What is your biggest pain point? An investigation of CS instructor obstacles, workarounds, and desires

Samim Mirhosseini
NC State University
Raleigh, North Carolina, USA
smirhos@ncsu.edu

Austin Z. Henley
Microsoft
Redmond, Washington, USA
austinhenley@microsoft.com

Chris Parnin
Microsoft
Raleigh, North Carolina, USA
chrisparnin@microsoft.com

ABSTRACT

Computer science instructors have one of the most crucial roles in training and making educational materials. However, they face many challenges everyday that make it difficult to provide a high-quality learning experience to their students. Additionally, trends in demand for computer science training is rapidly increasing and to meet this demand, classrooms need to run on a larger scale, which may exacerbate instructor pain points further. While many of the previous studies in the computer science education community have focused on how to improve the students' learning experience, in this study we investigate computer science instructors. It is paramount to understand how instructors can be supported more effectively while continuing to improve the material they use in their courses and allow them to focus on student needs. To understand these instructor challenges, we conducted semi-structured interviews with 32 computer science instructors at universities and community colleges to ask about their experiences in preparing course material, lecturing, grading, providing feedback to students, and what they wished they could change. In this paper, we summarize our findings as themes of challenges and pain points for instructors, the consequences of not solving them, and suggested guidelines that may help resolve or reduce these pain points.

KEYWORDS

Computer Science Education, Computer Science Instructors, Instructor Pain Points, CS Education Tools, AI in CS Education

ACM Reference Format:

Samim Mirhosseini, Austin Z. Henley, and Chris Parnin. 2023. What is your biggest pain point? An investigation of CS instructor obstacles, workarounds, and desires. In *SIGCSE '23: ACM Technical Symposium on Computer Science Education, March 15–18, 2023, Toronto, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Computer science instructors play a crucial role in the quality of training and educational experience that students receive. Instructors typically have many responsibilities such as creating material, delivering lectures, clarifying student questions, and grading student deliverables. Additionally, growing demands for meeting computer science enrollment has strained instructors with high teaching loads, as insufficient teaching faculty are available [5], across large and small institutes. Handling all of these responsibilities and addressing scale can be challenging in itself, but is especially made worse when instructors spend significant time grappling with hidden obstacles and invisible work to resolve them.

Many of the previous research studies have analyzed interventions for improving student learning and focused on the resulting experience from the perspective of students [4, 13, 20]. While analyzing experience of students can give us many useful insights, in this study, we believe it is just as important to understand what instructors experience, identify opportunities for addressing pain points, and ultimately, to help instructional experiences. Few studies have focused on the perspective of an instructor, for example study by Lau et al. [17], which focused on challenges specific to scaling four data-science courses to larger size. To our knowledge, this paper is one of the first to focus on obstacles instructors face across multiple levels, courses, and institutes.

To understand what pain points CS instructors experience and what their workarounds are, we conducted semi-structured interviews with 32 CS instructors regarding their experiences running their courses. These interviews were designed to clarify the common tasks that takes instructors' time, the possible pain points in completing these tasks, and how they (wish they could) overcome these pain points so far. The presented paper contributes to the continuing research on improving CS education, from the perspective of instructors. Finally, we share implications of our findings and provide suggestions for practices or tools that may be helpful to instructors, shortcomings of solutions that instructors currently tried, as well as barriers in adopting some solutions.

2 METHODOLOGY

To understand the challenges of computer science instructors, how they tackle those challenges, and what remains as a pain point we conducted a semi-structured interview with 32 participants. In this section we explain details of our qualitative study which was designed to answer the following research questions:

- **RQ1: What did instructors wish they could change?** Can instructors suggest what they would like to change that is not possible right now? What can we conclude with a high level view of these issues and the current resources available to instructors?
- **RQ2: What are current attempts of addressing pain points?** What are the workarounds instructors use to reduce or solve their pain points right now? What software solutions exist and what research has been conducted?

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGCSE '23, March 15–18, 2023, Toronto, Canada
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2.1 Interviews

We conducted semi-structured interviews with 32 computer science instructors to address our research questions. In this section we share details about participants and the interview process.

2.1.1 Demographics. In an attempt to reduce biased results toward any specific group of participants, we selected interviewees with varying teaching experiences as university instructors. We selected these instructors from 26 different institutions ranging from large R1 doctoral research universities to smaller institutions such as liberal arts and community colleges. To be more specific, interviewees included 3 adjuncts, 7 teaching faculty, and 22 tenured/tenure-track professors from 5 countries (25 from USA, two from India, and one each from Canada, Germany, Netherlands, Mexico, and Austria).

2.1.2 Recruitment. We recruited the participants for the interview by first selecting institutions and then recruiting from their affiliated instructors. We selected 26 institutions and connected with a sample of their instructors who were actively teaching in the same or previous academic term. We then used snowball sampling by asking our interview participants to help us connect with their colleagues or anyone that they think can have relevant experience.

2.1.3 Protocol. We conducted 30-minute semi-structured interviews virtually while transcribing and taking notes. During the interview, we followed a script which included questions about:

- Lecture structure and deliverables of students
- Availability of resources such as TAs and tools
- Time spent on materials, grading, and lecture preparation
- Techniques to improve learning experiences
- Most painful aspects of running courses
- Thoughts about potential solutions to pain points

We dynamically revised our script template based on the responses that we received from instructors to be able to extract more meaningful information from the conversations and understand motivation and importance of what they currently do or what they wish they were able to change. We intentionally asked about the pain points last, because at this stage, we have collected supporting information for why something can became a challenge or pain point, and discuss possible solutions.

2.2 Interview Analysis

Using our interview notes, we first focused on the last topic which was "*challenges and wishes*" of the instructors. We analyzed the instructor responses through thematic analysis [1] by reviewing the transcripts, coding the parts of interest, and then finding themes in the coded transcripts. Next we reviewed other topics covered in the interview transcripts to add supporting themes.

3 FINDINGS

In the analysis of our semi-structured interviews we learned about the pain points of instructors, the workarounds they use, and what they wish to do which is not possible at the current time. To provide context to our research findings, we first present an overview of how the participants spend their time in regard to their courses:

Content creation and maintenance: Instructors typically spend 6-8 hours, per class meeting, to create the content for a course. However, most of the interviewees mentioned this is only for the first time they teach, and in the future terms they focus on content maintenance which takes 2-3 hours per class meeting.

Lecture preparation: Instructors reported spending 10 minutes to 2 hours on lecture preparation depending on if they have taught the course before, their general teaching experience, and the tools they use.

Helping students: Instructors reported spending considerable time helping students individually or in small groups. Moreover, they invest even more time into creating efficient infrastructures for students to get additional help (e.g., TAs, study groups, online discussion boards, etc.).

Grading and feedback: Grading and giving feedback largely varies based on availability and skills of teaching assistants (TAs) as well as how much automation has been adopted in the course. Instructors reported spending up to 50% of their time on grading.

In the remainder of this section, we share the findings of this study to answer our research questions regarding pain points and workarounds.

3.1 RQ1: What did instructors wish they could change?

To answer our first research question, we asked instructors about challenges, pain points, and what they wish they could change. We categorized the responses into the themes below:

3.1.1 Where are students struggling? Instructors face substantial barriers in understanding what students are struggling with and where they are spending their time. In particular, instructors often only see the final output (e.g., homework submissions) but do not see the process that students go through to arrive at their solution. P27 framed this clearly: "*The challenge I have is knowing what students struggle with outside of class. I can't see where they get stuck and many don't ask questions.*" Similarly, P17 discussed the need to know where a given student is: "*I want to see the progress for one student. They might be making the same mistake over and over in different assignments.*". P21 expressed that if they could answer these questions, then they could "*apply some pedagogy there*". P30 told us "*students are doing lots and lots of things but I can't process all they are doing*" and that it is a "*lot of pipelines all strung together*". Only possible option at the moment is "*to scour all different places to understand students progress—code, Q&A, quizzes, git logs, etc.*".

Instructors mentioned they want to "*do things more in class that guarantees that [students] are not just sitting there silent. But kind of force them to wrestle with the ... issue on the on the table*" (P5). However, the current tooling available to instructors doesn't allow them to verify this live during class sessions. P5 continued, "*I basically got the out of class situation like that covered. It's the in-class version...*".

3.1.2 Answering students' questions. Students do tend to ask many questions, possibly in an attempt to overcome their struggles. However, these questions can overwhelm instructors and TAs during class, at office hours, through email, and through other software like Piazza. P18 shared their experience: "*Although the presence of tutors helped addressing some of the queries raised by the students, it was difficult to address all of them in an organized fashion, as much*

of it was lost in the barrage of questions that were raised.". They continued, especially in a large course, it is "*difficult to pay attention to individual doubts of students*". Furthermore, the questions are "*repetitive in nature*" (P17) and "*variations of the same*" (P25). To complicate this further, questions are often asked in batches, or as P19 put it, "*If they don't understand an assignment or get stuck, they will wait until last minute or after the due date to say something.*", which puts a strain on instructors and TAs to respond promptly.

It is particularly problematic to troubleshoot technical issues related to the student's computer or development environment. P23 said that the hardest questions from students are of the type, "*Why isn't Docker working on my laptop anymore?*" or "*Why does Python no longer exist?*" These questions are also "*the most painful thing*" for P23's teaching assistants to address.

3.1.3 Limited TA support. Towards supporting instructors in answering questions and identifying where students are struggling, departments will hire teaching assistant. However, these TAs are a finite and highly sought after resource in departments. Many institutions, especially community colleges and smaller institutions, "*struggle with TA support*" (P17). In fact, several instructors revealed that they have little to no TA support beyond the large introductory courses. As P3 said, "*TAs... I wish we had those*".

Instructors mentioned that even when they do have TAs, they may require considerable time to manage and may not be reliable. As P17 put it, "*TAs have their own biases ... some TAs are not mature programmers*". They continued that in some cases TA are limited by experience, for example "*TAs sometimes only run the unit tests and never read the code, [so] two submissions that were nearly identical, but one got [high] marks and the other got [low] marks*". Although P26 said they had sufficient TAs in their courses, they warned that the TAs also ask them a significant number of questions.

Other universities have restrictive policies on the work that TAs can perform. For example, "*in terms of grading, we have kind of a policy that, TAs are allowed to grade multiple choice questions, but for other questions like open-ended questions or coding problems, they're not really expected to grade those unless [instructor is] there to supervise them*" (P10), so as a result it is just easier for the instructor to grade directly. In a different institution, the instructor explained "*one of the university policies is [that] we don't allow students to grade other students work*" (P3). Similarly, another instructor shared that in their department, TAs are only allowed to lead group study sessions and manage in-class activities, but cannot grade.

3.1.4 Grading & feedback. Grading is a time-consuming tasks that nearly all of the instructors mentioned. There are many dimensions to grading, including returning grades to students in a timely manner, providing high-quality qualitative feedback, covering all artifacts that students produce, transparency in why points were taken off, and being consistent and fair across students and assignments. Instructors gave conflicting interpretations of grading as either a chore or an opportunity. For example, P9 said, "*grading is probably the biggest burden of the courses*" and P20 said, "*grading is an impossible task*". In contrast, P13 prefers to grade things themselves even if they has TAs "*because [of] the feedback I can get from ... their homework and assignments*".

A particularly challenging aspect of grading is developing, sharing, and adhering to rubrics. P27 designs their grading rubrics to minimize time spent grading. P20 said they makes a new rubric that focuses on different metrics for almost every assignment but knows this "*is not a great solution*". P23 emphasized "*transparency in grading and feedback is a priority*" and said that a good rubric is the key. They continued about the need for high-quality rubrics, "*in an ideal world, they can self grade or know exactly why they got their grade*" and one way they have done so is through "*boolean*" grading that eliminates subjectivity. However, even this is still time consuming for P23 and their TAs.

Although automated grading does exist, the "*human touch*" is valuable to instructors, especially as they deal with increasing enrollment. P22 said, "*our program is pretty well scaled, and it is a huge challenge to give quality feedback.*". P23 remarked that they wish for more automation of mundane tasks, but expressed that they are strongly opposed to automating feedback to students: "*I think this is the wrong direction for education. Striping away community and humanity from learning.*". Other instructors (P26, P30, and P32) shared a similar sentiment that their contribution is that of one-on-one feedback and to help the students build a community in the classroom. The COVID-19 pandemic reinforced the need for these personal interactions for P32.

3.1.5 Course material preparation. Another common challenge for instructors is the high cost of developing or adopting course materials to use as examples, assignments, lecture notes, quizzes and supplemental reading in their courses. P15 mentioned they cannot change the whole assignments often because they "*just don't have the resources*". Additionally, "*finding those is one of the biggest challenges, because ... there are so many different resources here and there. This does that, this does that ... [instructors] just don't have the time, especially at a smaller institution*" (P3). Having more examples or variety of assignments could benefit students both as additional resources as well as a way to prevent plagiarism.

Shifting courses to online or to hybrid modalities, such as a response to the COVID-19 pandemic, presented additional barriers and work for instructors. For example, P16 mentioned the problem of keeping students engaged, "*everyone's there at first and then it's just like monotonically decreasing. Until I get to end of semester, then it's like 20% or 15% of the students are there, and so that's my biggest problem right now of like how do I get people how do I incentivize lecture without just forcing them to go by holding their grade hostage.*" They continued, "*I've checked and there's very little few few people watching these videos*". As P8 explains, moving content online is an upfront investment, but may not take much time to maintain: "*When I was making the videos it was taking significant amount of time to make the videos and once those are done, then I don't spend very much time preparing now*". These challenges were also published in a recent case study of transitioning a course online because of COVID-19 closures [2].

3.1.6 Administrative tasks. There is managerial and administrative tasks involved with running any course, or as P16 put it, "*grunt work*". Examples include managing the social dynamics in class, preparing multiple course delivery formats, accreditation tasks, enforcing academic honesty policies, assigning teams, dealing with LMS quirks, transitioning to new software systems, and managing

individual student accommodations. While these tasks may not necessarily be the most practical use of an instructor's time, instructors usually have to do these tasks because of other limited resources.

Time spent on administrative tasks often prevented instructors from making improvements in the course. For example, P9 told us "*I would perfect the projects, if ... I could pause the clock.*", P4 told us they want to "*make things as close to industry as possible to help with smooth transitions from student to software engineer*", and P8 wanted to adopt mastery learning [9] model and "*have them do it again and do it again ... fix it and resubmit [until] I'm satisfied with it ... it is a really important learning process ... but takes a lot of time and effort on my part*". P22 expressed the desire to maintain notes on improvements for future versions of the same course but added that doing so is a chore.

Furthermore, identifying when students cheat was a concern for many instructors, especially in regards to moving courses online for COVID-19 and scaling to support higher enrollments. P19 explained that they have a 30-page syllabus to make it clear what is allowed and not allowed, along with quizzes covering academic dishonesty. P17 and P29 mentioned that it is not feasible for them to put as much time into checking for plagiarism as they should, but it is important to them.

3.2 RQ2: What are current attempts of addressing pain points?

To answer our second research question, we analyzed and categorized the current efforts used by instructors to address the pain points they mentioned in the previous section, which includes software tools and pedagogical techniques:

Interactive textbooks/exercises: Two of the instructors that we interviewed had created their own interactive textbooks, one based on Jupyter Notebooks and another using a proprietary interactive platforms that they developed. As instructors explained, interactive textbook "*fixes a problem we had before*" (P14) which was keeping content up-to-date. Interactive (executable) textbooks allows instructors to ensure the material remains up to date through a continuous integration mechanism. P14 said programming related textbooks "*come with all the usual well maintenance problems of code*", and with their interactive textbook "*when somebody, somewhere out there in the world changes some Python package, such that my code ... no longer works, I get notified the day after*" (P14).

Additionally, instructors mentioned they were able to use their interactive textbook to encourage a more active learning experience. With a traditional textbook students can read the code "*but well, not much fun to read code, you won't be able to execute it. But [in interactive format] it's more fun to actually toy with it*" (P14) right in their web browser, while they read the course material or watch a lecture. However, a recent study found that students had significantly fewer interactions with their eBook and fewer study days during the pandemic than they had before [31].

Online IDEs and code visualizers: Some instructors adopted online IDEs and code visualizers which may increase engagement in class and in assignments. Such tools have been previously studied and found to create a faster feedback cycle for students [15] and eliminate computing environment configuration issues. P12 said "*we want to give the students a uniform platform, we don't want them to need to switch to [a separate IDE] ... WebLab is really an IDE in the browser. So we code in a window in a browser and we push compile and outcomes everything", they continued "We also have unit tests in there ... The students can program and they can run their solution against the unit tests that we provide.*"

Another instructor from a smaller university mentioned they adopted using Java Visualizer[1] which an online tool based on Guo's Python Tutor [12]. This online tool allows students run a code snippet in their browser, visualize it and share their snippet using a permanent link generated by the tool. In this course, Java Visualizer was used for teaching the coding examples, as well as student submissions. When we asked the instructor what pain point they tried to address by adopting this tool, they mentioned, Java Visualizer has helped their students who are new to programming "*so that they can see what is happening when they're coding in memory and they can immediately see what's occurring*" (P3). Additionally, they mentioned some of their students did not have access to a capable computer, so having everything online in the browser alleviated environment issues. Other institutions aimed to eliminate these technical issues by standardizing the computers used or by moving to cloud-based software (e.g., P3, P10, P17, P24, P25, and P27).

Automated grading: Instructors have adopted several different tools and approaches for automated grading. Two examples of automated grading implementations used Web-CAT[2] [7], and GitHub Actions[3] to run an internally developed grading tool. This automation setup requires significant time investment by the instructors and makes changing the content more difficult [17]. However, in both cases the instructors were able to reduce the grading time and the TAs could simply "*look at the code to catch things that automated test does not catch*" (P5) instead of manually checking everything. This has allowed the instructors and TAs use the time they saved to help students in the learning process. Another example of that is Harvard's Check50 tool [27], which is an API-based tool that gives feedback to students before assignments are submitted as well as automates portions of the grading. This approach may also help the instructor promote mastery learning [9]. As P5 explained with this approach, their students "*can submit as many times as they want. Looking at the feedback, looking at the results of those tests and fix anything that those tests address and resubmit.*"

Flipped classroom: Several instructors mentioned they switched to a flipped classroom method of teaching or some hybrid model which has enabled more time for in-class interaction with students. This style involves assigning readings and pre-recorded lecture videos for students to watch and use the class meeting time to answer questions and work on interactive live coding. This change of teaching style was primarily adopted as a result of COVID-19 pandemic in 2020 [2], however instructors decided to continue to use this method even though universities are back to normal operations at the time of our study. As P15 explained, they had two main motivations to keep using a flipped classroom. First is that it has allowed them to have extra time to dedicate for interaction with the students in the class. And second is the "*the feedback ... from students the last couple of years is that they actually prefer having this lightweight flipped classroom model where they can just watch*

[1] https://pythontutor.com/java.html
[2] https://web-cat.org/projects/Web-CAT
[3] https://github.com/features/actions

the theory part, they can scroll back, they can pause they can try stuff on their own. And in the live lecture they get the interaction with the lecturer".

Peer instruction: One of instructors mentioned they adopted extra "office hour" time with small groups of students to run peer instruction [22] sessions which was seen as a positive change to increase understanding of concepts and engagement to work on examples together. However, as P8 explained this approach also had some weaknesses, "*it took a lot of time because I had to have so many meetings to have meetings at small, so it was not very efficient use of my time.*" Several instructors shared that their departments now require student-led study groups for core courses, which they believed to be a positive addition without taking time from instructors (P26, P28, and P32). Research supports the use of UTAs as an effective means to scale classes through peer instruction and student groups [18]. Alternatively, one school experimented with using a department-wide question and answering site (essentially a private Stack Overflow), with promising results [14].

4 LIMITATIONS

Our qualitative approach in this study may introduce certain limitations to the results. First, the findings may carry a level of interpretation of the researchers as they analyzed the responses in the interviews using thematic analysis. Second, our methodology uses *semi-structured* interviews and by definition, the questions asked during the interview or their order may slightly change based on the conversation. These variations may have some effects on the responses we received from instructors. Third, we covered the transitional changes in courses material and teaching styles as a result of COVID-19 pandemic as well as some associated pain points (3.1.5), however, it is difficult to predict whether any of those changes would revert to pre-pandemic norms over time and when. Finally, although the number of our participants was larger than most of the related studies, our findings may not be representative across demographics.

5 RELATED WORK

In both Lau et al. [17] and our work, the overarching goal is to find challenges that instructors experience. However, Lau et al. [17] focuses on challenges related to *maintaining content* in a *large scale data science course*, while we focus on challenges related to *assessments, integration of technology in class and coding* that can potentially effect pedagogy in different computer science classroom and institutions sizes. Lau et al. [17] method also differs in using case studies and previous experience while we conduct semi-structured interviews with computer science instructors.

Other related works with similar goals are from Yadav et al. [29, 30]. These studies have a goal of understanding challenges and experience of computer science teachers in K-12, with a focus on the increasing need for training new instructors with the growing computer science education demand. Similar to this study, Yadav et al. [29, 30] work used interviews with the instructors and qualitative analysis as the method for conducting the study. Some of the findings from this study, and the work by Yadav et al. [29, 30], such as grading challenges, are common and hold relevant. More broadly speaking, our study can also relate to other studies that offer a solution to pedagogy challenges in computer science classrooms.

6 DISCUSSION

Our findings categorized the pain points of CS instructors, and described some of the workarounds that instructors have tried in their workflow so far. We present implications for CS education researchers, instructors, and toolsmiths for reducing instructor pain points, and improving learning experiences for students.

Helping instructors surface student struggles: Instructors had low visibility into the student struggles (3.1.1), seeing where they got stuck, or identifying issues and answer questions in a timely fashion (3.1.2). Additionally, when instructors provide feedback or grade submissions (3.1.4), gathering information about student activities involved many "*mechanical actions to get all the info and put it in one spot*". For example, if an instructor wanted to verify if students performed effective code reviews of pull requests and made sufficient contributions to a project, they would have to navigate and visit each pull request to manually inspect review comments, and then navigate through the commit logs, possibly accounting for nuances such as commits on different branches or possible peer-coding activities. Often surfacing simple information would have helped instructors, for example, P16 used mastery grading, but their gradebook system did not display number of attempts, making it difficult to see if a student was "*trying a bazillion times*".

In general, instructors need help discovering and consolidating data that's "*in a bunch of places to better find the students that are struggling*" (P16). Some research efforts offer a promising start. Mysore and Guo [19] profiled student activity on tutorials and overlaying heatmaps of activity hotspots, encountered error messages, and embedded screencast videos of user actions. Adopting and extending this approach in other contexts, can help surface unknown struggles and provided actionable feedback for improving instructional material and assignments. Learning dashboards [28] have been used in a variety of contexts, including monitoring progress on assignments [6], visualizing group work contributions [10], and guiding students in self-reflection [24]. However, capturing these activities across a suite of tools, customizing and personalizing analysis to course content, and deploying these systems remain a challenge. Finally, simple interventions, such as weekly surveys can identify struggling students or teams [23].

All together, visibility into student struggles can help instructors identify misconceptions, sources of frustration, and problems in instructional material. Once struggles are better known, instructors will have the opportunity explore "*less defined assignments and then later dive in to see what they're doing*" (P21) and use the insights as "*guard rails for rabbit holes*".

Helping instructors leverage automation for class operations: Instructors frequently spent time on tasks that were "*repetitive in nature*" (P17) and "*variations of the same*" (P25), including answering questions (3.1.2), preparing assignments and quizzes (3.1.5), and administrative tasks (3.1.6). For some instructors, these problems could be overcome by scaling with more TAs (25 TAs on average for P12's class), but for many other instructors, they lacked TAs or TAs were limited in what actions they were allowed to do (3.1.3).

Automation of class operations offers a potential way to scale teaching efforts, allowing instructors to spend more time with students that need help or developing new course material. While decades in the making [3, 26], today AI systems have improved and with the introduction of large-language models (LLMs), such as OpenAI Codex[4]. Such models have been used recently for automatically generating "novel" and "applicable" programming exercises [25], generating solutions for introduction assignments [8], generating final exams [33], and answering questions [16]. Along with traditional AI tutor systems (e.g., AutoTutor [21]), virtual TAs (e.g., Jill Watson[11]), and approaches such as automated program repair [32], these systems offer considerable support for reducing the effort involved in aiding struggling students, responding to questions, preparing assignments, managing TAs, and grading.

However, automation brings its own challenges. Autograders may be viewed by students as unforgiving and opaque since they do not adequately explain why points were taken off. For example, P10 described when an interactive textbook gives a grade, it deducts points even if there is only a whitespace difference with the solution. They continued this is "*in my opinion, unreasonable and my students struggle so much with it and they spend hours trying to get the white space correct in their program when in reality that's not what I want them spending spending time on*". P17 shared a similar observation, that autograders are "*too harsh*" and still require a human to review the grades. They described an example where a student received a 0 for a submission to the autograder, but the cause was a minor issue that should have only resulted in a few points deduction.

Although instructors can automate their classes using LLM-based technologies, it also means students can automate their assignments. In fact, there is a growing concern[5] that Copilot, an intelligent code suggestion tool that uses Codex, can complete homework assignments in seconds with little conceptual knowledge. A professor recently wrote a satirical article on how to cope with students using Copilot in your class[6]. Others have taken to Twitter to discuss how to design "Copilot-proof" assignments[7].

Instructors stressed the importance of promoting a diverse and inclusive classroom, which automation may negatively impact. For example, P17 worried that when a student sees the feedback from their autograder, the student will lose confidence because of how strict the system is. P29 has trouble finding assignments and lecture notes online that are designed to be more equitable and inclusive. They explained that they do not want homeworks consisting of abstract problems to be solved in a terminal, but rather the homeworks should be culturally relevant. Furthermore, they stated that having a database or system to generate these assignments would push the classroom to be more diverse.

Helping instructors reduce friction in course delivery: Instructors faced obstacles maintaining code exercises and course material (3.1.5), troubleshooting technical issues in student's computing environments (3.1.2), keeping students engaged (3.1.5). Common workarounds, included outsourcing course content to interactive textbooks, and integrating cloud-based programming environments. Benefits included, monitoring student progress, supporting active learning experiences for students, reducing questions related to technical environment issues.

However, these workarounds often did not completely resolve these obstacles and other desires remained. Instructors using existing interactive textbooks, such as zyBooks, could not modify material, and found content is often difficult to align with existing course structure and were "*pedagogically not nearly as solid*" (P5). P10 quickly realized students could not perform many basic operations, such as working with file systems, for their course because "*zyBooks is ... just kind of a simulation*". Similarly, P14, who made their own interactive textbook based on Jupyter notebooks, had to invest significant time. They explained, "*many things I had to invent and build myself. I had to build quite an infrastructure around this to make this whole thing workable for me*". They continued "*Jupyter Notebooks typically are not being used by programmers ... they are typically used by data scientists*" so it has a different purpose and it's "*awful in terms of code*" support out of the box. Finally, instructors often had to navigate usage limits as well as longevity associated with cloud-based solutions. For instance, P28's department adopted a cloud platform, however that platform "*disappeared a year or two after migrating to it*", and as a result, they became very hesitant to invest in adopting external tools.

Advances in educational technology seem like a game of whack-a-mole, addressing one challenge often brings another one. For example, while interactive textbooks have several benefits and reduce certain pain points, instructors now have to deal with significantly reduced engagement associated interactive textbooks [31]. Moving forward, educational technology needs a design that simultaneously addresses or blends several needs, such as improving student engagement, interactivity and liveness, peer interaction, and advanced computing environments. For example, a participant described a live document that embedded code editors, slides, and quizzes, backed by a real environment, and was shareable by instructors and peers "*that would be a perfect blend*" (P9).

7 CONCLUSION

In this work, we conducted interviews with 32 computer science instructors about obstacles, workarounds, and desires when teaching their courses. We identified several pain points, including limited access to resources, repetitive tasks, such as manually gathering information about student activities for grading, or answering variations of the same question, troubleshooting technical issues in students' computing environments, and endless administrative tasks. Common workarounds included automated grading, peer instruction, outsourcing course content to interactive textbooks, and integrating cloud-based programming environments. Instructors identified several desires, including (1) more visibility into their students struggles, problem-solving process, and programming environments, (2) better support for automating repetitive tasks, and (3) improved interactive environments and streamlined course delivery. In general, we believe that investing in the educators will improve the learning experiences and quality of life for everyone.

REFERENCES

[1] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

[4] https://openai.com/blog/openai-codex/
[5] https://twitter.com/search?q=copilot%20homework&src=typed_query
[6] https://itnext.io/coping-with-copilot-b2b59671e516
[7] https://twitter.com/deliprao/status/1557913160140656640

[2] Alexander Brooks, Caroline Hardin, Jennifer Scianna, Matthew Berland, and Laura Hobbes Legault. 2021. Approaches to Transitioning Computer Science Classes from Offline to Online. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Virtual Event, Germany) *(ITiCSE '21)*. ACM, 81–87. https://doi.org/10.1145/3430665.3456366

[3] Maud Chassignol, Aleksandr Khoroshavin, Alexandra Klimova, and Anna Bilyatdinova. 2018. Artificial Intelligence trends in education: a narrative overview. *Procedia Computer Science* 136 (2018), 16–24. https://doi.org/10.1016/j.procs.2018.08.233 7th International Young Scientists Conference on Computational Science, YSC2018, 02-06 July2018, Heraklion, Greece.

[4] Amy Cook, Alina Zaman, Eric Hicks, Kriangsiri Malasri, and Vinhthuy Phan. 2022. Try That Again! How a Second Attempt on In-Class Coding Problems Benefits Students in CS1. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1* (Providence, RI, USA) *(SIGCSE 2022)*. ACM, 509–515. https://doi.org/10.1145/3478431.3499362

[5] Computing Research Association (CRA). 2017. The phenomenal growth of CS Majors since 2006. Retrieved July 20, 2022 from https://cra.org/data/generation-cs/phenomenal-growth-cs-majors-since-2006/

[6] Nicholas Diana, Michael Eagle, John Stamper, Shuchi Grover, Marie Bienkowski, and Satabdi Basu. 2017. An Instructor Dashboard for Real-Time Analytics in Interactive Programming Assignments. In *Proceedings of the Seventh International Learning Analytics & Knowledge Conference* (Vancouver, British Columbia, Canada) *(LAK '17)*. ACM, 272–279. https://doi.org/10.1145/3027385.3027441

[7] Stephen H. Edwards. 2003. Improving Student Performance by Evaluating How Well Students Test Their Own Programs. *J. Educ. Resour. Comput.* 3, 3 (sep 2003), 1–es. https://doi.org/10.1145/1029994.1029995

[8] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Australasian Computing Education Conference* (Virtual Event, Australia) *(ACE '22)*. Association for Computing Machinery, New York, NY, USA, 10–19. https://doi.org/10.1145/3511861.3511863

[9] James Garner, Paul Denny, and Andrew Luxton-Reilly. 2019. Mastery Learning in Computer Science Education. In *Proceedings of the Twenty-First Australasian Computing Education Conference* (Sydney, NSW, Australia) *(ACE '19)*. ACM, 37–46. https://doi.org/10.1145/3286960.3286965

[10] Niki Gitinabard, Sarah Heckman, Tiffany Barnes, and Collin Lynch. 2022. Designing a Dashboard for Student Teamwork Analysis. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 446–452. https://doi.org/10.1145/3478431.3499377

[11] Ashok K. Goel and David A. Joyner. 2017. Using AI to Teach AI: Lessons from an Online AI Class. *AI Magazine* 38, 2 (Jul. 2017), 48–59. https://doi.org/10.1609/aimag.v38i2.2732

[12] 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* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 579–584. https://doi.org/10.1145/2445196.2445368

[13] Sara Hooshangi, Margaret Ellis, and Stephen H. Edwards. 2022. Factors Influencing Student Performance and Persistence in CS2. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 286–292. https://doi.org/10.1145/3478431.3499272

[14] Stefan Hugtenburg and Andy Zaidman. 2022. First Impressions of Using Stack Overflow for Education in a Computer Science Bachelor Programme. 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, 1146. https://doi.org/10.1145/3478432.3499046

[15] Lennart C.L. Kats, Richard G. Vogelij, Karl Trygve Kalleberg, and Eelco Visser. 2012. Software Development Environments on the Web: A Research Agenda. In *Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Tucson, Arizona, USA) *(Onward! 2012)*. ACM, 99–116. https://doi.org/10.1145/2384592.2384603

[16] Kalpesh Krishna, Aurko Roy, and Mohit Iyyer. 2021. Hurdles to Progress in Long-form Question Answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 4940–4957. https://doi.org/10.18653/v1/2021.naacl-main.393

[17] Sam Lau, Justin Eldridge, Shannon Ellis, Aaron Fraenkel, Marina Langlois, Suraj Rampure, Janine Tiefenbruck, and Philip J. Guo. 2022. The Challenges of Evolving Technical Courses at Scale: Four Case Studies of Updating Large Data Science Courses. In *Proceedings of the Ninth ACM Conference on Learning @ Scale* (New York City, NY, USA) *(L@S '22)*. Association for Computing Machinery, New York, NY, USA, 201–211. https://doi.org/10.1145/3491140.3528278

[18] Diba Mirza, Phillip T. Conrad, Christian Lloyd, Ziad Matni, and Arthur Gatin. 2019. Undergraduate Teaching Assistants in Computer Science: A Systematic Literature Review. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. ACM, 31–40. https://doi.org/10.1145/3291279.3339422

[19] Alok Mysore and Philip J. Guo. 2018. Porta: Profiling Software Tutorials Using Operating-System-Wide Activity Tracing. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (Berlin, Germany) *(UIST '18)*. Association for Computing Machinery, New York, NY, USA, 201–212. https://doi.org/10.1145/3242587.3242633

[20] Nadia Najjar, Anna Stubler, Harini Ramaprasad, Heather Lipford, and David Wilson. 2022. Evaluating Students' Perceptions of Online Learning with 2-D Virtual Spaces. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 112–118. https://doi.org/10.1145/3478431.3499396

[21] Benjamin D Nye, Arthur C Graesser, and Xiangen Hu. 2014. AutoTutor and family: A review of 17 years of natural language tutoring. *International Journal of Artificial Intelligence in Education* 24, 4 (2014), 427–469.

[22] Leo Porter, Cynthia Bailey Lee, and Beth Simon. 2013. Halving Fail Rates Using Peer Instruction: A Study of Four Computer Science Courses. 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, 177–182. https://doi.org/10.1145/2445196.2445250

[23] Kai Presler-Marshall, Sarah Heckman, and Kathryn T. Stolee. 2022. Identifying Struggling Teams in Software Engineering Courses Through Weekly Surveys. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 126–132. https://doi.org/10.1145/3478431.3499367

[24] Jose Luis Santos, Sten Govaerts, Katrien Verbert, and Erik Duval. 2012. Goal-Oriented Visualizations of Activity Tracking: A Case Study with Engineering Students. In *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge* (Vancouver, British Columbia, Canada) *(LAK '12)*. Association for Computing Machinery, New York, NY, USA, 143–152. https://doi.org/10.1145/2330601.2330639

[25] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1* (Lugano and Virtual Event, Switzerland) *(ICER '22)*. Association for Computing Machinery, New York, NY, USA, 27–43. https://doi.org/10.1145/3501385.3543957

[26] Janet Ward Schofield, Debra Evans-Rhodes, and Brad R. Huber. 1990. Artificial Intelligence in the Classroom: The Impact of a Computer-Based Tutor on Teachers and Students. *Social Science Computer Review* 8, 1 (1990), 24–41. https://doi.org/10.1177/089443939000800104 arXiv:https://doi.org/10.1177/089443939000800104

[27] Chad Sharp, Jelle van Assema, Brian Yu, Kareem Zidane, and David J. Malan. 2020. An Open-Source, API-Based Framework for Assessing the Correctness of Code in CS50. 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, 487–492. https://doi.org/10.1145/3341525.3387417

[28] Katrien Verbert, Sten Govaerts, Erik Duval, Jose Luis Santos, Frans Assche, Gonzalo Parra, and Joris Klerkx. 2014. Learning Dashboards: An Overview and Future Research Opportunities. *Personal Ubiquitous Comput.* 18, 6 (aug 2014), 1499–1514. https://doi.org/10.1007/s00779-013-0751-2

[29] Aman Yadav, Sarah Gretter, and Susanne Hambrusch. 2015. Challenges of a Computer Science Classroom: Initial Perspectives from Teachers. In *Proceedings of the Workshop in Primary and Secondary Computing Education* (London, United Kingdom) *(WiPSCE '15)*. Association for Computing Machinery, New York, NY, USA, 136–137. https://doi.org/10.1145/2818314.2818322

[30] 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.

[31] Iman YeckehZaare, Gail Grot, Isadora Dimovski, Karlie Pollock, and Elijah Fox. 2022. Another Victim of COVID-19: Computer Science Education. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 913–919. https://doi.org/10.1145/3478431.3499313

[32] Jooyong Yi, Umair Z. Ahmed, Amey Karkare, Shin Hwei Tan, and Abhik Roychoudhury. 2017. A Feasibility Study of Using Automated Program Repair for Introductory Programming Assignments. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 740–751. https://doi.org/10.1145/3106237.3106262

[33] Sarah Zhang, Reece Shuttleworth, Derek Austin, Yann Hicke, Leonard Tang, Sathwik Karnik, Darnell Granberry, and Iddo Drori. 2022. A Dataset and Benchmark for Automatically Answering and Generating Machine Learning Final Exams. https://doi.org/10.48550/ARXIV.2206.05442