Stop Reinventing the Wheel! Promoting Community Software in Computing Education

Jeremiah Blanchard, John R. Hott, Vincent Berry, Rebecca Carroll, Bob Edmison, Richard Glassey, Oscar Karnalim, Brian Plancher, Seán Russell

▶ **To cite this version:**

Jeremiah Blanchard, John R. Hott, Vincent Berry, Rebecca Carroll, Bob Edmison, et al.. Stop Reinventing the Wheel! Promoting Community Software in Computing Education. ITiCSE-WG 2022 - 27th ACM Conference on Innovation and Technology in Computer Science Education, Jul 2022, Dublin, Ireland. pp.261-292, 10.1145/3571785.3574129 . lirmm-04025606

HAL Id: lirmm-04025606

https://hal-lirmm.ccsd.cnrs.fr/lirmm-04025606

Submitted on 12 Mar 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Stop Reinventing the Wheel!

Promoting Community Software in Computing Education

Jeremiah Blanchard*
Dept. of Engineering Education
University of Florida
Gainesville, Florida, USA
jjb@eng.ufl.edu

John R. Hott*
Dept. of Computer Science
University of Virginia
Charlottesville, Virginia, USA
jrhott@virginia.edu

Vincent Berry
Polytech Engineering School
LIRMM - Univ Montpellier, CNRS
Montpellier, France
vincent.berry@umontpellier.fr

Rebecca Carroll
Dept. of Computer Science
Full Sail University
Winter Park, Florida, USA
rebeccac@fullsail.edu

Bob Edmison
Dept. of Computer Science
Virginia Tech
Blacksburg, Virginia, USA
bedmison@vt.edu

Richard Glassey
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
glassey@kth.se

Oscar Karnalim†
School of Information and Physical Sciences
University of Newcastle
Callaghan, NSW, Australia
oscar.karnalim@uon.edu.au

Brian Plancher
Dept. of Computer Science
Barnard College, Columbia University
New York City, NY, USA
bplancher@barnard.edu

Seán Russell
School of Computer Science
University College Dublin
Dublin, Ireland
sean.russell@ucd.ie

ABSTRACT

Historically, computing instructors and researchers have developed a wide variety of tools to support teaching and educational research, including exam and code testing suites and data collection solutions. However, these tools often find limited adoption beyond their creators. As a result, it is common for many of the same functionalities to be re-implemented by different instructional groups within the Computing Education community. We hypothesise that this is due in part to discoverability, availability, and adaptability challenges. Further, instructors often face institutional barriers to deployment, which can include hesitance of institutions to rely on community developed solutions that often lack a centralised authority and may be community or individually maintained.

To this end, our working group explored what solutions are currently available, what instructors needed, and the reasons behind the above-mentioned phenomenon. To do so, we reviewed existing literature and surveyed the community to identify the tools that have been developed by the community; the solutions that are currently available and in use by instructors; what features are needed moving forward for classroom and research use; what support for extensions is needed to support further Computing Education research; and what institutional challenges instructors and researchers are currently facing or have faced in using community software solutions. Finally, the working group identified factors that limited adoption of solutions. This work proposes ways to integrate and improve the availability, discoverability, and dissemination of existing community projects, as well as ways to manage and overcome institutional challenges.

CCS CONCEPTS

• **Social and professional topics** → **Student assessment**; • **Software and its engineering** → *Software libraries and repositories*; **Open source model**.

KEYWORDS

educational tools, computing education, computing education research, open source software, community software

ACM Reference Format:
Jeremiah Blanchard, John R. Hott, Vincent Berry, Rebecca Carroll, Bob Edmison, Richard Glassey, Oscar Karnalim, Brian Plancher, and Seán Russell. 2022. Stop Reinventing the Wheel!: Promoting Community Software in Computing Education. In *2022 ITiCSE Working Group Reports (ITiCSE-WGR '22), July 8–13, 2022, Dublin, Ireland*. ACM, New York, NY, USA, 32 pages. https://doi.org/10.1145/3571785.3574129

1 INTRODUCTION

Practically from the birth of computing disciplines, instructors and researchers have built, used, and published software for community use to assist one another with student assessment and research in computing coursework [51, 79, 81]. At first, source was shared through journals and books [51]. As the Computing Education Research (CER) community grew, so too did the avenues for distribution, with the rise of networking and eventually ubiquitous

*Working group leader
†Also with Maranatha Christian University.


This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike International 4.0 License.

ITiCSE-WGR '22, July 8–13, 2022, Dublin, Ireland

© 2022 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0010-1/22/07.
https://doi.org/10.1145/3571785.3574129

access and the unprecedented capacity for sharing source through the Internet. The ease of publication and lack of centralised listing and filtering mechanisms led to difficulty in discovering tools, a problem that persists to this day [119].

Even when tools are discovered, instructors and researchers may find that institutions are hesitant to endorse the use of solutions that do not have a centralised authority, as is common in community-driven projects. By comparison, centralised solutions offered by large technology firms are more easily discovered through marketing and name-recognition. These firms also offer a clear, recognised authority that can be held accountable, assuaging many institutional concerns during risk assessment processes. These solutions almost always come at a cost, though, whether directly via fees, or indirectly via a surrender of some degree of privacy, or both [134].

Some instructors and researchers find the costs associated with such platforms burdensome and/or objectionable, particularly when fees are ultimately passed on to students, and they may be an insurmountable hurdle to students and faculty from socio-economically disadvantaged populations or regions. As a result, instructors and researchers often resort to developing in-house solutions, essentially reinventing the wheel by generation and institution [51, 79, 81, 82]. However, there are some community-driven, non-profit solutions that have developed within the CER community and that have managed to achieve success, both in terms of discoverability and institutional deployment [43, 46]. Likewise, many Open Source Software (OSS) projects have found success and wide utilisation. There are lessons that the CER community can learn from these examples that could be applied more broadly to help instructors and researchers build on existing tools and frameworks.

This working group aimed to address these research questions:

- RQ1 What existing tools are available for computing educators?
- RQ2 What challenges need to be addressed to support better software use, software development, availability, and discoverability of existing CER community projects?
- RQ3 What suggested directions could address the challenges to using, developing, and discovering CER community projects?

To answer these questions, the working group explored what instructional and research solutions are currently available, those that are in demand, reasons for the challenges to wider adoption and collaboration, and potential avenues to support community software development and adoption in the future by identifying and building on successes in the CER and OSS communities. Specifically, the working group:

(a) reviewed literature, identifying existing community software in computing education / CER and its functionality, barriers to discoverability and adoption of such software, and successful models to support community development;
(b) deployed an international survey to educators and education researchers soliciting feedback on experiences, challenges, and initiatives related to community software development and use in classrooms and research, as well as the current needs of the computing education and CER communities;
(c) analysed results, in concert with reviewed literature, to identify ways to integrate and improve the availability and discoverability of existing CER community projects, as well as manage and overcome institutional challenges; and
(d) developed suggested directions for a combined effort of the CER community to identify and disseminate new solutions.

2 SIMILAR STUDIES

Prior literature reviews and surveys on computing education tools over the past ten years have focused primarily on highlighting tools for narrow domains such as visualisation systems [169] and plagiarism detectors [129]. Of the few works we discovered that attempted to describe tools for computing education more broadly, we found them to be most useful as an illustrative example of tool categories instead of a comprehensive list of available tools.

2.1 Domain-specific Reviews

Program visualisation systems were covered in 2013 by Sorva et al. [169]. They focused on systems or tools intended for teaching beginners about the runtime behaviour of computer programs. Interestingly, they observed that most of these systems appear to have been short-lived research prototypes, and that only a small handful of them have been used outside their site of creation. They also concluded that, at the time, "open-source communities working on these systems are almost nonexistent" [169]. As potential remedies, they cite forum launches and financial support to lower collaboration thresholds.

Similarly, Sorva et al. [170] reviewed and provided a large list of resources and solutions for generic programming visualisation systems, but their work narrowly focused on introductory CS courses.

A systematic literature review of plagiarism detectors was published in 2019 by Novak et al. [129]. It covered not only available tools but also other aspects of plagiarism detection including common code disguises, similarity algorithms, and evaluation instruments. According to the review, JPlag [142], MOSS [158], and Sherlock from University of Warwick [87] are the three most commonly mentioned in the literature. The authors also stressed the importance of evaluation instruments: not many evaluation metrics and data sets are available for comparison.

A review of automated assessment tools for programming assignments was recently conducted by Paiva et al. [132]. This review focused on the supported types of exercises, security measures adopted, testing techniques used, type of feedback produced, and the information offered to the teacher to optimise learning. It observed that papers published over the last 5 to 6 years mainly focus on tool development, static analysis techniques and feedback [132, Fig. 3] and concluded that there is strong need for empirical studies examining the role and value of automated assessment techniques.

2.2 Comprehensive Reviews and Surveys

The SIGCSE recurring panel entitled "Technology We Can't Live Without," which highlights computing education technology and tools, has had seven near-annual issues since 2015 [55, 56, 58–61]. These are informal surveys of panel members' software and hardware tools for their courses and serve as quick touch points, since they are informal panels of small groups of instructors and are not designed to be thorough comprehensive surveys of tools in computing education. However, they do list tools including: email notification aids (i.e., Boomerang for Gmail); classroom engagement

sites (i.e., Kahoot! and EDPuzzle Socrative); classroom management (i.e., Github and Dropbox); and many others.

Furthermore, Brusilovsky et al. [26] also listed tools for: program Visualisation (Jeliot, jGRASP, BlueJ); algorithm visualisation (Animal, jHAVE); coding tools (CodeLab, CloudCoder, CodeAcademy); algorithm and program simulation tools (TRAKLA2, JSAV, UUhistle); problem-solving support tools (Problets, js-parsons, WadeIn, ELM-ART, JavaGuide); classroom management (Github, Dropbox, Gradescope, ZipGrade, Google Drive); and social navigation systems and others (Educo, Progressor). However, as the focus of the work was on the need for smart learning content in computing education, the tools and their features are not described in detail.

3 METHODS

Our inquiry, guided by our research questions, was designed to expand on prior work by producing a comprehensive review of current tools, listing challenges to the development and adoption of CER community projects, and providing suggested directions to address those challenges. To do so, we performed a comprehensive literature review and conducted a community survey. Results from both efforts form the basis of our list of existing tools available for computing educators (RQ1), while the survey results highlight challenges and suggested solutions (RQ2 and RQ3).

3.1 Community Survey

A survey was developed for members of the computing education research community to address the following topics:

(1) The most important tools participants use and their satisfaction with those tools;
(2) Tools that have been developed by participants;
(3) Features and/or tools participants would like to see;
(4) Challenges in using open source and commercial software;
(5) Barriers to community software deployment, development, and/or maintenance.

We first asked respondents about up to five tools that they find most important in their work. For each tool, we asked the respondent to provide details about the tool, including their satisfaction with it (Table 1). They rated satisfaction on a seven-point Likert scale from "Extremely Dissatisfied" to "Extremely Satisfied". We also asked respondents to identify their favourite and least favourite tools, as well as what tools or features they would like to see in the future (Table 3). We then asked respondents if they participated in the development of any tools, and asked them for information of up to five of these tools (Table 2). We further asked them to identify the top three challenges they encountered when using open source and commercial software, respectively, and when developing tools. We provided space for respondents to leave additional information (Table 3). Finally, the survey concluded with demographic questions about the respondent's institution, career, and course-load (Table 8). Additionally, all questions were voluntary; respondents could skip any question they did not want to answer.

The survey was distributed to the SIGCSE mailing list, Teaching Track Faculty discussion on Piazza, ITiCSE 2022 Moodle site, and to the members' colleagues. Participation in the survey was voluntary and the data was collected anonymously. However, it should be mentioned that since respondents were asked to include information on tools they developed, this impacted the anonymity if their responses.

The resulting anonymous survey data was then parsed by working group members and analysed to uncover trends. Before analysis, we restructured data for analysis. For example, in preparing to analyse the question "What is your least favorite tool?" and "Why?", the names of the tools themselves were extracted from the text and curated (unifying the cases across answers and correcting typos in names and other issues). Once the data were prepared, the results were processed to generate charts and tables. Key results are included in Sections 5 and 7. We also analysed various additional cuts of the data to see if any of the demographic data affected the overall results and found these complementary analyses to be inconclusive. This indicated to us that the trends we uncovered were broadly applicable, holding across institution type, class size, TA support, years of experience teaching, percentage of time dedicated to teaching, and other demographic metrics. Finally, we also added to our main lists of tools any tool that was reported by no less than 4 of the respondents (10% of the survey group) and not already uncovered by the literature search, as this indicated that the tool was likely used by a relatively large portion of the community.

3.2 Literature Review

We paired our community survey with a comprehensive literature review to identify available tools and summarise historical challenges to adoption, as well as successful models of dissemination for community-developed and maintained software. The literature review was compiled in a quasi-systematic manner, combining the benefits of systematic review and traditional review. Our methodology is therefore intended to be replicable while also allowing some flexibility in the process. This was critical due to the challenges of discoverability, which necessitated casting a very wide net in the search process (see Section 7). Additionally, the review was performed from scratch instead of relying on existing literature reviews for consistency across topics, since prior works tended to focus narrowly on a chosen topic, have differing methodologies, and were published in different years. Our review focused on summarising recent trends of available tools in the last 10 years.

Our literature review methodology consisted of six subsequent steps. First, a number of computing education and research topics that involve the use of tools were defined based on discussions amongst the authors of this working group report (see Table 5). While the topics do not guarantee identification of all tools, they are likely to capture the most common and/or popular tools.

Search queries for each of the tool categories were then defined. Due to differences in nomenclature for each of the tool categories, we specifically tailored dedicated queries per category, as seen in Table 6. In most cases, multiple simple queries were attempted; however, for autograders and polling, one query was constructed to catch multiple related search terms. Some required a broadening or relabelling of the topic area to include relevant tools. For example, plagiarism detectors are commonly used to help instructors identifying plagiarism, an act of reusing either colleagues' code or one's own code without proper acknowledgement [36][1]. Given that

[1]Such an act is often referred to as collusion when authors of the original work are aware about the misconduct but let that happen [52].

Table 1: Important Tools Questions

Q#	Question	Type
Q1	Link to the tool's website (if applicable)	Text Entry
Q2	How satisfied are you with...?	Likert
Q3	Please share any additional information on your satisfaction with...	Text Entry
Q4	Tool Category [See Table 5]	Check boxes
Q5	Does the tool provide any prepacked related course content?	Yes/No
Q6	If so, to what extent are you using that prepacked course content?	Text Entry
Q7	How did you find out about...? [See Table 12]	Checkboxes
Q8	Is ... open source?	Yes/No
Q9	If so, who maintains...?	Text Entry

Table 2: Tool Development Questions

Q#	Question	Type
Q13	What is the tool?	Text Entry
Q14	What problem does it solve? [See Table 5]	Check boxes
Q15	How widely is it being used?	Text Entry
Q16	Who maintains the tool?	Text Entry
Q17	How was the tool released?	Text Entry
Q18	Is it open source?	Yes/No
Q19	Does it integrate with any other tools? Why did you integrate it with those tools?	Text Entry
Q20	Is there anything else you'd like to tell us about this tool?	Text Entry

Table 3: Additional Survey Questions

Q#	Question	Type
Q10	What is your favorite tool that you can't live without? Why?	Text Entry
Q11	What is your least favorite tool that you'd love to replace? Why?	Text Entry
Q12	What new tools or features of tools do you wish existed? Why?	Text Entry
Q21	What are your top 3 challenges to using a new open source tool? [See Table 4]	Text Entry
Q22	What are your top 3 challenges to using a new commercial tool? [See Table 4]	Text Entry
Q23	What are your top 3 challenges to using a developing a new tool? [See Table 4]	Text Entry
Q24	Is there anything else you would like to share about barriers to tool development or use?	Text Entry
Q25	Is there anything else you would like to tell us about tool development or use?	Text Entry

Table 4: Challenges with Developing or Using Tools

C#	Challenge
C1	Concerns of data security
C2	Concerns over reliability
C3	Concerns over accountability
C4	Concerns about a centralized authority
C5	Concerns over price or cost sharing
C6	Concerns over content IP (intellectual property)
C7	Concerns over development time
C8	Concerns over integration / learning time
C9	Concerns over maintenance time / long-term support
C10	Compatibility with LMS in use

Table 5: Computing Education Topics

T#	Topic	Section
T1	Autograders	6.1
T2	Plagiarism (similarity) detectors	6.2
T3	Code sharing and execution environments	6.3
T4	Class management / monitoring	6.5
T5	Open textbooks	6.6
T6	Polling	6.7
T7	Visualisations	6.4

Table 6: Computing education topics and their queries for the comprehensive literature review

ID	Topic	Queries
T1	Autograders	[All: autograd*] OR [All: "auto grader"] OR [All: "auto grading"] OR [All: "auto grade"]
T2	Plagiarism (similarity) detectors	"code plagiarism", "code similarity", "text plagiarism", "text similarity"
T3	Code sharing and execution environments	"IDE Plugin", "computer-based testing", "IDE", "block-based programming", "block programming", "dual-modality programming", "multimodal programming"
T4	Class management / monitoring	"classroom management" AND "tool", "learning management system", "e-learning AND computer science", "course management"
T5	Open textbooks	"open textbook", "textbook" +open, "etextbook", "e-textbook"
T6	Polling	ACM: [[All: poll] OR [All: polling]] AND [Abstract: tool] AND [Abstract: education] IEEE: ("Full Text & Metadata":"poll" OR "Full Text & Metadata":"polling") AND ("IEEE Terms":Education) AND ("Abstract":tool)
T7	Visualisations	"Algorithm Visualisation", "Algorithm Visualization", "Interactive content", "smart learning content", "interactive learning activities", "interactive learning activity", "Program visualisation"

plagiarism detection is often based on similarity across submissions and assessments in computing education expect code in addition to regular text submissions, four queries were used: "code plagiarism", "code similarity", "text plagiarism", and "text similarity". Likewise, quizzing and polling are terms that are used broadly within and without the education domain. We therefore focused our search queries on "poll" and "polling" terms due to the high percentage of false positives brought by queries on "quiz" or "quizzing."

The second step of the literature review was to deploy the queries on both IEEE Xplore and the ACM Digital Library. These queries were executed from 24 May to 10 September 2022. The latter database is known for its specific focus on the computing community while the former is for the wider community of engineers. We also considered other potentially relevant databases (e.g., Scopus and Springer). However, these databases cover a much wider scope of work and fields outside of computing; as such, they yield a low signal-to-noise ratio (e.g., many more irrelevant results than relevant ones[2]) or a large number of false-positives (i.e., many retrieved but irrelevant results [38]), limiting their value to our search. We are additionally aware of the existence of publication venues that are indexed by neither ACM Digital Library nor IEEE Xplore (e.g., Taylor & Francis' Computer Science Education and Elsevier's Computers & Education). However, we did not include those as this would entail consideration of other publication venues while defining minimum threshold for "good" publication venues can be subjective and debatable. To ensure articles focused on recent work relevant to the CER community, we limited our search to the last ten years (April 2012 to April 2022).

Third, per query and database, the results were checked sequentially for relevance. Once six or more irrelevant papers in a row were visited, we terminated the search. We identified this threshold through a trial-and-error process. For class management (T4) and polling (T6), the threshold was increased to ten irrelevant papers due to a higher rate of irrelevant papers. Relevancy was defined based on title, abstract, and keywords. Papers were only included if they discussed tools related to the topic and were not already added to the literature review article set.

Fourth, for all resulting papers, we excluded papers presenting opinion (as these are by definition subjective), algorithmic or technical methods (as they represent ongoing experiments), and inaccessible tools – i.e., those without working links, without explicit information about how to access the tools, and that could not be found via simple Google search. We also excluded papers that required author contact to access, as we classified such tools as inaccessible. Finally, we excluded papers that reviewers could not comprehend (i.e., written in a language other than English or lacking clarity) and papers that were duplicates of those already identified in our search, as some papers were included in results from multiple queries or were indexed by both ACM Digital Library and IEEE Xplore.

Fifth, tools presented in the resulting papers were listed and compiled. Other tools mentioned in these papers were also included if they were commonly mentioned or deemed important for research development of the topic. As mentioned earlier, we also added in additional tools that were reported by ≥10% of survey respondents.

Sixth, characteristics of the resulting tools were analysed and summarised based on their website and/or relevant publication(s). We also tried some tools to gain better understanding of them.

3.3 Other Known Tools

Some tools that do not appear in the literature review nevertheless are well known and used in Computing Education. While the reasons vary and are intended to be addressed in part by this work, they include commercial tools and those developed by large foundations that are advertised rather than published in peer-reviewed venues and works. Despite their lack of appearance in the scientific literature and their relatively low response rate in the community survey, the use of these tools plays an important role in Computing Education and Computing Education Research, so we include them in the appendix to provide a more complete picture.

[2]https://www.nngroup.com/articles/signal-noise-ratio/

4 STATISTICAL FINDINGS

4.1 Community Survey

Our community survey received 43 responses, with 39 voluntarily providing their institution names. Two of the institutions (5.2%) offer secondary education; sixteen offer bachelor programs (41%); five offer master programs (12.8%); and another sixteen (41%) offer PhD programs. Most of the respondents also provide their country information. A majority of them are from US (24) while others are from France (4), Indonesia (3), Brazil (2), Australia (1), Canada (1), India (1), and Switzerland (1). A discussion on possible reasons for the low number of respondents is included in section 7.2.

4.2 Literature Review

The number of papers and tools discovered at each stage of our literature review are shown in Table 7, and we summarised these results in Figure 1. The "search result" refers to papers retrieved from the ACM Digital Library and IEEE Xplore databases; the full queries used can be found in Table 6. *A priori* (initially) relevant papers were retrieved papers that we considered relevant based on their title, abstract, and keywords, so long as they were found before the search is terminated. As mentioned in Section 3.2, papers with inaccessible tools were excluded. These are papers indicating tools with no explicit access information and that we could not elucidate through simple web search queries. Some papers also presented methods for doing a particular task, which were excluded as well. Papers were also excluded if they were primarily opinion pieces, could not be comprehended due to grammatical issues, or reported tools whose instructions were not written in English. Additional tools were also added where appropriate from the paper references.

Our queries retrieved 23,693 search results of which 2.5% (584) were considered initially relevant based on their title, abstract, and keywords before the check reached six or ten consecutive irrelevant results. Around three fourths of them (72.8% or 425 papers) were excluded, mostly because they either described inaccessible tools (131) or only presented methods (174). Nineteen tools were added to our list as they were merely mentioned in the *a priori* (initially) relevant papers but appeared to be relevant to development of the topic. In total, we reported on 140 tools from our literature review (see Table 7 for their distribution across considered topics).

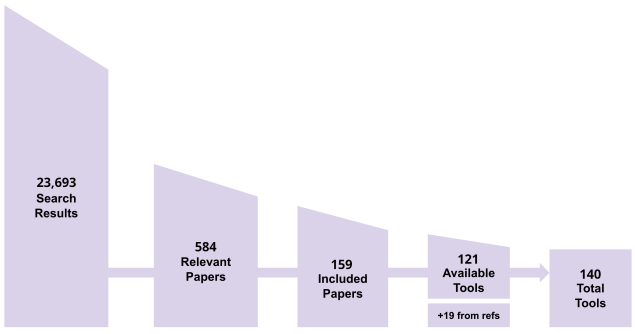

Figure 1: 23,693 initial search results led to only 140 tools.

5 PERCEPTIONS OF TOOLS

Survey respondents completed 159 tool reports covering 103 distinct tools (as some tool were reported by several respondents). We present below the findings of these survey questions regarding CER community members' perceptions of existing tools, work that is needed, and the challenges in developing and deploying tools in computing education.

5.1 Satisfaction with Existing Tools

The results from our survey show that the community on the whole has a relatively favourable opinion of the wide variety of tools they use and diverse viewpoints on how they would like to improve them. That said, on average the least favourable class of tools were Learning Management Systems (LMS), which we speculate may be caused by the fact that the adoption of an LMS is typically decided at the institutional level and not by each individual course instructor, meaning that instructors often do not have a choice regarding which LMS they can use.

When asked about their "favorite" tool, the responses were quite diverse, with a wide array of tools mentioned and very few overlapping responses. Of the 31 responses, 3 (9.7%) mentioned Gradescope; 2 each (6.5% each) mentioned Beamer, Python, and WebCAT. All other responses were unique. That said, a common theme emerged throughout the responses as most of the favourite tools were either autograders or coding environments designed to better enable students to debug their own code and to avoid IT issues with environment setup and provisioning.

When asked about their "least favorite" tool, the responses were even more diverse with only unique responses except for one: Canvas. In fact, of the 26 responses to this question, 30.8% (n=8) reported that Canvas was their least favourite tool, citing various issues with different parts of the tool, with one respondent noting that it "does everything sort of okay, but nothing particularly well."

When asked about their satisfaction with commonly used tools, respondents were generally positive, with only 5 reports (3.1%) of dissatisfaction with a tool. Of the tools that at least 10% of respondents reported, the lowest average satisfaction was reported for Canvas and MOSS (both between somewhat satisfied and satisfied), matching the fact that Canvas was the most common "least favorite" tool. Similarly, the tools with the highest satisfaction scores were Zoom, Gradescope, and Python (all between satisfied and very satisfied), with Gradescope appearing from the "favorite" tool list.

Finally, a total of 23 respondents indicated tools or features they would like to see in the future. Of these, 17.4% ($n = 4$) indicated they would like to see improved plagiarism detection tools, while 13.0% ($n = 3$) hoped for improved autograder options, and 13.0% ($n = 3$) wanted better integration and features in learning management systems. Another 8.7% ($n = 2$) wanted better code execution environments, and 8.7% ($n = 2$) wanted improved tools for code sharing. All other responses were unique to individual users.

5.2 Challenges in Software Use & Development

We asked survey respondents to select their top three challenges with using open source tools, using commercial tools, and developing their own tools. 38 respondents completed the sections on using open source and commercial tools, while 37 respondents completed

Table 7: Statistics of papers and tools from our comprehensive literature review with T1-T7 topics taken from Table 5

Metrics	T1	T2	T3	T4	T5	T6	T7	Total
Search result	680	1818	14464	4603	638	138	1352	23693
A priori (initially) relevant papers	74	80	74	187	31	48	90	584
Excluded papers								
Unavailable tools	25	19	13	25	7	13	29	131
Only presenting methods	28	50	0	65	10	5	16	174
Other reasons	0	7	41	45	5	10	12	120
Additional tools from references of *a priori* relevant papers	0	18	0	1	0	0	0	19
Reported tools	21	22	20	21	3	20	33	140

the section on developing tools. A list of possible challenge options are detailed in Table 4. Overall we found that the overwhelming challenge with commercial tools was cost, while for open source tool use and tool development it was the time needed to develop, integrate, and maintain a reliable product.

5.2.1 Using Open Source Software. As shown in Figure 2, we found that the biggest concerns with the use of open source tools were maintenance (n = 27, 71.1%); integration / learning time (n = 24, 63.2%); and reliability (n = 20, 52.6%). This indicated that while open source tools may provide a great solution, educators may be wary of relying on a tool and spending time to learn it when it may not continue to be actively developed and maintained.

5.2.2 Using Commercial Software. We also found that the overwhelmingly biggest concern with the use of commercial software is the price / cost (n = 29, 76.3%) (Figure 2). Notably, educators were more worried about data security (n = 15, 39.5%) for commercial tools than for open source tools (n = 11, 26.8%), even though commercial companies, in theory, have increased resources to spend on such efforts. It's worth noting that commercial options tend to have higher visibility and are therefore more likely to be targeted than their open-source counterparts. As with open source tools, integration / learning time still remains a large concern (n = 14, 36.8%).

5.2.3 Developing Custom Software. As shown in Figure 3, survey respondents are, for the most part, concerned with the same topics with tool development as with open source software integration. The biggest change is that there is a much smaller focus on integration / learning time (n = 10, 24.4%) and a much larger focus on development time (n = 28, 75.7%), which follows from the need to develop instead of learn a tool. Following that, we again find maintenance (n = 26, 70.3%) and reliability (n = 14, 37.8%) as the leading concerns.

5.3 Accessibility and Discoverability

We also analysed the survey results to identify how the respondents reported discovering each tool (Figure 4). The most common way respondents reported discovering tools was through colleagues (n = 88, 55.3%), with most of those colleagues being at the respondent's home institution (n = 72, 45.3%). Another 34 (21.4%) reports indicated that the tool was found via an Internet search; 11.9% (n = 19) were installed as part of the institution's LMS, and 10.7% (n = 17) were inherited with a course's materials. Another

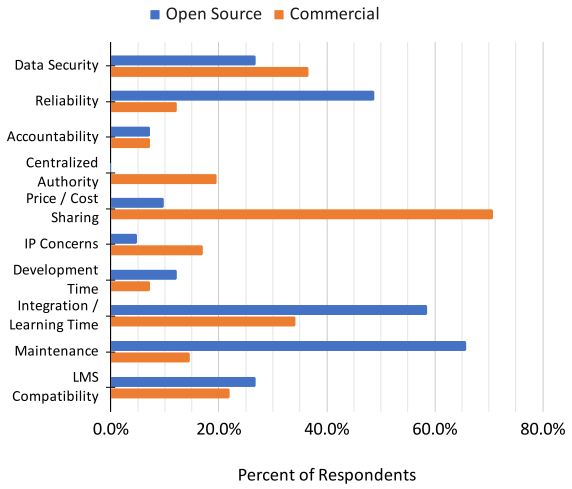

Figure 2: Respondents reported challenges integrating both open source and commercial software.

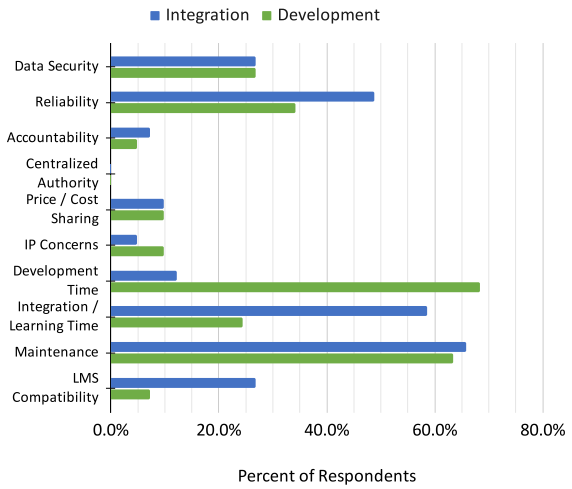

Figure 3: Respondents reported challenges in integrating open source tools and developing new tools.

16 (10.1%) were created by the respondents themselves. Critical to this study, only 7.5% (n = 12) of tools were found via papers (2.5%, n = 4) and conferences (5.0%, n = 8), and no other community mechanisms (e.g., community publications or websites) were mentioned in the free responses of completed surveys. For each tool report, respondents could indicate more than one method of discovery, so percentages summed to more than 100%.

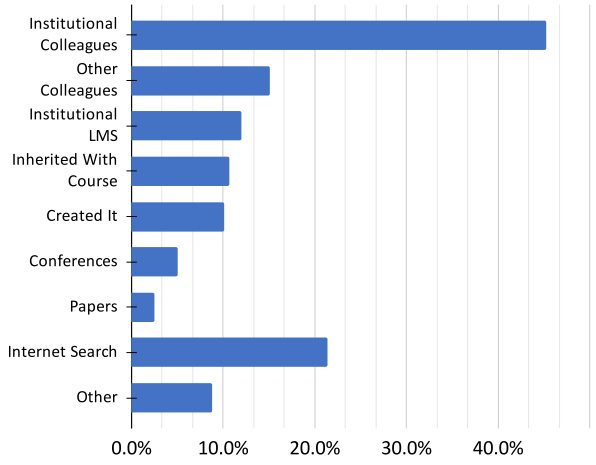

Figure 4: Respondents reported how they discovered the tools they use, noting that in most cases it was through colleagues.

5.4 Most Important Tools: Proprietary vs. Open Source

A total of 39 respondents reported what they considered to be their "most important tools." The most commonly reported tools were a mix of open source and proprietary solutions: Canvas (n = 14, 35.9% of respondents), open source; Moodle (n = 6, 15.4%), open source; Gradescope (n = 6, 15.4%), proprietary; Repl.it (n = 4, 10.3%), proprietary; Python (n = 4, 10.3%), open source; MOSS (n = 4, 10.3%), proprietary; and Zoom (n = 4, 10.3%), proprietary. All other tools were mentioned by less than 10% of respondents.

While the tools included a mix of open source and proprietary projects, it is notable that every project in this set has a centralised corporation or foundation authority (and all are commercial except for the Python Software Foundation). Of the tools with at least two respondents, 20 of 25 have such a central authority.

6 DISCOVERED TOOLS

In this section, we summarize the discovered tools from the literature review and community survey in each of the seven topic areas (Table 5). As mentioned earlier, for tools identified in the survey to be included in this list, the tool must have been mentioned by at least 4 respondents, or 10% of the overall responses. As such, there are many well-known tools that did not meet this criteria, including Visual Studio Code, Eclipse, and IntelliJ. It may be that respondents viewed these tools as so ubiquitous as to be universally known to the community, and thus no mention of them was needed, however they were also not discovered in the literature review.

A full list of discovered tools and links, including those from the community survey that did not meet inclusion criteria (Table 19), can be found in the appendix. To improve the discoverability of these tools, and to enable them to be updated in the future, we have also posted our full list to GitHub at https://csed-tools.github.io/.

6.1 Autograders

Autograders have been used in Computing Education for decades [76] and are well-used today [132]. Due to the contexts in which the autograders were created, it is worth noting that an autograder may be only a portion of a larger tool; that is, there is significant overlap among our tool categories. Many of the autograders discovered were either embedded as part of a larger learning system, such as those found in EarSketch [168] and OpenDSA [122], or built for a specific use case, such as Reveal [88] and TermAdventure [174]. The more popular autograders according to survey results, Gradescope and Web-CAT [45], are more generalised to support additional use cases and programming languages.

In the context of our literature review, we identified 18 autograders and related plug-ins that met our discovery threshold, as described below, and listed in Table 13.

Alloy4Fun [111] is a web application that allows students to write Alloy [83] formal specification models, similar to the interface of the Alloy Analyzer. To build-in autograding support, Alloy4Fun provides a method for instructors to embed secret paragraphs in exercise models; these exercises can then be shared with students who must then meet the specification. The authors provide a method for downloading and visualising overall statistics from the exercises.

AutoGrader [108] compares student submissions to a correct reference implementation, rather than using test cases provided by the instructor. It generates test inputs using a white-box fuzzer, then compares program output as well as any deviation in execution path to the reference implementation to determine correctness.

AutoStyle [35, 186] is a web-based system that allows instructors to provide hints to students on style to improve readability and conciseness. After initial submissions, student code is analysed for Assignment-Branch-Conditional (ABC) score and similarity, then clustered and displayed to the instructor. The instructor then assigns hints to each cluster, after which new submissions will be assigned a cluster using a k–nearest neighbours approach to determine immediate feedback.

ChocoPy [131] is a subset of the Python 3.6 language augmented with static typing to enable students to develop a RISC-V compiler for the language. The authors provide a reference manual for the language as well as an web-based IDE that supports compiling code to assembly. As part of their course implementation, the authors include autograders that compare parsed abstract syntax trees and execute student-generated RISC-V code.

CPSGrader [90] is an autograding system for cyber-physical systems and robotics labs. It uses machine learning to generate test benches to test correctness of student controllers.

EarSketch [157, 168] is a web application, which includes a companion online textbook, to teach introductory programming with Python. Students write Python code to create music. The authors describe an autograding system for assignments that determines if the student code produces the desired musical output.

GitGrade (2018) [193] is a submission system built on top of a Gitlab instance at the University of Washington. It provides interfaces for assignment submissions, code reviews, and autograding scripts, as well as interfacing with Canvas.

Gradescope (2014)[3] is a tool that was mentioned by 15% of the survey respondents. It is a grading solution that is used to evaluate student answers to online assignments, including those that are handwritten, and can be used for quizzes given to students. It's somewhat student-friendly in the sense that the phone app allows them to scan then submit answers they wrote on a paper sheet.

JupyterCanvasSubmit [67] provides a Jupyter notebook to be distributed to students in JupyterHub that will connect with Canvas. It allows a student to submit an HTML version of their notebook to Canvas.

Learn-OCaml [29] is a web-based environment for the OCaml programming language. It includes an autograder that provides feedback to the student and commits their work and results to a Git repository. Ceci et al. [31] extended the functionality by creating an infrastructure to also record and track unsuccessful compilations.

Mastery Learning Quiz App [15] is a quizzing application to support Mastery Learning [19]. To use this tool, students were provided paper quizzes and asked to submit their responses in an online form. A Python script autogrades the form submissions, analyses the results, and provides high-level feedback on learning outcomes.

OpenDSA Programmed Instruction [122] extends the OpenDSA [160] textbook to include true/false, multiple choice, select all, fill in the blank, and other programmed instruction exercises. Student answers are autograded before they are allowed to move to the next section of the textbook.

Reveal [88] is an exam environment with integrated autograding functionality. The system utilises exam-specific virtual machines (VMs) that are installed on computer lab machines. The VMs provide limited access to external resources through a whitelist, including access to Google's cache of sites such as StackOverflow[4], without allowing students to post new questions to those sites. Reveal provides a method for students to obtain hints, or solutions, in multi-part questions and submit their work when they have completed a question. It also includes autograders for both coding questions and student-supplied integration tests.

SnapCheck [184] is an autograding test framework for Snap! [71], Scratch, and other visual programs. It is implemented in Snap! and provides a web-based and API framework to define test cases.

SQL File Evaluation (SQLFE) [182] is an evaluation tool for MySQL and Oracle SQL queries. It scores submitted answers based on interpretation of the query, as well as similarity of the query to instructor-provided answers. Additionally, it can apply the queries to a known sample data set, and compare the results from the student query to the results generated by the instructor's query=.

Submitty [137] provides course management along with language-independent autograding for programming assignments. It has been expanded in 2019 by Maicus et al. [115] to include autograding of distributed algorithms and in 2020 [114] for computer graphics submissions.

TermAdventure [174] teaches command-line terminal skills; it is written in Go and executed from the command line. Students complete challenges that are defined by instructors, which are autograded based on test conditions defined in the challenge.

uAssign [12] teaches command-line terminal skills using an in-browser terminal emulator. Instructors create assignments for students to complete, which are then autograded by the system.

Web-CAT [45] is an automated grading system designed around test-driven development that incorporates interactive coding instruction. Its plug-in-style architecture supports scale and customisation. One notable feature is the support for student-written tests. It has been extended multiple times in the literature, such as adding support for additional languages and submission types including Jupyter Notebooks [117] and Habanero-Java [10].

6.2 Plagiarism (Similarity) Detectors

In the context of plagiarism detectors, we identified 22 tools and their domains, links, and sources that can be seen in Table 14. Despite the name, plagiarism detectors do not actually detect plagiarism (and collusion) [95]. They only report striking similarities among student submissions and let instructors determine which are potentially a result of misconduct. Sometimes, these tools are referred to as similarity detectors.

Seventeen of the tools are dedicated to code while seven of them can also accommodate text via limited human language preprocessing. Most code plagiarism detectors do intra-corpal comparison [105], where both source and copied submissions are assumed to be in the same corpus. CodeQuiry is the only tool that facilitates inter-corpal comparison with some publicly-available code. While inter-corpal comparison can certainly help instructors to identify more instances of plagiarism and collusion, it is not as straightforward as with regular (human) text, where strikingly similar wording is a strong indicator of copying. With computer programs, it is far more likely that independently developed programs doing the same task will be coincidentally similar.

Many plagiarism detectors can be used locally, which may be necessary to conform with the data privacy requirements of many institutions [165]. CodeQuiry, MOSS, and Lichen are the only tools that require the programs to be submitted over the internet.

Regarding the user interface, around two thirds of code plagiarism detectors (11 of 17) only offer command line interface (CLI). Using such tools might be challenging for some computing educators due to their lack of familiarity with CLI [4]. CodeQuiry and Lichen provide graphical user interface (GUI) as web applications while other tools with GUI provide those as the desktop ones.

Three code plagiarism detectors are commercial: AntiCutAndPaste, CodeQuiry, and CodeMatch. Since these tools typically have

[3] https://gradescope.com
[4] https://stackoverflow.com

a trial version, computing educators interested in using these tools are expected to take that opportunity to measure its suitability prior purchasing.

CodeMatch has the largest coverage of languages (45), followed by Copy/Paste Detector (25) and MOSS (24). These tools might be suitable for institutions teaching various programming languages. Sherlock (Sydney) can cover any languages, but that means it uses no language-specific pre-processing and the result might be slightly less accurate. Keywords and syntax for examples, might vary across programming languages and cannot be treated in the same way.

It is worth noting that some tools only accommodate a few programming languages as they consider many language-specific aspects for higher accuracy. JPlag generalises some program statements; BPlag uses a program dependency graph to detect similarity; and Power & Waldron's tool relies on Verilog code behaviour.

Five text plagiarism detectors are found in the literature. Turnitin, Safeassign, and UniCheck accommodate both inter-corpal and intra-corpal comparisons, can only be used online, are featured with GUI (web-based), and are commercial. WCopyFind and plagiarismchecker.com are the non-commercial ones. The former can be used locally and only supports intra-corpal comparison while the latter can search similar text on the internet (inter-corpal comparison).

Safeassign covers the largest number of text languages (22), followed by UniCheck (18) and Turnitin (9). plagiarismchecker.com and WCopyFind claim that they can cover any text languages. However, such claim means no language-specific pre-processing is involved and the tools might have less accurate result.

AntiCutAndPaste is a plagiarism detector with both graphical user interface and command line interface. It is a commercial tool but users can try some features for free. The tool covers C++, C#, Delphi, Java, Visual Basic, and general text.

BPlag [34] is a Java plagiarism detector that deals with extreme code disguises, where the copied code looks completely different to the original one although sharing the same semantic. The tool determines similarity based on dynamic execution behaviour. It is featured with command line interface and it might be useful for broadly-specified assessments (e.g., creating a 2D game with at least two moving objects) and open-ended assessments (e.g., create a game without specific instructions) [165].

CCFinder [91] is a code clone detector that can be used for detecting code plagiarism. Although code clone detection is not quite similar to code plagiarism detection [24], both tasks rely on code similarity. The tool has a graphical user interface and covers: C, C++, C#, COBOL, Java, and Visual Basic.

CodeMatch is a commercial code plagiarism detector with the largest coverage of programming language (46 as of this writing). The tool can be used locally and it is featured with a graphical user interface. Users can use a trial version with limited features.

CodeQuiry is a commercial plagiarism detector covering 20 programming languages (see the list on their website) and regular text. Unique to this tool, it also compares submitted code to 20 billion lines of public code. The graphical user interface is web-based and users need to submit their code online.

Copy/Paste Detector is the part of PMD source code analyser project with a command line interface. The tool can do intra-corpal comparison for 25 programming languages (see the list on the website).

Deckard [85] is another code clone detector that can be used for code plagiarism detector apart from CCFinder [91] and SourcererCC [156]. The tool only covers two programming languages (C and Java) but it is argued to be scalable for large data sets. It is featured with command line interface.

JPlag [142] is a common code plagiarism detector [129] that can also handle general text. The tool was previously designed as a web service like MOSS [158], but now it is designed as a Java desktop application with a command line interface. Further, the code is available on GitHub and it is maintained by the community. The tool currently covers six programming languages: C, C++, C#, Java, Python, and Scheme.

Lichen [136] is a plagiarism detector in Submitty, an open source course management, assignment submission, exam and grading system from the Rensselaer Center for Open Source Software. It can report striking similarities among student submissions written in C++, Java, MIPS, Python and general text.

MOSS [158] is another popular code plagiarism detector [129]. It is a public web service accessible via command line interface. A number of community contributions to the tool are listed on the website, some of which are about the development of graphical user interface. According to the website, MOSS covers 24 programming languages and it is managed by Stanford University.

Plaggie [5] is somewhat similar to JPlag [142] except that it is exclusive to Java submissions. It was developed when JPlag was still a web service and JPlag's code was not open for public.

plagiarismchecker.com is a free online plagiarism detector for any regular text. The tool can search similar text from the internet and report it to the user.

Power and Waldron [140]'s Tool is dedicated to check plagiarism on Verilog hardware description language. The tool is written in C++ and the code is publicly available.

Safeassign (2013) is a commercial text plagiarism detector in Blackboard, a learning management system. The tool can do both intra-corpal and inter-corpal comparisons and it covers 25 human languages including English, Chinese, Malay, and German.

Sherlock (Sydney) is a plagiarism detector with command line interface. The tool can handle any language (both programming and human languages) as it relies on no language-specific pre-processing. The tool was maintained by the University of Sydney. However, the original link is not accessible anymore and a set of developers extracted the code from Google cache and put that on GitHub.

Sherlock (Warwick) [87] (not related to the tool of the same name mentioned above) is a desktop plagiarism detector with graphical user interface. The tool is maintained by the University of Warwick and it is integrated to BOSS submission system [86]. It currently covers C, C++, Java, and general text.

SIM [64] is an early desktop plagiarism detector with command line interface. It was developed with C and it covers 8086 assembler code, C, C++, Java, Lisp, Miranda, Modula-2, Pascal, and general text.

SourcererCC [156] is a code clone detector that is applicable to code plagiarism detection. It is featured with a command line interface and covers C, C++, C#, and Java. The code is available for public.

STRANGE [94] is actually a module to report detailed similarities and differences between two source code files. However, the tool can also act as a standalone similarity detector for Java and Python submissions via command line interface. The tool has been further developed in two directions: comprehensiveness[5] [96] (reporting three layers of similarity, especially useful for trivial and strongly directed assessments [165]) and scalability[6] (replacing running Karp-Rabin greedy string tiling [187] with locality sensitive hashing algorithms). Both are featured with graphical user interface for convenience.

Turnitin is a common commercial plagiarism detector for regular text written in nine languages: Danish, Dutch, English, French, German, Norwegian, Portuguese, Spanish, and Swedish. The tool supports both intra-corpal and inter-corpal comparison. Further, it is featured with intuitive graphical user interface.

UniCheck is somewhat similar to Turnitin except that it covers more human languages (18, see full list on the website). All languages covered by Turnitin except Swedish are also covered by UniCheck.

WCopyFind is another plagiarism detector that can work with any regular text; it does not have language-specific pre-processing. Unlike other text plagiarism detectors, WCopyFind can run locally and it only supports intra-corpal comparison.

6.3 Development Environments

There are a wide variety of development environments that are used. Some are common to classrooms and industry, and others are specifically built to teach programming. In particular, block-based development environments are well represented by several core projects that have spawned multiple derivative projects. The tools listed here are some of the less common ones that we found instructors are using in the classroom. Table 15 contains the details of the tools, their URLs, and how the tool was discovered.

Alice Netbeans Plugin [40] facilitates combining the use of Alice and Java in a CS1 course. It allows students to take the projects they start in Alice, a language developed to teach programming using video and 3-D graphics, and convert them to Java, showing the student how the languages relate, and also how they are different.

Amphibian [18] is a visualisation plugin for the IntelliJ IDE that allows students to switch between block-based and text-based interpretations of Java code. This allows instructors to use both types of representations within their course materials.

Blockly [181] is a block-based programming environment that runs in a web browser. It is developed by Google and released under an open source license. Such is the popularity of the project that it has led to multiple derivative projects in different contexts, such as **BlocklyDuino**, **Blockly@rduino**, **BlocklyProp**, **BlocklyTalky** [161] and **BrickLayer** [53].

cs50.io [7] is a web-based IDE for education. It provides an in-browser code editor, as well as a terminal window and elevated privileges to the student during their session. The browser editor functions as a front-end to a Docker container that hosts a full Ubuntu OS instance. More recently, it adopted GitHub codespaces[7] as the backend for the editor.

MakeCode [14] is both a platform and block-based programming environment for programming microcontroller-based devices. It is developed by Microsoft and released under an open source license. It is available for a number of different microcontroller platforms, including the BBC micro:bit, the Arduino, and the Adafruit Circuit Playground Express.

Repl.it is an in-browser IDE that provides multi-user interaction within the editor, allowing students to work in teams on programming projects. Instructors can monitor coding sessions, and provide feedback within the context of the editor. Also, it provides an automated grader to assess student submissions.

Reveal [89] is a lab environment exam platform, which intends to provide student developers with access to a full suite of development resources, while preventing collusion between test takers. It provides a hinting platform, as well as autograding functionality.

Scratch [148] is a block-based programming environment aimed at children in the 8-16 age range. It was developed by the MIT Media Lab, has been translated to 70+ languages, and is released under an open source software license. It has also been used to teach programming to younger children using the **Scratch Jr** project [50] and older children as they progress in their development using **Patch** [149]. It has also spawned a collection of works to help teachers analyse Scratch projects, such as the **Scratch Analysis Tool** [33], **Dr Scratch** [127] and **Hairball** [20].

SDES [173] or Simple and Dynamic Examination System is an automatic exam question generator. It supports plugins to allow it to automatically generate questions based on arbitrary input data or from XML-formatted files. Additionally, the authors emphasise the tools ability to offer more secure examinations, as the tool runs completely standalone from other tools.

Snap! [72] is a block-based programming environment, much like Blockly and Scratch, which similarly targets both 8-14 year-olds, but also includes more advanced features, such as recursion, to help support older learners. It is developed by the University of California, Berkeley, and is released under an open source software license. An extension to Snap! includes the ability to introduce parallelism into student projects [49].

[5] https://github.com/oscarkarnalim/CSTRANGE
[6] https://github.com/oscarkarnalim/SSTRANGE
[7] https://code.cs50.io/

Thonny [6] is a Python IDE that provides alternative ways for tracing code, visualisations for the call stack geared to novice programmers, as well as collecting analytic data about student activity within the tool. It is also extensible through a plugin architecture.

6.4 Visualisations

The use of program and algorithm visualisations to improve the understanding of algorithms has been practised in computer science education for decades [11]. Initial tools supported the visualisations of sorting algorithms and of various graph algorithms [25]. While this still describes a large number of current tools, alternative applications of visualisations in computer science education have been developed over the intervening years.

In addition to typical algorithm visualisation, new applications of visualisation are used to aid learning in cryptography, operating systems, data structures, and control flow. In congruence with the wider application of visualisation has been the shift in how applications were deployed. Early examples of visualisation tools could only be used on a single platform. Over time this changed and later instances offered support for multiple platforms. This trend continued, with many of the more recently produced tools are developed as web applications such that only a browser is required for use.

The remaining text in this section introduces the tools that were discovered through the literature review and survey. Only tools where availability could be determined, as a binary or through open source code, were included. Table 16 contains the details of the tools, the platform(s) they support, their URLs, and how the tool was discovered.

AlgoRythmics [153] provides interactive visualisations for a small number of algorithms. However, for each of these algorithms the environment provides a number of different modes of interaction from animations to interactively performing the steps of the algorithms to completing portions of the code in a fill in the blanks manner.

AlgoTouch [54] provides a framework for learning about algorithms through manipulating data in a visualisation. AlgoTouch is designed for use with large touchscreen devices or interactive whiteboards. The only issue limiting its wider use is that the available documentation and examples are only available in French. Development appears to be ongoing at time of publishing.

Alice [145] is a block-based programming environment for the creation of animations, interactive narratives, or simple 3D games. Alice is designed to teach logical and computational thinking skills, fundamental principles of programming and to be a first exposure to object-oriented programming. Development on this project is ongoing at the time of publication.

Animal [152] is one of the oldest traditional algorithm visualisation tools that remains available in computer science education. Animal comes equipped with a large number of prepared animations to aid learning of data structures, algorithms, cryptography, searching, sorting and more. Animal requires no complex installation or set-up, students or teachers can download a single executable jar file. The addition of custom animations does require a number of extra steps. Development has continued on this tool for a long time, however the last versioned release was in 2018.

AnimOS CPU-Scheduling [102] provides a framework for testing and visualising a number of CPU scheduling algorithms. The behaviour of processes and scheduling algorithms can be customised and the selections will be shown in an animated visualisation. The system captures and presents relevant statistics from the scheduling simulations for better understanding of the differences between algorithms. It is unclear if there has been any further development since this tool was created.

BlueJ [16] is a Java development environment that is designed specifically for beginners. Included in its long list of features is the ability to visualise the execution of Java code. Development on this project appears to be ongoing at the time of publication.

BRIDGES [28] is a library that provides a basic interface for visualisation in early computer science education. Students can create interactive representations of their data with very little code to aid understanding of different concepts. In addition, recent versions also include additional APIs for the creation of basic games that can be played on the web or smartphone devices. Development on this project is ongoing at the time of publication.

Choc [128] is a toolkit for teachers to craft small programs that can be explored interactively. The programs are visual in nature and execution is controlled through the use of sliders. While this tool shows some potential, there has been no development since 2013.

Cryptography Visualisation Tools are collection of six visualisation systems for explaining different cryptographic algorithms (DESvisual, AESvisual, RSAvisual, SHAvisual, ECvisual, & VIGvisual). They allow students to interactively work through the different cryptography algorithms with questions at different stages. Bucking the trend of more modern tools, each is available as a separate program with binaries for Windows, Linux and Mac. There is no evidence of continuing development on any of the above tools.

CrypTool(s) bundles a collection of resources and tools for learning about cryptography, with versions available as binaries on Windows (CT1 & CT2), a Java version (JCryptTool/JCT) and finally an web based version (CrypTool-Online/CTO). Resources are available to explain algorithms, animate them and even includes cryptography based games and contests. Development on all of the tools in this section appears to be ongoing at time of publishing.

Data Structure Visualizations, provided by David Galles [57] is one of the earliest examples of content provision moving towards the web, it started life as a Java project in the previous decade, was subsequently rewritten in Flash for the web and finally ported to HTML5/Javascript. The site contains a large variety of visualisations for different data structures and algorithms. In addition, the full source code is provided as well as a tutorial describing how to create custom visualisations. It is unclear if there has been any ongoing development with respect to this project since 2011.

Dynamic Data Structures (DDS) [144] provides a web-based interactive simulation of manipulating node objects in a linked list. Students can create nodes and references and manipulate them to simulate different list operations. DDS features accurate portrayal of reference management, such that it is easy to accidentally lose objects when they can be cleared using a simulated garbage collector.

Additional tools available are a prototypical BTree manipulating tool as well as tools for manipulating directed and undirected graphs. These tools are listed as in development, however it is unclear if any development has been done since 2019.

ExplorViz [74] is a software visualisation system, rather than program or algorithm visualisation. The aim of this software is to present the user with a representation of the architecture, runtime behaviour and development process using a number of static or interactive 2D and 3D visualisations. This tool remains under development at time of publishing.

Flap.js [103] is designed as a web based alternative to JFlap. Flap.js is developed by students (under the supervision of a faculty mentor) specifically for use in classes about the theory of computability. The project remains under development at time of publication.

Flow [27] is different from traditional approaches to program or algorithm visualisation. Rather than focus on data and how it is manipulated, Flow visualises the flow of control during the execution of a Java application. Flow can be particularly useful when introducing students to concepts like call stacks or when describing the inversion of control for UI programming. Flow is available as a standalone program, but may more often be used as a plugin for the IntelliJ IDEA development environment. It is unclear if development is ongoing as the latest version of the plugin was released in 2020.

Highway Data Examiner (HDX) [178] provides a framework for visualising a number of graph algorithms as well as a number of real world data sets that they can be tested on. The project aims to provide data sets small enough that algorithms can be worked through manually as well as large scale examples representing places in the real world. Algorithms can then be directly visualised as an overlay over the real maps from OpenStreetMap[8]. Work on HDX and the datasets it utilises appears to be ongoing at time of publishing.

JFLAP [151] is a collection of visual tools which can be used as an aid in learning concepts of formal languages and automata theory. JFLAP evolved from a collection of existing tools which were ported to Java and continuously built upon for many years. The most recent version of JFLAP (7.1) was released in 2018, though it appears that a consistent effort has been under way since 2015 to port the functionality to HTML5 and JavaScript to enable its use within the OpenDSA interactive eTextbook (see Flap.js above). JFLAP is provides visualisations of the represented systems as well as interactive components for learning how they work. In addition, OpenFlap content within the OpenDSA interactive eTextbook can be provided in the form of gradable exercises the students must complete.

jGRASP [39] is a lightweight development environment created to provide visualisations to improve the comprehensibility of software. It provides control structure diagrams, UML class diagrams, and run-time visualisation for primitives and objects. It is available as standalone software as well as plugins for Eclipse and IntelliJ IDEA. Development is ongoing at the time of publication.

Jive [84] is an eclipse plugin that provides visualisations of the execution of Java code. This visualisations are presented in a number of different forms and adds features such as reverse stepping and the extraction of finite state machines from the execution trace of a program. Development appears to be ongoing on this project up until 2021, and may be continuing at time of publication.

JavaScript Algorithm Visualization Library (JSAV) [93] provides the scaffolding in JavaScript to produce algorithms visualisations for use within web pages and specifically eTextbooks like OpenDSA. Last updated in 2019, the JSAV project itself does not contain a large number of examples that can be used. However, due to its heavy use within the OpenDSA interactive eTextbook there are a large number of examples available[9].

JSVEE and Kelmu [166] are libraries that enable the visualisation of programs on the web. This is similar in concept to JSAV, but intended instead for the visualisation of programs. JSVEE allows the creation of embeddable animations showing the execution of programs, Kelmu enables the addition of annotations to these animations to provide extra information and improve learning. The animations produced are ideal for use within on-line interactive eTextbooks.

JSVEE supports Python and Scala, but animations must be generated using a transpiler to create the custom JavaScript that is required. JSVEE and Kelmu were last updated in 2020.

JSON-based Algorithm Animation Language [180] (JAAL) is an attempt to define a language that can be used to represent algorithm animations. Typically, most implementations either have a custom representation or can only be represented in code. One novel feature of this tool is that user can interactively perform the steps of the algorithm and be scored on correctness with feedback. Development on this project is ongoing at time of publication. **Marble MLFQ** [97] is similar in design and purpose to AnimOS. Marble MLFQ is however focused on only the multilevel feedback queue rather than the different algorithms available in AnimOS. Marble MLFQ has not been developed since 2018.

Moodle Trace Generator [155] generates code tracing quizzes that provide feedback in the form of program visualisations. Questions can be generated in a number of forms for both the C and Python programming languages. Development of this project is ongoing at time of publication.

Omnicode [92] is a live programming environment with always-on run-time value visualisations. The project appears to have ceased development when it was first published.

Online Python Tutor [69] is an early example of a web based program visualisation tool. Originally supporting only the Python programming language, support has since been added for Java, C, C++ and JavaScript. Online Python Tutor has been used by millions to help understand the execution of their code. Active development by the original author appears to have ended in 2019, however a large number of forks of the code exist with continuing development.

[8] https://www.openstreetmap.org/

[9] https://github.com/OpenDSA/OpenDSA/tree/master/AV

Pathfinding Visualizer ThreeJS allows the user to explore how a number of 2-dimensional searching algorithms operate. The user can create 3-dimensional representations of mazes and view the progress of the searching algorithms from different points within the simulation.

Programming Language Interpreter for Visualization of Execution Trace (PLIVET) was initially developed as PVC [80] but has evolved into its current form. PLIVET is a web based application for visualising the execution of C programs. In function, this is the same as Python Tutor Online, but PLIVET adds a small number of features such as IO support and modifies the way programs are represented to make them more understandable. PLIVET appears to be still under development a time of publishing.

Recursion Tree Visualizer [133] visualises the execution of a number of recursive algorithms. There are a number of pre-populated examples, but custom code can be added in either Python or Node. The execution of the functions are shown through the generation of a tree representing function calls and the returned values at each point.

TigerJython [99] is a programming environment for Python. It is primarily designed to improve novice experience through the use of modified error messages, but it does also provide visualisations to aid students in the debugging process. TigerJython remains under development at the time of publication.

Thread Safe Graphics Library [3] (TSGL) is a library created to enable students to learn about the behaviour of multi-threaded programs through visualisation. This includes the ability to visualise the execution of existing parallel code and visualisations of classical problems like the dining philosophers. TSGL appears to be under continued development, but the latest updated was in 2021.

Thread Safe Audio Library [2] (TSAL) is not a visualisation library or too, but instead a sonification tool. Designed to help visually impaired students learn computing concepts the library provides APIs for producing sounds based on the the operations in code. As examples, several sorting algorithms were used to create distinct sonifications. This project appears to be under continued development at time of publishing.

UUhistle [167] is another program visualisation tool for introductory programming education. UUhistle is designed to help novice programmers improve their code-tracing skills and understanding of programming concepts and programming-language constructs. UUhistle is no longer under development as it has been replaced by JSVEE and Kelmu.

Vamonos [30] provides another library for showing algorithm visualisations within a browser. Vamonos has not been actively developed since 2016, but provides a library of pre-made visualisations that can be used. The unique feature provided by Vamonos is that it can output visualisations into a single HTML file containing all of the required CSS and JavaScript.

VeriSIM [141] is an online learning environment designed to aid the understanding of modelling as well as UML diagrams and their creation. This project appears to be under continued development at time of publishing.

VisuAlgo [70] is a platform for algorithm visualisation on the web. Since its initial development, a large number of visualisations have been created and are available on the platform. In addition to the available visualisations, VisuAlgo also enables students to take automatically generated and graded quizzes to help ensure mastery of the relevant topics. VisuAlgo works in multiple languages and has been in continuous development since its creation, with current work is focusing on adapting the system for use on tablets and phones.

Willow [125] is a web based tool that combines elements of algorithm visualisation and program visualisation. Objects are visualised as nodes connected by references and arrays are represented as bars, all of which are updated as the code is executed. In addition, the stack is represented showing the changes over time in a very similar manner to the Flow plugin. Willow has not been actively developed since 2020.

6.5 Class Management

Adoption of Learning Management Systems (LMS) has grown in the last 30 years [68] and the need for remote access for higher education has only accelerated in the wake of the COVID-19 pandemic [135]. Discussion around LMSs may have once been about the auxiliary benefits of using the tool [107] but it is now considered an expectation for most higher education institutions.

We considered Learning Management Systems any tool designed to assist instructors in classroom management and included features for student assessment and tracking, class communication, and content delivery. For the sake of this review, the following terms fall under this grouping:

- Learning Activity Management Systems (LAMS)
- Learning Content Management System (LCMS)
- Course Management Systems (CMS)
- Virtual Learning Environments (VLE)

Through the survey and literature review, this resolved into 10 Learning Management Systems, three tools designed to integrate with LMSs, and four useful in classroom monitoring and interactions. Table 17 contains the details of the tools, the platform(s) they support, their URLs, and how the tool was discovered.

Blackboard [68] began as CourseInfo and held much of the North American education space within six years of its launch. It has since acquired and folded in other products such as WebCT (2005) and Angel Learning (2010) as part of its feature set [135] and re-branded to Blackboard Learn (Blackboard). It can be cloud-based and offers the ability to integrate with third party applications [68]. Favorite tools in a 2019 survey of students and faculty at a small university [9] singled out learning content, assignments, and grading tools. This also mentioned that faculty at their institution focused on using Blackboard as a content repository which suggests a slow adoption of other LMS features.

A usability study comparing Blackboard Learn and Canvas found similar satisfaction ratings, suggesting that these two were equivalent systems for the student experience [68]. Another investigation on general student acceptance of Blackboard also documented [162] positive experiences with the platform.

Some noted severe shortcomings in Blackboard that were overcome with the addition of Microsoft's Sharepoint content management system [189] but the feature set may have evolved to cover these gaps in the seven years since that article.

Because Blackboard is one of the older LMS tools, it is much-discussed in the literature but it was not mentioned by respondents in our survey.

Canvas [68] by Instructure has increased its market-share considerably in the North American region since its initial launch. Like Blackboard and MoodleCloud it can operate entirely via the cloud and integrate third party applications in addition to the typical LMS features of calendar, activity submission, and content delivery [68]. Canvas supplies some limited support to group work within its discussion board system. This LMS is also touted as "the world's most reliable LMS" with a 99.9% up-time in one study [48].

Coursera [41] is a proprietary web-based LMS developed with the goal of increasing student capacity without increasing infrastructure costs in the Massive Open Online Course (MOOC) format [41]. To do this, it incorporates automatic test-based grading, forums, and content delivery. Coursera's features also include integrations with other LMSs such as Blackboard Learm, Moodle, Canvas and D2L.

Doubtfire [146] is one of the newer LMS solutions which advertises itself as a lightweight learning management system. Some distinguishing features include portfolio grading, the creation and the management of student groups, and an Agile-centered task management philosophy with burn charts for student tasks. Literature in the review noted that instructors used Doubtfire and its audio feedback tools, in addition to the LMS used at their intuition (BrightSpace by D2L), as a way to provide better discussion opportunities between online and campus students and their tutors [146].

Google Classroom is a light-weight cloud web based freemium model LMS. One interesting feature is the ability for this LMS to export data for further analysis but this LMS does not yet allow for integration with edtech tools and third-party ad-ons.

Gooru [112] is a free customisable LMS designed for content sharing and collaboration between courses. One notable feature is summary data analysis for monitoring student activity and engagement.

Gradecraft [77] was built at the University of Michigan with the philosophy of gamification to improve student engagement. This brings with it features such as a leveling system, achievement badges, and leaderboards. "Levels are unlocked" instead of "prerequisites met" along with a goal-setting Points Planner.

Isaac Computer Science [183] is a free computing education learning platform designed for secondary schools in the United Kingdom. This platform offers gamified interactive learning content aimed at students working towards UK standardised assessments.

Moodle, and its cloud version MoodleCloud, is known for supporting higher institutions with added customisation. Moodle was the most popular learning platform in the literature review by far due to its mutability as many of the papers focused on institution variants of Moodle.

Notably, one Moodle variant was tuned for low-bandwidths to help cope with the needs of rural, low-income areas in developing nations [192].

Taman Belajar [191] is web-based free LMS designed for MOOC interaction that was found to be easy and interesting to use and appropriate at the secondary level [191].

6.5.1 *LMS Integrations.*

HKU Space Soul [78] is a mobile application that functions as an integration with Moodle 2.0 to use Mobile Learning (M-Learning) to aid in knowledge scaffolding, content access, and reduce cognitive load [78].

One researcher cited a "glaring shortcoming" in the sweeping lack of support for computing courses in today's LMS [143].

Real Talk [147] Audio feedback tool developed for integration with the DoubtFire LMS. This scaling system is meant for teacher-asynchronous and student-synchronous discussions for online courses.

WhatsApp is a Mobile Instant Messaging Service (MIM) that facilitates near-synchronous communication across web or mobile clients. One novel use of this tool included using it to leave asynchronous audio feedback on student work in addition to working with the Doubtfire LMS [146] with a positive response from students. This can also be leveraged by integrating it via API with other systems such as web-based social discussion platform TutorSpace. This allowed for better content distribution and faster response time in student-teacher communication [113].

6.5.2 *Classroom Monitoring and Interaction.*

Elgg OSN [179] is more of an engine than a tool itself but success was cited in using it to create a custom Online Social Network (OSN) as a supplement to the learning management system. Default features include blog, discussion board, file gallery and profile for the purpose of community building.

Microsoft Teams [8] is an MIM created by Microsoft as part of the MS Office 365 suite and was used to offer online tutoring support during the COVID-19 pandemic, when travel and in-person classes were limited.

Piazza is a commercial tool that enables asynchronous interactions between and among students, teaching assistants, and instructors. Instructors can also endorse and moderate answers and comments posted by students.

RepoBee [65] is an open source command-line tool that helps with management of Git repositories on GitHub, GitLab or Gitea. Features include templates and a plugin system that allows for customisation and integration into systems like GitHub Classrooms.

Spatial [138] is a metaverse 3D visualisation tool that was leveraged in one study by integrating it with its learning management system (Moodle 4.0) environment during the COVID-19 lockdown. This virtual classroom had cognitive impact even when delivered over varying broadband speeds.

Because the adoption of LMS is typically made at the institutional level and not with each individual course, this list may have

less variation and experimentation than other categories of tools presented here. We can speculate that this may also contribute to Learning Management Systems being the "least favourite tool" among our survey responses.

6.6 Open Textbooks

The number of open textbooks[10] has increased in recent years, but many of them are not evaluated in published research. As such, our review of the literature did not reveal many open textbooks or tools to create open textbooks[11].

Jupyter Notebooks[12] were originally developed as tooling for interactive data science. They have since been enhanced [42] to allow instructors to include executable coding examples and assignments within the Notebook. Auto-grading support has been built into some existing tools as well [118].

OpenDSA[13] began as an open-source data structures and algorithms book, which was moved to an e-book format to allow interactive visualisations to be included inline in the text [160]. It has been extended to be a subject-agnostic, full-featured e-text framework that also includes inline quizzes, programming exercises, and LMS integration. It also collects extensive analytics which are available to instructors for insights about how their course material is being received by students [47][44].

Runestone Academy[14] provides an authoring and delivery platform for open-source interactive textbooks focused on computer science. Runestone's authoring tools allow instructors to create their own textbooks with interactive elements, including assessments, and live examples, and then host them on the Runestone infrastructure. Several popular first CS books are available from Runestone, including *Java,Java,Java* by Morelli, et al. [15], and CSAwesome, the College Board-endorsed curriculum for the AP Computer Science A course [16].

6.6.1 Other Tools for Creating Open Textbooks. Even with the variety of tools reported being used to create open textbooks, a large number of papers we found talked about using more general, existing open-source tools to create textbooks. Included in this set (but not exhaustive by any means) are tools for taking meta-formatted text files, and converting them to static HTML/CSS with some Javascript interactivity (i.e. Hugo[17], Jekyll[18], Sphinx[19]), GitHub[20]. Also, there were a number of discussions about creating cross-platform mobile applications using widely available frameworks (see Flutter[21] and Cordova [22] as examples).

6.6.2 Approaches and Standards. We identified a number of papers which, while they didn't identify open-textbooks directly, had what we thought were useful ideas for addressing issues around creating and using open textbooks. These includes:

- Instructional design model proposals when using open texts to support online courses [150].
- A model for creating a virtual environments that could be used as a basis for textbooks [63].
- A process for converting traditional texts into online, open texts [130].
- A process for reusing and refactoring open text materials to account for the change in context (i.e. different regions) where the materials are used [37][22].
- A review of studies on the effectiveness of open course materials used in higher education [188].
- Models for evaluating the motivational influences on using open textbooks in introductory CS courses [139][159].
- Managing versioning of curricular materials in a distributed fashion [116].

6.7 Polling / Quizzing

By nature, polling and quizzing are not specific to CS education, and indeed most papers we found describe tools used in different domains. Following this, we decided to include in our analysis tools that are not CS-centred as long as they can be used in computing education or modified to that purpose. Asking questions to students is also a facility included in many larger-scope tools or tools centred on a specific category (e.g., virtual conference software). Thus, for the poll/quiz category we needed more restrictive queries than for other types of tools studied. Moreover, as the same query was not giving meaningful results in both investigated databases, we needed to customise those queries separately. As a result, we obtained a larger ratio of number of relevant papers over number of result papers.

What's more, when analysing the query results, limiting ourselves to examining only the metadata content of found papers led to missing interesting results. Thus, unlike the other topics, we examined the full text of the reviewed papers to detect poll/quiz tools or such functionalities in larger-scope tools. For the same reason, we used a 10-in-a-row stopping criterion to examine query results.

In total, 35 tools were identified (see Table 18), 21 coming from the literature review, eight more from the survey and six more were previously known to the working group but were not identified in either the literature review nor the survey.

We fount that most tools in this topic are commercial, which we believe is largely due to the fact that there is a broad educational market at stake: polls and quizzes apply in all domains, in connection to many platforms or kinds of other tools[23], and from

[10] An open textbook is defined as an electronic document that has an open license that makes it free to use and change. Different "free" licenses may require attribution of the original authors and/or licensing new contributions in a free form.
[11] There are a wide variety of textbooks offered by textbook publishers in some type of electronic form, often including interactive content and knowledge assessments built into the text. As our focus is on open-source, free-to-use texts, we do not enumerate them here.
[12] https://www.jupyter.org/
[13] https://opendsa-server.cs.vt.edu/
[14] https://blog.runestone.academy
[15] https://runestone.academy/ns/books/published/javajavajava/book-1.html
[16] https://course.csawesome.org
[17] https:/gohugo.io/
[18] https:/jekyllrb.com/
[19] https:/sphinx-doc.org/
[20] https://github.com
[21] https://www.flutter.dev/
[22] https://cordova.apache.org/
[23] e.g. virtual meeting tools (Zoom, Webex, Teams, GatherTown) or content management solutions (Google Docs Editor).

students starting in primary school, continuing through secondary and higher education, and into employment.

As a consequence, we found few papers presenting a tool designed by the authors, and still less presenting open-source tools. Most papers reported about the use of existing tools. Indeed, the fact that there is a large corpus of existing commercial polling/quizzing tools competing for market shares leads most of them to have appealing characteristics (i.e. easy to use, technically advanced, well-documented and supported, offering a free plan and integration to various LMS). This in turn leaves little room for the proposal of new tools in the topic, save for specific niches [66]. Moreover, the COVID-19 pandemic has put a much greater emphasis on educational technologies, e.g. polling [164], leading many instructors to experiment with existing tools and report about their findings through research papers in education [32, e.g.].

Below, we briefly review the tools found in the literature review and survey. We do not mention polling/quizzing tools integrated by default in an LMS.

Acadly is mostly known for capturing attendance (it uses WiFi to propagate a local signal throughout the classroom via mesh-networking). Questions can be asked to students through various kinds of polls supporting image and Math-Tex content. This tool can be used either through a browser or within a dedicated app. It integrates with LMSs and Zoom. [32] use it for proposing game-based learning.

Adobe Captivate allows users to produce interactive presentations using the slide show metaphor (i.e., alternating a slide of text, with a slide containing a video followed by a slide containing a quiz). Kolås [100] report that content was initially rendered in Flash and now in HTML5 format. It includes several kinds of quizzes such as drag-n-drop or short text answers.

Google App Scripts is a tool that allows users to make connections between different kinds of cloud-based tools such as Google forms and presentations but also whiteboards. Kloos et al. [98] shows how this tool facilitates information flow between tools without having to do it manually. Smoothing the process to switch from one activity to the other one (e.g., along the Bloom's Taxonomy) reduces the instructor's mental load while conducting classes.

Facebook is a social media whose use for teaching was more thoroughly investigated since the COVID-19 pandemic. A social media allows to enhance social interaction, more focused learning, more 'amiable' classroom environment [190]. When students are enrolled in a class group, they can be proposed polls and Yu and Liu [190] reports that this led to more democratic and participatory forms of dialog.

Factitious [66] is one of the very few open-source tools we found for quizzing. The authors created a new tool because they needed to fit a particular niche: detecting fake news and improving news literacy. In their case, they wanted to control a game environment to provide contextualised feedback.

GatherTown is a proximity-based video conferencing solution where people are represented by game-like avatars on a map and where people in the same map location can interact. This system can be used in distance learning settings to recreate a sense of awareness to other students and, in particular, can help students to form teams by assembling their personas around a common location of this virtual world. Siegel et al. [164] reviews several innovative works relying on this platform.

Google Forms is a long existing tool for writing surveys. It is integrated with Google classroom (2014) and since 2016 it also allows to compose quizzes typically for educational purposes. It is often used in combination with Google Docs [21].

GoToMeeting and GoToWebinar are commercial virtual meeting tools (VMT) that include audience poll and survey functionalities. These tools can be embedded in virtual classrooms, and Wang and Lee [185] reports on its use in a physical classroom.

HapYak is a commercial solution to create interactive videos. Among others, the content creator can poll viewers and embed quizzes as well in its online videos, response being recorded for individual viewers. Interactive videos allow to engage students in an active learning position compared to simple videos. Kolås [100] advises to use them in MOOCs to avoid high drop-out rates.

iClicker is a commercial audience response system that is famous for coming with hardware remote controls that students use to indicate their answers. Current versions use smartphones to replace the remote controls, though institutions might provide students with the hardware to avoid social discrimination. This tool can be used for attendance recording, polling or quizzing students. Shryock [163] reports that it was through the use of peer interactions and classroom discussions that the tool positively affected the learning experience.

Kahoot! is one of the most frequently mentioned tools to question students, e.g. to gamify day-to-day lessons [32, 32, 73, 98, 164]. This web app offers different kinds of quizzes but no direct LMS integration.

OpenIRS-UCM [62] is one of the very few open source software proposed in this category. It is an interactive polling system written in Java and compatible with different hardware clicker systems as well as smartphones. Moreover it offers LMS integration and quiz results can be exported.

PollEverywhere is a frequently-cited tools [21, 32, 121], because of the many questioning features it allows (for example, polls, surveys, quiz, word clouds). It can be used in a classroom or for a larger audience. The company maintaining it is open-sourcing portions of software.

Polly is one of those systems that gained new interest during the COVID-19 crisis [123]. It integrates with communications tools like Slack and Microsoft Teams (which can be useful in the context of student projects), as well as video conferencing platforms like Zoom.

Socrative is a quizzing tool that allows multiple choice, true/false and short answer questions. There is a dedicated app for the teacher and one for students. Questions can flow according to a rhythm decided by the instructor or alternatively by the student. The tool allows instructors to give contextualised feedback at the end of each question (as in H5P), making it a good fit for auto-evaluation assessments.Both Chan and Lo [32] and Kloos et al. [98] mention

this tool as being a popular choice, yet it didn't show up in our survey responses. This is probably due to the fact that it targets the K-12 education market.

TwitchTV is a live streaming platform mainly used by gamers, but the fact that people in the audience compete to have the streamer's attention and that polls are available to get feedback from the audience makes a tool that can be used in some teaching context [171].

Veriguide [106] was introduced as a web toolbox to promote the enhancement of teaching and learning in writing, such as student writing project or internship reports. It contains different components including an online polling system. It also provides an anti-plagiarism system that seems to have overtaken other initial functionalities since then.

Wooclap is a quizzing and polling web service that can integrate with LMS and Microsoft Powerpoint. It can be used in class to engage students, with or without the dedicated phone app, and also in distance learning applications.

Zoom is a virtual meeting tool whose usage exploded during the COVID-19 pandemic. This tool appears several times in the literature review [32, 123] and is also mentioned in more than 15% of the answers to the community survey. The tool allows hosts to poll their audience in several ways, including answer selection, agree/disagree items, short text answers. Polls can be planned in advance and their result be exported.

7 DISCUSSION

Some surprising issues arose during the analysis of the survey responses and the literature review assembly. These highlight not only possible faults within the information collected but also some trends of concern in computing education academia, including the availability of tools presented in peer-reviewed literature and the general discoverability of tools.

7.1 Availability

A recurring theme in our literature review was that a staggering number of tools were unavailable to reviewers (64.1%, $n = 227$), and as such, were excluded from our literature corpus. Surprisingly, this included a large number of tools that were touted as "open source" solutions. In some cases, this was because the tool required the author to be contacted for access to the tool; in others, the links were broken or pointed to an landing page without a link to reach the tool itself; and in yet others, no link or method for obtaining or using the tool was provided at all.

It is possible that some tools may have lost supporting infrastructure or stewardship of its author or authoring organisation during the 10 year review period. It is also possible that some authors of tools have been hesitant to make tools available due to security concerns, especially with respect to web environments under active development that may become the targets of security attacks or other related breaches. These concerns are understood and appreciated by the reviewers and, we suspect, the broader CER community. Conversely, it is important for the community to be able to access tools in order to review them properly, and for reproducibility; additionally, it is reasonable to expect that solutions advertised (in literature or otherwise) and "open source" solutions are indeed open and accessible to the research and teaching communities.

7.2 Discoverability

Respondents on the survey gave us a window into how participants in the CER community find and use tools. Overall, discoverability is a very large challenge as the vast majority (83%) of tools reported were found through colleagues or inherited from a course or institution. In fact, more people built their own tools (10.1%) than found tools via the community's publications and venues (7.5%). Unsurprisingly, therefore, some commonly used tools (e.g., Piazza) did not appear in the literature review. This poses a significant challenge to tool discovery for new educators entering the field, especially those at smaller institutions.

7.3 Development & Deployment

Tool deployment can be challenging for both open source and commercial tools. That said, the major barriers and challenges are quite different between the two.

For commercial software, there were significant concerns from the CER community over price or cost sharing of a tool. Notably, educators were more worried about data security for commercial tools than for open source tools even though commercial companies, in theory, have increased resources to spend on such efforts. We postulate that the concern for cost is driven by the fact that institutions, departments, and instructors have limited budgets. The heightened concern of giving data to an organisation outside the university (data security) may be driven by increased level of trust in in-house tools or tools that keep data in house.

For open source software, the concern over price is replaced with concerns around maintenance, integration/learning time, and reliability. We postulate that, because many open source tools are often not actively maintained and developed once they are released, educators are wary of having to update the tool themselves or fix bugs as issues arise. Similarly, these tools often require more manual effort on the part of the educator to implement and learn how to use. Both of these issues are exacerbated by the fact that anyone can easily publish open source software without proper user testing and/or intention to make the software useful for other educators. In short, open source tools often trade cost for time, which can be an issue for educators who are already time constrained.

We also find it intriguing that developing tools (whether open source or commercial) faces the same concerns as integrating open source tools, once the shift from integration/learning time to development time is taken into account. We postulate that this is again driven by fears around the time commitment needed to keep the tools usable and reliable, especially when there may not be many collaborators helping educators develop and maintain a tool.

Finally, as mentioned earlier, we also analysed various additional cuts of the data to see if any of the demographic data affected the overall results and found all of those analyses to be inconclusive. This indicates to us that the trends we uncovered are broadly applicable, holding across institution type, class size, TA support, years of experience teaching, percentage of time dedicated to teaching, and other demographic metrics.

8 LIMITATIONS

We recognise that our experiences as computer science educators influenced the areas of tooling we considered, how we formulated the searches for the literature review, and even where we submitted our surveys.

8.1 Literature Review

We have identified several limitations with our literature review:

- Each topic area was queried independently. This required the use of different relevancy criteria and search queries to produce enough results without overwhelming the search with an extensive number of false positives.
- Human error could have occurred in judging initial relevance of papers by abstract and title without reading the full paper.
- The quality of indexing and results returned by query engines of the repositories we searched may vary. It's likely that additional resources would be found by extending the literature review to include other search engines such as Google Scholar, which may also result in differing "relevance" scoring and surfacing different papers.
- Literature varied on the focus of tools or methodology using tools. There may be some variance between working group members in deciding whether the tool was discussed enough to justify its inclusion in the review.
- Our inclusion criteria are not guaranteed to select all relevant papers.
- The literature review aimed to list available tools without considering their coverage and their number of users in the CER community; it is possible that some tools only work for specific use cases and environments.

8.2 Community Survey

Likewise, we identified several limitations to the community survey:

- The timeline for the survey likely influenced the small number of responses ($n = 43$). Survey distribution was driven by the timeline of the conference, which was in the summer when most instructors are on holiday. We also asked for responses within two weeks.
- The venues to which we submitted the request likely influenced the number of responses. In particular, the SIGCSE mailing list subscribers were asked to simultaneously complete several ITiCSE working group surveys, which we suspect led to "survey fatigue".
- The type of responses we received were likely influenced by the venues through which we advertised. We focused on computing education venues, but it may be that the respondents covered a specific segment of computer science instructors, specifically those who teach introductory classes. Thus, we may not have captured some tools used by instructors in more specific courses (*e.g.*, tools associated with teaching mobile application development). Expanding the set of groups would likely have resulted in a more diverse set of responses.
- The length of the survey may have deterred some respondents from completing it. We aimed for a comprehensive survey that covered tooling areas that we hypothesised most instructors would find familiar. However, this resulted in a lengthy 38 question survey with several additional open response fields. We had a number of partially-complete survey responses, which may have been due to the survey length.
- The definition of a tool is somewhat subjective and resulted in a elevated degree of variance in survey responses. Python, for example, was submitted 4 times and could be classified as a means to make tools rather than a tool on its own. Respondents may have assumed that the survey was requesting *novel* tools and omitted those that they take for granted as known within the computing education community. There may also be the assumption that some tools provide a very basic level of classroom support (one extreme example could be Microsoft Office and Google Docs application suites) and decided not to include them.

8.3 Other Potential Biases

We include in the appendix a list of tools that we have used ourselves, in addition to those identified in the literature review and survey. Tools reported based on working group members' experience are subjective, but we have ensured that the listed tools were available publicly, useful to computing educators, and, in our judgement, relevant to the discussion.

Another limitation centres on the "openness" of the tools. When discussing tools incurring a fee, our survey conflated "commercial" and "proprietary" solutions; however, these terms are not synonymous. Some tools are "commercial" and "open source", were the tool is free to use, and the "license" pays for support. Other tools that are "commercial" and "open source" have a core set of features that is available as an open-source tool, but then include additional, value-added features that are only available via a license. Further investigating the more subtle aspects of software licensing and monetary models - and how they impact the CER community - is an important area of future investigation.

9 CONCLUSION AND FUTURE WORK

In this work we aimed to better understand why many tools with similar functionalities continue to be re-implemented by different instructional groups within the Computing Education community and often find limited adoption beyond their creators.

We began by exploring **RQ1:** *What are existing tools available for computing educators?* To do so we performed a literature review and community survey and detailed our results in Section 6 with additional data and tools available in the appendix. While not a completely exhaustive list of tools, we hope that this data can serve as a foundational starting point for those searching for software tools for computer education.

By analysing both the survey results and the experiences of finding tools through a literature review, we developed an answer to **RQ2:** *What are challenges that need to be addressed to support better software use, software development, availability, and discoverability of existing community projects?* Detailed in Sections 5 and 7, we found that availability and discoverability were major impediments to tool use; most educators found their tools through word of mouth from colleagues or inherited their tools through a course or institution. We also found that fears of integration, maintenance, and development time leading to a lack of reliability were a major

impediment to tool creation and open source tool use. Conversely, price / cost was the major limiting factor for commercial tool use and data security was a higher concern for commercial than open source tools.

Finally, we conclude with an answer to **RQ3:** *What suggested directions could address the challenges to using, developing, and discovering CER community projects?* The outcomes of our literature review and community survey suggest two main directions for tool use and development in the computing education community going forward - specifically, that open source tools described in the literature should provide an archival link and that a central repository is needed to enhance discoverability and adoption of these tools.

Papers with open source tools should include an archival link to the code. It is not sufficient note that readers may "ask the author" for code advertised as open source; likewise, a download link on a personal website is not sufficient for archival purposes. Such solutions construct brittle systems where broken links and deprecated email addresses render tools permanently inaccessible. We argue that publications that cannot provide an archival link should not claim that their tool is open source, as these works are not truly accessible to the community and for posterity. Tool archives should exist in a durable location for longevity's sake. The question of the appropriate venue for archival is a significant question that must be addressed; commercial services, such as GitHub, may provide a more durable home for open source projects, but raise other issues, such as inappropriate use as training material for AI services, or possibly eventual closure as an active host, as happened with code.google.com - which had been host to over 1.4 million open source projects. Efforts such as [44] are paving the way for how frameworks can be created for sharing computer science content more broadly. Ultimately, links to software should be required at paper submission, to the extent practical, taking into account funding restrictions, patent applications, or similar restrictions that might warrant exception. Such exceptions should be clearly justified by authors upon submission and reviewed by the venue of publication.

The community requires the construction, maintenance, and moderation of a a central tool repository. This would not only ensure easy access to archival links from publications but would also enable a central location for early career professionals to more quickly and easily find relevant tools. Furthermore, such a repository needs to be maintained and moderated to overcome barriers to tool adoption and use. Similar efforts have proven successful for introductory computer science course materials—e.g., EngageCSEdu [124], which provides open access to peer-reviewed material. By following EngageCSEdu's model as an example, tools may be submitted for peer-review to the repository, then presented with context for their use, without needing to find the developers' websites or git repositories. Finally, we note that this repository should also welcome closed source tools (with a corresponding binary) and open source tools without papers. It should therefore be the job of the maintainer to ensure the submitted tools are up to a standard that enables (relatively) easy use and adoption by the community.

Tools that reach a "critical mass" of use should be eligible for some form of community support. Tools that are widely adopted incur a cost that is borne by the original developers and the institutions. This cost can be non-trivial, even when academic discounts (e.g., for hosting costs) are utilised. If a tool is "widely-adopted," as defined by the community, there should be support for that tool. At a minimum, the community has an interest in avoiding bit rot, as tools can cease to work due to incompatibility issues. A community-developed resource pool could help to address this burden, and also address concerns of ongoing maintenance and security. We could also leverage development and software engineering courses that require students to participate to Free Open-Source Software projects [23, 104]. The community could maintain a list of community tools seeking new contributors, contact points, and how / when volunteers can become involved.

The breadth of tools that have been developed for the computing education community by the community shows the commitment that instructors and staff have to making computing accessible to anyone. This report shows that many of these tools await new adopters, but that there are also many other tools that still need to be built. We hope that this report provides some clarity for what is available, and what is needed.

As mentioned previously, this working group has compiled the data we collected at https://csed-tools.github.io/. We encourage members of the CER community to submit new tools, edits, and maintenance pull requests on the source site for this repository at https://github.com/csed-tools/csed-tools.github.io/. Our hope is that we, as a community, might use this as a starting point, that we might ***stop reinventing the wheel*** - and instead work together to build the tools we and our students will use in the coming decades.

REFERENCES

[1] Saša Adamović, Irina Branović, Dejan Živković, Violeta Tomašević, and Milan Milosavljević. 2011. Teaching interactive cryptography: the case for CrypTool. In *IEEE Conf. ICEST*. https://doi.org/10.13140/2.1.1065.0564

[2] Joel C Adams, Bryce D Allen, Bryan C Fowler, Mark C Wissink, and Joshua J Wright. 2022. The Sounds of Sorting Algorithms: Sonification as a Pedagogical Tool. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 189–195. https://doi.org/10.1145/3478431.3499304

[3] Joel C Adams, Patrick A Crain, Christopher P Dilley, Christiaan D Hazlett, Elizabeth R Koning, Serita M Nelesen, Javin B Unger, and Mark B Vande Stel. 2018. TSGL: A tool for visualizing multithreaded behavior. *J. Parallel and Distrib. Comput.* 118 (2018), 233–246. https://doi.org/10.1016/j.jpdc.2018.02.025

[4] Alireza Ahadi and Luke Mathieson. 2019. A comparison of three popular source code similarity tools for detecting student plagiarism. In *21st Australasian Computing Education Conference*. https://doi.org/10.1145/3286960.3286974

[5] Aleksi Ahtiainen, Sami Surakka, and Mikko Rahikainen. 2006. Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises. In *Sixth Baltic Sea Conference on Computing Education Research*. https://doi.org/10.1145/1315803.1315831

[6] Aivar Annamaa. 2015. Introducing Thonny, a Python IDE for Learning Programming. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '15)*. Association for Computing Machinery, New York, NY, USA, 117–121. https://doi.org/10.1145/2828959.2828969

[7] Dan Armendariz, David J. Malan, and Nikolai Onken. 2016. A Web-Based IDE for Teaching with Any Language (Abstract Only). 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, 711. https://doi.org/10.1145/2839509.2844712

[8] Karina Assiter. 2020. Experiences Offering an Online Version of Computer Science Support (Peer Tutoring) to Undergraduate Computer Science Majors in the Era of COVID-19. *J. Comput. Sci. Coll.* 36, 2 (oct 2020), 96–107.

[9] Mohammed Awad, Khouloud Salameh, and Ernst L. Leiss. 2019. Evaluating Learning Management System Usage at a Small University. In *Proceedings of the 2019 3rd International Conference on Information System and Data Mining*

(Houston, TX, USA) *(ICISDM 2019)*. Association for Computing Machinery, New York, NY, USA, 98–102. https://doi.org/10.1145/3325917.3325929

[10] Maha Aziz, Heng Chi, Anant Tibrewal, Max Grossman, and Vivek Sarkar. 2015. Auto-grading for parallel programs. In *Proceedings of the Workshop on Education for High-Performance Computing - EduHPC '15*. ACM Press. https://doi.org/10.1145/2831425.2831427

[11] Ron Baecker. 1973. Towards animating computer programs: a first progress report. In *Proceedings of 3rd Man-Computer Communications Seminar (CMCCS '73)*. National Research Council of Canada, Ottawa, Ontario, Canada, 4:1—-4:10. https://doi.org/10.20380/GI1973.04

[12] Jacob Bailey and Craig Zilles. 2019. uAssign. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. ACM. https://doi.org/10.1145/3287324.3287458

[13] Kannan Balasubramanian. 2021. Experiments with the Cryptool Software. In *Research Anthology on Blockchain Technology in Business, Healthcare, Education, and Government*, Information Resources Management Association (Ed.). IGI Global, Hershey, PA, USA, 424–432. https://doi.org/10.4018/978-1-7998-5351-0.ch025

[14] Thomas Ball, Abhijith Chatra, Peli de Halleux, Steve Hodges, Michał Moskal, and Jacqueline Russell. 2019. Microsoft makecode: embedded programming for education, in blocks and typescript. In *Proceedings of the 2019 ACM SIGPLAN Symposium on SPLASH-E*. 7–12.

[15] Elisa Baniassad, Alice Campbell, Tiara Allidina, and Asrai Ord. 2019. Teaching Software Construction at Scale with Mastery Learning: A Case Study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE. https://doi.org/10.1109/icse-seet.2019.00027

[16] Michael Berry and Michael Kölling. 2014. The State of Play: A Notional Machine for Learning Programming. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE '14)*. Association for Computing Machinery, New York, NY, USA, 21–26. https://doi.org/10.1145/2591708.2591721

[17] Vincent Berry, Arnaud Castelltort, Chrysta Pelissier, Marion Rousseau, and Chouki Tibermacine. 2022. ShellOnYou: Learning by Doing Unix Command Line. In *27th ACM Conference on on Innovation and Technology in Computer Science Education*. 379–385. https://doi.org/10.1145/3502718.3524753

[18] Jeremiah Blanchard, Chistina Gardner-McCune, and Lisa Anthony. 2019. Amphibian: Dual-Modality Representation in Integrated Development Environments. In *2019 IEEE Blocks and Beyond Workshop (B&B)*. 83–85. https://doi.org/10.1109/BB48857.2019.8941213

[19] Benjamin S Bloom. 1968. Learning for Mastery. Instruction and Curriculum. Regional Education Laboratory for the Carolinas and Virginia, Topical Papers and Reprints, Number 1. *Evaluation comment* 1, 2 (1968), n2.

[20] Bryce Boe, Charlotte Hill, Michelle Len, Greg Dreschler, Phillip Conrad, and Diana Franklin. 2013. Hairball: Lint-inspired static analysis of scratch projects. In *Proceeding of the 44th ACM technical symposium on Computer science education*. 215–220.

[21] Uugangerel Bold and Borchuluun Yadamsuren. 2019. Use of Social Media as an Educational Tool: Perspectives of Mongolian University Educators. In *Proceedings of the 10th International Conference on Social Media and Society* (Toronto, ON, Canada) *(SMSociety '19)*. Association for Computing Machinery, New York, NY, USA, 233–243. https://doi.org/10.1145/3328529.3328564

[22] Pavel Boytchev and Svetla Boytcheva. 2019. Innovative ELearning Technologies in the Open Education Era. In *Proceedings of the 20th International Conference on Computer Systems and Technologies* (Ruse, Bulgaria) *(CompSysTech '19)*. Association for Computing Machinery, New York, NY, USA, 324–331. https://doi.org/10.1145/3345252.3345300

[23] Grant Braught and Farhan Siddiqui. 2022. Factors Affecting Project Selection in an Open Source Capstone. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1* (Dublin, Ireland) *(ITiCSE '22)*. Association for Computing Machinery, New York, NY, USA, 358–364. https://doi.org/10.1145/3502718.3524760

[24] Romain Brixtel, Mathieu Fontaine, Boris Lesner, Cyril Bazin, and Romain Robbes. 2010. Language-independent clone detection applied to plagiarism detection. In *10th IEEE Working Conference on Source Code Analysis and Manipulation*. https://doi.org/10.1109/SCAM.2010.19

[25] M H Brown. 1988. Exploring algorithms using Balsa-II. *Computer* 21, 5 (1988), 14–36. https://doi.org/10.1109/2.56

[26] Peter Brusilovsky, Stephen Edwards, Amruth Kumar, Lauri Malmi, Luciana Benotti, Duane Buck, Petri Ihantola, Rikki Prince, Teemu Sirkiä, Sergey Sosnovsky, Jaime Urquiza, Arto Vihavainen, and Michael Wollowski. 2014. 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* (Uppsala, Sweden) *(ITiCSE-WGR '14)*. Association for Computing Machinery, New York, NY, USA, 31–57. https://doi.org/10.1145/2713609.2713611

[27] Yoann Buch, Yiquan Zhou, and Tair Sabirgaliev. 2016. Flow. http://findtheflow.io

[28] David Burlinson, Mihai Mehedint, Chris Grafer, Kalpathi Subramanian, Jamie Payton, Paula Goolkasian, Michael Youngblood, and Robert Kosara. 2016. BRIDGES: A System to Enable Creation of Engaging Data Structures Assignments with Real-World Data and Visualizations. 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, 18–23. https://doi.org/10.1145/2839509.2844635

[29] Benjamin Canou, Roberto Di Cosmo, and Grégoire Henry. 2017. Scaling up Functional Programming Education: Under the Hood of the OCaml MOOC. *Proc. ACM Program. Lang.* 1, ICFP, Article 4 (aug 2017), 25 pages. https://doi.org/10.1145/3110248

[30] Brent Carmer and Mike Rosulek. 2015. Vamonos: Embeddable visualizations of advanced algorithms. In *2015 IEEE Frontiers in Education Conference (FIE)*. 1–8. https://doi.org/10.1109/FIE.2015.7344263

[31] Alana Ceci, Hanneli C. A. Tavante, Brigitte Pientka, and Xujie Si. 2021. Data Collection for the Learn-OCaml Programming Platform. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. ACM. https://doi.org/10.1145/3408877.3439579

[32] Sumie Chan and Noble Lo. 2021. Impacts of Gamification in Remote Learning in Tertiary Education *(ICSLT 2021)*. Association for Computing Machinery, New York, NY, USA, 26–34. https://doi.org/10.1145/3477282.3477290

[33] Zhong Chang, Yan Sun, Tin-Yu Wu, and Mohsen Guizani. 2018. Scratch analysis Tool (SAT): a modern scratch project analysis tool based on ANTLR to assess computational thinking skills. In *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE, 950–955.

[34] Hayden Cheers, Yuqing Lin, and Shamus P. Smith. 2021. Academic source code plagiarism detection by measuring program behavioral similarity. *IEEE Access* 9 (2021). https://doi.org/10.1109/ACCESS.2021.3069367

[35] Rohan Roy Choudhury, HeZheng Yin, Joseph Moghadam, and Armando Fox. 2016. AutoStyle: Toward Coding Style Feedback At Scale. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion* (San Francisco, California, USA) *(CSCW '16 Companion)*. Association for Computing Machinery, New York, NY, USA, 21–24. https://doi.org/10.1145/2818052.2874315

[36] Georgina Cosma and Mike Joy. 2008. Towards a definition of source-code plagiarism. *IEEE Transactions on Education* 51, 2 (May 2008). https://doi.org/10.1109/TE.2007.906776

[37] Tim Coughlan, Rebecca Pitt, and Patrick McAndrew. 2013. Building Open Bridges: Collaborative Remixing and Reuse of Open Educational Resources across Organisations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 991–1000. https://doi.org/10.1145/2470654.2466127

[38] W Bruce Croft, Donald Metzler, and Trevor Strohman. 2010. *Search Engines: Information Retrieval in Practice*.

[39] James Cross, Dean Hendrix, and David Umphress. 2014. Dynamic Program Visualizations for Java (Abstract Only). In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA) *(SIGCSE '14)*. Association for Computing Machinery, New York, NY, USA, 749–750. https://doi.org/10.1145/2538862.2539028

[40] Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Steve Cooper. 2012. Mediated Transfer: Alice 3 to Java. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (Raleigh, North Carolina, USA) *(SIGCSE '12)*. Association for Computing Machinery, New York, NY, USA, 141–146. https://doi.org/10.1145/2157136.2157180

[41] Balakrishnan Dasarathy, Kevin Sullivan, Douglas C. Schmidt, Douglas H. Fisher, and Adam Porter. 2014. The Past, Present, and Future of MOOCs and Their Relevance to Software Engineering. In *Future of Software Engineering Proceedings* (Hyderabad, India) *(FOSE 2014)*. Association for Computing Machinery, New York, NY, USA, 212–224. https://doi.org/10.1145/2593882.2593897

[42] Roland DePratti. 2020. Jupyter Notebooks versus a Textbook in a Big Data Course. *J. Comput. Sci. Coll.* 35, 8 (apr 2020), 208–220.

[43] Martin Dougiamas and Peter C Taylor. 2002. Interpretive Analysis of an Internet-based Course Constructed using a New Courseware Tool called Moodle. In *2nd Conference of HERDSA (The Higher Education Research and Development Society of Australasia)*. 7–10.

[44] Bob Edmison, Stephen H. Edwards, Margaret Ellis, Lujean Babb, Chris Mayfield, Nick Swanye, Youna Jung, and Marthe Honts. 2023. Toward a New State-level Framework for Sharing Computer Science Content. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education* (Toronto, ON, Canada) *(SIGCSE '23)*. Association for Computing Machinery, New York, NY, USA.

[45] Stephen H Edwards. 2003. Using test-driven development in the classroom: Providing students with concrete feedback on performance. In *Proceedings of the International Conference on Education and Information Systems: Technologies and Applications (EISTA'03)*.

[46] Stephen H. Edwards. 2014. Work-in-Progress: Program Grading and Feedback Generation with Web-CAT. In *First ACM Conference on Learning @ Scale Conference*. ACM, 215–216. https://doi.org/10.1145/2556325.2567888

[47] Margaret Ellis, Clifford A. Shaffer, and Stephen H. Edwards. 2019. Approaches for Coordinating ETextbooks, Online Programming Practice, Automated Grading, and More into One Course. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 126–132. https://doi.org/10.1145/3287324.3287487

[48] Anatalia N. Endozo, Solomon Oluyinka, and Richard G. Daenos. 2019. Teachers' Experiences towards Usage of Learning Management System: CANVAS. In *Proceedings of the 2019 11th International Conference on Education Technology and Computers* (Amsterdam, Netherlands) *(ICETC 2019)*. Association for Computing Machinery, New York, NY, USA, 91–95. https://doi.org/10.1145/3369255.3369257

[49] Annette Feng, Mark Gardner, and Wu chun Feng. 2017. Parallel programming with pictures is a Snap! *J. Parallel and Distrib. Comput.* 105 (2017), 150–162. https://doi.org/10.1016/j.jpdc.2017.01.018 Keeping up with Technology: Teaching Parallel, Distributed and High-Performance Computing.

[50] Louise P Flannery, Brian Silverman, Elizabeth R Kazakoff, Marina Umaschi Bers, Paula Bontá, and Mitchel Resnick. 2013. Designing ScratchJr: Support for early childhood learning through computer programming. In *Proceedings of the 12th international conference on interaction design and children*. 1–10.

[51] George E. Forsythe and Niklaus Wirth. 1965. Automatic Grading Programs. *Commun. ACM* 8, 5 (may 1965), 275–278. https://doi.org/10.1145/364914.364937

[52] Robert Fraser. 2014. Collaboration, collusion and plagiarism in computer science coursework. *Informatics in Education* 13, 2 (2014). https://doi.org/10.15388/infedu.2014.10

[53] Michelle Friend, Michael Matthews, Victor Winter, Betty Love, Deanna Moisset, and Ian Goodwin. 2018. Bricklayer: elementary students learn math through programming and art. In *Proceedings of the 49th ACM technical symposium on computer science education*. 628–633.

[54] Patrice Frison. 2015. A Teaching Assistant for Algorithm Construction. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '15)*. Association for Computing Machinery, New York, NY, USA, 9–14. https://doi.org/10.1145/2729094.2742588

[55] Ria Galanos, Michael Ball, John Dougherty, Joe Hummel, and David J. Malan. 2018. Technology We Can't Live Without!, Revisited. 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, 1043–1044. https://doi.org/10.1145/3159450.3159629

[56] Ria Galanos, Whitaker Brand, Sumukh Sridhara, Mike Zamansky, and Evelyn Zayas. 2017. Technology We Can't Live Without! Revisited. 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, 659–660. https://doi.org/10.1145/3017680.3017691

[57] David Galles. 2011. Data Structure Visualizations. https://www.cs.usfca.edu/~galles/visualization/index.html

[58] Dan Garcia, Zelda Allison, Abigail Joseph, David J. Malan, and Kristin Stephens-Martinez. 2022. Technology We Can't Live Without! (COVID-19 Edition). In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1043–1044. https://doi.org/10.1145/3478432.3499221

[59] Dan Garcia, Tiffany Barnes, Art Lopez, Chinma Uche, and Jill Westerlund. 2021. Technology We Can't Live Without!, Revisited. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 138–139. https://doi.org/10.1145/3408877.3432571

[60] Daniel D. Garcia, Leslie Aaronson, Shawn Kenner, Colleen Lewis, and Susan Rodger. 2016. Technology We Can't Live Without!, Revisited. 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, 236–237. https://doi.org/10.1145/2839509.2844668

[61] Daniel D. Garcia, Eric Allatta, Manuel Pérez-Quiñones, and Jeff Solin. 2015. Technology We Can't Live Without!. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, Missouri, USA) *(SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 597–598. https://doi.org/10.1145/2676723.2677336

[62] Carlos Garcia Sanchez, Fernando Castro, Jose Ignacio Gomez, Christian Tenllado, Daniel Chaver, and Jose Antonio Lopez-Orozco. 2012. OpenIRS-UCM: An Open-Source Multi-Platform for Interactive Response Systems. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (Haifa, Israel) *(ITiCSE '12)*. Association for Computing Machinery, New York, NY, USA, 232–237. https://doi.org/10.1145/2325296.2325352

[63] Eric Gilbert. 2015. Open Book: A Socially-Inspired Cloaking Technique That Uses Lexical Abstraction to Transform Messages. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 477–486. https://doi.org/10.1145/2702123.2702295

[64] David Gitchell and Nicholas Tran. 1999. Sim: a utility for detecting similarity in computer programs. In *30th SIGCSE Technical Symposium on Computer Science Education*. https://doi.org/10.1145/299649.299783

[65] Richard Glassey and Simon Larsén. 2020. Towards Flexible and Extensible Git-Based Course Management with RepoBee. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 537–538. https://doi.org/10.1145/3341525.3393999

[66] Lindsay Grace and Bob Hone. 2019. Factitious: Large Scale Computer Game to Fight Fake News and Improve News Literacy. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI EA '19)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3290607.3299046

[67] Emily Gubski and Steven Wolfman. 2020. Jupyter/Canvas Submission Framework Integration. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM. https://doi.org/10.1145/3328778.3372670

[68] Ma. Janice J. Gumasing, Abigail B. Vasquez, Angelo Luis S. Doctora, and William Davin D. Perez. 2022. Usability Evaluation of Online Learning Management System: Comparison between Blackboard and Canvas. In *2022 The 9th International Conference on Industrial Engineering and Applications (Europe)* (Barcelona, Spain) *(ICIEA-2022-Europe)*. Association for Computing Machinery, New York, NY, USA, 25–31. https://doi.org/10.1145/3523132.3523137

[69] Philip J Guo. 2013. 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)*. Association for Computing Machinery, New York, NY, USA, 579–584. https://doi.org/10.1145/2445196.2445368

[70] Steven Halim, Zi Chun Koh, Victor Bo, Huai Loh, and Felix Halim. 2012. Learning Algorithms with Unified and Interactive Web-Based Visualization. , 53-68 pages. http://www.comp.nus.edu.sg/

[71] Brian Harvey, Daniel D. Garcia, Tiffany Barnes, Nathaniel Titterton, Daniel Armendariz, Luke Segars, Eugene Lemon, Sean Morris, and Josh Paley. 2013. SNAP! (Build Your Own Blocks) (Abstract Only). In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 759. https://doi.org/10.1145/2445196.2445507

[72] Brian Harvey, Daniel D Garcia, Tiffany Barnes, Nathaniel Titterton, Daniel Armendariz, Luke Segars, Eugene Lemon, Sean Morris, and Josh Paley. 2013. Snap!(build your own blocks). In *Proceeding of the 44th ACM technical symposium on Computer science education*. 759–759.

[73] H. Hashim, N. A. Salim, and M. Kassim. 2018. Students' Response on Implementation of Kahoot in the Classroom. In *2018 IEEE 10th International Conference on Engineering Education (ICEED)*. 1–4. https://doi.org/10.1109/ICEED.2018.8626899

[74] Wilhelm Hasselbring, Alexander Krause, and Christian Zirkelbach. 2020. ExplorViz: Research on software visualization, comprehension and collaboration. *Software Impacts* 6 (nov 2020), 100034. https://doi.org/10.1016/j.simpa.2020.100034

[75] Felienne Hermans. 2020. Hedy: A Gradual Language for Programming Education. In *ACM Conference on International Computing Education Research*. https://doi.org/10.1145/3372782.3406262

[76] Jack Hollingsworth. 1960. Automatic Graders for Programming Classes. *Commun. ACM* 3, 10 (oct 1960), 528–529. https://doi.org/10.1145/367415.367422

[77] Caitlin Holman, Stephen Aguilar, and Barry Fishman. 2013. GradeCraft: What Can We Learn from a Game-Inspired Learning Management System?. In *Proceedings of the Third International Conference on Learning Analytics and Knowledge* (Leuven, Belgium) *(LAK '13)*. Association for Computing Machinery, New York, NY, USA, 260–264. https://doi.org/10.1145/2460296.2460350

[78] Patrick Hung, Jeanne Lam, Chris Wong, and Tyrone Chan. 2015. A Study on Using Learning Management System with Mobile App. In *2015 International Symposium on Educational Technology (ISET)*. 168–172. https://doi.org/10.1109/ISET.2015.41

[79] Sheung-Lun Hung, Iam-For Kwok, and Raymond Chan. 1993. Automatic Programming Assessment. *Computers & Education* 20, 2 (1993), 183–190. https://doi.org/10.1016/0360-1315(93)90086-X

[80] Ryosuke Ishizue, Kazunori Sakamoto, Hironori Washizaki, and Yoshiaki Fukazawa. 2018. PVC: Visualizing C Programs on Web Browsers for Novices. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 245–250. https://doi.org/10.1145/3159450.3159566

[81] Julia Isong. 2001. Developing an Automated Program Checkers. *Journal of Computing Sciences in Colleges* 16, 3 (2001), 218–224.

[82] David Jackson. 1996. A Software System for Grading Student Computer Programs. *Computers & Education* 27, 3-4 (1996), 171–180. https://doi.org/10.1016/S0360-1315(96)00025-5

[83] Daniel Jackson. 2012. *Software Abstractions: logic, language, and analysis*. MIT press.

[84] S. Jayaraman, B Jayaraman, and D Lessa. 2017. Compact visualization of Java program execution. *Software: Practice and Experience* 47, 2 (feb 2017), 163–191. https://doi.org/10.1002/spe.2411

[85] L Jiang, G Misherghi, Z Su, and S Glondu. 2007. DECKARD: scalable and accurate tree-based detection of code clones. In *29th International Conference on Software Engineering*. https://doi.org/10.1109/ICSE.2007.30

[86] Mike Joy, Nathan Griffiths, and Russell Boyatt. 2005. The BOSS online submission and assessment system. *Journal on Educational Resources in Computing* 5, 3, Article 2 (Sep 2005). https://doi.org/10.1145/1163405.1163407

[87] Mike Joy and Michael Luck. 1999. Plagiarism in programming assignments. *IEEE Transactions on Education* 42, 2 (1999). https://doi.org/10.1109/13.762946

[88] An Ju, Ben Mehne, Andrew Halle, and Armando Fox. 2018. In-class coding-based summative assessments: tools, challenges, and experience. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. ACM. https://doi.org/10.1145/3197091.3197094

[89] An Ju, Ben Mehne, Andrew Halle, and Armando Fox. 2018. In-Class Coding-Based Summative Assessments: Tools, Challenges, and Experience. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018)*. Association for Computing Machinery, New York, NY, USA, 75–80. https://doi.org/10.1145/3197091.3197094

[90] Garvit Juniwal, Sakshi Jain, Alexandre Donzé, and Sanjit A. Seshia. 2015. Clustering-Based Active Learning for CPSGrader. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*. ACM. https://doi.org/10.1145/2724660.2728702

[91] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. 2002. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering* 28, 7 (2002). https://doi.org/10.1109/TSE.2002.1019480

[92] Hyeonsu Kang and Philip J. Guo. 2017. Omnicode: A Novice-Oriented Live Programming Environment with Always-On Run-Time Value Visualizations. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (Québec City, QC, Canada) *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 737–745. https://doi.org/10.1145/3126594.3126632

[93] Ville Karavirta and Clifford A Shaffer. 2013. JSAV: The JavaScript Algorithm Visualization Library. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '13)*. Association for Computing Machinery, New York, NY, USA, 159–164. https://doi.org/10.1145/2462476.2462487

[94] Oscar Karnalim and Simon. 2021. Explanation in code similarity investigation. *IEEE Access* 9 (2021). https://doi.org/10.1109/ACCESS.2021.3073703

[95] Oscar Karnalim, Simon, and William Chivers. 2019. Similarity detection techniques for academic source code plagiarism and collusion: a review. In *IEEE International Conference on Engineering, Technology and Education*. https://doi.org/10.1109/TALE48000.2019.9225953

[96] Oscar Karnalim, Simon, and William Chivers. 2022. Layered similarity detection for programming plagiarism and collusion on weekly assessments. *Computer Applications in Engineering Education* In press (2022). https://doi.org/10.1002/cae.22553

[97] Spencer Killen, Evan Giese, Huy Huynh, and Indratmo. 2017. Marble MLFQ: An educational visualization tool for the multilevel feedback queue algorithm. In *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. 663–669. https://doi.org/10.1109/IEMCON.2017.8117201

[98] Carlos Delgado Kloos, Carmen Fernández-Panadero, Carlos Alario-Hoyos, Pedro Manuel Moreno-Marcos, María Blanca Ibáñez, Pedro J. Muñoz-Merino, Boni García, and Iria Estévez-Ayres. 2022. Programming Teaching Interaction. In *2022 IEEE Global Engineering Education Conference (EDUCON)*. 1965–1969. https://doi.org/10.1109/EDUCON52537.2022.9766697

[99] Tobias Kohn and Bill Manaris. 2020. Tell Me What's Wrong: A Python IDE with Error Messages. 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, 1054–1060. https://doi.org/10.1145/3328778.3366920

[100] Line Kolås. 2015. Application of interactive videos in education. In *2015 International Conference on Information Technology Based Higher Education and Training (ITHET)*. 1–6. https://doi.org/10.1109/ITHET.2015.7218037

[101] Nils Kopal, Olga Kieselmann, Arno Wacker, and Bernhard Esslinger. 2014. CrypTool 2.0. *Datenschutz und Datensicherheit - DuD* 38, 10 (2014), 701–708. https://doi.org/10.1007/s11623-014-0274-7

[102] Gregor Kotainy and Olaf Spinczyk. 2014. AnimOS CPU-Scheduling. https://ess.cs.tu-dortmund.de/Software/AnimOS/CPU-Scheduling/

[103] Andrew Kuo, Lexseal Lin, Diana Issatayeva, and Mia Minnes. 2022. Flap.js. https://github.com/flapjs

[104] Stan Kurkovsky. 2022. Using Scaffolding to Simplify FOSS Adoption *(ITiCSE '22)*. Association for Computing Machinery, New York, NY, USA, 587–588. https://doi.org/10.1145/3502717.3532163

[105] Thomas Lancaster and Fintan Culwin. 2004. A comparison of source code plagiarism detection engines. *Computer Science Education* 14, 2 (2004). https://doi.org/10.1080/08993400412331363843

[106] Tak Pang Lau, Shuai Wang, Yuanyuan Man, Chi Fai Yuen, and Irwin King. 2014. Language Technologies for Enhancement of Teaching and Learning in Writing. In *Proceedings of the 23rd International Conference on World Wide Web* (Seoul, Korea) *(WWW '14 Companion)*. Association for Computing Machinery, New York, NY, USA, 1097–1102. https://doi.org/10.1145/2567948.2580058

[107] Shu-Sheng Liaw. 2008. Investigating students' perceived satisfaction, behavioral intention, and effectiveness of e-learning: A case study of the Blackboard system. *Computers & Education* 51, 2 (2008), 864–873. https://doi.org/10.1016/j.compedu.2007.09.005

[108] Xiao Liu, Shuai Wang, Pei Wang, and Dinghao Wu. 2019. Automatic Grading of Programming Assignments: An Approach Based on Formal Semantics. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. 126–137. https://doi.org/10.1109/ICSE-SEET.2019.00022

[109] Jun Ma, Jun Tao, Melissa Keranen, Jean Mayo, Ching-Kuang Shene, and Chaoli Wang. 2014. SHAvisual: A Secure Hash Algorithm Visualization Tool. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE '14)*. Association for Computing Machinery, New York, NY, USA, 338. https://doi.org/10.1145/2591708.2602663

[110] Jun Ma, Jun Tao, Jean Mayo, Ching-Kuang Shene, Melissa Keranen, and Chaoli Wang. 2016. AESvisual: A Visualization Tool for the AES Cipher. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. Association for Computing Machinery, New York, NY, USA, 230–235. https://doi.org/10.1145/2899415.2899425

[111] Nuno Macedo, Alcino Cunha, José Pereira, Renato Carvalho, Ricardo Silva, Ana C.R. Paiva, Miguel Sozinho Ramalho, and Daniel Silva. 2021. Experiences on teaching alloy with an automated assessment platform. *Science of Computer Programming* 211 (nov 2021), 102690. https://doi.org/10.1016/j.scico.2021.102690

[112] Salvador John M. Magalong and Brando C. Palomar. 2019. Effects of Flipped Classroom Approach Using Gooru Learning Management System on Students' Physics Achievement. In *Proceedings of the 10th International Conference on E-Education, E-Business, E-Management and E-Learning* (Tokyo, Japan) *(IC4E '19)*. Association for Computing Machinery, New York, NY, USA, 75–78. https://doi.org/10.1145/3306500.3306540

[113] Jyotirmaya Mahapatra, Saurabh Srivastava, Kuldeep Yadav, Kundan Shrivastava, and Om Deshmukh. 2016. LMS Weds WhatsApp: Bridging Digital Divide Using MIMs. In *Proceedings of the 13th International Web for All Conference* (Montreal, Canada) *(W4A '16)*. Association for Computing Machinery, New York, NY, USA, Article 42, 4 pages. https://doi.org/10.1145/2899475.2899485

[114] Evan Maicus, Matthew Peveler, Andrew Aikens, and Barbara Cutler. 2020. Autograding Interactive Computer Graphics Applications. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM. https://doi.org/10.1145/3328778.3366954

[115] Evan Maicus, Matthew Peveler, Stacy Patterson, and Barbara Cutler. 2019. Autograding Distributed Algorithms in Networked Containers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. ACM. https://doi.org/10.1145/3287324.3287505

[116] Srikesh Mandala and Kevin A. Gary. 2013. Distributed Version Control for Curricular Content Management. In *2013 IEEE Frontiers in Education Conference (FIE)*. 802–804. https://doi.org/10.1109/FIE.2013.6684936

[117] Hamza Manzoor, Amit Naik, Clifford A. Shaffer, Chris North, and Stephen H. Edwards. 2020. Auto-Grading Jupyter Notebooks. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM. https://doi.org/10.1145/3328778.3366947

[118] Hamza Manzoor, Amit Naik, Clifford A. Shaffer, Chris North, and Stephen H. Edwards. 2020. *Auto-Grading Jupyter Notebooks*. Association for Computing Machinery, New York, NY, USA, 1139–1144. https://doi.org/10.1145/3328778.3366947

[119] Fenwick McKelvey and Robert Hunt. 2019. Discoverability: Toward a Definition of Content Discovery Through Platforms. *Social Media+ Society* 5, 1 (2019). https://doi.org/10.1177%2F2056305118819188

[120] Dhruv Misra. 2020. Pathfinding Visualizer in 3D. https://github.com/dhruvmisra/Pathfinding-Visualizer-ThreeJS%0A

[121] Guillaume de Moffarts and Sébastien Combéfis. 2020. Challengr, a Classroom Response System for Competency Based Assessment and Real-Time Feedback with Micro-Contests. In *2020 IEEE Frontiers in Education Conference (FIE)*. 1–4. https://doi.org/10.1109/FIE44824.2020.9273992

[122] Mostafa Mohammed, Piexuan Ge, Samnyeong Heo, and Clifford A. Shaffer. 2021. Support for Programmed Instruction in an eTextbook. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. ACM. https://doi.org/10.1145/3408877.3439586

[123] György Molnár and David Sik. 2020. The virtual toolkit of digital instruction and its application in digital work forms. In *2020 11th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*. 000597–000600. https://doi.org/10.1109/CogInfoCom50765.2020.9237855

[124] Alvaro E Monge, Cameron L Fadjo, Beth A Quinn, and Lecia J Barker. 2015. EngageCSEdu: engaging and retaining CS1 and CS2 students. *ACM Inroads* 6, 1 (2015), 6–11.

[125] Pedro Moraes and Leopoldo Teixeira. 2019. Willow: A Tool for Interactive Programming Visualization to Help in the Data Structures and Algorithms Teaching-Learning Process. In *Proceedings of the XXXIII Brazilian Symposium on*

Software Engineering (SBES 2019). Association for Computing Machinery, New York, NY, USA, 553–558. https://doi.org/10.1145/3350768.3351303

[126] Andrés Moreno, Niko Myller, Erkki Sutinen, and Mordechai Ben-Ari. 2004. Visualizing Programs with Jeliot 3. In *Working Conference on Advanced Visual Interfaces*. 373–376. https://doi.org/10.1145/989863.989928

[127] Jesús Moreno-León and Gregorio Robles. 2015. Dr. Scratch: A web tool to automatically evaluate Scratch projects. In *Proceedings of the workshop in primary and secondary computing education*. 132–133.

[128] Nate Murray and Ari Lerner. 2013. Choc Traceable Programming. https://www.newline.co/choc/

[129] Matija Novak, Mike Joy, and Dragutin Kermek. 2019. Source-code similarity detection and detection tools used in academia: a systematic review. *ACM Transactions on Computing Education* 19, 3 (2019). https://doi.org/10.1145/3313290

[130] Hidehisa Oku, Kayoko Matsubara, and Masayuki Booka. 2013. Feasibility Study of PDF Based Digital Textbooks for University Students with Difficulty to Handle Print Textbooks. In *Proceedings of the 7th International Convention on Rehabilitation Engineering and Assistive Technology* (Gyeonggi-do, South Korea) *(i-CREATe '13)*. Singapore Therapeutic, Assistive & Rehabilitative Technologies (START) Centre, Midview City, SGP, Article 14, 4 pages.

[131] Rohan Padhye, Koushik Sen, and Paul N. Hilfinger. 2019. ChocoPy: a programming language for compilers courses. In *Proceedings of the 2019 ACM SIGPLAN Symposium on SPLASH-E - SPLASH-E 2019*. ACM Press. https://doi.org/10.1145/3358711.3361627

[132] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2022. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Trans. Comput. Educ.* 22, 3, Article 34 (jun 2022), 40 pages. https://doi.org/10.1145/3513140

[133] Bruno Papa. 2020. Recursion Tree Visualizer. https://recursion.vercel.app/

[134] Panagiotis Papadopoulos, Nicolas Kourtellis, Pablo Rodriguez Rodriguez, and Nikolaos Laoutaris. 2017. If You Are Not Paying for It, You Are the Product: How Much Do Advertisers Pay to Reach You?. In *Internet Measurement Conference*. ACM, 142–156. https://doi.org/10.1145/3131365.3131397

[135] Diane Peters. 2021. *Learning management systems are more important than ever*. Retrieved 2022-07-08 from https://www.universityaffairs.ca/features/feature-article/learning-management-systems-are-more-important-than-ever/

[136] Matthew Peveler, Tushar Gurjar, Evan Maicus, Andrew Aikens, Alexander Christoforides, and Barbara Cutler. 2019. Lichen: customizable, open source plagiarism detection in Submitty. In *50th ACM Technical Symposium on Computer Science Education*. https://doi.org/10.1145/3287324.3293867

[137] Matthew Peveler, Evan Maicus, and Barbara Cutler. 2019. Comparing Jailed Sandboxes vs Containers Within an Autograding System. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. ACM. https://doi.org/10.1145/3287324.3287507

[138] M. Pigultong. 2022. Cognitive Impacts of Using a Metaverse embedded on Learning Management System for Students with Unequal Access to Learning Resources. In *2022 10th International Conference on Information and Education Technology (ICIET)*. 27–31. https://doi.org/10.1109/ICIET55102.2022.9779045

[139] Kerttu Pollari-Malmi, Julio Guerra, Peter Brusilovsky, Lauri Malmi, and Teemu Sirkiä. 2017. On the Value of Using an Interactive Electronic Textbook in an Introductory Programming Course. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '17)*. Association for Computing Machinery, New York, NY, USA, 168–172. https://doi.org/10.1145/3141880.3141890

[140] James F. Power and John Waldron. 2020. Calibration and analysis of source code similarity measures for Verilog hardware description language projects. In *51st ACM Technical Symposium on Computer Science Education*. https://doi.org/10.1145/3328778.3366928

[141] Prajish Prasad and Sridhar Iyer. 2020. VeriSIM: A Learning Environment for Comprehending Class and Sequence Diagrams Using Design Tracing. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training* (Seoul, South Korea) *(ICSE-SEET '20)*. Association for Computing Machinery, New York, NY, USA, 23–33. https://doi.org/10.1145/3377814.3381705

[142] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. 2002. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science* 8, 11 (2002).

[143] Nick Rahimi and Nancy L. Martin. 2020. Challenges and Strategies for Online Teaching in Information Technology and Other Computing Programs. In *Proceedings of the 21st Annual Conference on Information Technology Education* (Virtual Event, USA) *(SIGITE '20)*. Association for Computing Machinery, New York, NY, USA, 218–222. https://doi.org/10.1145/3368308.3415369

[144] Robert Ravenscroft. 2018. An HTML5 Browser Application for Modeling and Teaching Linked Lists: (Abstract Only). In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 1106. https://doi.org/10.1145/3159450.3162232

[145] Saquib Razak, Huda Gedawy, Wanda P Dann, and Donald J Slater. 2016. Alice in the Middle East: An Experience Report from the Formative Phase. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 425–430. https://doi.org/10.1145/2839509.2844593

[146] Jake Renzella and Andrew Cain. 2020. Enriching Programming Student Feedback with Audio Comments. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training* (Seoul, South Korea) *(ICSE-SEET '20)*. Association for Computing Machinery, New York, NY, USA, 173–183. https://doi.org/10.1145/3377814.3381712

[147] Jake Renzella, Andrew Cain, and Jean-Guy Schneider. 2021. Real Talk: Illuminating Online Student Understanding with Authentic Discussion Tools. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 886–892. https://doi.org/10.1145/3408877.3432484

[148] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (Nov 2009). https://doi.org/10.1145/1592761.1592779

[149] William Robinson. 2016. From scratch to patch: Easing the blocks-text transition. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*. 96–99.

[150] Virginia Rodés, Pollyana Notargiacomo Mustaro, Ismar Frango Silveira, Nizam Omar, and Xavier Ochôa. 2014. Instructional Design Models to Support Collaborative Open Books for Open Education. In *Proceedings of the XV International Conference on Human Computer Interaction* (Puerto de la Cruz, Tenerife, Spain) *(Interacción '14)*. Association for Computing Machinery, New York, NY, USA, Article 93, 7 pages. https://doi.org/10.1145/2662253.2662346

[151] Susan H Rodger, Julian Genkins, Ian McMahon, and Peggy Li. 2013. Increasing the Experimentation of Theoretical Computer Science with New Features in JFLAP. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '13)*. Association for Computing Machinery, New York, NY, USA, 351. https://doi.org/10.1145/2462476.2466521

[152] Guido Rößling, Markus Schüler, and Bernd Freisleben. 2000. The ANIMAL Algorithm Animation Tool. In *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSE conference on Innovation and Technology in Computer Science Education (ITiCSE '00)*. Association for Computing Machinery, New York, NY, USA, 37–40. https://doi.org/10.1145/343048.343069

[153] Palma Rozalia Osztian, Zoltan Katai, and Erika Osztian. 2020. Algorithm Visualization Environments: Degree of interactivity as an influence on student-learning. In *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–8. https://doi.org/10.1109/FIE44824.2020.9273892

[154] Seán Russell. 2021. Automatically Generated and Graded Program Tracing Quizzes with Feedback. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 2* (Virtual Event, Germany) *(ITiCSE '21)*. Association for Computing Machinery, New York, NY, USA, 652. https://doi.org/10.1145/3456565.3460054

[155] Seán Russell. 2022. Automated Code Tracing Exercises for CS1. In *Computing Education Practice 2022* (Durham, United Kingdom) *(CEP 2022)*. Association for Computing Machinery, New York, NY, USA, 13–16. https://doi.org/10.1145/3498343.3498347

[156] Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko, Chanchal K. Roy, and Cristina V. Lopes. 2016. Sourcerercc: scaling code clone detection to big-code. In *38th International Conference on Software Engineering*. https://doi.org/10.1145/2884781.2884877

[157] Avneesh Sarwate, Creston Brunch, Jason Freeman, and Sebastian Siva. 2018. Grading at scale in earsketch. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*. ACM. https://doi.org/10.1145/3231644.3231708

[158] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. 2003. Winnowing: local algorithms for document fingerprinting. In *International Conference on Management of Data*. https://doi.org/10.1145/872757.872770

[159] Patrick Seeling. 2015. Assessing student views of traditional, free, and interactive modifications for an introductory networking course. In *2015 IEEE Frontiers in Education Conference (FIE)*. 1–4. https://doi.org/10.1109/FIE.2015.7344286

[160] Clifford A. Shaffer. 2016. OpenDSA: An Interactive ETextbook for Computer Science Courses. 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, 5. https://doi.org/10.1145/2839509.2850505

[161] R Benjamin Shapiro, Annie Kelly, Matthew Ahrens, and Rebecca Fiebrink. 2016. BlockyTalky: A physical and distributed computer music toolkit for kids. NIME.

[162] Parisa Shayan, Roberto Rondinelli, Menno van Zaanen, and Martin Atzmueller. 2019. Descriptive Network Modeling and Analysis for Investigating User Acceptance in a Learning Management System Context. In *Proceedings of the 23rd International Workshop on Personalization and Recommendation on the Web and Beyond* (Hof, Germany) *(ABIS '19)*. Association for Computing Machinery, New York, NY, USA, 7–13. https://doi.org/10.1145/3345002.3349288

[163] Kristi J. Shryock. 2015. Engaging students inside the classroom to increase learning. In *2015 IEEE Frontiers in Education Conference (FIE)*. 1–7. https://doi.

org/10.1109/FIE.2015.7344076

[164] Angela A. Siegel, Mark Zarb, Bedour Alshaigy, Jeremiah Blanchard, Tom Crick, Richard Glassey, John R. Hott, Celine Latulipe, Charles Riedesel, Mali Senapathi, Simon, and David Williams. 2022. Teaching through a Global Pandemic: Educational Landscapes Before, During and After COVID-19. In *Proceedings of the 2021 Working Group Reports on Innovation and Technology in Computer Science Education* (Virtual Event, Germany) *(ITiCSE-WGR '21)*. Association for Computing Machinery, New York, NY, USA, 1–25. https://doi.org/10.1145/3502870.3506565

[165] Simon, Oscar Karnalim, Judy Sheard, Ilir Dema, Amey Karkare, Juho Leinonen, Michael Liut, and Renée McCauley. 2020. Choosing Code Segments to Exclude from Code Similarity Detection. In *ACM Working Group Reports on Innovation and Technology in Computer Science Education*. https://doi.org/10.1145/3437800.3439201

[166] Teemu Sirkiä. 2016. Jsvee & Kelmu: Creating and Tailoring Program Animations for Computing Education. In *2016 IEEE Working Conference on Software Visualization (VISSOFT)*. 36–45. https://doi.org/10.1109/VISSOFT.2016.24

[167] Teemu Sirkiä and Juha Sorva. 2012. Exploring Programming Misconceptions: An Analysis of Student Mistakes in Visual Program Simulation Exercises. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '12)*. Association for Computing Machinery, New York, NY, USA, 19–28. https://doi.org/10.1145/2401796.2401799

[168] Sebastien Siva, Tacksoo Im, Tom McKlin, Jason Freeman, and Brian Magerko. 2018. Using Music to Engage Students in an Introductory Undergraduate Programming Course for Non-Majors. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM. https://doi.org/10.1145/3159450.3159468

[169] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education* 13, 4 (Nov 2013). https://doi.org/10.1145/2490822

[170] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Trans. Comput. Educ.* 13, 4, Article 15 (nov 2013), 64 pages. https://doi.org/10.1145/2490822

[171] Alina Striner, Andrew M. Webb, Jessica Hammer, and Amy Cook. 2021. Mapping Design Spaces for Audience Participation In Game Live Streaming. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 329, 15 pages. https://doi.org/10.1145/3411764.3445511

[172] Filip Strömbäck, Linda Mannila, and Mariam Kamkar. 2022. Pilot Study of Progvis: A Visualization Tool for Object Graphs and Concurrency via Shared Memory. In *Australasian Computing Education Conference* (Virtual Event, Australia) *(ACE '22)*. Association for Computing Machinery, New York, NY, USA, 123–132. https://doi.org/10.1145/3511861.3511885

[173] Sari Sultan and Ayed Salman. 2019. Automatically Generating Exams via Programmable Plug-Ins, and Generic XML Exam Support. In *Proceedings of the 10th International Conference on E-Education, E-Business, E-Management and E-Learning* (Tokyo, Japan) *(IC4E '19)*. Association for Computing Machinery, New York, NY, USA, 184–188. https://doi.org/10.1145/3306500.3306504

[174] Marek Šuppa, Ondrej Jariabka, Adrián Matejov, and Marek Nagy. 2021. TermAdventure: Interactively Teaching UNIX Command Line, Text Adventure Style. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. ACM. https://doi.org/10.1145/3430665.3456387

[175] Jun Tao, Jun Ma, Melissa Keranen, Jean Mayo, and Ching-Kuang Shene. 2012. ECvisual: A Visualization Tool for Elliptic Curve Based Ciphers. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)*. Association for Computing Machinery, New York, NY, USA, 571–576. https://doi.org/10.1145/2157136.2157298

[176] Jun Tao, Jun Ma, Melissa Keranen, Jean Mayo, Ching-Kuang Shene, and Chaoli Wang. 2014. RSAvisual: A Visualization Tool for the RSA Cipher. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. Association for Computing Machinery, New York, NY, USA, 635–640. https://doi.org/10.1145/2538862.2538891

[177] Jun Tao, Jun Ma, Jean Mayo, Ching-Kuang Shene, and Melissa Keranen. 2011. DESvisual: A Visualization Tool for the DES Cipher. *J. Comput. Sci. Coll.* 27, 1 (oct 2011), 81–89.

[178] James D. Teresco. 2012. Highway Data and Map Visualizations for Educational Use. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (Raleigh, North Carolina, USA) *(SIGCSE '12)*. Association for Computing Machinery, New York, NY, USA, 553–558. https://doi.org/10.1145/2157136.2157295

[179] Brian Thoms and Evren Eryilmaz. 2018. Social Software Design To Facilitate Service-Learning In Interdisciplinary Computer Science Courses. 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, 497–502. https://doi.org/10.1145/3159450.3159572

[180] Artturi Tilanterä, Giacomo Mariani, Ari Korhonen, and Otto Seppälä. 2021. Towards a JSON-based Algorithm Animation Language. In *2021 Working Conference on Software Visualization (VISSOFT)*. 135–139. https://doi.org/10.1109/VISSOFT52517.2021.00026

[181] Jake Trower and Jeff Gray. 2015. Blockly Language Creation and Applications: Visual Programming for Media Computation and Bluetooth Robotics Control. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, Missouri, USA) *(SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 5. https://doi.org/10.1145/2676723.2691871

[182] Paul J. Wagner. 2020. The SQL File Evaluation (SQLFE) Tool. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM. https://doi.org/10.1145/3328778.3372599

[183] Jane Waite, Andrea Franceschini, Sue Sentance, Mathew Patterson, and James Sharkey. 2021. *An Online Platform for Teaching Upper Secondary School Computer Science*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3481282.3481287

[184] Wengran Wang, Chenhao Zhang, Andreas Stahlbauer, Gordon Fraser, and Thomas Price. 2021. SnapCheck: Automated Testing for Snap! Programs. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. ACM. https://doi.org/10.1145/3430665.3456367

[185] Ye Diana Wang and Seungwon "Shawn" Lee. 2013. Embedding Virtual Meeting Technology in Classrooms: Two Case Studies. In *Proceedings of the 14th Annual ACM SIGITE Conference on Information Technology Education* (Orlando, Florida, USA) *(SIGITE '13)*. Association for Computing Machinery, New York, NY, USA, 83–90. https://doi.org/10.1145/2512276.2512279

[186] Eliane S. Wiese, Michael Yen, Antares Chen, Lucas A. Santos, and Armando Fox. 2017. Teaching Students to Recognize and Implement Good Coding Style. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*. ACM. https://doi.org/10.1145/3051457.3051469

[187] Michael J Wise. 1996. YAP3: improved detection of similarities in computer program and other texts. In *27th SIGCSE Technical Symposium on Computer Science Education*. https://doi.org/10.1145/236452.236525

[188] Billy T.M. Wong and Kam Cheong Li. 2019. Using Open Educational Resources for Teaching in Higher Education: A Review of Case Studies. In *2019 International Symposium on Educational Technology (ISET)*. 186–190. https://doi.org/10.1109/ISET.2019.00046

[189] Bruce Worobec and Robert Bryant. 2016. Using Sharepoint as a Limited Learning Management System. *J. Comput. Sci. Coll.* 32, 2 (dec 2016), 11–18.

[190] Fu-Yun Yu and Yu-Hsin Liu. 2015. Social Media as a Teaching and Learning Tool for In-class Q&A Activities to Promote Learning and Transform College Engineering Classroom Dynamics: The Case of Facebook. In *2015 IEEE 15th International Conference on Advanced Learning Technologies*. 299–300. https://doi.org/10.1109/ICALT.2015.11

[191] Budi Yulianto, Andyni Khosasih, Evawaty Tanuar, and Yuventia Prisca Diyanti Todalani Kalumbang. 2021. Taman Belajar: Learning Management System (LMS) That Provides Free Massive Open Online Course (MOOC) for School Students. In *2021 4th International Conference on Education Technology Management* (Tokyo, Japan) *(ICETM'21)*. Association for Computing Machinery, New York, NY, USA, 52–58. https://doi.org/10.1145/3510309.3510318

[192] Tiguiane Yélémou, Borlli Michel Jonas Somé, and Wilfried Kiélem. 2018. An Enhanced Moodle-based Learning Management System to Account for Low Bandwidths. In *2018 1st International Conference on Smart Cities and Communities (SCCIC)*. 1–4. https://doi.org/10.1109/SCCIC.2018.8584553

[193] Jeremy K. Zhang, Chao Hsu Lin, Melissa Hovik, and Lauren J. Bricker. 2020. GitGrade. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM. https://doi.org/10.1145/3328778.3372634

A DEMOGRAPHIC SURVEY QUESTIONS

Table 8 lists the various optional demographic questions included in our survey.

Table 8: Demographic Survey Questions

Q#	Question	Type
Q26	Institution Level [See Table 9]	Multiple Choice
Q27	Institution Name	Text Entry
Q28	Institution Location	Text Entry
Q29	Stage of Career [Table 10]	Multiple Choice
Q30	What is the approximate distribution of your workload in your role between research, teaching, administration, and other?	Text Entry
Q31	What is the largest class you teach? [See Table 11]	Multiple Choice
Q32	What is your TA to student ratio? That is, for how many students do you get one TA on average?	Text Entry

Table 9: Institution Levels

O#	Option
O1	Primary Education
O2	Secondary Education
O3	Tertiary Education: Undergraduate (Bachelor's) Awarding
O4	Tertiary Education: Master's Awarding
O5	Tertiary Education: PhD Awarding/Other

Table 10: Stage of Career

O#	Option
O1	0-5 years teaching experience
O2	6-15 years teaching experience
O3	16+ years teaching experience

Table 11: Class Size

O#	Option
O1	Small (<50 students)
O2	Medium (50-100 students)
O3	Large (100-199 students)
O4	Very Large (200+ students)

B DISCOVERED AND OTHER KNOWN TOOLS

Table 12 lists the different discovery methods that survey respondents could indicate for the tools they mentioned. For completeness, we list all the discovered tools in the literature review and community survey, along with information on how to access those tools. Tools appear in the list divided by type of tool if they were mentioned by at least 10% of respondents in the survey or were found during the literature review process. Table 13 displays the list of autograders; Table 14 contains the list of plagiarism detectors; development environments and plugins are listed in Table 15; visualization tools are displayed in Table 16; class management and LMS systems are in Table 17; and Table 18 lists the polling and quizzing tools. Since there were fewer open textbook tools, their information has been included in Section 6.6. Additionally, Table 19 lists all tools with links that were included in the survey responses but did not meet the 10% threshold. Finally, Table 20 lists a number of tools that did not appear in the either the literature review or survey but may be useful to the Computing Education community.

Table 12: Discovery categories for tools reported in the community survey

D#	Discovery Method
D1	Searching the internet
D2	From colleagues at other institutions
D3	From colleagues at your institution
D4	"Inherited" when starting on a course
D5	By attending conferences
D6	By reading papers
D7	Built into institution's LMS
D8	I created it

Table 13: Discovered auto-grading tools with two possible sources: literature review (rev) and community survey (sur)

Name	Domain	Link	Source(s)
Alloy4Fun [111]	Alloy Models	http://alloy4fun.inesctec.pt/	rev
AutoGrader [108]	Code	https://github.com/s3team/AutoGrader	rev
AutoStyle [35, 186]	Code	https://github.com/autostyle/autostyle	rev
ChocoPy [131]	Code	https://chocopy.org/	rev
CPSGrader [90]	Code	http://cpsgrader.org/	rev
EarSketch [157, 168]	Code	https://earsketch.gatech.edu/	rev
GitGrade [193]	Code	https://gitgrade.cs.washington.edu/	rev
Gradescope	Code	https://gradescope.com	sur
JupyterCanvasSubmit [67]	Code	https://github.com/eagubsi/JupyterCanvasSubmit	rev
Learn-OCaml [31]	Code	https://github.com/ocaml-sf/learn-ocaml	rev
Mastery Learning Quiz App [15]	Assignment	https://github.com/tiara-allidina7/MasteryLearningQuizApp	rev
OpenDSA Programmed Instruction [122]	Quiz/Practice	https://opendsa-server.cs.vt.edu/	rev
Reveal [88]	Code in Exams	https://github.com/ace-lab/reveal	rev
SnapCheck [184]	Code	https://github.com/emmableu/SnapCheck	rev
SQLFE [182]	Code	https://github.com/wagnerpj42/SQL-File-Evaluation	rev
Submitty [137]	Code	https://submitty.org	rev
Submitty: Graphics Applications [114]	Code	https://submitty.org	rev
Submitty: Distributed Algorithms [115]	Code	https://submitty.org	rev
TermAdventure (2021) [174]	Terminal	https://github.com/NaiveNeuron/TermAdventure	rev
uAssign [12]	Terminal	https://github.com/jakebailey/ua	rev
Web-CAT [45]	Code	https://github.com/web-cat	sur
Web-CAT Jupyter Plugin [117]	Code	https://github.com/web-cat/web-cat-plugin-JupyterPlugin	rev

Table 14: Discovered plagiarism (similarity) detectors with two possible sources: literature review (rev) and community survey (sur).

Name	Domain	Link	Source(s)
AntiCutAndPaste	Code & text	http://www.plagiarism-report.com/anticutandpaste/	rev
BPlag [34]	Code	https://github.com/hjc851/BPlag	rev
CCFinder [91]	Code	http://www.ccfinder.net/	rev
CodeMatch	Code	https://www.safe-corp.com/products_codematch.htm	rev
CodeQuiry	Code & text	https://codequiry.com/	rev
Copy/Paste Detector	Code	https://pmd.github.io/latest/pmd_userdocs_Copy/PasteDetector.html	rev
Deckard [85]	Code	https://github.com/skyhover/Deckard	rev
JPlag [142]	Code & text	https://github.com/jplag/JPlag	rev
Lichen [136]	Code & text	https://submitty.org/instructor/course_management/plagiarism	rev
MOSS [158]	Code	https://theory.stanford.edu/~aiken/moss/	rev, sur
Plaggie [5]	Code	https://www.cs.hut.fi/Software/Plaggie/	rev
plagiarismchecker.com	Text	http://www.plagiarismchecker.com/	rev
Power and Waldron [140]'s Tool	Code	https://github.com/johnwaldron-tcd/codemark-verilog-cleaner-tokeniser	rev
Safeassign	Text	https://help.blackboard.com/SafeAssign/Student/Submit_SafeAssign	rev
Sherlock (Sydney)	Code & text	https://github.com/diogocabral/Sherlock	rev
Sherlock (Warwick) [87]	Code & text	https://warwick.ac.uk/fac/sci/dcs/research/ias/software/sherlock/	rev
SIM [64]	Code & text	https://dickgrune.com/Programs/similarity_tester/	rev
SourcererCC [156]	Code	https://github.com/Mondego/SourcererCC	rev
STRANGE [94]	Code	https://github.com/oscarkarnalim/strange	rev
Turnitin	Text	https://www.turnitin.com/	rev
UniCheck	Text	https://unicheck.com/	rev
WCopyFind	Text	https://plagiarism.bloomfieldmedia.com/software/wcopyfind/	rev

Table 15: Discovered development environments and plugins with two possible sources: literature review (rev) and community survey (sur).

Name	Link	Source(s)
Alice Netbeans Plugin [40]	https://github.com/TheAliceProject/alice3	rev
Amphibian [18]	https://github.com/cacticouncil/amphibian	rev
Blockly [181]	https://developers.google.com/blockly	rev
cs50.io [7]	https://cs50.io	rev
Makecode [14]	https://www.microsoft.com/en-us/makecode	rev
Repl.it	https://replit.com/	sur
Reveal [89]	https://github.com/ace-lab/reveal	rev
Scratch [148]	https://scratch.mit.edu/	rev
SDES [173]	https://github.com/SariSultan/SDES	rev
Snap! [72]	https://snap.berkeley.edu/	rev
Thonny [6]	https://thonny.org	rev

Table 16: List of Visualisation tools that can be used teaching computer science with two possible sources: literature review (rev) and community survey (sur).

Name	Platform	Link	Source(s)
AlgoRythmics [153]	Web	https://algorythmics.ms.sapientia.ro/	rev
AlgoTouch [54]	Java	https://algotouch.irisa.fr/	rev, sur
Alice [145]	Java	https://www.alice.org/	rev
ANIMAL [152]	Java	http://www.algoanim.net/	rev
AnimOS CPU-Scheduling [102]	Web	https://ess.cs.tu-dortmund.de/Software/AnimOS/CPU-Scheduling/	rev
BRIDGES [28]	Web	http://bridgesuncc.github.io/	rev
BlueJ [16]	Java	https://www.bluej.org/	rev
Choc [128]	Web	https://www.newline.co/choc/	rev
Cryptography Visualization Tools(DESvisual [177], AESvisual [110], RSAvisual [176], SHAvisual [109], ECvisual [175], & VIGvisual [175])	Multi	https://pages.mtu.edu/~shene/NSF-4/	rev
CT1 [1], CT2 [101], JCT & CTO [13]	Windows, Java, Web	https://www.cryptool.org/en/	rev
Data Structure Visualizations [57]	Web	https://www.cs.usfca.edu/~galles/visualization/index.html	rev
DDS [144]	Web	http://dsviewer.org/	rev
ExplorViz [74]	Java, Web	https://www.explorviz.net/	rev
Flap.js [103]	Web	https://github.com/flapjs	sur
Flow [27]	Java, Plugin	http://findtheflow.io	rev
HDX [178]	Web	https://courses.teresco.org/metal/hdx/	rev
JAAL [180]	Web	https://github.com/atilante/JAAL	rev
JFLAP [151]	Java	https://www.jflap.org/	rev
jGRASP [39]	Java, Plugin	https://www.jgrasp.org/	rev
Jive [84]	Plugin	https://cse.buffalo.edu/jive/	rev
JSAV [93]	Web	https://github.com/vkaravir/JSAV	rev
JSVEE and Kelmu [166]	Web	https://github.com/Aalto-LeTech/jsvee	rev
Marble MLFQ [97]	Web	https://learn-mlfq.herokuapp.com/	rev
Moodle Trace Generator [154]	Web	https://github.com/CSTools-UCD/moodle-trace-generator	rev
Omnicode [92]	Web	https://github.com/eddings/Omnicode	rev
Online Python Tutor [69]	Web	https://pythontutor.com/	rev, sur
Pathfinding Visualizer ThreeJS [120]	Web	https://github.com/dhruvmisra/Pathfinding-Visualizer-ThreeJS	rev
PLIVET [80]	Web	https://github.com/RYOSKATE/PLIVET	rev
Progvis [172]	Windows, Linux	https://storm-lang.org/index.php?q=06-Programs%2F01-Progvis.md	rev
Recursion Tree Visualizer [133]	Web	https://github.com/brpapa/recursion-tree-visualizer	rev
TigerJython [99]	Java	https://tigerjython.ch/en	rev
TSGL [3]	C++ lib	https://github.com/Calvin-CS/TSGL	sur
TSAL [2]	C++ lib	https://github.com/Calvin-CS/TSAL	rev, sur
UUhistle [167]	Java	http://www.uuhistle.org/index.php	rev
Vamanos [30]	Web	https://rosulek.github.io/vamonos/	rev
VeriSIM [141]	Web	https://verisim.tech	rev
VisuAlgo [70]	Web	https://visualgo.net/	rev, sur
Willow [125]	Web	https://github.com/pedro00dk/willow	rev

Table 17: Discovered class management tools with two possible sources: literature review (rev) and community survey (sur).

Name	Platform	Link	Source(s)
Blackboard Learn	Web	https://www.blackboard.com/	rev
Canvas	Multi	https://www.instructure.com/canvas	rev
Coursera	Web	https://www.coursera.org/	rev
Doubtfire	Web	https://doubtfire.io/	rev
Elgg	Web/PHP	https://elgg.org/	rev
Google Classroom	Web	https://edu.google.com/	rev
Gooru	Web	https://gooru.org/about/	sur
GradeCraft	Web	https://gradecraft.com/	rev
HKU Space Soul	Mobile	https://soul2.hkuspace.hku.hk/	rev
Isaac Computer Science	Web	https://isaaccomputerscience.org/	sur
Moodle	Multi	https://moodle.org/	rev,sur
MicrosoftTeams	Multi	https://www.microsoft.com/en-us/microsoft-teams/group-chat-software	rev
Piazza	Multi	https://piazza.com/	sur
RealTalk	Multi	https://zenodo.org/record/4780955#.YssRiFjMI8M	rev
RepoBee	Multi	https://repobee.org/	rev
Taman Belajar	Web	https://tamanbelajar.com/	rev
WhatsApp	Mobile	https://www.whatsapp.com/	rev

Table 18: Discovered polling/quizzing tools with two possible sources: literature review (rev) and community survey (sur). For each one we indicate if there is at least one LMS with which it *integrates* (more closely than just an iframe embedding).

Name	LMS Integration	Link	Source(s)
Acadly	yes	https://www.acadly.com/	rev
Adobe Captivate	yes	https://www.adobe.com/fr/products/captivate.html	rev
AMC QCM	no	https://www.auto-multiple-choice.net/index.fr	sur
Facebook	no	https://facebook.com	rev
Factitious	no	http://factitious-pandemic.augamestudio.com/	rev
Formative Quizzing	yes	https://yantraedu.com/	sur
Gathertown	no	https://www.gather.town/	rev
Google Apps script	no	https://developers.google.com/apps-script	rev
Google Forms	no	https://www.google.com/intl/en/forms/about/	rev, sur
GoToMeeting	no	https://www.goto.com	rev
HapYak	yes	https://corp.hapyak.com	rev
iclicker	yes	https://www.iclicker.com/	rev
Kahoot!	no	https://kahoot.it	rev, sur
Kytos	no	https://github.com/tychonievich/quizzes	sur
Ms Office Mix		https://office-mix.apponic.com/	rev
OpenIRS-UCM	yes	http://openirs-ucm.sourceforge.net/	rev
PollEverywhere	yes	https://www.polleverywhere.com/	rev
Polly	no	http://www.polly.ai/	rev
Quizlet	no	https://quizlet.com	
Quizizz	yes	https://quizizz.com/	sur
Slido	no	https://www.sli.do/	rev, sur
Socrative	no	https://socrative.com	rev
TwitchTV	no	https://www.twitch.tv	rev
Veriguide	yes	https://veriguide1.cse.cuhk.edu.hk/portal/plagiarism_detection/index.jsp	rev
Wooclap	yes	https://www.wooclap.com	rev
Zoom	yes	https://zoom.us/	rev

Table 19: Tools listed by respondents of the community survey that did not meet the inclusion criteria, i.e., have less than 4 mentions, but included a URL.

Name	Link	# Mentions
Beamer	https://bitbucket.org/rivanvx/beamer/ and https://tikz.dev/	3
GitHub Classroom	https://classroom.github.com	3
Google Docs	https://docs.google.com	3
Google Meet	meet.google.com	2
Git	https://git-scm.com/	2
GitHub	https://github.com	2
Visual Studio Code	https://code.visualstudio.com	2
ACP	https://acp.foe.auckland.ac.nz/	1
AMC QCM	https://www.auto-multiple-choice.net/index.fr	1
Archimedes	https://github.com/tychonievich/archimedes	1
AutoMultipleChoice	https://www.auto-multiple-choice.net	1
Coding Exams	https://github.com/deternitydx/coding-exams	1
Compilatio	https://www.compilatio.net/	1
E-Strange	https://e-strange.org/	1
Eclipse	https://eclipse.org	1
Emacs	https://www.gnu.org/software/emacs/	1
Explain Everything App	https://explaineverything.com/	1
ffmpeg	http://ffmpeg.org/	1
Formative Quizzing	https://yantraedu.com/	1
Geany	https://www.geany.org/	1
GitLab	https://about.gitlab.com/	1
hedy	https://www.hedycode.com/	1
IntelliJ	https://www.jetbrains.com/idea/	1
Java Tutor	https://pythontutor.com/java.html	1
Jenkins	https://www.jenkins.io/	1
Js-Parsons	http://js-parsons.github.io/	1
junit	https://junit.org/junit5/	1
Kytos	https://github.com/tychonievich/quizzes	1
LaTeX	https://www.latex-project.org/	1
MyDigitalHand	https://beta.mydigitalhand.org	1
Notion	https://www.notion.so/	1
OH Queue	https://www.eberly.cmu.edu/ohq/#/	1
Otter Autograder	https://otter-grader.readthedocs.io/en/latest/	1
Padlet	https://padlet.com/dashboard	1
Pugofer (Haskell) interpreter	https://github.com/rusimody/pugofer	1
Pylagiarism	https://github.com/defeo/pylagiarist	1
pytest	https://docs.pytest.org/en/7.1.x/	1
quizlet	https://quizlet.com/	1
Rephactor	https://rephactor.com//	1
RStudio	https://www.rstudio.com/	1
Coding Exams (Sherlock)	https://github.com/deternitydx/coding-exams	1
Slack	https://slack.com/	1
Slido	https://www.slido.com/	1
TigerJython	https://tigerjython.ch/en	1
UCSD Lecture Podcasting	http://podcast.ucsd.edu	1
Unreal Engine 5	https://www.unrealengine.com/en-US/	1
Visual Studio	https://visualstudio.microsoft.com/	1
VMWare Workstation	https://www.vmware.com/	1
Win Merge	https://winmerge.org/	1
Xournal++	https://xournalpp.github.io/	1
YouTube	https://www.youtube.com/	1
7Zip	https://www.7-zip.org/	1

Table 20: Additional tools known to the working group authors that may be useful to the Computing Education community; some of them are featured with references for further reading.

Name	Link	Purpose
Ceebot	http://www.ceebot.com/ceebot/index-e.php	Teaching tool
CSTRANGE [96]	https://github.com/oscarkarnalim/CSTRANGE	Plagiarism (similarity) detector
Discord	https://discord.com	Discussion/classroom tool
DOMJudge	https://www.domjudge.org/	Autograder
Hedy [75]	https://www.hedycode.com/	Teaching tool
Jeliot 3 [126]	http://cs.joensuu.fi/jeliot/	Program visualisation/ IDE
Kattis	https://open.kattis.com/	Autograder
MS PowerPoint	https://www.microsoft.com/en-us/microsoft-365/powerpoint	Presentation
Nearpod	https://nearpod.com	Poll/ quiz
NetSupport	https://www.netsupportschool.com/	Class management/monitoring
Overleaf	https://overleaf.com/	LaTeX collaboration
Reddit	https://reddit.com	Discussion/classroom tool
Sakai	https://www.sakailms.org/	Class management/monitoring
ShellOnYou [17]	https://shellonyou.fr	Autograder
TopHat	https://tophat.com	Poll/quiz
Typeform	https://www.typeform.com/examples/polls/	Poll/quiz
Votar	https://votar.libre-innovation.org/index.en.html	Poll/quiz
Xamarin Workbooks	https://docs.microsoft.com/en-us/archive/msdn-magazine/2016/connect/xamarin-workbooks-the-interactive-future-of-technical-docs	Interactive programming