2023 Annual Conference & Exposition
Baltimore Convention Center, MD | June 25 - 28, 2023
The Harbor of Engineering Education for 130 Years
Paper ID #37937

Work-in-progress: Exploring the computer science curriculum from undergraduate students' perspectives

Dr. Hye Rin Lee, University of Delaware

Hye Rin Lee is a NSF postdoctoral fellow at the University of Delaware. She received her Ph.D. at the University of California, Irvine with a concentration in Human Development in Context. Her research interests include motivation, psychological interventions, role models, academic engagement, and higher education.

Sotheara Veng, University of Delaware
Yiqin Cao, University of Delaware

M.Ed in Educational Technology MS. in Statistics

Juliana Baer, University of Delaware
Teomara Rutherford, University of Delaware
Austin Cory Bart

Work-in-progress: Exploring the computer science curriculum from undergraduate students' perspectives

***Abstract*—With large attrition rates among computer science (CS) majors, it is clear that CS undergraduates face challenges completing their degrees. Although much research has tested various teaching strategies and how course outcomes are associated with drop-out rate, little attention has been paid to using a bottom-up, student-centered, qualitative approach with a large sample to understand how to improve required CS courses and curricula. In the present study, we investigated CS college students' self-reported perceptions of curriculum design and instruction. We invited feedback from undergraduate students who enrolled in CS courses from various stages of the program (*N* = 445) at a large public Mid-Atlantic university. Specifically, we evaluated what students in CS would change to their required CS courses and/or course sequence through open-ended responses. Results of thematic coding of these responses revealed that students wanted clear connections between courses, course content and program design that were in line with practical skills used in the CS industry, and more effective academic advising and assistance from instructors. Implications and areas of future research will be discussed with respect to beneficial reforms to enhance student learning experiences in CS programs.**

***Keywords*—computer science, course sequence, curriculum design, higher education**

I. INTRODUCTION

With rapid technological advancements, computer scientists are needed more than ever to support our nation's economy and global competitiveness. However, approximately 59% of college students in computer science (CS) programs drop out [1]. Many efforts have been made to reduce this rate [e.g., 2, 3]. One area of research that has been examined to reduce this high attrition rate is CS course and curricula design [4, 5, 6]. Investigating the CS undergraduate program is important for finding strategic ways to improve student learning and motivation to continue in the CS pathway [5, 6]. Prior literature on students' pathways through CS has focused on both broad [e.g., frameworks for curricular design and curriculum philosophies; 7, 8, 9, 10, 11] and specific aspects [e.g., teaching and learning approaches; 3, 12] of the CS courses and curricula. These studies lack important information on why and how certain curricular elements are more or less successful. Undergraduate students themselves have agency regarding their decisions to persist in the CS major [13, 14, 15], yet few studies to date have used a bottom-up, student-centered, qualitative approach to understand how to improve CS courses and curricula. A qualitative approach can help researchers gain greater insights into students' experiences in the program without limiting their responses to predetermined lists as in typical top-down survey research [16, 17]. Furthermore, as students' progress along the CS undergraduate pathway, they may gain greater insights into their program and into what actions could be undertaken to improve their success. However, much of the prior work focuses on students who are in the beginning stages of their CS pathway, such as introductory CS courses [e.g., 2, 3, 4]. In the current study, we leveraged a student-centered, bottom-up, qualitative approach to examine CS undergraduate students' perspectives toward course content and curriculum sequence. In particular, we asked students from diverse points in the CS pathway (beginning, mid, and end-of-degree) about their suggestions for required CS courses and course sequences. This method allows us to detect deeper interpretations and explanations into students' CS experiences in their own voices.

II. LITERATURE REVIEW

A. Positionality

We approach our work as researchers grounded in theories of STEM education, specifically motivation and self-regulated learning [18, 19, 20, 21, 22]. Five out of the six authors did not graduate with a bachelor's degree in CS. As such, our lived educational experiences may limit the way we understand students' relevance and value of CS courses and/or sequences. As a team, we approached our work using a pragmatic lens in order to positively impact changes to CS courses and/or sequences. Moreover, due to our methodological approach, we do not frame our work under a specific theory; we focus on using students' own words (i.e., bottom-up approach) to guide implications for improving CS courses and/or sequences. Moreover, we believe that the environment plays an important role in understanding students' perspectives on CS courses and/or sequences for promoting motivation and performance in the CS pathway. Learning is a social construct that is influenced by the people and circumstances around the person [23]. If the needs of the learners are not taken into account when designing the classroom, it can negatively impact their ability to learn [24, 25]. Therefore, in the present study, we believe that investigating student-centered responses about the CS pathway (e.g., class environment, course sequence) is vital to improve student learning and persistence in CS.

B. CS Curriculum Design

There have been great strides in determining potential areas of improvement among CS courses and programs [e.g., 26, 27, 28]. One area of literature focuses on college students not having the necessary skills and knowledge to work in the industry [29, 30, 31, 32]. In regard to their soft skills, graduates tend to struggle with their verbal [31, 33] and writing [28, 30, 34] skills, in particular, clearly articulating their problems when they need help [29]. In regard to technical skills, graduates often lack the ability to use a number of industry software tools, such as configuration management and database tools [29, 32, 35]. Another area of the literature focuses on college students having difficulties in the CS program due to their required mathematics courses [36, 37, 38]. For example, students found an Automata Theory course challenging because it required a strong mathematics foundation on logic and problem-solving skills [38]. A strong background in mathematics, however, is thought to benefit learning in CS, especially in relation to algorithm design, computations, and data skills [39]. Other areas of the literature have explored issues about the CS curricula related to collaborative work [40], gender equality [41], and knowledge assessment [42]. Although this research has illuminated a constellation of issues around CS courses and curriculum, more work is needed to understand what challenges are the most pressing and salient to students. Identifying specific challenges noted by students can contextualize existing research and present new ideas for instructors and administrators, allowing greater prioritization and improvements.

Although prior researchers have aimed to identify and address specific ways to improve CS courses and curriculum [e.g., 2, 43, 44], there has been less attention on identifying more holistic suggestions for improvement across years in the program using students' own perspectives. Most prior work used quantitative rather than qualitative methods to understand the deficiencies and effectiveness of CS courses and curricula [2, 26, 40]. Quantitative methods can be helpful in gathering large datasets relatively quickly, but also can reduce the details of students' perspectives [16]. More work is needed using qualitative methods to detect CS students' deeper interpretations and explanations.

C. Current Study

In the current study, we use data gathered from college students enrolled in CS courses in the fall 2022 academic term at a large public Mid-Atlantic university. As part of their courses, students completed a survey related to their CS and course-specific motivation and learning. For the purposes of the present study, we used students' responses to two open-ended questions on the survey about their viewpoints toward CS courses and curricula. The study was approved by the university's Institutional Review Board. The data were used to answer the following research questions (RQs):

1. Do students want to change something about the required CS courses and sequence?
2. What changes do students suggest to the required CS courses and sequence?

3. What do students want to change the most/least about the required CS courses and sequence?

III. METHODOLOGY

A. Participants

Six hundred twenty-four participants in CS courses from various stages in the CS program at a large Mid-Atlantic university were invited to complete four surveys, approximately every three weeks of the fall 2022 academic semester, about their learning experiences and motivational beliefs. Participants received half a point of extra credit from their course instructor per survey. The invitation noted the terms of the extra credit and provided a link to the survey via Qualtrics. Students who did not wish to complete the survey were provided the option of an alternative assignment for extra credit. For the purposes of this study, we focused on the first survey where 445 students opted to participate. The analysis sample was limited to 305 college students who answered the optional open-ended responses.

B. Measures

Students answered two open-ended questions about their required CS courses and curricula during the beginning of their fall 2022 academic term on the first survey. Questions asked, "What changes would you suggest to the required CS courses and/or course sequences?" and "Is there anything else you would like to tell us?"

C. Analysis

To answer the first RQ, student responses from the question about changes they would suggest to the required CS courses and/or course sequences were categorized into (1) yes, I suggest change; (2) no, I do not suggest change; and (3) I do not know.

To answer the second RQ, we first developed a diagram of the CS course sequence to familiarize ourselves with the current structure. Data analysis of student responses involved using an inductive, thematic approach in multiple stages [45, 46]. Three researchers (i.e., second, third, and fourth authors) independently identified patterns in the data using Google Sheets before reviewing with the first author for consensus. Each coder would go through multiple rounds of coding to create sub-sub-codes, sub-codes, codes, sub-categories, and categories. The initial round involved using in-vivo (i.e., verbatim phrases and/or words in responses) and descriptive (i.e., summarizing phrases and/or words in responses) techniques [46]. Subsequent rounds after the first round of coding involved grouping the current codes into larger codes (e.g., going from sub-codes to codes, codes to sub-categories). Coders also individually created a codebook and diagrams on the online collaborative tool Jamboard to clearly define and distinguish between codes. Afterwards, in the next stage of data analysis, coders met to reach consensus for discrepancies in order to better understand the meaning of the data [47]. In the meetings, a group coding framework was established, which included sub-sub-codes, sub-codes, codes, sub-categories, and categories. Each discrepancy was redefined and clearly distinguished between similar codes through discussions between researchers. Unresolved discrepancies between the three coders and all codes were reviewed with the first author. The transformation of codes and rationale behind each code change were documented in analytic memos.

To answer the third RQ, frequencies of codes, sub-categories, and categories were calculated.

IV. RESULTS

A. RQ1

Findings showed that 58% of responding undergraduate students (n = 176) suggested changes to the required CS courses and/or course sequences. On the other hand, 14% of responding undergraduate students (n = 44) were satisfied with the required CS courses and/or course sequences and did not suggest any changes. Finally, 28% of responding undergraduates (n = 85) expressed uncertainty over what changes to suggest to the required CS courses and/or course sequences.

B. RQ2

Through the analysis of college students' suggestions for changes to CS courses and/or course sequences, three categories of codes were identified: program-level, course-level, and instruction-level changes. Within each category, there were multiple levels of coding that captured the specific changes proposed by the students (see Appendix 1).

First, undergraduate students suggested program-level changes to the required CS courses and/or course sequences (see Appendix 1). The changes that students suggested were related to (a) "course requirements," (b) "program content," (c) "course sequence," and (d) "language sequence." Regarding course requirements, students wanted to (a) remove a number of courses from the program requirements (e.g., automata theory, assembly language); (b) add courses as prerequisite to other courses to prepare them better for those courses; and (c) make a number of courses required for the program. Moreover, students wanted the content of the program to be more relevant to the skills and knowledge required in the industry. Concerning the course sequence, students believed that some courses should have been introduced earlier within their own divisions, such as data structure and introduction to algorithms, as well as across lower and upper divisions, including logic for CS and introduction to algorithms. Furthermore, students suggested changes to the sequence of the languages taught in the program, including the positions of Python, Dr. Racket, and Java in the program.

Second, undergraduate students suggested course-level changes to the required CS courses and/or course sequences (see Appendix 1). In particular, they wanted a change in (a) content within a course; or (b) performance assessment within a course. In regard to course content, students wanted more (a) "connections" between courses; (b) "improvements" in having a more balanced content between computer and information sciences and computer engineering, specific courses (e.g., introductory to computer science, data structures, front-end programming), and courses that connect to industry; (c) "including" topics into the CS curriculum, such as artificial intelligence/robotics, development (i.e., application development, web development), data structure, functional programming, mathematic courses for CS, MATLAB, website development, other teaching electives, and more courses in general; and (d) "number of languages taught" (i.e., too many CS software languages taught or want more variety in CS software languages). In regard to course performance, students wanted to change how their course performance is "assessed." For example, they did not like group work and wanted fewer or no exams.

Third, undergraduate students suggested instruction-level changes to the required CS courses and/or course sequences (see Appendix 1). Students wanted some changes to (a) how "knowledge" is imparted; and (b) "academic support" during instruction. In terms of knowledge building, they suggested (a) better estimation of students' knowledge; (b) providing more overview information of courses to help students understand courses better; (c) reducing the amount of workload; and (d) slowing down the pace of teaching. With respect to academic support, students requested (a) more supportive teaching approaches; and (b) better assistance through providing more practice in class and from supporting agents, including professors, advisors, and teaching assistants.

Fourth, undergraduate students suggested environmental-level changes to the required CS courses and/or course sequences (see Appendix 1). Particularly, they would like the program (a) to be more friendly towards students of all levels and (b) more "inclusive" of students, regardless of their gender and ethnicities. Some students expressed that they felt isolated, intimidated, or stressed. Regarding the exclusivity of the program, a number of students who identified as women noted that they felt discriminated against based on their gender and race.

C. RQ3

Among students' suggestions for the required CS courses and course sequences, program-level changes (n = 115) were the most frequently mentioned. Among program-level changes, those assigned the "remove" code were the most mentioned. Most students wanted to specifically remove two courses: Automata Theory (n = 19) and Assembly Language (n = 13). Although other courses were mentioned, they were mentioned less frequently: Database Systems (n = 1), Introduction to Human-Computer Interaction (n = 1), Network (n = 1), and mathematics courses like Linear Algebra (n = 1), and Statistics (n = 1). The least mentioned changes were related to the sequence of languages (n = 5). Most of the responses related to avoiding particular languages as their first language to learn, including Dr. Racket (n = 2) and Python (n = 1).

From responses related to the instruction level (n = 64), wrong assumptions of students' knowledge were the most frequently mentioned suggestion (n = 18): instructors more often overestimated (n = 13) than underestimated their knowledge (n = 5). Six responses from students mentioned the problem of having an overwhelming amount of workload.

Regarding course-level changes (n = 52), most students would like to see a wider variety of content relevant to the job market (n = 20), such as web development, application development, cloud computing, MATLAB, and functional programming. A few students also requested a greater variety of languages (n = 3).

For environment-level changes, most responses were related to their negative feeling towards the program (n = 6): the majority felt stressed out and intimidated in the program (n = 5). Students also expressed their feeling of being excluded (n = 4). Most of the respondents who noted feeling excluded believed that they felt excluded in the program on the basis of their gender (n = 3).

V. DISCUSSION

In this study we explored college students' perspectives from various years in the CS program on required CS courses and curricula using a bottom-up, student-centered, qualitative approach. Our pragmatic approach meant we adopted a method that could help instructors at this large Mid-Atlantic University improve their CS courses and/or curricula. First, results showed that despite some students expressing either satisfaction or indifference with the CS courses and/or curricula, a majority of students suggested modifications to the overall program structure, course topics, and instructional approaches. For example, students thought that courses in the CS program did not connect well with each other. Aligned with Bruner's [48] Spiral Curriculum Framework, learning often starts with introducing a topic, mastering that topic, revisiting that topic in a higher-level course, and finally making connections to other topics in the higher-level courses. As students responded that the latter part of the learning cycle (i.e., creating connections between courses) is missing in the CS pathway, one approach instructors can use is to not only revisit concepts from prior courses, but also ask students to reflect on how this revisited concept relates to the new course topic. This learning approach has been found to be successful in other domains, such as biological, chemical, and engineering programs [49, 50, 51].

Additionally, findings showed that students wanted the courses to be more relevant to the industry's demands. Consistent with prior work, there might be a gap between the CS courses and curricula and the knowledge and skills needed in the field, such as communication skills [31, 33] and the ability to use software tools [29, 35, 32]. Therefore, there should be more focus on providing those skills and explicitly stating how each learning activity is useful for their future career goals. Particularly, students were not able to see the usefulness of their theoretical courses (e.g., Automata Theory and Assembly Programming Language) to the job market. Because acquiring both theoretical and practical knowledge is important for students' learning in higher education [52, 53], instructors can use real world problems to convey their usefulness to the industry or may want to revisit the role of these courses in the sequence.

Not only did students discuss how theoretical courses like Automata Theory and Assembly Programming were irrelevant to their future career goals, but they also found the course topics difficult. In order to enhance students' learning, instructors can require prerequisite courses and explore ways to teach courses more effectively. For example, in Automata Theory, prior work has found that visualization programs, such as Java Formal Languages and Automata Package (JFLAP) helped students understand the material better and make the material more enjoyable [54, 55, 56, 57]. Other teaching strategies that can be effective for students' learning include: discussing CS concepts using historical context, connecting course topics with concrete programming examples mastered prior to the current course, and solving practical practice problems pertaining to the real world [58]. In Assembly Programming, previous literature has found that hands-on interactions through a compact device called Raspberry Pi [59] and video games [43] positively impacted students' learning experiences. Instructors in these CS courses and curricula might try using various techniques like those described to enhance students' learning.

Findings also suggested that there were students who felt intimidated and stressed in the program. In a CS class with mixed ability, low achieving students may form self-doubt, whereas high achievers may feel frustrated when working with students with lower abilities. Tafliovich et al. [60] found that many students felt intimidated in their CS courses because they knew that there were other students who had more achievements, such as published websites and applications. Moreover, students with more prior CS related experience felt that students with fewer skills were not able to sufficiently contribute to the group work [60]. Gender discrimination also contributed to the negative feeling experienced by students. This aligns with the finding by Bunderson and Christensen [61] where 20% of women students believed that they were treated differently in CS courses either by their classmates or teaching assistants. As a result, women students may feel less comfortable in CS programs, leading to higher attrition for those that identify as in this group [62]. It is crucial to prioritize effort to ensure that all students, regardless of their skill levels and gender, feel comfortable and supported in their learning.

VI. LIMITATIONS

Some students self-identified their race or gender in their answer; connecting all responses to demographics may have allowed us more insight into how students from underrepresented groups responded. This study is limited to a sample from one large Mid-Atlantic university; it may be useful to include a greater number of programs from universities of different sizes for better generalizability and comparative studies. Finally, as this work provides only a glimpse into different issues within the CS pathway, it may be helpful to expand data collection both longitudinally and with different types of data.

VII. CONCLUSIONS AND FUTURE WORK

This work provides insights into how programs and instructors can potentially improve CS courses and sequences, thereby lowering the attrition rate in such programs. In the present study, our findings allowed us to see students' points of view about their CS program, including program structure, course topics, teaching methods, and environment. Students want a more interconnected structure of the program, more relevance to the industry's demands in the curriculum, and more effective teaching. Implications for instructors and administrators are discussed throughout the paper. Future research can expand the reach of this work to other institutions; can triangulate with other data sources, such as observations; and can test the feasibility and impact of following some of the students' suggestions.

REFERENCES

[1] X. Chen, “STEM attrition: College students’ paths into and out of STEM fields.,” National Center for Education Statistics, Washington DC, 2013

[2] T. Howles, “A study of attrition and the use of student learning communities in the computer science introductory programming sequence,” *Computer Science Education*, vol. 19, no. 1, pp. 1–13, Mar. 2009, doi: 10.1080/08993400902809312.

[3] L. E. Margulieux, B. B. Morrison, and A. Decker, “Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples,” *International Journal of STEM Education*, vol. 7, no. 1, pp. 1–16, 2020.

[4] L. J. Barker, C. McDowell, and K. Kalahar, “Exploring factors that influence computer science introductory course students to persist in the major,” *ACM SIGCSE Bulletin*, vol. 41, no. 1, pp. 153–157, Mar. 2009, doi: 10.1145/1539024.1508923.

[5] T. Beaubouef and J. Mason, “Why the high attrition rate for computer science students,” *ACM SIGCSE Bulletin*, vol. 37, no. 2, pp. 103–106, Jun. 2005, doi: 10.1145/1083431.1083474.

[6] K. J. Bunker, L. E. Brown, L. J. Bohmann, G. L. Hein, N. Onder, and R. R. Rebb, “Perceptions and influencers affecting engineering and computer science student persistence,” in *2013 IEEE Frontiers in Education Conference (FIE)*, 2013, pp. 1138–1144.

[7] B. Burd *et al.*, “The internet of things in undergraduate computer and information science education: Exploring curricula and pedagogy,” Jul. 2018. doi: https://doi.org/10.1145/3293881.3295784.

[8] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi, “The structure and interpretation of the computer science curriculum,” *Journal of Functional Programming*, vol. 14, no. 4, pp. 365–378, Jul. 2004, doi: https://doi.org/10.1017/S0956796804005076.

[9] J. C. Knight, J. C. Prey, and Wm. A. Wulf, “Undergraduate computer science education: A new curriculum philosophy & overview,” *ACM SIGCSE Bulletin*, vol. 26, no. 1, pp. 155–159, Mar. 1994, doi: https://doi.org/10.1145/191033.191093.

[10] P. Machanick, “Principles versus artifacts in computer science curriculum design,” *Computers & Education*, vol. 41, no. 2, pp. 191–201, Sep. 2003, doi: https://doi.org/10.1016/s0360-1315(03)00045-9.

[11] B. C. Wilson and S. Shrock, “Contributing to success in an introductory computer science course: A study of twelve factors,” *ACM SIGCSE Bulletin*, vol. 33, no. 1, pp. 184–188, Mar. 2001, doi: https://doi.org/10.1145/366413.364581.

[12] A. Robins, J. Rountree, and N. Rountree, “Learning and teaching programming: A review and discussion,” *Computer Science Education*, vol. 13, no. 2, pp. 137–172, Jun. 2003, doi: https://doi.org/10.1076/csed.13.2.137.14200.

[13] S. Katz, D. Allbritton, J. Aronis, C. Wilson, and M. L. Soffa, “Gender, achievement, and persistence in an undergraduate computer science program,” *ACM SIGMIS Database: the DATABASE for Advances in Information Systems*, vol. 37, no. 4, pp. 42–57, Nov. 2006, doi: https://doi.org/10.1145/1185335.1185344.

[14] G.Y. Lin, “Self-efficacy beliefs and their sources in undergraduate computing disciplines,” *Journal of Educational Computing Research*, vol. 53, no. 4, pp. 540–561, Nov. 2015, doi: https://doi.org/10.1177/0735633115608440.

[15] M. S. Peteranetz, L.-K. Soh, D. F. Shell, and A. E. Flanigan, “Motivation and self-regulated learning in computer science: lessons learned from a multiyear program of classroom research,” *IEEE Transactions on Education*, vol. 64, no. 3, pp. 317–326, Aug. 2021, doi: https://doi.org/10.1109/te.2021.3049721.

[16] J. C. Libarkin and J. P. Kurdziel, “Research methodologies in science education: The qualitative quantitative debate,” *Journal of Geoscience Education*, vol. 50, no. 1, pp. 78–86, 2002.

[17] M. Paechter and B. Maier, “Online or face-to-face? Students’ experiences and preferences in e-learning,” *The Internet and Higher Education*, vol. 13, no. 4, pp. 292–297, Dec. 2010, doi: https://doi.org/10.1016/j.iheduc.2010.09.004.

[18] A. Bandura, “Social cognitive theory of self-regulation,” *Organizational Behavior and Human Decision Processes*, vol. 50, no. 2, pp. 248–287, Dec. 1991, doi: https://doi.org/10.1016/0749-5978(91)90022-l.

[19] J. S. Eccles *et al.*, “Expectancies, values, and academic behaviors,” in *Achievement and Achievement Motivation*, San Francisco, CA: W. H. Freeman, 1983, pp. 75–146.

[20] H. W. Marsh, “A multidimensional, hierarchical model of self-concept: Theoretical and empirical justification,” *Educational Psychology Review*, vol. 2, no. 2, pp. 77–172, Jun. 1990, doi: https://doi.org/10.1007/bf01322177.

[21] P. R. Pintrich and E. V. de Groot, “Motivational and self-regulated learning components of classroom academic performance.,” *Journal of Educational Psychology*, vol. 82, no. 1, pp. 33–40, 1990, doi: https://doi.org/10.1037/0022-0663.82.1.33

[22] B. J. Zimmerman, “Attaining self-regulation,” in *Handbook of Self-Regulation*, Academic Press, 2000, pp. 13–39. doi: https://doi.org/10.1016/b978-012109890-2/50031-7.

[23] L. Vygotsky, *Mind in Society: The Development of Higher Psychological Processes*. Cambridge, Mass.; London: Harvard University Press, 1978.

[24] J. S. Eccles and C. Midgley, “Stage-environment fit: Developmentally appropriate classrooms for young adolescents,” *Research on Motivation in Education*, vol. 3, no. 1, pp. 139–186, 1989.

[25] C. R. Ellerbrock and S. M. Kiefer, “The interplay between adolescent needs and secondary school structures: Fostering developmentally responsive middle and high school environments across the transition,” *The High School Journal*, vol. 96, no. 3, pp. 170–194, 2013, doi: 10.1353/hsj.2013.0007.

[26] R. A. Alturki, “Measuring and Improving Student Performance in an Introductory Programming Course,” *Informatics in Education*, vol. 15, no. 2, pp. 183–204, Nov. 2016, doi: https://doi.org/10.15388/infedu.2016.10.

[27] A. Pears *et al.*, “A survey of literature on the teaching of introductory programming,” *ACM SIGCSE Bulletin*, vol. 39, no. 4, pp. 204–223, Dec. 2007, doi: 10.1145/1345375.1345441.

[28] C. B. Simmons and L. L. Simmons, “Gaps in the computer science curriculum: an exploratory study of industry professionals,” *Journal of Computing Sciences in Colleges*, vol. 25, no. 5, pp. 60–65, 2010.

[29] A. Begel and B. Simon, “Struggles of new college graduates in their first software development job,” in *ACM SIGCSE Bulletin*, Feb. 2008, vol. 40, no. 1, pp. 226–230. doi: 10.1145/1352322.1352218.

[30] D. J. Byrne and J. L. Moore, “A comparison between the recommendations of computing curriculum 1991 and the views of software development managers in Ireland,” *Computers & Education*, vol. 28, no. 3, pp. 145–154, Apr. 1997, doi: 10.1016/s0360-1315(97)00006-7.

[31] D. Hagan, “Employer satisfaction with ICT graduates,” in *Proceedings of the Sixth Australasian Conference on Computing Education*, 2004, vol. 30, pp. 119–123.

[32] H. Tang, S. Lee, and S. Koh, “Educational gaps as perceived by IS educators: A survey of knowledge and skill requirements,” *Journal of Computer Information Systems*, vol. 41, no. 2, pp. 76–84, 2001.

[33] D. Tesch, G. F. Braun, and E. Crable, “An examination of employers’ perceptions and expectations of is entry-level personal and interpersonal skills,” *Information Systems Education Journal*, vol. 7, no. 1, 2006.

[34] T. C. Lethbridge, “A survey of the relevance of computer science and software engineering education,” in *Proceedings 11th Conference on Software Engineering Education*, 1998, pp. 56–66.

[35] J. C. Carver and N. A. Kraft, “Evaluating the testing ability of senior level computer science students,” in *2011 24th IEEECS Conference on Software Engineering Education and Training (CSEE&T)*, 2011, pp. 169–178.

[36] K. Goldman *et al.*, “Identifying important and difficult concepts in introductory computing courses using a delphi process,” Mar. 2008. doi: https://doi.org/10.1145/1352135.1352226.

[37] R. Logozar, M. Horvatic, I. Sumiga, and M. Mikac, “Challenges in teaching assembly language programming–Desired prerequisites vs. students’ initial knowledge,” in *2022 IEEE Global Engineering Education Conference (EDUCON)*, 2022, pp. 1689–1698.

[38] N. Pillay, “Learning difficulties experienced by students in a course on formal languages and automata theory,” *ACM SIGCSE Bulletin*, vol. 41, no. 4, p. 48, Jan. 2010, doi: 10.1145/1709424.1709444.

[39] T. Beaubouef, “Why computer science students need math,” *ACM SIGCSE Bulletin*, vol. 34, no. 4, p. 57, Dec. 2002, doi: 10.1145/820127.820166.

[40] T. Prickett, J. Walters, L. Yang, M. Harvey, and T. Crick, “Resilience and effective learning in first year undergraduate computer science,” in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, 2020, pp. 19–25.

[41] D. Stoilescu and D. McDougall, “Gender digital divide and challenges in undergraduate computer science programs,” *Canadian Journal of Education*, vol. 34, no. 1, pp. 308–333, 2011.

[42] Y. Wang, H. Li, Y. Feng, Y. Jiang, and Y. Liu, “Assessment of programming language learning based on peer code review model: Implementation and experience report,” *Computers & Education*, vol. 59, no. 2, pp. 412–422, 2012.

[43] J. Kawash and R. Collier, “Using video game development to engage undergraduate students of assembly language programming,” in *Proceedings of the 14th Annual ACM SIGITE Conference on Information Technology Education*, 2013, pp. 71–76.

[44] J. Kay *et al.*, “Problem-based learning for foundation computer science courses,” *Computer Science Education*, vol. 10, no. 2, pp. 109–128, 2000.

[45] V. Braun and V. Clarke, “Thematic analysis,” *APA Handbook of Research Methods in psychology, Vol 2: Research Designs: Quantitative, Qualitative, Neuropsychological, and Biological.*, vol. 2, no. 2, pp. 57–71, 2012, doi: 10.1037/13620-004.

[46] J. Saldaña, *Goodall's verbal exchange coding: An overview and example*, vol. 22, no. 1. SAGE Publications Sage CA: Los Angeles, CA, 2016, pp. 36–39.

[47] C. E. Hill, S. Knox, B. J. Thompson, E. N. Williams, S. A. Hess, and N. Ladany, "Consensual qualitative research: An update.," *Journal of Counseling Psychology*, vol. 52, no. 2, p. 196, 2005.

[48] J. S. Bruner, *The Process of Education*. Cambridge, Mass: Harvard University Press, 1960.

[49] D. DiBiasio, L. Comparini, A. G. Dixon, and W. M. Clark, "A project-based spiral curriculum for introductory courses in ChE: III. Evaluation," *Chemical Engineering Education*, vol. 35, no. 2, pp. 140–146, 2001.

[50] Lohani, Vinod K, M. L. Wolfe, T. Wildman, K. Mallikarjunan, and J. Connor, "Reformulating general engineering and biological systems engineering programs at Virginia Tech," *Advances in Engineering Education*, vol. 2, no. 4, p. n4, 2011.

[51] S. Vemuru, S. Khorbotly, and F. Hassan, "A spiral learning approach to hardware description languages," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2013, pp. 2759–2762.

[52] N. Entwistle, "Concepts and conceptual frameworks underpinning the ETL project," University of Edinburgh, Edinburgh, 2003.

[53] V. MCcune and D. Hounsell, "The development of students? ways of thinking and practising in three final-year biology courses," *Higher Education*, vol. 49, no. 3, pp. 255–289, Apr. 2005, doi: 10.1007/s10734-004-6666-0

[54] C. W. Brown and E. A. Hardisty, "RegeXeX: An interactive system providing regular expression exercises," *ACM SIGCSE Bulletin*, vol. 39, no. 1, pp. 445–449, Mar. 2007, doi: 10.1145/1227504.1227462.

[55] S. H. Rodger, E. Wiebe, K. M. Lee, C. Morgan, K. Omar, and J. Su, "Increasing engagement in automata theory with JFLAP," *ACM SIGCSE Bulletin*, vol. 41, no. 1, pp. 403–407, Mar. 2009, doi: 10.1145/1539024.1509011.

[56] A. Stoughton, "Experimenting with formal languages," in *Thirty Sixth SIGCSE Technical Symposium on Computer Science Education*, 2005, p. 566.

[57] Vieira, M. Augusto, and N. J. Vieira, "Language emulator, a helpful toolkit in the learning process of computer theory," *ACM Sigcse Bulletin*, vol. 36, no. 1, pp. 135–139, 2004.

[58] C. I. Chesñevar, M. P. González, and A. G. Maguitman, “Didactic strategies for promoting significant learning in formal languages and automata theory,” *ACM SIGCSE Bulletin*, vol. 36, no. 3, pp. 7–11, Jun. 2004, doi: 10.1145/1026487.1008002.

[59] J. Kawash, A. Kuipers, L. Manzara, and R. Collier, “Undergraduate assembly language instruction sweetened with the raspberry pi,” in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 2016, pp. 498–503.a

[60] A. Tafliovich, J. Campbell, and A. Petersen, “A student perspective on prior experience in CS1,” in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 2013, pp. 239–244

[61] E. D. Bunderson and M. E. Christensen, “An analysis of retention problems for female students in university computer science programs,” *Journal of Research on Computing in Education*, vol. 28, no. 1, pp. 1–18, 1995.

[62] C. Mooney, B. A. Becker, L. Salmon, and E. Mangina, “Computer science identity and sense of belonging: a case study in Ireland,” in *Proceedings of the 1st International Workshop on Gender Equality in Software Engineering*, 2018, pp. 1–4.

APPENDIX I. QUALITATIVE FINDINGS: SUGGESTED CHANGES IN THE CS COURSES AND/OR COURSE SEQUENCES

Category	**Sub-category**	**Code**	**Example**	**Sub-code**
Program: suggesting a program-level change (*n* = 115)	Suggestions on course requirements (*n* = 69)	Suggestions on course requirements (*n* = 69)	"I think calculus shouldn't be required to take CIS courses."	remove CS course: suggesting that a CS course should be removed a CS course (*n* = 41) **Sub-sub-codes**: ● remove, CS[Automata Theory] (*n* = 19) ● remove, CS[Machine Organization and Assembly Language] (*n* = 13) ● remove, CS[Introduction to Systems Programming] (*n* = 2) ● remove, same [Data Structures and Introduction to Algorithms] (*n* = 1) ● remove, CS[Computers, Ethics, and Society] (n = 1) ● remove, CS[Logic and Programming] (*n* = 1) ● remove, CS [Data Structures] for stats for stats (n = 1) ● remove, CS[Data Structures] for stats (n = 1) ● remove, CS[Introduction to Systems Programming] for stats (n = 1) ● remove, CS[Introduction to CS II] (n = 1)
				remove, MATH course: suggesting that a mathematics course should be removed (*n* = 6) **Sub-sub-codes**: ● remove, MATH[Statistical Methods] (*n* = 2) ● remove, MATH[Analytic Geometry & Calculus B/C] (*n* = 1) ● remove, MATH[Discrete Mathematics] (*n* = 1) ● remove, MATH, general (*n* = 1) ● remove, MATH, calculus (*n* = 1)

				remove, prereq restrictions: suggesting that a prerequisite course should be removed (*n* = 1)
				remove, lab science: suggesting that a lab science course should be removed from requirement (*n* = 4)
		prereq: suggesting prerequisite for a course (*n* = 9)	"Not have CS[Introduction to CS II] has a prereq[uisite] for CS [Data Structure] can make the challenge of CS [Data Structure] even bigger for students who come in without OOP [Object-Oriented Programming] experience and requires a significant amount of time in CS [Data Structure] to review basic OOP ideas."	prereq, course: suggesting that a specific course need a prerequisite (*n* = 7) **Sub-sub-codes**: ● prereq, for CS[Automata Theory] (*n* = 1) ● prereq, for CS[Data Structures] (*n* = 1) ● prereq, for CS[Introduction to Systems Programming] (*n* = 1) ● prereq, CS[Introduction to CS II] for CS[Data Structures] (*n* = 1) ● prereq, CS[Introduction to CS II] for CS[Introduction to Systems Programming] (*n* = 1) ● prereq, CS[Introduction to CS II] & CS[Introduction to Systems Programming] for CS[Data Structures] (*n* = 1) ● prereq, between CS[Introduction to Systems Programming] and CS[Data Structures] (*n* = 1)
				prereq, general: suggesting that a course need a prerequisite without specifying a specific course name (*n* = 2)
		make req: suggesting that a course should be required for the program (*n* = 6)	"I don't understand how some courses are optional (i.e., linear algebra) but later on, you need linear algebra for several classes."	make req, CS course: suggesting that a CS course should be required for the program (*n* = 3) **Sub-sub-codes**: ● make req, CS[Introduction to Human-Computer Interaction] (*n* = 1)

				• make req, CS[Database Systems] (*n* = 1) • make req, CS[Introduction to CS2] for CE (n = 1)
				make req, MATH course: suggesting that a MATH course should be required for the program (*n* = 2) **Sub-sub-codes**: • make req, MATH[Statistical Methods] (*n* = 1) • make req, MATH[Linear Algebra] (*n* = 1)
				make req, CS content: suggesting that a CS content should be required for the program (*n* = 1) **Sub-sub-codes**: • make req, CS content network (*n* = 1)
		sequence change, suggesting a change in course sequence (*n* = 16)	"I think that data structures should be an earlier course, most of the software engineering internship interviews that I have are concepts from data structures…"	sequence change, upper division course: suggesting a change in the sequence within upper division courses **Sub-sub-codes**: • sequence change, CS [Senior Design] (*n* = 1) • sequence change, CS[Operating Systems] after CS[Parallel Programming] (*n* = 1) • sequence change, CS[Computers, Ethics, and Society] to first year (*n* = 1) • sequence change, CS[Logic and Programming] before CS[Automata Theory] (*n* = 1)
				sequence change, lower division course: suggesting a change in the sequence within (*n* = 5) **Sub-sub-codes**: • sequence change, CS[Data Structures] earlier (*n* = 2) • sequence change, CS[Data Structures] (*n* = 1) • sequence change, CS[Introductory CS for Engineers] with CS[Introduction to Systems Programming] and CS[Introduction to CS II] (*n* = 1)

				• sequence change, CS[Introductory CS for Engineers] should not be before CS[Introduction to Systems Programming] (*n* = 1)
				sequence change, across division: suggesting a change in the sequence within/between upper and lower division courses (n = 2) **Sub-sub-codes**: • sequence change, CS[Logic in CS] and MATH[Discrete Mathematics] concurrently (n = 1) • sequence change, CS[Data Structures] and CS[Introduction to Algorithms] earlier (n = 1)
				sequence change, CS content: suggesting a change in the sequence of a CS content (n = 4) **Sub-sub-codes**: • sequence change, object-oriented programming earlier in CS[Introduction to CS I] (n = 2) • sequence change, networks earlier (n = 1) • sequence change, command earlier (n = 1)
		sequence, lang: suggesting a change in the sequence of languages taught in the program (*n* = 5)	"Learning Java should actually go first over learning python. Java is a more readable language, which makes it easier to understand…"	sequence, lang, put Java before Python (*n* = 2)
				sequence, lang, do not start with Dr. Racket (*n* = 2)
				sequence, lang, do not start with python (*n* = 1)
	--	industry relevant design: suggesting that courses or content	"I think some courses teach material that may be slightly outdated or may be different from	industry relevant design, less theories: suggesting reducing the number of theoretical courses (*n* = 6)
				industry relevant design, general: suggesting that courses or content of the program should be more relevant to the

		of the program should be more relevant to the industry demands (*n* = 13)	what people currently work with (in the workforce)."	industry demands, in general without providing any details (*n* = 6)
				industry relevant design, updated materials: suggesting that program materials should be updated to be more relevant to current trends (*n* = 1)
	--	available: suggesting that the capacity of a certain required/prerequisite course should be considered when the program is planned (*n* = 10)	"Currently, [Data Structure] is a huge gatekeeper to the rest of the Comp. Sci. curriculum. Caused a lot of problems in scheduling."	available, course: suggesting that the capacity of a certain course should be increased by mentioning the course name (*n* = 2) **Sub-sub-codes**: ● available, course, CS[Introduction to Software Engineering] (*n* = 1) ● available, course, CS[Data Structures] (*n* = 1)
Instruction: suggesting instructional-level changes (*n* = 64)	Knowledge: suggesting a change in how instructors understand students' learning and knowledge (*n* = 36)	assume know: suggesting that instructors should estimate students' knowledge better (*n* = 18)	"I feel like sometimes professors forget some people have no experience with coding at all. So, when professors talk fast about some topics, I tend to get lost."	assume know, overestimate: suggesting that instructors should avoid overestimating students' knowledge (*n* = 13)
				assume know, underestimate: suggesting that instructors should avoid underestimating students' knowledge (*n* = 5)
		overview info: suggesting that more general information about the course should be provided to	"More direct telling you what you are supposed to be learning about at specific points in the course."	

		students, such as the content covered in the course, course syllabus, and other suggested materials/content (*n* = 7)		
		workload: suggesting more appropriate amount of workload including assignments, classwork, and projects (*n* = 6)	"...students are given massive projects and are basically told to just figure it out… most of which took ridiculous amounts of hours to finish…"	
		pace: suggesting that the pace of teaching should be slowed down (*n* = 5)	"They go at such a fast pace sometimes. Need to spend more time with fundamentals and basics."	pace, general: suggesting that the pace of teaching should be slowed down in general without specifying the course name (*n* = 4)
				pace, course: suggesting that the pace of teaching should be slowed down in general by specifying the course name (*n* = 1) **Sub-sub-code:** ● pace, course, CS[Data Structures] (*n* = 1)
	Academic Support (*n* = 28)	better teaching: suggesting that more effective	"...Past the Python course, you're launched into Java and C. Both very relevant and overall good to	better teaching, general: suggesting that more effective teaching approaches should be used in general without specifying the course name (*n* = 4)

		teaching approaches should be used (*n* = 14)	learn. Except you have to learn both at the same time with very little help and actual teaching…”	better teaching, engagement: suggesting that more effective engagement tools or teaching approaches should be used (*n* = 4)
				better teaching, course: suggesting that more effective teaching approaches should be used in a specific course (*n* = 4) **Sub-sub-codes**: ● better teaching, course, CS[Computers, Ethics, and Society] (*n* = 1) ● better teaching, course, CS[Introduction to Algorithms] (*n* = 1) ● better teaching, course, CS[Introduction to CS II] (*n* = 1) ● better teaching, course, CS[Principles of Computing], [CS with Web Applications], [Introductory CS for Engineers], [Introduction to CS I] [a different] campus (*n* = 1)
				better teaching, consistent quality: suggesting that there should be consistent quality across different instructors teaching the same course (*n* = 2)
		assist: suggesting that more assistance should be provided to students in their study (*n* = 14)	“I would definitely suggest more office hours and even labs for people to go to for help, especially during the first week of school so people don’t get behind. I went to my lab the first week and the TA did not show up.”	assist, practice: suggesting that more assistance should be provided to students through giving more practices (*n* = 6) **Sub-sub-codes**: ● assist, practice, general (*n* = 4) ● assist, practice, python after CS[General CS for Engineers] and CS[Introduction to CS] after and 108 (*n* = 1) ● assist, practice, coding (*n* = 1)

				assist, meeting, advisor: suggesting that more assistance should be provided to students through meeting with advisors (*n* = 5)
				assist, teaching assistant: suggesting that more assistance should be provided to students through meeting with teaching assistant (*n* = 1)
				assist, meeting, instructor: suggesting that more assistance should be provided to students through meeting with instructors at office hours (*n* = 1)
				assist, disability support service: suggesting that more assistance should be provided to students through providing better disability support service (*n* = 1)
Environment: suggesting environmental-level changes (*n* = 10)	--	feel, negative: feeling towards the program, such as feeling stressed out, intimidated, or isolated (*n* = 6)	"I am not a strong coder and I feel very intimidated by many of the students I've worked with..."	feel, negative, stress and intimidated (*n* = 5)
				feel, negative, isolated (*n* = 1)
		exclusion: feeling excluded due to their gender or ethnicity (*n* = 4)	"There is a heavy amount of judgment and discouraging behaviors amongst males against females in [CS], that makes the learning experience/environment uncomfortable."	exclusion, gender (*n* = 3)
				exclusion, race (*n* = 1)
Course: suggesting a course-level	Course Content: suggesting a	include: suggesting that a new topic should	"I think there should be a web development concentration offered to CIS students."	include, development: suggesting that a course related to development should be added (*n* = 7) **Sub-sub-codes**:

change (*n* = 52)	change in the content within courses (*n* = 46)	be added to the course (*n* = 20)		● include, web development (*n* = 5) ● include, app development (*n* = 1) ● include, development, general (*n* = 1)
				include, matlab: suggesting MATLAB should be added in different course aspects (*n* = 2) ● include, MATLAB for python (*n* = 1) ● include, MATLAB for engineering (*n* = 1)
				include, A.I, robotics (*n* = 2)
				include, better tech elective (*n* = 1)
				include, data structure (*n* = 1)
				include, functional programming (*n* = 2)
				include, more class variety (*n* = 2)
				include, math for CS (*n* = 1)
				include, terminology in CS[Introductory CS for Engineers] (*n* = 1)
		improve: suggesting an improvement to a course (*n* = 11)	"Getting feedback from the industry and basing courses on that. Not learning useless things which will never be used in the real-world. Also having interview prep sessions since that is a big	improve, real world: suggesting an improvement to a course by adding content that are more practical to the real world (*n* = 6) **Sub-sub-codes**: ● improve, real world, general (*n* = 3) ● improve, real world, algorithms (*n* = 2) ● improve, real world, live coding (*n* = 1)

			part of the trivia part for getting jobs.”	
				improve, course: suggesting an improvement to a course or a topic in a course (*n* = 5) **Sub-sub-codes**: ● improve, course, front end (*n* = 1) ● improve, course, CS[Data Structures] (*n* = 1) ● improve, course, CS[Introduction to CS II] (*n* = 1) ● improve, course, CS[Introductory CS for Engineers] (*n* = 1)
				improve, balance CPEG & CS: suggesting an improvement to course content by providing balanced content from CS and engineering (*n* = 1)
		lang: suggesting a change to the number of languages taught in the program by either reducing or adding it (*n* = 9)	“I'm not too sure if I just haven't gotten to that level yet, but I think that further learning of certain coding languages might help rather than learning more and more different languages.”	lang-too many: suggesting that the number of programming languages taught should be reduced (*n* = 6)
				lang-variety: suggesting that there should be a wider variety of languages taught (*n* = 3)
		connect: suggesting more connections of content between courses (*n* = 6)	“The courses rarely, if at all, build on top of existing topics. They are always teaching us something new which makes it difficult to be good at a particular thing.”	--

	--	assess: suggesting a change related to how their course performances are assessed ($n = 6$)	"I am also not a great test-taker, so I enjoy computer courses that have no exams and just do projects and assignments."	assess, no/less test, general ($n = 3$)
				assess, no group work ($n = 2$)
				assess, no/less test, no handwritten test ($n = 1$)