Escaping the CS Dungeon: Modern College Curricula within and Beyond Computing

Audrey Marie DeHoog, University of Florida

Audrey DeHoog has received a bachelor's and a master's degree in computer science from the University of Florida Herbert Wertheim College of Engineering in Gainesville, Florida. She is currently in the application process to start an engineering education PhD program in the Fall 2025 semester. She is interested in CS education, diversity and inclusion in engineering education, and higher education.

Dr. Jeremiah J Blanchard, University of Florida

Jeremiah Blanchard is an Assistant Engineer at the University of Florida in the Computer & Information Science & Engineering Department, where he teaches and conducts research in computer science education. Previously, he served as Program Director of Gam

Ms. Amy Wu, University of Florida

Amy Wu is a undergraduate researcher in Computer Science from the University of Florida. Her research focuses on understanding and building communication for learning technologies and experiences. She is always seeking for more opportunities in human-AI interaction and collaboration.

Dr. John R. Hott, University of Virginia

John R. Hott is an Assistant Professor of Computer Science at the University of Virginia. His research interests include scaling the classroom using LLMs and Artificial Intelligence, student use of collaboration policies, student engagement with course resources, academic integrity, and tools to support Computing Education and Computing Education Research.

Escaping the CS Dungeon: Modern College Curricula within and Beyond Computing

Abstract

The prevalence of technology across disciplines has created a need for non-computing majors to learn programming and computing practices. Despite this, there is a lack of research documenting current programming course offerings for non-computing majors. Therefore, it is unclear how common these courses are, what fields these courses support, what programming languages are used in the courses, and what departments and institutions offer these courses. To answer these questions, we explored how and where computing is integrated into STEM and non-STEM undergraduate programs within different disciplines based on 81 survey responses of instructors teaching introductory programming courses at the 50 largest public universities in the United States. Our results indicate that computing courses are commonly offered to students of STEM and non-STEM disciplines in a variety of different programming languages.

Introduction

There are many incentives for non-computing students to learn to program in addition to their regular curriculum. Key motivations include the growing demand for all undergraduates to be knowledgeable about basic principles of STEM concepts to be successful in the increasingly technological and global economy [1]. While there has been a rise in the number of non-computer science majors interested in learning programming and other computational skills in recent years [2], an emphasis on fostering non-STEM student interest in STEM concepts is still needed to meet the increasing demand of technical knowledge [3].

To better understand programming course offerings for computing and non-computing majors, we collected data on the curriculum of fundamental programming courses at the top 50 largest public universities of the National Center for Education Statistics (NCES). Online surveys were distributed to instructors through email and included questions on which programming tools, learning tasks, and collaborative approaches are used within their courses. We also asked in which department the course was taught. The courses were divided into three categories—computing, STEM, and non-STEM—to examine how prevalent programming courses are in non-computing departments as well as which disciplines consistently offer programming courses.

Background

Offering programming courses for non-STEM majors does not come without dilemmas, since the process of teaching STEM related subjects may differ between STEM and non-STEM majors [4]. Specifically, STEM and non-STEM students often exhibit differences in pre-college influences, academic performance, and institutional expectations [5]. Additionally, students generally associate courses including mathematical or technical content with a higher difficulty than courses with content related to the arts, humanities, and other non-STEM related subjects [6]. This association may make computer science courses and topics appear more daunting to non-STEM majors. It is vital to overcome this and foster non-STEM interest in programming, as a student's attitude towards computer science education is more important than a student's prior computer training when it comes to success in an introductory programming course [7]. For instance, computer science courses for non-STEM majors should expose students to the connections between computing coursework and their occupational choices [8]. Tailoring programming courses based on non-STEM disciplines allows students to bring their own knowledge and interests, and combine it with what they learn in their programming classes [9]. It is also difficult for non-STEM educators to teach STEM concepts to students as they may not be well versed in these subjects and therefore not prepared to teach them to students [10]. Instructors are critical for encouraging and developing interest in STEM among students, so it is important to provide them with the tools and resources necessary for their students' success with STEM concepts [8].

Collaborative learning can also be a useful tool for creating a more welcoming environment for non-STEM majors. Chowdhury et al. [9] found that non-STEM majors with little to no previous programming experience found semester long small groups to be useful for learning computational thinking. Their students were more comfortable asking questions about the content because they were able to relate more to a peer who is at the same learning level rather than a professor [9]. Additionally, students who program together create better programs and are less likely to fail a course [11]. Overall, collaborative learning is effective in reducing the anxiety level of beginners, strengthening student's understanding of fundamental computing concepts, and creating a positive atmosphere for learning [12].

Another critical aspect of making computing more accessible for non-STEM majors is selecting a programming language that is better for beginners with little to no experience. For instance, Python is growing in popularity for introductory programming courses for many reasons, such as its simpler syntax and the availability of graphics and other libraries that make learning Python more engaging [13]. Additionally, Python makes it easier to tailor the complexity of the programs used in the course [14]. However, further research is still needed on the effects of teaching introductory programming courses in Python [13].

Incorporating STEM subjects in non-STEM curricula would not only require a considerable amount of rethinking and redesigning course structures, but would also require a deeper understanding of how STEM should be taught to non-STEM students [10]. Therefore, it is reasonable to conclude that both further research and tools promoting a more welcoming environment for learning programming concepts are needed to make computing more accessible to non-STEM students. By collecting data on existing introductory programming courses, we aim to contribute to improving how fundamentals are taught to students with non-computing backgrounds, improving collaboration among non-computing students in programming, and advancing the field of research as a whole.

Methods

In this study, we collected data from introductory programming courses at the top 50 largest public universities available for review in the United States, according to the NCES. Community colleges—which focus on 2-year programs and transfers to 4-year institutions—were not included, as their curriculum does not represent that typical of a 4-year university. Additionally, small or private universities were not included because their course curricula is often not publicly available or standardized. Small institutions are also not a strong representation of our target population and could lead to outliers in the data.

For each of the 50 universities, we conducted a keyword search with the terms "Java," "C++," "C#," "C," "Python," "JavaScript," and "Programming" of all available course listings at the university within the previous year. We then checked the listings for beginner programming courses, which were identified by the course having no prerequisites. The current instructors of these courses (n=132) were contacted through email and asked to fill out an online survey regarding their course curriculum. The instructors self reported their institution name, course name, course department, and average number of students enrolled in the class at one time. The survey also included multiple choice, select all, and free response questions to gain further insight into the introductory programming courses currently offered. The data collected from these questions included:

- Programming language(s) used in the course.
- Environments and tools used for instruction; e.g., Integrated Development Environment (IDE).
- The strengths and weaknesses of these tools/environments.
- Instructor policies on collaboration within the course.

Once the surveys were completed, courses were grouped into disciplines based on the department offering the course. Most groups consist of courses from different universities with the same or similar department name; however, outliers were grouped into a discipline based on the degrees offered in the department at the university. We further categorized the courses as computing, STEM, or non-STEM based on the intended student audience. To determine if a course could be categorized as computing, the course must be in the curriculum of a degree program at the university that requires a data structures or equivalent higher level programming course. This means that while some courses in disciplines such as digital media or data science had multiple programming courses in their curriculum, these degree programs did not include classes with deeper programming concepts and therefore were categorized as STEM or non-STEM rather than computing. For the courses that did not meet the computing standard, factors such as the course department, type of degree the department conferred, and courses in the department were used to determine if a course was STEM or non-STEM. To be considered a STEM course, the department of the course must offer in-depth math and science courses and offer degree programs that require those types of courses. Any course that did not meet the STEM requirements were categorized as non-STEM. Furthermore, any courses in a department that is traditionally non-STEM, such as linguistics or music, were classified as non-STEM. Each course was reviewed on a case-by-case basis to ensure that it was appropriately categorized. The categories of the courses were then used to determine the disciplines that were represented in each category. Since course discipline is based on course department and course category is based on the intended student audience, some courses fall within the same discipline but are in different categories. This is seen most notably with the computer science and computer science engineering discipline (CS/CSE) as many courses were offered by CS/CSE related departments, but did not meet the requirements to be considered computing (Table 2-3).

Findings

In this section, we present the results of the surveys of course structure. This includes programming languages used, programming environments used, and instructor perspectives on collaboration in fundamental programming classes. We also consider the course discipline to examine the difference between computing, STEM, and non-STEM course structure.

Of the 132 instructors contacted, we collected a total of 81 responses. The number of instructors and the number of courses are used interchangeably, as each response is representative of that course and the instructor's course policies. We categorized the reported departments into discipline areas. Computer science was the most frequently reported discipline, making up 25.93% of total instructor disciplines. The second most frequently reported department was Engineering, making up 12.35% of total instructor disciplines (Table 1). We also further categorized each department into three categories: computing, STEM, and non-STEM. Of the total 81 responses, 25.93% of the courses were categorized as computing and 54.32% were categorized as STEM (Figure 1). While computing and STEM departments were more common, non-STEM disciplines still included a notable 19.75% of total reported departments (Figure 1). The most frequently reported discipline in the computing category was the computer science and computer science engineering (CS/CSE) discipline which made up 76.29% of the computing courses (Table 2). Additionally, the most common STEM course discipline was engineering which made up 22.73% of the courses categorized as STEM (Table 3). Of the non-STEM courses, media and art was the most common discipline with 31.25%. The remaining instructors were almost evenly distributed between the business, geography, and linguistics disciplines with 25.00%, 18.75%, and 18.75% of the total non-STEM disciplines respectively (Table 4). It is also important to note that the media and art discipline, a non-STEM category, made up 6.17% of total instructor departments, making it the sixth highest reported discipline (Table 1). The computing and STEM categories both contain courses within the same discipline. This includes information science, data science, and CS/CSE. All of these courses were offered by a department that fell into their respective discipline; however, some of the courses met the computing category requirements, while others did not and were therefore categorized as STEM. Information science and data science courses each made up 4.67% of the computing discipline, the lowest reported computing courses (Table 2). Information science, data science, and CS/CSE courses each made up 11.36% of the STEM courses (Table 3).

Table 1: Disciplines Represented in all Courses (N=81)

Discipline	Count	Percentages
CS/CSE	21	25.93%
Engineering	10	12.35%
Earth Science	6	7.41%
Information Science	6	7.41%
Data Science	6	7.41%
Media & Art	5	6.17%
Stats	5	6.17%
Math	4	4.94%
Business	4	4.94%
Biology	3	3.70%
Linguistics	3	3.70%
ECE	3	3.70%
Geography	3	3.70%
Physics	1	1.23%
Psychology	1	1.23%

Table 2: Disciplines Represented in Computing (N=21)

Discipline	Count	Percentages
CS/CSE	16	76.19%
ECE	3	14.29%
Information Science	1	4.76%
Data Science	1	4.76%

Table 3: Disciplines Represented in STEM (N=44)

Discipline	Count	Percentages
Engineering	10	22.73%
Earth Science	6	13.64%
Information Science	5	11.36%
Data Science	5	11.36%
Stats	5	11.36%
CS/CSE	5	11.36%
Math	4	9.09%
Biology	3	6.82%
Physics	2	2.27%

Table 4: Disciplines Represented in non-STEM (N=16)

Discipline	Count	Percentages
Media & Art	5	31.25%
Business	4	25.00%
Geography	3	18.75%
Linguistics	3	18.75%
Psychology	1	6.25%

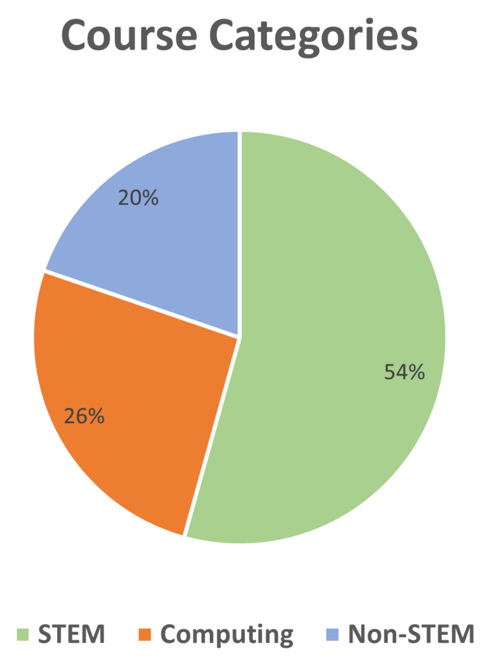

Figure 1: Course Categories Percentages

A variety of programming languages and environments were reported. Python was by far the most popular language with 53.09% of the surveyed instructors reporting using the language in their course (Table 5). Furthermore, 38.10% of computing instructors, 54.55% of STEM instructors, and 68.75% of non-STEM instructors reported using Python in their course, making Python the most reported language in each individual category as well (Tables 6-8). MATLAB and R were also notably popular languages, making up 17.28% and 19.75% of reported languages respectively (Table 5). Most of the instructors that reported using these languages came from STEM disciplines, as no non-STEM instructors reported using MATLAB (Table 8) and each language only made up 4.67% of computing course languages (Table 6). Of the 29.55% STEM instructors who reported using MATLAB in their course (Table 7), approximately half of the cases were from 8 of the total 10 engineering instructors. Only one engineering instructor reported using R in their course. In addition to MATLAB and R, there are several instances of popular programming languages in one or more disciplines that are not as frequently used—or not used at all—in other disciplines. One example is C++, which was reported being used by 28.57%

of computing instructors and 6.82% of STEM instructors, but no non-STEM instructors reported using this language. Another example is the programming language Visual Basic, which was only reported by instructors in STEM disciplines (Table 7).

Table 5: Programming Languages Used

Language	Percentages
Python	53.09%
R	19.75%
MATLAB	17.28%
Java	14.81%
C++	11.11%
JavaScript	7.41%
C	6.17%
SQL	4.94%
Visual Basic	2.47%
PHP	2.47%
Other(s)	6.17%

Table 6: Programming Languages Used in Computing Courses

Language	Percentages
Python	38.10%
Java	38.10%
C++	28.57%
C	23.81%
JavaScript	9.52%
R	4.76%
SQL	4.76%
PHP	4.76%
MATLAB	4.76%
Other(s)	9.52%

Table 7: Programming Languages Used in STEM Courses

Language	Percentages
Python	54.55%
MATLAB	29.55%
R	20.45%
C++	6.82%
Visual Basic	4.55%
Java	4.55%
SQL	2.27%
JavaScript	2.27%
Other(s)	2.27%

Table 8: Programming Languages Used in Non-STEM Courses

Language	Percentages
Python	68.75%
R	37.50%
JavaScript	18.75%
Java	12.50%
SQL	12.50%
PHP	6.25%
Other(s)	12.50%

The preferences in programming languages is reflected in the environments used. Some of the most reported IDEs, such as PyCharm and Spyder, are Python language programming environments. The most frequently reported programming environment was Jupyter Notebooks, with 24.69% of instructors reporting using it for their course. Furthermore, 19.75% of instructors reported using a programming environment other than the ones listed in the survey (Table 9). In addition to reporting the coding environment used, instructors were asked a series of free-response questions to record their opinions on their preferred IDE. Many instructors reported ease of access and beginner friendliness to be the main strengths of the environments they use. Many also preferred free, open-source, and cross-platform integrated editors, such as Spyder. (Table 9). Additionally, several instructors reported having trouble with the debugging tools in their selected programming environment.

Table 9: Environments Used in Courses

Environment	Count
Jupyter Notebooks	24.69%
Visual Studio Code	20.99%
RStudio*	11.11%
PyCharm	11.11%
ZyLabs	9.88%
Spyder*	7.41%
Google Colab	7.41%
Visual Studio	6.17%
IntelliJ IDEA	6.17%
Replit	6.17%
Eclipse IDE	4.94%
IDLE*	4.94%
Netbeans*	4.94%
MATLAB*	3.70%
Atom	3.70%
Notepad++*	3.70%
ArduinoIDE	2.47%
XCode	2.47%
Anaconda*	2.47%
Clion*	2.47%
Other	19.75%

*Not listed in the survey options

We also asked a series of questions regarding collaborative work and the instructor's reasoning for including or not including it as part of the coursework. Many instructors had a positive view of collaboration with 50.62% of the surveyed instructors reporting at least some required collaboration in their course and 12.35% optional collaboration (Table 10). On the other hand, 37.04% of instructors surveyed allowed no collaboration (Table 10). Of the 21 computing instructors surveyed, 28.57% did not require collaboration, 9.52% reported optional collaboration, and 61.90% reported at least some required collaboration in their courses (Table 11). Of the 44 STEM instructors surveyed, 43.18% did not require collaboration, 18.18% reported optional collaboration, and 38.64% reported requiring at least some collaboration in their courses (Table 12). Finally, of the 16 non-STEM instructors surveyed, 31.25% reported no collaboration and 68.75% reported required at least some collaboration. None of the non-STEM instructors reported optional collaboration (Table 13).

Those who reported having no collaborative project work were asked for their reasoning for their decision. Of those instructors, 23.33% claim collaboration makes fair grading difficult, 23.33% claim collaboration facilitates cheating, and 16.67% claim students learn better individually. The remaining 36.67% cite other reasons for not including collaboration in their course (Table 14). While there are a variety of other reasons instructors do not allow collaboration, one of the recurring themes is the inability to gauge individual performance in group projects and how to ensure each student is mastering the course content.

Table 10: Percentages of Level of Collaboration in Instructor's Course(s) (N=81)

Collaboration Level	Percentages
Optional Collaboration	12.35%
Requires Some Collaboration	50.62%
No Collaboration	37.04%

Table 11: Percentages of Level of Collaboration in Computing Courses (N=21)

Collaboration Level	Percentages
Optional Collaboration	9.52%
Requires Some Collaboration	61.90%
No Collaboration	9.52%

Table 12: Percentages of Level of Collaboration in STEM Courses (N=44)

Collaboration Level	Percentages
Optional Collaboration	18.18%
Requires Some Collaboration	38.64%
No Collaboration	43.18%

Table 13: Percentages of Level of Collaboration in Non-STEM Courses (N=16)

Collaboration Level	Percentages
Optional Collaboration	0.0%
Requires Some Collaboration	68.75%
No Collaboration	31.25%

Table 14: Instructor Reasons for not Allowing Collaboration

Reason	Percentages
Facilitates Cheating	23.33%
Makes Fair Grading Difficult	23.33%
Students Learn Better Individually	16.67%
Other(s)	36.67%

Discussion

As expected, CS/CSE made up most of the surveyed instructor disciplines; however, our survey still provided insight into multiple introductory programming courses in a variety of disciplines. For instance, STEM departments such as earth science, information science, data science, statistics, and math all included notable portions of the instructors surveyed. Additionally, engineering was the most common STEM department with the second-most number of overall reported courses. Computing skills are becoming increasingly important for STEM majors as advanced technology becomes more integrated throughout STEM disciplines [15]. It has also been found that the computational thinking skills learned in programming are critical for STEM students in the twenty-first century [16]. Therefore, it is evident that programming courses are becoming a fundamental part of STEM curricula. In addition to STEM, there were also a significant number of non-STEM disciplines with introductory programming courses as well. The media and art discipline was particularly notable, as it was one of the most frequently reported course disciplines. Media and art courses were reported at a similar or even higher rate than most of the STEM and computing disciplines, except for computer science and engineering. Like many other disciplines, there has been a rise in the use of technology throughout the art discipline. We can see this in subjects such as video game design, animation, computer graphics, and even robotics [17]. Digital media can help students achieve their goals in both the arts and computing disciplines while simultaneously expanding the scope and capabilities of both [17]. While the other non-STEM disciplines were less frequent, it is still worth noting that the business, geography, and linguistics disciplines were all reported with similar numbers as some STEM and computing disciplines. Furthermore, many of the introductory programming courses provided to these students were tailored to their chosen discipline, such as "Programming for Biologist" or "Computational Techniques for Finance." Courses designed to accommodate a variety of students interests and backgrounds can offer a more motivating and engaging context for learning programming and CS concepts [18]. The media and art discipline is a prime non-STEM example of this concept, as core computer science concepts can be taught through media computation and art software [19]. Commonly reported non-computing courses were a part of interdisciplinary degree programs with an emphasis on programming, such as interactive media or game design. However, there were also a number of non-STEM courses in programs not traditionally focused on programming, such as linguistics and political science. Even if students do not need to actually code in their careers, programming courses can still provide non-computing majors with the means to better communicate with programmers more effectively [1]. Being able to work on end-user programming tasks such as data analysis and project management can improve students perceived job marketability in the software industry [1]. Overall, the goal of introductory programming courses for non-computing majors is to provide students will computing skills that will be applicable to their future careers. Computing has the potential for impact across the range of disciplines, not just in computing fields [19].

In addition to course offerings, we were also able to identify trends in course structure. Introductory programming courses often focus on teaching the principles of object-oriented programming. Python is a beginner friendly programming language for students to learn fundamental coding concepts (functions, variables, for loops, etc.) without the syntactical overheads presented by other languages such as Java [14]. Because of this, Python has steadily grown in popularity as the language to teach beginners to code [20], which is reflected in the results of our survey. Python was by far the most popular language used by instructors in their introductory programming courses, regardless of discipline. The programming languages R and MATLAB were also notably popular among computing and STEM courses, MATLAB being especially popular among engineering instructors. This is likely due to their applications in statistics, general engineering, and other purposes outside of pure computer science programming. R and MATLAB are arguably more useful for non-computing majors, as they can be used in a variety of fields without having to learn deeper programming concepts. Visual Basic was also popular among STEM instructors, specifically engineering instructors as all of the courses that used Visual Basic were in an engineering discipline. While there may be some uses for Visual Basic in computing or non-STEM disciplines, it is likely most popular in engineering, and STEM as a whole, due to its application in simulating physical and mathematical phenomena and processes. Visualization of this complex content can foster deeper understanding of STEM-related material, which makes Visual Basic a valuable programming language for STEM educators and students [21]. Furthermore, C was the only popular programming language that was reported exclusively by computing instructors. This suggests that knowledge of C is not in high demand outside of computing disciplines. By studying instances of popular programming languages in one or more disciplines and by identifying which languages are popular in computing but not in STEM or non-STEM disciplines, we can examine which languages are most useful to STEM and non-STEM undergraduate students. Of the programming environments instructors reported using, most chose their IDEs based on ease of access, beginner user friendliness, and no cost for university students. A student's attitude towards and perception of programming greatly impacts their motivation to learn computer science [22]; therefore, it is important that these IDE preferences are noted in the development of any tools or resources for introductory programming to ensure that students new to computing have accessible and welcoming options. Furthermore, collaboration has become a significant aspect of programming, as several studies have found there are many benefits to collaborative programming in both the classroom and in the programming industry [23]. Collaboration was encouraged, or at least allowed, in the majority of the computing and non-STEM courses surveyed. Although some instructors did not allow collaboration, citing a variety of reasons, allowing some form of collaboration appears to be preferred, especially among computing instructors. This suggests that most of the surveyed instructors share this positive view on collaboration. However, based on the results of the survey, STEM instructors appear to prefer no collaboration in their courses. Collaboration can be beneficial for STEM students in general, as it can improve the learning experience for students and enhance skills such as problem solving or creativity [24]. Despite this, no collaboration was the most common level of collaboration allowed in the STEM courses surveyed. This appears to be mostly due to the science, technology, and math courses, as all but three engineering instructors allowed some level of collaboration in their course. This is likely due to the importance of collaboration in the engineering field. Overall, the results do not provide any insight on the benefits or drawbacks of collaboration in the courses of the surveyed instructors; however, some collaboration may offset the challenges that the instructors face in teaching the programming courses and improve the perspectives of non-computing students within their programming classes.

Limitations & Future Work

Overall, the study provided a strong foundation for future research on introductory programming courses for non-majors; however, it still had several limitations. For instance, the course data collected is only representative of a large 4-year public university. Additionally, while the survey provided useful data on instructor course policies, we could not gain an in-depth understanding of their reasoning behind these policies. Furthermore, our research did not include feedback from the students on the courses. Student feedback on the course structure would be a valuable insight on what formats work the best for beginners learning to code. Our survey would also benefit from more emphasis on the collaborative aspects of programming courses.

Future research will include applying our observations to future STEM and non-STEM courses and analyzing the instructors' and students' perspectives on their performances throughout the semester. In a follow-up study, we will perform another survey that will focus on collaborative coursework. This survey will provide more information on how collaboration can improve learning in introductory programming courses.

Conclusion

In our study, we examined university courses that taught introductory programming courses to identify their prevalence in disciplines within and outside of computing. To learn more about the courses offered, we identified the 50 largest public universities in the United States and examined their course offerings, programming languages, and collaboration policies. Of the 81 total courses reviewed, we found that computing courses are offered at a variety of STEM and non-STEM disciplines. Additionally, we were able to identify trends in programming languages and environments used, as well as course policies in regards to collaboration. The results of the survey and our analysis provide insight into the disciplines in which computing coursework is becoming more prevalent, use cases within those disciplines, and what techniques they employ. Our findings will support future research of fundamental programming courses beyond computing disciplines.

References

[1] P. K. Chilana, C. Alcock, S. Dembla, A. Ho, A. Hurst, B. Armstrong, and P. J. Guo, "Perceptions of non-CS majors in intro programming: The rise of the conversational programmer," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 251–259, Oct. 2015.

[2] P. Ruvolo, D. Mir, and Z. Dodds, "Embracing our Future: CS Courses and Curriculum for Non-CS-majors," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, (New York, NY, USA), p. 1234, Association for Computing Machinery, Feb. 2019.

[3] J. E. L. Shin, S. R. Levy, and B. London, "Effects of role model exposure on STEM and non-STEM student engagement," *Journal of Applied Social Psychology*, vol. 46, no. 7, pp. 410–427, 2016. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jasp.12371.

[4] G. Jin and T. Bierma, “STEM for non-STEM Majors: Enhancing Science Literacy in Large Classes,” *Journal of College Science Teaching*, vol. 042, no. 06, 2013.

[5] A. M. Gansemer-Topf, A. Kollasch, and J. Sun, “A House Divided? Examining Persistence for On-Campus STEM and Non-STEM Students,” *Journal of College Student Retention: Research, Theory & Practice*, vol. 19, pp. 199–223, Aug. 2017.

[6] D. F. Whalen and I. I. Mack C. Shelley, “Academic Success for STEM and Non-STEM Majors,” *Journal of STEM Education: Innovations and Research*, vol. 11, Feb. 2010.

[7] M. M. Rahim, A. H. Seyal, and M. N. A. Rahman, “Factors Affecting Softlifting Intention of Computing Students: An Empirical Study,” *Journal of Educational Computing Research*, vol. 24, pp. 385–405, June 2001.

[8] M. E. Engberg and G. C. Wolniak, “College Student Pathways to the STEM Disciplines,” *Teachers College Record*, 2013.

[9] B. Chowdhury, A. C. Bart, and D. Kafura, “Analysis of Collaborative Learning in a Computational Thinking Class,” in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, (Baltimore Maryland USA), pp. 143–148, ACM, Feb. 2018.

[10] D. J. Shernoff, S. Sinha, D. M. Bressler, and L. Ginsburg, “Assessing teacher education and professional development needs for the implementation of integrated approaches to STEM education,” *International Journal of STEM Education*, vol. 4, p. 13, June 2017.

[11] C. McDowell, L. Werner, H. Bullock, and J. Fernald, “The effects of pair-programming on performance in an introductory programming course,” in *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, SIGCSE ’02, (New York, NY, USA), pp. 38–42, Association for Computing Machinery, Feb. 2002.

[12] A. M. O. Hmelo-Silver, Cindy E., “Introduction: What is Collaborative Learning?: An Overview,” in *The International Handbook of Collaborative Learning*, Routledge, 2013. Num Pages: 15.

[13] N. Alzahrani, F. Vahid, A. Edgcomb, K. Nguyen, and R. Lysecky, “Python Versus C++: An Analysis of Student Struggle on Small Coding Exercises in Introductory Programming Courses,” in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE ’18, (New York, NY, USA), pp. 86–91, Association for Computing Machinery, Feb. 2018.

[14] A. Jayal, S. Lauria, A. Tucker, and S. Swift, “Python for Teaching Introductory Programming: A Quantitative Evaluation,” *Innovation in Teaching and Learning in Information and Computer Sciences*, vol. 10, pp. 86–90, Feb. 2011. Publisher: Routledge _eprint: https://doi.org/10.11120/ital.2011.10010086.

[15] S. I. Swaid, “Bringing Computational Thinking to STEM Education,” *Procedia Manufacturing*, vol. 3, pp. 3657–3662, Jan. 2015.

[16] R. Prinsley and E. Johnston, *Transforming STEM teaching in Australian primary schools: everybody’s business.* Dec. 2015.

[17] K. A. Peppler and Y. B. Kafai, “Creative Coding: Programming for Personal Expression,” (Rhodes, Greece), 2009.

[18] A. Forte and M. Guzdial, “Motivation and nonmajors in computer science: identifying discrete audiences for introductory courses,” *IEEE Transactions on Education*, vol. 48, pp. 248–253, May 2005. Conference Name: IEEE Transactions on Education.

[19] M. Guzdial, “A media computation course for non-majors,” in *Proceedings of the 8th annual conference on Innovation and technology in computer science education*, ITiCSE ’03, (New York, NY, USA), pp. 104–108, Association for Computing Machinery, June 2003.

[20] R. M. Siegfried, K. G. Herbert-Berger, K. Leune, and J. P. Siegfried, “Trends Of Commonly Used Programming Languages in CS1 And CS2 Learning,” in *2021 16th International Conference on Computer Science & Education (ICCSE)*, pp. 407–412, Aug. 2021. ISSN: 2473-9464.

[21] Z. Akhatayeva, K. Sagindykov, B. Mukushev, N. Kurmangaliyeva, and A. Karipzhanova, “Visual Basic and MathCAD used for Visualization and modeling STEM education,” *Education and Information Technologies*, vol. 29, pp. 23011–23026, Dec. 2024.

[22] M. Biggers, A. Brauer, and T. Yilmaz, “Student perceptions of computer science: a retention study comparing graduating seniors with cs leavers,” *ACM SIGCSE Bulletin*, vol. 40, pp. 402–406, Mar. 2008.

[23] L. Williams and R. Kessler, “The effects of ”pair-pressure” and ”pair-learning” on software engineering education,” in *Thirteenth Conference on Software Engineering Education and Training*, pp. 59–65, Mar. 2000. ISSN: 1093-0175.

[24] K. Kärkkäinen and S. Vincent-Lancrin, “Sparking Innovation in STEM Education with Technology and Collaboration: A Case Study of the HP Catalyst Initiative,” tech. rep., OECD, Paris, Apr. 2013.