AUTOMATIC EVALUATION OF COMPUTER PROGRAMS USING MOODLE'S VIRTUAL PROGRAMMING LAB (VPL) MODULE

Dominique Thiebaut
Dept. Computer Science
Smith College, Northampton, MA 01064, USA

ABSTRACT

With the increasing enrollments in CS courses and the growing interest in providing MOOCs, automated grading of student programs is becoming a necessity. This paper describes our 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 show the module 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 not only the assignment, but also the solution program, the testing environment, and its validation. However, once this phase is over, very little attention is needed.

INTRODUCTION

As enrollments are steadily increasing in computer science courses 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. To cope, the choice is to, either, lower the frequency with which students return assignments, evaluate only a random sample of assignments, include teaching assistants in the grading process, hire graders, devote more personal time to the grading, or use automated evaluation systems. All have drawbacks and advantages.

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

Although the module 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 at our institution, and growing expertise, and we plan on adopting the module more widely in the future.

There are many caveats, though. The most noticeable is a time shift, requiring a significant amount of work before releasing an assignment, and practically none after the assignment is released. The instructor's time must now include the preparation of the assignment, the preparation of the solution program, the creation of scripts that will evaluate the student program, the testing of the

Accepted for presentation at CCSCNE 2015.

setup, and the creation of a new VPL activity on Moodle. Once these actions are performed, the instructor's attention to the assignment is minimal. 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 help detect fraudulent attempts (such a writing a program that prints the solution instead of computing it). The VPL module provides a *similarity test* to flag submitted programs that have a high level of similitude, a useful feature for spotting possible fraud. After this half-hour intervention, the instructor is basically free of the assignment.

VPL Setup and Operation

VPL is one of many Moodle modules. It requires a typical two-step installation process to integrate with this *Learning Management System*. Although it doesn't require it, a dedicated separate execution server, or jail server for short, is highly recommended. This jail server runs the test scripts along with 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].

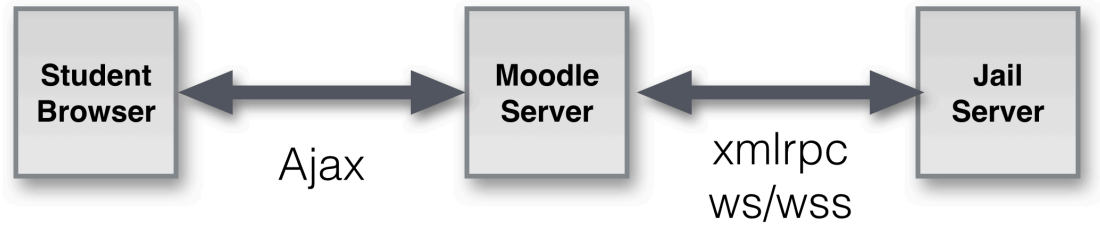

Figure 1: technologies used in student to Moodle VPL connections.

A student interacts with the system as illustrated in Figure 1. The students interact with VPL through a browser. When a student submits her program (copy/paste, edit, upload), the Moodle server packages the instructor's test script along with the student program in an archive, and ships it to the VPL jail server. There, the test scripts are executed in a *sandbox* environment, and the captured output is sent back to the Moodle server. The output typically contains feedback comments 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 module. A window in the student browser reports the feedback and the grade, as illustrated in Figure 2. The grade is automatically incorporated in the student's internal record, under the rubric selected by the instructor (lab, homework, quiz, etc.). Note that VPL has an automated feature that allows the instructor to define the input that must be fed to the student program, and the expected output for each input. VPL automatically assigns a grade between

0 and 100% proportional to the number of outputs that correctly match the expected outputs. This feature can be used for very simple assignments.

VPL Features and Options

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

- The number of languages supported is quite large. The VPL Web site [1] 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 successfully implemented 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 and expertise.
- The instructor defines how the student program is evaluated and graded. This allows for testing properties of a program other than its output to a given input. For example, in assembly language, it may be important for an assignment to generate a program with as small a footprint in memory 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.
- VPL uses HTML5, Ajax, and WebSocket, in an effort to free students from having to use Java-enabled browsers.
- 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, and reveal it only after the due date.
- The instructor can 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 submission clicks. The largest recorded number of submission for a given assignment is 51 times.
- The instructor controls the resources needed by the jail server, including stack or dynamic space.
- For a given VPL activity, the programs submitted can be those of an individual student, or from a group of students.
- Access to the submission page can be restricted by IP address, a feature probably more useful in a MOOC environment, or protected by a password.

EXAMPLE VPL MODULE

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 file exists and has the right name, and 2) the file contains a function named *randomInt()*, and 3) the function returns an integer. Two programs

form the evaluation system provided by the instructor: a python program that attempts to use 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
```

CustomEval.py, below, tests the student program and 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].

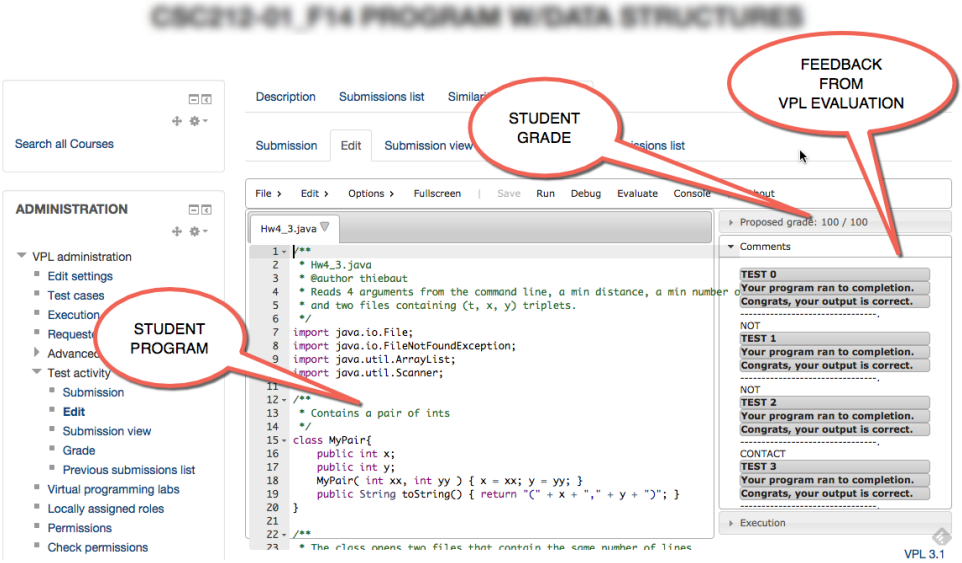

Figure 2: Student view of VPL module in the browser.

OBSERVATIONS

A couple of months of practice with VPL have been satisfying enough to warrant continuing this experiment. We see VPL as a good solution to cope with our growing enrollments. We share here some of our early observations, and provide additional comments.

- Adopting VPL requires a major *pedagogical shift* for the faculty used to grading by hand the weekly or bimonthly assignments. One loses an important connection to the students, and to a sense of their acquisition of the class material. Students become more anonymous and disconnected from the teaching experience. Colleagues teaching large classes are familiar with this situation, but those of us teaching at smaller institutions must adapt to this shift. Steps must be taken to maintain some level of connection. This can be accomplished through quick reviews of the submitted work after each deadline. Because the work is already graded, a quick scan is sufficient to get a sense of the students performance.
- 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 maintaining a semi-opaqueness of the way the programs are tested. Invariably, some students 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. However, a completely opaque test that does not show the input used for testing, the output generated in response to the test,

nor the expected output, has proven to be a solid recipe for creating mob scenes outside one's office. Some explanatory feedback is necessary, but should not be a blueprint followed by students to create hacks that bypass the mission of the assignment. We found that provided a simplified version of the test program with the assignment greatly helps students. Giving fully documented test programs at the beginning of the semester and slowly reducing them to simple explanations of what the test programs will do will hopefully satisfy the students while helping them acquire the skills required for testing their own programs well.

- Limiting the number of submissions allowed by students for an assignment is tempting, but should be done with care. Allowing for an unlimited number of submissions encourages students to develop their programs directly in the edit window of the VPL module, and bypass the IDE used in class. Recognizing that some students need the flexibility of submitting many times, and value the feedback provided by VPL, the limit should be greater than 1, but probably less than 10. Slowly decreasing the number of submissions allowed as the semester progresses is probably wise.

CONCLUSIONS

The financial appeal MOOCs have on officials of educational institution, and the fast increase we see in CS courses nationwide are two pressures that require solutions. One option is to adapt the way we teach and evaluate students in programming intensive 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 assignments in an effort to share ideas, and solutions, hoping others can benefit from our experience.

REFERENCES

[1] Juan Carlos Rodríguez-del-Pino , VPL General Documentation, on-line document, http://vpl.dis.ulpgc.es/index.php/en/support, captured Nov 2014.

[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 repository, http://cs.smith.edu/dftwiki/index.php/Moodle_VPL_Tutorials