Evaluating Programming Ability in an Introductory Computer Science Course

A.T. Chamillard
Department of Computer Science
U.S. Air Force Academy, CO 80840
719-333-7101
Tim.Chamillard@usafa.af.mil

Kim A. Braun
Department of Computer Science
U.S. Air Force Academy, CO 80840
719-333-4126
Kim.Braun@usafa.af.mil

Abstract

There are numerous ways to evaluate student programming ability, all of which have benefits and drawbacks. In this paper we discuss how we have combined a number of those evaluation techniques to assess student programming ability in an introductory computer science course and statistically analyze the relationships of student performance using the different evaluation techniques.

Keywords

Programming ability, programming evaluation, introductory computer science, collaborative learning.

1 Introduction

Professors teaching a course containing a non-trivial amount of programming are faced with a difficult decision - how do we evaluate the programming ability of the students in the class? The problem is not so much that we have no evaluation techniques; rather, the problem is that we can evaluate programming ability in so many ways it is difficult to determine the "right" way in a particular course. The problem is exacerbated by the fact that certain techniques may help a student's learning while hindering our ability to assess that individual's programming ability.

For example, researchers have found that collaborative learning can be an effective learning technique for many students [2]. The ability to discuss concepts and implementation details with other students during the course of a project helps certain students learn the material more effectively. The benefits of this approach seem clear, but how do we then evaluate an individual's ability from a project they created in collaboration with others?

If we require students to complete programming assignments totally on their own, we can easily evaluate each individual's programming ability. However, some students may struggle far longer than they need to with certain concepts or program constructs, when simply asking a fellow student could clarify the topics quickly. Does the benefit of easy evaluation outweigh the drawback of potential wasted time and frustration on the part of the students?

Because all the evaluation techniques have both benefits and drawbacks, it seems reasonable to develop a combination of those techniques. In this way, we can incorporate techniques that support learning (but hinder individual evaluations), techniques that allow easy individual evaluations (but may hinder learning), and techniques in between. This paper describes how we have combined a number of evaluation techniques in an introductory computer science course and demonstrates, through statistical analysis, the differences and relationships between those evaluation techniques.

All students attending the U.S. Air Force Academy (called USAFA hereafter) are required to take an introductory course in computer science (CompSci 110). Because the course is taken by all students in either their freshman or sophomore year, it assumes no prior knowledge about computers. The key topic in this course is problem solving with computers. Since students need to know how to solve problems before they can solve them using computers, we start by helping the students develop their problem solving skills. They then learn how to use these skills to solve problems using computers and the Ada programming language.

Students are evaluated in a variety of ways in the course, for a total of 1,000 points. The points are allocated to collaborative labs (180 points), a group case study (75 points), lab practica (160 points), tests (300 points), a final examination (250 points), and other miscellaneous activities (35 points). All but the miscellaneous activities measure student programming ability to some degree, whether individually or as part of a larger group effort.

The next section discusses the use of collaborative labs, while Section 3 presents our use of a group case study. Section 4 describes the lab practica we have incorporated. Section 5 provides statistical analysis of student performance on the various evaluation methodologies and

SIGCSE 2000 3/00 Austin, TX, USA
1-58113-213-1/00/0003...

the relationships between them, and the final section provides our conclusions.

2 Collaborative Labs

One method we use to evaluate programming ability is through the use of collaboration on programming labs. Students are given a total of 6 such labs over the course of the semester, with 2 hours of class time devoted to each of 4 labs. The labs range in difficulty from the first lab, which requires numerous variable declarations and simple textual and numeric input and output, to the last lab, which requires the use of arrays and file input/output.

In Lab 1, students learn to declare variables of various data types, accomplish input and output of those variables, and perform calculations. The key concept in the second lab is the use of procedures as a way to decompose a problem. The third lab introduces the students to selection statements to control execution paths in their program. Lab 4 requires that students use condition-controlled iteration to validate user input and count-controlled iteration to print a set number of calculated values. In Lab 5, students declare and use arrays as well as using count-controlled iteration to access elements of those arrays. The final lab requires that students combine arrays with file input and output. Testing considerations are included in each of the 6 labs.

Students are allowed to collaborate with each other when completing these labs, but only to a certain extent. For example, one student can ask another about a concept, a particular Ada construct, or a syntax error, but two students are not allowed to sit down at a single computer to complete the lab together. Each student must submit their own lab for a grade.

The rationale behind this approach is to make learning the programming concepts as easy as possible for the students. While USAFA instructors tend to hold extensive office hours, it is also reasonable to let students seek help on these assignments from their classmates. In this way, we try to avoid some of the frustration students feel when faced with a programming problem that they have to solve without help. Additionally, some students have learning styles that indicate that they learn better by studying with others [1]; collaborative work on the labs helps provide those students with an effective study environment. Conversely, allowing extensive collaboration on these labs makes it difficult to evaluate each individual's skill level.

3 Group Case Study

To expose students to the dynamics of group work, and to let them approach a problem that is larger than would be reasonable for a single student to complete in an introductory course, we divide the students into teams (of 2 to 4 students) to complete a group case study at the end of the semester.

Historically, this case study has been a game of some sort; in the recent past, students have implemented a portion of Connect-4, Battleship, Othello, and billiards games. On the last day of the class, we play the student programs against each other in a tournament, and allocate extra credit points based on group placings in the tournament.

The case study appears to be a very motivational experience for the students, with even the less gifted students developing strategies to help them in the tournament. Since the case study does not present any new programming concepts, it is not clear whether they gain additional programming skill while completing the case study. Anecdotal evidence suggests that the students are motivated by the case study, and they also gain experience developing portions of a program in a small group, which we believe to be beneficial. This again leads to difficulty, however, when we try to assess each individual's programming ability.

4 Individual Lab Practica

While we recognize the benefits of the collaboration on the lab assignments and the group case study, that same collaboration makes it difficult for us to evaluate individual programming skill for each student. To help us with this individual evaluation, we have students complete 2 lab practica over the course of the semester.

A lab practicum is an in-class lab that the students are required to complete within a set period of time (85 minutes). Students must develop and test a program solving a problem that they are presented with at the beginning of the time period. They are allowed to use a handout containing syntax for all the programming constructs covered in the course and a sheet listing common programming errors (and their solutions). They are not allowed to use any other materials, and the instructors will only answer questions about the problem (rather than helping students correct syntax errors, for example). In essence, these practica serve as programming exams that test the students' individual programming skill.

In the first practicum, held in the middle of the semester, students use procedures, selection statements, and both condition-controlled and count-controlled iteration. The second practicum, held approximately three weeks before the end of the semester, requires that students also use arrays and implement file input and output.

Not surprisingly, these practica are a source of stress for the students. In addition to the normal stress associated with taking an exam, they are also faced with the pressure of completing a program in a set period of time. We ensure that the lab practica are similar to the collaborative lab assignments they have already completed, but we have found that scores on the practica are significantly lower than on the collaborative labs (see following section). We have also found, however, that the students seem to be more inclined to seek extra help on programming concepts they don't understand as the time to take a practicum approaches. This evaluation instrument may therefore both support student learning (as they try to understand

programming concepts) and evaluation of each individual's programming skill.

5 Statistical Analysis

In previous sections, we have described each of the evaluation techniques we use to evaluate student programming ability, and we have noted strengths and weaknesses of each technique in an informal manner. Because all of the techniques have the same general goal, it is also interesting to compare student performance on each of the evaluation techniques. Specifically, we would like to examine the differences between the distributions of scores on each evaluation technique, as well as the relationships between student performance on each of the evaluation techniques.

Our dataset includes scores for 1,822 students from 4 semesters: Fall 1997, Spring 1998, Fall 1998, and Spring 1999. We begin by examining the standard summary statistics for each of the techniques; means and standard deviations are provided in Figure 1. All values in the figure are given in percentages. For the final examination, we include only the portion of the exam that specifically addresses programming, and the test percentages reflect student scores on the complete tests, including non-programming questions (because data at a finer resolution is not available). We also note that only 1,712 students are included in the final exam statistics, since the other 110 students were excused from the final exam.

The summary statistics provide us the opportunity to make some informal observations. For example, we note that the percentages on the collaborative work (labs, case study) are much higher than the scores for the work that students accomplish individually (practica, tests, and final exam). Part of this difference is surely due to the fact that the individual work is conducted in a controlled, timed environment, while for the collaborative work the students can use as much time as they need to accomplish the assignments. Some of the difference may also be accounted for by the collaborative versus individual nature of the evaluations, but it is unclear how we can quantify the extent of this effect.

One of the challenges faced by the administrators of this course is responding to pressures to reduce the number of evaluations performed over the course of the semester. Although the evaluation techniques in the course have different benefits and drawbacks for both the students and the instructors, we would also like to formally demonstrate that each of the evaluation techniques provides a unique evaluation opportunity within the course. More specifically, we would like to demonstrate that each of the evaluation techniques evaluates different student skills, or at least evaluates the same skills in slightly different ways. In other words, we would like to compare the means for the different evaluation techniques to show that the evaluation techniques yield different distributions.

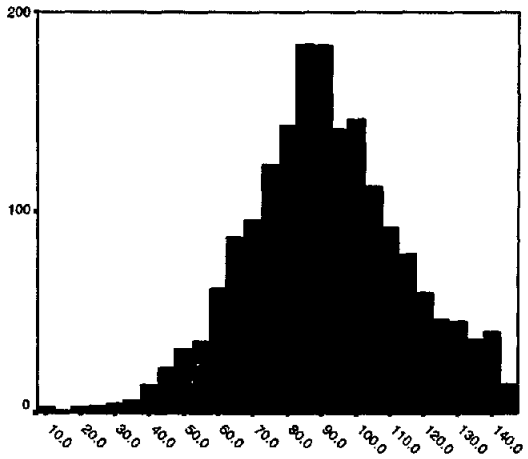
The paired samples t-test is a very common method for comparing the means of two distributions. The t-test uses the means and standard deviations for the two distributions to try to reject a null hypothesis that the two distributions could have been drawn from populations with equal means. One of the underlying assumptions of the t-test, however, is that the distributions being compared are normal distributions. We can check normality in a number of different ways. Informally, we can examine the distributions to see if they "look" like normal distributions. More formally, we can conduct the Kolmogorov-Smirnov Test for normality on each distribution. The Kolmogorov-Smirnov Test tries to reject a null hypothesis that a particular distribution is normal (thereby strongly implying that it is non-normal). In Figure 2. we provide the "most normal" (labs) and "least normal" (case study) distributions for the evaluation techniques. Informal inspection of the distributions indicates that they are not normal. When we apply the Kolmogorov-Smirnov Test to the distributions, we are able to reject the null hypothesis, again implying that each of the distributions is non-normal. In other words, we do not have good informal or statistical support for the assertion that the distributions are normal, so the paired samples t-test does not apply.

There are other, non-parametric tests that we can apply instead of the t-test. For example, the Wilcoxon Signed Ranks Test does not assume normality of the distributions being compared [3]. It does, however, assume that the distributions are symmetrical (they are not, by inspection). Instead, we can use the weaker Sign Test to compare the means. When we do so, we find that we have statistically significant evidence that each of the evaluation results are drawn from populations with different means. We therefore have firm evidence that each of the evaluation techniques is different from the others, lending support to our position that all of the techniques should be retained in the course.

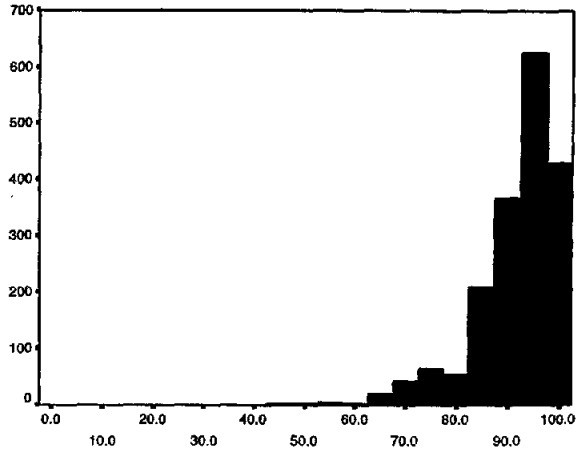
It is also interesting to consider how each student's performance on the evaluation techniques are related. For example, does a student who does well on the collaborative

	Labs	Case Study	Practica	Tests	Final Exam
Mean	90.81	91.78	77.90	73.56	74.99
Standard Deviation	23.40	8.55	14.85	11.31	16.84

Figure 1. Summary Statistics



Labs



Case Study

Figure 2. Example Distributions

labs tend to do well (relatively speaking, since the mean is lower) on the lab practica? To examine these relationships, we correlated each pair of evaluation technique results. For each such correlation we calculated Pearson's Correlation Coefficient and a measure of statistical significance (p). The coefficient ranges from -1.0 to 1.0, with a coefficient magnitude close to 1.0 indicating a strong linear relationship and a magnitude close to 0.0 indicating no linear relationship. All of the correlations were found to be statistically significant using the common guideline (p=0.05); the correlations are given in Figure 3. We point out before continuing that correlation is not a measure of causality; it simply measures the linear relationship between two variables. Additionally, we note that a low correlation only indicates that the variables are not linearly associated; they could still be related in some non-linear way.

While all of the correlations were statistically significant, we limit our discussion to those correlations that are also relatively strong (for our data, above 0.4). The strongest

Pair	Correlation
Labs/Case Study	0.249
Labs/Practica	0.569
Labs/Tests	0.603
Labs/Final Exam	0.484
Case Study/Practica	0.155
Case Study/Tests	0.193
Case Study/Final Exam	0.145
Practica/Tests	0.618
Practica/Final Exam	0.593
Tests/Final Exam	0.679

Figure 3. Correlation Results

correlation we find is between the Tests and the Final Exam. This is not particularly surprising, since they are very similar evaluation techniques in terms of both format and intent, though we do note that the tests also evaluate non-programming knowledge.

The next 2 strongest correlations are for the collaborative lab/test pair and the practica/test pair. The strength of these correlations indicates that students who do well on the programming assignments, whether they are collaborative or individual, also tend to do well on the tests. This assertion is further supported by the fact that the correlations for the collaborative lab/final exam and practica/final exam pairs are also relatively strong, again implying that students who do well on the programming assignments also tend to do well on more traditional tests (in this case, the final exam). We also note that the correlations between the practica and the tests and final exam are slightly higher than those between the labs and the tests and final exam. We explain this by pointing out that the practica are more like exams than the collaborative labs are, particularly given the controlled and timed environment for the practica.

The final correlation to discuss is between the collaborative labs and the practica. The strength of this correlation is certainly not surprising, since both evaluation techniques focus on the development and testing of programs. We might even expect the correlation coefficient to be higher than it is, but we believe this is not the case because the labs are collaborative while the practica are accomplished individually. This argument is particularly compelling given the fact that students with a propensity for collaborative work would tend to do better on the labs than the practica.

It is important to note that all of the weak correlations involve the group case study. Performance on the case study does not appear to be strongly connected to either the

programming assignments or the tests or final exam. We suspect that this is caused by a number of factors, including the fact that this is a pure group project (in contrast to the collaborative labs or individual practica), that group dynamics in the groups of 2 to 4 students play a part in each group's performance, and that multiple programmers are working on the project, so each student's grade is based on the work of others as well as their own work. We therefore do not view the group case study as a crucial evaluation technique, especially given the distribution provided in Fig. 2. Instead, we view it as an opportunity to let our students experience the benefits and drawbacks of working in a group environment.

6 Conclusions

Instructors teaching beginning programming skills are faced with difficult decisions when selecting which evaluation tools to use to evaluate those skills. Some techniques seem to lend themselves to cooperative learning while making assessment of individual skills difficult, other techniques lead to straightforward assessment but may not support student learning as well as they could, and yet other techniques seem to fall somewhere in the middle.

In this paper, we described how we have incorporated a number of evaluation techniques in an introductory computer science course. We also showed through a statistical comparison of means that each of the evaluation techniques evaluates a different set of skills (or the same set in different ways). Finally, we examined and discussed the correlations in student performance on the evaluation techniques.

There are still open questions remaining, of course. For instance, it would be instructive to measure the effects of collaborative learning compared to individual learning on an identical set of tasks (such as the labs described in Section 2). We could certainly design such an experiment, but we would of course have to ensure that neither group would be "punished" for participating in the experiment (e.g., receive lower grades).

References

[1] Chamillard, A.T. and Karolick, D., Using Learning Style Data in an Introductory Computer Science Course, In *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*, New Orleans, Louisiana, March 1999.

[2] Davis, B.G., *Tools for Teaching*, Jossey-Bass Publishers, San Francisco, CA, 1993.

[3] Neter, J., Wasserman, W., and Whitmore, G.A., *Applied Statistics*, Allyn and Bacon, Inc., Boston, MA, 1978.