Issues in Information Systems
Volume 24, Issue 2, pp. 139-152, 2023

DOI: https://doi.org/10.48009/2_iis_2023_112

Considerations for updating programming courses

Azad Ali, *Indiana University of Pennsylvania, azad.ali@iup.edu*
Pankaj Chaudhary, *North Carolina A&T State University, pchaudhary@ncat.edu*
Kustim Wibowo, *Indiana University of Pennsylvania, kwibowo@iup.edu*

Abstract

Educators in computing-related degree programs (CS, IT, and IS) may need to regularly update their courses and programs to keep them relevant to the in-demand industry skills. Continuous and rapid changes require regular revisions in computing programs and curricula. Courses that teach computer programming, especially introductory programming language courses, need to be updated to reflect the languages in demand in the industry. Not being current with curriculum updates for these courses will likely put the programs behind due to outdated curricula. This is the case for the computer science (CS) program at a university located in Western Pennsylvania. The syllabi for the programming courses have not been updated for some time. In the process of updating the programming courses, an important question to address is the critical factors to consider when updating the syllabi of the programming course(s). This paper reviews the relevant literature and discusses the factors to be taken into consideration when updating computer programming courses. A set of recommendations for course updates are provided toward the end of the paper.

Keywords: Programming courses, computer programming, challenges of updating programming courses

Introduction

Educators in computing-related programs like computer science (CS), information technology (IT), and information systems (IS) are familiar with changes in curriculum and the need for regular curriculum revisions. Among the courses that need to be and are updated regularly are courses that teach computer programming languages (Becker & Quille, 2019). Various programming courses could fall under different categories and names in the reference ACM curriculum (Becker et al., 2022; Cao, 2023). Some computing programs require their students to complete more than one programming course. In comparison, others may require only one course, but the common issue among computing programs is that programming courses require regular revisions and updates to stay current with technological advances.

The constant technological changes, changing philosophies, and increasing use of abstraction in programming require updating course content regularly. These changes are often reflected in the course syllabi that describe the content of the courses. Outdated courses usually do not sit well with students regarding the program being in step with the industry. For example, not teaching Python in a computing program today may not sit well with most current and prospective students. Since programming is a critical skill in all computing programs, the programming course being out of tune may affect student enrollment as students may look unfavorably to outdated courses and programs (Krishnamurthi & Fisler, 2019).

The CS program in the university, triggering this inquiry, needed an update in the programming language courses. As part of a regular curriculum review, it was inferred that an update to the programming language was needed to bring the programming courses up to date with the industry trends. Four courses covering programming languages are being reviewed. The course syllabi had also not been updated for some time.

Though the choice of languages is sometimes easily made through the survey of practitioner literature, this exercise was undertaken as a formal structured academic project. A literature review of academic literature was conducted to infer the state of teaching computer programming courses and arrive at an informed decision. The focus was on the first two programming courses in the degree program. These are referred to as CS1 and CS2 in this manuscript.

An initial review of the literature revealed that the primary factors to consider when reviewing programming course syllabi are:

- The choice of programming language to teach in the course.
- The challenges that students will face during the learning of programming language.
- The future trends of programming courses.

Based on the initial findings, a more thorough literature review was conducted, and the results were categorized according to the three primary factors. These findings are presented in the following sections. These findings are used to draw recommendations for updating the CS1 and CS2 course content/syllabi.

Programming languages rankings

Most programming courses cover a single programming language though some courses may cover multiple programming languages. Such is usually the case with courses teaching programming language theory and history. Courses similar to CS1 and CS2 are more likely to cover a single programming language (Baltes & Diehl, 2018). Hence the question boils down to the choice of the programming language for the two courses with the additional question of whether the same language should be covered in both courses or different languages can be chosen.

Multiple factors influence the decision regarding the choice of programming language, and the decision is anything but simple and/or straightforward (Becker et al., 2022; Cheah, 2020). Of all these factors, there are typically three factors that are of most concern to IT educators. These include the popularity of the different programming languages, the job market for various programming languages, and which languages are trending in the industry and the market.

Many programming languages could be used in a course. This number has been on the rise though some trusty and enduring languages have been around for some time, like Java, C, and C++. New languages like Rust, Julia, and others have emerged over the last few years. Ali and Smith (2014) suggested that there are at least 2000 languages that are in use in various software today.

Upson (2023) provides a lower bound based on mainstream usage and new software development and suggests somewhere between 250-2500 coding languages. But even with the lower bound of 250, the number is still high enough. This, combined with the additional factors outlined before, makes the selection of language a non-trivial task.

Computing program educators often use the term “widely used programming language” as a justification for the selection and use of a programming language over others. Kumar and Dahiya (2017) suggested that the concept of “widely used programming language” cannot be measured that easily and suggested some formal criteria to determine usage:

- Counting the number of times a given language is searched in web search engines.
- Job advertisements, specifically, the number of jobs available for a given programming language.
- Books sold for a programming language.
- The number of projects on popular websites such as GitHub and SourceForge.

The exact number of programming languages used in the industry or taught in colleges is hard to determine. Many programming languages are in use but are not mentioned in industry literature frequently. Liu and Wu (2018), for example, note different programming languages like Looking Glass, Eliza, PiE, TT Programming, and Alice, which are used and supported by several well-known universities; however, they are not mentioned in any significant way in the practitioner literature regularly. These languages also do not appear in various programming language rankings (like the TIOBE index) compiled by the industry. The mission of universities is to teach concepts and critical thinking; hence, what is popular is not necessarily appropriate. For example, students may be better positioned when they learn strongly typed languages than weakly typed ones like Python. The distinction may also be made between academic language, scripting languages, general purpose languages, and others (Coursera, 2023).

Programming languages can be categorized as scripting languages, compiled languages, markup languages, web languages, and others. Each of these has different benefits and characteristics. Scripting languages are sometimes easy to teach. However, often do not provide the benefit of forethought and careful program layout. Coursera.com lists the following types of programming languages:

- Procedural programming language
- Functional programming languages
- Object-oriented programming languages
- Scripting languages
- Logic programming languages

Coursera.com also classifies programming languages in terms of their intended functionality.:

- Front end (client side) versus back end
- High-level versus low-level programming languages
- Interpreted versus compiled languages.

Selection of a particular type of programming language based on its characteristics and fulfilling the course objectives may prove non-trivial. This selection may be further complicated if two different kinds of languages are deemed to be apt for CS1 and CS2 to provide a wider exposure. Selection of the programming language in the two courses may need some formal research and the use of formal criteria to rank them for appropriateness.

Popularity of programming languages

The popularity of programming languages is an essential criterion since it reflects the demand for skills from the industry that graduates need to have. University educators and practitioners often refer to the "Popularity of Programming Languages" concept as a reason for selecting one language over the other.

Different practitioner websites, industry organizations, and professional organizations like IEEE and ACM, use somewhat different methodologies to measure the popularity of programming languages. As such, the ranking they presented often differs somewhat though primarily, they appear to be in the same ballpark.

Some organizations like TIOBE (https://www.tiobe.com/tiobe-index/) have been measuring popularity regularly and have, in some sense, become somewhat authoritative voices on language popularity. TIOBE, which calls itself "the software quality company" lists monthly index tables ranking the programming languages in use. The TIOBE programming index table ranks the popularity of programming languages according to factors such as the number of skilled engineers worldwide, courses and third-party vendors, how often it is searched by search engines, etc. (Larson, 2022).

The table below provides the top 10 language rankings for April 2023. The table also presents comparative statistics from a year ago and the rise and fall of different languages. It should also be noted that all languages on this list are not the latest, and some interesting candidates are on the list like Assembly and Delphi/Pascal. It is interesting to see C in #2 place. C, which was the cornerstone of most computing curricula, is not so based on the limited knowledge of the authors from different 4-year institutions.

Table 1: TIOBE Index of Programing Language as of April 2023 (https://www.tiobe.com/tiobe-index/)

Ranking as of April 2023	Ranking as of April 2022	Programming Language	Rating
1	1	Python	14.51%
2	2	C	14.41%
3	3	Java	13.23%
4	4	C++	12.96%
5	5	C#	8.21%
6	6	Visual Basic	4.4%
7	7	JavaScript	2.10%
8	9	SQL	1.68%
9	10	PHP	1.36%
10	13	Go	1.28%

Language popularity is a good proxy for industry demand. However, other factors, like the course learning outcomes, play an important role. For example, if a deep and conceptual understanding of object orientation (abstraction, encapsulation, inheritance, and polymorphism) is desired, Java may be the language of choice. However, C may be the language of choice if the objective is to train students to write code that works closely with the OS (especially Linux and Unix).

Market Demand for Programming Languages

As mentioned before, language popularity is a good proxy for industry demand. Other factors also drive the industry demand. When recruiting graduates, the industry is often more forward-looking than backward-looking, which means that the industry is looking to build capabilities in what is to come. Younger graduates are easy to train and mold to different future roles. Cao (2023) argues that the curriculum selection of a programming language should be driven by the job opportunities available for that language.

Some job ads are specific in requiring experience and knowledge of particular programming language(s). If there is a higher demand for jobs that require knowledge of a particular programming language, then it is logical to assume that more students will gravitate toward it. The opposite could be true for languages that have lower job demands. The market demand for a programming language can be measured in several ways, the most popular of which is counting the job ads that advertise requirements for programming language by name or the programming platforms that are used by big companies and organizations.

The demand for jobs requiring a given programming language may not be the sole criterion for gauging market demand. Larson (2022) suggests that not all high-demand jobs indicate better hiring opportunities. Larson (2022) provided another critical factor: the number of applicants per job ad. A higher number of applicants per job ad reduces the chances for a graduate to succeed in obtaining the job opportunity. Hence, caution may be needed when looking at the number of job openings for a particular language. With the oncoming of Artificial Intelligence (AI) co-pilots that aid in coding, the availability of more sample codes for AI to learn from the market may shift unexpectedly within the next few years.

A pertinent factor when considering the market demand for a programming language is the concept of skill gaps and their duration. The skill gap is prominent, especially when a new programming language and area of emphasis emerges in the market. An example of this may be things like quantum computing and Julia as a programming language.

During the early stages of using the new programming language, expertise will be needed to work using the new language. As programming languages become more entrenched in the industry, this skill gap does not remain the same. This is more due to various tools jumping on board to offer courses, programmers more willing to upskill themselves, and the industry actively working to reduce the skills gap. So, the trend of increase slows down more rapidly.

The Institute of Electrical and Electronics Engineers (IEEE) routinely publishes articles, podcasts, and infographics devoted to sciences and engineers. One of their publications is a magazine called the “IEEE Spectrum”, which contains articles about technology and science. A recent article in “IEEE Spectrum” listed the ranking of programming languages. The ranking is listed in three formats, first based on job demand, second based on the spectrum of programming languages, and third based on the trending of programming language. Figure 1 below shows the 2022 ranking of programming languages as published by IEEE Spectrum based on job demand.

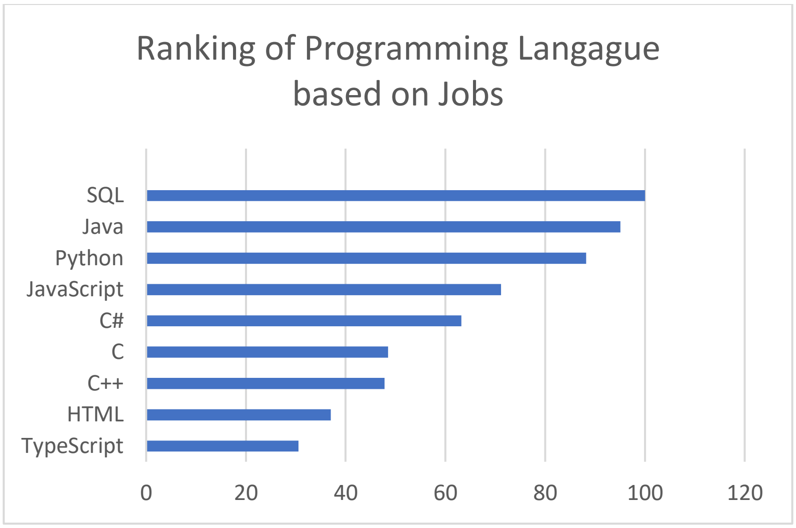

Figure 1: Ranking of Programming Languages Based on Job Demand
Source: https://spectrum.ieee.org/top-programming-languages-2022#toggle-gdpr

The spectrum of programming languages is another way that IEEE ranks programming languages. Spectrum implies the flexibility of the programming languages in their application to lower-level (system level) and higher-level (application level) tasks. Traditionally this has been the forte of C and C++, which are typically used to code different operating systems and Integrated Development Environments (IDEs). Put in another word, it refers to the adaptability of the language coupled with the ease of learning the language. Figure 2 below lists the Spectrum of programming languages as presented by IEEE Spectrum.

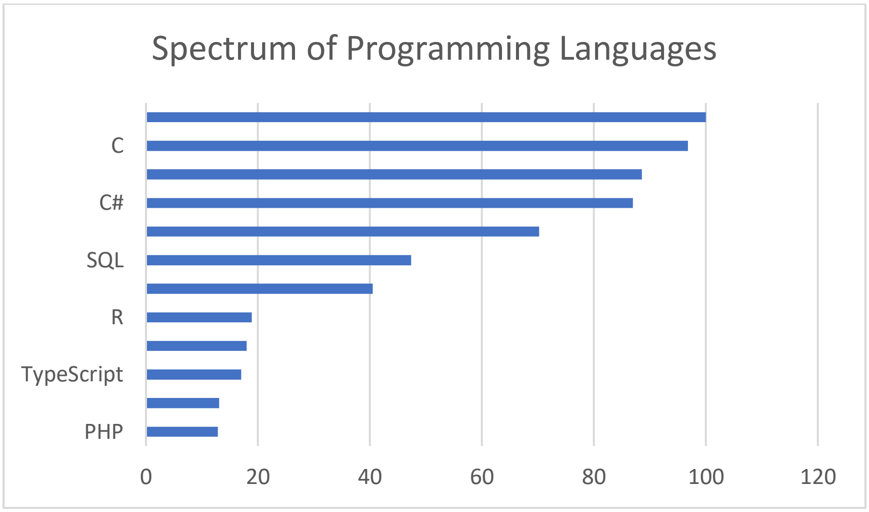

Figure 2: The Spectrum of Programming Languages
Source: https://spectrum.ieee.org/top-programming-languages-2022

Programming Language Longevity

Application maintenance is an essential consideration in the choice of programming languages. Several organizations in the financial industry are finding application maintenance a considerable lift with the attrition of COBOL programmers. The question of the availability of skilled personnel in a programming language, as more and more languages emerge to suit different tasks and applications, is critical since applications are maintained for a long time, especially those custom developed. How long will the programming language survive or have a critical mass of skilled personnel is pertinent. IEEE refers to this as the "Trending" of the programming language. With programming languages appearing in short intervals, it is hard to predict that specific programming languages will be around for a long (Tozzi, 2023). A significant amount of legacy code is written in Basic, COBOL, and Fortran. The legacy code or current code that may be deemed as a legacy is expected to be multiplied hundreds and even thousands of times. The factors contributing to the trending of programming languages are also increasing (Kumar & Dahiya, 2017). This is exacerbated by language evolution too. As languages evolve, some features, library functions, and syntax are deprecated. A good example is the Python 2.8 code vs. 3.x code. Some specific factors that lead to the longevity of a given programming language include the following:

- Number of times the programming language searches on popular websites (e.g. StackExchange.com)
- Length of programming language used in the industry.
- Jobs available for the programming language

The IEEE Spectrum lists different programming languages and their usage trends. Figure 3 below ranks the trending of programming languages:

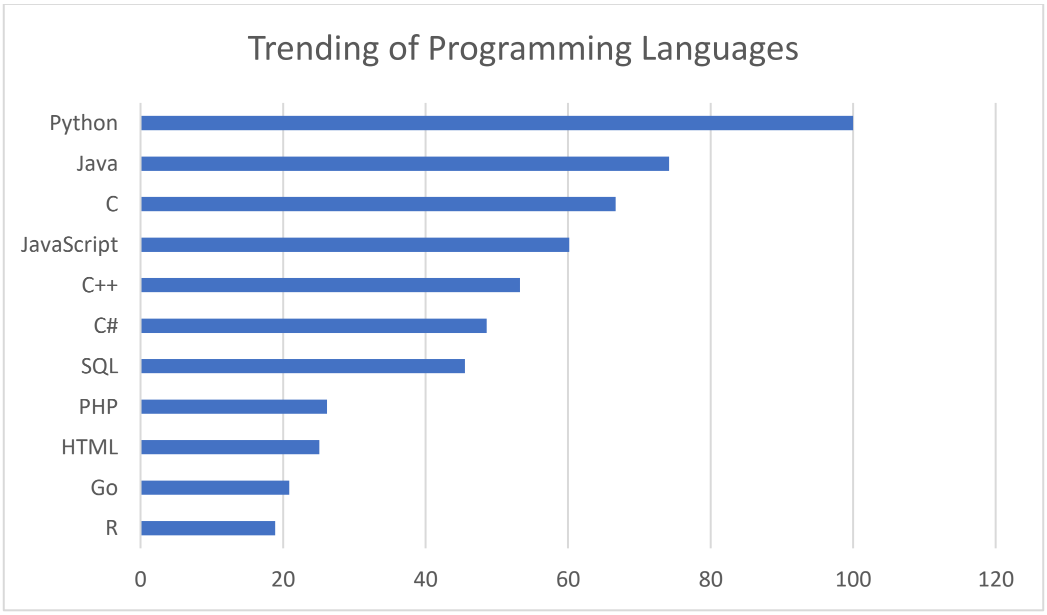

Figure 3: The Trending of Programming Languages
Source: https://spectrum.ieee.org/top-programming-languages-2022

Hence another criterion for the selection of a programming language may be the longevity of the programing language or the anticipated longevity of the language based on the application base and other factors. The different indexes and rankings discussed above are considerations that educators should keep in mind when considering a selection of programming languages to teach.

Programming language learning curve

Notwithstanding the ranking criteria discussed above, the learning curve associated with learning a language and/or the ease or difficulty of learning a language on the part of the students is another important consideration. Factors that lead to difficulties in learning a programming language were explored through a systematic literature survey of academic literature. The literature review reveals that learning to program is still considered challenging despite technological advances and the use of many tools to help in this endeavor. This difficulty also shows up in the high attrition rate among IT students after they take their first programming languages (Cheah, 2020). Three factors are prominently featured in academic literature when learning programming languages. These are the rigid syntax of a programming language, the unfamiliar structure followed in a programming language, and the general non-social perspectives of computing majors.

Rigid syntax

The syntax and semantics of programming languages remain to challenge numerous students as they learn computer programming and contribute to the high attrition among students taking programming courses. Liu and Wu (2018) noted the following about the syntax and semantics of programming languages:

> Programming remains a dark art for beginners or even professional programmers. Experience indicates that one of the first barriers to learning a new programming language is the rigid and unnatural syntax and semantics (p. 110).

The syntax of programming languages is rigid. Missing a comma or a semicolon or replacing a comma with a semi will generate an error message. This error message is often cryptic and not easy to understand to trace the origins of the error. When students get an error message, often they may resort to trial and error and not a systematic way of resolving the syntax error. Usually, an error in line 10 is due to an error on line 9, which tends to bewilder many students. This may take the student a long time steering at the code to work. Origins of this stem from the strict one-to-one mapping between syntax and semantics, which is atypical of the conversation in an everyday spoken language where variation in syntax produces almost still similar semantics.

Code editors that come with various IDEs help with syntax through autocomplete. AI-based autocompletes, which are often more intelligent than the normal ones like the Microsoft Copilot available with GitHub have further made keeping up with syntax easier. They give error messages and underline them differently to aid in better code writing. They also have too detailed messages about the error, which may sometimes be cryptic to a newbie to programming.

Despite all these technological enhancements, the issue of syntax still poses challenges to learning to program. Liu and Wu (2018) introduced what is termed as “Natural Languages”. They suggested steps that could move the student through understanding the program and then coding to minimize the effect of the challenges of learning the syntax of the programming languages.

Unfamiliar structure

The structure of the programming language is unfamiliar and foreign to many students. The structure of a COBOL program consists of four divisions. Each division is specialized for a specific task of the program. Changes in one division have a cascading effect on divisions in the program. Large programs are divided into smaller modules to manage large programs, and low coupling and cohesion are recommended so that each module can be maintained independently of each other.

That concept of structure has changed in newer programming languages. Programs are no longer divided into distinct sections (as in the case of COBOL) and students can code programs more easily than before (Ali & Smith, 2104). As long as the task being accomplished by the program is not too complex, the program can be limited to manageable lines of code; students take input, do some calculations, and produce some output. But the structure becomes complex as the tasks get more complicated and the programs do more. Code is divided into several modules, each of which may reside in a separate file and then be imported into other modules. The movement of data between these different modules is governed by a complex set of rules relating to scope. Within a single program, most common block-structured languages have somewhat different scope rules, making understanding the structure more complex. Added to this, weakly typed and strongly typed languages and dynamic binding can introduce structural complexities that may make it harder for students to program, especially when the course is beginning in programming. There is no doubt that programming is a complex task, and understanding nuances is required to be able to program well. However, sometimes this can lead to increased attrition of the students. Programming languages that remove some of the complexities, like weakly typed and script languages can help alleviate some problems and issues.

Programming as non-social activity

The issue of difficulties in the first programming course is not limited to the selection of the programming language, the syntax, and the structure; instead, it extends to the general perception of programming and to the method in which it is taught as an individual activity. Teague and Roe (2008) noted the following about programming:

> "Programming is often perceived as a solitary occupation, one which is conducted in a competitive, rather than collaborative environment. This is often reinforced at university where introductory programming subjects' assessment consists of individual assignments Programming is seen as a competitive occupation whose model student is the stereotypical 'geeky' young male, and this can lead initially to alienation, diminution of confidence, and subsequent lack of interest (p. 148)".

This perception of programming as a non-social/individualistic activity adds to the difficulty of learning to program and is not easy to overcome. The concerns about academic integrity and the prevalence of social loafing in student teamwork encourage the promotion of individual work and may discourage collaboration. While this is for a good reason, sometimes it is detrimental to the learning process. Students may sometimes learn better when they work in teams and complement each other. The advent of eXtreme Programming (XP) is based on the same concept. In a situation of individual work, for a student facing challenges, motivational issues often come into play quickly.

Most beginning examples of learning to program are simple tasks that take longer to develop. For example, a program for changing temperature from Fahrenheit to centigrade teaches the vital concept of input, processing, output, variable, and testing; spending an hour to write a program like this does not make sense to many students. After all, they can achieve the same result by pressing a few buttons on the calculator or at on their cell phones. Team programming, discussion boards, wikis, and other collaboration mechanisms can alleviate some of these issues by aiding students to complement each other's understanding. Programming languages and accompanying platforms that allow collaboration while writing code may be another consideration in choosing a programming language.

Future trends

With the arrival of artificial intelligence (AI) and Generative Pre-Trained Transformer (GPT) many tasks can now be offloaded to AI bases systems (Denny et al., 2023). Depending upon the complexity of the tasks, these systems can generate work/solutions of reasonable quality. When the stakes are not too high, the product can substitute for the work of humans, like summarizing the salient points of a football game. AI has also been employed to automatically generate programming code based on the user description of the problem to be solved (Sarsa et al., 2022).

The generation of programming code is not limited to one or a few programming languages; instead, the code can be generated for multiple programming from the same description (Denny et al., 2023). Also, this is not limited to small tasks that can be used for small teaching examples. Instead, the generated code could be based on large systems that involve multiple tasks and modules. For more complex tasks, however, it is essential to state that the code should be tested for performance accuracy. If testing fails, fixing the code often requires a deeper understanding of the language and advanced troubleshooting skills.

In light of these impressive AI tools, the pertinent question is whether it is time to switch from teaching programming languages the way it has been taught or switch to using GPT to teach the students how to generate code for different tasks. It is tempting that instead of using long hours to write code or just simply asking GPT with a few descriptions, it might generate the same that is asked for. The answer to this question has been provided in the sense that the code generated by these AI tools is not always correct, and the more complex the code, the deeper the knowledge of programming language is needed to correct this code.

A different approach from code generation is intelligent code completion using AI addons to IDE like GitHub Copilot and OpenAI Codex. These tools generally perform well to complete the assigned tasks (Denny et al., 2023) though the argument of trust but verification still holds. That requires knowledge of the programming language's concepts, syntax, and semantics.

Often, the question is how teaching programming should be approached considering recent developments in AI tools. Should one resort to the Microsoft GitHub Copilot as an add-on to Visual Studio and use a language that can be used with Visual Studio IDE, or should one use a language supported by OpenAI Codex? How much of the traditional teaching pedagogy should be part of the teaching of CS1 and CS2 courses? Sarsa et al., (2022) are very much in favor of using AI platforms to generate code, but at the same time note that teaching students to write programming code is more helpful than if they started to use these platforms and tools.

It should be noted that this discussion is similar to one that started while teaching HTML and web programming. The initial phases focused on teaching HTML coding with the knowledge tags and manipulating these tags using JavaScript. However, as more software applications were developed to create web pages using click and drop, more academic institutions introduced courses to teach this software, like MS Frontpage and Adobe Dreamweaver. This led to the introduction of a new classification of software called Web Authoring Software. This software provides tools that authors click and drop and as a result, generate web pages that are volumes of lines of HTML and JavaScript. It is undeniably tempting to teach the students about this web authoring software so they can use it for personal and work purposes.

As the structure of a website becomes increasingly more complex, the knowledge of HTML tags, the Document Object Model (DOM), and JavaScript becomes essential to achieve the end goals. Highly complex and customized websites are often made using a combination of auto-generation and manual customization. Manual customization needs a detailed and deep knowledge of underlying technologies. A similar argument can be made for code generation using AI add-ons. Complex code can only be written manually and will need a deep knowledge of the programming concepts and the programming language. Hence, the utility of teaching programming through a language of choice still holds for computing majors, distinguishing them from non-expert or causal coders.

Discussion and findings

Based on the criteria considered so far, one may develop recommendations for selecting a programing language to teach in CS1 and CS2 courses.

Selection of a programming languages

The indexes ranking various programming languages present a consistent pattern for which languages appear at the top. Python, C, C++, Java, C#, and SQL seem amongst the top 10 consistently. Thus, choosing any of these languages and claiming to have been selected from the top programming languages is a safe bet for most educators and justifiable to some extent. However, the group of C languages (C, C++, Java, and C#) are primarily similar in many aspects, so commonsense dictates that are selecting two of these languages to teach in the CS1 and CS2 courses may create some duplication both in terms of concepts and syntax, however with the benefit of reinforcing concepts over two semesters. This can lead to deep understanding. On the other hand, it may be more advantageous to provide exposure to different languages, given the variety of languages in use. Given the computing discipline's spirit of continuous learning, it is recommended that exposure is perhaps more important than depth in the initial stages of the computing curriculum. Thus, a combination of C-like language and another non-C-like language is recommended for CS1 and CS2 courses. Python has consistently ranked as the top programming language in various indexes and rankings for the past few years, and several educators have recommended using it for CS1 (Gass, 2022; Srinath, 2017).

Another consideration for teaching programming languages is the level of emphasis on object orientation or the focus on a paradigm (Krishnamurthi & Fisler, 2019). The two common paradigms in programming are object-oriented programming and procedural programming. The language with the most robust emphasis on object orientation is Java. Hence, this may be the language of choice for CS2, given that object-oriented programming is a critical knowledge area for most computing programs.

Dealing with the challenges to learning programming

Educators in computing programs generally agree that learning to program remains a challenging task to many and is a source of attrition among most computing majors, especially CS majors. Some strategies may help to alleviate this problem. The "Learning by Doing" method is a better way to deliver lectures in the classroom (Denny et al., 2021). Guiding the students to do more rather than lecturing them on how to do it helps them learn more and may make it simpler to complete the tasks (Kumar & Dahiya, 2017).

Programming, an individualistic and non-social activity, is a challenge many computing programs struggle with, exacerbating student attrition. Creating an environment that encourages collaboration, teamwork, and social interaction among the students is best. Clubs, support groups, or similar activities may help. However, these activities are often conducted outside the classroom and need to be integrated into the pedagogy of teaching programming. Doing exercises that encourage communication regarding assignments could help. For example, the “classroom communities” platform that helps student engagement (such as Yellowdig) may help create this kind of communication (Martin et al., 2017).

Introducing real-time and automatic feedback while the programming is being done could help with the issue of syntax (Denny et al., 2021). This feedback can be helped through automatic grading with an unlimited or generous number of attempts that are available in different publisher textbook platforms like Zybooks. Auto grading, along with instantaneous real-time feedback and allowance for more attempts, can help students improve their code and troubleshoot their program in a specific way. Though, there needs to be some willingness on the part of the students to use this for learning and/or seeking help when running into issues rather than seeking solutions on platforms like Quizlet, Chegg, and others.

Addressing issues with future trends

There is a general agreement in the industry that AI-based tools for code generation and completion will progress further and become more sophisticated over the next few years. However, as discussed earlier, a conceptual understanding of programming is still needed to validate and verify the generated code. In many cases, understanding the code generated and fixing and modifying it will become a valuable skill.

Testing of generative AI to produce code had mixed results. While simple code was easily generated, the attempt to create a crawler to glean financial data from websites and compile it was a complete failure. An attempt to fix the code was an arduous task since the generated code combines various existing codes that the platform thinks will solve the problem. The time to not teach programming from a conceptual viewpoint and maybe from a philosophical view is not yet here, and the market will need graduates with both conceptual and practical understanding and associated programming skills.

Recommendations and Conclusions

Based on the review of the literature and discussions forwarded in this paper following recommendations may be presented regarding the selection of programming language in a typical CS1 and CS2 course:

- Python as the programming language is recommended as the language to teach for CS1. The name of the course may be changed from “Problem-Solving and Structured Programming” to something more reflective of the use of Python in the course.
- For CS2, Java is recommended as the programming language of choice. The reason for selection is the strong object orientation of Java and the ability to map the concepts of abstraction, encapsulation, polymorphism, and inheritance directly to the language constructs. These concepts give students a grounded orientation to “Object Oriented Programming”. The course title may reflect the focus on object-oriented programming. It may be simply named as “Object Oriented Programming”.
- Introducing communication and collaboration platforms like YellowDig in programming courses can help address the isolation issue and potentially help with attrition. However, introducing something like this may require structural changes to the course and to the pedagogy.
- Something that has proven to overcome the challenges of learning to program is the use of “Learning by Doing”. Giving students more assignments to work on, some with hand holding to start with and then continue as they progress through learning.

- The use of AI tools may be incorporated in higher-level courses (like juniors and seniors). These are beneficial tools, and students should learn and are already employed in the industry. They will provide an advantage to students when they graduate and are in the job market. However, learning to program traditionally first will be more helpful to the students directly into learning these tools.

References

Ali, A., & Smith, D. (2014). A debate over the teaching of a legacy programming language in an information technology (IT) program. *Journal of Information Technology Education: Innovations in Practice*, *13*,111-127. Retrieved from http://www.jite.org/documents/Vol13/JITEv13IIPp111-127Ali0773.pdf

Baltes, S., & Diehl, S. (2018, October). Towards a theory of software development expertise. In *Proceedings of the 2018 26th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering* (pp. 187-200).

Becker, B. A., & Quille, K. (2019, February). 50 years of cs1 at sigcse: A review of the evolution of introductory programming education research. In *Proceedings of the 50th acm technical symposium on computer science education* (pp. 338-344).

Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2022). Programming Is Hard--Or at Least It Used to Be: Educational Opportunities And Challenges of AI Code Generation. *arXiv preprint arXiv:2212.01020*.

Cao, P. (2023). 8 Most In-Demand Programming Languages for 2023: Results from Last 14-Month Job Listings Analysis. https://www.scribd.com/document/628436768/8-Most-In-Demand-Programming-Languages-for-2023-Results-from-Last-14-Month-Job-Listings-Analysis-by-Peng-Cao-Dec-2022-Bootcamp

Cheah, C. S. (2020). Factors contributing to the difficulties in teaching and learning of computer programming: A literature review. *Contemporary Educational Technology*, *12*(2), ep272.

Coursera (2023). 5 Types of Programming Languages. Retrieved May 1st, 2023 from https://www.coursera.org/articles/types-programming-language

Denny, P., Kumar, V., & Giacaman, N. (2023, March). Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (pp. 1136-1142).

Denny, P., Whalley, J., & Leinonen, J. (2021, February). Promoting early engagement with programming assignments using scheduled automated feedback. In *Proceedings of the 23rd Australasian Computing Education Conference* (pp. 88-95).

Gass, S. (2022). Top Programming Languages 2022 > Python's still No. 1, but employers love to see SQL skills. Retrieved April 30th, 2023 from https://spectrum.ieee.org/top-programming-languages-2022

Krishnamurthi, S., & Fisler, K. (2019). 13 Programming Paradigms and Beyond. *The Cambridge handbook of computing education research*, 377.

Kumar, K., & Dahiya, S. (2017). Programming languages: A survey. *International Journal on Recent and Innovation Trends in Computing and Communication*, *5*(5), 307-313.

Larson, Q. (2022). What Programming Language Should I learn First in 2022? [Solved]. Retrieved April 30th, 2023 from https://www.freecodecamp.org/news/what-programming-language-should-i-learn-first-19a33b0a467d/

Liu, X., & Wu, D. (2018). From natural language to programming language. In *Innovative methods, user-friendly tools, coding, and design approaches in people-oriented programming* (pp. 110-130). IGI Global.

Martin, M. C., Martin, M. J., & Feldstein, A. P. (2017). Using Yellowdig in Marketing Courses: An Analysis of Individual Contributions and Social Interactions in Online Classroom Communities and Their Impact on Student Learning and Engagement. *Global Journal of Business Pedagogy, 1(1)*, 55–74.

Sarsa, S., Denny, P., Hellas, A., & Leinonen, J. (2022, August). Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1* (pp. 27-43).

Srinath, K. R. (2017). Python–the fastest growing programming language. *International Research Journal of Engineering and Technology*, *4*(12), 354-357.

Teague, D., & Roe, P. (2008). Collaborative learning-towards a solution for novice programmers. In *Proceedings of the Tenth Australasian Computing Education Conference in Conferences in Research and Practice in Information Technology-CRPIT Volume 78* (pp. 147-153). Australian Computer Society.

Tozzi, C. (2023). Top Programming Language Trends in 2023. *ITPro Today*. Retrieved April 30th, from https://www.itprotoday.com/programming-languages/top-programming-language-trends-2023

Upson, M. (2023). How Many Programming Languages Are There? Retrieved May 24, 2023 from https://www.bestcolleges.com/bootcamps/guides/how-many-coding-languages-are-there/