AUTOMATIC EVALUATION OF COMPUTER PROGRAMS USING MOODLE'S VIRTUAL PROGRAMMING LAB (VPL) PLUG-IN

Dominique Thiébaut
Department of Computer Science
Smith College
Northampton, MA 01060
dthiebaut@smith.edu

ABSTRACT

With the increased enrollment in CS courses and the growing interest in providing MOOCs, automated grading of student programs is becoming more of a need than an option. This paper describes the first experience of using Moodle's Virtual Programming Lab (VPL) for the automatic evaluation of students' program in two of our programming courses. Early experimentation in several courses shows the plug-in to be flexible and robust, allowing for sophisticated ways to test student programs. The automatic grading requires a significant shift of faculty time, putting the bulk of the effort up-front, preparing the assignment, the testing environment, and its validation. However, once this phase is over, very little effort is required once the assignment deadline is passed.

INTRODUCTION

As enrollment is steadily increasing in computer science departments nationwide [3], faculty and assisting staff find themselves having to spend an increasing amount of time grading and evaluating student programs on a regular basis. The choice is to either, 1) lower the frequency with which students return assignments, 2) evaluate only a random sample of assignments, 3) include teaching assistants in the grading process, 4) hire graders, 5) devote more personal time to the grading, or 6) use automated evaluation systems. All have drawbacks and advantages.

A few different automated grading systems are on the market [4], and this paper presents an early evaluation of the use of Virtual Programming Lab (VPL), a plug-in for the *learning management system* Moodle [1]. We are currently using it in three different courses; Introduction to Computer Science, Data Structures in Java, and Assembly Language, all taught during the Fall 2014 semester. VPL is a project spearheaded by Prof. Juan Carlos Rodríguez-del-Pino at the University of las Palmas de Gran Canaria, in Spain.

Although the plug-in documentation is succinct, the on-line support provided by Rodríguez-del-Pino is remarkable. Furthermore, the richness of options VPL provides, its overall robustness, and its ability to incorporate the instructor's code as part of the test frame, have contributed to a successful experience along with a growing expertise, and we plan on adopting the plug-in more widely in the future.

There are many caveats, though. The most noticeable one is a time shift, requiring a significant amount of work before releasing an assignment, and practically none after the deadline is passed. Before the assignment is released, the instructor must spend time preparing the assignment, writing a solution program,

generating scripts that will evaluate the student program, testing the complete setup, and incorporating all this as a new VPL activity in Moodle. After the assignment is closed, a brief review of the submissions will ensure that everybody received a grade, and a quick scan of submitted programs will detect fraudulent attempts. The VPL plug-in provides a *similarity test* to flag submitted programs that have a high level of similarity, a useful feature for spotting possible cheating. After this half-hour intervention, the instructor is basically free of the assignment.

VPL Setup and Operation

VPL is a Moodle plug-in, and requires a dedicated separate execution server, or *jail server* for short. This jail server runs the test scripts on the programs submitted by the students. Should a student program crash the jail server, the Moodle server is unaffected.

We provide a quick summary of the VPL operations now. Interested readers can find additional information on the VPL Web site [1][6].

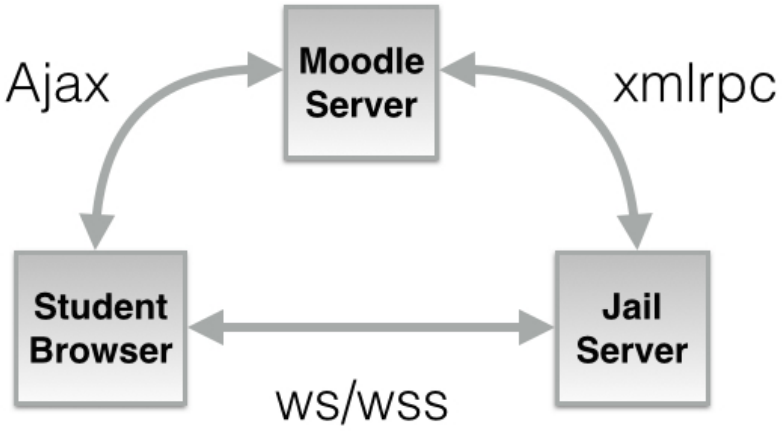

Figure 1: technologies used in student to Moodle VPL connections.

A student interacts with the system as illustrated in Figure 1. When a student submits her program, the Moodle server packages the instructor's test script along with the student program in an xmlrpc message, and ships it to the VPL jail server. There, the test scripts are executed in a *sandboxed* environment, and the captured output is sent back to the Moodle server. The output typically contains feedback comments to the students generated by the test scripts ("Output OK", "Your function is not recursive", "Your program timed out (infinite loop?)") , along with a grade, also computed by the test scripts. These comments and grades are formatted following a simple syntax supported by the plug-in. A window in the student browser reports the feedback and the grade. Figure 2 shows the instructor's view, after having changed the role to a student. The grade is automatically incorporated in the student's internal record, under the rubric selected by the instructor (lab, homework, quiz, etc.).

VPL Features and Options

VPL supports many options and features. We list here those we believe computer science instructors will find most interesting.

- The number of languages supported is quite large. The VPL Web site [3] lists Ada, C, C++, C#, Fortran, Haskell, Java, Matlab, Octave, Pascal, Perl, Php, Prolog, Python, Ruby, Scheme, SQL, and VHD languages as supported. We have been successfully implementing VPL assignments for Python, Java, and Assembly language. Any language with a compiler or interpreter supported by Linux with executable that output text can be evaluated. Testing programs outputting graphics requires additional tools.
- The instructor defines how the student program is evaluated and graded. This allows for testing properties of a program other than its. For example, in assembly language, it may be important for an assignment to generate a program with as small a memory footprint as possible. The instructor can create a script that will measure the static footprint of the student program and assign a grade inversely proportional to the program's byte size.
- The instructor can define the rubric under which a VPL grade is assigned. This is controlled per VPL-assignment.
- The instructor can make the grade visible to the student, or not. In the latter case, the grade is revealed after the due date.
- At the time of this writing, the instructor cannot limit the number of submissions for a given program, or set of programs. A survey of some 30 VPL assignments given this semester indicates that approximately 85% of the students submit a given program less than 10 times, while the remaining 15% fall in a long tail of recorded clicks. The largest recorded number of submission for a given assignment is 51 times.
- The instructor can controls the resources needed by the jail server.
- For a given VPL activity, the programs submitted can be those of an individual student, or from a group of students.
- Access to the plug-in, including submission, can be restricted by IP address.
- The instructor can enforce for programs to be typed by hand in the submit window, and disable copy/paste of program code.

EXAMPLE VPL ACTIVITY

We now present a simple VPL module for evaluating a Python assignment. The requirement for the student is to create a Python program containing a function called *randomInt()* that returns a random integer. The given grade depends on several criteria: 1) the Python file exists and bears the right name, 2) the program contains a function named *randomInt()*, and 3) the functions returns an integer. Two scripts form the evaluation system provided by the instructor: a python program that attempts to import the student's function, and a bash script that launches the python program. VPL requires that the main script, *vpl_evaluate.sh*, must generate a second script, *vpl_execution*, and it is this second script that is executed on the jail server. The *vpl_evaluate.sh* script, below, is remarkably short:

```
#! /bin/bash
# vpl_evaluate.sh

echo "#! /bin/bash" > vpl_execution
echo "python3.4 customEval.py">> vpl_execution

chmod +x vpl_execution
```

The program *customEval.py*, below, tests the student program by importing and running its function. It also outputs the final grade that will be picked-up by the VPL module, and added to the student's record. Special prefixes must be added to each output line for VPL to parse it correctly.

```
# customEval.py
def comment(s):
    '''formats strings to create VPL comments'''
    print('Comment :=>> ' + s)

def grade(num):
    '''formats a number to create a VPL grade'''
    print('Grade :=>> ' + str(num))

try:
    import randomInts
except:
    comment("unable to import randomInts")
    grade(0)
    exit()

try:
    randomInts.randomInt
except:
    comment("randomInts.randomInt isn't defined")
    grade(25)
    exit()

try:
    if(type(randomInts.randomInt()) == int):
        comment("great job!")
        grade(100)
    else:
        comment("randomInt doesn't return an int as required.")
        grade(90)
except:
    comment("randomInts.randomInt crashes")
    grade(75)
```

Note that the main bash script is extremely short and relies on the python test program for doing the heavy lifting. It is possible to generate a testing system where the bash script does most of the evaluation work, and the attached programs just provide an infrastructure for the testing. We provide several longer and more sophisticated VPL examples on our own Web site [5].

OBSERVATIONS

A couple months of practice with Moodle and its VPL plug-in have been satisfying enough to warrant continuing this experiment. We see VPL as a good solution to cope with the growing enrollment and the need to test regularly growing numbers of students anxious to learn programming.

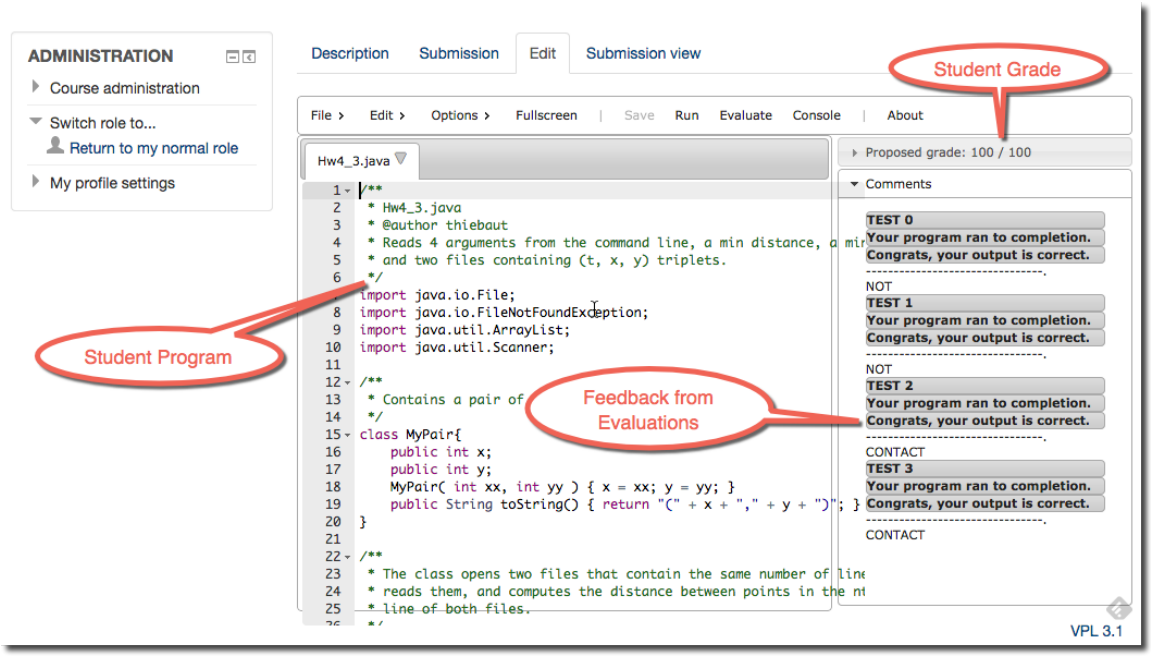

Figure 2: View of VPL program evaluation in the browser.

We share here some of our early observations, and provide additional comments.

- Adopting VPL requires a major *pedagogical shift* for the faculty used to grade by hand weekly assignments. One loses an important connection with one's students, and a sense of their understanding of the class material. Steps must be taken to maintain some level of connection. This can be accomplished through quick reviews of the submitted work after the deadline
- The bulk of the time spent on an assignment is concentrated *prior to the release of the assignment*. This time is considerable the first time one sets up new VPL activities, as one has to anticipate many different ways for student programs to fail, or to miss the idea behind the assignment. Specialized tests must be written to capture all possible shortcomings in student submissions.
- *Randomization of inputs* is necessary, as well as a semi-opaqueness of the way the programs are tested. While the stronger students will pride themselves in submitting programs that answer the problem at hand using the correct approach, some will be tempted to bypass the assignment and create programs that spit out the expected answer. This can be circumvented by randomizing the input data used to test the programs, and by not fully disclosing the manner in which programs are tested. Some explanatory feedback is necessary, but should not be blueprint for creating programs that bypass the mission of the assignment.
- While the number of submissions performed by a student is currently not limited, we believe such a feature would be beneficial. We have observed students using Eclipse to generate their Java program, submit their first draft

to VPL, and then stay in the VPL edit window to finish up getting rid of the final bugs.

- The due date and time of an assignment can be recorded in the test script, allowing the script to lower the grade by some selected amount for every day past the deadline.

CONCLUSIONS

Both the financial appeal of MOOCs and the fast increase we see in CS departments nationwide require changing the way we assess programming skills. We have chosen to implement the automated evaluation of student programs in three of our heavily enrolled classes. Our early experience with VPL is positive. The wide array of programming languages VPL supports, its robustness of implementation, and the flexibility it offers compensate for its complexity of use, and its currently sparse documentation. We have started releasing scripts we have generated for various assignment in an effort to share our experience, ideas, and solutions, hoping others can benefit from our experiment.

ACKNOWLEDGEMENTS

We wish to thank Juan Carlos Rodríguez-del-Pino for his quick and friendly help with our many questions about VPL, and for his insightful comments on an early draft of this paper.

REFERENCES

[1] Juan Carlos Rodríguez-del-Pino, Enrique Rubio-Royo, Zenón, and J. Hernández-Figueroa, A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features, *Proc. WorldComp12,* July 2012, Las Vegas, USA.
[2] Gray, I. M., Hyde, D. R., Jekyll, M. R., NP-complete problems with no known optimal solutions, *Proc. 1st Conference on Hard, Hard Problems*, 1 (1), 100-799, 1999.
[3] Graduate Education, Enrollment, and Degrees in the United States, National Science Foundation, Feb. 2014, on-line document, http://www.nsf.gov/statistics/seind14/index.cfm/chapter-2/c2s3.htm, captured Nov 2014.
[4] J. M. del Alamo, A. Alonso, M. Cortés, Automated Grading and Feedback Providing Assignments Management Module, *Proc. Int'l Conf. Educ., Research, & Innov*., Nov. 2012, Madrid, Spain.
[5] D. Thiebaut, Moodle VPL Tutorials, on-line document, http://cs.smith.edu/dftwiki/index.php/Moodle_VPL_Tutorials, captured Nov. 2014.
[6] Activities: Virtual Programming Lab, on-line document, https://moodle.org/plugins/view.php?plugin=mod_vpl, captured Jan. 2015.