An Online System for Monitoring and Assessing the Programming Process

Philip E. Robinson, *Member, IEEE* and Johnson Carroll, *Member, IEEE*

***Abstract*—Traditional assessment of programming ability, as well as more recent automated assessment systems, consider only the completed program submitted by the student. We present a system which automatically monitors and assesses the code-production process as well as the final product, allowing adaptive feedback and assessment of programming competency. Our assessment system is based on open-source components which provide a full online programming environment and allows for a flexible scripting interface to the assessment process, which can monitor student actions during the programming task. The system was implemented for an introductory programming class of nearly 400 students, and an example of the automated assessment is presented.**

***Index Terms*—Computer Science Education, Learning Management System (LMS), Online Assessment, Open-Source**

I. Introduction

COMPUTER programming is frequently a hurdle to engineering and computer science students, particularly those who do not have substantial personal experience with computers. Programming is one of the more practically-focused subjects in an engineering curriculum. Students usually learn *how to program* rather than *about programming*, and learning is achieved through practice almost from the first day (e.g., the "Hello, World!" program). Many introductory programming courses and books present the material in a tutorial format, encouraging students to work through examples and experiment with structures. This format familiarizes the student with the development environment, and (at least initially) encourages a trial-and-error approach to exploring the capabilities of the language and structures with feedback from the programming environment. The same model of practice persists into a professional programming setting: experienced programmers usually create and interact with code in an interactive manner (though the experienced programmer should not need or use the same amount of trial-and-error as the new learner).

Manuscript received July 22, 2016.

P. Robinson is with the Department of Electrical and Electronic Engineering Science, University of Johannesburg, Auckland Park, South Africa, 2006 (email: philipr@uj.ac.za).

J. Carroll is with the Faculty of Engineering and the Built Environment, University of Johannesburg, Auckland Park, South Africa, 2006 (email: jcarroll@uj.ac.za).

In this paper, we demonstrate a system which can automatically monitor and assess both the student-generated code as well as the students' code-production process. The assessment and evaluation occur in real-time as the students program, mirroring the interactive programming activity, and allow for dynamic and adaptive assessment of programming ability.

A. Assessment of Programming Ability

Traditional assessment (via written tests and exams) in higher education encourage study methods that are not practice-centered [1]. Given that programming is a practical activity introduced through practice, it follows that authentic and valid assessment of programming ability would take the form of programming exercises; such a conclusion is firmly supported by the theory of constructive alignment [2]. Further, assessment of programming should incorporate instant feedback, iterative submissions, and an inherently trial-and-error approach (within reason) that reflects both training and practice. Yet many programming classes are assessed via written exams despite general discomfiture with that format [3-4]. We note the distinction raised by some programming instructors between the ability to program and the understanding of programming concepts [4]: other assessment formats may certainly be more accurate and valid measures of student understanding, but ability is best assessed through demonstration of that ability.

Demonstration-based assessment of programming is frequently avoided because of the significant logistical effort required or limited to assignments for which authorship cannot be guaranteed [5]. When one attempts to assess not only the product (submitted code) but the process that students follow when programming, the logistical effort is even more immense. One documented implementation by Bennedsen and Caspersen [6] utilized five teaching assistants, a lecturer, and an examiner to test 20 students at a time.

B. Automated Programming Assessment and Monitoring

When student numbers are large, programming instructors (who are often quite adept at programming) frequently devise methods of automated programming assessment (APA). There have been many efforts over the years to assess programming assignments automatically, and numerous studies have documented the benefits and drawbacks of automated assessment systems (surveys are presented in [7] and [8]). A

significant and frequently highlighted benefit is the scalability and consistency of assessment, which is increasingly important in the modern online education context [9].

Various sources, including [7] and [8] classify approaches to APA according to whether they require execution of the submitted code (dynamic) or not (static). Static APA is concerned with code length and structure, while dynamic APA is generally based on whether a submitted program correctly handles a number of carefully chosen test cases [10]. Others have sought to further classify student programs according to efficiency, complexity, and other measures [11]. Much of the literature on APA systems concerns the selection of appropriate test cases for increasingly complex programming assignments [10], or for testing structure and sub-functionality of the submitted program [12].

All of the APA systems and implementations we could find are not live, in the sense that the student works on the problem, submits her code, and then the code is assessed. Even those systems which incorporate automated feedback provide that feedback only post-submission, though may allow repeated submissions [13, 14]. By contrast, our system *monitors* and can adapt or assess based on the student actions throughout the testing and debugging phase of the students' code production process. This provides insight into the students' approach to a programming problem, and allows the system and/or instructor to adapt the presentation based on process students follow.

In the rest of the paper, we identify aspects of the programming process that can be monitored, describe structure and implementation of our online monitoring and assessment system, give an example of a practical implementation in a class of nearly 400 students, and indicate some of the unique ways in which it can be used to assess student programming.

II. Monitoring & Assessing the Programming Process

As one is introduced to programming and develops into a more capable and experienced programmer, the approach to solving a programming problem becomes increasingly important. One's proficiency in a programming language and problem solving skills are revealed not simply through the code that one produces, but through efficiency and agility demonstrated while producing that code. Traditional assessment of programming, even when automated, can evaluate efficiency through time limits, but provides little insight into the process followed by a student. Such insight is doubly important during introductory classes and formative assessment, where a poor approach to problems may encourage habits that significantly hamper performance later in the curriculum.

TABLE I
Significant Aspects of the Programming Process

Competency area	Process indicator
Language proficiency, debugging method (trial-and-error vs reflection)	Compilation attempt pattern
Language structures	Types of compiler errors (syntax, data type)
Algorithmic methods and structures	Types of logical errors
Understanding the problem, mapping to algorithmic structures	Types of functional errors

In Table I, we identify several competency areas that can be difficult to isolate from student-produced programs but are immediately evident from observing the programming process. For example, a student may submit a working program, but the pattern and frequency of successful and unsuccessful compilation attempts will indicate whether a student is resorting to exhaustive trial-and-error searches for a correct solution. The system we present here allows the instructor to choose which indicators to monitor and how to respond to student behavior. When a competency area is deemed essential, the appropriate process indicator can be incorporated into the assessment rubric (e.g., limiting the number of attempts to compile). Otherwise, the indicator can be monitored to provide tailored advice or feedback to a student who may be pursuing inefficient or misguided solutions.

III. The Online Monitoring and Assessment Tool

The online learning management system (LMS) employed in the teaching and assessment of the course described in this work is built on the open-source Moodle learning platform, which has a thriving development community actively producing and maintaining of a large library of plug-ins. Moodle is written in PHP, so can be deployed on a wide variety of operating systems, web servers and database systems. Rodríguez-del-Pino, et al., created the Virtual Programming Lab (VPL) plug-in for the purpose of enabling automated and guided assessment of programming assignments [15]. The VPL system consists of a plug-in module that runs on the Moodle platform and a jail server which is used to execute student code submissions. The Moodle module provides the facilities for instructors to build and present programming assignments to the students.

For the students, the system provides a capable in-browser source code editor and console which facilitates the running of 27 different programming languages. When a student wants to

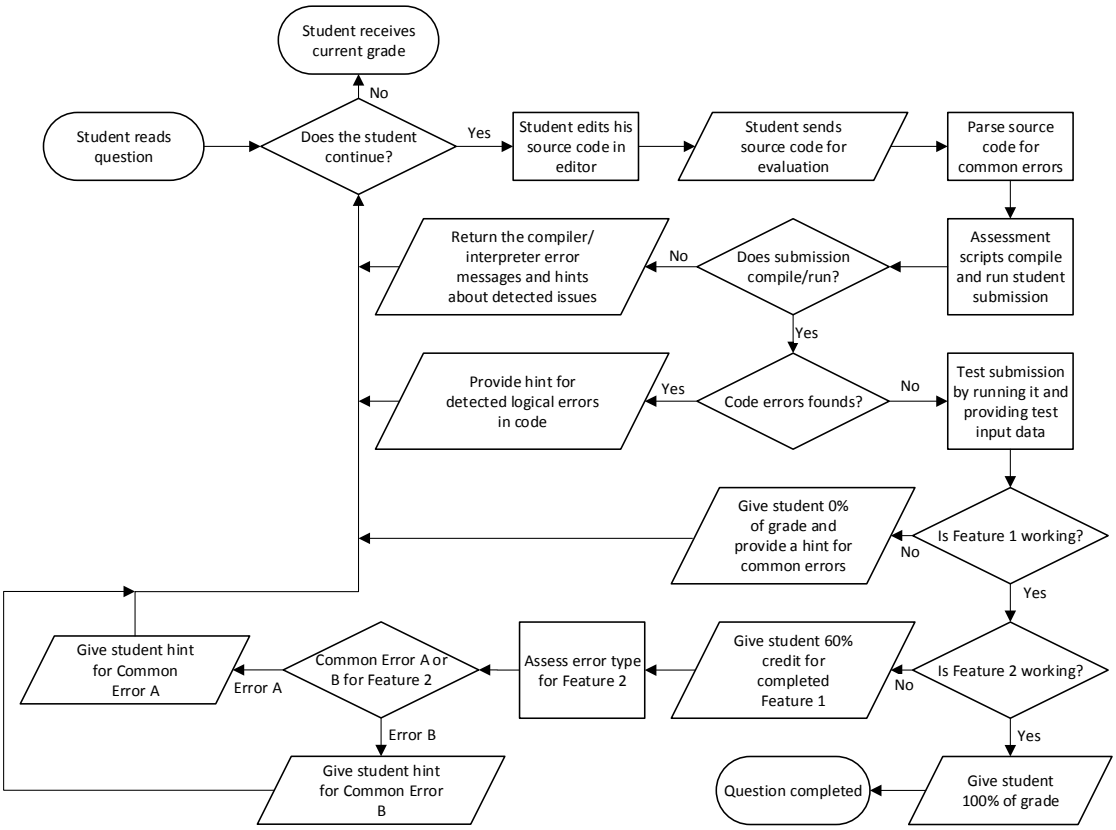

Fig. 1. Illustration of live monitoring and assessment procedure. This figure is based on the case study of a real question administered to the class in 2016.

execute their code the system sends the submission to a separate jail server which executes the code in the appropriate execution environment, using either compilers or interpreters, in a *chroot* jail which protects the jail server from badly written and malicious code.

A. VPL for Assessment

The VPL system allows instructors to build programming assignments that are assessed using test cases by specifying the data that will be input into the program and what the expected output is. In addition to the rigid test case system VPL provides the ability to use custom scripts to automatic more complex assessment tasks. The initial scripts VPL employs to assess a submission are BASH scripts and no examples or documentation is provided on how to use this interface for novel assessment schemes. However, it is possible to call a variety of scripting environments from within these scripts. Thiébaut provides a number of examples of scripted grading using Python in VPL [16]. In this work, we take this idea a step further and use the scripting interface to implement both static and dynamic assessment with live contextual guidance.

Assessment scripts examine the students code and the behavior of their programs for grading purposes and provide contextual feedback to the students, to help guide them to the correct solution. This is achieved by writing scripts that prepare the student submission for execution in the relevant environment, executes the submitted code, provides it with inputs and assesses the output of the code. It is also possible to use scripts to examine the submitted source code directly to enforce certain requirements or detect common mistakes. These scripts can then automatically provide a grade to the student along with comments about common mistakes detected in the submission. The scripted feedback is provided along with the feedback from the relevant execution system, allowing students to iteratively improve their code to find and fix bugs until they meet the requirements for the assignment.

While setting a scripted activity in VPL requires more time from the instructor than setting a question for a written assessment, the required effort does not scale with class size as with manual grading. Once the assessment and the scripts are in place they can be administered to a class of any size. Problems are also very easy to reuse, and problems set for tests can become practice problems after the test.

IV. Case Study

The system was used to administer an introductory programming class consisting of 400 students in 2016. The system was used for homework assignments, tutorials and exams. Examinations, conducted in a computer lab, required the same amount of human resources to administer as the traditional written assessments and it required far less time to set up the scripted questions as to manually mark exam papers.

Figure 1 shows the logical flow of the assessment and feedback process for a single problem administered by the system. The problem required students to write a function that accepts an array of values approximating a curve and, by approximating the derivative and finding roots, return the points nearest the extrema of the curve. Finding the approximate derivative via the specified method is referred to as Feature 1 in Figure 1 and was worth 60% of the grade, while finding and returning the extrema is referred to as Feature 2, worth the remaining 40% of the grade. (In this case the programming language, Octave, made examining the state of particular variables during execution trivial; for a compiled language, diagnostic code can be automatically inserted into the submitted source code to export the state of important variables during execution.) For this problem, Feature 2 cannot be completed without first completing Feature 1, though in general it is possible to assess multiple independent features.

After reading the problem, the student composes source code in the editor inside the browser. When ready, the student clicks "Evaluate" and the code is transmitted to the execution servers. The assessment scripts will firstly parse the code itself to look for common logical errors in the source code, static assessment. Next, the VPL module compiles or interprets and attempts to run the submitted code. If there is a problem with the code that causes it not to run, the error output from the compiler/interpreter is presented to the student along with any hints about detected common issues. In this example, a common problem was a buffer overflow when indexing the array. The student now has the opportunity to modify his code using this feedback and resubmitting.

When the code compiles and runs, the next step is to check for any further common structural errors in the code itself. In this case, the code is checked to confirm that the function is present and utilizes the correct input and output parameters. If a problem exists, a hint is provided as feedback and the student can adapt the code. Once the code is running and complies with the structural specification, the system runs number of test cases ala dynamic automated assessment. First, the array that contains the calculated samples of the derivative estimate is examined for correctness (Feature 1). If the values contained in the array are incorrect then the student is informed that their mark is currently 0% and, if relevant, a hint is provided. A common problem, in this case, was that the array should contain one sample less than the input data. If Feature 1 works correctly, then various test cases are run for the whole program. If the program fails Feature 2 tests, then the student is informed of the 60% grade. For this problem, there are two possible common errors that could be reported: incorrect number of extrema or incorrect extrema positions. Depending on which error is present, the student receives a relevant hint. The student can iterate through this process until he gets both features working and receive a grade of 100% or he can decide that Feature 2 is taking up too much time and give up. He will then receive the 60% grade for only completing Feature 1.

This example demonstrates how the system is used to monitor the various aspects of the programming process listed in Table I while the student engages with completing the problem. Though this example only *assesses* the two key functions of the program, compilation/interpretation attempts and program structure are monitored throughout program development. As the student works iteratively toward a complete solution, the system is able to provide contextual feedback.

V. Conclusion

The automated programming monitoring and assessment system presented here is novel in that it is able to monitor both the student programming process and final functionality of student submissions. The information gleaned from this process can be used to automatically provide guidance to the student, evaluate the process being followed, and assess the submission for grading purposes. The system is built to allow for live and iterative interactions with the student which is a far more authentic context for assessing programming skills than a traditional exam. A case study is presented of a real question as administered to a 400-student class, demonstrating one instance of the logical flow of the assessment and feedback process that the system is capable of.

The system we have demonstrated produces an incredibly rich trove of data about student activity while studying and during the assessments. This data can be mined to answer numerous questions about the student's behavior and the administration of the course, and to quantifiably examine the detailed programming process followed by both successful and unsuccessful students. Further analysis of this type of data is a rich field for future work.

References

[1] L.B. Nilson, Specifications Grading: Restoring Rigor, Motivating Students and Saving Faculty Time, Virginia: Stylus Publishing, 2014.

[2] J.B. Biggs, C. Tang, Teaching for quality learning at university: What the student does. McGraw-Hill Education (UK), 2011.

[3] S. Shuhidan, M. Hamilton, and D. D'Souza, "Instructor perspectives of multiple-choice questions in summative assessment for novice programmers," Computer Science Education, 20:3, 2010, pp. 229-259.

[4] J. Sheard, Simon, A. Carbone, D. D'Souza, and M. Hamilton. "Assessment of programming: pedagogical foundations of exams." Proc. of the 18th ACM conference on Innovation and Technology in Computer Science Education (ITiCSE '13), 2013, pp. 141-146.

[5] M. Kaya and S. A. Özel, "Integrating an online compiler and a plagiarism detection tool into the Moodle distance education system for easy assessment of programming assignments" Computer Applications in Engineering Education, 23, 2015, pp. 363-373.

[6] J. Bennedsen and M. E. Caspersen, "Assessing Process and Product - A Practical Lab Exam for an Introductory Programming Course," Proc. Frontiers in Education. 36th Annual Conference, San Diego, CA, 2006, pp. 16-21.

[7] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä. "Review of recent systems for automatic assessment of programming assignments." Proc. of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10), 2010, pp. 86-93.

[8] T. Staubitz, H. Klement, J. Renz, R. Teusner and C. Meinel, "Towards practical programming exercises and automated assessment in Massive Open Online Courses," Teaching, Assessment, and Learning for Engineering (TALE), 2015 IEEE International Conference on, Zhuhai, 2015, pp. 23-30.

[9] V. Pieterse. “Automated Assessment of Programming Assignments.” Proc. of the 3rd Computer Science Education Research Conference (CSERC '13), Open Univ., Heerlen, The Netherlands, 2013, pp. 45-56.
[10] T. Tang, R. Smith, S. Rixner, and J. Warren. “Data-Driven Test Case Generation for Automated Programming Assessment.” Proc. of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE ’16), 2016, pp. 260-265.
[11] X. Liu, “A new automated grading approach for computer programming.” Computer Applications in Engineering Education, 21, 2013, pp. 484-490.
[12] M. Vujošević-Janičić, M. Nikolić, D. Tošić, and V. Kuncak. “Software verification and graph similarity for automated evaluation of students' assignments.” Inf. Softw. Technol. 55:6, 2013, pp. 1004-1016.
[13] K. Buffardi and S. H. Edwards. “A formative study of influences on student testing behaviors.” Proc. of the 45th ACM technical symposium on Computer science education (SIGCSE '14), 2014, pp. 597-602.
[14] T. Wang, X. Su, P. Ma, Y. Wang, K. Wang, “Ability-training-oriented automated assessment in introductory programming course,” Computers & Education, 56:1, 2011, pp. 220-226.
[15] J.C. Rodríguez-del-Pino, R-R. Enrique, and H-F. Zenón, “A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features,” Proc. of the Intl. Conference on e-Learning, e-Business, Enterprise Information Systems, & e-Government, 2012.
[16] D. Thiébaut, "Automatic evaluation of computer programs using Moodle's virtual programming lab (VPL) plug-in." Journal of Computing Sciences in Colleges, Vol. 20, Issue 6, 2015.