

# A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features

Juan C. Rodríguez-del-Pino, Enrique Rubio-Royo, Zenón J. Hernández-Figueroa

Departament of Informática y Sistemas  
University of Las Palmas de Gran Canaria  
35017 Las Palmas de Gran Canaria, Spain

**Abstract** - This paper describes VPL, a Virtual Programming Lab module for Moodle, developed at the University of Las Palmas of Gran Canaria (ULPGC) and released for free uses under GNU/GPL license. For the students, it is a simple development environment with auto evaluation capabilities. For the instructors, it is a students' work management system, with features to facilitate the preparation of assignments, manage the submissions, check for plagiarism, and do assessments with the aid of powerful and flexible assessment tools based on program testing, all of that being independent of the programming language used for the assignments and taken into account critical security issues.

**Keywords:** programming learning, virtual lab, automatic assessment, plagiarism.

## 1 Introduction

Achievement of computer programming skills requires a lot of training by means of real program-development assignments. Managing and assess the students' submissions for those assignments could be a very complex task. Availability of tools to organize the assignments, receive and storage the submissions, support automatic or semi-automatic assessment and provide feedback could be very helpful. Furthermore, integration of that kind of tools into a Learning Management System is an essential feature in order to improve their performance.

Some tools for this purpose have developed along the time. STYLE [1] and CAP [2] focus on the automatic evaluation of program's style and syntax. PASS [3] focus on assessment capabilities. Some tools focus on execution for a specific programming language, as PACER [4], for ELI, a language similar to C, or HoGG [5] and ELP [6] for Java. More general tools focus on course management features, as Ceilidh [7], CourseMaster [8] and Boss [9].

This paper describes VPL, a Virtual Programming Lab module for Moodle, developed at the University of Las Palmas of Gran Canaria (ULPGC) and released for free uses under GNU/GPL license. Downloading of VPL is available at <http://vpl.dis.ulpgc.es>; also an on-line demo site is available at <http://demovpl.dis.ulpgc.es>. Figure 1 shows the VPL homepage.

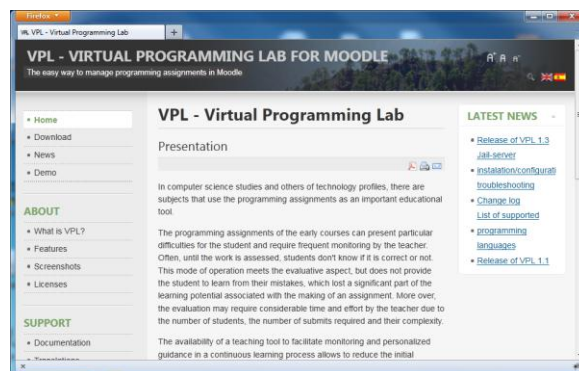


Figure 1. VPL homepage

VPL is designed to achieve the following goals:

- To be an open source tool, freely distributable and capable to be enriched with external contributions. For this reason it is distributed under GNU/GPL license.
- To be independent of the programming language. To use a particular programming language it is only required that the appropriate compiler is installed in the jail system (see section II for details). Currently, there are available jail systems with installations for Ada, C, C++, C#, FORTRAN, Haskell, java, Octave, Pascal, Perl, PHP, Prolog, Python, Ruby, Scheme, SQL and VHDL. Any user can install other languages if required.
- To provide a very simple development environment in order to smooth the learning curve to the beginners.
- To provide tools to support automatic and semi-automatic assessment, including tools for plagiarism checking.
- To be conscious of security issues. To avoid security breaches when executing students' code, all executions are performed in a separated and restricted environment (the above mentioned jail system).

Following sections describe the VPL architecture, the types of activities that can be done using VPL, how to

configure those activities, the use of VPL to assess submissions and check for plagiarism, and our experience as VPL users along the test phase of the module.

## 2 VPL architecture

### 1.1 General features

VPL is composed of three elements: a Moodle's module, a browser-based code editor and a jail component (Fig. 2).

The code editor is a java applet providing basic features to edit, run, debug and evaluate programs code in a very simple code-development environment. To use full features a web browser with JavaScript and support for Java 1.5 applets is required.

The Moodle's module provides the typical features of this kind of component (backup and restore, integration with the grading book, course reset, events control, role-based access ...) but also specific features such as: submission management, assessment support and anti-plagiarism features. The VPL module requires a Moodle version 1.9.x and PHP5 or higher.

The jail component is the server in charge of compile and executes the code submitted by the students in a secure environment. It runs a linux chroot command to provide a restricted version of the host file system with some read-only limitations. To run or evaluate a submission is required to have at least one jail server. The jail service needs Ubuntu (recommended) or Red Hat compatible linux distribution..

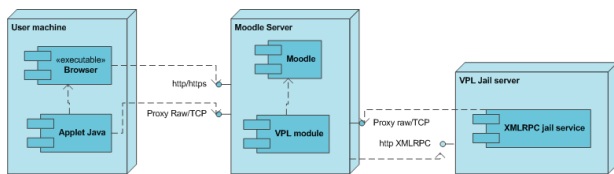


Figure 2. VPL components

The jail server attends requests for both interactive and non-interactive executions (Fig. 3), the difference between them is that the second type requires to the request data include a key, a server and a communication port, which are used to redirect the execution input/output data.

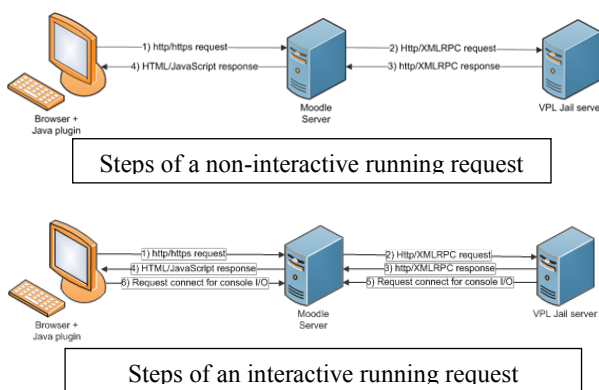


Figure 3. Types of execution request

To provide execution in console the Moodle servers need to open at least two ports, a larger number is recommended. It takes more time than usual to execute the PHP scripts that run a submission so it is necessary increase time limit in the PHP configuration.

### 1.2 Network topologies

The VPL module uses a double proxy to communicate, by a side with the Internet clients, to attend their requests, and by other side with the jail servers, to perform the running tasks associated to those requests. This permits a variety of network topologies. The simplest topology runs both the jail server and the Moodle server in the same computer, although they have to communicate via an intranet. This solution loses the security advantages provided by the isolation of the servers on different computers. A more suitable topology joins a Moodle server with one or more separate jail servers, which may be in a private network.

A more powerful topology, that improves the resources spending, is to share multiple jail servers among multiple Moodle servers (Fig. 4). This configuration may adapt itself to workload peaks by changing the number of jail servers in use by a Moodle server in order to attend in a proper way the variations on the requirements of the execution tasks. The drawback of this configuration is that the jail servers must be in a public domain in order to make them available to all the Moodle's servers without increasing the network complexity.

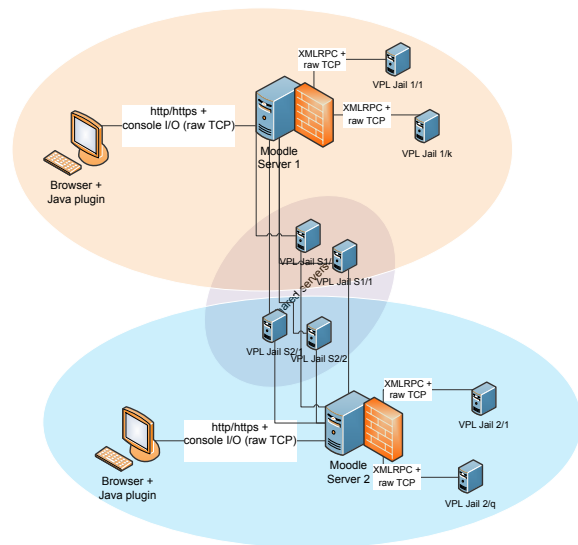


Figure 4. A complex net topology

Using multiple jails serves not only support scalability and improves performance, but also provides fault tolerance. When an execution request is received by the VPL module, it takes the list of available jail servers and randomly selects one that is not marked as having a previous fail into a specific range of time. Then VPL sends the server an availability request; if the response to this request is true, the execution request is assigned to the server, else a new server is selected. If no server is found, the process is repeated taken into account the servers previously failed.

### 3 Types of VPL activities

VPL can be used to configure, manage and assess a range of learning activities which can be classified by type or scope [10]. By type, the learning activities may be: examples, cloze or puzzle exercises, and code development exercises. By scope they may be: out-classroom tasks, or in-classroom exams.

#### 1.3 Learning activities by type

Examples are activities where the students are provided with both the description of a problem and the program code that solves that problem. The students may interact with the code (running or debugging) to see how it works.

Examples may be mutable or immutable, depending on if the student can or no modify the code. An immutable example must be marked as "example" in the activity's configuration window. The student who modifies the code of a mutable example can always reset it to its original state.

Cloze exercises and puzzle exercises are especially appropriate for beginners. In the same way than the previous activity, they provide the students with the description of a problem and the program code that solves that problem, but now the student must modify the code.

In the case of the cloze exercises, portions of the code have been deleted and the student has to fill in the blanks in order to do the code works in the appropriate way.

In the case of the puzzle exercises, the code has been disordered and the student has to sort it in order to do the code works in the appropriate way.

A code development exercise provides the students only with the description of a problem (although some code may be sometimes included). The student has to develop the code to solve the problem using the appropriate techniques. This kind of exercises is a traditional way to achieve programming skills at any level by means of an intensive training in solving problems, in the believe that: 'doing programs is how we can learn to programming'.

#### 1.4 Learning activities by scope

Attending to the conditions under what a VPL's activity can be done; there are two ways to use VPL: out-classroom tasks, or in-classroom exams.

Out-classroom tasks are designed as long-term activities to be done into a time period that may extend by days or weeks. During the active period of the activity the student may try so many solutions as he or she likes.

Out-classroom activities can be done anywhere, without direct supervision of an instructor, although help can be obtained from the instructors as long as they offer it in some way (e-mail, forums, face-to-face...).

In-classroom exams are activities designed to be done in a short-term and in a restricted environment, under instructor's supervision. So, they can be configured to require a password, be done in a specific local area

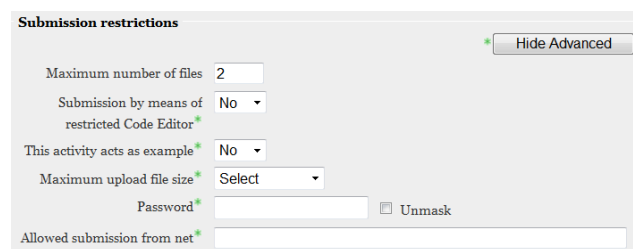
network, or do not permit the editor's copy and paste features.

As for our-classroom activities, multiple attempts of solution could be tried. In both cases, the student can get immediate feedback for each try, if so configured.

### 4 Configuring VPL activities

#### 1.5 Basic configuration form

The creation of a VPL activity begins by filling its basic configuration form, which includes, as for majority of Moodle's modules: activity's name, short description, availability period (with visibility and due dates), grading options, grouping... In addition, an VPL activity may include data as: maximum number of files to submit, maximum size of those files, restrictions on editing, network, password... (Fig. 5).



The screenshot shows a form titled "Submission restrictions" with a "Hide Advanced" button. The form contains several fields: "Maximum number of files" (text input with value 2), "Submission by means of restricted Code Editor" (dropdown menu with value No), "This activity acts as example" (dropdown menu with value No), "Maximum upload file size" (dropdown menu with value Select), "Password" (text input with an "Unmask" checkbox), and "Allowed submission from net" (text input).

Figure 5. VPL basic configuration form restrictions

When the basic configuration form is completed, the instructor can configure five other groups of features: full description, tests cases, options, requested files and advanced features.

#### 1.6 Full description and test cases

The Full description tabsheet shows a html editor where the description of the problem to solve must be written to be shown to the student.

The Tests cases tabsheet permits configure simple input-output based test cases to check programs correctness in order to elaborate an assessment reports both to provide feedback to the students and to support grading. The configuration of each test case must include: case short description, input to test the program, expected output for that input, and grade reduction when the test case fails (Fig. 6).

```
vpl_evaluate.cases
1 case = Divisible por 4 pero no por 100
2 input = 2012
3 output = 2012 es bisiesto
4 grade reduction = 25%
5 case = Divisible por 4 y por 100, pero no por 4
6 input = 1900
7 output = 1900 no es bisiesto
```

Figure 6. Example of test case configuration

#### 1.7 The Options tabsheet

The Options tabsheet (Fig 7) serves to configure some general options for execution of submissions tests. The first feature in the Options tabsheet serves to indicate if

the activity is based on another VPL activity, on this way it is possible to build a hierarchy of activities which share some features via heritage.

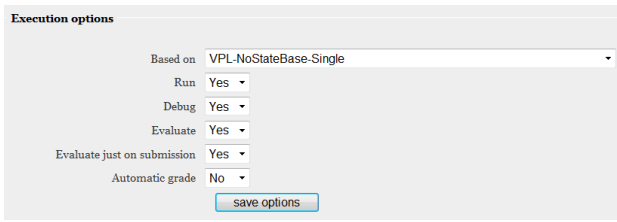


Figure 7. Options tabsheet

The rest of the options in the Options tabsheet are for configuring if the students can run, debug or evaluate their submissions, if the exercises will be evaluated just when submitted, and if the results of the automatic evaluation will become in the final assessment or they will be used only as a help for a human evaluator.

### 1.8 Requested files

The Requested files tabsheet (Fig. 8) serves to put obligatory names for the files that the student have to submit to complete the activity. The max number of files was put in the basic configuration form. The student can submit any number of files, until the max, with any names, but if names for files are specified in the Requested files tabsheet, the student must submit at least those files.

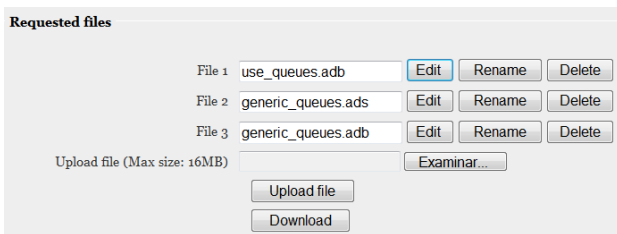


Figure 8. Requested files tabsheet

The instructor may provide initial contents for the specified files, by example, to configure a cloze exercise.

### 1.9 The advanced tabsheet

The configuration options described in the previous sections suffice to configure an activity with an useful test system. The Advanced tabsheet serves, among other things, to configure a more powerful test system.

The basic tests are black-box tests based on console input-output that check for functionality of the submitted code. Using the advanced options we can configure more powerful functionality tests, based on unit testing frameworks than can be similar, for example, to the well-known JUnit for Java [11]. In addition to the unit tests, we can also configure other types of test like style checking or coverage tests, so producing an assessment report that takes into account not only functionality, but many other parameters linked to the code quality. To configure the advanced tests we can include some files (Fig. 9) to be joined with the submitted files to prepare the tests. We must also configure some script files that will control the

execution, debugging and evaluation (running the tests) of the submissions.

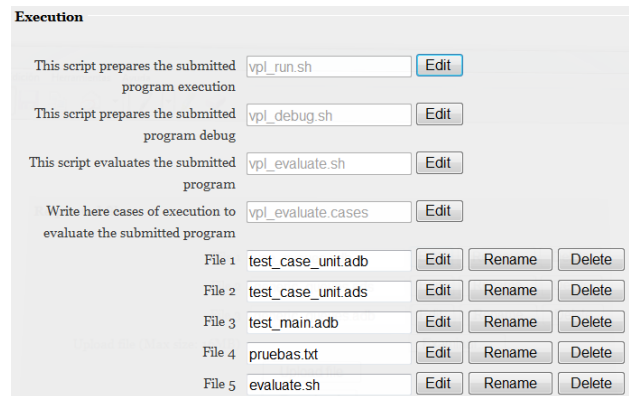


Figure 9. Execution files for advanced testing

The Advanced tabsheet includes also the configuration of other features (Fig. 10) which usually do not need modification, such as limits for execution (time, memory and disk use) or designation of specified jails for the activity.

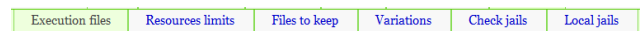


Figure 10. Advanced options tabsheet

## 5 Assessment support

VPL offers both automatic and computer-aided assessment. Selection of one of them is made by checking the appropriate option in the Options tabsheet, as described above.

The key for the assessment support is the configuration of the program tests; this can be made by simply listing a set of input-output tests cases or by configuring a more complex tests framework.

VPL automatically executes the configured tests and produces a report that includes a list of failed tests, with explanatory comments, and a grade proposal. This report could be used in three ways: to provide feedback to the students while they are developing their solutions to the exercise (formative assessment), to produce the final assessment if automatic evaluation is configured, or to help a human evaluator to assess the submission.

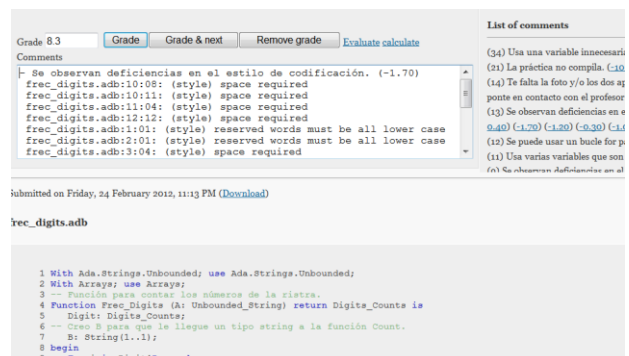


Figure 11. Evaluation window

When evaluating a submission the human evaluator uses a window (Fig. 11) where appears the assessment report produced by the test system and a list of the comments added to other submissions previously evaluated. The human evaluator can modify the report deleting comments, adding new comments or reusing comments from the list, and recalculate the grade.

## 6 Plagiarism checking

Plagiarism is a real problem ([12], [13], [14]) that must be faced from different perspectives: formation, prevention, and prosecution. Both, prevention and prosecution can benefit from technological development as plagiarism itself does.

VPL includes a tool to check plagiarism among source code. The main goal of this tool is to detect plagiarism among submissions for a task in a course, but it can include other sources, like submissions for the same task in previous semesters, or similar tasks from other courses, which are a probable source of plagiarism.

The process to find similarities among source files is composed of three steps: tokenization, comparison and clusterization (Fig. 12).

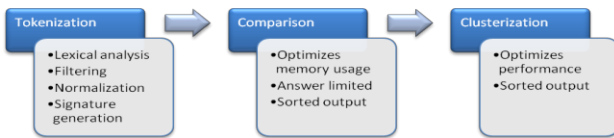


Figure 12. Steps to find similarities among source files

Tokenization is the process to get a normalized signature from every file in order to perform an efficient comparing to find similarities among them. It is composed of three phases: lexical analysis, filtering and normalization (Fig. 13). The lexical analysis extracts the tokens that represent the elements of a program (they depend on the programming language), then those tokens are filtered to delete those that are irrelevant for comparison, and finally expressions are normalized into a canonical form, producing the program signature.



Figure 13. Tokenization phases

Signatures are normalized representation for the source code files, extracted from them in order to optimize the comparison process. The form of the signature depends on the metric to be used in comparison. VPL use three different metrics [15] which, when comparing two signatures, produce a number in the range 0.0 to 1.0, where 0.0 means "totally equals" and 1.0 means "totally different". Using three metrics takes advantage of the fact that they are affected in a different way by the modifications of the code. Table I shows the effect of typical code changes on the metrics (F = Filtered, A =

Affected, NA = Not Affected, SA = Slightly affected, ASC = Affected by Size of Changes, ANC = Affected by Number of Changes).

TABLE I. METRICS AFFECTATION

Change \ Metric	Comments	Name of Identifiers	Code reorder	Systematic change	Complex change
Metric 1	F	F	NA	SA	A
Metric 2	F	F	NA	ASC	ASC
Metric 3	F	F	A	ANC	ANC

Experience using the tool has shown that some plagiarism cases present not a one-to-one relationship, but a group relationship where usually nobody has a detailed knowledge of all the participants, for example, student A may lend its work to students B and C, without mutually knowledge of B and C; student C may lend the work to student D, without knowledge of A and B, and so on [16]. To provide the reviewers with information about that kind of event, the system incorporates algorithms to identify clusters of most similar files.

The visualization system of the anti-plagiarism tool permits to visualize lists of pairs of similar files, clusters of similar files, or file-to-file similarities (Fig. 14).

```

61 @Override
62 public String toString()
63 String s="";
64 if(grado)!=0 && vector.get(0)!=0
65 s="0";
66 else
67 for(int i=grado();i>0;i--) {
68 if(vector.get(i)!=0) {
69 s=s+"2";
70 if(vector.get(i)==-1 && i>0)
71 if(vector.get(i+1)!=0)
72 s=s+"-";
73 if(Math.abs(vector.get(i))>1 ||
74 s==vector.get(i);
75 if(i>1)
76 s=s+"*";
77 else if(i==1)
78 s=s+"*";
79 }
80 }
81 }
82 return s;
83 }
84 }
85 }
86 //La función valor que produce un valor real "v", es <<<
87 public float valor(float v)
88 float sum=0;
89 for(int i=0;i<grado();i++){
90 sum=(float)vector.get(i)*(float)Math.pow
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
  
```

Figure 14. File-to-file visual comparison

## 7 Experience of use

VPL current version was released in September 2011, and is used for nine courses in the current academic year (2011/2012) with a total of 1181 students. At the moment of writing this paper (the second semester is not ended) 376 VPL activities have been configured for those courses, producing 18664 submissions (these counts only the final submissions, not the multiple tentative submissions done during the submission period).

A previous release was tested during the academic year 2009/2010 using only a course with 208 students and 42 activities, producing 1315 final submissions, and then it was used in the academic year 2010/2011 for five courses, with 661 students and 264 activities, producing 5140 final submissions.

During this period, VPL was publicly available and was developed and used for about 50 academic institutions over the world. The comments and suggestions made by these users also have contributed to the current version.

Some polls made to the students during the academic year 2011/2012 reveal that VPL activities were the best valued learning resources, with about a 80% of students declaring that the utility of VPL activities for their learning was very high and about a 16% declaring that it was high.

## 8 Conclusions and future work

This paper describes VPL, a powerful tool to manage and assess computer programming exercises freely distributable under GNU/GPL license. The main advantage of this tool is its integration in a popular learning management system, as Moodle is. This integration provides access to all the features of that kind of platforms.

A major feature of VPL is its capacity to produce complete assessment reports based on program testing. Moreover the required program tests can be configured in a very flexible way, ranging from simple input-output tests to complex combinations of unit tests, coverage tests or style tests.

Another important feature of VPL is the embedded tool to check submissions for plagiarism. It is important because plagiarism is a big problem in academia, as many studies have shown.

The current version of VPL (1.4) runs under Moodle 1.9.x. By the beginning of the 2012 summer will be released the version adapted to Moodle 2.x, including new and improved features.

The architecture of VPL will be re-engineered to include the case when the Moodle server is running in a cluster. The problem with the clusters is that, using the current design, the communication between the console and the jail for an interactive execution must be done through the Moodle server, but, if there are a cluster of servers responding the requests, it is not possible know what server has to manage the communication.

The Java applet editor will be changed by one developed using HTML5. Currently the Java applets do not work properly on systems like iPad, or tablets running Android.

## Acknowledges

We wish to acknowledge the support of the Department of Informática y Sistemas of the University of

Las Palmas de Gran Canaria to the development of VPL. We also wish to acknowledge the coordinators and teachers of the courses that have used VPL during its test phase for their useful comments.

## References

- [1] Rees, Michael J. «Automatic assessment aids for Pascal programs.» SIGPLAN Not. (ACM) 17 (1982): pp 33-42.
- [2] Schorsch, Tom. «CAP: an automated selfassessment tool to check Pascal programs for syntax, logic and style errors.» SIGCSE '95: Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education. ACM, 1995, pp 168-172.
- [3] Choy, M., S. Lam, CK Poon, FL Wang, YT Yu, and L. Yuen. «Towards Blended Learning of Computer Programming Supported by an Automated System.» Blended Learning, 2007: 9.
- [4] Palakal, Mathew J., Frederick W. Myers, and Carla L. Boyd. «An interactive learning environment for breadth-first computing science curriculum.» SIGCSE Bull. (ACM) 30 (1998), pp 1-5.
- [5] Morris, D.S. «Automatic grading of student's programming assignments: an interactive process and suite of programs.» Frontiers in Education, 2003. FIE 2003. 33rd Annual. 2003. S3F-1-6 vol.3.
- [6] Truong, Nghi, Peter Bancroft, y Paul Roe. «A web based environment for learning to program.» Australian Computer Society, Inc., 2003, pp 255-264.
- [7] Foubister, S.P., GJ Michaelson, and N. Tomes. «Automatic assessment of elementary Standard ML programs using Ceilidh.» Journal of Computer Assisted Learning 13 (1997), pp 99-108.
- [8] Higgins, Colin A., Geoffrey Gray, Pavlos Symeonidis, and Athanasios Tsintsifas. «Automated assessment and experiences of teaching programming.» J. Educ. Resour. Comput. (ACM) 5 (2005): 5.
- [9] Joy, Mike, Nathan Griffiths, y Russell Boyatt. «The boss online submission and assessment system.» J. Educ. Resour. Comput. (ACM) 5 (2005): 2.
- [10] Rodríguez-del-Pino, J.C.; Rubio-Royo, E.; Hernández-Figueroa, Z. Uses of VPL. 5th International Technology, Education and Development Conference (INTED). 2011, pp 743-748.
- [11] Beck, K. and Gamma, E. JUnit cookbook Available on-line at: <http://JUnit.sourceforge.net/doc/cookbook/cookbook.htm>. 2002
- [12] Crittenden, V. L.; Hanna, R. C. & Peterson, R. A. *The cheating culture: A global societal phenomenon*. Business Horizons, 2009, 52, 337-346
- [13] Hughes, J. & McCabe, D. *Academic misconduct within higher education in Canada*. Canadian Journal of Higher Education, 2006, 36, 1
- [14] McCabe, D. L. Cheating among college and university students: A North American perspective. International Journal for Educational Integrity, 2005, 1, 1-11.
- [15] Rodríguez-del-Pino, J.C.; Rubio-Royo, E.; Hernández-Figueroa, Z. Fighting plagiarism: metrics and methods to measure and find similarities among source code of computer programs in VPL. 3rd International Conference on Education and New Learning Technologies (EDULEARN). 2011, pp 4339-4346.
- [16] Lesner, B., Brixtel, R., Bazin, C., & Bagan, G. (2010). A novel framework to detect source code plagiarism: now, students have to work for real!. Proceedings of the 2010 ACM Symposium on Applied Computing, ACM, 2010, pp 57-58.