A Successful Online Systems Class using Scaffolded Active Learning and Formative Assessment*

Michael O. Lam and Dee A. B. Weikle
Computer Science
James Madison University
Harrisonburg, VA 22807
{lam2mo, weikleda}@jmu.edu

Abstract

This last year has been a challenge for faculty transitioning from in-person instruction to online instruction. We approached the semester with serious concerns but discovered our hybrid course transitioned well to being completely online with a few key modifications: video versions of all lectures, Google slide implementations of in-class labs, course procedures that allowed students to choose their breakout room in Zoom, and randomization of question selection for midterm and final exams. In person, this course made thoughtful use of active learning and formative assessment to scaffold students into summative assessment. The online version does the same and, in our opinion, this contributed to the online success. This paper discusses the scaffolding formative assessments, how they build to the summative module tests and midterm and final exams as well as how we implemented active learning online. We also provide links to the formative labs and videos that made the active learning work.

1 Background and Related Work

Here we provide a discussion of formative assessment, primarily through selected work from computer science education literature because a comprehensive review is outside the scope of this paper.

Formative assessment is assessment designed to facilitate student learning. The assessment highlights student misconceptions for instructors so that they can tailor instruction and help students to self-assess and to correct misconceptions. This correction can take place in the classroom or in some cases by feedback to the student followed by an opportunity to retake the assessment. In contrast, summative assessment does not provide opportunities for students to resubmit work and is designed primarily for grading. Interested readers can see Black [1] and Dunn [6] respectively for a more in-depth literature review of formative assessment and a critique of the related research.

Nelson [9] provides an example formative assessment for tracing JavaScript programs along with a discussion of Kane's framework for assessment validity. It provides a detailed analysis of their formative assessment arguing for its validity using the four parts of Kane's framework: scoring, generalization, extrapolation and use. This article is particularly useful for its discussion of assessment design, explaining the importance of granularity in formative assessment such that questions clearly identify the specific skill being assessed and examples of confounds such as slips (misreading an operator) and guesses that might obscure the results of such an assessment. By carefully structuring their formative assessment they demonstrate the application of the validity argument and the process of designing such an assessment. Blaheta [2] describes transforming traditional pen and paper homework assignments in an Artificial Intelligence class into cooperative formative assessments by allowing students to submit homework in self-selected groups, receive comments, and resubmit for a final grade. Duarte [5] describes a blended learning environment where engineering students are given formative assessments in a learning management system (LMS) in addition to attending class face-to-face with the goal of improving learning outcomes and preventing premature withdrawal or unexpected failure at the end of the course. She saw performance correlation between the formative assessments in the LMS and final exam performance. Cauley and McMillan [4] discuss five best practices in structuring formative assessment: 1) provide learning targets, 2) give targeted feedback, 3) clearly attribute successes to effort, 4) encourage self-assessment, and 5) help students set attainable goals. Finally, Grover [7] provides an argument for formative assessment (specifically in the K-12 context but we believe the argument is applicable to higher education as well).

Our work is not as formal as that in Nelson, but is more of a combination of the approaches taken by Blaheta and Duarte. We make use of four types of formative assessment in the in-person version: 1) reading/conceptual quizzes in our LMS (Canvas), 2) instant-feedback questions during short lectures, 3) labs on paper that may include running or developing small programs, and 4) larger programming assignments in C where students are able to run provided

unit and integration tests to make sure their code is working properly prior to grading. In addition, we provide links to the labs via Google slides and companion videos that enabled this approach to work completely online. Section 2 describes the overall structure of the course and goes into detail about the different formative assessments.

2 Course Structure

Our course, Computer Systems I, is a basic computer architecture class combined with medium- to large-sized programming projects in C along with selected basic operating systems concepts. It serves as the foundation upon which our other computer systems courses build. Figure 1 is a high-level view of the topics covered. The course currently uses Computer Systems: A Programmer's Perspective by Randal E. Bryant and David R. O'Hallaron (CS:APP) [3]. The learning outcomes are as follows:

1. Explain the machine-level representation of data and code.
2. Summarize the architecture of a computer.
3. Explain how powerful, complex systems can be built from simple logic circuits.
4. Translate high-level code blocks into assembly and machine language
5. Write code to emulate the functionality of a computer.
6. Cultivate a sense of power and control over computer systems.
7. Gain an appreciation for the tools that facilitate software development.
8. Develop a sense of play when writing code.
9. Appreciate the principles and complexity of systems-level software.

Pre-pandemic, this course was a hybrid course designed for a 75-minute twice-per-week meeting format. Some assessments were completed online in Canvas, but there was a significant in-person experience. Before each in-person class period, students were assigned a reading from CS:APP and a Canvas quiz to complete online prior to coming to class. Each class started with a chance for students to ask questions about the quiz followed by a mini-lecture with instant-feedback questions. The second part of each class consisted of a lab worksheet that students worked on in groups. In addition, there were five programming projects, five Canvas module tests, and two in-class paper exams (a midterm and a final). The module tests were summative timed assessments associated with each module that students could take over a period of 2-3 days after a module had been fully covered in class. The modules correspond to the five high-level topics shown in Figure 1.

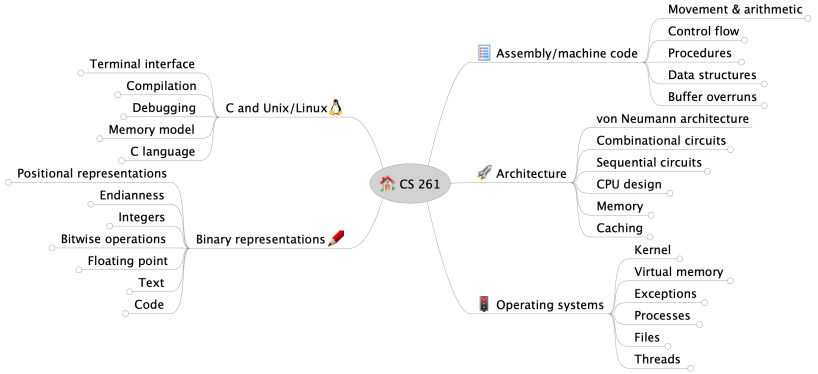

Figure 1: Computer Systems 1 Topics

2.1 Canvas Quizzes

The Canvas quizzes began as a way to make students accountable for the reading assignments but have evolved over time to include additional questions on basic concepts. These quizzes are formative because students see which questions they get wrong and are allowed to review the material, ask questions of each other or the instructor, and retake the quiz once. The quizzes make it clear what the instructors want students to know coming in to class. As class begins, the answers are available to students and they have the opportunity to ask questions about any remaining confusion from the quiz.

2.2 Instant-Feedback Questions

After students read about the content for the day and take the quiz, faculty lecture on the material in class, emphasizing concepts and giving examples. These lectures include another type of formative assessment – instant-feedback questions geared toward making sure students can remember key lecture concepts and do simple problems associated with those concepts. We used Socrative, an online Q&A platform that allows students to enter a code and answer questions associated with an ongoing lecture. Students are given credit for simply participating regardless of whether their answer is correct. The instructor can see the incorrect answers and can share the distribution of answers with the students. The instructor then gives the right answer, explaining how to get it, and why the other answers are wrong. This is another opportunity to correct student misconceptions in class.

2.3 In-class Paper Labs

Once the lecture is finished, the second part of the 75-minute class is dedicated to doing paper labs in groups of 2-3 students. These labs may ask students to run code on a lab machine or the student's personal computer. Each lab is a two-page series of questions designed to take students deeper into the material, scaffolding them into later exams or programming assignments. During this time, faculty monitor progress and answer student questions or point out mistakes as they crop up. Students must each turn in their own paper by the next day, but they are encouraged to work in groups both inside and outside of class. With labs due the next day, students also have an opportunity to work together more, go to open TA lab hours, or ask questions on a Q&A forum such as Piazza. Q&A responses are shared across all sections and monitored by both instructors. Questions can be answered by other students in the class or by the instructors and viewed by all students. In this forum, students can ask and answer questions showing up as anonymous to other students. With feedback during and outside of class, the labs provide another deeper formative assessment. Labs are then loosely graded with incorrect answers marked, but about half of the points awarded simply for turning in the assignment. Solutions are released to show students what was expected or possible correct answers. As with other course elements, discussion with faculty about any remaining confusion is encouraged, but students are not allowed to redo the labs after grading. A short description of each lab is included in Table 1.

2.4 Programming Assignments

Programming assignments for this course are described in detail by Weikle [11]. These assignments are designed to illustrate the underlying concepts while practicing C programming. Projects include tasks such as opening a binary file and reading in an object code header, loading object code regions into a memory array, decoding and disassembling machine code instructions from that code, and simulating the fetch-decode-execution cycle of machine code instructions. We use the Y86-64 instruction set architecture from CS:APP [3] with our own custom ELF-like object file format.

Programming assignments are released every 2-3 weeks and require students to apply the concepts after they have been introduced in readings, class, and labs. Students complete the programming assignments individually and are expected to collaborate only on concepts. In addition to the specification, students are given all the tests used to determine the functionality of their code, some of which are private (pre-compiled and stripped of symbols) while others are public (source code provided). These tests are organized into sets of tests for a particular grade level in a manner inspired by specification grading.

All tests for a particular grade level must be passed as well as all tests for any grade below that level to receive that particular grade. For example, to get a D a student would only need to pass the D level tests, but to get a C, they would have to pass all the D tests and all the C tests. Students are able to run the whole test suite at any time in development and receive automated feedback about their errors. This also enables them to look at the public tests for examples of how to test their code themselves. Finally, the bundling of tests into grade levels gives students a general guide to the most effective order of functionality implementation (i.e., the lower grade levels are the easier tasks to implement). In addition to the automated feedback and support, students can visit TA lab hours and instructor office hours as well as post questions to the previously mentioned Q&A forum.

3 Transitioning Online

The transition to a fully-online format kept the same routine with two significant modifications: 1) videos instead of lectures and 2) synchronous class in Zoom using Google slides for labs. We provided pre-recorded videos rather than lecture via Zoom because the videos can be downloaded and watched at any time even with a slow connection, be slowed down or sped up at the discretion of the student, and used for review or if a student missed class. One side effect of this change is that now the entire class period could be used for active learning using the labs, which we knew would be more difficult online. To maximize the effectiveness of the videos we limited their length to an average of 5-10 minutes and created playlists so students could break up their viewing sessions (this also simplifies searching for review content later). The instant-feedback questions were still provided in the videos with a prompt to pause the video and answer the question, followed by a description of the answer.

Active learning is key to keeping students engaged and maximizing learning. We found our method worked well to provide students support with technical content as well as helping them interact with other students to make friends and keep motivated. Prior to class, we set up a Google Doc with directions, announcements, and a set of breakout rooms (a Zoom feature that allows one call to have multiple isolated sub-calls; this feature is now also provided by several other videoconferencing providers) that students could sign into when they came to class. We also set up Google Slides with the lab questions as a static PDF background, adding text boxes or base diagrams for students to edit as their answers to the questions. [10]

When students entered the Zoom session (linked from Canvas), they also opened the folder for the day that contained the sign-up sheet and the empty lab template. Students signed up for a breakout room and then made their

own copy of the Google slide version of the lab. After an initial discussion with announcements, the instructor opened the breakout rooms and students worked together in these groups to complete the labs. Much like when students come to class in person and tend to sit with the same group of classmates each class period, the students in a particular breakout room were often the same, creating friendships and strong working relationships. Instructors could then monitor the Google slides in real time, making comments as students worked to encourage them or help correct their thinking. Students could also ask for an instructor to come into the breakout room to ask for help if they were stuck or unsure. If several groups were asking for help with the same issue, the instructor had the ability to bring everyone back to the main room, give a quick explanation of the issues allowing for questions, and then send everyone back to their breakout rooms.

Labs were due toward the end of the day. To turn in labs, each student had to download their own Google slide version as a PDF file and submit to Canvas. This allowed students to have more time if they needed and some would get together outside of class time to complete labs or to help each other with difficult concepts or sections. In most cases though, student could complete the majority of the lab in class. Labs were still loosely graded by the instructor with the majority of the credit going to turning in something. Solutions were released after the due date similar to in-person classes. The formative aspect of these assignments is that faculty can give feedback and guide students, noticing misconceptions and correcting them during class just as if we were in person.

The last modification we made in the online environment was the administration of the midterm and final exams. In person, these exams are held in the classroom and proctored by the instructor. Online, we made these exams in Canvas using primarily multiple choice, matching, and fill-in-the-blank questions. Our primary concern was to test similar skills online to what we have done in person to make sure that students are prepared for the next class in our systems curriculum. To this end, we used very similar questions or the same questions as our previous exams. Occasionally this required us to modify the question type slightly (e.g., change draw-a-diagram to multiple-choice with plausible distractors). A second concern was to prevent online cheating to the extent possible. Here we chose to provide an environment that made it convenient for students to be honest, realizing that it is impossible to prevent dishonestly completely in an online format and preferring to avoid disadvantaging honest students for whom more intrusive cheat detection would be an issue (e.g., internet connection quality or anxiety disorders). Our first mitigation strategy was to re-parameterize the questions from our four previous years of exams and use the Canvas feature which will randomly pick a question from a quiz bank for each question on an exam. When giving the exam, this makes it

highly unlikely that any two students will be given the same exam. In addition, we asked students to indicate they were upholding the honor code and required them to turn on their webcam video for a few minutes at the beginning of the exam session, greeting them by name as they did so. These latter techniques were motivated by the research that indicates institutional honor codes and better faculty-student connections reduce cheating. We felt that establishing contact right at the beginning of class reminded students of our connection in normal class periods and reminded them that we valued them doing their own work on the test. [8]

Our lab materials are available under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License: `https://bit.ly/lam-weikle-ccsc21`

4 Discussion

When initially contemplating teaching Computer Systems I online, we were concerned we would lose more students than in a typical in-person semester. However, our experience was that with this format our D/F/W rate was similar and grades on individual assignments were similar if not slightly higher. There are many reasons grades might have been slightly higher including: incremental improvements were made to the schedule and assignments, all exams were open book and open note online, videos could be reviewed when class was missed, withdrawals were allowed at the very end of the semester, and we may have graded more generously in the middle of the pandemic. However, we also received very positive feedback on student evaluations in both the survey comments section and via personal conversations. Anecdotally, we also saw new relationships formed between students, and the faculty felt connected to the majority of the students in class, similar to our prior experience in person.

We were particularly pleased with dedicating the entire class period to labs and active learning, primarily because more students were able to complete the labs during the class period but also because it reduced lecture prep time during the semester. We intend to continue using these videos and add to them for future in-person semesters so labs can make use of the entire class period. We also found that having students submit their labs electronically as PDFs significantly reduced the time spent grading because it is integrated with our LMS. We plan to continue this policy in person as well, observing that high-quality scanning apps are widely available for cell phones. Such a policy also makes it easier to accept late work on an ad-hoc basis.

5 Future Work

While our course was designed thinking about formative assessment and how each aspect of the course would build on other aspects, the transition online has encouraged us to be even more intentional about the formative aspect. We intend to review the Canvas quizzes and add some of the instant-feedback questions from the lectures and possibly more directed feedback for wrong answers. We also intend to reconsider the reading assignments in an attempt to reduce the amount of text required and focus on the most important concepts, in part because there is now a greater amount of time required to watch the videos before class. Finally, we intend to consider the work in Nelson [9] with the goal of making more rigorous arguments for the validity of our formative assessments in future semesters.

6 Conclusion

We were pleasantly surprised by how well the transition to online teaching went in our hybrid course built around formative assessments. Our primary contributions in this effort include video versions of all lectures, Google slide implementations of in-class labs, course procedures that allowed students to choose their breakout room in Zoom, and randomization of question selection for midterm and final exams. Informally, we found that student performance was equivalent or possibly even better than in previous semesters with tangible benefits to the instructors as well, and we plan to continue using (and improving) this course design in the future.

References

[1] Paul Black and Dylan Wiliam. Assessment and classroom learning. *Assessment in Education: Principles, Policy & Practice*, 5(1):7–74, 1998.

[2] Don Blaheta. Reinventing homework as cooperative, formative assessment. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, page 301–306, New York, NY, USA, 2014. Association for Computing Machinery.

[3] Randal E. Bryant and David R. O'Hallaron. *Computer systems: A programmer's perspective*. Pearson, 2019.

[4] Kathleen M. Cauley and James H. McMillan. Formative assessment techniques to support student motivation and achievement. *The Clearing*

House: A Journal of Educational Strategies, Issues and Ideas, 83(1):1–6, 2010.

[5] Marina Duarte. Formative assessment in b-learning: Effectively monitoring students learning. In *Proceedings of the Second International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM '14, page 497–501, New York, NY, USA, 2014. Association for Computing Machinery.

[6] K. Dunn and S. W. Mulvenon. A critical review of research on formative assessment: The limited scientific evidence of the impact of formative assessment in education. *Practical Assessment, Research and Evaluation*, 14:1–11, 2009.

[7] Shuchi Grover. *Toward A Framework for Formative Assessment of Conceptual Learning in K-12 Computer Science Classrooms*, page 31–37. Association for Computing Machinery, New York, NY, USA, 2021.

[8] Patricia A. Hutton. Understanding student cheating and what educators can do about it. *College Teaching*, 54(1):171–176, 2006.

[9] Greg L. Nelson, Andrew Hu, Benjamin Xie, and Amy J. Ko. Towards validity for a formative assessment for language-specific program tracing skills. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*, Koli Calling '19, New York, NY, USA, 2019. Association for Computing Machinery.

[10] Jill Staake. Google slides 101: Tips and tricks every teacher needs to know. `https://www.weareteachers.com/google-slides/`, Dec 2020.

[11] Dee A. B. Weikle, Michael O. Lam, and Michael S. Kirkpatrick. Automating systems course unit and integration testing: Experience report. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 565–570, New York, NY, USA, 2019. Association for Computing Machinery.

Table 1: Labs

Lab Number and Name	Description
01-intro	Introduction to a computer system
02-cmd_line	Linux command line and C compilation
03-c_intro	C memory model and pointers
04-arrays_strings	C arrays and strings
05-structs_io	C structs and I/O using fread and fgets
06-getopt_debug	Command line parameter parsing and debugging
07-binary_info	Binary/ hex representation and bitwise operations
08-integers	Integer encodings (unsigned, 2's compl.) and shifts
09-bin_arith, fp-intro	Binary arithmetic, intro to floating point
10-floating_point	Floating point representations
11-asm_intro	X86-64 assembly basics
12-asm_data	X86-64 data movement and arithmetic
13-asm_ctrlflow	X86-64 control flow
14-asm_proc	X86-64 procedures and runtime stack
15-asm_misc	X86-64 data structures and floating point
16-y86	Y86-64 simplified assembly (from CS:APP)
17-cmb_circuits	Combinational circuits
18-seq_circuits	Sequential circuits
19-arch_pipelining	Architecture and pipelining lab
20-y86_semantics	Y86 semantics lab w/ CPU pipeline stages
21-memory	Memory hierarchy and technologies
22-caching	Caching and set associativity
23-virtual_mem	Virtual memory and address translation
24-processes	Exceptions and processes
25-files	File systems and I/O
26-threads	Threads (preview of next systems course on concurrent computing)