New Orleans JAZZED about Engineering Education

Paper ID #16290

Using Automatic MATLAB Program Testing for a First-Year Engineering Computation Course

Prof. Bruce W. Char, Drexel University

Bruce Char is Professor of Computer Science in the Department of Computing of the College of Computing and Informatics at Drexel University. He is interested in the use of automatic feedback systems in engineering and computer science education.

Isuru Daulagala, Drexel University

Isuru Daulagala received his BS in Electrical and Computer Engineering from Drexel University, Philadelphia PA in 2014. He is currently a graduate student in the Electrical and Computer Engineering Department at Drexel University. His research interests include Physical Design, High Performance Computing and developing and improving tools in Engineering Education. From September 2014 to March 2015, he was an intern at NVIDIA Corporation at Santa Clara, CA.

Prof. Nagarajan Kandasamy, Drexel University

Naga Kandasamy is an Associate Professor in the Electrical and Computer Engineering Department at Drexel University where he teaches and conducts research in the area of computer engineering, with specific interests in embedded systems, self-managing systems, reliable and fault-tolerant computing, distributed systems, computer architecture, and testing and verification of digital systems. He received his Ph.D in 2003 from the University of Michigan. Prior to joining Drexel, he was a research scientist at the Institute for Software Integrated Systems, Vanderbilt University, from 2003-2004.

Prof. Kandasamy is a recipient of the 2007 National Science Foundation Early Faculty (CAREER) Award and best student paper awards at the IEEE International Conference on Autonomic Computing in 2006 and 2008, and the IEEE Pacific Rim Dependability Conference in 2012. He is a senior member of the IEEE.

Dr. Thomas T. Hewett, Drexel University

Tom Hewett is Professor Emeritus of Psychology and of Computer Science at Drexel University. His teaching included courses on Cognitive Psychology, Problem Solving and Creativity, the Psychology of Human-Computer Interaction, and the Psychology of Interaction Design. In addition, he has taught one-day professional development courses at both national and international conferences, and has participated in post-academic training for software engineers. Tom has worked on the design and development of several software projects and several pieces of commercial courseware. Some research papers have focused on the evaluation of interactive computing systems and the impact of evaluation on design. Other research papers have explored some of the pedagogical and institutional implications of universal student access to personal computers. In addition, he has given invited plenary addresses at international conferences. Tom chaired the ACM SIGCHI Curriculum Development Group which proposed the first nationally recognized curriculum for the study of Human-Computer Interaction. Tom's conference organizing work includes being Co-Chair of the CHI '94 Conference on Human Factors in Computing Systems and Program Chair for the 2013 Creativiey and Cognition Conference. In 2014 he was the recipient of the ACM SIGCHI Lifetime Service Award.

Work in Progress: Using automatic MATLAB program testing for a Large First Year Engineering Computation Course

Abstract

This work in progress describes our inaugural use of Cody Coursework to provide 24/7 automatic feedback for MATLAB programming practice in homework and in labs, as well as assessment in proctored quizzes given in lab periods, in a class of approximately 1000 first year engineering students currently run in approximately 35 lab sections. One of the most basic principles of instruction is that students get better with practice; the beneficial effects of practice are enhanced by timely feedback. For computer programming, scarce grading and feedback resources can be augmented by automatic testing tools that provide basic feedback on whether student programming is meeting expectations for correct output behavior. Cody Coursework provides a self-service web interface to a cloud-based service that informs students of the results of instructor-provided tests for programs they have submitted. Instructors can obtain summary and detailed reports of class results over the web and as CSV format downloads. The instructor interface allows simple means of providing problems and ways to check correctness and performance for labs, assignments and quizzes. We have developed dozens of Cody exercises as part of normal course development activities in the past year deployed in a conventional course with face-to-face lecture and lab time, a textbook, homework, and exams.

Our evaluation of Cody Coursework as a teaching-learning tool is an ongoing activity that involves two phases, two years, and two types of evaluation, with the first type of evaluation being applied iteratively as needed. The first phase and type of evaluation is formative. As we continue to implement tools, materials, exercises, etc., during this phase we regard the students as being both consultants and co-evaluators of the tools. Gathering student feedback involves using two surveys and the usage analytics provided by Cody Coursework itself. We have constructed two surveys, one given initially and one after the course as the course is concluding. The first survey will help us clarify how many students bring which types of experience to doing their evaluations and what their expectations are for using computers in the future, both immediate and long range. These evaluations are also expected to assist faculty in calibrating their ongoing choices of topics and exercise difficulty level to better fit the course goals and audience. A second formative questionnaire is focused on students providing feedback on effectiveness and usability of Cody Coursework. In the subsequent second phase, summative evaluation will be conducted through an IRB-reviewed investigation of the effects of Cody Coursework and autograded exercises upon student learning.

Introduction

Cody Coursework provides a web interface for automatically recording the results of testing student program solutions written in MATLAB against instructor provided tests. It is a relatively new service, first released in 2013. While there has been several decades' worth of work and experience with automatic grading of programs, there have few reports on Cody Classroom to

date. Our inaugural use of it provided feedback for programming practice in homework and in labs, as well as assessment in proctored quizzes given in lab periods, in a class of approximately 1000 first year engineering students currently run in approximately 35 lab sections. In this paper, we explain our goals and methodology for problem development as well as course administration to use Cody effectively. We also explain how to use the information provided through the Cody administrative interface for assessment and to analyze student learning patterns. We discuss the benefits and limitations of using a tool such as Cody for an introductory programming class.

The rationale for autograding in engineering computation

One of the most basic principles of instruction is that students get better with practice enhanced by timely feedback[1]. Computer programming courses have recently spiked in popularity[2], leading to large classes. Automatic testing tools such as Cody Coursework can provide immediate basic feedback on whether student programming is meeting instructional expectations for correct input-output behavior. Automatic assessment in programming has been in use for over a decade for those within the computer science education community who have the resources and determination to use pioneering developments[3], [4], with proven beneficial learning effects[5], [6]. Widely accessible and supported tools, in particular those that work with MATLAB, are just beginning to make their way into the educational mainstream.

A description of Cody Coursework

Cody Coursework[7]–[9] is a autograding service designed by Mathworks to autograde problems in MATLAB. It provides a self-service web interface to a cloud-based service that informs students of the results of instructor-provided tests for programs they have submitted. Instructors can obtain summary and detailed reports of class results over the web and as CSV format downloads. The instructor interface allows simple means of providing problems and ways to check correctness and performance for labs, assignments and quizzes. The service is composed of a front-end web interface in which students submit solutions to questions designed by the instructors. Then the solution is compared against reference solutions submitted by the instructors on various data sets. This computation task is run in the back end as is done through a cloud hosting service such as Amazon Web Services (AWS).

The web interface from a student's point-of-view is shown in Figure 1. In Cody Coursework all questions should be part of an "assignment". Each assignment can have any number of questions. A start and end time can only be set at the assignment level, hence all questions in a given assignment have a common administering time frame.

When a student selects a question, its description will be shown in the right panel. The student then goes on to submit the solution to a given solution in a specified space. The solution is run through multiple tests and the status of each test is provided to the student. The test suite and results after a solution is submitted is shown in Figure 2. Each test is visible and the result of the test is shown. In this case, the solution passed through two tests but failed one.

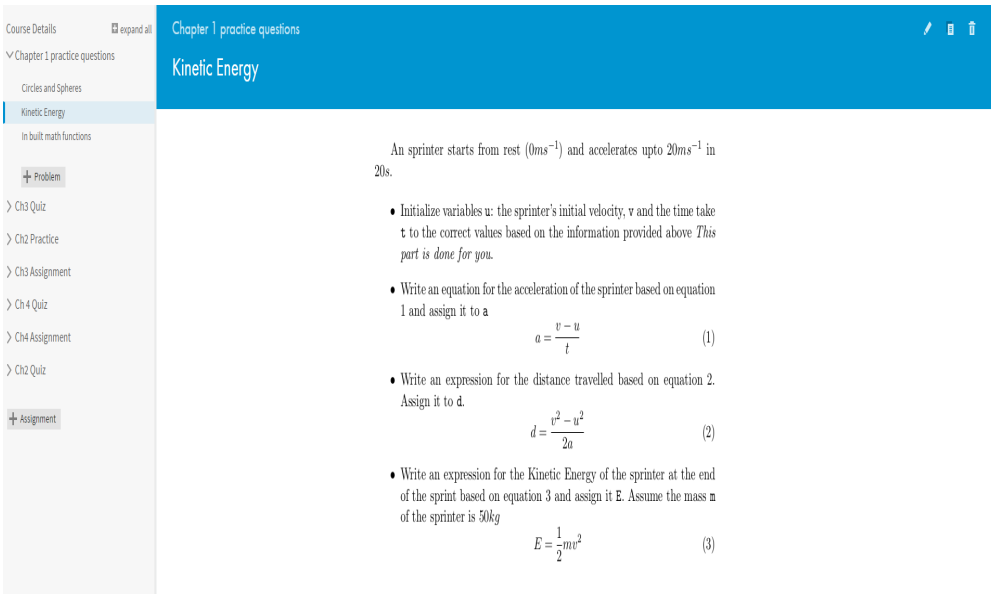

Figure 1. The Cody Coursework interface from a student's point-of-view

Figure 2: The test suite and test results

The interface from an instructor's point-of-view is similar. It allows the instructor to not only access but also modify the problem description, change the test suite as well as inspect solutions by students. The setup used to generate questions is shown in Figure 3.

The question description and the question template are visible to the students. The testing suite is primarily composed of a script that first creates a local copy of a reference function, which is used to evaluate a given solution. This reference solution usually resides in a server and a local copy is created using the MATLAB command **urlwrite**. Furthermore, local copies of other files that can be used in the testing process can also be created with this command. The format of a reference file is a MATLAB protected file format (.p). This hides information about the implementation of the reference function from the student.

Cody Coursework questions can primarily be of two types, scripts, and functions. In a script style question, the whole script is evaluated at once. Figure 1 contained a question regarding scripts. In this question, equations for various concepts associated with kinetic motion should be expressed with given variables. Then the testing suite is composed of assigning various values for the given variables and checking the values of expressions against these variables. In a script style question, Cody Coursework exploits the scope of the workspace variables in MATLAB.

In a function-style question a student completes a function with a given name. This function is tested against a reference solution. A function-style question is shown in Figure 3. In creating reference functions, it should be noted that if the reference function has the same function signature and behavior as the student's, the student could call the reference function from within his function rather than writing a solution that works autonomously. Hence the reference function's definition is extended to not only compute the correct answer but also to call the student function and compare the result. The reference function is redefined to return a Boolean result, which indicates whether the student function matches the correct answer as calculated by the reference function.

In both styles of questions, it is important to use random inputs, or a very large number of inputs, to prevent students from returning deterministic output values that can be output by the program without computing the solution. For example, if the testing suite only contained a few tests, then after a few runs with feedback, the student could write a program of the form "if A then output result1 else if B then output result 2" – producing the correct input-output behavior without being able to compute any other results correctly.

How Cody autograding was used in the class

The course consisted of a weekly one-hour lecture in which new concepts were first introduced by the faculty instructor and a two-hour lab in which these concepts were further elaborated using examples by teaching assistants. During each lab the teaching assistant would help students complete an ungraded, non-Cody, lab assignment. After this assignment, students would take a quiz administered through Cody Coursework that was 30 minutes long and generally consisted of one question. These quizzes contained material covered during the previous week's lecture and lab. The course also had two Cody-administered take-home assignments, which were done over weekends. These usually consisted of multiple questions.

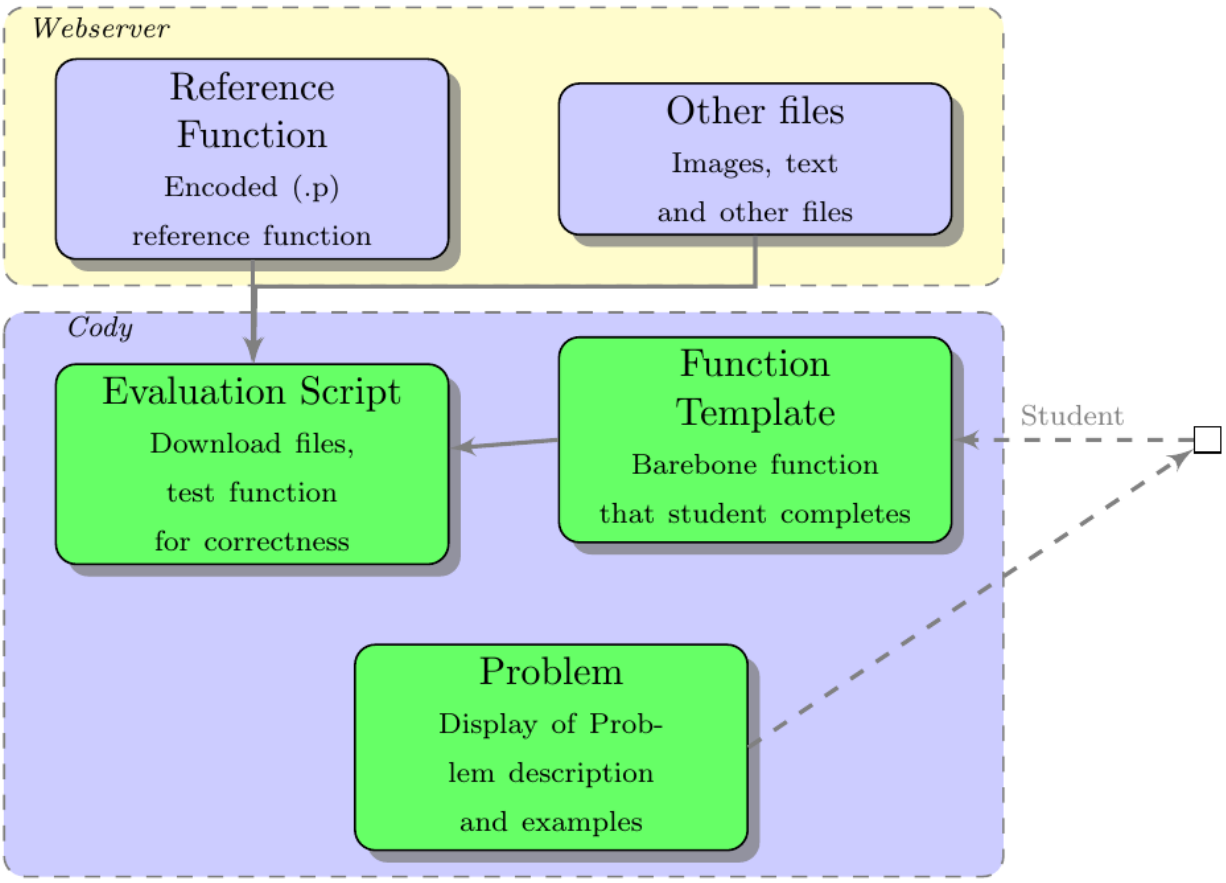

Figure 3: The test setup used in Cody Coursework

There was also a final exam and a midterm. These were traditional in-classroom proctored paper-and-pencil tests.

The course grade weighting was: quizzes 30%, assignments 20%, midterm exam 15%, and final exam 35%.

Autograding and authentic learning effort

1. *Checking for plagiarism.* Pieterse in her survey[3] points out that plagiarism may be a significant issue in a large course, where it may be frustrating for some students to figure out the fine details that may be preventing acceptance by automatic testing. Any system that only checks for correct output and execution is susceptible to inauthentic efforts based on taking copies of code (from friends, or from posts on social media). This leads to grading strategies where only proctored situations are heavily weighted. If the instructors are inclined to "save students from their shortcomings" by monitoring even unproctored efforts, then further measures such as automatic plagiarism detectors such as Moss, JPlag or similar systems[10], [11] can be adopted Follow-up or prosecution of inauthentic work can become a significant effort by staff. It may be useful to work out course policies and staff activities that affect this, as well as the information campaign to establish a culture that eschews binging on inauthentic activity.

In our case, since the take-home assignments were unproctored efforts, the students were asked to test their code via Cody as well as to submit their solutions on Blackboard Learn, an on-line~~online~~ course management system. The submitted solutions were then run through Moss by the teaching assistants to detect plagiarized code or code taken from on-line~~online~~ sources.

2. *Deterring simulation of a solution with finite case reaction.* The testing script has a small number of fixed tests for the program. This is attractive to the Cody author (less work for the testing script). However, it allows the submission of programs that provide the correct output only for the tests and not a general solution -- a program of the form "If input is A then output result1 else If input Is B then output result2 ….. else if input is F then output result6". The way to have a small number of tests without making them fixed is to use random number generation to select the inputs. By setting the seed so that it is different with each test run, this kind of approach is less likely to succeed.

3. *Deterring simulation of a solution by calling the reference code.* To provide more flexibility, the testing script calls a reference solution to calculate the solution on the fly to compare it to the student's computation. There are two issues here. The first is that the Cody environment loads the reference solution into the working directory where the test is run. So the student program could, if it knows the name of the reference solution, call it instead of computing the solution autonomously. It could also, as a side effect, print out the file containing the function definition as part of the test execution.

The first tactic can be deterred by modifying the reference function as explained in the previous section, so that the student can't easily create a test-passing function just by invoking the reference function. The latter tactic can be "deterred" by storing the reference solution in .p format (encrypted but still available for execution within MATLAB).

(2) and (3) suggest that there may be other ways to strategies for getting the autograder to accept the code without it containing an authentically produced student solution by having the student solution invoke information available in the working directory of the test execution, or accessing other files over the Internet. In part this is a design consideration to the execution environment available within Cody Coursework highly authentic with respect to that found ordinarily in MATLAB. Other systems, such as Maple TA, avoid some of the problems with Cody Courseware because they put the student program execution in a sandbox that cannot access other files or resources. Another run-time difference for other autograding systems is that, rather than having a standard "full disclosure" output of test execution that was available in Cody, the authors of the grading script get to select and program the information that they want to disclose to the student after the test run.

4. *Quick visual checks for outlying submissions.* Mathworks has suggested the following as a detection strategy: Cody provides a "Solution Map" [7], [12] that shows student submissions graphed as length of solution versus time. This can be used to detect certain kinds of outliers, for example, if students adopt strategies in (2) or (3) that turn out to have programs that are much different in size than an "authentic" solution. While not automatic, this analysis can be performed rapidly.

The cost of developing and using Cody problems

Pieterse points out[3] that one of the significant differences between autograded problems and normal hand-graded assignments is that autograding requires significant up-front effort to create tests and test results. Ambiguities or imprecision of specification can be left in manually graded assignments, leaving it to the graders to reward creative interpretations or solutions .The need for extra precision and removal of ambiguity often requires extra testing and quality assurance before the release of an autograded problem. In a course that used Maple TA, a similar autograding system, it was found that the problem authors estimated that five hours were needed to create, write up, and carefully test an autograded problem[13].

Balancing this is the fact that most of the cost of the use of autograded problems is that the cost of development does not depend on class size, so that large classes or multiple uses can reduce the per-student cost.

In the initial round of problem development, each quiz and assignment question took around 30 minutes to develop. This includes writing a problem description, writing reference functions, function template, testing scripts and uploading them to Cody and a fileserver. There were around 10 different questions that were generated for the same quiz. Each student was given only one of the questions, as the quiz was given across multiple time periods so needed different versions to provide some exam security.

There were around 40 questions generated for the 2 assignments. Each student got three of them. The majority of the questions generated were re-used from the first year to the second year. Only small amounts of question authorship time are necessary in reuse.

There were also ten practice questions, given without grade credit for the students to help prepare for quizzes and exams. Six of them were given in the first weeks, when the students were first learning how to work with Cody. These questions were completely reused from the first year to the second year.

We believe that an analysis similar to that found by Char[13] for Maple TA problem development holds for Cody, meaning that the per-student cost of problem development works out to a few dollars, less if the problems are reused in multiple terms. The current economics of Cody are more favorable than Maple TA since the testing is provided for free by Mathworks, whereas most automatic feedback services require covering the costs of operation, including licenses, system administration, and system acquisition and maintenance. For classes involving a thousand students or more, this would add a few more dollars per student should Mathworks begin charging for use of Cody Coursework. A definitive accounting will be in subsequent reports about this work, once the factors in start-up and long-term operation are acquired and analyzed.

Student and instructor reaction to use of Cody: instructor assessment

The lead instructor did not believe that they changed their teaching style due to the inclusion of autograding. However, they observed two effects on students.

1. *Given the large class size, the use of Cody allowed the TAs to focus more closely on those students who needed more tutoring.* More specifically, the TAs were able examine code that did not pass the Cody tests more closely and identify common errors in logic that students made when writing their code. These errors were then discussed in the subsequent lab session along with the correct solution. From the students' point of view, many of them found the error messages provided from Cody, especially those related to syntax errors, to be helpful for debugging.

2. *Initial experiences indicate that an additional educational issue for students is to learn how to work with remote testing and situations where the tests go beyond a few static cases.* The students were told to debug, as best as possible, their code on a local MATLAB installation prior to submitting to the Cody system. In terms of learning programming, Cody had minimal to no impact in the instructors' opinion, other than facilitating (1) Students were noticed repeatedly submitting the same incorrect code to the system hoping for the code to pass the tests. In part this was because of insufficient understanding of how the testing framework worked: the student assumed that since the code performed correctly on a particular (or a single) data set locally, the Cody system was "incorrectly" flagging their code as faulty, forgetting that randomized inputs were being used to test the code. The repeated submission phenomenon was a reaction similar to what happens when confronted with a car or appliance that is suspected of being faulty; this is in part due to the student not having a good mental model of the likely causes of failure or good working explanations for why the remote Cody results are different from the local results. This is consistent with the observations made by Pieterse[3], who noted that a number of studies of other autograding situations found that student testing maturity was an issue in successful use.

Evaluation of effort to date

From the perspective of gathering information useful in making decisions about courseware changes, new or alternative course exercises, course structure, etc. our evaluation of Cody Coursework and auto-grading as a teaching-learning tool is an ongoing activity that involves two phases, two years, and two types of evaluation. The two types of evaluation, as described by Scriven[14] are formative and summative. Formative evaluation involves gathering information focused on assessing the performance of the software which can be useful in fine-tuning the software and guiding the project towards its intended goals and relatively final form before looking at its impact on the students. In other words, are there unanticipated factors associated with the software or conduct of the course that necessitate a re-design of the plans, that would interfere with reaching the intended final goal or that might require setting one or more new goals? Summative evaluation however, is focused on the processes of judging or assessing the impact of the new software on student performance. In other words, is the new structure, tool, etc. more successful or beneficial than the previously implemented alternative in achieving the goals of the course?

In our project, the first type of evaluation is being applied iteratively as needed. While we continue to implement tools, materials, exercises, etc., during the formative evaluation phase of this project we regard the students as being both consultants and co-evaluators of the tools, etc. In our case, gathering feedback from our student consultants has involved using two surveys and the usage analytics provided by Cody Coursework itself.

We have constructed and administered a beginning-of-the-class survey with the goal of understanding more about the levels of experience and types of knowledge students bring to the course. We also have constructed a second post-class questionnaire asking the students to provide feedback on the effectiveness and usability of Cody Coursework as they used it in their course.

The first survey focuses on clarifying how many students bring which types of experience to doing their coursework, evaluations and on what their expectations are for using computers in their future, both in the immediate educational context and in the long range of their possible career paths and activities. This information is intended to assist the faculty in calibrating and/or modifying their ongoing choices of topics and exercise difficulty levels to better help the course audience achieve the course goals.

The second questionnaire, administered after the students have used Cody Coursework for a term asks students to evaluate it and give advice about its future use in the course. Essay responses are solicited to many of the questions to help us better understand what will probably be diverse student viewpoints about advantages, issues, and problems. Course achievement, as well as attitudes and expectations about computing from the first survey can be correlated to assessment. The purpose is two fold: to improve practices using Cody clarifying supplementary teaching materials for the exercises or for the practice of program testing itself, and to help make a set of recommendations to the Cody developers about ideas for improvement of both the features of Cody and the User Interface and Interaction flow with the Cody software. For example, some students might find it easier to learn when first presented with a programming problem broken up into parts each separately tested with Cody, before being asked to assemble the pieces into a program that solves the entire problem. This kind of "scaffolding" needs to be balanced pedagogically with the need to remove it eventually, since the desired goal behavior (the "real world" skill) is to program entire problems without such help.

The process of developing the second or summative evaluation phase of this project will require a major shift in focus. In this case summative evaluation will involve looking at the students as human subjects in a research study. We will be planning on looking at the effects of Cody Coursework, and other aspects of the course structure, on student performance variables, e.g., course grades, completion percentages, exercises and examination performance. In other words, we will have shifted from a post hoc analysis and review mode into an intentional manipulation of variables mode.

This shift of focus in how we view the students will require development of a human-subjects research proposal to be reviewed by the University's Institutional Review Board (IRB) as an investigation of the effects of Cody Coursework and auto-graded exercises upon student learning. During the coming months we expect to be engaged in re-evaluating course decisions, developing a written statement of the research plan to be conducted, including a final draft of any questionnaires and other measures to be used, as required by the University's IRB review protocols. Our current expectation is that there will be no experimental manipulation involved and primary analytic tools will involve correlational investigation of relationship. There are two reasons for this; first, we want to determine the existence of any relationships between the

software tools and student performance before introducing anything such as detailed work habit reports or personality variables, and, second approval for a proposal that minimizes the collection of new personal information that would not otherwise exist will lead to less complicated IRB conditions for approval of the research.

Future work

Another possibility as indicated by existing functionality in other systems is to generate problem variants. While this is not currently possible for Cody, it is possible for other autograding systems such as Maple TA[13], [15], [16]. Other systems allow only a limited number attempts for a variant before it is discarded and the student given a new variant to solve to discard brute force trial-and-error thinking. Other systems monitor student desktop activity, which allows a greater insight into authentic learning effort, as well as step-by-step insight into how students are approaching the implementation of their solutions (possibly through data mining)[17].

Auto generated problem variants have the advantage of making it easier to provide comparable quizzes and exercises when proctoring must run across many different class periods, as long as everyone gets the same hints about what might be given as a test during the proctoring period. Furthermore, problems with variants can be used both for practice and for assessment. QuizPack, an automated assessment system for C programming was found to encourage practice and enhancement accomplishment in programming when used in such circumstances[5].

We believe that the immediate feedback of Cody is most beneficial when the student is close to being able to complete a solution unaided -- the desired goal state of the learning. Using Cody's feedback to infer how to fix fundamental defects in algorithmic is not likely to be very productive use of time for the student. Stopping when Cody accepts a solution as correct also does not encourage further reflection on code improvement that might be sorely needed if the solution was obtained by "Brownian motion" code modification. Until functionality permits increased sophistication in analysis and feedback, instructors must work to devise policies and pedagogical use that heads off scenarios where students become mired in low-productivity use of the feedback system. For example, Malmi[18] et al. report better learning results from autograding are enhanced when a rationing scheme for resubmissions is instituted; this evidently causes more thinking and reflection.

Correctness of course is not the only merit criteria in programming. There is also execution efficiency, code readability and maintainability, and algorithmic simplicity to be cultivated in beginning programmers. There is no general consensus on the prominence these criteria should play in introductory programming courses. Automatic feedback for these criteria is not currently available in Cody. In some cases we might expect them to appear in the near future (e.g. Web-CAT [19] allows style checking), while other tools are emerging from MOOC tool research [20].

Conclusions

Our work so far has worked out a number of details for productive use of Cody Coursework autograding in a large course. We expect that in steady state operation, the economic case for Cody Coursework use will be established, taking into account the costs of development and the

amount of use. This establishes relevance and significance barring egregious negative effects on learning (which we have not observed to date).

With the basic framework for smooth and coherent use of autograding in the course, we can now put more effort into determining the differential effects on learning for different configurations or deployments of feedback. We also expect to learn principles and factors affecting successful long-term use of automatic feedback exercises as applicable to MATLAB programming. However, we expect that functionality of automatic feedback systems for programming will continue to rapidly evolve. Thus research into educational effects of a fixed functionality must contend with experimentation with significant new features and practices.

Acknowledgements

The Cody Coursework development team at Mathworks patiently answered many of our questions about it. Mathworks and the Drexel University College of Engineering, financially supported the development of the exercises used in our course.

References

[1] J. Hattie and H. Timperley, "The power of feedback," *Rev. Educ. Res.*, vol. 77, no. 1, pp. pp. 81–112, 2007.
[2] E. Lazowska, E. Roberts, and J. Kurose, "Tsunami or sea change? Responding to the explosion of student interest in computer science," *NCWIT 10th Anniv. Summit*, vol. 1, 2014 [Online]. Available: http://lazowska.cs.washington.edu/NCWIT.pdf
[3] V. Pieterse, "Automated Assessment of Programming Assignments," in *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research*, Open Univ., Heerlen, The Netherlands, The Netherlands, 2013, pp. 4:45–4:56 [Online]. Available: http://dl.acm.org/citation.cfm?id=2541917.2541921
[4] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of Recent Systems for Automatic Assessment of Programming Assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, New York, NY, USA, 2010, pp. 86–93 [Online]. Available: http://doi.acm.org/10.1145/1930464.1930480
[5] P. Brusilovsky and S. Sosnovsky, "Individualized Exercises for Self-assessment of Programming Knowledge: An Evaluation of QuizPACK," *J Educ Resour Comput*, vol. 5, no. 3, Sep. 2005 [Online]. Available: http://doi.acm.org/10.1145/1163405.1163411
[6] K. VanLehn, "The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems.," *Educ. Psychol.*, vol. 46, no. 4, pp. 197 – 221, 2011.
[7] Mathworks, "MATLAB Cody Coursework," *MATLAB Cody Coursework -- Grade MATLAB programming assignments automatically*, 2013. [Online]. Available: http://www.mathworks.com/academia/cody-coursework/index.html. [Accessed: 04-Jan-2014]
[8] Mathworks, "Cody Coursework for Instructors - MATLAB & Simulink," 2013. [Online]. Available: http://www.mathworks.com/help/coursework/cody-coursework-for-instructors.html. [Accessed: 04-Jan-2014]
[9] Mathworks, "About Cody," 2013. [Online]. Available: http://www.mathworks.com/matlabcentral/about/cody/. [Accessed: 27-Oct-2013]
[10] A. Aiken and others, "Moss: A system for detecting software plagiarism," *Univ. California–Berkeley See Www Cs Berkeley Eduaikenmoss Html*, vol. 9, 2005.
[11] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with JPlag," *J UCS*, vol. 8, no. 11, p. 1016, 2002.
[12] N. Gulley, "Cody's Solution Map » MATLAB Spoken Here," 02-Apr-2012. [Online]. Available: http://blogs.mathworks.com/community/2012/04/02/codys-solution-map/. [Accessed: 27-Jan-2016]
[13] B. Char, "Developing questions for Maple TA using Maple library modules and non-mathematical computation," Drexel University Department of Computer Science, 2011 [Online]. Available: http://www.drexel.edu/~/media/Files/cs/techreports/DU-CS-11-04.ashx

[14] M. Scriven, “The Methodology of Evaluation,” in *Perspectives of Curriculum Evaluation*, Chicago: Rand McNally, 1967.
[15] B. Char, “ASEE Webinar: Advanced Online Testing Solutions in a Freshman Engineering Computation Lab - Recorded Webinar - Maplesoft,” 16-Oct-2013. [Online]. Available: http://www.maplesoft.com/webinars/recorded/featured.aspx?id=569. [Accessed: 19-Oct-2013]
[16] Maplesoft, “Maple T.A. - Web-based Testing and Assessment for Math Courses - Maplesoft,” 2013. [Online]. Available: http://www.maplesoft.com/products/mapleta/. [Accessed: 04-Jan-2014]
[17] A. Altadmri and N. C. Brown, “37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data,” in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 2015, pp. 522–527.
[18] L. Malmi, V. Karavirta, A. Korhonen, and J. Nikander, “Experiences on Automatically Assessed Algorithm Simulation Exercises with Different Resubmission Policies,” *J Educ Resour Comput*, vol. 5, no. 3, Sep. 2005 [Online]. Available: http://doi.acm.org/10.1145/1163405.1163412
[19] S. H. Edwards and M. A. Perez-Quinones, “Web-CAT: automatically grading programming assignments,” *SIGCSE Bull*, vol. 40, no. 3, pp. 328–328, Jun. 2008.
[20] E. L. Glassman, J. Scott, R. Singh, P. J. Guo, and R. C. Miller, “OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale,” *ACM Trans Comput-Hum Interact*, vol. 22, no. 2, pp. 7:1–7:35, Mar. 2015.