Caches as an Example of Machine-gradable Exam Questions for Complex Engineering Systems

Suleman Mahmood
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL, USA
msm6@illinois.edu

Mingjie Zhao
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL, USA
mingjie7@illinois.edu

Omar Khan
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL, USA
mkhan259@illinois.edu

Geoffrey L. Herman
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL, USA
https://orcid.org/0000-0002-9501-2295

***Abstract*—This Innovative Practice Full Paper presents a framework for generating computer-based exams for complex engineering systems (such as cache memories) that can be machine graded while still offering partial credit for students. Complex multi-faceted engineering systems often require long, multi-part problems to fully assess students' understanding of those systems. Cache memories represent one such system in computer architecture courses. Traditionally, we assessed students' understanding of caches using comprehensive, multi-part questions in a paper-based exam. Grading these exams was time-consuming and relied on subjective grading. To cope with rising enrollment, we sought to address these issues by developing machine administered and gradable exams that did not heavily rely on multiple-choice questions or exact numerical responses. Additionally, this system needed to provide partial credit, a common expectation of our students. We developed a cache simulator to use as a back-end for our questions. We used the simulator to develop exam questions and new homework assignments to help students practice cache memory concepts. To give students access to fair partial credit, we allowed multiple submissions for the exam questions with limited feedback. We also awarded partial credit for answers within certain tolerance of the correct answer. The partial credit awarded reduced as deviation from the correct answer increased. Consequently, students could correct minor mistakes or propagating errors which are common reasons for awarding partial credit. To evaluate the effect of the switch from paper-based to computerized exam, we ported questions from one of our paper-based exams to a computerized exam. We evaluated the differences in student performance on paper-based version and the computerized version of the questions and found mixed results with students performing comparably or better than the paper-based exam on the computer-based exam. We also surveyed students about their experience with the computer-based exam. Students overwhelmingly indicated a preference for the computer-based exam. We believe that ideas from our work can be used to automate generation, administration, and grading of complex multi-part questions in engineering disciplines beyond computer architecture.**

***Index Terms*—Computerized Testing, Cache Simulator**

I. Introduction

Engineering students must learn how to decompose and solve novel and complex problems [1]–[4]. Engineering instructors frequently discuss that it is important for students to develop systems-level thinking: Students need to be able to reason not only about how each component works but how changes in one component or aspect of a system affects all other aspects of that system [5]–[7]. Consequently, instructors may deploy multi-part exam questions that explore students' understanding of multiple facets of an engineering system and how they interact to assess students' problem-solving skills and systems thinking. Indeed, even the Principles and Practice of Engineering (PE) exams use multi-part questions [8]. We have historically used these types of multi-part questions in our computer architecture exams to assess how students reason about the complex interactions between computer code, data structures, and cache design.

A common feature of multi-part questions is that the solution for a part of a question may depend on solutions from other parts. For example, students may analyze a system under one set of constraints and then repeat that analysis under another set of constraints and then be asked to compare and contrast the performance of the system under the different constraints (e.g., laminar vs. turbulent flow or transient vs. steady state responses). In these types of situations, instructors may award partial credit to students for whatever aspects of the problem the student was able to correctly analyze and may even need to award partial credit when errors propagate from one part to the next. Awarding partial credit like this requires case-by-case decisions, which makes grading a time-consuming process, especially for classes with large enrollments. It can also lead to subjective grading even in the presence of a well-defined rubric when multiple instructors (e.g., graduate teaching assistants) divide the grading. In addition to the grading difficulties, the solutions of the questions become commonly available among students over a few semesters necessitating overhauling questions. These considerations and limitations ultimately led us to believe that continued use of paper-based exams was untenable, but we did not want to turn to multiple-choice or short-answer questions to assess students' knowledge.

Modern learning management systems can help in addressing some of the problems described above. At our university, we use PrairieLearn [9] which is an open-source, online problem-driven learning system [10]. PrairieLearn was designed to enable instructors to create homework assignments and exams that can harness the full power of the web to assess students' knowledge. For example, in PrairieLearn, students can draw free body diagrams or finite state machines or circuits. Students can write code in almost any programming language from Python to R to Matlab and have it auto-graded. This system lets us build any arbitrary piece of software to assess students' knowledge in whatever manner we deem best.

Consequently, PrairieLearn provides many flexible mechanisms to grade students' work consistently and automatically. When designing assessments in PrairieLearn though, instructors must carefully consider how to fairly award partial credit, especially if a question has multiple parts. In our dialogue with other instructors who are or are considering using PrairieLearn, they often mention that designing multi-part questions is either impossible or overly difficult. When an instructor manually grades a paper-based exam, the instructor can rely on their professional judgment and adjust their grading rubric on the fly in response to unexpected student errors or creatively correct answers. Additionally, instructors can potentially grade students' scratch work rather than just the final answer submitted for grading, allowing the instructor to grade students' process in addition to their answers. In contrast, most automatic grading systems can award credit based only on students' final answer which is checked against a solution determined prior to seeing students' answers.

Rather than attempt to replicate the strengths of paper-based exams for grading, we focus instead on the strengths of a computer-based exam, namely the ability to immediately grade each student's exam and provide feedback. These mechanisms let us assist students' problem-solving process rather than grade it by helping students identify and correct their own mistakes during the exam. We adopted a grading policy that awarded students partial credit if their answer was within a tolerance factor of the correct solution. Students could continue to earn additional partial credit by re-submitting answers with the amount of partial credit decreasing after each subsequent submission.

In this paper, we describe how we created computer-based, machine-gradable, multi-part exam questions about caches in our computer architecture course. We then provide an evaluation of how this paradigm shift affected students' exam grades and present survey data describing students' reactions to this shift in exam modality. This project is also part of a larger effort to promote the adoption of PrairieLearn, more broadly in STEM education and was undertaken partly in response to many inquiries about how to successfully design multi-part engineering exam problems in PrairieLearn. We hope that our effort will provide an instructive example for instructors in STEM education who are considering using computer-based exams or alternate approaches to assess their students' learning.

II. Background

Modern computer architectures use a hierarchy of memories consisting of a large memory that accesses its data slowly and small memories that access their data quickly. This memory hierarchy must be invisible to the programmer so that the memory system appears be a single large memory that is also fast. In this hierarchy, caches are small, fast memories. Computer architects must design these caches to be able to quickly access frequently used data without knowing in advance what data will be used. Sahuquillo et al. [11] argued that students must learn three tasks to understand caches: mapping functions (mapping a memory address to a cache block), data reuse (spatial and temporal locality), and replacement algorithms (which element to replace when there is a cache miss). We present a brief overview of some relevant related work.

A. Difficulties in Learning Cache Memory

Cache memory is considered one of the most difficult topics in a computer architecture course. Porter et al. [12] tested students on commonly taught topics in a computer architecture course including caches. They found that only 40% of the students were able to correctly answer basic questions related to cache design after completing a full computer architecture course. Anecdotally, our students have also identified caches as one of the most difficult topics in the course. Caches require students to coordinate their knowledge about many concepts that are known to be important and difficult such as memory models and state [13], [14]. The concept of state has previously been described as a threshold concept - troublesome concept to learn that transforms how one understands a field [15], [16]. Students possess many misconceptions about state, struggling to distinguish the state of the system from the systems' inputs or outputs and how/when state is changed [17], [18]. These misconceptions directly lead to misconceptions about caches.

Grigoriadou et al. [19] interviewed 20 students to discover misconceptions related to cache design. They found that students have a tendency of building overly simple mental models of caches. These simpler models result in missing steps while performing cache operations such as not fully working out the new state of a cache after a hit (data requested by processor is in the cache) or a miss(data requested by processor is not in the cache). Working out the state of the cache also needs knowledge of how data is stored in memory and the order in which data is accessed. All these factors combine to make cache memory a topic which has a high cognitive load.

B. Cognitive Load and Sub-goal Labeling

Cognitive load is defined as the amount of information that needs to be retained in the working memory while working out a problem [20]. As described earlier, working with cache memory requires students to keep track of the state of the cache, state of the program, and memory contents. Breaking down a large task into smaller tasks, where each small task requires a small amount of information in the working memory, is an effective way of reducing cognitive load. The process of breaking down a large task into smaller tasks is called sub-goal labelling [21]. It is a commonly used technique in STEM education [21]–[24]. In computer science, sub-goal labeled examples have been explored for teaching computer programming [25], [26]. In our context, cache problems are divided into multiple parts with the general idea that students start solving the problem by looking at a few well-defined memory addresses and generalize the patterns as they move forward in the question.

C. Cache Simulators

Cache simulators are programs that take a cache memory configuration and a list of memory addresses as input and determine if each memory access is going to be a hit or a miss. Because predicting cache performance is such a cognitively demanding task, cache simulators can be used to help engineers empirically test what type of cache would perform best for an application. Cache simulators are also often used to help students in learning cache concepts. Multiple cache simulators are freely available with varying features. One of the earliest cache simulators for teaching purposes was Cachesim [27]. Sahuquillo et al. [11] extended SPIM (one of the most commonly used simulators for MIPS assembly language programming) to include cache simulation in it. SPIM is commonly used to teach the MIPS Instruction Set Architecture in computer architecture courses. In our implementation, we used a custom cache simulator which we also use in our lab assignments. We use the cache simulator in our project to facilitate the rapid development and grading of exam questions.

D. Course Context

Our computer architecture course is a required course for all computer science majors, enrolling between 300 and 400 students per term. The course is taught with a mixture of large, active-learning lectures and smaller discussion sections. The course teaches basic digital logic, computer organization, assembly programming, pipelining, caches, and introduces concepts of parallel computing. Our computer architecture course relies on PrairieLearn for delivering homework assignments and most exams. Students take exams in a secure, proctored computer-based testing facility [28]. The course has historically used paper-based exams to assess students' knowledge of caches because instructors wished to use multi-part questions. In Fall 2019, we switched to a computer-based, machine-gradable exam for caches in PrairieLearn.

III. Design Considerations

In this section, we describe our experience of designing the system and explain the design criteria we found to be helpful so that others may be able to extrapolate and replicate our efforts in other contexts.

On our cache exam question, we want to assess whether students can estimate the cache performance (e.g., number of hits/misses) of a piece of code running on a cache with given parameters and be able to understand how performance changes when small changes are made either to the code or the cache parameters. Traditionally, our paper-based exam usually had one, 5-part question to assess cache memory concepts.

1) Students must determine the first five or six memory accesses of the program and determine if they will be hits or misses. This part helps the students begin tracing the code and its basic behavior.
2) Students must determine the contents of the cache at a critical point in the program's execution to help them begin identifying larger patterns in how the code accesses data.
3) Students must determine the total number of memory accesses to demonstrate that they understand the overarching data access patterns of the code.
4) Students must determine the total number of cache misses to demonstrate that they understand how the memory access patterns interacted with the cache.
5) Students are shown a slight alteration to their original problem statement and asked to reason about how that alteration would affect their calculations in parts 3 and 4.

In a paper-based exam all students received the same question with same parameters because all students took the same exam at the same time in a proctored setting.

Because our computer architecture course has approximately 300 registered students every semester, there is no computer lab large enough for all students to take the same exam at the same time. Given these constraints, we deliver computerized exams asynchronously in a secure, proctored computer-based testing facility (CBTF) that allows students to take an exam anytime during a specified exam window. Due to the asynchronous nature of the exam, all students cannot be given the same question for exam security and integrity reasons. Instead, students are given randomized versions of the same question or questions that have the same learning objectives, so that the exact numerical answers will be different for each student but the general process for deriving an answer is the same for all students. These constraints required that we design our exam questions so that new variants could be developed rapidly or automatically with minimal additional effort from the instructor. Historically, developing even one variant of the question might take the instructor several hours to author, refine, and provide a complete solution.

When designing a parameterized question to be randomized, the instructor must be aware that some combinations of parameters may make a question significantly easier or harder. By way of analogy, suppose an instructor wished to test whether students could determine the length of the third side of a triangle given two other sides and an angle. If some students were given 3:4:5, right-angle triangle or equilateral triangles which have well known properties for this type of task while other students were not given these special classes of triangles, the exam question would be unfair. The instructor would need to make sure that all students were given the same general class of triangles or only the same special cases. Therefore, the instructor should randomly generate only a small number of parameters, within reasonable numbers (e.g., randomly determine one side of a 3:4:5 triangle) and determine the rest of the parameters based on these parameters to ensure that all variants have the same basic properties. For our cache problems, the instructor needed to make many targeted decisions of this type such as ensuring that every variant should result in frequent cache collisions or whether the data from the program should be considerably larger or smaller than the total cache size. These decisions make it a little longer for the instructor to write the initial question, but this time is more than recouped by the time saved from grading.

We also had to decide how to award partial credit. In the paper-based exam, students were awarded partial credit if their scratch work identified key conceptual parts of the questions such as identifying the number of loop iterations, identification of cache conflicts, and recognizing common access patterns such as row or column traversals. This information is available in a paper-based exam from the students' scratch work but is not easily graded in a computer-based exam where students only enter their final answer. However, because we can give students immediate feedback and students can submit an answer multiple times, we designed a partial credit system that focused on giving students partial credit for eventually figuring out small parts of the exam question. A primary benefit of this partial credit design is that students have a clear, natural incentive for reviewing their solutions and making a best effort attempt to eventually find the correct solution, potentially maximizing their learning.

Finally, we needed a way to quickly generate solutions based on the randomly generated parameters. Thus, a cache simulator was essential for this last step. We chose to run the cache simulator during the question generation rather than the grading phase, so that the simulator would need to run only once to get the correct answers.

Based on our experience, we recommended the following design constraints for creating multi-part exam questions, specifically for caches, but potentially also for other engineering systems:

- Instructor must have a way to control the parameter generation for the question.
- The machine grader should be able to keep track of multiple student submissions to award partial credit based on how students respond to feedback.
- Instructor should be able to easily reuse and maintain parts of the question.
- The system (i.e., cache) simulator should be available to the instructor through a simple interface.
- The system (i.e., cache) simulator should be efficient enough to be able to complete running during question generation phase.

IV. System Design and Implementation

In this section, we describe our specific implementation of the design criteria. Figure 1 provides the high-level system diagram. Major sections of the system diagram are described below.

A. *PrairieLearn*

PrairieLearn is Python based and has been designed to primarily support question generators written in Python, therefore most of our development also uses Python. The exception here is the cache simulator which is written in C++. Some of the requirements are satisfied by PrairieLearn out of the box. PrairieLearn provides the question frontend, which is a web interface that displays the question to the student and receives students' answers. PrairieLearn also natively provides a mechanism to gradually reduce the credit a student receives when using multiple submissions to solve a problem. For example, an instructor can set a question to award 100% credit for a first submission and then award 80% of the remaining possible credit for a second submission. If a student earned 50% credit on their first submission and 90% credit on their second submission, their final grade for the question would be $(100*0.5+80*(0.9-0.5) = 82\%)$. We awarded 100%, 90%, 75%, 50%, 30%, 10% credit for each subsequent submission.

When creating an exam, PrairieLearn calls the `generate` function written by the instructor to create a new variant of the question. The `generate` function selects random cache parameters and random variants of C code segment, where variants differed on the total size of a data structure being operated on and the data type of the data structure (i.e., int vs. doubles).

The `generate` function then calls the Cache Analyzer class within the Cache Analyzer Module with cache parameters and C code for analysis. Once analysis is complete, the `generate` function can call various getter functions in the Cache Analyzer class to get information such as list of accessed addresses and whether those accesses were hits or misses or get information such as the total number of accesses, hits, and misses. The former getter functions provide solutions for parts 1-2 and the latter functions provide solutions for parts 3-5 as described is Section III. The `generate` function stores the analysis results to be used for grading. Since an instructor will want to re-use this cache analyzer in the generation of more problems, the Cache Analyzer Module provides a simple interface - only one class (Cache Analyzer) is exposed.

After completing problem generation, PrairieLearn serves the question to the student. The students' answer is sent to a separate Python function, `grade`, on submission. If the instructor does not implement a `grade` function, then PrairieLearn uses its default grading. We implemented our own `grade` function to award points for answers within different levels of tolerance of the correct solution with gradual decrease in credit according to the following rules.

1) Answer within 2% of solution: score = 100%

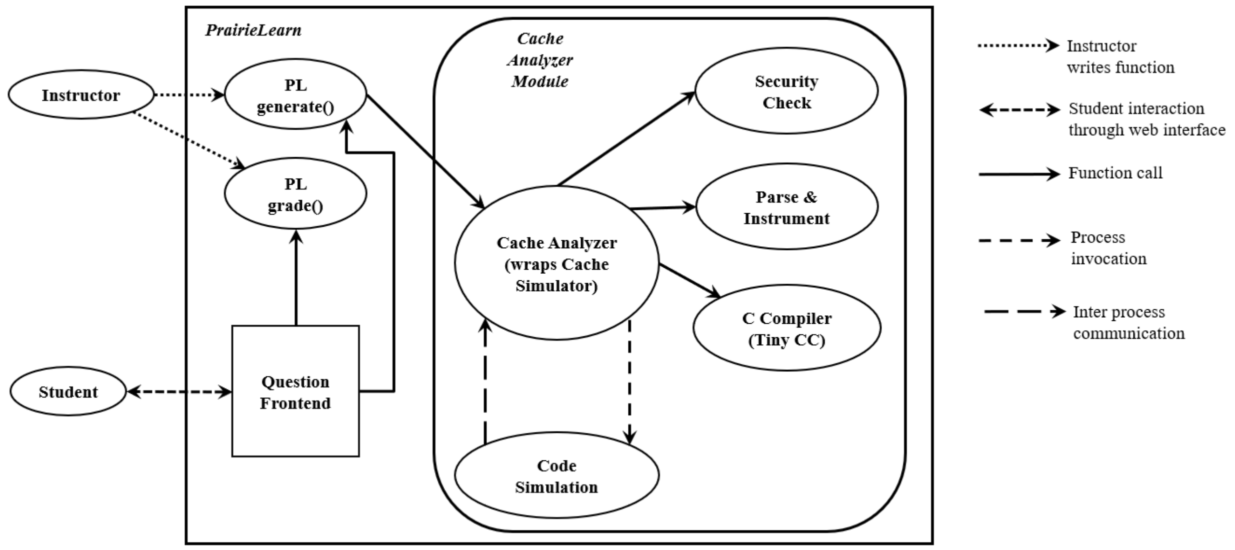

Fig. 1. System Diagram

2) Answer within 5% of solution: score = 98%
3) Answer within 10% of solution: score = 90%
4) Answer within 20% of solution: score = 66.7%
5) Answer within 30% of solution: score = 33.3%

The `grade` function determines the score for an answer submission. Submission score from the `grade` function is combined with PrairieLearn's default scoring system for multiple submissions described earlier in this section to get the final score for the student.

B. Cache Analyzer Module

Cache Analyzer Module includes the Cache Analyzer class and functions for Security Check, Parsing and Instrumentation, a C Compiler and callbacks for Code Simulation.

1) Cache Analyzer class: Our cache simulator is written in C++. In order to use this in our Python code, it is wrapped inside a Python class called Cache Analyzer. The Cache Analyzer class also manages all external communication for the Cache Analyzer Module. The class receives the cache parameters and code for simulation from the `generate` function. It initializes the cache simulator and uses the other components available in the module to get memory addresses for cache simulator which can then determine if each access is a hit or a miss. Finally, the Cache Analyzer class provides functions to query the simulator for number of accesses, number of hits, number of misses and for a list of addresses with their hit/miss status.

2) Security Check: The security checker is optional and serves as a technique with potential to prevent any untested C code from crashing the system unexpectedly. It injects the code to catch common exceptions and ensure the code returns without errors (using $longjmp$ etc.) It also checks for system calls, illegal inclusion of header files, and prohibited function calls. In addition, it can compile and run the code and try to detect compile and run-time errors, as well as dead loops, in advance. For the use case described in this paper, security check is disabled because the only code that is simulated is written by the instructor. But it is a useful feature if we decide to simulate student supplied code in future.

3) Parse and Instrument: After security check, the code is passed to a custom parser. The parser utilizes regular expressions and simple recursive parse trees to analyze C control structures, e.g. IF/ELSE, FOR loop, and simple expressions. We use the parser to identify the use of data structures that are stored in memory. All accesses to these data structures are instrumented to record the accessed addresses so our simulator can have the memory addresses it needs for simulation. In particular a read of the form $a[i]$ becomes $(access((unsigned)\&a[i] - (unsigned)a, id_a), a[i])$, where $access(accessed\_address, internal\_variable\_id)$ is the interface provided by Cache Analyzer class to record an access. Notice that we are reporting difference between the accessed address and the base address of the data structure and an internal tracking ID for the variable. We use the difference and the ID to realign the address to the instructor supplied base address for the data structure. This is an important step to ensure that the simulation works according to instructor's specification. Similarly, a write access of the form of $a[i] = exp$ is expanded to $(a[i] = exp, access((unsigned)\&a[i] - (unsigned)a, id_a), a[i])$. In order to run the instrumented code, we embed it into a main function to make it a complete program that can be executed.

4) C Compiler: Once we have a complete C program, the next step is to compile and run it. In order to compile and run the code as quickly as possible, we decided to use TinyCC, an open-source C compiler [29]. TinyCC provides a *libtcc* library which has good compatibility with C standards, fast compilation speed, and ability to directly run the compiled program from memory. We utilized a wrapper Python class to embed the C library into our Python codebase. Compiling the generated program produces an in-memory executable which can be run to generate a stream of accessed memory addresses.

5) Code Simulation: The Cache Analyzer modules starts the executable generated by the compiler as a separate process. This process reports memory accesses to the Cache Analyser module through inter process procedure calls. Cache Analyzer passes the addresses it receives through these calls to the simulator to determine if they are hits or misses. When the code running process dies, the simulation is complete and data is available for query.

C. Limitations of the Cache Simulation

We run the simulation at the time of question generation. The `generate` function is called with a timeout of 20s. We have observed that during this time our cache simulator can process approximately 2 million accesses. This puts a limit on the size of the problem that we can generate. In our work on question development, we found that we can still capture the essence of the question if we reduced both the size of the data and cache accordingly. In our question development we tried to keep the total number of accesses below 200,000 to avoid over-burdening the system.

V. Question Development

Because of the asynchronous nature of the machine administered exam, we needed to develop multiple exam questions of roughly equal difficulty. We started this process by identifying core learning objectives and articulating features of the code and cache that would facilitate exploring those learning objectives. For example, the code should have copious re-use of the same pieces of data many times and the total amount of data in the program should be bigger than the total size of the cache, forcing cache conflicts. With our learning objectives as a guide, we used questions like the following to define the parameter spaces for each question variant.

- What is the range of data size to cache size ratio that consistently results in the behavior that we want to test?
- What range of values other parameters like associativity and block size can take without fundamentally changing the problem?
- What should be the relative base addresses for the different data structures?
- What parameter change would result in an interesting variation in the problem for last part?
- What is the maximum size of the data that can result in approximately 200000 accesses?
- What is the minimum size of the data that will keep the problem non-trivial?

After an initial brainstorming session with our entire course staff, we generated a pool of 24 code segments that met these criteria and converted 20 of these into candidate exam problems. We created a template for the 5-parts of our analysis problem (See Section III. With this template, we were able to rapidly generate new exam questions, primarily by just copy and pasting code fragments into the template and then running a few test cases to catch errors and debug.

One aspect of our design that greatly facilitated this design process was that we were able to almost completely abstract away the simulator during our question design process - we only had to worry about time-out issues for our simulator. Consequently, the primary difference between creating a paper-based and computer-based exam once we had a framework was that the instructor must determine a range of interesting values for all parameters of the question and code them in the generate function instead of picking one value in the range for the paper-based exam. Identifying parameter spaces was the most time-consuming process of developing the new questions, taking 3 instructors a month to develop 20 questions. Since one exam question would take the same number of instructors two weeks to grade, this trade-off was well worth it.

VI. Evaluation

To evaluate our transition to a computer-based exam, we primarily wanted to determine 1) whether this transition did not harm students' general performance on the exam (i.e., the grade distribution stayed about the same) and 2) that students felt that the computer-based exam was fair.

To evaluate the first concern, we had administered the same cache analysis question in both the Fall 18 (FA18) and Spring 19 (SP19) terms and converted that same question into a computer-based exam question in Fall 19 (FA19). We did not return physical copies of students' exams in FA18 and SP19 to maintain some degree of exam security. The FA18 and SP19 variants were identical except that they used different randomly generated exam parameters (i.e., the same C code was given for students to analyze but variable names were changed and the total size of the cache and other cache parameters were changed such that the mathematical analysis remains same). 63 students in FA19 variants were given exactly the same problem as FA18 and SP19 students, with only variations in the randomly generated parameters. The remaining students were given variants that tested the same learning objectives but also had variations in the C code that students were given to analyze. We statistically compare students' performance on just the exam questions that were kept the same separately and then with all exam variants included. In all semesters, we offered first- and second-chance exams [30] so that students would have an opportunity to improve their mastery of content. We used variants of the same second-chance exam across all three terms.

To evaluate the second concern, we surveyed students about the switch to the computer-based exam from paper-based during our end-of-semester survey. FA19 students were aware of the structure and nature of the paper-based exam because they were given old paper-based exams to study from (the same exams FA18 and SP19 students were given) and review sessions were conducted using the paper-based exams.

A. Analysis of student performance

For the FA19 computer-based exam, we allowed students 6 submissions with decreasing weights as described in Section IV-A. The average number of submissions for the questions was 4.60 for the first-chance exam and 4.84 for second chance exam indicating that students used multiple submissions frequently.

During our analysis of the first-chance exam, we discovered one error in our grading procedures. When assessing whether students could determine if specific data elements were in the cache at a given execution point (part 2), we asked students to check boxes for each element of a data structure that was in the cache. Unfortunately, our grading process automatically revealed the correct answers for these checkboxes by marking each check box right or wrong individually. Consequently, students could immediately deduce the correct answer and get full points on that part of the question on later submissions without working through the problem. We fixed this flaw before second-chance testing. Table I shows that students took advantage of this bug. On the first-chance exam, the percentage of correct answers rose from just to 25.9% to 72.9%, a much greater rise than in the second-chance when the percentage of correct answers only rose from 46.2% to 57.5%. To minimize the effect of this bug, we consider only the first submission for Part 2, which constituted 30% of the question grade and consider the submissions with normal decreasing weight for the remaining 70% of the question points. We also compare just the first submission of students' grades separately from students' scores after the final submission. We do not compare second-chance exam scores, because any differences in student performance on the exam may alter which populations of students opted to take the second-chance exam.

TABLE I
FULL GRADE PERCENTAGE IN PART 2

Chance	**Sub 1**	**Sub2**	**Sub3**	**Sub4**	**Sub5**	**Sub6**
1st	25.9	72.9	89.1	91.8	92.6	93.5
2nd	46.2	57.5	65.1	68.9	70.8	72.6

TABLE II
FA19 (FIRST CHANCE) WEIGHTED SCORE BY SUBMISSION

Measure	**Sub 1**	**Sub2**	**Sub3**	**Sub4**	**Sub5**	**Sub6**
samples	340	340	340	340	340	340
mean	53.62	59.23	61.15	61.93	62.15	62.18
sd	24.08	23.42	22.8	22.56	22.47	24.46

Table II shows how students' exam averages evolved with each submission. A one-way ANOVA revealed that students' scores increased statistically significantly $[F(5, 2034) = 7.17, p < 0.001]$ with Tukey post-hoc tests revealing that the first submission was statistically significantly lower than all other submissions (all $p < 0.05$) but that all other submissions revealed no statistically significant difference. In other words, students' scores improved from their first to their second submission, but additional gains were marginal after the first submission. Based on this analysis, we compare students' first and last submissions from FA19 with students' exam scores from FA18 and SP19.

Table III shows the means and standard deviations for only those exams that had the same C code across semesters. A

TABLE III
COMPARISON OF FA18, SP19, AND FA19 FIRST SUBMISSION AND FA19 SIXTH SUBMISSION FOR FIRST-CHANCE CACHE EXAM, ONLY EXACT SAME C CODE VARIANT

Measure	FA18	SP19	FA19 **Sub1**	FA19 **Sub6**
samples	310	264	63	63
mean	54.48	54.34	55.30	61.68
sd	23.5	25.12	17.83	16.57

one-way ANOVA revealed no statistically significant differences between students' scores across terms and number of submissions $[F(3, 696) = 1.86, p = 0.13]$

TABLE IV
COMPARISON OF FA18, SP19, AND FA19 FIRST SUBMISSION AND FA19 SIXTH SUBMISSION FOR FIRST-CHANCE CACHE EXAM, ALL VARIANTS

Measure	FA18	SP19	FA19 **Sub1**	FA19 **Sub6**
samples	310	264	340	340
mean	54.48	54.34	53.62	62.18
sd	23.5	25.12	24.08	24.46

Table IV shows the same data except it includes all exam variants. A one-way ANOVA revealed a statistically significant difference $[F(3, 1250) = 9.61, p < 0.001]$ with Tukey post-hoc testing revealing a statistically significant difference between FA19 last submissions and the FA19 first submission and the other terms (all $p < 0.05$). There were no significant differences in student performance on the first submission in FA19 and student performance during the FA18 and SP19 semesters ($p = 0.97$ and $p = 0.98$ respectively). This analysis suggests that just providing multiple tolerance levels provided an equivalent amount of partial credit for student solutions as grading students' processes did for written exams. Providing multiple submissions with decreasing credit provided additional partial credit beyond what was available to students on the paper-based exam.

B. *Analysis of student perceptions*

On our end-of-semester FA19 survey, we asked students five, 5-point Likert scale items about their perceptions of the relative fairness of the computer-based versus paper-based exams. This data is limited by the fact that the FA19 students did not actually take the paper-based exam and only used it for studying. Results from this survey are shown in Figure 2. This data shows that students overwhelmingly preferred the computer-based version of the exam.

VII. DISCUSSION

The move to a computer-based, multi-part exam was generally positive for both students and instructors. Our data indicates that students performed comparably (in terms of points earned) on both the paper-based and the first submission of computer-based versions of the exam. Additionally, students expressed a general preference for the computer-based exam to the paper-based exam. Anecdotally, our experience as instructors was also positive.

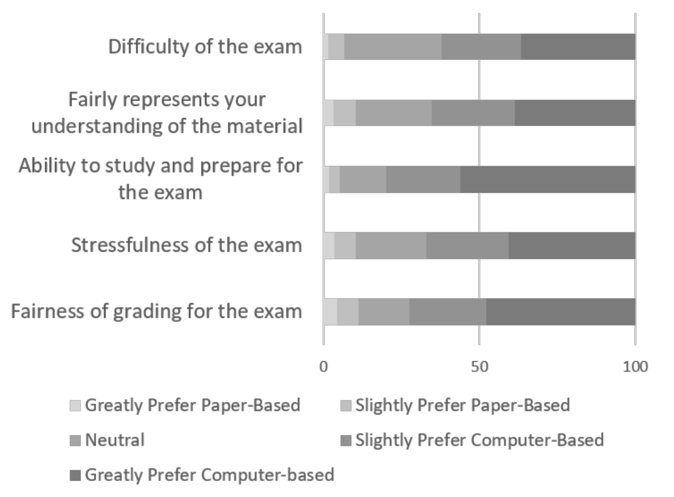

Fig. 2. Student preferences for paper-based vs. computer-based exams. More than 50% of students preferred computer-based exams for all questions.

By taking advantage of the computer-based environment, we were able to allow multiple submissions with limited feedback. Our data shows that students performed better with multiple submissions, correcting mistakes and ultimately earning more points than their peers who took the paper-based exam. These findings suggest a number of potential benefits to using computer-based testing in this manner. First, students rather than instructors are responsible for finding and correcting mistakes providing students with more opportunities to learn. Second, by giving students a way to find and redress their mistakes, we as instructors received far fewer complaints and regrade requests from students. With our paper-based exams, upwards of 10-20% of students (30-60 students in our class) would petition for regrade and more partial credit, sometimes for legitimate oversights or mistakes on the part of the instructors. With our computer-based exams, there were only one or two complaints. Third, we allowed students to have a total of 6 submissions. Our data shows that the score increase difference between the first two submissions was statistically significant but there was no significant increase after the second submission. This indicates that providing even one extra submission may be enough to maximally benefit students. Conversely, giving students unlimited submissions in an exam context seems unlikely to create issues of grade inflation beyond allowing a second submission. Instructors could possibly create a less stressful exam environment by encouraging students to keep trying and learning until they succeed during an exam. This modality for testing could possibly help foster growth mindsets for students. Future research could further explore the effects of providing fewer or unlimited submissions during a computer-based exam.

Our findings were consistent regardless of whether students were given an exam with exactly the same question as the paper-based exam or variants that tested the same learning objectives but with different code fragments. The consistency of students' scores across variants suggests that the added care needed to carefully identify the learning objectives made for a more reliable and fair exam design process.

Our survey results indicate that students preferred the computer-based exam. One possible reason for this preference may be that students are used to taking exams in the computer based testing facility as they take rest of their exams for this course there. We do note that the students believed most strongly that the move to computer-based exams helped them study better and prepare for the exam and the grading of the exam was fairer. We believe that the students had this perception because we published pre-determined tolerance levels for awarding partial credit that removed any ambiguity for how partial credit worked. Clear grading standards are generally considered a best practice and are a primary argument for using rubrics and criterion-referenced grading [31]. It's possible that awarding partial credit based on tolerances could be a best practice for paper-based exams as well. However, students may be more willing to accept tolerance-level-based grading in a computer-based context, where students may reasonably expect that a computer cannot try to understand the quality of their reasoning, whereas tolerance-level-based grading may be rejected for paper-based exams because there is an expected social contract that the instructor will try to find all the ways that a student understood the material. Future research could explore the effect of tolerance-level-based grading of quantitative engineering problems.

VIII. Conclusion

The initial development of the framework and exam questions was time consuming task, but it continues to save significant time for grading and has accelerated our ability to write new exam questions for the system because we no longer need to write solution guides and rubrics. We now have a pool of 20 questions that we can reuse in future runs of the course. By having the simulation-based autograder, we are also now able to develop practice problems for homework assignments more easily, providing students with more learning resources. As instructors, we believe that we already have had a phenomenal return on our investment in building the framework.

We believe that the simulator-based mechanism for developing questions described in this paper can be quite general. As engineers and scientists, we often rely on simulators to help us reason about complex systems, these same simulators can be deployed effectively in our classrooms. Notably, we used our simulator to test students' analysis skills, but the same system could also potentially be used to test students' design skills. For example, we could ask students to choose parameters that would optimize a system and then run the simulator to validate and grade students' responses. We plan to try exactly this approach in the next iteration of our class. We hope that our reflections and the design principles we shared in Section III will help other instructors build their simulation-based testing environments for complex engineering systems.

References

[1] F. A. Phang, A. N. Anuar, A. A. Aziz, K. Mohd Yusof, S. A. H. Syed Hassan, and Y. Ahmad, "Perception of complex engineering problem solving among engineering educators," in *Engineering Education for a Smart Society*, M. E. Auer and K.-S. Kim, Eds. Cham: Springer International Publishing, 2018, pp. 215–224.

[2] J. Funke and P. Frensch, *Complex problem solving: the European perspective—10 years after*. New York: Lawrence Erlbaum, 2007, p. 25–47.

[3] N. A. of Engineering and N. A. of Engineering, *The Engineer of 2020: Visions of Engineering in the New Century*. Washington, DC: The National Academies Press, 2004. [Online]. Available: https://www.nap.edu/catalog/10999/the-engineer-of-2020-visions-of-engineering-in-the-new

[4] D. Jonassen, J. Strobel, and C. Lee, "Everyday problem solving in engineering: lessons for engineering educators," *Journal of Engineering Education*, vol. 95, no. 2, p. 139–151, 2006.

[5] C. L. Dym, "Engineering design: So much to learn," *International Journal of Engineering Education*, vol. 22, no. 3, pp. 422–428, 2006.

[6] D. Fordyce, "The development of systems thinking in engineering education: an interdisciplinary model," *European Journal of Engineering Education*, vol. 13, no. 3, pp. 283–292, 1988. [Online]. Available: https://doi.org/10.1080/03043798808939427

[7] M. F. Ercan and J. Caplin, "Enabling systems thinking for engineering students," in *2017 IEEE 6th International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, 2017, pp. 1–5.

[8] D. Newman, *Civil Engineering: Problems & Solutions*, 16th ed. Kaplan AEC Education, 2005.

[9] M. West, G. Herman, and C. Zilles, "Prairielearn: Mastery-based online problem solving with adaptive scoring and recommendations driven by machine learning," *age*, vol. 26, p. 1, 2015.

[10] PrairieLearn, "Prairielearn/prairielearn," Apr 2020. [Online]. Available: https://github.com/PrairieLearn/PrairieLearn

[11] J. Sahuquillo, N. Tomas, S. Petit, and A. Pont, "Spim-Cache: A Pedagogical Tool for Teaching Cache Memories Through Code-Based Exercises," *IEEE Transactions on Education*, vol. 50, no. 3, pp. 244–250, aug 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4287124/

[12] L. Porter, S. Garcia, H.-W. Tseng, and D. Zingaro, "Evaluating student understanding of core concepts in computer architecture," in *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. ACM, 2013, pp. 279–284.

[13] K. Goldman, P. Gross, C. Heeren, G. Herman, L. Kaczmarczyk, M. C. Loui, and C. Zilles, "Identifying important and difficult concepts in introductory computing courses using a delphi process," *SIGCSE Bull.*, vol. 40, no. 1, p. 256–260, Mar. 2008. [Online]. Available: https://doi.org/10.1145/1352322.1352226

[14] L. C. Kaczmarczyk, E. R. Petrick, J. P. East, and G. L. Herman, "Identifying student misconceptions of programming," in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 107–111. [Online]. Available: https://doi.org/10.1145/1734263.1734299

[15] J. Boustedt, A. Eckerdal, R. McCartney, J. E. Moström, M. Ratcliffe, K. Sanders, and C. Zander, "Threshold concepts in computer science: Do they exist and are they useful?" *SIGCSE Bull.*, vol. 39, no. 1, p. 504–508, Mar. 2007. [Online]. Available: https://doi.org/10.1145/1227504.1227482

[16] D. Shinners-Kennedy, *The Everydayness of Threshold Concepts*. Leiden, The Netherlands: Brill — Sense, 2008, pp. 119 – 128. [Online]. Available: https://brill.com/view/book/edcoll/9789460911477/BP000010.xml

[17] G. L. Herman, C. Zilles, and M. C. Loui, "Flip-flops in students' conceptions of state," *IEEE Transactions on Education*, 2012.

[18] G. L. Herman and D. S. Choi, "The affordances and constraints of diagrams on students' reasoning about state machines," in *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ser. ICER '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 173–181. [Online]. Available: https://doi.org/10.1145/3105726.3106172

[19] M. Grigoriadou, E. Kanidis, and A. Gogoulou, "A Web-Based Educational Environment for Teaching the Computer Cache Memory," *IEEE Transactions on Education*, vol. 49, no. 1, pp. 147–156, feb 2006. [Online]. Available: http://ieeexplore.ieee.org/document/1593884/

[20] J. Sweller, "Cognitive load during problem solving: Effects on learning," *Cognitive science*, vol. 12, no. 2, pp. 257–285, 1988.

[21] R. Catrambone, "The subgoal learning model: Creating better examples so that students can solve novel problems." *Journal of experimental psychology: General*, vol. 127, no. 4, p. 355, 1998.

[22] R. K. Atkinson, R. Catrambone, and M. M. Merrill, "Aiding transfer in statistics: Examining the use of conceptually oriented equations and elaborations during subgoal learning." *Journal of Educational Psychology*, vol. 95, no. 4, p. 762, 2003.

[23] L. E. Margulieux, "Subgoal labeled instructional text and worked examples in stem education," Ph.D. dissertation, Georgia Institute of Technology, 2014.

[24] P. Ayres and J. Sweller, "Locus of difficulty in multistage mathematics problems," *The American Journal of Psychology*, pp. 167–193, 1990.

[25] B. B. Morrison, L. E. Margulieux, and M. Guzdial, "Subgoals, context, and worked examples in learning computing problem solving," in *Proceedings of the eleventh annual international conference on international computing education research*, 2015, pp. 21–29.

[26] L. E. Margulieux, M. Guzdial, and R. Catrambone, "Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications," in *Proceedings of the ninth annual international conference on International computing education research*, 2012, pp. 71–78.

[27] C. A. Prete, "Cachesim: A graphical software environment to support the teaching of computer systems with cache memories," in *Conference on Software Engineering Education*. Springer, 1994, pp. 317–327.

[28] C. Zilles, M. West, D. Mussulman, and T. Bretl, "Making testing less trying: Lessons learned from operating a computer-based testing facility," in *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2018, pp. 1–9.

[29] TinyCC, "Tinycc/tinycc," Aug 2018. [Online]. Available: https://github.com/TinyCC/tinycc

[30] G. Herman, K. Varghese, and C. Zilles, "Second-chance testing course policies and student behavior," in *2019 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2019, pp. 1–7.

[31] K. J. Klauer, "On criterion-referenced grading models," *Journal of Educational Statistics*, vol. 9, no. 3, pp. 237–251, 1984. [Online]. Available: http://www.jstor.org/stable/1165009