets[4]

[1, 2, 3, 4] *Departamento de Informática y Sistemas. Universidad de Las Palmas de Gran Canaria. Grupo de innovación educativa TILDE. (ESPAÑA)*
zhernandez@dis.ulpgc.es, mdiaz@dis.ulpgc.es, jcrodriguez@dis.ulpgc.es, fcarreras@dis.ulpgc.es

Abstract

The introduction and use of New Technologies in the learning field not only does imply that the process should be structured, planned and done according to the more adequate criteria. On the one hand, we must work on the educative contents, their creation process, features and objectives because these contents are, with no doubt, an essential part in the process that will lead us to increase the quality of learning. On the other hand, these new technologies should be used for something else, like introducing new elements that help in the teaching-learning processes.

A Learning Management System (LMS) is a software application accessed through a web server that, making use of web technology, manages and broadcasts educative content to support the training that takes place in the classroom and the distance learning activities. A Learning Content Management System (LCMS) is a management system used to teach, like the LMS. Its web technology allows users to create and manage the contents of a learning program.

The most appropriate form of making use of these kinds of systems is to create content modular pieces which could be personalized, managed and reused.

A LMS can broadcast different types of educative material: text files, slides, animations, audio and/or video files or even software applications. It can manage the access to these contents, the work done with the applications and the assessment of this work.

Programming is one of the essential areas taught in university studies of Computer Science and other engineering degrees, as well as in diplomas of Computer Science. At present, it is a knowledge acquired through tutorial lessons and the practice with different tools for programming such as compilers, debuggers, interpreters, among others.

In computer programming the unit testing is a method used to ascertain if individual pieces of source code work as they should, that means, that for specific input data the output data are correct.

The framework introduced in this work is designed to offer lecturers a new tool to develop units testing with the specific aim of supporting the assessment of programming languages students' practical work. The modules are developed for Ada programming language. This tool is especially useful when it is combined with a web platform to manage the work of the students in these subjects. Its main features are the flexibility and ease of use. Consequently, it is really useful to have a homogeneous framework that allows developing systematically a huge amount of tests.

These tools form a powerful application which greatly lightens the students work load at the initial stage of programming. During this initial period they will neither have to deal with the complexities of the installation and the configuration of these types of tools, nor with the understanding of multiple options which they present, therefore being able to concentrate on the comprehension of the programming structures and the programming language to be studied.

This paper exposes the experience acquired in the creation of software modules that configures a framework to create software unit testing. They have been developed for subjects of programming languages of various official degrees from the University of Las Palmas de Gran Canaria.

Keywords: On-line teaching tools, e-learning, programming languages, learning management systems, moodle.

1 INTRODUCTION

The University of Las Palmas de Gran Canaria (ULPGC) uses Moodle as Learning Management System for all its official courses. Moodle is a Open Source[1] Course Management System freely available and widely used for both online and face-to-face courses. It offers a wide range of modules to manage different kinds of learning tasks and anybody can develop a new module to fit a new task.

The achievement of computer programming skills requires a lot of training by means of computer programming assignments. The assessment of these assignments can require a long time and effort by the teacher, so it is hard to give a continuous assessment and adequate feedback to the students, which is essential to get a good learning. Availability of tools to assist and support the assessment of computer programming assignments can be very useful to improve the learning process, by reducing the time employed in feedbacks, among other advantages.

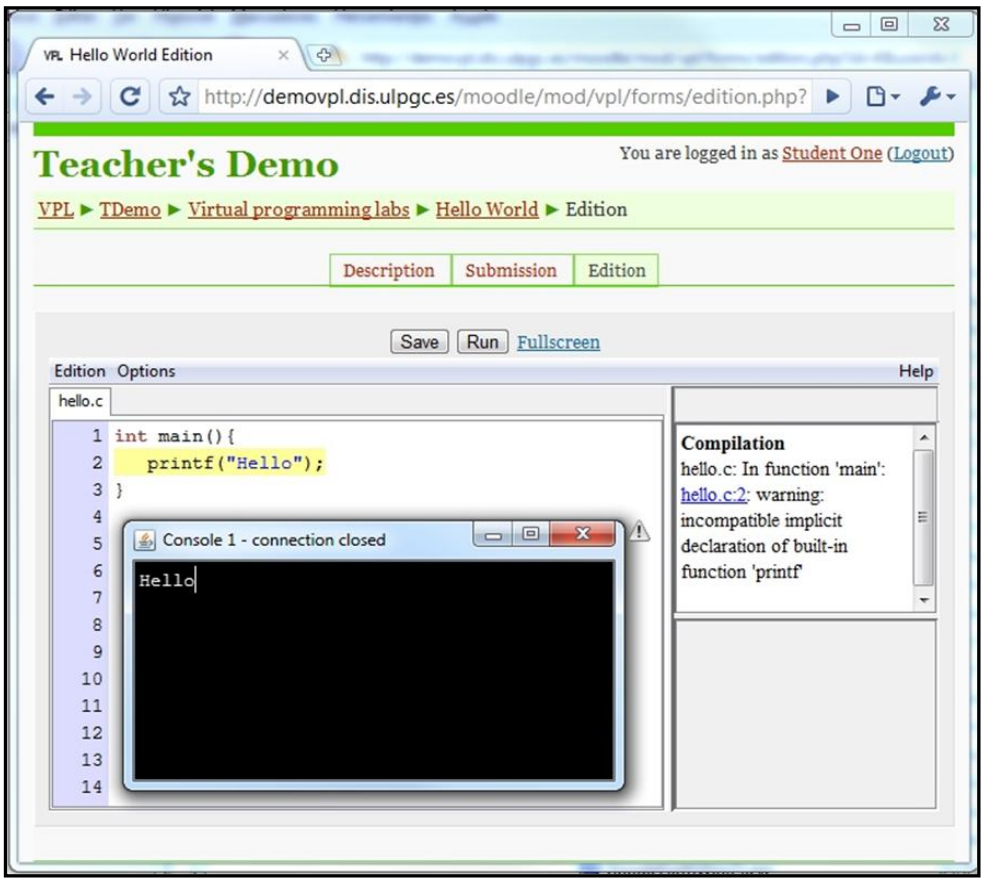

Fig 1: VPL's student's screen.

Recently, it has been presented a new Moodle's module named VPL [1] (standing for Virtual Programing Lab) developed at ULPGC. The “Fig. 1”, shows VPL's student's screen. VPL is designed to manage, test, assess and support grading of programming assignments, being independent from the programming language used to develop such tasks. This language independence relies on the ability of the teacher to establish proper tests for the code submitted by the students.

When they are guided by educational goals, code tests must not only detect program's bugs, but also produce useful feedback to improve the learning process, as well as propose a grade that must be validated by the teacher.

A popular framework to code tests is the xUnit family of unit test frameworks, being JUnit [4] for Java one of its best-known representatives and AUnit [5] is the one for Ada programming language. Unfortunately, the xUnit family is designed for general purposes and do not meet the requirements listed above. So it is necessary to deeply extend the xUnit frameworks to fit the learning requirements,

[1] Open Source. Model of development and delivery of software applications. This model promotes free access to the source code.

or develop a new kind of framework having the desired features. The simplest solution depends on the specific features of the xUnit for each language.

At the School of Informatics Engineering of the ULPGC, we use Ada as first programming language for our undergraduate courses; the fitness of Ada for this purpose is supported by our experience and the one presented by other authors [2] [3].

This paper presents a framework developed to accomplish the requirements of testing for the kind of pieces of code used to facilitate the students their first approach to programming language discipline. It has been designed to be very simple to use, and it mainly consists of three Ada's packages whose features and functionality are described in the next sections.

2 DESCRIPTION OF THE SOFTWARE PACKAGE USED IN THE TESTS.

The test system is based on three different packages: *Report_Unit, Base_Test_Case and Generic_Test_Unit.*

2.1 Report_Unit package.

The *Report_Unit* package provides the *Report Class,* which represents a *Results Report* of a test set. This report is organized in a set of items that represents the test results. Each item consists of: a descriptive header that must be unique; a set of comments that describes the errors detected in the tests related to that item; the penalty –accumulated for these errors-, on the exercise assessment; and the maximum penalty applied in that item. The "Fig. 2", shows an item as it is shown to the teacher by the *Report Class*. The item description is included, as well as the applied penalty (-4,00 points) and three comments that explain the errors found for that item. The maximum item penalty is not shown.

Case: To reverse a linked list with more than one node. (-4.00)

- ***Comment>>*** The list is not the one expected, is: L→ 5→ 4→ 3→ 5→ 3→ 2→null.
  But should be: L→ 2→ 3→ 5→ 3→ 4→ 5→null.
- ***Comment>>*** Unnecessary nodes are created.
- ***Comment>>*** Some nodes are destroyed unnecessarily.

Fig 2: Example of an item.

Report Class provides methods in order to: create an item (*Create_Item*), for this class it is necessary to specify its header and maximum penalty value; add comments to an item (*Add_Comments*), with a penalty that increases the one of the item until the maximum value is reached; and show the report (*Show*). This report is shown in the console as plain text with format marks used to be interpreted by VPL. Besides, *Report* class provides the *Merge* method that allows mixing two reports in a new one which gathers the contents of the original reports, weighing the assessment according to a previous specification. This is useful to make a global assessment based on the combination of independent tests.

A special method, *Report_Internal_Error*, is used to warn about the possibility of errors in the tests system.

2.2 Base_Test_Case_Unit package.

Base_Test_Case_Unit package provides the *Base_Test_Case Class* which represents basic information considered common to any test case. This includes: a description (*Description*), the discount to apply to the assessment if the exercise fails for that case (*Penalty*), the waiting time before stopping the case execution (*Time_Out*), information indicating if an exception should be raised when executing the case (*Waited_Exception*); information indicating if an exception is raised when executing the case (*Raised_Exception*), and the output data that should show the console (*Output*). For the information described above, both the update and consult methods are given.

When deriving the *Base_Test_Case* class to represent a specific test case, information about the test data, which must be passed to the subprogram, and the results expected, will be included. Once the subprogram to test is executed, the results obtained will be compared to those expected.

Base_Test_Case Class also provides both methods: *Report_Exception* and *Report_Output_State*. These methods are used to include information about the exceptions frequency and the output shown

in the console when executing the test case. For this a parameter of the *Report Class, among others,* is needed.

Likewise, it provides three different abstract methods which are key to the tests system as they are defined for each particular test case. The first one, *Load*, is used to load the case data input from a file whose name is passed as a parameter. The second one, *Execute*, is the one invoked to execute the test case. This method must be defined when deriving the class and must include the call made to the subprogram to test. And finally the third one, *Analyze,* which is invoked once the test is finished in order to create its report in a parameter of the *Report Class*.

Finally, there exists an order function, identified with the operator "<", which establishes an order among the *Base_Test_Case* class objects. This order is based on its description. The order function can be redefined in the derived classes.

2.3 Generic package Generic_Test_Unit.

The generic package *Generic_Test_Unit* has as generic parameters ("Fig. 3"): one class, Test_Case and an order function "<". The class derives from *Base_Test_Case* and represents the test case class to be executed. The order function in this class is identified by the "<" operator.

```
generic
   type Test_Case is new Base_Test_Case with private;
   with function "<" (Left, Right : Test_Case)
   return Boolean is <>;
package Generic_Test_Unit is
```

Fig 3. Generic_Test_Unit header.

The generic package *Generic_Test_Unit* provides the *Tester Class*, the one in charge of the tests execution. *Tester Class* stores a set of cases of *Test_Case Class*.

Tester Class has:

Add, a method to add a test case to a pre-established test case set.

Load, a method to load a test case set from a file, this is made by calling the Load method of the Test_Case Class for each case.

Reset, a method to reset (clear) the test data once the tests are performed.

Test, a method to execute the tests.

The *Test* method makes the iteration by the set of test cases, according to the order previously defined by the *orden* function. It executes the tests for each case and generates a report (contained) in a parameter of the *Report Class*.

For each case:

It creates an item in the tests report, on which the comment lines of the test case will be added.

It executes the test case by making a call to the Execute method of Test_Case Class, printing the screen in the console as well as the exceptions produced.

If there are not exceptions that avoid the call, it will invoke to the Analyze method of Test_Case Class in order to analyze the results obtained in the test execution and to complete the item of the corresponding report. This means including the adequate comments to report about the errors and making a proposal assessment.

The *Test* method includes a parameter: *Verbose*, which activates the additional information displayed on each stage of the test execution.

2.4 Other useful packages.

In addition to the packages and classes described above, there exist others which examine different aspects that are outside the functioning of the subprogram and that are considered important from a teaching point of view.

The package called *Estilo* provides operations in order to analyze the programming style and the comment density on the subprogram. At present the style analysis is limited to the use of GNAT[2] compiler, since it offers an option to make studies of the level of adjustment of the programming code to compile. This code uses the recommendations of GNAT Style manual and, this way, the function that analizes the style has to study only the result of the compilation in order to generate the corresponding assessment. The analysis of the comment density is made following an heuristic approach that assures that all the modules (instruction blocks) with a minimum extension have the adequate proportion of comment lines, within a pre-established range.

3 TEST PREPARATION.

The first step in a test preparation is to extend the *Base_Test_Case Class* in order to include specific information of the case. The data input are passed to the subprogram call as well as the expected results must be defined. Besides, the methods *Load*, *Execute* and *Analyze* must also be defined. The first one requires a format for the data input representation; these data will be stored in a file. The second one will include, apart from the call to the subprogram, the actions which could be performed before and after this call, in order to be correctly executed. Finally, the third one will define the way in which the results obtained must be compared as well as the conclusions of this comparison. “Fig. 4” shows an example of *Base_Test_Case Class* extension. This corresponds to the test of a subprogram that counts the number of nodes in a binary tree.

```
type Test_Case is
  new Base_Test_Case with record
   Tree_To_Test : Árbol;
   Result       : Natural;
   Expected     : Natural;
end record;
```

Fig 4. Base_Test_Case extention.

The main subprogram of a test program using the proposed scheme is quite simple. “Fig. 5” shows an example, which starts with a context clause that includes the test packages described above. In the declaration code the *Generic_Test_Unit* package is instanced with the class used in the test case to be executed (*Test_Case*) and both variables are declared: one from *Report Class* and the other one from *Tester Class*. The subprogram’s body only has five instructions: calls to the style and comments analysis operations; the loading of tests case from the file *pruebas.txt* (call to the *Load* method), the execution of tests and the creation of the corresponding reports (call to the *Test* method), as well as the display of the tests report (call to the *Show* method). In the last part an exception control is established. This one is developed in case that any kind of error in the system test appears (like not being able to access the file with the input data tests)

[2] GNAT. Compiler for software written in Ada programming language. Is free-software distributed under the GNU General Public License.

```
with Report_Unit;    use Report_Unit;
with Test_Case_Unit; use  Test_Case_Unit;
with Generic_Test_Unit;

procedure Probar_Num_Nodos is
   package Test_Unit is new
     Generic_Test_Unit (Test_Case);
   use Test_Unit;
   Node_Count_Tester : Tester;
   Node_Count_Report : Report;
begin
   --  Style Analisys

   GNAT_Style ("compilado.out", Num_Nodos_Report);

   Comments ("Node_Count.adb", Num_Nodos_Report);

   --  Subprogram test

   Num_Nodos_Tester.Load ("tests.txt");
   Num_Nodos_Tester.Test (Node_Count_Report,
                          Verbose => True);

   --  Test results

   Show (Node_Count_Report);
exception
   when E : others =>
      Report_Internal_Error
        (Num_Nodos_Report,
         Exception_Name (E) &
         " with " &
         Exception_Message (E));
end Probar_Num_Nodos;
```

Figure 5: Test program example.

4 PRACTICAL EXPERIENCE.

Computer Science Systems and Computer Science Management Technical Engineering degrees as well as Computer Science Engineering bachelor degree belong to the ULPGC since 1997. In these degrees, the first subject that teaches programming languages is *Programming Methodology I*. In this subject Ada is used as programming language.

In the first semester of 2009-2010 200 students from these three degrees attended the subject *Programming Methodology I*. A total of 27 exercises were proposed. These exercises consist of the writing of small subprograms to fix simple problems related with each topic of the subject. Among them we find: input/output, character strings, arrays, files or linked lists.

The exercises delivered were submitted to automatic review, using the framework presented. The results are immediately reported to the students. For the same exercise more than one deliver can be done, within a previously established period of time. This way the students can benefit of the information provided with the result of the tests in order to improve the exercise.

For the assessment of each exercise, an average of 39.4 tests case were prepared. The students performed an average of 10.21 deliveries for each exercise with a total amount of 55159 deliveries. This means the execution of 2173264.6 automated test.

This has lead to a very important reduction of the teacher's work load in comparison to the previous courses, where the tests case demanded a volume of work that almost tripled the work of the actual course. It has been shown, in the creation of a larger number of tests case, that it does not allow as much error possibilities as in the exercises assessed previously.

5 CONCLUSIONS AND FUTURE LINES.

In this work we expose a test package set written in Ada programming language. These test packages allow developing subprogram tests written in this language easily. These tests are developed for the assessment of practical exercises of the students of programming languages subjects.

The simplicity of the system provides the support with completely operational tests which only require the extension of some classes and the definition of some specific methods.

An important feature of this system is modularity. Data tests and code are separated. Data test are stored in a file, this way they can be added, or different tests case can be modified without needing to rewrite the programming code.

Not only does the system perform tests based in data input parameters and the expected results, but it is also able to analyze the results displayed and to control the proper use of the exceptions.

The report of the tests result, which includes a proposal assessment, is generated using a format that is compatible with the practical exercises management used in VPL. In the future the formats will be extended to others, like XML[3], a new option also compatible to be used in VPL.

The educative resource exposed is developed to work with Ada programming language, but the philosophy followed in its development can be easily implemented in other programming languages object-oriented. Nowadays we are working in the analysis of JUnit[4], in a deep research on the possibilities to apply this philosophy as an extension of this unit. Another possibility is to make a development as a specific framework. In the same way it is being analyzed for C++.

This framework makes the use of automatic practical exercises assessment remarkably easy, helping to make the most of all its features. This web platform accessible from the Internet, will provide programming languages students with an easier and quicker way to learn on each stage of the process. Teachers can dedicate more time to activities for a best quality in teaching processes, providing motivation for the docent contents, the results, and guiding the students in a more personalized way.

The use of the compiler through the web is an added advantage. This way, students can use it regardless of time and place. Students can use the environment, the docent contents and the work made with no need to use neither storage devices nor software installations [6].

The possibility to have remote virtual labs increases their efficiency, and reduces the cost of the resources used in the teaching processes [7].

In the teaching field, the education routes are scheduled by teachers with the aim that each student can follow their own learning process.

This is made according to the objectives of the subject, its contents, the students' profile, the virtual environment where they are executed, and the different resources used [8].

Another of the research lines concerns the introduction of education routes in VPL. Students can follow the lessons and exercises with a time planning adapted to their needs. This way it will be easier to design learning processes [9] for those lecturers with students that attend their courses and who belong to different levels and present different learning needs.

The education routes could be implemented according to a collection of rules whose data input are: the time planning, the different tools used, the activities proposed and the assessment. The decision-making process in this module is developed taking these data input and the information stored in the VPL database obtained through data mining techniques[5] [10].

[3] XML. Is an eXtensible Markup Language. A set of rules for encoding documents, developed by W3C (World Wide Web Consortium).
[4] JUnit. JavaUnit. Unit testing framework for Java programming language. It is used in the development of test-driven.
[5] Data mining techniques. I a set of techniques used to extract patterns from data.

ACKNOWLEDGMENT

This work was been supported by the “Quality and Educative Innovation Vice-chancellorship” from the ULPGC. The project, LEARNING OBJECTS has been developed by the educative innovation group TILDE.

REFERENCES

[1] Juan Carlos Rodríguez; Enrique Rubio; Zenón J. Hernández. *VPL: Laboratorio Virtual de Programación para Moodle*. Actas de las XVI Jornadas de Enseñanza Universitaria de Informática, Santiago de Compostela, Spain, July 7-9, 2010, pp 429-435

[2] Debora Weber-Wulff. *Ada as a first Language*. In: Entwicklung von Software-Systemen mit Ada. Ada-Deutschland Workshop, Bremen, 1998.

[3] Jesús J. García Molina; Francisco J. Montoya Dato; José L. Fernández Alemán; Mª José Majado Rosales. *Una introducción a la programación. Un enfoque algorítmico*. Paraninfo 2005. ISBN: 9788497321853.

[4] E. Gamma and K. Beck. *JUnit*. At http://www.junit.org.

[5] Ed Falis. *Aunit, Ada unit testing framework*. At http://libre.adacore.com/libre/tools/aunit/.

[6] Guerra, C., Afonso, M.D., Santana, I., Quesada, R., *OLC, On-Line Compiler to Teach Programming Languages*. International Journal of Computers, Communications & Control. Vol. III (2008), No. 1, pp. 69-79

[7] Lawson, E.A. Stackpole, W. (2006). IT education - online education: *Does a virtual networking laboratory result in similar student achievement and satisfaction?*. Proceedings of the 7th conference on Information technology education SIGITE’06. (October 2006). ACM Press.W.

[8] C. Gráinne, 2007, *Making sense of today’s technology-enhanced environment for learning: rethinking student and teacher roles*. IADIS International Conference e-Learning.

[9] Horton W.K., 2006, *E-Learning by design*. ISBN-13: 978-0-7879-8425-0. ISBN-10: 0-7879-8425-6.

[10] J.M. Domenech, J. Lorenzo, 2007, *A Tool for Web Usage Mining* 8th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL’ 07) 16-19 December, 2007 Birmingham, UK.