The Future of Computing Education Materials

PETER BRUSILOVSKY*, University of Pittsburgh, USA
BARBARA J. ERICSON, University of Michigan, USA
CAY S. HORSTMANN, San José State University, USA
CHRISTIAN SERVIN, El Paso Community College, USA
FRANK VAHID, University of California, Riverside / zyBooks, USA
CRAIG ZILLES, University of Illinois, USA

For six decades, ACM and the broader computing community have gathered to establish guidelines for Computer Science (CS) curricula [3]. In Spring 2021, a steering committee was formed to establish new curricular guidelines and recommendations for CS education for the next decade. This effort is currently referred to as CS2023. As part of this effort, a collection of articles, named curricular practices, will include several articles by educators on the design and delivery of Computer Science programs. One of the curricular practices is the Future of CS educational materials, where educators, administrators, authors, and practitioners predict the factors that will influence the CS community over the next ten years based on current trends.

CCS Concepts: • **Social and professional topics → Computer science education**.

Additional Key Words and Phrases: computer science, educational materials, learning, feedback, textbook, videos, animation, homework, assessment, automation, adaptive, sharing

ACM Reference Format:
Peter Brusilovsky, Barbara J. Ericson, Cay S. Horstmann, Christian Servin, Frank Vahid, and Craig Zilles. 2023. The Future of Computing Education Materials. In *CS2023: ACM/IEEE-CS/AAAI Computer Science Curricula, Curricula Practices.* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

CS education (CSEd) builds on educational materials, such as textbooks, presentation slides, labs, and question test banks. Over the past 10-20 years, those materials have grown to include videos, animations, in-class activities, online homework systems, and auto-graded programming exercises. Educational materials aim to improve student success and elevate the instructor's role. This paper focuses on the future of educational materials in CS education, including research on effective approaches. Educational materials are becoming more active, involving immediate feedback to students throughout the formative and summative stages of learning. Educational materials include more artificial intelligence (AI) to provide good feedback, adapt to the learner, and provide help. More materials are cloud-based and researchers are starting to collect and analyze data for continual improvement. Materials increasingly aim to support social aspects of learning, including peer learning and coaching. Open education resources (OER) continue to grow, as do products from educational technology companies. The need is growing to customize, create, and share materials and quickly find high-quality materials. Automation is increasing for administering a class, interoperability of tools continues to grow in importance, and LMS capabilities continue to improve. Emphasis on materials that support traditionally underrepresented students is increasing. Materials increasingly support mastery-based learning and emphasize student success over student filtering. Educational materials will continue to include both sanctioned and unsanctioned online resources like homework help websites, cloud-based code repositories, and video sharing sites, posing challenges to and opportunities for instructors. This work aims to describe the future of educational materials in CS education and to encourage educators and administrators to recognize areas for development, adaptation, and innovation.

CS Education has always been a pioneer in developing and exploring new types of learning materials. The research on interactive learning content such as algorithm animations and program visualizations [11], programming problems with automatic assessment [7], and intelligent tutoring systems [2] started in CSEd field over 30 years ago and inspired similar research in other fields. While these types of tools are not really new, what we see today is a notable expansion of the variety of these tools and their rapid integration into thousands of CS classes. The increased use of interactive materials is now encouraging transition from the application of individual tools to various structured collection of learning materials, from interactive e-books [22] to integrated practice systems [12]. This transition motivated a recent interest in interoperability of learning content and infrastructures to support content integration [13] as well as growing interest in collecting learning data and using educational data-mining for further improvement of learning process [44]. The remarkable advances of AI research are also impacting many sides of CSEd from personalized learning to AI pair programming [23] to AI support for instructors [63]. It is important that the CSEd community understands these new trends and leverages new opportunities.

2 ADAPTIVE LMS / TUTORS

TBA

3 LARGE LANGUAGE MODELS (LLMS)

With the advent of Large Language Models (LLMs), such as OpenAI's ChatGPT, OpenAI's Codex, and GitHub's Copilot, there are both new opportunities and concerns for the future of computing education. ChatGPT is a chatbot developed by OpenAI that is built on

*all authors contributed equally to this research.

CS2023: ACM/IEEE-CS/AAAI Computer Science Curricula, ,

GPT-3 (Generative Pre-trained Transformer 3rd generation) which is a neural network machine learning model that was trained using data from the internet. Codex was built on top of OpenAI's GTP-3 and was trained on public Python repositories on GitHub [16]. GitHub's Copilot was built on top of Codex and can serve as an AI pair-programmer in Visual Studio Code. Instructors are concerned that students will use LLMs to automatically generate code for their class work and not learn how to write programs themselves [23]. However, students have long been able to find code for common programming problems on the internet [26] or even pay someone else to write code for them [52]. What is different now is that LLMs are being directly integrated into Integrated Development Environments (IDEs) like Visual Studio Code, which makes it much easier for students to use them to generate code [23]. However, a study with 24 programmers using Copilot found that the generated code was not always correct and programmers found the generated code difficult to understand, edit, and debug [77]. One the other hand, the programmers felt that the generated code was useful as a starting place, even if it was buggy. The generated code was more likely to be buggy in the harder problem that they tested. Programmers who successfully solved the harder problem broke the problem into parts and generated the code for each part. Another study found that Codex could generate simple Python programs in both procedural style and object-oriented style correctly, but not more complex programs for a course on automation and controls [10]. The code generated by Codex for exam problems in Python can outscore most students [23] and can solve the "Rainfall Problem" which has long been used in computing education research [65, 51, 24, 72, 73]. LLMs will also likely improve with time.

There are already attempts to help instructors detect use of LLMs and some instructors will no doubt ban the use of LLMs in their course. While LLMs will generate different answers for the same prompt, they only generate a few different answers so students who use LLMs in a larger course will likely have the same code as other students. Another indicator can be that the code style is different than the recommended style for a course. If all programming is done in an instrumented system another sign is that the time it takes to create the code is much shorter than normal. Some instructors will also increase the weight of proctored exams in controlled environments where LLMs can not be used in order to verify that the student can really program on their own.

Perhaps we should not restrict the use of LLMs, but instead harness them to improve computing education. In a recent study with 69 novices (aged 10 to 17), the use of Codex to generate code followed by a code modification task led to significantly higher code authoring completion rates and scores than writing the equivalent code without Codex [47]. We have long known that writing code from scratch is a high cognitive load task that can easily overwhelm novice programmers [6, 80, 78]. One of the recommended ways to reduce cognitive load is to use completion tasks rather than whole tasks like write code from scratch [79]. With LLMs we could put more emphasis on comprehending, modifying, debugging, testing, and extending code rather than writing code from scratch.

The ICAP theory describes four modes of learner engagement: Interactive, Constructive, Active, and Passive [18, 17]. It hypothesizes that interactive results in the greatest learning, then constructive, then active, and finally passive. An example of passive mode is a typical lecture where the learner is not taking any notes. An example of active mode is answering multiple choice questions. An example of constructive mode is a learner writing notes in their own words during lecture. An example of interactive mode is a discussion between learners who are explaining why they picked different answers to a multiple-choice question. LLMs could improve learning in computing if they result in more interaction. This is consistent with Bloom's finding that group with a good tutor per one to three students performed on average two standard deviations higher than a group in a conventional classroom with one teacher for 30 students [9].

LLMs may be a low cost and scalable way to offer personalized help to students who struggle while writing code. They can be used to explain code to students, fix code, and generate code [53]. This could improve the success of students in introductory programming courses as well as the diversity of students. Women and people from marginalised groups are often more at risk of failure in introductory programming courses since they tend to have less prior programming experience. Instructors can also use LLMs to generate problems, explanations, test cases, and translate code from one language to another [64]. One approach is to use LLMs to generate a mixed-up code (Parsons) problem to guide struggling students to a correct solution when they are stuck while writing code [39]. We may even be able to generate a personalized Parsons problem, one that is based on the student's incorrect solution using LLMs and use an LLM to generate an explanation if the student achieves a correct solution but doesn't understand the solution. If students know that they can achieve a correct solution with this type of scaffolding they may be less likely to rely on LLMs to generate all the code for them.

4 SMART LEARNING CONTENT

Traditional content in digital learning materials consists of text, images, videos, and simple user interaction such as multiple-choice or fill-in-the-blanks quizzes. [14] coins the term "Smart Learning Content" for digital content with advanced interactive features. Here is a non-exhaustive list of typical examples:

- Program visualization (JEliot [54], JGrasp [19], Python Tutor [32]: Providing a visual display of the flow of a program
- Algorithm visualization: Showing the progress of an algorithm, updating a visual representation of a data structure
- Interactive tracers and tracer toolkits (TRAKLA2 [48], JSAV [45], CodeCheck Tracer [37]): Involving students in the tracing of programs or algorithms by asking for predictions of data updates and decisions
- Autograders (such as Web-CAT [21] and CloudCoder [40]): Evaluating student submissions of complete programs or program snippets
- Toolkits for block programming, Parsons puzzles ([81], [43]): Authoring of problems in which students compose programs from generic or problem-specific blocks
- Tracers for automata, Turing machines, simulated CPUs, and digital circuits (see for example [61], [83])
- Simple activities, for example, practicing the evaluation of nested expressions [49])

Ideally, it should be possible for teachers to locate appropriate smart learning content, adapt it to their needs, embed or link it in their content delivery system of choice, and receive feedback on student activity in their learning management system.

4.1 Algorithm Visualizations

It is instructive to review the history of algorithm visualizations as a typical example of smart learning content. An algorithm visualization provides an animated trace of the execution of an algorithm, generally with controls for single-stepping, controlling the animation speed, and obtaining additional problem instances. These visualizations date back to at the 1980s, but the advent of Java applets in 1996 led to wide-spread distribution ([56]) since applets ran in off-the-shelf web browsers without requiring any software installation.

The Algoviz catalog [68] listed over 500 visualizations as of 2010. Sadly, both the catalog and most of the animation are no longer alive in 2023. Modern browsers no longer execute Java applets. There are some JavaScript-based visualizations ([8]), but it appears that JavaScript is significantly less popular as a development tool among the CS educational community than Java was.

Moreover, research on the efficacy of algorithm visualization has not been encouraging ([42]). The relatively passive activity of watching an animation does not appear to contribute to significantly improved learning outcomes, but asking students to predict steps may be a viable strategy ([76]).

A notable example of such interactive content is the OpenDSA project [69], which makes use of the JSAV toolkit [45]. The toolkit enables the creation of both passive and predictive visualizations. The OpenDSA textbook contains a large number of such problem instances. Using a toolkit greatly facilitates the construction of multiple instances with a consistent user interface. Other such toolkits include CodeCheck Tracer and Galant [74]).

4.2 Smart Programming Content

An autograder is a service that checks whether student-provided code is correct, by compiling and executing the code. Instructors provide problem descriptions and test harnesses. Some autograders (such as CodeWorkout [20], CodingBat [59], CodeCheck [36]) include collections of problems. These can be useful materials, particularly for introductory classes. Most autograders also allow instructors to create new materials. [57] contains a recent comprehensive survey.

Several autograders (including CodeWorkout and CodeCheck) provide integration with learning management through the LTI standard. Problem instances can be assigned in learning management systems, and the student scores flow into the gradebook.

Embedding autograder problems inside documents, such as an assignment or lab manual, is not widely supported. The OpenDSA project has built a custom CodeWorkout integration. With some effort, CodeCheck problems can be added to HTML documents and served as LTI resources.

A Parsons puzzle [60] is a mechanism for teaching programming in the small. Students arrange lines of code in the correct order and indentation, skipping any distractors. Sites such as [1], [34], and CodeCheck make it easy to produce Parsons puzzles and obtain URLs that can be linked or embedded. Some of these tools support LTI.

A Jupyter notebook is a document that combines text and code snippets in Python or another language for which there is a "Jupyter kernel". The document can explain a computation, or it can invite the user to experiment with the code. Hosting notebooks on the internet requires a service (such as Google Colab), but the JupyterLite project uses WASM to run notebooks entirely in browsers. Automatic grading and LMS integration are currently rudimentary.

4.3 Barriers to Adoption

Instances of smart learning content are plentiful, yet it is not easy to integrate or adapt external content in one's own teaching materials. Commercial and OER publishers provide platform for the authoring and display of electronic books (such as Wiley Zybooks, Pearson Revel, OpenDSA). However, each platform comes with its own limited set of smart content types. Authors must master the platform-specific problem editors and have no or limited ability to enhance the smart content.

A potential solution would be EPUB3 [28], an HTML based ebook standard which allows arbitrary JavaScript content. However, the EPUB for Education effort [75], which defines an API to communicate scores and persist student work, has stalled.

[14] notes the challenge of conceptual integration. It is not enough to link or embed existing smart learning content from different sources. Each element needs to be customizable, for example to a specific natural language, programming language, coding style, mathematical notation, or terminology. This customization is possible when the integrator has the ability to edit the content (for example, the HTML description and JavaScript configuration). It fails if the contents is immutably locked inside a platform.

Learning management systems allow the integration of smart learning content via the SCORM, LTI, and xAPI standards. This integration works well when the smart contents is delivered through a publishing platform, but it is not suitable for instructor-authored lecture notes, lab manuals, or assignments. An exception is Moodle which allows instructors to produce assignments that are composed of multiple LTI activities.

To carry out instructional research, finer-grained activity tracing would be desirable. Inclusion in adaptive systems would require generation of new problem instances. Such communication is currently only achieved in monolithic systems.

In 2014, [14] proposed a flexible architecture for course authoring, smart contents reuse, and data collection. Unfortunately, we are today no closer to a solution for flexible deployment of smart learning content.

5 PEER TEAM AND WORKBOOKS

Peer-led Team Learning (PLTL) is an educational resource and model to provide supplementary outside-classroom education provided by students to other students. The model was first introduced by [31] for science fields such as Biology and Chemistry. It was then adopted in the US to other areas, including computer science. The model is used as a mentoring practice from instructors to students and from

students to students. In computer science education, variations of the model have been adopted in two-year programs, such as community and technical colleges, to disseminate CS 1 and 2 concepts and material in addition to providing *coaching* capabilities to students using face-to-face and online modality [67]. Similarly, the model has been used to build and increase relationships between two and four-year institutions by increasing students' confidence in studying CS and continuing career paths by transferring to four-year institutions [29]. The model recognizes that *peer-leaders*, the students who mentor other students, increase marketable skills such as self-efficacy in teaching computer science and improved content knowledge, communication, and leadership skills [41]. Accordingly, it is one of the reasons used to increase participation and active recruiting of under-represented groups in introductory courses in computer science [38]. Due to the fast enrollment in computer science programs (such as high school partnerships, community colleges, and four-year) raises challenges to accommodate students in classrooms and support peer teaching environments quickly. To address these challenges and keep retention in classrooms, tools such as [70] for tracking one-to-one peer teaching interactions in peer teaching environments.

5.1 The usage of Quick Response (QR) Codes

gain popularity during COVID-19 [**statista**] as the piece of technology to locate digital data. Due to the ubiquity of mobile devices, QR codes allow students to access materials immediately on their mobile device (not necessarily a personal computer). This aspect has brought the opportunity to bring different techniques to incorporate innovative pedagogical techniques.

6 OPEN EDUCATIONAL RESOURCES

Open Educational Resources (OERs) are important in education, primarily at higher education levels, for several reasons:

- they are openly licensed (permits that anyone can use the work, modify it, and distribute based on the discretion of the author)
- they are free and available for anyone (student, instructor, researcher) depending on the available repository
- they are motivated to provide an alternative and enhanced educational paradigm from the current educational models.

These characteristics are significant and attractive to institutions that concentrate on the CS core or at least for the first two years of CS Education, such as liberal arts colleges and community/technical colleges that pursue similar objectives in their mission, e.g., [35, 66].

7 EBOOKS

Prior to digital technologies, the main way for publishing learning content was through physical textbooks. Being written for paper, such learning content was mostly limited to text and static figures, with precise page layouts. The advent of personal computers and then the web led to learning content throughout the 2000s being increasingly available online. The first wave of such content consisted largely of physical textbook content being ported to the online format of pdf, often called "ebooks" for electronic books. Such content yielded advantages of access from different locations or devices, electronic search, easy highlighting and annotation, and sometimes lower cost, but with a drawback for some of being less comfortable to read versus paper. Over time, such textbooks were ported to more digital-native forms, such as HTML/CSS for the web, or various digital book formats designed for tablets or book readers, yielding improved readability. The EPUB format is a standard starting around 2011 for publishing textbook content on various platforms, making use of HTML/CSS and including support for audio, video, graphics, text-to-speech, math equation displays, and some interactivity. Electronic books thus increasingly added features beyond text and figures, such as links to videos, or simple self-assessment questions. An example is the OpenStax college textbook library (openstax.org), whose textbooks are available in either physical, pdf, or html form, with the latter including additional features.

In the web era, separate online homeworks systems were developed that automated many of the end-of-section exercises found in textbooks, such as WebAssign (webassign.net) or like CodeLab (turingscraft.com), MyProgrammingLab (https://mlm.pearson.com/northamerica/mypr or MindTap (https://www.cengage.com/mindtap/). Continuing that trend, The Runestone system (https://runestone.academy/) was developed with an emphasis on free interactive textbooks available online, with interactive questions and coding homework problems built directly into the system.

A recent trend is for textbooks to be written natively for the web, wherein authors make use of all available features, which includes dynamic figures (animations), interactive questions, custom tools or simulations, videos, and more, in addition to text and static figures. As such, those "textbooks" tend to shift much of the learning to those other features, and contain much less text. An example is zyBooks (zyBooks.com), whose system integrates text, figures, animations, interactive questions, homework problems, and programming assignments in one "book".

Another trend is configurability, wherein an instructor can rearrange the order of a book's sections, can remove sections, can supplement the book's content with their own sections, and can combine sections from different books, all resulting in one book that directly matches a course's topic schedule. Many of the above systems support such configurability.

Electronic textbooks increasingly record student activity data, such as a student's submitted answers to interactive questions, the date and time when a student watched a video and how much was watched, or the time a student spends reading some text (making use of mouse activity). A key use of such activity is by instructors who assign points for student activity, such as for completing the reading of a chapter, for answering interactive questions, or for doing homework problems. Another use is to assist instructors in course design, for example by noting how much time students require for activities and adjusting accordingly, or by highlighting subjects or homework problems causing students to struggle so that instructors can adjust their class time in response. Data can also give publishers and authors unique insights into how students use their content, such as which items they study carefully versus skimming over, which items cause struggle, or how well a student learned a subject, so that publishers and authors can continually improve their content. Publishers can also examine how instructors are using their content, such as which content sections are

most commonly kept versus removed, which section orderings are most common, etc., to help publishers focus their limited effort on the most important content sections. Furthermore, publishers can explicitly solicit data from instructors and students from within electronic textbooks, such as allowing them to provide feedback on any item in the content like a particular animation or homework problem, or allowing them to rate sections, so that publishers can quickly detect and improve problematic items, or can efficiently receive suggested improvements.

Students have long requested numerous "worked examples" from which to learn. Publishers typically include some worked examples, but they are difficult to create and maintain, and can take up much space. Recently, AI tools have demonstrated the capability of generating examples automatically, such as "Write a program to determine the median in a list of numbers," which can yield a program in a variety of languages, along with a thorough description of each part. As such, one might expect future electronic books to incorporate such AI tools, proposing examples for students to generate and allowing students to go further as well.

Due to the above, the availability of a physical textbook to accompany electronic textbooks may be decreasing. First, a physical textbook may not match the online book due to configuration by the instructor. Even if printed from the configured electronic textbook, the printout may not have the careful page-layout of traditional books, and will lack interactive questions, animations, videos, etc. Physical textbooks also cannot record data. As such, physical instances of electronic textbooks are commonly used for secondary purposes, like keeping permanent access to the content, enabling access without the internet or a computer, or making studying for exams a bit easier.

Traditional textbooks were published using an "editions" model, wherein a textbook would be published in a given year as an edition such as "3rd edition", and that edition would be stable for several years, until authors made a revision to the textbook to publish the next edition. However, electronic textbooks can be updated more frequently, and thus the editions model is beginning to give way to a "continual publishing" model, wherein substantial revisions may occur more frequently like every year or every 6 months, careful not to disrupt a class's active term. And, minor revisions such as error corrections or wording improvement can be made more frequently, even daily, akin to how modern internet software may experience frequent updates.

New features appearing in electronic textbooks include built-in discussion forums and help. Traditional discussion forums are separate applications, but discussion threads often focus on specific items like homework problems, labs, figures, etc. As such, creating or accessing such threads directly attached to such items makes the discussion more available to students. Furthermore, help is also often associated with specific items, and thus help in the form of hints, previously answered questions, or even live chat with a human, may be best if directly attached to the relevant item. In fact, AI tools have shown their ability to answer many student questions, like "What does this program do?" or "Why isn't this program correctly outputting the average?", with immediate replies from the AI tool acting as a discussion or help itself.

Textbook prices became a key concern when such prices rose dramatically over recent decades. Interestingly, the rise of electronic textbooks does not automatically decrease prices, because physical books are surprisingly inexpensive to print and thus their cost contributes only a small fraction to a book's price, such as 10%. Most of the price goes to cover all the other costs of producing a textbook: paying authors, editors, graphic designers, reviewers, attorneys, administrators, administrative assistants, human resource staff, plus buildings/computers to support those people, plus bookstore markup as well. Nevertheless, prices of electronic textbook prices may be lower than physical textbooks because of a more directly relationship between publisher and user, since most electronic textbooks are acquired via a subscription and thus revenue can go directly between the user and publisher without middlemen like rental companies or used-book sellers siphoning revenue without actually contributing to creating the product itself. Such increased "sell through" can enable publishers to better amortize their creation costs over larger numbers of users. On the other hand, electronic textbooks can be much harder to create and maintain, requiring engineers, content creators, analysts, and more, to focus on building and maintaining the software platform, protect it from attacks, ensure accessibility for various disabilities, provide technical support, analyze data and improve content, and more concerns that are largely non-existent for physical textbooks.

Looking ahead based on the above trends and project forward, one might expect future CS electronic books to:

- Have extensive interactivity, including interactive questions, animations, videos, simulations, integrated homework, integrated labs, and more, including support for known effective practices like active/flipped classrooms, pair programming, etc.
- Support extensive configurability: moving, removing, supplementing, and even mixing books. In fact, one might imagine each textbook as a convenient grouping of topics, like a chapter but one level higher. With sections from different books being merged, consistency in authoring style across textbooks may become increasingly important for publishers.
- Collect and analyze data extensively: Auto-collected data enables instructors to award points and may save instructors time. Such data, plus explicitly collected feedback, can help instructors and publishers continually improve the content.
- Continually publish: Multi-year stable editions will likely be replaced by content that incurs minor updates daily, and more substantial updates on a scale of months (careful to not disrupt active classes).
- Include built-in discussion forums, live help, self-assessment, and some adaptivity.
- Use AI tools throughout, to generate worked examples, auto-generate self-assessments, answer questions in a discussion/help forum, and more.

8 ASSESSMENT

Assessment plays many important roles in education. First, much of the learning happens when we try to apply our knowledge[33, 46, 62]. Second, assessment can be an opportunity for feedback, so

that we know what we've mastered and what we still need to work on [82]. Finally, assessment can act as an extrinsic motivator; independent of our students' intrinsic motivation, they live complicated lives with many competing priorities, and homework deadlines and summative assessment can provide a valuable nudge to get students to spend time on task.

In formative assessment, where the goal is student learning, three key elements are productive time on task, feedback, and spaced repetition. These elements suggest having a large collection of practice problems available to students spanning the whole spectrum of difficulty, so students can be working in their Zone of Proximal Development. Construction, calibration, and maintenance of such a large problem bank is a larger effort than is feasible by a single instructor, so will likely come through either corporate investments and/or faculty collaboration in open education resource repositories that can amortize the cost over many classrooms.

We expect three techniques to prove useful for constructing large problem banks. First, item generation [30] involves writing problem templates that can be programatically instantiated to generate a range of problems, a technique often used for computation oriented problems (e.g., convert this IEEE floating point representation to scientific notation). Second, for programming questions, approaches that can generate versions that are isomorphic [25, 58] to a given base question, either manually or automatically. Finally, the aforementioned use of large language models will assist in generation of novel questions [64].

Because of the correctness of much of work in computing can be evaluated programmatically, we expect that auto-grading will continue to play a significant role in formative assessment, precisely because it enables immediate feedback, allowing students to attempt problems, receive feedback, remediate misconceptions, and then attempt another version of the problem in quick succession, while the material is still in the student's working memory. Today most autograders are deployed for work that has clear correctness criteria, but further adoption of machine learning will extend auto-grading into additional domains (e.g., free text responses, style, design) that are largely graded manually.

With a large, auto-grading pool of diverse kinds of problems, we can give students the opportunity to frequently self-assess, at every step of instruction: during first contact with material (e.g., reading), while learning to apply it (e.g., in-class activities, homework), and during review (e.g., around summative assessment). Such assessments may point students back to sections or activities determined to be needed areas of improvement, allowing students to strengthen their knowledge and then re-assess. Such systems can be adaptive, giving students who answer correctly fewer questions, and potentially suggesting which problems would be productive to work on next. The goal should be to minimize the friction between a student's desire to practice and their ability to do so, while maximize the learning gain from each hour of time on task. All students interactions with the system should be automatically tracked to relieve course staff from manual grade entry.

In summative assessment, where learning is still important, but the priority shifts to evaluating what a student has mastered, we have the additional requirement of security. If we give a student an exam, we want to make sure the work that is being scored is the student's work. While we have existing strategies for this with paper exams for residential universities (e.g., proctoring), computer-based exams offer compelling advantages for CS courses. On a computer-based exam, students can write code with access to a compiler and debugger, making it a much more authentic task, and having the students' submissions in a digital form make it easier to grade efficiently (independent of whether we use auto-grading or an online tool to facilitate manual grading). Because of the sophistication of modern browsers and CS students comfort working with computers, one can actually run more sophisticated questions on computer-based exams than paper-based ones.

The downside of computer-based exams is that they potentially enable students to communicate with others and services like ChatGPT. In the long run, proctoring alone is unlikely to prevent this communication and some form of "locking down" the machine is required. Two approaches may be viable. First, students can take computer-based exams on institutional machines, either in the form of "lab exams" [5, 15] or a dedicated computer-based testing facility [85], or by forcing students to run lock-down browser software that controls what can be done on bring-your-own-device (BYOD) exams.

While many large courses have reduced the administrative burden of homework to a minimum (through the use of online submission and significant use of auto-grading), exams are still a high effort activity for most instructors. Bringing the effort of exam administration close to that for current homework offers a lot of potential pedagogical advantages. First, running frequent assessment can improve student performance and reduce their test anxiety [4, 71]. Second, if running an exam is cheap, we can permit students to re-take exams when they demonstrate that they haven't mastered the material [55, 50, 27]. In addition to auto-grading, reducing the administrative burdens of exams requires reducing the burden of administration (e.g., dealing with conflicts) and proctoring. A centralized computer-based testing facility addresses these through a centralized proctoring service and running exams asynchronously and providing software support for exam scheduling and administration [84].

REFERENCES

[1] Laurent Abbal. *Parsons Puzzle Creator.* URL: https://codepuzzle.io (visited on 02/28/2022).

[2] J.R. Anderson and B. Reiser. "The LISP tutor". In: *Byte* 10.4 (1985), pp. 159–175.

[3] William F. Atchison et al. "Curriculum 68: Recommendations for Academic Programs in Computer Science: A Report of the ACM Curriculum Committee on Computer Science". In: *Commun. ACM* 11.3 (Mar. 1968), pp. 151–197. ISSN: 0001-0782. DOI: 10.1145/362929.362976. URL: https://doi.org/10.1145/362929.362976.

[4] Robert L Bangert-Drowns, James A Kulik, and Chen-Lin C Kulik. "Effects of frequent classroom testing". In: *The journal of educational research* 85.2 (1991), pp. 89–99.

[5] Joao Paulo Barros et al. "Using lab exams to ensure programming practice in an introductory programming course". In: *ACM SIGCSE Bulletin* 35.3 (2003), pp. 16–20.

[6] Klara Benda, Amy Bruckman, and Mark Guzdial. "When life and learning do not fit: Challenges of workload and communication in introductory computer science online". In: *ACM Transactions on Computing Education (TOCE)* 12.4 (2012), pp. 1–38.

[7] S. Benford, E. Burke, and E. Foxley. "Learning to construct quality software with the Ceilidh system". In: *Software Quality Journal* 2 (1993), pp. 177–197.

[8] Jeremiah Blanchard et al. "Stop Reinventing the Wheel! Promoting Community Software in Computing Education". In: *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education.* ITiCSE-WGR '22. Dublin, Ireland: Association for Computing Machinery, 2022, pp. 261–

292. ISBN: 9798400700101. DOI: 10.1145/3571785.3574129. URL: https://doi.org/10.1145/3571785.3574129.

[9] Benjamin S Bloom. "The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring". In: *Educational researcher* 13.6 (1984), pp. 4–16.

[10] Robert W Brennan and Jonathan Lesage. "Exploring the Implications of OpenAI Codex on Education for Industry 4.0". In: *Service Oriented, Holonic and Multi-Agent Manufacturing Systems for Industry of the Future: Proceedings of SOHOMA 2022*. Springer, 2023, pp. 254–266.

[11] Marc H. Brown and Robert Sedgevick. "Techniques for algorithm animation". In: *IEEE Software* 2.1 (1985), pp. 28–39.

[12] Peter Brusilovsky et al. "An integrated practice system for learning programming in Python: design and evaluation". In: *Research and Practice in Technology Enhanced Learning* 13.18 (2018), pp. 18.1–18.40. DOI: 10.1186/s41039-018-0085-9. URL: https://doi.org/10.1186/s41039-018-0085-9.

[13] Peter Brusilovsky et al. "Building an Infrastructure for Computer Science Education Research and Practice at Scale". In: *the Seventh ACM Conference on Learning @ Scale*. ACM, 2020, pp. 211–213. DOI: 10.1145/3386527.3405936. URL: https://doi.org/10.1145%2F3386527.3405936.

[14] Peter Brusilovsky et al. "Increasing Adoption of Smart Learning Content for Computer Science Education". In: *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference*. ITiCSE-WGR '14. New York, NY, USA: Association for Computing Machinery, June 2014, pp. 31–57. ISBN: 9781450334068. DOI: 10.1145/2713609.2713611. URL: https://doi.org/10.1145/2713609.2713611 (visited on 02/20/2023).

[15] Jacabo Carrasquel, Dennis R. Goldenson, and Philip L. Miller. "Competency testing in introductory computer science: the mastery examination at Carnegie-Mellon University". In: *SIGCSE '85*. Mar. 1985.

[16] Mark Chen et al. "Evaluating large language models trained on code". In: *arXiv preprint arXiv:2107.03374* (2021).

[17] Michelene TH Chi. "Translating a theory of active learning: An attempt to close the research-practice gap in education". In: *Topics in Cognitive Science* 13.3 (2021), pp. 441–463.

[18] Michelene TH Chi and Ruth Wylie. "The ICAP framework: Linking cognitive engagement to active learning outcomes". In: *Educational psychologist* 49.4 (2014), pp. 219–243.

[19] James H. Cross and T. Dean Hendrix. "JGRASP: A Lightweight IDE with Dynamic Object Viewers for CS1 and CS2". In: *SIGCSE Bull.* 38.3 (June 2006), p. 356. ISSN: 0097-8418. DOI: 10.1145/1140123.1140266. URL: https://doi.org/10.1145/1140123.1140266.

[20] Stephen H. Edwards and Krishnan Panamalai Murali. "CodeWorkout: Short Programming Exercises with Built-in Data Collection". In: ITiCSE '17. Bologna, Italy: Association for Computing Machinery, 2017, pp. 188–193. ISBN: 9781450347044. DOI: 10.1145/3059009.3059055. URL: https://doi.org/10.1145/3059009.3059055.

[21] Stephen H. Edwards and Manuel A. Perez-Quinones. "Web-CAT: Automatically Grading Programming Assignments". In: *SIGCSE Bull.* 40.3 (June 2008), p. 328. ISSN: 0097-8418. DOI: 10.1145/1597849.1384371. URL: https://doi.org/10.1145/1597849.1384371.

[22] Barbara Ericson et al. "An eBook for Teachers Learning CS Principles". In: *ACM Inroads* 6.4 (2015), pp. 84–86. DOI: 10.1145/2829976.

[23] James Finnie-Ansley et al. "The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming". In: *Australasian Computing Education Conference*. ACE '22. Virtual Event, Australia: Association for Computing Machinery, 2022, pp. 10–19. ISBN: 9781450396431. DOI: 10.1145/3511861.3511863. URL: https://doi.org/10.1145/3511861.3511863.

[24] Kathi Fisler. "The recurring rainfall problem". In: *Proceedings of the tenth annual conference on International computing education research*. 2014, pp. 35–42.

[25] Max Fowler and Craig Zilles. "Superficial Code-guise: Investigating the Impact of Surface Feature Changes on Students' Programming Question Scores". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 2021, pp. 3–9.

[26] Robert Fraser et al. "Collaboration, collusion and plagiarism in computer science coursework". In: *Informatics in Education-An International Journal* 13.2 (2014), pp. 179–195.

[27] Dan Garcia et al. "Achieving "A's for All (as Time and Interest Allow)"". In: *Proceedings of the Ninth ACM Conference on Learning@ Scale*. 2022, pp. 255–258.

[28] Matt Garish and Dave Cramer. *EPUB 3.2 Final Community Group Specification 08 May 2019*. URL: https://www.w3.org/publishing/epub3/epub-spec.html (visited on 02/28/2022).

[29] Ann Q. Gates et al. "Using Peer-Led Team Learning to build university-community college relationships". In: *2015 IEEE Frontiers in Education Conference (FIE)*. 2015, pp. 1–7. DOI: 10.1109/FIE.2015.7344094.

[30] Mark J Gierl and Thomas M Haladyna. *Automatic item generation: Theory and practice*. Routledge, 2012.

[31] David K. Gosser and Vicki Roth. "The Workshop Chemistry Project: Peer-Led Team-Learning". In: *Journal of Chemical Education*. Vol. 75. 2. 1998, p. 185. DOI: 10.1021/ed075p185. URL: https://doi.org/10.1021/ed075p185.

[32] Philip J. Guo. "Online Python Tutor: Embeddable Web-Based Program Visualization for Cs Education". In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. SIGCSE '13. Denver, Colorado, USA: Association for Computing Machinery, 2013, pp. 579–584. ISBN: 9781450318686. DOI: 10.1145/2445196.2445368. URL: https://doi.org/10.1145/2445196.2445368.

[33] Richard R Hake. "Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses". In: *American journal of Physics* 66.1 (1998), pp. 64–74.

[34] Shlomi Hod. *Parsons Puzzle Creator*. URL: https://parsons.problemsolving.io (visited on 02/28/2022).

[35] Beryl Hoffman and Barbara Ericson. "CSAwesome: A Free Curriculum and Ebook for Advanced Placement Computer Science A (CS1 in Java)". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 2021, p. 1352. ISBN: 9781450380621. URL: https://doi.org/10.1145/3408877.3432504.

[36] Cay S. Horstmann. *CodeCheck*. URL: https://codecheck.io (visited on 02/28/2022).

[37] Cay S. Horstmann. *CodeCheck Tracer*. URL: https://horstmann.com/codecheck/tracer.html (visited on 02/28/2022).

[38] Susan Horwitz et al. "Using Peer-Led Team Learning to Increase Participation and Success of under-Represented Groups in Introductory Computer Science". In: *SIGCSE Bull.* 41.1 (Mar. 2009), pp. 163–167. ISSN: 0097-8418. DOI: 10.1145/1539024.1508925. URL: https://doi.org/10.1145/1539024.1508925.

[39] Xinying Hou, Barbara Jane Ericson, and Xu Wang. "Using Adaptive Parsons Problems to Scaffold Write-Code Problems". In: *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*. 2022, pp. 15–26.

[40] David Hovemeyer and Jaime Spacco. "CloudCoder: A Web-Based Programming Exercise System". In: *J. Comput. Sci. Coll.* 28.3 (Jan. 2013), p. 30. ISSN: 1937-4771.

[41] Sarah Hug, Heather Thiry, and Phyllis Tedford. "Learning to love computer science: Peer leaders gain teaching skill, communicative ability and content knowledge in the CS classroom". In: *SIGCSE'11 - Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Jan. 2011). DOI: 10.1145/1953163.1953225.

[42] CHRISTOPHER D. HUNDHAUSEN, SARAH A. DOUGLAS, and JOHN T. STASKO. "A Meta-Study of Algorithm Visualization Effectiveness". In: *Journal of Visual Languages & Computing* 13.3 (2002), pp. 259–290. ISSN: 1045-926X. DOI: https://doi.org/10.1006/jvlc.2002.0237. URL: https://www.sciencedirect.com/science/article/pii/S1045926X02902375.

[43] Petri Ihantola and Ville Karavirta. "Open Source Widget for Parson's Puzzles". In: *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*. ITiCSE '10. Bilkent, Ankara, Turkey: Association for Computing Machinery, 2010, p. 302. ISBN: 9781605588209. DOI: 10.1145/1822090.1822178. URL: https://doi.org/10.1145/1822090.1822178.

[44] Petri Ihantola et al. "Educational Data Mining and Learning Analytics in Programming". In: *Proceedings of the 2015 ITiCSE on Working Group Reports*. ACM, July 2015. DOI: 10.1145/2858796.2858798. URL: https://doi.org/10.1145%2F2858796.2858798.

[45] Ville Karavirta and Clifford A. Shaffer. "JSAV: The JavaScript Algorithm Visualization Library". In: *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '13. Canterbury, England, UK: Association for Computing Machinery, 2013, pp. 159–164. ISBN: 9781450320788. DOI: 10.1145/2462476.2462487. URL: https://doi.org/10.1145/2462476.2462487.

[46] Jeffrey D. Karpicke and Henry L. Roediger. "The Critical Importance of Retrieval for Learning". In: *Science* 319.5865 (2008), pp. 966–968. DOI: 10.1126/science.1152408. eprint: https://www.science.org/doi/pdf/10.1126/science.1152408. URL: https://www.science.org/doi/abs/10.1126/science.1152408.

[47] Majeed Kazemitabaar et al. "Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming". In: *arXiv preprint arXiv:2302.07427* (2023).

[48] Ari Korhonen, Lauri Malmi, and Panu Silvasti. "TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises". In: *Kolin Kolistelut-Koli Calling, Oct. 3-5, 2003 in Koli Finland*. University of Joensuu and University of Helsinki, 2003, pp. 48–56.

[49] Aravind K. Krishna and Amruth N. Kumar. "A Problem Generator to Learn Expression: Evaluation in CSI, and Its Effectiveness". In: *J. Comput. Sci. Coll.* 16.4 (Apr. 2001), pp. 34–43. ISSN: 1937-4771.

[50] Chen-Lin C Kulik and James A Kulik. "Mastery testing and student learning: A meta-analysis". In: *Journal of Educational Technology Systems* 15.3 (1987), pp. 325–345.

[51] Antti-Jussi Lakanen, Vesa Lappalainen, and Ville Isomöttönen. "Revisiting rainfall to explore exam questions and performance on CS1". In: *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. 2015, pp. 40–49.

[52] Thomas Lancaster and Robert Clarke. "Contract cheating: The outsourcing of assessed student work". In: *Handbook of academic integrity* 1 (2016), pp. 639–654.

[53] Stephen MacNeil et al. "Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book". In: *arXiv preprint arXiv:2211.02265* (2022).

[54] Andrés Moreno et al. "Visualizing Programs with Jeliot 3". In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '04. Gallipoli, Italy: Association for Computing Machinery, 2004, pp. 373–376. ISBN: 1581138679. DOI: 10.1145/989863.989928. URL: https://doi.org/10.1145/989863.989928.

[55] Jason W Morphew et al. "Frequent mastery testing with second-chance exams leads to enhanced student learning in undergraduate engineering". In: *Applied Cognitive Psychology* 34.1 (2020), pp. 168–181.

[56] Thomas L. Naps. "Algorithm Visualization on the World Wide Web—the Difference Java Makes!" In: *Proceedings of the 2nd Conference on Integrating Technology into Computer Science Education*. ITiCSE '97. Uppsala, Sweden: Association for Computing Machinery, 1997, pp. 59–61. ISBN: 0897919238. DOI: 10.1145/268819.268839. URL: https://doi.org/10.1145/268819.268839.

[57] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. "Automated Assessment in Computer Science Education: A State-of-the-Art Review". In: *ACM Trans. Comput. Educ.* 22.3 (June 2022). DOI: 10.1145/3513140. URL: https://doi.org/10.1145/3513140.

[58] Miranda C Parker et al. "A Pair of ACES: An Analysis of Isomorphic Questions on an Elementary Computing Assessment". In: *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*. 2022, pp. 2–14.

[59] Nick Parlante. *CodingBat*. URL: https://codingbat.com (visited on 02/28/2022).

[60] Dale Parsons and Patricia Haden. "Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses". In: *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*. ACE '06. Hobart, Australia: Australian Computer Society, Inc., 2006, pp. 157–163. ISBN: 1920682341.

[61] M. Procopiuc, O. Procopiuc, and S. H. Rodger. "Visualization and Interaction in the Computer Science Formal Languages Course with JFLAP". In: *Proceedings of the 26th Annual Frontiers in Education - Volume 01*. FIE '96. USA: IEEE Computer Society, 1996, pp. 121–125. ISBN: 0780333489.

[62] Christopher A Rowland. "The effect of testing versus restudy on retention: a meta-analytic review of the testing effect." In: *Psychological bulletin* 140.6 (2014), p. 1432.

[63] Sami Sarsa et al. "Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models". In: *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*. 2022, pp. 27–43.

[64] Sami Sarsa et al. "Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models". In: *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1*. ICER '22. Lugano and Virtual Event, Switzerland: Association for Computing Machinery, 2022, pp. 27–43. ISBN: 9781450391948. DOI: 10.1145/3501385.3543957. URL: https://doi.org/10.1145/3501385.3543957.

[65] Otto Seppälä et al. "Do We Know How Difficult the Rainfall Problem Is?" In: *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. Koli Calling '15. Koli, Finland: Association for Computing Machinery, 2015, pp. 87–96. ISBN: 9781450340205. DOI: 10.1145/2828959.2828963. URL: https://doi.org/10.1145/2828959.2828963.

[66] Christian Servin and Nadia Karichev. "Computer Science Fundamentals Open Educational Resource: A Video-Based Practice to Learning Programming". In: *J. Comput. Sci. Coll.* 37.7 (Apr. 2022), pp. 55–61. ISSN: 1937-4771.

[67] Christian Servin, Myshie Pagel, and Ernest R. Weeb. "An Authentic Peer-Led Team Learning Program for Community Colleges: A Recruitment, Retention, and Completion Instrument for Face-to-Face and Online Modality". In: *Proceedings of the 54th ACM Technical Symposium on Computing Science Education (Toronto, ON, Canada) (SIGCSE '23)*. New York, NY, USA: Association for Computing Machinery, 2023. DOI: https://doi.org/10.1145/3545945.3569851.

[68] Clifford A. Shaffer et al. "Building an Online Educational Community for Algorithm Visualization". In: *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. SIGCSE '10. Milwaukee, Wisconsin, USA: Association for Computing Machinery, 2010, pp. 475–476. ISBN: 9781450300063. DOI: 10.1145/1734263.1734421. URL: https://doi.org/10.1145/1734263.1734421.

[69] Clifford A. Shaffer et al. "OpenDSA: Beginning a Community Active-EBook Project". In: *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*. Koli Calling '11. Koli, Finland: Association for Computing Machinery, 2011, pp. 112–117. ISBN: 9781450310529. DOI: 10.1145/2094131.2094154. URL: https://doi.org/10.1145/2094131.2094154.

[70] Aaron J. Smith et al. "My Digital Hand: A Tool for Scaling Up One-to-One Peer Teaching in Support of Computer Science Learning". In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '17. Seattle, Washington, USA: Association for Computing Machinery, 2017, pp. 549–554. ISBN: 9781450346986. DOI: 10.1145/3017680.3017800. URL: https://doi.org/10.1145/3017680.3017800.

[71] David H Smith IV et al. "Investigating the Effects of Testing Frequency on Programming Performance and Students' Behavior". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 2023, pp. 757–763.

[72] Elliot Soloway. "Learning to program= learning to construct mechanisms and explanations". In: *Communications of the ACM* 29.9 (1986), pp. 850–858.

[73] Elliot Soloway, Jeffrey Bonar, and Kate Ehrlich. "Cognitive strategies and looping constructs: An empirical study". In: *Communications of the ACM* 26.11 (1983), pp. 853–860.

[74] Matthias Stallmann et al. *Galant: A Graph Algorithm Animation Tool*. Tech. rep. TR-2016-08. North Carolina State University, Aug. 2016.

[75] David Stroup et al. *EPUB 3.2 Final Community Group Specification 08 May 2019*. URL: https://idpf.org/epub/profiles/edu/spec/ (visited on 02/28/2022).

[76] David Scot Taylor et al. "Predictive vs. Passive Animation Learning Tools". In: *SIGCSE Bull.* 41.1 (Mar. 2009), pp. 494–498. ISSN: 0097-8418. DOI: 10.1145/1539024.1509038. URL: https://doi.org/10.1145/1539024.1509038.

[77] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. "Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models". In: *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. 2022, pp. 1–7.

[78] Jeroen JG Van Merrienboer and John Sweller. "Cognitive load theory and complex learning: Recent developments and future directions". In: *Educational psychology review* 17.2 (2005), pp. 147–177.

[79] Jeroen JG Van Merriënboer and Marcel BM De Croock. "Strategies for computer-based programming instruction: Program completion vs. program generation". In: *Journal of Educational Computing Research* 8.3 (1992), pp. 365–394.

[80] Jeroen JG Van Merriënboer, Paul A Kirschner, and Liesbeth Kester. "Taking the load off a learner's mind: Instructional design for complex learning". In: *Educational psychologist* 38.1 (2003), pp. 5–13.

[81] Mauricio Verano Merino and Koen van Wijk. "Workbench for Creating Block-Based Environments". In: *Proceedings of the 15th ACM SIGPLAN International Conference on Software Language Engineering*. SLE 2022. Auckland, New Zealand: Association for Computing Machinery, 2022, pp. 61–73. ISBN: 9781450399197. DOI: 10.1145/3567512.3567518. URL: https://doi.org/10.1145/3567512.3567518.

[82] Benedikt Wisniewski, Klaus Zierer, and John Hattie. "The Power of Feedback Revisited: A Meta-Analysis of Educational Feedback Research". In: *Frontiers in Psychology* 10 (2020). ISSN: 1664-1078. DOI: 10.3389/fpsyg.2019.03087. URL: https://www.frontiersin.org/articles/10.3389/fpsyg.2019.03087.

[83] Gregory S. Wolffe et al. "Teaching Computer Organization/Architecture with Limited Resources Using Simulators". In: *SIGCSE Bull.* 34.1 (Feb. 2002), pp. 176–180. ISSN: 0097-8418. DOI: 10.1145/563517.563408. URL: https://doi.org/10.1145/563517.563408.

[84] Craig Zilles et al. "Making testing less trying: Lessons learned from operating a Computer-Based Testing Facility". In: *2018 IEEE Frontiers in Education (FIE) Conference*. San Jose, California, 2018.

[85] Craig B Zilles et al. "Every University Should Have a Computer-Based Testing Facility." In: *CSEDU (1)*. 2019, pp. 414–420.