Executable Exams: Taxonomy, Implementation and Prospects

Chris Bourke
chris.bourke@unl.edu
University of Nebraska–Lincoln
Lincoln, NE, USA

Yael Erez
yaelerez@campus.technion.ac.il
Technion Israel Institute of Technology
Haifa, Israel

Orit Hazzan
oritha@ed.technion.ac.il
Technion Israel Institute of Technology
Haifa, Israel

ABSTRACT

Traditionally exams in introductory programming courses have tended to be multiple choice, or "paper-based" coding exams in which students hand write code. This does not reflect how students typically write and are assessed on programming assignments in which they write code on a computer and are able to validate and assess their code using an auto-grading system.

Executable exams are exams in which students are given programming problems, write code using a computer within a development environment and submissions are digitally validated or executed. This format is far more consistent with how students engage in programming assignments.

This paper explores the executable exam format and attempts to gauge the state-of-the-practice and how prevalent it is. First, we formulate a taxonomy of characteristics of executable exams, identifying common aspects and various levels of flexibility. We then give two case studies: one in which executable exams have been utilized for nearly 10 years and another in which they've been recently adopted. Finally, we provide results from faculty surveys providing evidence that, though not standard practice, the use of executable exams is not uncommon and appears to be on the rise.

CCS CONCEPTS

• **Applied computing → Computer-managed instruction**.

KEYWORDS

introductory computer science, executable exams, assessment, taxonomy, autograder, paper-based exams

ACM Reference Format:
Chris Bourke, Yael Erez, and Orit Hazzan. 2023. Executable Exams: Taxonomy, Implementation and Prospects. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023), March 15–18, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3545945.3569724

1 INTRODUCTION

Use of automated programs and tools for assessing programming assignments in introductory programming courses is well-established. Instructors have been designing, implementing, and using tools since at least the early 1960s. Douce et al. [6] provides a good literature review and history of automated program grading. They identify three distinct generations of development: the first generation of single-use, ad-hoc methods, a second generation of multipurpose tools and frameworks, and a third generation of web-based tools.

It could be said that we are currently in the fourth generation of development, characterized by the wide-spread adoption of browser-based development environments and the explosion of commercially available autograder solutions [9, 15, 20]. While automated grading has long-been ubiquitous in computer science classes, the recent growth in enrollments of non-majors, and the development of programming courses by non-CS departments has created a huge demand for automated grading and assessment.

The literature and research involving autograder tools, methodology and best practices is well-documented. However, the context for auto-grading has focused on programming assignments. It is not readily apparent how common these tools are used in more summative assessments such as exams, quizzes, etc. Traditionally, exams have been "paper-based" in which students may write programs or code snippets using pencil and paper and submit "hard copies." In contrast, an *executable exam* is an assessment where a student writes code using a computer within an environment in which their submission is digitally validated or executed in some manner. Generally, students produce "soft copies" (code, data, or other electronic artifacts), submit them through a digital submission system and (may) receive some form of feedback using an auto-grading system.

In this paper we attempt to establish a line of investigation for how automated grading tools can and are being used in exams. We first give some background suggesting that the use of "executable exams" is on the rise. We then present two case studies of the use of executable exams in large Computer Science I courses. The first is a course in which executable exams have been used for nearly 10 years while the second focuses on a more recent adoption of executable exams. We then present results from surveys of faculty on attitudes toward these types of exams. As challenges in higher education are rapidly changing how we deliver our courses, instructors will naturally seek out new options and tools to address them. We hope this paper provides some of those options.

2 BACKGROUND

Douce et al. [6] provide a good background on the history and development of auto-grading systems. More recently, autograder systems have been used with respect to exams in various ways. Lappalainen et al. [14] report improved performance when students started with a paper-based exam and then were allowed to use an executable environment. Piech & Gregg [17] developed an application for the administration of exams, providing students with basic

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGCSE '23, March 15–18, 2023, Toronto, ON, Canada.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9431-4/23/03…$15.00
https://doi.org/10.1145/3545945.3569724

syntax highlighting but without the ability to compile or receive feedback on correctness. Corley et al. [5] conducted a year-long study on the impact of traditional paper-based exams and an exam that allowed students to write submissions using a full IDE and found no statistically significant difference in overall student outcomes. However, similar to Piech & Gregg, students were not given any feedback on the correctness of their programs. Kurniawan et al. [13] document their experiences using a fully executable exam environment in conjunction with browser lock-down software to address academic integrity concerns.

Numerous commercial solutions have emerged in recent years giving students the ability to submit programs electronically and for instructors to automate grading. The explosion in commercial solutions in recent years suggests that many more instructors are utilizing these tools in general and perhaps specifically with respect to exams. One such company, zyBooks has reported a substantial increase in the number of instructors utilizing their autograding and executable code framework for exams with nearly 148 courses (books) with enrollment of over 16,000 students utilizing executable exams in some form in 2021 [9] with observed growth over the three years prior.

3 TAXONOMY OF CHARACTERISTICS

For the purposes of this investigation we define an *executable exam* as an exam in which a student writes code using a computer within an environment in which their submission is digitally validated or executed in some manner. This is in contrast to "traditional" exams that may include multiple choice questions (which may or may not be digitally graded) or paper-based exams in which students handwrite code in a format that is not easily validated or executable.

The large variety of introductory computing courses (language choice, focus, methodology, etc.) necessarily means that there is a similar variety in how executable exams are designed and administered. For this reason, we have attempted to identify the major characteristics of executable exams and to formulate a preliminary taxonomy of the various options that instructors may choose for each. We have described each option below in order of what we believe to be the most restrictive to most flexible. By no means is this an exhaustive list and even within each category, instructors may take a hybrid approach, choosing some aspects on different exams or among problems on an exam. Many of these characteristics are related to each other and a choice of one may necessarily restrict choices in another.

- *Time/Synchronicity* - How much time are students allowed to complete the exam and how flexible is this time constraint?
  - Synchronous: whether in-person or remote, students are required to take the exam at a specific start and end time.
  - Windowed: students are given a larger window in which they may choose an exam start time, but once started, they only have a limited time to complete it.
  - Asynchronous: the exam is designed to be completed within a short time (ex: 1–2 hours) but students are given a much larger window (ex: 24 hours) to complete it and may take the entire time if they wish. This is analogous to "take home" exams.
- *Autograder Feedback* - During the exam students may be given a variety of feedback from the environment or autograder.
  - No feedback – the environment does not give any feedback to the student; it only exists to give them a platform to easily submit code digitally.
  - Syntactical feedback – if appropriate, the environment may give feedback on whether or not the code compiles or is syntactically correct.
  - Correctness feedback – the environment may give the student feedback as to the correctness or soundness of their code. Even this could take a variety of forms. Students may be given (coded) formal test cases which provide not only correctness feedback, but also a resource by which to debug and troubleshoot. The autograder system may serve either as a blackbox (simply reporting the number of test cases passed/failed) or some (partial) feedback as to *how* (providing expected output and comparing it with their actual output) or if sophisticated enough, *why* their code is wrong.
- *(Re)submissions* - Limits may be placed on how many times students may submit solutions.
  - One shot - students may be restricted to a single submission rendering any feedback mechanism irrelevant (save for determining a final grade).
  - Limited resubmissions - students may be allowed to resubmit after being provided some feedback by the autograder system but may be limited to a small number (ex: 2–3) of resubmissions.
  - Judicial resubmissions - students are allowed to resubmit but it may come at a cost or other restriction. For example, each resubmission may deduct a certain number of points from the final grade.
  - Unlimited resubmissions - students are allowed any number of resubmissions within the time constraint of the exam.
- *Allowed Resources* - Since students utilize a computer, executable exams present a unique opportunity (and challenge) to the resources that students are allowed to have. In general, we assume that exams are closed with respect to human collaboration.
  - Closed exam: no resources other than the feedback mechanisms provided by the environment or autograder are allowed.
  - Physical resources: this could include hand written or printed notes, code samples, textbook, etc.
  - Digital resources: this may include code/language documentation, a digital textbook, typed notes or code samples. This may include a restricted or "whitelisted" web resources.
  - Unrestricted: students are allowed unrestricted web access.
- *Proctoring* - A variety of proctoring mechanisms may be used to mitigate the risks of cheating.
  - Human proctors: students take the exam while an individual or team of individuals monitors them. This may include taking the exam in a dedicated testing center or with instructional staff (TAs) or it may even mean taking the exam remotely with human monitors watching via a webcam.
  - Virtual proctors: a variety of software has been developed for automated proctoring. This can include "lock down" browsers to restrict resources that students can access or even more invasive facial recognition and monitoring software.
  - Detection software: during or after an exam, other less invasive software can be used to detect plagiarism. This may include simply examining submission logs (to ensure submissions are

made from local IPs) or more sophisticated software such as MOSS [19].
  - Honor system - Students may be trusted to act in accordance with an honor system and to formally or informally attest that they have not violated academic integrity.
- *Delivery - Hardware* - Practical concerns as well as concerns over equity may affect how an executable exam is delivered. It may not be the case that all students have access to personal laptops or instructors may wish to limit devices students are allowed to use during an executable exam.
  - Required use of instructional or institution labs - students may be required to take the exam using only devices provided by the department or institution.
  - BYOD (Bring Your Own Device) - students may be allowed or expected to bring their own device and take the exam using it exclusively; or as a backup or for those without a personal device, a lab computer or laptop checkout may be made available.
- *Delivery - Software* - A variety of software may be required or allowed during an executable exam.
  - Text editor - a bare-bones text editor which may or may not include basic syntax highlighting.
  - Instructors may require that specific handin and autograding software (commercial, open-source or home-grown) be used. Such software may come with lock-down features to address academic integrity (see Proctoring).
  - Instructors may provide or mandate that a particular IDE or development tool set be used.
  - Unrestricted development environment - students may be allowed to use whatever environment they choose as they would with regular programming assignments.
- *Grading* - Once submitted, executable exams may be graded automatically or with some level of human intervention.
  - 100% automated - only an autograder system is used for assessment with no human intervention (other than specific troubleshooting cases). Typically this means that assessment is based entirely on correctness.
  - Mostly automated with some human intervention. Typically this would mean that correctness is verified by the autograder test suite while other aspects (style, documentation, design, soundness or efficiency) are verified by a human grader.
  - Mostly human with some automation. In some cases, an instructor may wish to (re)verify correctness to ensure that students had not simply hard-coded solutions (in an attempt to undermine the autograder system) or so that partial credit can be awarded for "nearly" correct solutions that nevertheless fail a more rigorous and unforgiving autograder.
  - 100% human - a grader may manually run test cases in an ad-hoc manner as well as examining the code. In this case, the executability of the exam may simply serve as a check for syntactically correct code.

4 CASE STUDIES

4.1 Executable Exams in an Established CS1 Course

Executable exams have been used in some computer science courses at the University of Nebraska-Lincoln since at least 2012 and exclusively used in the flagship Computer Science I course since 2017 [3]. This course serves Computer Science and Computer Engineering majors as well as a substantial number of non-majors from dozens of disciplines. The course uses C and is delivered as a hybrid course with parallel in-person and online sections with an enrollment of over 200 students.

A midterm and final exam account for 25% of the assessment and are delivered as executable exams with programming exercises similar to what students complete for weekly assignments. Exam questions have consisted of simple from-scratch programs as well as moderately complex programs with starter code provided. Exams are designed to be completed in 1–2 hours. The most recent iterations have been asynchronous: students are given a 24 hour window in which to complete the exam and are allowed to take as much or as little of that time as they wish with no location restrictions.

All exam artifacts (code) are submitted electronically to a (home grown) web-based handin and grader system [2]. Students are typically provided a small suite of illustrative test cases in both the handout (to demonstrate how the code should work) and/or informal and formal unit tests. Students are allowed to submit, revise, and resubmit as many times as they like during the exam time. The autograder system provides feedback on syntax as well as correctness.

Because exams are asynchronous, students are allowed any resources, physical or electronic including course notes, library documentation or even open google searches. However, students are barred from direct collaboration with students or individuals. They are allowed to ask *clarifying* questions of the instructor or Learning Assistants (LAs), but the same level of help that we would typically provide on assignments is not given. Exam handouts and instructions are provided in an LMS and any starter code is provided via GitHub. Students have the opportunity to use our computer labs and university laptop checkouts, but nearly every student has a personal laptop.

One challenge of this setup is the lack of a strong proctoring mechanism. However, grader logs are monitored and analyzed and all submissions are run through MOSS [19] to detect plagiarism. Despite the openness of the exam, integrity violations have been rare. We credit the use of other best practices to discourage and dissuade integrity violations: a clear academic integrity policy including code usage guidelines, frequent reminders of the course expectations, demonstration of the effectiveness of MOSS (using anonymized submissions from a previous semester) and a strong department-level policy (three strike policy) reinforced in other introductory courses.

Exams are graded based on coding style, documentation, design (which includes efficiency) and correctness (the majority of assessment). Correctness is nearly 100% automated with points awarded proportionally to the number of test cases the code passes. Other categories are assessed manually by LAs using a prescribed rubric. LAs provide line-by-line feedback using codePost [15].

4.1.1 Outcomes. Student outcomes in this course have been consistent and comparable to historic offerings in which executable exams were not used. Data from the last several years is presented in Table 1. For every exam, more than 90% of students enrolled submitted at least one part of the exam. Grade distributions are also consistent year-to-year and comparable to traditional exams in previous offerings.

Exam grades are also highly positively correlated to final course (percentage) scores (with coefficients of .768 to .907). Since exams only account for 25% of the final grade, we interpret this strong correlation as a good indicator that executable exams are a valid assessment and a good measure of student learning outcomes.

Table 1: Exam mean, grade distributions and correlations of midterm and final exams 2019–2021. Correlations are between exam scores and final overall percentage. "100" are the percentage of students who received perfect scores and are also included in the "A" numbers. No final was given in 2020 due to COVID accommodations.

	2019		2020	2021	
	Midterm	Final	Midterm	Midterm	Final
Mean	73.35%	86.36%	79.86%	78.69%	75.16%
100	35.6%	38.4%	26.1%	20.8%	21.9%
A	43.7%	49.3%	48.3%	51.1%	51.0%
B	3.4%	11.6%	11.4%	9.8%	7.1%
C	13.2%	9.4%	1.7%	9.2%	2.6%
D	4.0%	7.2%	8.0%	4.0%	1.3%
F	35.6%	22.5%	25.0%	24.9%	38.1%
Corr.	0.792	0.907	0.861	0.768	0.823

4.1.2 Discussion. The ability to get feedback and to resubmit assignments has been shown to have a positive effect on student outcomes [11] and we've found the same is true for executable exams. Some educators have raised concerns that providing feedback during an exam has some potential negative risks [5, 17]. Specifically, students may get stuck on one part, struggling to get it correct and failing the rest of the exam. In a non-executable exam, they argue, students would move on and at least have a chance at the rest of the exam. This evidence is anecdotal at best and our experience directly contradicts these concerns (we also provide some suggestions to address these concerns below).

We've found that making exams as similar to the weekly labs and hacks (programming assignments) also makes students more comfortable and reduces the stress and anxiety associated with exams. The ability to receive feedback and resubmit also makes the assessment less summative than traditional exams.

4.2 Case Study 2: Introducing Executable Exams into CS1 Course

Our second case study involves the recent introduction of executable exams into a CS1 course at Technion - Israel Institute of Technology. This course serves Computer Science, Computer Engineering, Electrical Engineering and Data Science majors. The course uses the C programming language and is delivered on campus, with enrollment of about 1000 students each year.

The assessment of this CS1 course is based on 5 homework assignments (10%) and a final exam (90%). The final exam includes one multiple choice complexity question for three given functions, and three questions in which the students are given a problem and are asked to write code to solve it.

Up until the 2020 winter semester, this CS1 course was taught in a synchronous manner on campus and the final exam was paper-based. Due to the pandemic, the course was taught online in the 2020–2021 winter semester, and the final exam of the course was given online synchronously, with human proctors. This provided an opportunity to conduct our first executable exam. The question types in the exam were unchanged from the pre-pandemic paper-based exams.

In the 2021 summer semester, the CS1 course returned to campus, and it was decided to conduct an executable exam on campus with human proctors. A group of students from the previous semester also took a makeup paper-based exam that was identical to the final executable exam of the summer semester.

In the 2021 winter semester, which consisted of more than 1000 students, it was not feasible to test all students simultaneously on campus due to limited space and power outlets. Instead, students were given the opportunity to choose to participate in an executable exam. Out of the 1000 students 300 registered for the executable exam. Stratified sampling was used to randomly select half of them to take the executable exam (based on faculty, age and gender). This made it possible to compare results between different students' groups. We compared between students who had executable exam, those who also registered but had paper-based exam, and those who chose not to register and took the paper-based exam. We also compared between men and women.

In all exams, students were provided with a small suite of written illustrative test cases in the handout (to demonstrate how the code should work). Students were not allowed any resources, physical or electronic but were allowed to ask clarifying questions of the instructor or Learning Assistants. However, the same level of help that we would typically provide on assignments was not given. The students who took the executable exam were also provided with the test cases digitally, and they were allowed to compile and execute their program as many times as they like during the exam time.

One challenge of this setup is the lack of a strong proctoring mechanism, especially during the online exam. In the online exam the students were asked to use two cameras: one aimed at the computer screen, and the other aimed at the student, and they could submit as many times as they want to our autograder. In the following semesters when the exam was on campus, the students brought their personal laptops and had to disconnect from the network during the exam, and could not submit to our autograder. At the end of the exam they could submit their answers to an autograder.

Exams were graded manually based on efficiency and correctness using a prescribed rubric, while taking into consideration the results of the automatic grader.

4.2.1 Outcomes. Student outcomes in this course have been very consistent. Full data is presented in Table 2. The average final score is typically around 69% with a couple of exceptions. Students in spring 2021 who took the paper-based exam (with content similar

to the executable exam of summer 2021) had a lower average of 61%. We attribute this to the fact that it was a makeup exam which typically has weaker students. In the next semester, averages between the executable exam and paper-based exams were much closer with a gap of only 3 points. Overall, students who had opted-in to the executable exam scored higher than those who did not opt-in for it.

Table 2: Results of final exams 2020–2021. For each exam, the mean score, standard deviation (SD) and the total number (N) of students is reported. Terms (Sp = Spring, Su = Summer, W = Winter) and exam types (E = Executable, PB = Paper-based) are abbreviated. For winter 2021, those who registered for the executable exam but were not selected and took the paper-based exam are reported separately from those who opted for the paper-based exam.

	W '20	Sp '21	Su '21	W '21		
	E	PB	E	E	PB	
					Registered to E	All
Mean	69%	61%	71%	66%	69%	64%
SD	19	18	18	18	18	20
N	975	106	35	100	113	575

Differences in gender were also compared (Table 3) for the winter 2021 exam. Results indicate that the difference in scores between men and women were remarkably consistent between the paper-based and executable exam groups. Though men scored slightly higher, the consistency indicates that the executable exam format likely did not have a negative impact based on gender.

Table 3: Results of men and women of the final exams winter 2021. For each column, the mean score, std (SD), and total number (N) of students is reported.

	Executable		Paper based	
	Men	Women	Men	Women
Mean	69%	57%	66%	57%
SD	18	14	20	20
N	78	22	306	149

An interesting observation is that men opted for the executable exam format at a much higher rate (175/480 = 36%) versus women (51/309 = 15.5%). We intend to further explore this difference.

4.2.2 Discussion. This case study highlighted several advantages of executable exams compared with paper-based exams. The evaluation process, in general, and syntax checking specifically, were easier and faster. During the evaluation process code can be compiled, executed, tested, and auto-graded. This enhances the objectivity and consistency of the evaluation process [1, 10, 12, 16, 18].

One impediment to administering executable exams for all students is a lack of physical space and resources. There was a strong preference by faculty to administer exams on campus in order to have a stronger proctoring mechanism. Future solutions include administering multiple exams at different time slots and providing a computer to students who don't have one.

Another potential risk of executable exams is the possibility of shifting the assessment focus from algorithmic principles and problem solving to machine-verifiable input/output correctness, and lower completion rates than paper-based exams [5]. Our results, however, did not demonstrate a substantial difference in the completion rate between the two exam formats. The grades of the final exam suggest that students performance is independent of the exam format, though the shift of focus requires a deeper investigation.

Table 4: Case Study Course Profile Summary Summary

Course Profile	Case Study 1	Case Study 2
Students/Year	200	1000
Years	10	1
Question Types	Programs	Functions, Complexity
Exam Proportion	25%	90%
Taxonomy Profile		
Time/Place	Asynchronous, remote	Synchronous, in-person
Autograder Feedback	Syntax & correctness	Syntax & correctness
(Re)submissions	Unlimited	Limited
Allowed Resources	Unrestricted	Closed
Proctoring	Honor System	Human
Delivery - Hardware	BYOD	BYOD
Delivery - Software	Unrestricted	Specific IDE
Grading	Mostly automated, some human	Mostly human, some automation

4.3 Summary

The two case studies are summarized in Table 4. We have mapped the major characteristics back to our taxonomy to illustrate its usefulness. We note that both case studies represent successful administration of executable exams but vary significantly in the specific characteristics.

5 FACULTY PERSPECTIVES

The case studies provide detailed experiences in designing and administering executable exams, but we were also interested in a broader perspective. In particular, how widespread is the use of executable exams in other introductory computer science and coding classes? We developed a short survey and solicited feedback from professional CS educators via the SIGCSE listserv. We received 20 responses and present an overview of our findings here. While not a large sample (and it is somewhat biased toward dedicated higher education research professionals), it does provide some insight as to the current state of the practice.

We found that the use of executable exams is not uncommon, but it is by no means the standard practice. About 40% of respondents indicated they were using executable exams with the remainder using traditional paper-based coding or multiple choice/true false/etc. format exams (which may or may not be autograded but do not fall under our definition of executable exam). However, even among those that indicated they used more "traditional" exams, the majority 60% stated they were thinking of adopting executable exams. This gives us evidence that executable exams are becoming more

common and may (likely sooner rather than later) become standard practice.

5.1 Executable Exams - Characteristics

Among those that indicated they gave executable exams, we asked what the typical characteristics of their exams were with respect to our taxonomy. We describe the most common answers below.

- *Time/Synchronicity* - All respondents indicated that exams were limited to a 1-2 hour period with the majority administered synchronously. About a quarter allowed a window in which to take the exam but the exam time itself was still limited.
- *Grader Feedback* - All respondents indicated that the autograder provided feedback on syntax to students during the exam. A good majority indicated that feedback also included correctness.
- *Submission* - In alignment with the autograder feedback, the same majority said that they allowed an unlimited number of resubmissions based on autograder feedback. This makes sense as there would be no point in providing correctness feedback if students were not able to do anything about it. About a third only allowed 1 or a limited number of resubmissions.
- *Open Resources* - All respondents indicated that students were allowed some form of physical or digital resource with the most common being language documentation.
- *Proctoring* - No one indicated that exams were allowed to be collaborative. Most relied on traditional in-person monitoring. Only a small minority used electronic proctoring such as a lock down browser.
- *Delivery* - Most exams use a BYOD setup with the others using institutional labs to deliver the exam. Some respondents indicated that students were given a choice (especially to accommodate those without a personal laptop).
- *Assessment* - The vast majority indicated that assessment of exams relied entirely on correctness. Only a minority indicated that style, documentation, design, and efficiency were also factors. However, only about a third indicated that assessment was 100% automated (relying on the autograder system entirely). The majority used a human grader to *some* extent.

5.2 Advantages

All survey respondents (regardless of whether or not they used executable exams) were asked to identify potential advantages of using executable exams. The most obvious and common responses are the same as for utilizing autograders for regular assignments. Executable exams provide scalability for grading, require minimal resources for administration and assessment and makes the assessment process more transparent and objective.

Instructors also identified executable exams as a more realistic setup to how students coded compared to paper-based exams and certainly to artificial "paper-based" exams. Responses indicated that it also allowed students to focus on problem solving and logic rather than syntax or compiler issues.

One respondent pointed out that executable exams present a *unique* opportunity to assess students debugging and refinement abilities by tracking submission/resubmissions based on autograder feedback.

5.3 Disadvantages

Respondents were also asked to identify potential disadvantages or risks to using executable exams.

Technical issues during an exam (connectivity, power supply, computer reliability, etc.) provide the biggest risk to instructors. Having to troubleshoot during a high-stress exam can magnify student anxiety and take focus away from the core concepts the exam is meant to assess.

Others pointed out that fully automated grading can limit the potential to award partial credit. Unforgiving test suites present a binary outcome: a test either passes or fails. A student solution may be nearly correct except for a minor mistake that results in a disproportionate number of tests failing. For example: a student may correctly code the logic to determine if a year is a leap year but accidentally flip true/false resulting in all tests failing.

Another concern was that students may get stuck on one problem of get stuck on the minutiae trying to perfect it with respect to grader feedback at the cost of not completing other major parts of the exam.

However, by far the most common concern was with academic integrity and a perceived risk of increased opportunities for cheating. The use of a computer opens many new possible ways to cheat but at the same time offers instructors the opportunity to use more sophisticated tools for plagiarism detection. Interestingly, several respondents indicated they had adopted executable exams or were thinking of doing so as a *response* to integrity issues on regular programming assignments. We intend to further explore this perspective.

Many of these concerns and risks have documented mitigation strategies [4, 7, 8].

6 CONCLUSION & FUTURE WORK

We have presented evidence that executable exams are not uncommon in introductory computer science courses and are likely to be more widely adopted going forward. We have presented two case studies of successful administration in large CS1 courses and identified good practices for doing so. We have developed an initial taxonomy of characteristics of executable exams that can and will be refined going forward.

On balance, executable exams may only provide a slightly better experience for students or there may be little difference in outcomes. However, all things being equal, the scalability and time saved in grading itself justifies the use of executable exams

For instructors that have already adopted executable exams, we hope this paper gives them confidence they are not alone and prompts further discussion and development of best practices. For those that are considering adopting executable exams, we hope this has given you the confidence to do so.

Further research and data on executable exams is necessary as they become more prevalent. In particular, we have already begun gathering data and feedback on student perspectives and plan a tier-analysis on future exams to determine whether or not executable exams disproportionately benefit one group of students (high/low performing) or not.

REFERENCES

[1] João Paulo Barros, Luís Estevens, Rui Dias, Rui Pais, and Elisabete Soeiro. 2003. Using Lab Exams to Ensure Programming Practice in an Introductory Programming Course. *SIGCSE Bull.* 35, 3 (June 2003), 16–20. https://doi.org/10.1145/961290.961519

[2] Chris Bourke. 2015. CSE Webgrader. https://github.com/cbourke/grade/. https://github.com/cbourke/grade/

[3] Chris Bourke. 2021. *Development & Evolution of a Computer Science I Course.* Technical Report. University of Nebraska-Lincoln. https://digitalcommons.unl.edu/prtunl/202/ Available, https://digitalcommons.unl.edu/prtunl/202/.

[4] Binglin Chen, Matthew West, and Craig Zilles. 2017. Do Performance Trends Suggest Wide-Spread Collaborative Cheating on Asynchronous Exams?. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale* (Cambridge, Massachusetts, USA) *(L@S '17)*. Association for Computing Machinery, New York, NY, USA, 111–120. https://doi.org/10.1145/3051457.3051465

[5] Jonathan Corley, Ana Stanescu, Lewis Baumstark, and Michael C. Orsega. 2020. *Paper Or IDE? The Impact of Exam Format on Student Performance in a CS1 Course.* Association for Computing Machinery, New York, NY, USA, 706–712. https://doi.org/10.1145/3328778.3366857

[6] Christopher Douce, David Livingstone, and James Orwell. 2005. Automatic test-based assessment of programming: A review. *ACM J. Educ. Resour. Comput.* 5 (2005), 4.

[7] Yael Erez and Orit Hazzan. 2021. 10 Tips for Implementing Executable Exams. Communications of the ACM BLOG@CACM. https://cacm.acm.org/blogs/blog-cacm/253802-10-tips-for-implementing-executable-exams/fulltext

[8] Max Fowler, David H. Smith, Chinedu Emeka, Matthew West, and Craig Zilles. 2022. Are We Fair? Quantifying Score Impacts of Computer Science Exams with Randomized Question Pools. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 647–653. https://doi.org/10.1145/3478431.3499388

[9] Chelsea Gordon, Roman Lysecky, and Frank Vahid. 2020. *The rise of the zyLab program auto-grader in introductory CS courses.* Technical Report. zyBooks. https://docs.google.com/document/d/e/2PACX-1vQYwxlY738_9zFFwOer1kKTNGuJx1Qe3IDW8XHf_OOYbaq9Drf_a9ljCqjcHY9Vv4ryPK423W7FmHwZ/pub Updated March 2022.

[10] Scott Grissom, Laurie Murphy, Renée McCauley, and Sue Fitzgerald. 2016. Paper vs. Computer-Based Exams: A Study of Errors in Recursive Binary Tree Algorithms. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 6–11. https://doi.org/10.1145/2839509.2844587

[11] Amanda M. Holland-Minkley and Thomas Lombardi. 2016. Improving Engagement in Introductory Courses with Homework Resubmission. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 534–539. https://doi.org/10.1145/2839509.2844576

[12] Amruth N. Kumar. 2010. Closed Labs in Computer Science I Revisited in the Context of Online Testing. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (Milwaukee, Wisconsin, USA) *(SIGCSE '10)*. Association for Computing Machinery, New York, NY, USA, 539–543. https://doi.org/10.1145/1734263.1734443

[13] Oka Kurniawan, Norman Tiong Seng Lee, and Christopher M. Poskitt. 2020. *Securing Bring-Your-Own-Device (BYOD) Programming Exams.* Association for Computing Machinery, New York, NY, USA, 880–886. https://doi.org/10.1145/3328778.3366907

[14] Vesa Lappalainen, Antti-Jussi Lakanen, and Harri Högmander. 2016. Paper-Based vs Computer-Based Exams in CS1. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '16)*. Association for Computing Machinery, New York, NY, USA, 172–173. https://doi.org/10.1145/2999541.2999565

[15] Jérémie Lumbroso and James Evans. 2020. Making Manual Code Review Scale. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 1390. https://doi.org/10.1145/3328778.3367026

[16] Hannah E. Murphy. 2017. Digitalizing Paper-Based Exams: An Assessment of Programming Grading Assistant. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 775–776. https://doi.org/10.1145/3017680.3022448

[17] Chris Piech and Chris Gregg. 2018. BlueBook: A Computerized Replacement for Paper Tests in Computer Science. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 562–567. https://doi.org/10.1145/3159450.3159587

[18] Felipe Restrepo-Calle, Jhon J. Ramírez Echeverry, and Fabio A. González. 2019. Continuous assessment in a computer programming course supported by a software tool. *Computer Applications in Engineering Education* 27, 1 (Jan. 2019), 80–89. https://doi.org/10.1002/cae.22058

[19] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. 2003. Winnowing: Local Algorithms for Document Fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (San Diego, California) *(SIGMOD '03)*. Association for Computing Machinery, New York, NY, USA, 76–85. https://doi.org/10.1145/872757.872770

[20] Matthew West, Geoffrey L. Herman, and Craig Zilles. 2015. PrairieLearn: Mastery-based Online Problem Solving with Adaptive Scoring and Recommendations Driven by Machine Learning. In *2015 ASEE Annual Conference & Exposition*. ASEE Conferences, Seattle, Washington, 26.1238.1 – 26.1238.14. https://peer.asee.org/24575.