# CS1 Instructors: Flexibility in Content Approaches is Justified, and can Enable More Cross-University Cooperation

Frank Vahid
Computer Science and Engineering
University of California, Riverside
Riverside, California, USA
vahid@cs.ucr.edu
Also with zyBooks

## ABSTRACT

Many CS1 teachers focus on specific content approaches in CS1. Some want objects early, some functions early, some decisions/loops first. Some put emphasis on language details, some on language-neutral problem solving. Some demand real-world IDEs, code versioning tools, industry-quality comments, specifications, documentation, or test coverage. While the focus shows teachers care and may indeed provide benefits, those specific focuses can also prevent increased cooperation among universities in defining a more "common" CS1 curricula. With a more common curriculum, better content and tool support is enabled due to economies of scale. Such cooperation could yield a more powerful approach to teaching CS1, elevating the role of CS1 instructors. CS1 instructors, by being more flexible in their content approaches, may help show the college education community the great benefits of increased cooperation among universities, especially in the design and delivery of introductory gateway courses taken by large numbers of students. We describe results of discussions with over 100 instructors at over 50 universities during the past decade, highlighting frequently-stated content approaches that have little or no evidence supporting the approach and that may hamper cooperation, and we end by encouraging flexibility in content approaches to enable the community and publishers to provide better CS1 support.

## CCS CONCEPTS

• Social and professional topics - Professional topics - Computing education - Computing education programs - Computer science education - CS1

## KEYWORDS

CS1, programming, teaching, cooperation, common courses



*SIGCSE 2024, March 20–23, 2024, Portland, OR, USA.*

ACM ISBN 979-8-4007-0423-9/24/03.
https://doi.org/10.1145/3626252.3630816

**ACM Reference format:**

Frank Vahid. 2024. CS1 Instructors: Flexibility in Content Approaches is Justified, and can Enable More Cross-University Cooperation. In Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20--23, 2024, Portland, OR, USA. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3626252.3630816

## 1 INTRODUCTION

CS1 is a key class for attracting and keeping students in computing majors, and for giving non-majors a useful skill in future coursework and careers. Unfortunately, CS1 grades can be low and fail rates high [1, 2]. Many pedagogical techniques show great promise for improving CS1 grades and fail rates, such as pair programming, peer instruction, active learning, program scaffolding, many-small-programs, auto-grading, and more [3, 4, 5, 6, 7, 8].

Instructors strongly request more support items for their CS1 course offerings, including pre-made programming assignments, homeworks, quizzes, exams, solutions, instructor collaboration, support for pair programming, examples for active lectures and peer instruction, etc. [9] However, at the same time, instructors each retain focus on various specific content approaches in their courses. Such focus is often intended to yield a better CS graduate, but unfortunately can hamper the ability of the CS education community or publishers to create those requested support items. Examples of such specific approaches include teaching objects early or late, teaching functions early or late, or teaching detailed documentation techniques. These hamper the ability to create support items because such items differ greatly depending on the approach; for example, a series of quizzes differs dramatically in an early-functions course vs. a late-functions course.

Yet, somewhat surprisingly, most of those content approaches have little convincing research showing that they lead to better CS graduates or even to CS1 success. For example, instructors may argue teaching objects early is essential to CS students becoming good object-oriented programmers later, but no research supports this. As such, even though many position papers are framed around previous work and data, this paper's framing is founded on the lack of such papers. Instead, the main data supporting this paper is the "common sense" fact that courses having different approaches still ultimately yield good CS graduates -- "there are many paths to the same destination".

A key issue is that focus on specific content approaches prevents the community or publishers from providing better support items for CS1. For example, a standard set of randomized automated online quizzes or exams could save instructors time, could be higher quality, could include analysis and continual refinement, and could enable modern frequent testing techniques that improve learning and support mastery learning or specifications grading techniques. But, the series of quizzes/exams for an early objects course is drastically different than for a late objects course, thus splitting the community in two. Then, early versus late functions splits the community into four. Requiring documentation of every class or function/method splits the community into eight. And so on. Creating support items becomes exceedingly difficult. Thus, open source repositories, and publisher-provided items, tend to be far more limited in scope.

As such, this paper suggests that instructors take a more flexible attitude toward their CS1 content approaches, and instead focus on increasing the commonality among CS1 courses across universities, so that the CS education community and publishers can create better support items around those CS1 courses. Instructors have been suggesting for years that consensus is not possible [10], but perhaps today the community has enough information to recognize what matters and what does not in improving CS1 courses.

In this paper, we describe common CS1 approaches that (unintentionally) hamper the ability of creating better support items for CS1 courses, based on visits and discussions/interviews at over 50 CS departments across the U.S. We also discuss the potential benefits of having more commonality and cooperation in the offering of CS1's across universities.

## 2 DIFFERENT CS1 FOCUSES

Over the past 10 years, we have visited over 50 CS departments, to consult on improving CS education, including the idea of increased CS1 cooperation across universities. While instructors in general realize that increased cooperation could yield better content and tools, they often say "But, we do [specific content approach] and we won't change that". In this section, we highlight various specific content approaches that seem to unintentionally prevent the community from increased cooperation, yet that either don't seem to provide much benefit, or may in fact lead to higher CS1 attrition.

One dividing focus in CS1 courses seems to be whether objects should be taught early or later. As object-oriented-supporting languages like C++ and then Java became popular, many CS1 instructors stated the firm belief that students should learn objects early, suggesting that not doing so prevents students from properly thinking of data and actions from an object-oriented, information-hiding perspective in their future education and careers. In contrast, others believe, as one teacher put it, "before teaching students about information hiding, shouldn't students first have some information to hide?" We have found no research that supports either side of the argument; papers on the topic often conclude the best approach is unknown [11]. One study examined learning outcomes in CS1 and found no significant difference [12] between early or late objects; another examined over one year and also found no difference [13].

Similarly, some firmly believe students should learn to write functions (methods) early, so students will think of programming as a set of smaller tasks composed to solve a larger problem, and thus avoid writing large blobs of code. A CS1 instructor at a large state university said that if they didn't teach functions early, then even after introducing functions, students would still put most of their code in main(). Others believe students should first master expressions, decisions, loops, and even arrays, and start to encounter problems with large or redundant code, so they can truly appreciate the need for functions. We could not find any papers substantially addressing those approaches.

Some argue that recursion should be taught early, while others say iteration is easier to learn [14,15]. Interestingly, a recent talk by Google's C++ lead questioned why recursion is taught at all [16] since it is rarely used in practice and highly problematic in commercial code.

With decades of opportunity to determine whether "early" or "late" approaches are better, the community seems no closer to knowing whether early or late approaches are actually better; in fact, proposals to discuss the early/late topic are still ongoing [17]. Given that neither approach seems to be evolving as a clear winner even after decades, it may be reasonable to conclude that in fact any of these approaches are reasonable, and thus choosing an approach is more a matter of instructor preference than a substantial matter of student learning or success.

Some C++ teachers believe vectors were designed to overcome limitations of arrays, so vectors should be taught and arrays forbidden or introduced later. Others believe students should know how things work under the hood and thus students should learn to manage data in arrays, so they will better understand and appreciate vectors later. Some go even further and say vectors should never be used [18]. More debates exist within C++, and similar debates exist within Python, Java, and C.

As for language, some teachers believe Python is superior for learning programming, due to ease of use, laxer rules, built-in libraries, and more, and is better for students because of the language's wide use in practice. Others believe languages like Java, C++, or C are better because they are strongly-typed and have stricter rules, so students learn to be more precise. Among those, some believe Java is superior to C++ due in part to its object-oriented support. Others believe C++/C are superior due to doing less work behind the scenes and having a less-verbose syntax. Among those, some believe C is better due to being simpler and closer to hardware, while others believe C++ is an improvement and thus there's no need to teach C.

The different approaches are often even more specific. Some instructors we spoke with insist that for loops should be taught before while loops, while others emphasize the reverse. Some insist on particular conventions related to variable names, variable initializations during declaration (allowed or not), multiple variable declarations per line (allowed or not), brace style, etc.

Some instructors believe that a CS1 should focus mostly on problem solving, independent of any language (using flowcharts or pseudocode) at least for a while, or using a commercial language but lightly. Others believe that a language must be learned in detail first, and that problem solving using a language is a higher-order skill to be learned later. Again, the literature provides no clear answer. Some research finds results to be about the same, concluding the choice is mostly personal preference of the instructor [19]. In fact, the language neutral approach at the college level may even de-motivate students who want to learn an authentic language or may hamper their ability to switch to text languages, unless hybrid block/text languages are used [20].

Some believe a CS1 course should help a student master a particular language, such as learning intricacies and nuances of C++ syntax and semantics. Others believe CS1 should be focused on the basic constructs common in most languages. Again, no papers we could find addressed showing whether one or the other approach was better. Some instructors believe overemphasis on syntax may deter many students and yield higher attrition.

Some CS instructors focus on specific items beyond content approaches, focusing on what we might call "rigor" items. For example, some believe CS1 students should learn industry-quality tools and/or practices, while others believe that industry tools/practices can be taught in later classes. Some emphasize use of professional IDEs, of widely-used code versioning systems like GitHub, or of the Unix OS including the file system, console I/O, and command-line tools, while others start with simpler more education-focused tools in CS1 and teach other tools in later classes [21]. Some emphasize writing quality comments, while others say extensive comments in CS1 are an "anti-pattern" since clear identifiers and function-use yield readable programs in CS1. Some require careful specification before programming, via listing of preconditions and postconditions before writing a function/method, extensive documentation, test case coverage, etc. Others care more that the student is able to get decent but working code sooner, to motivate students early, with students learning the more advanced professional approaches later as programs become more complex. Some emphasize large challenging projects in CS1, while others suggest using smaller programs for most of the course and transitioning into larger programs later. As is the case with most approaches, little or no evidence suggests that these approaches that emphasize a more "rigorous" CS1 -- using industry tools, using professional IDEs or GitHub, using console I/O, writing quality comments, having large challenging programs early, etc. -- actually make a long term difference, given that students have many years to learn these skills. Instead, many say that emphasizing such "rigor" in CS1 may contribute to the high fail rates in CS. For the program size issue, for example, using smaller programs has been shown to benefit students, who then transition to larger programs later without issues [22].

Many of these specific focuses were brought up in the context of considering adoption of common material, such as an OER (Open Education Resources) textbook or a publisher's textbook and supporting tool. For example, the following is a summary of one actual discussion, similar to many: "I can't adopt that material because I teach loops before branches, but the material uses branches in their loops chapter". When asked if the instructor considered teaching branches before loops, they said: "Programming is largely about data processing, so we need students to be processing data first, before they start making decisions about that data". When asked if there was any evidence showing that the ordering mattered, the instructor replied "No, I don't need evidence to tell me what's obvious. And, I've been teaching it this way for a long time and don't want to have to change my slides". That conversation has been repeated, in various forms, with dozens of instructors, each with a different focus, such as regarding brace style ("I don't want students to see the brace style in that material"), syntax, early functions, and more.

## 3 MULTIPLE PATHS

Regarding CS1, a key position argued in this paper is:

***There are multiple paths to the same destination.***

The multiple paths above refer to different CS1 content approaches, and the destination refers to student success across CS courses and ultimately in their CS careers.

Of course, content choices do have some impact, and certain choices, like teaching go-to statements in CS1, could have strongly negative consequences later. And as noted, over-emphasis on rigor items too early may deter students from CS. Also, a bad teacher (e.g., unreasonably harsh or hostile) can cause much failure, and deter students from the CS major. But with rational content choices coupled with a competent teacher, most content variations seen today just don't seem to make that much difference in the long run.

What evidence supports the position that various content and process approaches can all lead to successful students? To some extent, one can view that the statement is largely self-evident, because hundreds of universities all use a variety of approaches, and yet turn out successful students. For example, Siegfried [23] reports the following language use in CS1/2 among 409 schools in 2019:

- Java -- 163
- Python -- 111
- C++ -- 83
- C -- 17

Surely, if one language was yielding dramatically better outcomes at the end of a 4-year degree, the community would know by now. But no such research findings exist. Rather, student success overall is likely not due to those early CS1 content choices, but more due to other features of how CS departments operate (like what kinds of instructors they employ, what resources they devote to giving students help, etc.). Thus, CS1 variety persists.

For this paper's purposes, we invited members of the SIGCSE email list (a list of computer science educators) to complete a survey, asking about features of their CS1 classes, and student success. 104 teachers responded. Results appear in Figures 1, 2, and 3. The survey results show that, despite much variety in course language and features, most students embark on successful careers, at least according to the instructors.

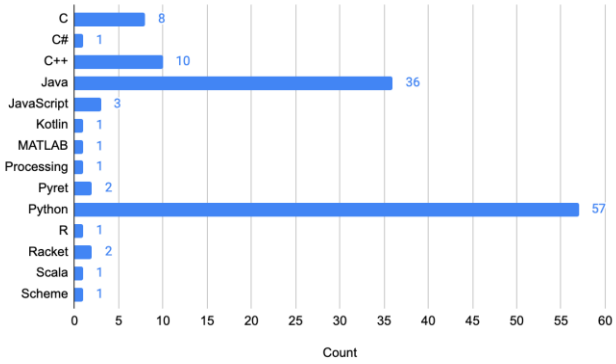


**Figure 1: Survey results for: What language does your CS1 use? (Check all that apply)**

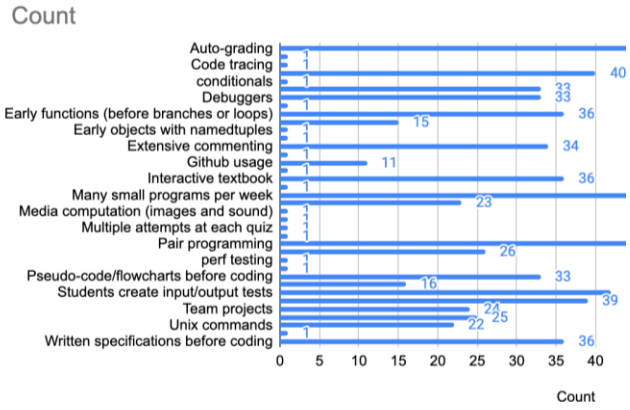


**Figure 2: Survey results for: What features does your CS1 include? (Check all that apply). Auto-grading count is 55, many small programs is 71 (cut off on right).**

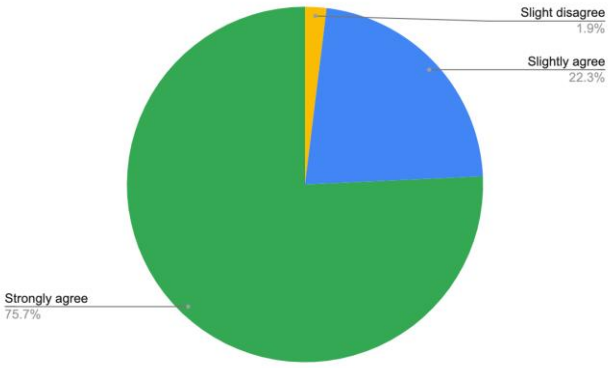


**Figure 3: Survey results for: Upon graduating, your CS students embark on good careers.**

Our school provides evidence that pedagogy and support matter greatly, and content approaches less so. In the 1990s and 2000s, our CS department had a freshman-retention rate of 60%, for 100-200 computing freshmen each year. Our CS1 used C++. Around 2010, we piloted a Python-based CS1 as part of an NSF project led by another university, but saw modest results. In 2012, we aggressively addressed pedagogy and support issues -- supportive instructors, interactive textbook, auto-grading, many-small-programs, flipped classrooms, allowed collaboration, and more. We did not change content -- still C++, still functions late, mostly the same exams, etc. With the pedagogy changes, our CS1 and CS2 grades improved dramatically, and freshman retention rate rose from 60% to 90% in 2013 and has been steady since. That 90% applies to any demographic: low income, first generation, underrepresented minority, or women, are all at ~90% retention.

The different content approaches surely make some difference in the short term. For example, teachers who have taught functions late and then tried functions early may see cleaner code by the term's end in their functions-early class, and thus prefer that approach. Or teachers may rely on github in CS2 and want incoming students already trained. But, these seem to be largely personal preferences or conveniences. Across 4 years, those early differences don't have lasting impact; students eventually learn to write good functions or use github as long as they are taught somewhere in the curriculum. "Retraining" habits might take time, but eventually students seem to get it. If such CS1 content issues significantly impacted student success in the long run (e.g., functions-late CS1's yield B.S. graduates who don't properly write functions), it would be quite shocking that nobody had published such data, despite 30+ years of intense CS education research.

## 4 WHY DOES THIS MATTER? COOPERATION

**(1) Distraction**: The focus on specific content approaches can distract instructors from focusing on pedagogy and support issues. Switching intro courses from Java to Python, for example, is a massive effort and yet may not provide expected benefits [24], and switching back is even more effort, as sometimes happens [25]. Effort is limited, and thus such content/process focus takes away from time that could be focused on developing peer instruction, setting up pair programming, integrating interactive learning content, etc. Believing such content/process items to be of paramount importance may also prevent people from even realizing that other things have bigger impacts. College instructors often still ignore fundamental principles of what is widely known to work in teaching [26], such as ensuring teachers are supportive and positive [6]. Active learning has been shown highly effective yet many instructors still use traditional lectures despite increased failure rates [27]. The Gates Foundation produced a memo (unpublished thus far) on college-level evidence based teaching, finding the most important factors for student success to include "active learning", "sense of belonging", "agency", "formative practice", and "transparency" -- all having big impacts on student success. Instructors might do well to put first things first, focusing primarily on these factors known to have big influences.

**(2) Limited cooperation**: Perhaps even more importantly, focus on specific CS1 content/process approaches preferred by local instructors leads to differences among classes that inhibit increased cooperation among universities. Today, thousands of CS1 instructors perform redundant work on class-related tasks,

like deciding on a topic schedule, selecting homework exercises, defining weekly programming assignments, creating quizzes and exams, providing help outside class time, trying to grade all these items in a timely manner, and more.

Today, many instructors see those tasks as the core of their job. But instead, if some number of CS1 instructors, perhaps whose courses were already "in the same ballpark", decided to agree on certain items (topic schedule, homework exercises, etc) to create a "common course,", several advantages would be gained, including:

- Less time creating those items per instructor
- Higher-quality items (more revision, more use)
- Increased automation (pooled resources can focus on automation)
- Round-the-clock help for students (shared help across schools, or a class-customized AI-based help system)
- Data analytics and continual improvement
- Catalyzation of an instructor community discussing ideas around teaching that "same" course

This approach is already done at many large schools with numerous CS1 course sections (e.g., 50) or dozens of offerings of the same course -- an instructor (or team) designs the structure of a single common course, and then instructors for each section/offering follow that common course. And, this is a popular model for AP Computer Science; common courses like [28] save teachers time, enable common training and an active community, and likely yield higher-quality courses.

In fact, one can view textbooks as being a step towards such common courses. Instructors give up some personalization, accepting the textbook author's main topic coverage, and maybe using the author's presentation slides, exam questions, programming assignments, and more.

zyBooks' growth in zyLabs usage [29] shows a benefit to the cooperation approach. They provide off-the-shelf programming assignments and a built-in program auto-grader. Instructors report large time savings in their own workload, averaging 9 hours per week of saved time -- a huge reduction. Some even report saving 20 hours per week. Yet, they also report improved student outcomes, saying they now had more time to help students, plan activities, etc. Those benefits required instructors to give up some personal content/process preferences (e.g., particular programming assignments, specific development tools, etc.), but in return yielded big gains for both instructors and students.

If CS1 instructors recognize that most CS1 content variations likely have little long-term impact, they may be more willing to cooperate more directly with other CS1 instructors. Specifically, groups of CS1 instructors might do well to come together and create "common courses" amongst themselves, including common:

- Topic arrangement and schedule
- Programming assignments
- Quizzes and exams
- Grade policies
- Data collection and analysis (anonymized for FERPA compliance)

Most CS1 instructors aren't allowed to change their programming language, so such groups might be based on their currently-used language (e.g., Python, or Java).

The common course might be designed by the group, or more likely by a small central team taking recommendations from the group, akin to how large institutions have a small course-design team and a larger team of instructors.

This approach will require giving up some local preferences. But in return, those instructors will get a more robust course, potentially improved student success, the ability to compare, and time savings that can be used to focus on higher-level things like improving pedagogical and support techniques.

The key pedagogical and support techniques to focus on may include:

- **Supportive teachers with rapport.** Successful CS1's seem to have supportive teachers -- those who make clear through words and deeds that, although CS1 isn't easy, they want all students to succeed. Those teachers also build rapport with their students -- the students "connect" with the teacher somehow. In fact, research shows rapport has a big impact on students staying in a class, and getting good grades. [6]
- **Active learning / peer instruction.** Students don't learn CS1 concepts/skills from long lectures, a point well-established in STEM in general. Successful CS1 teachers do some lecturing but largely use their time with students to engage the students in activities. For example, incorporating peer instruction into lectures has big impacts on reducing DFW rates [PoBa13]. Rapid polling, short activities, and other approaches work too.
- **Incremental outside activities with feedback:** Lecture time is limited. Extensive activities outside of lecture are really needed to learn CS1. That activity should be incremental -- properly "scaffolded" using the education term [3] -- so students aren't asked to bite off more than they can chew. Those activities need rapid feedback, so students know what they are doing wrong and can learn. Successful CS1s may use auto-graded online activities (interactive books, online quizzes, etc.).
- **Extensive help resources, including learning assistants** -- Some successful CS1s have numerous trained upperclassmen (juniors/seniors who did well in CS1) available to help CS1 students, often present in the students' programming lab room and often for long hours including evenings and weekends when students actually do their work [30]. Research shows the help from peers is often better than from expert teachers. As a bonus, the upperclassmen benefit as well, as teaching catalyzes their own learning while building social and communication skills. Other help resources include discussion forums, on-campus tutoring, and more. Pair programming is another way of providing help.

Clearly, content variety is still important -- many CS1 instructors should continue to explore and innovate. But not all 2000+ CS1 courses taught each year involve innovation; many if not most

instructors are just trying to teach a good course, and many are just trying to survive. And even for those who wish to innovate, one option is to use the common course as a base and then create variations thereof -- enabling comparison with the others teaching the common course "out-of-the-box"; one instructor told us that today they spend all their available time outside class just preparing materials and grading, and that a common course's time savings would finally enable them to experiment and innovate.

In fact, we organized such a "common course" for CS1 in Spring 2023, with 7 instructors at 7 different universities (including 2 community colleges) all teaching a nearly-identical course, using zyBooks for interactive auto-graded readings, homework, programming assignments, and quizzes. Midterm and final exams were provided as well, along with practice exams, plus a syllabus with weekly topics (the zyBook chapters were configured to match), grading policies, late exception request forms, and more. Instructors met bi-weekly. Students did well, and instructors unanimously recommended the approach for other instructors, highlighting time savings, a high-quality course, learning about state-of-the-art pedagogy and tools used in the course, and enjoying the camaraderie with the fellow instructors. Details are published in [31].

All those benefits required instructors to be more flexible in their content approaches -- a small loss, for a larger gain.

Some professors have noted that the self vs cooperation tradeoff exists in most human endeavors, and whereas people tend to naturally lead toward the self, focusing on cooperation can lead to many benefits.

## 5 SUMMARY

Many CS1 instructors today focus on specific content approaches, but with little evidence that such approaches substantially improve success. While itself not problematic, such focus can have two key drawbacks: (1) Such focus can distract efforts away from proven-successful pedagogical approaches, (2) Such focus can inhibit increased cooperation among universities, which is potentially a next level in CS education. As such, our "position" is that CS1 instructors would do well to be more flexible regarding content approaches, to enable increased focus on known-successful pedagogical and support approaches, and even adoption of "common courses" involving increased cooperation across universities.

## ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos. 2111323 and 2313793.

## REFERENCES

[1] Robins A. Learning edge momentum: A new account of outcomes in CS1. Computer Science Education. 2010 Mar 1;20(1):37-71.
[2] Watson C, Li FW. Failure rates in introductory programming revisited. Proceedings of the 2014 conference on Innovation & technology in computer science education 2014 Jun 21 (pp. 39-44).
[3] Hogan KE, Pressley ME. Scaffolding student learning: Instructional approaches and issues. Brookline Books; 1997.
[4] Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C. and Balik, S., 2003. Improving the CS1 experience with pair programming. ACM Sigcse Bulletin, 35(1), pp.359-362.
[5] Porter L., C. Bailey-Lee, B. Simon. Halving Fail Rates using Peer Instruction: A Study of Four Computer Science Courses, SIGCSE 2013.
[6] Glazier R.A.. Building Rapport to Improve Retention and Success in Online Classes. JPSC 2016.
[7] Porter, L., Bouvier, D., Cutts, Q., Grissom, S., Lee, C., McCartney, R., Zingaro, D. and Simon, B., 2016, February. A multi-institutional study of peer instruction in introductory computing. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education (pp. 358-363).
[8] Allen, J.M., Vahid, F., Downey, K. and Edgcomb, A.D., 2018, June. Weekly programs in a CS1 class: Experiences with auto-graded many-small programs (MSP). In 2018 ASEE Annual Conference & Exposition.
[9] zyBooks annual instructor survey (internal data), 2022.
[10] Siegfried, R.M., Chays, D. and Herbert, K., 2008, December. Will there ever be consensus on cs1?. In FECS (pp. 18-23).
[11] Lister, R., Berglund, A., Clear, T., Bergin, J., Garvin-Doxas, K., Hanks, B., Hitchner, L., Luxton-Reilly, A., Sanders, K., Schulte, C. and Whalley, J.L., 2006. Research perspectives on the objects-early debate. ACM SIGCSE Bulletin, 38(4), pp.146-165.
[12] Vilner, T., Zur, E. and Gal-Ezer, J., 2007. Fundamental concepts of CS1: procedural vs. object oriented paradigm-a case study. ACM SIGCSE Bulletin, 39(3), pp.171-175.
[13] Ehlert, A. and Schulte, C., 2009, August. Empirical comparison of objects-first and objects-later. In Proceedings of the fifth international workshop on Computing education research workshop (pp. 15-26).
[14] Turbak, F., Royden, C., Stephan, J. and Herbst, J., 1999. Teaching recursion before loops in CS1. Journal of Computing in Small Colleges, 14(4), pp.86-101.
[15] Mirolo, C., 2012, September. Is iteration really easier to learn than recursion for CS1 students?. In Proceedings of the ninth annual international conference on International computing education research (pp. 99-104).
[16] Winters, T. ITiCSE 2022 Keynote - The Gap Between Industry and Education, ITiCSE 2022, https://www.youtube.com/watch?v=Kxd3TWiJm20
[17] Henz, M., 2023. 'Early X or Late X'Questions for Discussing Curricular Practices in CS1 and CS2. In SIGCSE (2) (p. 1269).
[18] Reddit discussion, 2021, https://www.reddit.com/r/cpp/comments/qm6i25/my_professor_is_telling_us_to_never_use_vectors/
[19] Vahid, F., Downey, K., Areizaga, L. and Pang, A., 2023, March. Experiences Teaching Coral Before C++ in CS1. In Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (pp. 340-345).
[20] Blanchard, J., 2017, August. Hybrid environments: A bridge from blocks to text. In Proceedings of the 2017 ACM Conference on International Computing Education Research (pp. 295-296).
[21] Kölling, M., Quig, B., Patterson, A. and Rosenberg, J., 2003. The BlueJ system and its pedagogy. Computer Science Education, 13(4), pp.249-268.
[22] Allen, J.M., Vahid, F., Edgcomb, A., Downey, K. and Miller, K., 2019, February. An analysis of using many small programs in cs1. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education (pp. 585-591).
[23] R. M. Siegfried, K. G. Herbert-Berger, K. Leune and J. P. Siegfried, "Trends Of Commonly Used Programming Languages in CS1 And CS2 Learning," 2021 16th International Conference on Computer Science & Education (ICCSE), 2021, pp. 407-412, doi: 10.1109/ICCSE51940.2021.9569444.
[24] Gordon, C., Lysecky, R. and Vahid, F., 2022, August. Programming learners struggle as much in Python as in C++ or Java. In 2022 ASEE Annual Conference & Exposition.
[25] Hunt, J.M., 2015. Python in cs1-not. Journal of Computing Sciences in Colleges, 31(2), pp.172-179.
[26] Rosenshine, B., 2012. Principles of instruction: Research-based strategies that all teachers should know. American educator, 36(1), p.12.
[27] Freeman, S., Eddy, S.L., McDonough, M., Smith, M.K., Okoroafor, N., Jordt, H. and Wenderoth, M.P., 2014. Active learning increases student performance in science, engineering, and mathematics. Proceedings of the national academy of sciences, 111(23), pp.8410-8415.[28] https://runestone.academy/ns/books/published/csawesome/index.html
[29] Gordon, C.L., Lysecky, R. and Vahid, F., 2021, July. The rise of program auto-grading in introductory cs courses: A case study of zylabs. In 2021 ASEE Virtual Annual Conference Content Access.
[30] Talbot RM, Hartley LM, Marzetta K, Wee BS. Transforming undergraduate science education with learning assistants: Student satisfaction in large-enrollment courses. Journal of College Science Teaching. 2015 May 1;44(5):24-30.
[31] Vahid, F., and Pang, A., 2024. Experiences Teaching a CS1 Common Course across 7 Institutions. In Proceedings of the 55th ACM Technical Symposium on Computer Science Education.