# A Survey of K-12 Teacher Needs for an Online Programming Learning System

Ally Limke
anlimke@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Veronica Cateté
Marnie Hill
vmcatete@ncsu.edu
mehill6@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Tiffany Barnes
tmbarnes@ncsu.edu
North Carolina State University
Raleigh, NC, USA

## ABSTRACT

This article examines US K-12 computing teacher needs for a programming learning system. We surveyed 39 K-12 teachers about the necessity of programming learning system features. We found that teachers needed to view student code remotely, student code auto-save, differentiation of student assignments, and a tutorial library for students to learn about the programming environment. In addition to rating feature usefulness, we also asked teachers to list features or needs that the survey did not address. Through qualitative responses, we found that teachers wanted cheating prevention and detection, the ability to freeze and project code onto student screens, and student and classroom-level analytics. We also compare the needs of teachers who teach computing as the main subject in their classroom to the needs of teachers who integrate computing into another discipline. This research can inform the development of programming learning systems to better support teachers and their students.

## CCS CONCEPTS

• **Human-centered computing** → **Empirical studies in HCI**; • **Applied computing** → **Learning management systems**.

## KEYWORDS

Programming learning, K-12, learning systems, K-12 teachers, Online learning environments

**ACM Reference Format:**
Ally Limke, Veronica Cateté, Marnie Hill, and Tiffany Barnes. 2024. A Survey of K-12 Teacher Needs for an Online Programming Learning System. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '24), May 11–16, 2024, Honolulu, HI, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3613905.3651110

## 1 INTRODUCTION

As more states in the US are adopting computer science (CS) as a high school graduation requirement[8], it has become important to find ways to support K12 teachers in teaching CS and programming. In 2023, only 57.45% of high schools offered CS courses [10]. This is due to a lack of well-prepared CS teachers. When teachers do learn CS, it is important to understand how to support them in their everyday practice through programming learning systems (PLS).

Teachers are reluctant to adopt learning tools because of the limitations on their time in conjunction with their high workloads, the lack of training with the tools, and the high conceptual complexity of new tools [11]. We can ease the burden of adopting new classroom software by ensuring that tool designs align with the needs and processes of teachers. To do so, it is important to involve users in the design of new learning systems, especially when establishing design requirements [26]. Past co-design research can inform us of some design considerations for new learning environments. This includes supporting different class formats [33], mapping of learning environment content to curricula [33], and preserving teacher autonomy [1]. While prior research on the design of educational technologies is informative, we know that teachers' instructional processes are complex. These processes include 1. defining learning goals, 2. planning instruction, 3. teaching content, 4. evaluating learning, and 5. refining instructional methods [29]. The implementation of these processes differs between teachers, disciplines, grade levels, and classroom contexts.

In this study, we aim to understand the unique needs of K12 computing teachers in the United States as users of PLSs, especially those teaching students in introductory computing courses using block-based programming environments (BBPE). Many K12 CS teachers have minimal experience with computing topics or programming [41], and while professional development can help, there is still great need to support new CS teachers [23].

Online PLSs can help teachers in classroom orchestration. For example, GradeSnap [30], a grading tool for the Snap*!* BBPE, was created to address the time-consuming nature[5] and difficulty [40] of grading block-based programs. Other platforms, like Code.org [9] and Microsoft MakeCode[3], integrate curricula and automated grading into their platforms. However, PLSs that provide fully automated grading limit student creativity and teacher curriculum choices.

To develop robust PLSs that can support teachers in all stages of the instructional process [29], we first need to know what teachers need from these online platforms. In our prior work, we conducted participatory design with three K12 CS teachers, resulting in a list of 16 features needed in a PLS [24]. However, it is unclear whether these identified features apply to a broader sample of teachers and which features should be prioritized in PLSs for K12.

Therefore, we created and administered a survey to 39 K12 CS teachers in the United States to answer the research question: *What*


*CHI EA '24, May 11–16, 2024, Honolulu, HI, USA*

ACM ISBN 979-8-4007-0331-7/24/05
https://doi.org/10.1145/3613905.3651110

*do teachers see as the most important features needed in an online programming learning platform to support their instructional practice?*. We found the top 4 features for all teachers were: viewing student code remotely, auto-saving student work, differentiating assignments based on students' prior programming experience, and having a tutorial library students can reference to learn about the programming environment. A more nuanced investigation showed differing priorities based on the teacher's primary discipline, suggesting that teacher needs vary according to the amount of programming taught in their courses and the educational needs of their students. These results suggest that more research is needed to understand the changing needs of teachers based on the contexts in which they teach programming.

## 2 RELATED WORK

The increasing importance of CS skills [32] has prompted efforts to expand CS and programming education in U.S. K12 schools [2, 4], especially through a new high school CS course [14, 16], but also through integrating computing into other courses [19]. However, few K12 teachers have adequate CS backgrounds [41] or professional development resources to help them build their programming and pedagogical knowledge [23].

A systematic literature review shows that most K12 CS and computational thinking (CT) studies teach CS/CT using programming environments, sometimes augmented with physical tools, such as robotics kits [23]. There are many existing programming environments used in K12 education [15] including Scratch [27] and Snap! [17]. Learning pedagogical and technical aspects of new programming environments is a taxing time commitment for teachers [28], that is in addition to their existing duties.

A critical analysis of 73 systematic literature reviews shows that the most effective educational technologies are those "involving interaction, gamification, constructivism, student-centered learning, and feedback" [21], but only 19% of reviews examined teaching or pedagogical aspects. A survey of university-level CS educators on the features needed for an online learning environment for programming showed needs for programming assignment submission, automated assessment, interactive debugging support, algorithmic visualization, and plagiarism detection [34]. However, it is still unclear whether these needs are similar for K12 teachers, who teach primarily introductory programming, and who are more likely to be using block-based programming environments.

To better understand K12 teacher needs, in 2022, as part of a Research Experiences for Teachers (RET) program, we engaged three K-12 teachers in intensive week-long participatory design sessions [24] following the Make, Tell, Enact cycle by Sanders et al. [6, 35, 36]. Data collected included audio recordings and design artifacts including drawings, notes, and prototypes created in Figma [13]. First, we conducted the Tell step with a Future workshop using Troxler et al. and Lauttamak et al. frameworks [22, 39] to help the teachers outline their workflows, identify pain points in their workflows, and describe a world in which their problem no longer existed, often describing a technology solution. Then teachers identified realistic software solutions. Second, we conducted a 'Make' session that combined elements of a Sketch-in [25] with Live Prototyping. Here, the three teachers created a list of 16 new SnapClass features, that we use here and are listed in Table 1. Each solution was discussed, and then the teachers each chose a solution that was highest priority to them and designed for it.

One of our participants, Amanda, was a high school CS teacher who had been teaching for over 20 years. Amanda taught AP Computer Science Principles using the Snap*!* BBPE [4]. Her students mostly worked on open-ended programming projects, but she also assigned closed-ended tasks that allowed her to provide immediate and specific feedback. Amanda assigned independent and group work to prepare students for academic and industry careers. Amanda chose to design peer- and self-assessment features to help prepare students for the AP exam.

Becca was a seventh-grade science teacher who had been teaching for more than 10 years. Chad was a sixth grade science teacher who had been teaching for more than 20 years. Programming activities in Becca and Chad's classrooms were open-ended and collaborative, helping students creatively explore what they learned in science while also practicing programming and computational thinking. Becca designed an assignment differentiation feature, both to address different levels of programming experience and to address student educational needs. Chad designed a help-request queue to help him manage peer help. Chad already established a classroom culture where students would help one another during programming, a proposed strategy to improve access to help during programming [12].

## 3 METHODS

Our prior participatory design sessions gave us rich insights into the needs of a few K12 computing teachers. This study investigates how their needs align with the needs of the larger population of K-12 computing teachers. Therefore, we surveyed K12 teachers on the features they need in an online learning platform for programming. Starting from the list of 16 features created during the participatory design sessions [24], we asked teachers to rate the usefulness of each feature on a 5 point labeled scale: "Not useful", "Slightly useful", "Somewhat useful", "Very useful" and "Critical to my job", later converting these to scores from 1-5 and averaging them for each feature. Finally, we asked teachers to list any other features they might need in a programming assignment system. The teachers requested a total of 71 features. Two researchers each tagged the responses separately, first identifying if any of the requests overlapped with the 16 initial PD features in Table 1. The remaining requests were open-coded by the two researchers separately and then discussed together to find 14 newly discovered features shown in Table 2.

Teachers were eligible to participate if they lived in the United States and either taught CS or integrated computing activities into their classrooms. Participants were recruited through email, social media, and CS educator organizations. There were 39 responses, with 31 teaching computing as their primary subject, and 8 teaching other subjects, such as math or business education. Fifteen participants had 1-5 years of teaching experience, while 11 had 6-10, and 13 had 11 or more. The primary languages participants indicated using in their classrooms were block-based (32), Python (20), Java (14), and JavaScript (7), with 10 other languages listed.

**Table 1: Ranking of listed features.**

| Feature | Average | # of requests |
|---|---|---|
| View student code remotely | 4.41 | 1 |
| Autosaving | 4.32 | 1 |
| Differentiation for programming experience | 4.15 | 2 |
| Tutorial library for prog. env. | 4.13 | 0 |
| Group Work | 4.08 | 5 |
| Forgotten login credentials | 4.05 | 0 |
| Curricula Library | 4.03 | 15 |
| Differentiation for edu needs | 4 | 2 |
| Automated Grading | 3.97 | 3 |
| Customizable Rubrics | 3.77 | 1 |
| Student self-assessment | 3.72 | 0 |
| LMS integration | 3.71 | 1 |
| Rubric library | 3.62 | 1 |
| Help request queue | 3.5 | 0 |
| Export grades | 3.47 | 0 |
| Peer Assessment | 3.46 | 1 |

**Table 2: Qualitative responses of additional features.**

| Additional requested features | # of requests |
|---|---|
| Teacher demo / control screens | 6 |
| Scaffold ind. learning | 6 |
| Usability / Intuitive GUI | 6 |
| Help teachers prevent / identify cheating | 4 |
| Classroom level analytics | 3 |
| Create assignments / starter code | 3 |
| Individual student analytics | 2 |
| Web-based | 2 |
| Built-in instructions | 1 |
| Creating graded and ungraded assignments | 1 |
| Students add code explanations | 1 |
| Control assignment visibility | 1 |
| Timed assigments | 1 |
| Sort assignments into groups | 1 |

# 4 RESULTS

Survey results for quantitative feature rankings and open-ended responses are given in Tables 1 and 2. The seven most highly ranked needs by teachers in the quantitative survey were: 1) viewing student code remotely (4.41), 2) autosaving (4.32), 3) assignment differentiation for varying experience (4.15), 4) a tutorial library for the programming environment (4.13), 4) group work support (4.08), 5) forgotten login credentials (4.05), 6) a curricula library (4.03), and 7) assignment differentiation for differing educational needs (4.00). These results are visualized in Figure 1.

In the open-ended feature question responses, the most frequently requested feature was a curricula library, with 15 feature requests related to finding, creating, and customizing curricula. Teachers wanted familiar curricula to be integrated into a programming activity platform, including BJC [14], CSAwesome [18], CMU CS Academy [38], and Python for Everybody [37]. Three teachers who wanted curricula in the platform did not specify a specific curriculum, instead noting that any curricula should be "designed around student learning objectives". In terms of creating assignments, one teacher mentioned that they wanted to write 'starter code' for their students while another wanted to "create custom assignments with all the functionality as a built-in problem".

The second-most frequent feature, with 6 requests, focused on scaffolding independent problem-solving. Three teachers wanted real-time support for students including having "quick tips pop up", "ask[ing] provoking questions to guide students", and a button that says "click here for more help". Two teachers wanted static resources for student reference. Specifically, one wanted a "troubleshooting guide for fixing recurring error types" and another wanted "vocabulary reference". Finally, one teacher desired progress monitors to give "clear feedback to students on what they've completed and what they still need to accomplish" to orient their independent work. These responses are similar to our 3 prior participatory design teachers who wanted a tutorial library for the programming environment, to enable students to work more independently.

The third-most frequent open-ended feature listed was group work, with 5 requests. One teacher said that they structure their class so that "students always work collaboratively on labs and homework assignments". Teachers suggested three features that would support pair-programming in their classes: shareable screens, concurrent editing, and saving a copy of code to both student accounts.

On the quantitative survey, teachers ranked the following items with lower priority: 1) automated grading (3.97), 2) customizable rubrics (3.77), 3) student self-assessments (3.72), 4) LMS integration (3.71), 5) a rubric library (3.62), 6) a help request queue (3.5), 7) exporting grades (3.46), and 8) peer assessment (3.46). These results, which are visualized in Figure 2, suggest that many of the K12 teacher participants have already determined how to grade programs, or at least that they value other features that would be harder to do without built-in platform support.

## 4.1 Differences Between Teaching Contexts

We split the quantitative survey results into two groups: teachers who taught CS as the main subject in their classrooms (computing teachers) and those who primarily taught another subject (integrating teachers). Computing teachers have more formal training in computing (e.g. through an undergraduate major or a job) and therefore have a higher level of facility in managing the processes of assigning, collecting, and grading programs, and have more ability to quickly assess student programs and provide feedback. Integrating teachers have usually taken a short course (usually a 5-day workshop) on block-based programming and are voluntarily adding this new content to their classrooms. Therefore, each step of the teaching process offers extra barriers that are not part of their required job duties.

**Figure 1: Highest ranked needs from the survey.**

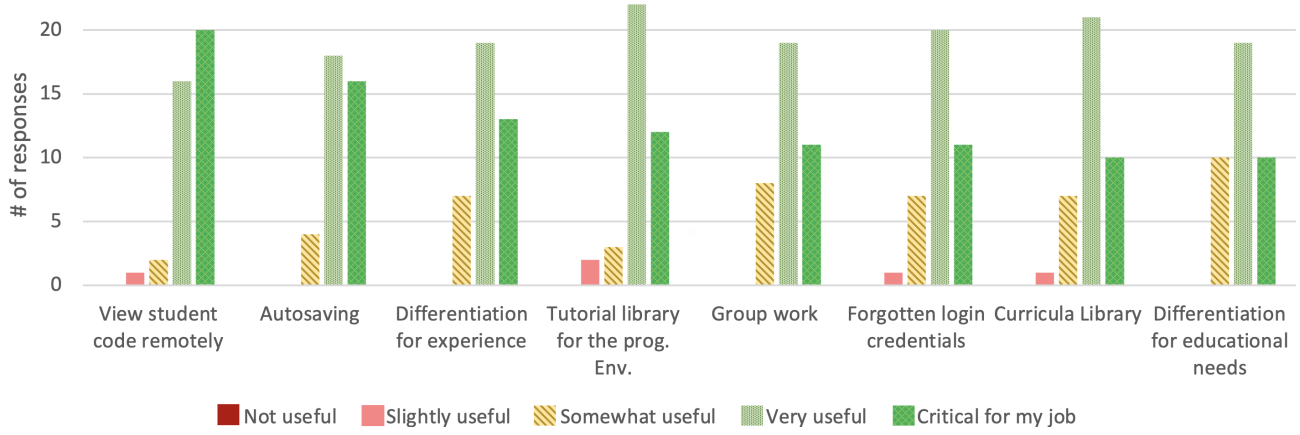
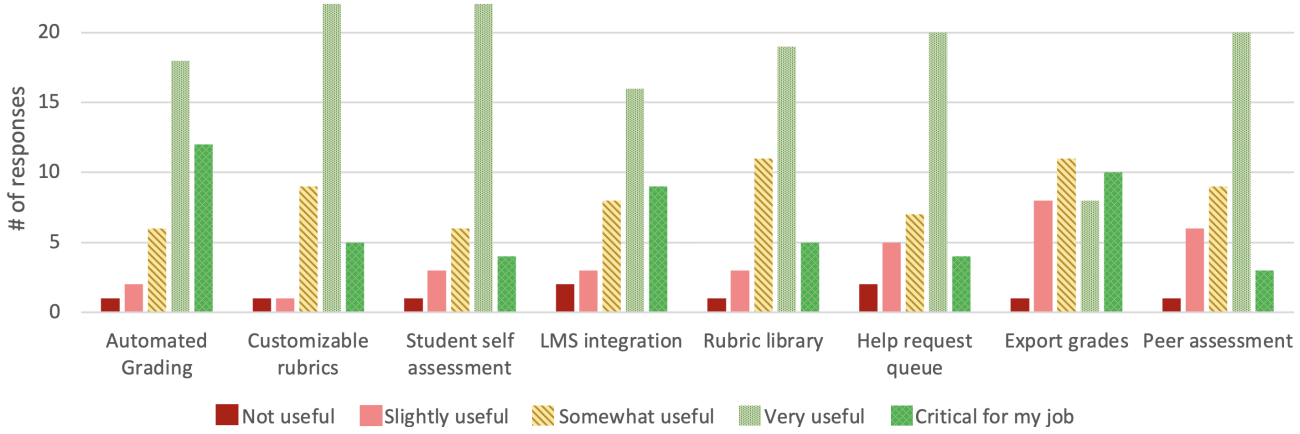



**Figure 2: Lowest ranked needs from the survey.**




Here, we report any features whose average scores differed by 0.5 or more between teacher primary subjects. Computing teachers rated the following features higher: automated grading (by 1.07), LMS integration (by .93), viewing student code remotely (by .67), forgotten login credentials (.54), and a curricula library (by .50). The highest integrating teachers needs were autosaving (4.25), differentiation for differing educational needs (4.13), differentiation for varying programming experience (4), and the tutorial library for the programming environment (4). The highest computing teacher needs were viewing student code remotely (4.55), autograding (4.19), autosaving (4.19), and differentiation for varying experience (4.19).

## 4.2 Additional Qualitative Results

Five participants who requested **additional features for assignments**, including: grouping the assignments by unit, creating timed assignments, creating graded or ungraded activities, controlling assignment visibility, and allowing students to add explanations to their code. Three other teachers wanted to create assignments with starter code.

Six teachers needed additional features to help them **lead** activities. One teacher wanted students to be "able to use their laptops to watch [them] demo instead of everyone wanting to sit closer to a TV or projection screen." Another teacher added that they wanted to lock student screens "so that students don't use other software during the lessons". One teacher wanted assignment instructions built into the programming environment.

Six teachers suggested additional features that would help them with **assessment**. Three wanted classroom level analytics and two wanted analytics about individual students. Four teachers wanted to prevent cheating by blocking AI support, a plagiarism checker, recording a log of student keystrokes, and keeping student assignments private.

Eight teachers requested features to improve **usability for students/school contexts**. For a programming assignment system to be useful, two teachers needed software to be web-based so that "no downloads are required" and it "works on Chromebooks". Six teachers emphasized usability, mentioning "legible font" and "intuitive navigation". Other needs included an immersive reader for

students and not using programming jargon that is inaccessible to beginners.

# 5 DISCUSSION

## 5.1 Results aligned with Participatory Design Findings

Teachers in the participatory design and in the survey expressed needing **support in addressing individual student needs**. It is typical for students to enter K-12 computing classrooms with varying levels of programming experience. This is partially due to programming courses being inequitably distributed across gender, race, geographical location, and disability [10]. Individualizing instruction, through a PLS, may help teachers provide more equitable instruction to their students. One approach to individualization is assignment differentiation - ranked third highest in need. Differentiation is the practice of creating different versions of the same assignment to meet various student needs/experience levels. For programming, assignments are often differentiated through providing more or less starter code. However, it is important to note that teachers also must differentiate assignments to meet varying educational needs, not just prior programming experience. In our survey, computing teachers were more likely to highlight differentiation for programming experience, while teachers with another primary teaching discipline were more likely to request differentiation to meet other student needs. This indicates that, as computing is increasingly integrated into mainstream classrooms, it will become more important to allow for differentiation of the content, both for prior experience and for diverse learners.

In both studies, teachers needed support in **finding curricula**. Two teachers in the participatory design sessions integrated programming into their science classrooms. These teachers needed support in finding programming activities that related to their science content. The 31 teachers, who rated a curricula library as "Very Useful" or "Critical", may want curricula built into a PLS as it can be laborious to transfer activities into learning systems, especially when curricula undergo updates. Although teachers expressed wanting a curriculum library, they did not want to be limited by provided curricula. One teacher wanted the "flexibility, where teachers can change assignments or the requirements of assignments, or pull from a bank of projects or questions". It may be important for teachers to change assignments to cater to their specific classroom contexts. Past research has shown that teachers are wary of pre-written exercises in a learning management system (LMS), because they don't match individual teaching styles; but also, that an LMS without exercises requires extensive work to build exercises [34]. Teachers often use activities from multiple curricula[31]; facilitating the combination of materials from multiple sources may provide teachers flexibility.

Teachers have a finite, and sometimes insufficient, amount of time to help students individually. Hence, they need to enable students to **solve problems independently**. The participatory design RET teachers requested a forum for students to ask assignment-related questions, allowing both the teacher and peers to provide answers as a student reference. The same sentiment is reflected in the six open-ended survey responses focused on scaffolding independent problem-solving. According to Kolb's learning cycle, effective learning happens when students have a concrete experience with a concept, reflect on the experience, use their reflection to modify their conceptualization, and then experiment with what they've learned [20]. Some teacher responses indicate they are thinking more deeply about student learning that may be happening in such learning cycles, with one teacher requesting a feature with "provoking questions to guide students to complete their assignment", scaffolding the reflection phase of the learning cycle. This feature would model for students how to think through problems they encounter. One teacher noted that the intelligent supports they might like to include could be abused by students, so teachers are seeing a need for systems that provide help but don't allow students to cheat or complete assignments without learning.

Survey and participatory design teachers both expressed a desire for **automated grading**. Although it was the 10th ranked feature, it seems important to at least a subset of teachers as there were three open-ended responses requesting autograding. This finding aligns with a previous survey completed with 31 CS educators who reported wanting automated feedback and assessment in a LMS [34]. Two teacher feature responses specifically needed the ability to write unit tests. However, many other teachers assign open-ended programming activities where creativity of theme, style, and function are encouraged. These types of projects are harder to write unit tests for and take teachers longer to grade. One participant echoed this problem saying they need "a way to read code that allows for multiple ways [for students to solve], yet still meet the criteria". Teachers who integrate computing into other subjects often grade student projects for completion which may explain why they rated the automated grading feature as a much lower need for their classrooms than the teachers who mainly teach computing.

## 5.2 Misaligned results

In the participatory design sessions, Amanda designed a **self and peer-assessment feature**. However, in survey results, these features ranked eleventh and sixteenth respectively. Amanda teaches AP Computer Science principles and is an AP exam grader. Amanda shared that the College Board expects students to work individually on written assignments and therefore, written reflections may prepare her students for the AP test. Other CS teachers may have their students work more collaboratively, as in Chad and Becca's classrooms. In those cases, students would already be getting constant feedback from their peers. Another factor impacting the ranking may be that teachers already have peer/self assessment practices that work for their classrooms, they may not have enough time to assign reviews, or they may not believe that peer and self assessments would improve student learning.

During the participatory design, Chad designed a **help-request queue** which was a digital list of students requesting help from the teacher. Not only would the system notify him when students needed help, but would automatically assign them a peer tutor. Chad's room is highly collaborative and Chad matches peers for tutoring. In the survey, a help-request queue ranked twelfth. Additionally, in the open-ended survey responses, there were no suggested classroom management features that would be used during project work time. It may be that teachers were not wanting or envisioning technical solutions to their classroom environment

problems. While research suggests that teachers could benefit from classroom management features [12], but that features need to be designed in a manner that matches teacher processes. For example, a curriculum-integrated game put red flags directly on student screens that needed help [7]. This may be a better alternative to a dashboard that is viewed from a single teacher monitor, as teachers often walk around their classrooms to monitor students [33]. Additionally, a help request feature might help teachers provide more equitable instruction by guiding them to students who may be less likely to receive an intervention in their class.

In the survey, there were four responses focusing on **preventing or identifying cheating** unlike the participatory design sessions who did not focus on cheating. This may be because avenues for cheating, like ChatGPT, have become more popular among students since the participatory design sessions were conducted in 2022. Alternatively, it may be that two of the teachers in the design sessions integrated computing into their science classrooms using a system where code sharing was very clunky, and graded students for completion rather than accuracy.

### 5.3 How teaching context affects programming learning systems

The differences between the teachers who instruct CS as the main subject in their classrooms (Computing teachers) and those who primarily teach another subject and integrate programming activities into that subject (Integrating teachers) can help developers determine how to prioritize features based on their target users.

Both **computing and integrating teachers agreed** that autosaving, a tutorial library for the programming environment, and assignment differentiation were top priorities. Both types of teachers lead programming activities that span multiple days. Saving code is important so no progress is lost. Next, a tutorial library for the programming environment may save teachers time answering students' questions about environment basics. Integrating teachers may not feel confident answering student programming questions, making the tutorials of even higher importance.

Automated grading and the ability to view student code remotely were in the **top rated features of the computing teachers**, but not the integrating teachers. The difference in the need for automated grading may be the result of the number of programs that computing teachers grade. Integrating teachers often only run a few computing activities a semester and grade based on completion. Next, viewing student code remotely might allow computing teachers to non-intrusively check student progress as computing teachers are more likely to have the programming skills to quickly evaluate and debug student code. Integrating teachers might be more focused on helping students understand their own discipline and CS/CT through coding rather than very specific programming learning objectives.

## 6 CONCLUSION

We administered a survey (n=39) to understand teacher needs in programming learning systems. While previous research findings gave us depth of insight into three teachers' needs, we wanted to know the needs of a broader sample of K-12 computing teachers in the United States. Our results can provide a basis for other researchers to design tools to support teachers leading programming activities. We found that teachers needed to view student code remotely, for student code to auto-save, differentiation of assignments, and to have a tutorial library about the programming environment for student reference. Through open-ended responses, we found that teachers wanted cheating prevention and detection, the ability to freeze and project code onto student screens, and student and classroom level analytics.

This study has limitations in sample size, and also a lack of information about teacher contexts and specific uses of computing in their classrooms. Our sample size was small and therefore statistical tests between computing and integrating K12 teachers were not appropriate in our context. Our results merely suggest that there may be a difference in needs between these populations, but more investigation is needed to confirm our results. Further, teachers' imaginations for features may be limited by their views of the structure of traditional learning systems and what they think is technologically possible within these systems. Future studies may first investigate, through interview or observation, K-12 computing teacher processes and struggles to understand how a system can support them. We believe that future work should include participatory design with teachers, as well as classroom studies, to design, develop, and evaluate features that will make the most impact on teacher practices and student learning for programming in K12.

## ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 2112635. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

## REFERENCES

[1] Pengcheng An, Kenneth Holstein, Bernice d'Anjou, Berry Eggen, and Saskia Bakker. 2020. The TA Framework: Designing Real-time Teaching Augmentation for K-12 Classrooms. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (<conf-loc>, <city>Honolulu</city>, <state>HI</state>, <country>USA</country>, </conf-loc>) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3313831.3376277

[2] Owen Astrachan, Jan Cuny, Chris Stephenson, and Cameron Wilson. 2011. The CS10K project: mobilizing the community to transform high school computing. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 85–86. https://doi.org/10.1145/1953163.1953193

[3] 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* (Athens, Greece) *(SPLASH-E 2019)*. Association for Computing Machinery, New York, NY, USA, 7–12. https://doi.org/10.1145/3358711.3361630

[4] College Board. 2023. AP Computer Science Principles Course and Exam Description. https://apcentral.collegeboard.org/media/pdf/ap-computer-science-principles-course-and-exam-description.pdf?course=ap-computer-science-principles Accessed: 2024-01-24.

[5] 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* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 215–220. https://doi.org/10.1145/2445196.2445265

[6] Eva Brandt, Thomas Binder, and Elizabeth B-N Sanders. 2012. Tools and techniques: Ways to engage telling, making and enacting. In *Routledge international handbook of participatory design*. Routledge, London, 145–181.

[7] Melissa N Callaghan, JJ Long, EA Van Es, Stephanie M Reich, and Teomara Rutherford. 2018. How teachers integrate a math computer game: Professional development use, teaching practices, and student achievement. *Journal of Computer Assisted Learning* 34, 1 (2018), 10–19.

[8] Code.org. 2023. Code.org recommends graduation requirements in computer science. https://codeorg.medium.com/code-org-recommends-graduation-requirements-in-computer-science-5f04fbf24de Accessed: 2024-01-25.

[9] code.org. 2024. Code.org platform for computer science curricula. https://code.org/ Accessed: 2024-01-25.

[10] code.org. 2024. State of CS Report. https://advocacy.code.org/stateofcs Accessed: 2024-01-25.

[11] Francesca Maria Dagnino, Yannis A Dimitriadis, Francesca Pozzi, Juan I Asensio-Pérez, and Bartolomé Rubia-Avi. 2018. Exploring teachers' needs and the existing barriers to the adoption of Learning Design methods and tools: A literature survey. *British Journal of Educational Technology* 49, 6 (2018), 998–1013.

[12] Nicholas Diana, Michael Eagle, John Stamper, Shuchi Grover, Marie Bienkowski, and Satabdi Basu. 2018. Peer tutor matching for introductory programming: Data-driven methods to enable new opportunities for help. In *ICLS 2018 Proceedings*. International Society of the Learning Sciences, Inc.[ISLS]., London, UK, 1377–1378.

[13] Figma. 2016. *Figma: the collaborative interface design tool.* Figma. http://figma.com

[14] Dan Garcia, Brian Harvey, and Tiffany Barnes. 2015. The beauty and joy of computing. *ACM Inroads* 6, 4 (2015), 71–79.

[15] Varvara Garneli, Michail N. Giannakos, and Konstantinos Chorianopoulos. 2015. Computing education in K-12 schools: A review of the literature. In *2015 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, Tallinn, Estonia, 543–551. https://doi.org/10.1109/EDUCON.2015.7096023

[16] Joanna Goode, Jane Margolis, and Gail Chapman. 2014. Curriculum is not enough: the educational theory and research foundation of the exploring computer science professional development model. 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, 493–498. https://doi.org/10.1145/2538862.2538948

[17] 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

[18] Beryl Hoffman and Barbara Ericson. 2021. CSAwesome: A Free Curriculum and Ebook for Advanced Placement Computer Science A (CS1 in Java). https://doi.org/10.1145/3408877.3432504

[19] Robin Jocius, Deepti Joshi, Yihuan Dong, Richard Robinson, Veronica Cateté, Tiffany Barnes, Jennifer Albert, Ashley Andrews, and Nicholas Lytle. 2020. Code, Connect, Create: The 3C Professional Development Model to Support Computational Thinking Infusion. 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, 971–977. https://doi.org/10.1145/3328778.3366797

[20] David A Kolb. 2014. *Experiential learning: Experience as the source of learning and development.* FT press, Upper Saddle River, New Jersey.

[21] Jennifer WM Lai and Matt Bower. 2020. Evaluation of technology use in education: Findings from a critical analysis of systematic literature reviews. *Journal of Computer Assisted Learning* 36, 3 (2020), 241–259.

[22] Ville Lauttamäki. 2014. Practical guide for facilitating a futures workshop. , 2–11 pages.

[23] Sang Joon Lee, Gregory M Francom, and Jeremiah Nuatomue. 2022. Computer science education and K-12 students' computational thinking: A systematic review. *International Journal of Educational Research* 114 (2022), 102008.

[24] Ally Limke, Nicholas Lytle, Sana Mahmoud, Maggie Lin, Marnie Hill, Veronica Cateté, and Tiffany Barnes. 2023. Participatory Design with Teachers for Block-Based Learning with SnapClass. In *2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, limke2023participatory, Washington, DC, USA, 173–178.

[25] Rosemary Luckin, Sadhana Puntambekar, Peter Goodyear, Barbara Louise Hopkins Grabowski, Joshua Underwood, Niall Winters, et al. 2013. *Handbook of design in educational technology.* Routledge, London.

[26] P. Luff, M. Jirotka, C. Heath, and D. Greatbatch. 1993. Tasks and social interaction: the relevance of naturalistic analyses of conduct for requirements engineering. In *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*. IEEE, San Diego, CA, USA, 187–190. https://doi.org/10.1109/ISRE.1993.324818

[27] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 1–15.

[28] Orni Meerbaum-Salant, Michal Armoni, and Mordechai (Moti) Ben-Ari. 2010. Learning computer science concepts with scratch. In *Proceedings of the Sixth International Workshop on Computing Education Research* (Aarhus, Denmark) *(ICER '10)*. Association for Computing Machinery, New York, NY, USA, 69–76. https://doi.org/10.1145/1839594.1839607

[29] V Meisalo, E Sutinen, and Jorma Tarhio. 2003. Modernit oppimisympäristöt-Tieto-ja viestintätekniikka opetuksen ja oppimisen tukena.

[30] Alexandra Milliken, Veronica Cateté, Amy Isvik, and Tiffany Barnes. 2020. Poster: Designing GradeSnap for Block-Based Code. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, Dunedin, New Zealand, 1–2. https://doi.org/10.1109/VL/HCC50065.2020.9127284

[31] Alexandra Milliken, Christa Cody, Veronica Catete, and Tiffany Barnes. 2019. Effective Computer Science Teacher Professional Development: Beauty and Joy of Computing 2018. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE '19)*. Association for Computing Machinery, New York, NY, USA, 271–277. https://doi.org/10.1145/3304221.3319779

[32] United States Department of Labor. 2023. U.S. Bureau of Labor Statistics. https://www.bls.gov/oes/tables.htm Accessed: 2024-01-25.

[33] Zhongxiu Peddycord-Liu, Veronica Cateté, Jessica Vandenberg, Tiffany Barnes, Collin F. Lynch, and Teomara Rutherford. 2019. A Field Study of Teachers Using a Curriculum-integrated Digital Game. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300658

[34] Guido Rößling, Mike Joy, Andrés Moreno, Atanas Radenski, Lauri Malmi, Andreas Kerren, Thomas Naps, Rockford J Ross, Michael Clancy, Ari Korhonen, et al. 2008. Enhancing learning management systems to better support computer science education. *ACM SIGCSE Bulletin* 40, 4 (2008), 142–166.

[35] Elizabeth B.-N. Sanders, Eva Brandt, and Thomas Binder. 2010. A framework for organizing the tools and techniques of participatory design. In *Proceedings of the 11th Biennial Participatory Design Conference* (Sydney, Australia) *(PDC '10)*. Association for Computing Machinery, New York, NY, USA, 195–198. https://doi.org/10.1145/1900441.1900476

[36] Elizabeth B.-N. Sanders and Pieter Jan Stappers. 2014. Probes, toolkits and prototypes: three approaches to making in codesigning. *CoDesign* 10, 1 (2014), 5–14. https://doi.org/10.1080/15710882.2014.888183 arXiv:https://doi.org/10.1080/15710882.2014.888183

[37] Charles Severance. 2016. Python for Everybody. https://www.py4e.com/. Accessed: 2024-03-08.

[38] Mark Stehlik, Erin Cawley, and David Kosbie. 2020. CMU CS Academy: A Browser-based, Text-based Introduction to Programming through Graphics and Animations in Python. 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, 1420. https://doi.org/10.1145/3328778.3372541

[39] Peter Troxler and Beate Kuhnt. 2007. Future workshops. The unthinkable and how to make it happen. , 483–495 pages.

[40] Aman Yadav, David Burkhart, Daniel Moix, Eric Snow, Padmaja Bandaru, and Lissa Clayborn. 2015. Sowing the seeds: A landscape study on assessment in secondary computer science education.

[41] Aman Yadav, Sarah Gretter, Susanne Hambrusch, and Phil Sands. 2016. Expanding computer science education in schools: understanding teacher experiences and challenges. *Computer science education* 26, 4 (2016), 235–254.